Package 'admiral'

June 12, 2026

Type Package

Title ADaM in R Asset Library

Version 1.5.0

Description A toolbox for programming Clinical Data Interchange Standards Consortium (CDISC) compliant Analysis Data Model (ADaM) datasets in R. ADaM datasets are a mandatory part of any New Drug or Biologics License Application submitted to the United States Food and Drug Administration (FDA). Analysis derivations are implemented in accordance with the ``Analysis Data Model Implementation Guide'' (CDISC Analysis Data Model Team, 2021, <https://www.cdisc.org/standards/foundational/adam>).

License Apache License (>= 2)

URL https://pharmaverse.github.io/admiral/, https://github.com/pharmaverse/admiral

BugReports https://github.com/pharmaverse/admiral/issues

Depends R (>= 4.1)

Imports admiraldev (>= 1.5.0), cli (>= 3.6.2), dplyr (>= 1.1.1), hms (>= 0.5.3), lifecycle (>= 0.1.0), lubridate (>= 1.7.4), magrittr (>= 1.5), purrr (>= 0.3.3), rlang (>= 0.4.4), stringr (>= 1.4.0), tidyr (>= 1.0.2), tidyselect (>= 1.1.0)

Suggests diffdf, DT, ggnewscale, ggplot2, here, htmltools, knitr, methods, pharmaversesdtm (>= 1.0.0), reactable, readxl, rmarkdown, testthat (>= 3.0.0), tibble, withr

VignetteBuilder knitr

Config/Needs/website gert, rmarkdown

Config/roxygen2/markdown TRUE

Config/roxygen2/roclets collate, namespace, admiraldev::rdx_roclet

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

NeedsCompilation no

Author Edoardo Mancini [aut, cre] (ORCID:
<https://orcid.org/0009-0006-4899-8641>),
Stefan Bundfuss [aut] (ORCID: <https://orcid.org/0009-0005-0027-1198>),
Arianna Cascone [aut] (ORCID: <https://orcid.org/0000-0001-5948-2831>),
Kristin Dahnert [aut],
Jeffrey Dickinson [aut],
Ross Farrugia [aut],
Fanny Gautier [aut] (ORCID: <https://orcid.org/0009-0004-3581-0131>),
Liam Hobby [aut],
Gordon Miller [aut],
Lina Patil [aut],
Ben Straub [aut],
F. Hoffmann-La Roche AG [cph, fnd],
GlaxoSmithKline LLC [cph, fnd]

Maintainer Edoardo Mancini <edoardo.mancini@roche.com>

Repository CRAN

Date/Publication 2026-06-12 13:20:02 UTC

Contents

admiral_adlb 5
admiral_adsl 6
atoxgr_criteria_ctcv4 6
atoxgr_criteria_ctcv4_uscv 8
atoxgr_criteria_ctcv5 9
atoxgr_criteria_ctcv5_uscv 10
atoxgr_criteria_ctcv6 11
atoxgr_criteria_ctcv6_uscv 12
atoxgr_criteria_daids 13
atoxgr_criteria_daids_uscv 15
basket_select 16
call_derivation 17
call_user_fun 19
censor_source 20
chr2vars 22
compute_age_years 22
compute_bmi 24
compute_bsa 25
compute_dtf 26
compute_duration 27
compute_egfr 30
compute_framingham 33
compute_map 36

compute_qtc . . . 37
compute_qual_imputation . . . 38
compute_qual_imputation_dec . . . 39
compute_rr . . . 40
compute_scale . . . 41
compute_tmf . . . 42
consolidate_metadata . . . 44
convert_blanks_to_na . . . 45
convert_date_to_dtm . . . 47
convert_dtc_to_dt . . . 50
convert_dtc_to_dtm . . . 52
convert_na_to_blanks . . . 55
convert_xxtpt_to_hours . . . 56
country_code_lookup . . . 61
count_vals . . . 63
create_period_dataset . . . 64
create_query_data . . . 66
create_single_dose_dataset . . . 70
date_source . . . 75
death_event . . . 77
default_qtc_paramcd . . . 78
derivation_slice . . . 79
derive_basetype_records . . . 79
derive_expected_records . . . 82
derive_extreme_event . . . 84
derive_extreme_records . . . 99
derive_locf_records . . . 113
derive_param_bmi . . . 118
derive_param_bsa . . . 122
derive_param_computed . . . 126
derive_param_doseint . . . 135
derive_param_exist_flag . . . 138
derive_param_exposure . . . 141
derive_param_extreme_record . . . 144
derive_param_framingham . . . 147
derive_param_map . . . 152
derive_param_qtc . . . 155
derive_param_rr . . . 158
derive_param_tte . . . 160
derive_param_wbc_abs . . . 182
derive_summary_records . . . 184
derive_vars_aage . . . 193
derive_vars_atc . . . 195
derive_vars_cat . . . 197
derive_vars_computed . . . 203
derive_vars_crit_flag . . . 206
derive_vars_dt . . . 210
derive_vars_dtm . . . 219

derive_vars_dtm_to_dt . . . 227
derive_vars_dtm_to_tm . . . 229
derive_vars_duration . . . 230
derive_vars_dy . . . 234
derive_vars_extreme_event . . . 236
derive_vars_joined . . . 240
derive_vars_joined_summary . . . 257
derive_vars_merged . . . 266
derive_vars_merged_lookup . . . 275
derive_vars_merged_summary . . . 278
derive_vars_period . . . 284
derive_vars_query . . . 287
derive_vars_transposed . . . 289
derive_var_age_years . . . 292
derive_var_analysis_ratio . . . 293
derive_var_anrind . . . 295
derive_var_atoxgr . . . 297
derive_var_atoxgr_dir . . . 298
derive_var_base . . . 302
derive_var_chg . . . 304
derive_var_dthcaus . . . 305
derive_var_extreme_dt . . . 308
derive_var_extreme_dtm . . . 312
derive_var_extreme_flag . . . 317
derive_var_joined_exist_flag . . . 326
derive_var_merged_ef_msrc . . . 341
derive_var_merged_exist_flag . . . 348
derive_var_merged_summary . . . 351
derive_var_nfrlt . . . 353
derive_var_obs_number . . . 369
derive_var_ontrtfl . . . 371
derive_var_pchg . . . 374
derive_var_relative_flag . . . 376
derive_var_shift . . . 379
derive_var_trtdurd . . . 381
derive_var_trtemfl . . . 382
desc . . . 391
dose_freq_lookup . . . 391
dthcaus_source . . . 392
event . . . 394
event_joined . . . 396
event_source . . . 401
example_qs . . . 403
exprs . . . 403
extract_unit . . . 404
filter_exist . . . 404
filter_extreme . . . 406
filter_joined . . . 408

filter_not_exist . . . 418
filter_relative . . . 420
flag_event . . . 423
get_admiral_option . . . 424
get_duplicates_dataset . . . 425
get_flagged_records . . . 426
get_many_to_one_dataset . . . 428
get_not_mapped . . . 429
get_one_to_many_dataset . . . 429
get_summary_records . . . 430
get_vars_query . . . 433
impute_dtc_dt . . . 435
impute_dtc_dtm . . . 439
list_all_templates . . . 443
list_tte_source_objects . . . 444
max_cond . . . 445
min_cond . . . 446
negate_vars . . . 447
params . . . 447
queries . . . 449
queries_mh . . . 450
query . . . 451
records_source . . . 453
restrict_derivation . . . 454
set_admiral_options . . . 456
slice_derivation . . . 458
transform_range . . . 460
tte_source . . . 462
use_ad_template . . . 463
yn_to_numeric . . . 464
%>% . . . 465

Index **466**

`admiral_adlb` *Lab Analysis Dataset*

Description

An example of lab analysis dataset

Usage

```
admiral_adlb
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 330 rows and 115 columns.

Source

Derived from the adlb template, then further filtered due to dataset size by the following USUBJIDs: 01-701-1015, 01-701-1023, 01-701-1028, 01-701-1033, 01-701-1034, 01-701-1047, 01-701-1097, 01-705-1186, 01-705-1292, 01-705-1310, 01-708-1286

See Also

Other datasets: admiral_adsl, example_qs, queries, queries_mh

---

admiral_adsl *Subject Level Analysis Dataset*

Description

An example subject level analysis dataset

Usage

```
admiral_adsl
```

Format

An object of class tbl_df (inherits from tbl, data.frame) with 306 rows and 54 columns.

Source

Derived from the dm and ds datasets using {admiral} (https://github.com/pharmaverse/admiral/blob/main/inst/templates/ad_adsl.R)

See Also

Other datasets: admiral_adlb, example_qs, queries, queries_mh

---

atoxgr_criteria_ctcv4 *Metadata Holding Grading Criteria for NCI-CTCAEv4 using SI unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv4 using SI unit where applicable

Usage

```
atoxgr_criteria_ctcv4
```

Format

An object of class `data.frame` with 42 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev4.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold SI unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v4.0** From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6`, `atoxgr_criteria_ctcv6_uscv`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_uscv`, `country_code_lookup`, `dose_freq_lookup`

`atoxgr_criteria_ctcv4_uscv` *Metadata Holding Grading Criteria for NCI-CTCAEv4 using USCV unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv4 using USCV unit where applicable

Usage

```
atoxgr_criteria_ctcv4_uscv
```

Format

An object of class `data.frame` with 48 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev4_uscv.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold USCV unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v4.0** From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, country_code_lookup, dose_freq_lookup

---

atoxgr_criteria_ctcv5 *Metadata Holding Grading Criteria for NCI-CTCAEv5 using SI unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv5 using SI unit where applicable

Usage

```
atoxgr_criteria_ctcv5
```

Format

An object of class `data.frame` with 40 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev5.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold SI unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.

- COMMENT: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables SOC, TERM, Grade 1, Grade 2,Grade 3,Grade 4,Grade 5, Definition are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v5.0** From these variables only 'TERM' is used in the {admiral} code, the rest are for information and traceability only.

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, country_code_lookup, dose_freq_lookup

---

atoxgr_criteria_ctcv5_uscv

Metadata Holding Grading Criteria for NCI-CTCAEv5 using USCV unit where applicable

---

Description

Metadata Holding Grading Criteria for NCI-CTCAEv5 using USCV unit where applicable

Usage

```
atoxgr_criteria_ctcv5_uscv
```

Format

An object of class data.frame with 46 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file data-raw/adlb_grading/ncictcaev5_uscv.json. The variables GRADE_NA_CODE, GRADE_4_CODE, GRADE_3_CODE, GRADE_2_CODE and GRADE_1_CODE in the json file are combined to create GRADE_CRITERIA_CODE, and then dropped from metadata. The dataset contains the following columns:

- SOC: variable to hold the SOC of the lab test criteria.
- TERM: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- Grade 1: Criteria defining lab value as Grade 1.
- Grade 2: Criteria defining lab value as Grade 2.
- Grade 3: Criteria defining lab value as Grade 3.
- Grade 4: Criteria defining lab value as Grade 4.
- Grade 5: Criteria defining lab value as Grade 5.
- Definition: Holds the definition of the lab test abnormality.

- GRADE_CRITERIA_CODE: variable to hold code that creates grade based on defined criteria.
- UNIT_CHECK: variable to hold USCV unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- VAR_CHECK: List of variables required to implement lab grade criteria. Use to check against input data.
- DIRECTION: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- COMMENT: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables SOC, TERM, Grade 1, Grade 2,Grade 3,Grade 4,Grade 5, Definition are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v5.0** From these variables only 'TERM' is used in the {admiral} code, the rest are for information and traceability only.

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, country_code_lookup, dose_freq_lookup

---

atoxgr_criteria_ctcv6 *Metadata Holding Grading Criteria for NCI-CTCAEv6 using SI unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv6 using SI unit where applicable

Usage

```
atoxgr_criteria_ctcv6
```

Format

An object of class data.frame with 43 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file data-raw/adlb_grading/ncictcaev6.json. The variables GRADE_NA_CODE, GRADE_4_CODE, GRADE_3_CODE, GRADE_2_CODE and GRADE_1_CODE in the json file are combined to create GRADE_CRITERIA_CODE, and then dropped from metadata. The dataset contains the following columns:

- SOC: variable to hold the SOC of the lab test criteria.
- TERM: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.

- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold SI unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v6.0** From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6_uscv`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_us` `country_code_lookup`, `dose_freq_lookup`

`atoxgr_criteria_ctcv6_uscv` *Metadata Holding Grading Criteria for NCI-CTCAEv6 using USCV unit where applicable*

Description

Metadata Holding Grading Criteria for NCI-CTCAEv6 using USCV unit where applicable

Usage

```
atoxgr_criteria_ctcv6_uscv
```

Format

An object of class `data.frame` with 48 rows and 13 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/ncictcaev6_uscv.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `GRADE_1`: Criteria defining lab value as Grade 1.
- `GRADE_2`: Criteria defining lab value as Grade 2.
- `GRADE_3`: Criteria defining lab value as Grade 3.
- `GRADE_4`: Criteria defining lab value as Grade 4.
- `GRADE_5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold USCV unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on NCI-CTC website defining the grading criteria. **Common Terminology Criteria for Adverse Events (CTCAE)v6.0** From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv4_uscv`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv5_uscv`, `atoxgr_criteria_ctcv6`, `atoxgr_criteria_daids`, `atoxgr_criteria_daids_uscv`, `country_code_lookup`, `dose_freq_lookup`

---

atoxgr_criteria_daids *Metadata Holding Grading Criteria for DAIDs using SI unit where applicable*

Description

Metadata Holding Grading Criteria for DAIDs using SI unit where applicable

Usage

```
atoxgr_criteria_daids
```

Format

An object of class `data.frame` with 63 rows and 14 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/DAIDS.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `SUBGROUP` : Description of sub-group of subjects were grading will be applied (i.e. >= 18 years)
- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `FILTER` : `admiral` code to apply the filter based on SUBGROUP column.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold SI unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.
- `COMMENT`: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables `SOC`, `TERM`, `SUBGROUP`, `Grade 1`, `Grade 2`,`Grade 3`,`Grade 4`,`Grade 5`, `Definition` are from the source document on DAIDS website defining the grading criteria, (Division of AIDS (DAIDS) Table for Grading the Severity of Adult and Pediatric Adverse Events). From these variables only 'TERM' is used in the `{admiral}` code, the rest are for information and traceability only.

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids_us country_code_lookup, dose_freq_lookup

atoxgr_criteria_daids_uscv *Metadata Holding Grading Criteria for DAIDs using USCV unit where applicable*

Description

Metadata Holding Grading Criteria for DAIDs using USCV unit where applicable

Usage

```
atoxgr_criteria_daids_uscv
```

Format

An object of class `data.frame` with 71 rows and 14 columns.

Details

This metadata has its origin in the ADLB Grading Spec json file `data-raw/adlb_grading/DAIDS_uscv.json`. The variables `GRADE_NA_CODE`, `GRADE_4_CODE`, `GRADE_3_CODE`, `GRADE_2_CODE` and `GRADE_1_CODE` in the json file are combined to create `GRADE_CRITERIA_CODE`, and then dropped from metadata. The dataset contains the following columns:

- `SOC`: variable to hold the SOC of the lab test criteria.
- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. 'Anemia' or 'INR Increased'. Note: the variable is case insensitive.
- `SUBGROUP` : Description of sub-group of subjects were grading will be applied (i.e. >= 18 years)
- `Grade 1`: Criteria defining lab value as Grade 1.
- `Grade 2`: Criteria defining lab value as Grade 2.
- `Grade 3`: Criteria defining lab value as Grade 3.
- `Grade 4`: Criteria defining lab value as Grade 4.
- `Grade 5`: Criteria defining lab value as Grade 5.
- `Definition`: Holds the definition of the lab test abnormality.
- `FILTER` : `admiral` code to apply the filter based on SUBGROUP column.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `UNIT_CHECK`: variable to hold USCV unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: List of variables required to implement lab grade criteria. Use to check against input data.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. 'L' is for LOW values, 'H' is for HIGH values. Note: the variable is case insensitive.

- COMMENT: Holds any information regarding rationale behind implementation of grading criteria.

Note: Variables SOC, TERM, SUBGROUP, Grade 1, Grade 2,Grade 3,Grade 4,Grade 5, Definition are from the source document on DAIDS website defining the grading criteria. [Division of AIDS (DAIDS) Table for Grading the Severity of Adult and Pediatric Adverse Events From these variables only 'TERM' is used in the {admiral} code, the rest are for information and traceability only.

See Also

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, country_code_lookup, dose_freq_lookup

---

basket_select *Create a* basket_select *object*

Description

Create a basket_select object

Usage

```
basket_select(name = NULL, id = NULL, scope = NULL, type, ...)
```

Arguments

name	Name of the query used to select the definition of the query from the company database. **Default value** NULL
id	Identifier of the query used to select the definition of the query from the company database. **Default value** NULL
scope	Scope of the query used to select the definition of the query from the company database. **Permitted values** "BROAD", "NARROW", NA_character_ **Default value** NULL
type	The type argument expects a character scalar. It is passed to the company specific get_terms() function such that the function can determine which sort of basket is requested **Default value** none
...	Any number of *named* function arguments. Can be used to pass in company specific conditions or flags that will then be used in user-defined function that is passed into argument get_terms_fun for function create_query_data(). **Default value** none

Details

Exactly one of `name` or `id` must be specified.

Value

An object of class `basket_select`.

See Also

`create_query_data()`, `query()`

Source Objects: `censor_source()`, `death_event`, `event()`, `event_joined()`, `event_source()`, `flag_event()`, `query()`, `records_source()`, `tte_source()`

`call_derivation` *Call a Single Derivation Multiple Times*

Description

Call a single derivation multiple times with some parameters/arguments being fixed across iterations and others varying.

Usage

```
call_derivation(dataset = NULL, derivation, variable_params, ...)
```

Arguments

`dataset`	Input dataset **Default value** `NULL`
`derivation`	The derivation function to call A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. All expected arguments for the derivation function must be provided through the `params()` objects passed to the `variable_params` and `...` arguments. **Default value** none
`variable_params`	A `list` of function arguments that are different across iterations. Each set of function arguments must be created using `params()`. **Default value** none
`...`	Any number of *named* function arguments that stay the same across iterations. If a function argument is specified both inside `variable_params` and `...` then the value in `variable_params` overwrites the one in `....` @details

It is also possible to pass functions from outside the {admiral} package to call_derivation(), e.g. an extension package function, or dplyr::mutate(). The only requirement for a function being passed to derivation is that it must take a dataset as its first argument and return a dataset.

Default value none

Value

The input dataset with additional records/variables added depending on which derivation has been used.

See Also

params() restrict_derivation() call_derivation()

Higher Order Functions: derivation_slice(), restrict_derivation(), slice_derivation()

Examples

```
library(dplyr, warn.conflicts = FALSE)
adsl <- tribble(
  ~STUDYID,   ~USUBJID,      ~TRTSDT,      ~TRTEDT,
  "PILOT01", "01-1307",           NA,           NA,
  "PILOT01", "05-1377", "2014-01-04", "2014-01-25",
  "PILOT01", "06-1384", "2012-09-15", "2012-09-24",
  "PILOT01", "15-1085", "2013-02-16", "2013-08-18",
  "PILOT01", "16-1298", "2013-04-08", "2013-06-28"
) %>%
  mutate(
    across(TRTSDT:TRTEDT, as.Date)
  )

ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,     ~AESTDTC,     ~AEENDTC,
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06"
)
```

```
adae <- ae %>%
  derive_vars_merged(
    dataset_add = adsl,
    new_vars = exprs(TRTSDT, TRTEDT),
    by_vars = exprs(USUBJID)
  )

## While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
## one can add multiple variables in one go
call_derivation(
  dataset = adae,
  derivation = derive_vars_dt,
  variable_params = list(
    params(dtc = AESTDTC, date_imputation = "first", new_vars_prefix = "AST"),
    params(dtc = AEENDTC, date_imputation = "last", new_vars_prefix = "AEN")
  ),
  min_dates = exprs(TRTSDT),
  max_dates = exprs(TRTEDT)
)

## The above call using `call_derivation()` is equivalent to the following
adae %>%
  derive_vars_dt(
    new_vars_prefix = "AST",
    dtc = AESTDTC,
    date_imputation = "first",
    min_dates = exprs(TRTSDT),
    max_dates = exprs(TRTEDT)
  ) %>%
  derive_vars_dt(
    new_vars_prefix = "AEN",
    dtc = AEENDTC,
    date_imputation = "last",
    min_dates = exprs(TRTSDT),
    max_dates = exprs(TRTEDT)
  )
```

`call_user_fun` *Calls a Function Provided by the User*

Description

[Deprecated]

Calls a function provided by the user and adds the function call to the error message if the call fails.

Usage

```
call_user_fun(call)
```

Arguments

call	Call to be executed **Default value** none

Value

The return value of the function call

See Also

Other deprecated: date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

---

censor_source *Create a* censor_source *Object*

Description

censor_source objects are used to define censorings as input for the derive_param_tte() function.

Note: This is a wrapper function for the more generic tte_source().

Usage

```
censor_source(
  dataset_name,
  filter = NULL,
  date,
  censor = 1,
  set_values_to = NULL,
  order = NULL,
  consider_end_dates = TRUE
)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets parameter of derive_param_tte(). **Default value** none
filter	An unquoted condition for selecting the observations from dataset which are events or possible censoring time points. **Default value** NULL

date
: A variable or expression providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol or expression is expected.

  Refer to derive_vars_dt() or convert_dtc_to_dt() to impute and derive a date from a date character vector to a date object.

  Default value none

censor
: Censoring value

  CDISC strongly recommends using 0 for events and positive integers for censoring.

  Default value 0

set_values_to
: A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA.

  Default value NULL

order
: Sort order

  An optional named list returned by exprs() defining additional variables that the source dataset is sorted on after date.

  Permitted values list of variables created by exprs() e.g. exprs(ASEQ).

  Default value order

consider_end_dates
: Should end dates be considered?

  If end dates are considered, the records which are after the end date are ignored and the censor value specified for the end date takes precedence.

  Permitted values TRUE, FALSE

  Default value TRUE

Value

An object of class censor_source, inheriting from class tte_source

See Also

derive_param_tte(), event_source()

Source Objects: basket_select(), death_event, event(), event_joined(), event_source(), flag_event(), query(), records_source(), tte_source()

Examples

```
# Last study date known alive censor

censor_source(
  dataset_name = "adsl",
  date = LSTALVDT,
  set_values_to = exprs(
    EVNTDESC = "ALIVE",
```

```
    SRCDOM = "ADSL",
    SRCVAR = "LSTALVDT"
  )
)
```

`chr2vars` *Turn a Character Vector into a List of Expressions*

Description

Turn a character vector into a list of expressions

Usage

```
chr2vars(chr)
```

Arguments

`chr`	A character vector
	Default value none

Value

A `list` of expressions as returned by `exprs()`

See Also

Utilities for working with quosures/list of expressions: `negate_vars()`

Examples

```
chr2vars(c("USUBJID", "AVAL"))
```

`compute_age_years` *Compute Age in Years*

Description

Converts a set of age values from the specified time unit to years.

Usage

```
compute_age_years(age, age_unit)
```

Arguments

age	The ages to convert. A numeric vector is expected. **Default value** none
age_unit	Age unit. Either a string containing the time unit of all ages in age or a character vector containing the time units of each age in age is expected. Note that permitted values are cases insensitive (e.g. "YEARS" is treated the same as "years" and "Years"). **Permitted values** "years", "months", "weeks", "days", "hours", "minutes", "seconds", NA_character_. **Default value** none

Details

Returns a numeric vector of ages in years as doubles. Note that passing NA_character_ as a unit will result in an NA value for the outputted age. Also note, underlying computations assume an equal number of days in each year (365.25).

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

The ages contained in age converted to years.

See Also

Date/Time Computation Functions that returns a vector: compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(impute_dtc_dt(), impute_dtc_dtm()

Examples

```
compute_age_years(
  age = c(240, 360, 480),
  age_unit = "MONTHS"
)

compute_age_years(
  age = c(10, 520, 3650, 1000),
  age_unit = c("YEARS", "WEEKS", "DAYS", NA_character_)
)
```

compute_bmi *Compute Body Mass Index (BMI)*

Description

Computes BMI from height and weight

Usage

```
compute_bmi(height, weight)
```

Arguments

height	HEIGHT value It is expected that HEIGHT is in cm. **Permitted values** numeric vector **Default value** none
weight	WEIGHT value It is expected that WEIGHT is in kg. **Permitted values** numeric vector **Default value** none

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

The BMI (Body Mass Index Area) in kg/m^2.

See Also

derive_param_bmi()

BDS-Findings Functions that returns a vector: compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation(), compute_qual_imputation_dec(), compute_rr(), compute_scale(), transform_range()

Examples

```
compute_bmi(height = 170, weight = 75)
```

compute_bsa *Compute Body Surface Area (BSA)*

Description

Computes BSA from height and weight making use of the specified derivation method

Usage

```
compute_bsa(height = height, weight = weight, method)
```

Arguments

height	HEIGHT value It is expected that HEIGHT is in cm. **Permitted values** numeric vector **Default value** `height`
weight	WEIGHT value It is expected that WEIGHT is in kg. **Permitted values** numeric vector **Default value** `weight`
method	Derivation method to use: Mosteller: sqrt(height * weight / 3600) DuBois-DuBois: 0.007184 * height ^ 0.725 * weight ^ 0.425 Haycock: 0.024265 * height ^ 0.3964 * weight ^ 0.5378 Gehan-George: 0.0235 * height ^ 0.42246 * weight ^ 0.51456 Boyd: 0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 - (0.0188 * log10(1000 * weight))) Fujimoto: 0.008883 * height ^ 0.663 * weight ^ 0.444 Takahira: 0.007241 * height ^ 0.725 * weight ^ 0.425 **Permitted values** character value **Default value** none

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

The BSA (Body Surface Area) in m^2.

See Also

derive_param_bsa()

BDS-Findings Functions that returns a vector: compute_bmi(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation(), compute_qual_imputation_dec(), compute_rr(), compute_scale(), transform_range()

Examples

```
# Derive BSA by the Mosteller method
compute_bsa(
  height = 170,
  weight = 75,
  method = "Mosteller"
)

# Derive BSA by the DuBois & DuBois method
compute_bsa(
  height = c(170, 185),
  weight = c(75, 90),
  method = "DuBois-DuBois"
)
```

---

compute_dtf *Derive the Date Imputation Flag*

Description

Derive the date imputation flag (*DTF) comparing a date character vector (--DTC) with a Date vector (*DT).

Usage

```
compute_dtf(dtc, dt)
```

Arguments

dtc	The date character vector (--DTC). A character date is expected in a format like yyyy-mm-ddThh:mm:ss (partial or complete). **Default value** none
dt	The Date vector to compare. A date object is expected. **Default value** none

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

The date imputation flag (*DTF) (character value of "D", "M" , "Y" or NA)

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(impute_dtc_dt(), impute_dtc_dtm()

Examples

```
compute_dtf(dtc = "2019-07", dt = as.Date("2019-07-18"))
compute_dtf(dtc = "2019", dt = as.Date("2019-07-18"))
compute_dtf(dtc = "--06-01T00:00", dt = as.Date("2022-06-01"))
compute_dtf(dtc = "2022-06--T00:00", dt = as.Date("2022-06-01"))
compute_dtf(dtc = "2022---01T00:00", dt = as.Date("2022-06-01"))
compute_dtf(dtc = "2022----T00:00", dt = as.Date("2022-06-01"))
```

---

compute_duration *Compute Duration*

Description

Compute duration between two dates, e.g., duration of an adverse event, relative day, age, ...

Usage

```
compute_duration(
  start_date,
  end_date,
  in_unit = "days",
  out_unit = "days",
  floor_in = TRUE,
  add_one = TRUE,
  trunc_out = FALSE,
  type = "duration"
)
```

Arguments

start_date	The start date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object. Refer to convert_dtc_to_dt() to obtain a vector of imputed dates. **Default value** none

`end_date` The end date

A date or date-time object is expected.

Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

Refer to `convert_dtc_to_dt()` to obtain a vector of imputed dates.

Default value none

`in_unit` Input unit

See floor_in and add_one parameter for details.

Permitted Values (case-insensitive):

For years: `"year"`, `"years"`, `"yr"`, `"yrs"`, `"y"`

For months: `"month"`, `"months"`, `"mo"`, `"mos"`

For days: `"day"`, `"days"`, `"d"`

For hours: `"hour"`, `"hours"`, `"hr"`, `"hrs"`, `"h"`

For minutes: `"minute"`, `"minutes"`, `"min"`, `"mins"`

For seconds: `"second"`, `"seconds"`, `"sec"`, `"secs"`, `"s"`

Default value `"days"`

`out_unit` Output unit

The duration is derived in the specified unit

Permitted Values (case-insensitive):

For years: `"year"`, `"years"`, `"yr"`, `"yrs"`, `"y"`

For months: `"month"`, `"months"`, `"mo"`, `"mos"`

For weeks: `"week"`, `"weeks"`, `"wk"`, `"wks"`, `"w"`

For days: `"day"`, `"days"`, `"d"`

For hours: `"hour"`, `"hours"`, `"hr"`, `"hrs"`, `"h"`

For minutes: `"minute"`, `"minutes"`, `"min"`, `"mins"`

For seconds: `"second"`, `"seconds"`, `"sec"`, `"secs"`, `"s"`

Default value `"days"`

`floor_in` Round down input dates?

The input dates are round down with respect to the input unit, e.g., if the input unit is 'days', the time of the input dates is ignored.

Permitted values `TRUE`, `FALSE`

Default value `TRUE`

`add_one` Add one input unit?

If the duration is non-negative, one input unit is added. i.e., the duration can not be zero.

Permitted values `TRUE`, `FALSE`

Default value `TRUE`

`trunc_out` Return integer part

The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.

Permitted values TRUE, FALSE
Default value FALSE

type — lubridate duration type.
See below for details.

Permitted values "duration", "interval"
Default value "duration"

Details

The output is a numeric vector providing the duration as time from start to end date in the specified unit. If the end date is before the start date, the duration is negative.

Value

The duration between the two date in the specified unit

Duration Type

The lubridate package calculates two types of spans between two dates: duration and interval. While these calculations are largely the same, when the unit of the time period is month or year the result can be slightly different.

The difference arises from the ambiguity in the length of "1 month" or "1 year". Months may have 31, 30, 28, or 29 days, and years are 365 days and 366 during leap years. Durations and intervals help solve the ambiguity in these measures.

The **interval** between 2000-02-01 and 2000-03-01 is 1 (i.e. one month). The **duration** between these two dates is 0.95, which accounts for the fact that the year 2000 is a leap year, February has 29 days, and the average month length is 30.4375, i.e. 29 / 30.4375 = 0.95.

For additional details, review the lubridate time span reference page.

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

See Also

derive_vars_duration()

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(impute_dtc_dt(), impute_dtc_dtm()

Examples

```
library(lubridate)

# Derive duration in days (integer), i.e., relative day
compute_duration(
  start_date = ymd_hms("2020-12-06T15:00:00"),
  end_date = ymd_hms("2020-12-24T08:15:00")
)
```

```
# Derive duration in days (float)
compute_duration(
  start_date = ymd_hms("2020-12-06T15:00:00"),
  end_date = ymd_hms("2020-12-24T08:15:00"),
  floor_in = FALSE,
  add_one = FALSE
)

# Derive age in years
compute_duration(
  start_date = ymd("1984-09-06"),
  end_date = ymd("2020-02-24"),
  trunc_out = TRUE,
  out_unit = "years",
  add_one = FALSE
)

# Derive duration in hours
compute_duration(
  start_date = ymd_hms("2020-12-06T9:00:00"),
  end_date = ymd_hms("2020-12-06T13:30:00"),
  out_unit = "hours",
  floor_in = FALSE,
  add_one = FALSE,
)
```

`compute_egfr` *Compute Estimated Glomerular Filtration Rate (eGFR) for Kidney Function*

Description

Compute Kidney Function Tests:

- Estimated Creatinine Clearance (CRCL) by Cockcroft-Gault equation
- Estimated Glomerular Filtration Rate (eGFR) by CKD-EPI or MDRD equations

Usage

```
compute_egfr(creat, creatu = "SI", age, weight, sex, race = NULL, method)
```

Arguments

`creat`	Creatinine A numeric vector is expected. **Default value** none
`creatu`	Creatinine Units A character vector is expected. Expected Values: `"SI"`, `"CV"`, `"umol/L"`, `"mg/dL"`

Default value `"SI"`

`age` Age (years)
A numeric vector is expected.

Default value none

`weight` Weight (kg)
A numeric vector is expected if `method = "CRCL"`

Default value none

`sex` Gender
A character vector is expected.
Expected Values: `"M"`, `"F"`

Default value none

`race` Race
A character vector is expected if `method = "MDRD"`
Expected Values: `"BLACK OR AFRICAN AMERICAN"` and others

Default value `NULL`

`method` Method
A character vector is expected.
Expected Values: `"CRCL"`, `"CKD-EPI"`, `"MDRD"`

Default value none

Details

Calculates an estimate of Glomerular Filtration Rate (eGFR)

CRCL Creatinine Clearance (Cockcroft-Gault)

For Creatinine in umol/L:

$$\frac{(140 - age) \times weight(kg) \times constant}{Serum\ Creatinine(\mu mol/L)}$$

$$Constant = 1.04\ for\ females, 1.23\ for\ males$$

For Creatinine in mg/dL:

$$\frac{(140 - age) \times weight(kg) \times (0.85\ if\ female)}{72 \times Serum\ Creatinine(mg/dL)}$$

units = mL/min

CKD-EPI Chronic Kidney Disease Epidemiology Collaboration formula

$$eGFR = 142 \times min(SCr/\kappa, 1)^{\alpha} \times max(SCr/\kappa, 1)^{-1.200} \times 0.9938^{Age} \times 1.012[if\ female]$$

SCr = standardized serum creatinine in mg/dL (Note SCr(mg/dL) = Creat(umol/L) / 88.42)

$$\kappa$$

= 0.7 (females) or 0.9 (males)

$$\alpha$$

= -0.241 (female) or -0.302 (male) units = mL/min/1.73 m2

MDRD Modification of Diet in Renal Disease formula

$$eGFR = 175 \times (SCr)^{-1.154} \times (age)^{-0.203} \times 0.742[if\ female] \times 1.212[if\ Black]$$

SCr = standardized serum creatinine in mg/dL (Note SCr(mg/dL) = Creat(umol/L) / 88.42)

units = mL/min/1.73 m2

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

A numeric vector of egfr values

See Also

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation(), compute_qual_imputation_dec(), compute_rr(), compute_scale(), transform_range()

Examples

```
compute_egfr(
  creat = 90, creatu = "umol/L", age = 53, weight = 85, sex = "M", method = "CRCL"
)

compute_egfr(
  creat = 90, creatu = "umol/L", age = 53, sex = "M", race = "ASIAN", method = "MDRD"
)

compute_egfr(
  creat = 70, creatu = "umol/L", age = 52, sex = "F", race = "BLACK OR AFRICAN AMERICAN",
  method = "MDRD"
)

compute_egfr(
  creat = 90, creatu = "umol/L", age = 53, sex = "M", method = "CKD-EPI"
)


base <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~AGE, ~SEX, ~RACE, ~WTBL, ~CREATBL, ~CREATBLU,
  "P01", "P01-1001", 55, "M", "WHITE", 90.7, 96.3, "umol/L",
  "P01", "P01-1002", 52, "F", "BLACK OR AFRICAN AMERICAN", 68.5, 70, "umol/L",
  "P01", "P01-1003", 67, "M", "BLACK OR AFRICAN AMERICAN", 85.0, 77, "umol/L",
  "P01", "P01-1004", 76, "F", "ASIAN", 60.7, 65, "umol/L",
```

```
)

base %>%
  dplyr::mutate(
    CRCL_CG = compute_egfr(
      creat = CREATBL, creatu = CREATBLU, age = AGE, weight = WTBL, sex = SEX,
      method = "CRCL"
    ),
    EGFR_EPI = compute_egfr(
      creat = CREATBL, creatu = CREATBLU, age = AGE, weight = WTBL, sex = SEX,
      method = "CKD-EPI"
    ),
    EGFR_MDRD = compute_egfr(
      creat = CREATBL, creatu = CREATBLU, age = AGE, weight = WTBL, sex = SEX,
      race = RACE, method = "MDRD"
    ),
  )
```

`compute_framingham` *Compute Framingham Heart Study Cardiovascular Disease 10-Year Risk Score*

Description

Computes Framingham Heart Study Cardiovascular Disease 10-Year Risk Score (FCVD101) based on systolic blood pressure, total serum cholesterol (mg/dL), HDL serum cholesterol (mg/dL), sex, smoking status, diabetic status, and treated for hypertension flag.

Usage

```
compute_framingham(sysbp, chol, cholhdl, age, sex, smokefl, diabetfl, trthypfl)
```

Arguments

`sysbp`	Systolic blood pressure A numeric vector is expected. **Default value** none
`chol`	Total serum cholesterol (mg/dL) A numeric vector is expected. **Default value** none
`cholhdl`	HDL serum cholesterol (mg/dL) A numeric vector is expected. **Default value** none
`age`	Age (years) A numeric vector is expected.

Default value none

`sex` Gender
A character vector is expected. Expected Values: 'M' 'F'

Default value none

`smokefl` Smoking Status
A character vector is expected. Expected Values: 'Y' 'N'

Default value none

`diabetfl` Diabetic Status
A character vector is expected. Expected Values: 'Y' 'N'

Default value none

`trthypfl` Treated for hypertension status
A character vector is expected. Expected Values: 'Y' 'N'

Default value none

Details

The predicted probability of having cardiovascular disease (CVD) within 10-years according to Framingham formula. See AHA Journal article General Cardiovascular Risk Profile for Use in Primary Care for reference.

For Women:

Factor	Amount
Age	2.32888
Total Chol	1.20904
HDL Chol	-0.70833
Sys BP	2.76157
Sys BP + Hypertension Meds	2.82263
Smoker	0.52873
Non-Smoker	0
Diabetic	0.69154
Not Diabetic	0
Average Risk	26.1931
Risk Period	0.95012

For Men:

Factor	Amount
Age	3.06117
Total Chol	1.12370
HDL Chol	-0.93263
Sys BP	1.93303
Sys BP + Hypertension Meds	2.99881
Smoker	.65451

Non-Smoker	0
Diabetic	0.57367
Not Diabetic	0
Average Risk	23.9802
Risk Period	0.88936

The equation for calculating risk:

$$RiskFactors = (log(Age) * AgeFactor) + (log(TotalChol) * TotalCholFactor) + (log(CholHDL) * CholHDLFactor)$$

$$Risk = 100 * (1 - RiskPeriodFactor^{e}xp(RiskFactors))$$

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

A numeric vector of Framingham values

See Also

derive_param_framingham()

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_map(), compute_qtc(), compute_qual_imputation(), compute_qual_imputation_dec(), compute_rr(), compute_scale(), transform_range()

Examples

```
compute_framingham(
  sysbp = 133, chol = 216.16, cholhdl = 54.91, age = 53,
  sex = "M", smokefl = "N", diabetfl = "N", trthypfl = "N"
)

compute_framingham(
  sysbp = 161, chol = 186.39, cholhdl = 64.19, age = 52,
  sex = "F", smokefl = "Y", diabetfl = "N", trthypfl = "Y"
)
```

`compute_map` *Compute Mean Arterial Pressure (MAP)*

Description

Computes mean arterial pressure (MAP) based on diastolic and systolic blood pressure. Optionally heart rate can be used as well.

Usage

```
compute_map(diabp, sysbp, hr = NULL)
```

Arguments

`diabp`	Diastolic blood pressure A numeric vector is expected. **Default value** none
`sysbp`	Systolic blood pressure A numeric vector is expected. **Default value** none
`hr`	Heart rate A numeric vector or `NULL` is expected. **Default value** `NULL`

Details

$$\frac{2DIABP + SYSBP}{3}$$

if it is based on diastolic and systolic blood pressure and

$$DIABP + 0.01e^{4.14 - \frac{40.74}{HR}}(SYSBP - DIABP)$$

if it is based on diastolic, systolic blood pressure, and heart rate.

This is a vector-oriented helper and is not usually called directly on a data frame with `%>%`.

Value

A numeric vector of MAP values

See Also

`derive_param_map()`

BDS-Findings Functions that returns a vector: `compute_bmi()`, `compute_bsa()`, `compute_egfr()`, `compute_framingham()`, `compute_qtc()`, `compute_qual_imputation()`, `compute_qual_imputation_dec()`, `compute_rr()`, `compute_scale()`, `transform_range()`

Examples

```
# Compute MAP based on diastolic and systolic blood pressure
compute_map(diabp = 51, sysbp = 121)

# Compute MAP based on diastolic and systolic blood pressure and heart rate
compute_map(diabp = 51, sysbp = 121, hr = 59)
```

---

compute_qtc *Compute Corrected QT*

Description

Computes corrected QT using Bazett's, Fridericia's or Sagie's formula.

Usage

```
compute_qtc(qt, rr, method)
```

Arguments

qt	QT interval A numeric vector is expected. It is expected that QT is measured in ms or msec. **Default value** none
rr	RR interval A numeric vector is expected. It is expected that RR is measured in ms or msec. **Default value** none
method	Method used to QT correction **Permitted values** "Bazett", "Fridericia", "Sagie" **Default value** none

Details

Depending on the chosen method one of the following formulae is used.

Bazett:

$$\frac{QT}{\sqrt{\frac{RR}{1000}}}$$

Fridericia:

$$\frac{QT}{\sqrt[3]{\frac{RR}{1000}}}$$

Sagie:

$$1000\left(\frac{QT}{1000}+0.154\left(1-\frac{RR}{1000}\right)\right)$$

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

QT interval in ms

See Also

derive_param_qtc()

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qual_imputation(), compute_qual_imputation_dec(), compute_rr(), compute_scale(), transform_range()

Examples

```
compute_qtc(qt = 350, rr = 857, method = "Bazett")

compute_qtc(qt = 350, rr = 857, method = "Fridericia")

compute_qtc(qt = 350, rr = 857, method = "Sagie")
```

---

compute_qual_imputation

Function to Impute Values When Qualifier Exists in Character Result

---

Description

Derive an imputed value

Usage

```
compute_qual_imputation(character_value, imputation_type = 1, factor = 0)
```

Arguments

character_value	Character version of value to be imputed **Default value** none
imputation_type	(default value=1) Valid Values: 1: Strip <, >, = and convert to numeric. 2: imputation_type=1 and if the character value contains a < or >, the number of of decimals associated with the character value is found and then a factor of 1/10^(number of decimals + 1) will be added/subtracted from the numeric value. If no decimals exists, a factor of 1/10 will be added/subtracted from the value. **Default value** 1
factor	Numeric value (default=0), when using imputation_type = 1, this value can be added or subtracted when the qualifier is removed. **Default value** 0

Value

The imputed value

See Also

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation_dec(), compute_rr(), compute_scale(), transform_range()

Examples

```
compute_qual_imputation("<40")
compute_qual_imputation(c("3", ">30.2"))
```

---

compute_qual_imputation_dec

Compute Factor for Value Imputations When Character Value Contains < or >

Description

Function to compute factor for value imputation when character value contains < or >. The factor is calculated using the number of decimals. If there are no decimals, the factor is 1, otherwise the factor = 1/10^decimal place. For example, the factor for 100 = 1, the factor for 5.4 = 1/10^1, the factor for 5.44 = 1/10^2. This results in no additional false precision added to the value. This is an intermediate function.

Usage

```
compute_qual_imputation_dec(character_value_decimal)
```

Arguments

`character_value_decimal`
: Character value to determine decimal precision

Default value none

Details

Derive an imputed value

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

Decimal precision value to add or subtract

See Also

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation(), compute_rr(), compute_scale(), transform_range()

Examples

```
compute_qual_imputation_dec("<40.1")
compute_qual_imputation_dec(c("0.35", "1"))
```

---

compute_rr *Compute RR Interval From Heart Rate*

Description

Computes RR interval from heart rate.

Usage

```
compute_rr(hr)
```

Arguments

hr	Heart rate A numeric vector is expected. It is expected that heart rate is measured in beats/min. **Default value** none

Details

Usually this computation function can not be used with %>%.

Value

RR interval in ms:

$$\frac{60000}{HR}$$

See Also

derive_param_rr()

BDS-Findings Functions that returns a vector: compute_bmi(), compute_bsa(), compute_egfr(), compute_framingham(), compute_map(), compute_qtc(), compute_qual_imputation(), compute_qual_imputation_d compute_scale(), transform_range()

Examples

```
compute_rr(hr = 70.14)
```

`compute_scale` *Compute Scale Parameters*

Description

Computes the average of a set of source values and transforms the result from the source range to the target range. For example, for calculating the average of a set of questionnaire response scores and re-coding the average response to obtain a subscale score.

Usage

```
compute_scale(
  source,
  source_range = NULL,
  target_range = NULL,
  flip_direction = FALSE,
  min_n = 1
)
```

Arguments

`source`	A vector of values to be scaled A numeric vector is expected. **Default value** none
`source_range`	The permitted source range A numeric vector containing two elements is expected, representing the lower and upper bounds of the permitted source range. Alternatively, if no argument is specified for `source_range` and `target_range`, no transformation will be performed. **Default value** `NULL`
`target_range`	The target range A numeric vector containing two elements is expected, representing the lower and upper bounds of the target range. Alternatively, if no argument is specified for `source_range` and `target_range`, no transformation will be performed. **Default value** `NULL`
`flip_direction`	Flip direction of the scale? The transformed values will be reversed within the target range, e.g. within the range 0 to 100, 25 would be reversed to 75. This argument will be ignored if `source_range` and `target_range` aren't specified. **Permitted values** `TRUE`, `FALSE` **Default value** `FALSE`

`min_n` Minimum number of values for computation

The minimum number of non-missing values in source for the computation to be carried out. If the number of non-missing values is below `min_n`, the result will be set to missing, i.e. `NA`.

A positive integer is expected.

Default value `1`

Details

Returns a numeric value. If source contains less than `min_n` values, the result is set to `NA`. If `source_range` and `target_range` aren't specified, the mean will be computed without any transformation being performed.

This is a vector-oriented helper and is not usually called directly on a data frame with `%>%`.

Value

The average of source transformed to the target range or `NA` if source doesn't contain `min_n` values.

See Also

BDS-Findings Functions that returns a vector: `compute_bmi()`, `compute_bsa()`, `compute_egfr()`, `compute_framingham()`, `compute_map()`, `compute_qtc()`, `compute_qual_imputation()`, `compute_qual_imputation_d`
`compute_rr()`, `transform_range()`

Examples

```
compute_scale(
  source = c(1, 4, 3, 5),
  source_range = c(1, 5),
  target_range = c(0, 100),
  flip_direction = TRUE,
  min_n = 3
)
```

---

compute_tmf *Derive the Time Imputation Flag*

Description

Derive the time imputation flag (`*TMF`) comparing a date character vector (`--DTC`) with a Datetime vector (`*DTM`).

Usage

```
compute_tmf(dtc, dtm, ignore_seconds_flag = TRUE)
```

Arguments

dtc	The date character vector (--DTC). A character date is expected in a format like yyyy-mm-ddThh:mm:ss (partial or complete). **Default value** none
dtm	The Date vector to compare (*DTM). A datetime object is expected. **Default value** none
ignore_seconds_flag	ADaM IG states that given SDTM (--DTC) variable, if only hours and minutes are ever collected, and seconds are imputed in (*DTM) as 00, then it is not necessary to set (*TMF) to "S". By default it is assumed that no seconds are collected and *TMF shouldn't be set to "S". A user can set this to FALSE if seconds are collected. The default value of ignore_seconds_flag is set to TRUE in admiral 1.4.0 and later. **Permitted values** TRUE, FALSE **Default value** TRUE

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

The time imputation flag (*TMF) (character value of "H", "M" , "S" or NA)

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(), impute_dtc_dt(), impute_dtc_dtm()

Examples

```
library(lubridate)

compute_tmf(dtc = "2019-07-18T15:25", dtm = ymd_hm("2019-07-18T15:25"))
compute_tmf(dtc = "2019-07-18T15", dtm = ymd_hm("2019-07-18T15:25"))
compute_tmf(dtc = "2019-07-18", dtm = ymd("2019-07-18"))
compute_tmf(dtc = "2022-05--T00:00", dtm = ymd_hm("2022-05-15T23:59"))
compute_tmf(dtc = "2022-05--T23:00", dtm = ymd_hm("2022-05-15T23:59"))
compute_tmf(
  dtc = "2022-05--T23:59:00",
  dtm = ymd_hms("2022-05-15T23:59:59"),
  ignore_seconds_flag = FALSE
)
```

`consolidate_metadata` *Consolidate Multiple Meta Datasets Into a Single One*

Description

The purpose of the function is to consolidate multiple meta datasets into a single one. For example, from global and project specific parameter mappings a single lookup table can be created.

Usage

```
consolidate_metadata(
  datasets,
  key_vars,
  source_var = SOURCE,
  check_vars = "warning",
  check_type = "error"
)
```

Arguments

`datasets`	List of datasets to consolidate **Permitted values** A named list of datasets **Default value** none
`key_vars`	Key variables The specified variables must be a unique of all input datasets. **Permitted values** A list of variables created by `exprs()` **Default value** none
`source_var`	Source variable The specified variable is added to the output dataset. It is set the name of the dataset the observation is originating from. **Permitted values** A symbol **Default value** `SOURCE`
`check_vars`	Check variables? If `"message"`, `"warning"`, or `"error"` is specified, a message is issued if the variable names differ across the input datasets (`datasets`). **Permitted values** `"none"`, `"message"`, `"warning"`, `"error"` **Default value** `"warning"`
`check_type`	Check uniqueness? If `"warning"` or `"error"` is specified, a message is issued if the key variables (`key_vars`) are not a unique key in all of the input datasets (`datasets`). **Permitted values** `"none"`, `"warning"`, `"error"` **Default value** `"error"`

Details

All observations of the input datasets are put together into a single dataset. If a by group (defined by `key_vars`) exists in more than one of the input datasets, the observation from the last dataset is selected.

Value

A dataset which contains one row for each by group occurring in any of the input datasets.

See Also

Creating auxiliary datasets: `create_period_dataset()`, `create_query_data()`, `create_single_dose_dataset()`

Examples

```
library(tibble)
glob_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI,
  "PULSE",      60,    100,
  "SYSBP",      90,    130,
  "DIABP",      60,     80
)
proj_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI,
  "SYSBP",     100,    140,
  "DIABP",      70,     90
)
stud_ranges <- tribble(
  ~PARAMCD, ~ANRLO, ~ANRHI,
  "BMI",        18,     25
)

consolidate_metadata(
  datasets = list(
    global = glob_ranges,
    project = proj_ranges,
    study = stud_ranges
  ),
  key_vars = exprs(PARAMCD)
)
```

---

convert_blanks_to_na *Convert Blank Strings Into NAs*

Description

Turn SAS blank strings into proper R `NA`s.

Usage

```
convert_blanks_to_na(x)

## Default S3 method:
convert_blanks_to_na(x)

## S3 method for class 'character'
convert_blanks_to_na(x)

## S3 method for class 'list'
convert_blanks_to_na(x)

## S3 method for class 'data.frame'
convert_blanks_to_na(x)
```

Arguments

x	Any R object
	Default value none

Details

The default methods simply returns its input unchanged. The `character` method turns every instance of `""` into `NA_character_` while preserving *all* attributes. When given a data frame as input the function keeps all non-character columns as is and applies the just described logic to `character` columns. Once again all attributes such as labels are preserved.

Value

An object of the same class as the input

See Also

Utilities for Formatting Observations: `convert_na_to_blanks()`, `yn_to_numeric()`

Examples

```
library(tibble)

convert_blanks_to_na(c("a", "b", "", "d", ""))

df <- tribble(
  ~USUBJID,   ~RFICDTC,
  "1001", "2000-01-01",
  "1002", "2001-01-01",
  "1003",           ""
)
print(df)
convert_blanks_to_na(df)
```

convert_date_to_dtm *Convert a Date into a Datetime Object*

Description

Convert a date (datetime, date, or date character) into a Date vector (usually *DTM).

Note: This is a wrapper function for the function convert_dtc_to_dtm().

Usage

```
convert_date_to_dtm(
  dt,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

dt

The date to convert.

A date or character date is expected in a format like yyyy-mm-ddThh:mm:ss.

Default value none

highest_imputation

Highest imputation level

The highest_imputation argument controls which components of the --DTC value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, NA is returned. For example, for highest_imputation = "D" "2020" results in NA because the month is missing.

If "n" is specified, no imputation is performed, i.e., if any component is missing, NA is returned.

If "Y" is specified, date_imputation should be "first" or "last" and min_dates or max_dates should be specified respectively. Otherwise, NA is returned if the year component is missing.

Permitted values "Y" (year, highest level), "M" (month), "D" (day), "h" (hour), "m" (minute), "s" (second), "n" (none, lowest level)

Default value "h"

date_imputation

The value to impute the day/month when a datepart is missing.

A character value is expected.

- The"first" and "last" keywords allow imputation to the first/last day/month. They can also be used to impute the year if used in conjunction with the min_dates or max_dates arguments. Some examples of this are available here.
- When highest_imputation is "M" or "D", the "mid" keyword can also be specified to impute missing components to the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).
- "<dd>" can be specified only if highest_imputation = "D". Missing days are imputed by the specified day, e.g. "10" for the 10th day of the month. The specified day should be valid for all months as otherwise an error might be issued. For example, date_imputation = "30" results in an invalid date of "2024-02-30" for the partial date "2024-02".
- "<mm>-<dd>" can be specified only if highest_imputation is "M", e.g. "06-15" for the 15th of June.

Permitted values a key-word, i.e. "first", "mid", "last", or "<mm>-<dd>" or "<dd>"

Default value "first"

time_imputation

The value to impute the time when a timepart is missing.

A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "first","last" to impute to the start/end of a day.

The argument is ignored if highest_imputation = "n".

Permitted values "first", "last", or user-defined

Default value "first"

min_dates Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
   ymd_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  highest_imputation = "M"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

For date variables (not datetime) in the list the time is imputed to "00:00:00". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

max_dates — Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

For date variables (not datetime) in the list the time is imputed to "23:59:59". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

preserve — Preserve lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.

For example "2019---07" would return "2019-06-07 if preserve = TRUE (and date_imputation = "mid").

Permitted values TRUE, FALSE

Default value FALSE

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

A datetime object

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(), impute_dtc_dt(), impute_dtc_dtm()

Examples

```
convert_date_to_dtm("2019-07-18T15:25:00")
convert_date_to_dtm(Sys.time())
convert_date_to_dtm(as.Date("2019-07-18"), time_imputation = "23:59:59")
```

```
convert_date_to_dtm("2019-07-18", time_imputation = "23:59:59")
convert_date_to_dtm("2019-07-18")
```

`convert_dtc_to_dt` *Convert a Date Character Vector into a Date Object*

Description

Convert a date character vector (usually `--DTC`) into a Date vector (usually `*DT`).

Usage

```
convert_dtc_to_dt(
  dtc,
  highest_imputation = "n",
  date_imputation = "first",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

`dtc`

The –DTC date to convert.

Permitted values a character date vector

Default value none

`highest_imputation`

Highest imputation level

The `highest_imputation` argument controls which components of the `--DTC` value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, `NA` is returned. For example, for `highest_imputation = "D"` `"2020"` results in `NA` because the month is missing.

If `"n"` (none, lowest level) is specified no imputation is performed, i.e., if any component is missing, `NA` is returned.

If `"Y"` (year, highest level) is specified, `date_imputation` must be `"first"` or `"last"` and `min_dates` or `max_dates` must be specified respectively. Otherwise, an error is thrown.

Permitted values `"Y"` (year, highest level), `"M"` (month), `"D"` (day), `"n"` (none, lowest level)

Default value `"n"`

`date_imputation`

The value to impute the day/month when a datepart is missing.

A character value is expected.

- The"first" and "last" keywords allow imputation to the first/last day/month. They can also be used to impute the year if used in conjunction with the min_dates or max_dates arguments. Some examples of this are available here.
- When highest_imputation is "M" or "D", the "mid" keyword can also be specified to impute missing components to the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).
- "<dd>" can be specified only if highest_imputation = "D". Missing days are imputed by the specified day, e.g. "10" for the 10th day of the month. The specified day should be valid for all months as otherwise an error might be issued. For example, date_imputation = "30" results in an invalid date of "2024-02-30" for the partial date "2024-02".
- "<mm>-<dd>" can be specified only if highest_imputation is "M", e.g. "06-15" for the 15th of June.

Permitted values a key-word, i.e. "first", "mid", "last", or "<mm>-<dd>" or "<dd>"

Default value "first"

min_dates — Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
   ymd_hms("2020-12-06T12:12:12"),
   ymd_hms("2020-11-11T11:11:11")
  ),
  highest_imputation = "M"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

max_dates — Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

Permitted values a list of dates, e.g. `list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))`

Default value `NULL`

`preserve` Preserve day if month is missing and day is present

For example `"2019---07"` would return `"2019-06-07"` if `preserve = TRUE` (and `date_imputation = "MID"`).

Permitted values `TRUE`, `FALSE`

Default value `FALSE`

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

a date object

See Also

Date/Time Computation Functions that returns a vector: `compute_age_years()`, `compute_dtf()`, `compute_duration()`, `compute_tmf()`, `convert_date_to_dtm()`, `convert_dtc_to_dtm()`, `convert_xxtpt_to_hours()`, `impute_dtc_dt()`, `impute_dtc_dtm()`

Examples

```
convert_dtc_to_dt("2019-07-18")
convert_dtc_to_dt("2019-07")
```

---

convert_dtc_to_dtm *Convert a Date Character Vector into a Datetime Object*

Description

Convert a date character vector (usually `--DTC`) into a Date vector (usually `*DTM`).

Usage

```
convert_dtc_to_dtm(
  dtc,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

dtc

The --DTC date to convert.

Permitted values a character date vector

Default value none

highest_imputation

Highest imputation level

The highest_imputation argument controls which components of the --DTC value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, NA is returned. For example, for highest_imputation = "D" "2020" results in NA because the month is missing.

If "n" is specified, no imputation is performed, i.e., if any component is missing, NA is returned.

If "Y" is specified, date_imputation should be "first" or "last" and min_dates or max_dates should be specified respectively. Otherwise, NA is returned if the year component is missing.

Permitted values "Y" (year, highest level), "M" (month), "D" (day), "h" (hour), "m" (minute), "s" (second), "n" (none, lowest level)

Default value "h"

date_imputation

The value to impute the day/month when a datepart is missing.

A character value is expected.

- The"first" and "last" keywords allow imputation to the first/last day/month. They can also be used to impute the year if used in conjunction with the min_dates or max_dates arguments. Some examples of this are available here.
- When highest_imputation is "M" or "D", the "mid" keyword can also be specified to impute missing components to the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).
- "<dd>" can be specified only if highest_imputation = "D". Missing days are imputed by the specified day, e.g. "10" for the 10th day of the month. The specified day should be valid for all months as otherwise an error might be issued. For example, date_imputation = "30" results in an invalid date of "2024-02-30" for the partial date "2024-02".
- "<mm>-<dd>" can be specified only if highest_imputation is "M", e.g. "06-15" for the 15th of June.

Permitted values a key-word, i.e. "first", "mid", "last", or "<mm>-<dd>" or "<dd>"

Default value "first"

time_imputation

The value to impute the time when a timepart is missing.

A character value is expected, either as a

- format with hour, min and sec specified as `"hh:mm:ss"`: e.g. `"00:00:00"` for the start of the day,
- or as a keyword: `"first"`,`"last"` to impute to the start/end of a day.

The argument is ignored if `highest_imputation = "n"`.

Permitted values `"first"`, `"last"`, or user-defined

Default value `"first"`

`min_dates` Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the `dtc` value are considered. The possible dates are defined by the missing parts of the `dtc` date (see example below). This ensures that the non-missing parts of the `dtc` date are not changed. A date or date-time object is expected. For example

```
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
   ymd_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  highest_imputation = "M"
)
```

returns `"2020-11-11T11:11:11"` because the possible dates for `"2020-11"` range from `"2020-11-01T00:00:00"` to `"2020-11-30T23:59:59"`. Therefore `"2020-12-06T12:12:12"` is ignored. Returning `"2020-12-06T12:12:12"` would have changed the month although it is not missing (in the `dtc` date).

For date variables (not datetime) in the list the time is imputed to `"00:00:00"`. Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. `list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))`

Default value `NULL`

`max_dates` Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

For date variables (not datetime) in the list the time is imputed to `"23:59:59"`. Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. `list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))`

Default value `NULL`

preserve — Preserve lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.
For example "2019---07" would return "2019-06-07 if preserve = TRUE (and date_imputation = "mid").

Permitted values TRUE, FALSE
Default value FALSE

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

A datetime object

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_xxtpt_to_hours(), impute_dtc_dt(), impute_dtc_dtm()

Examples

```
convert_dtc_to_dtm("2019-07-18T15:25:00")
convert_dtc_to_dtm("2019-07-18T00:00:00") # note Time = 00:00:00 is not printed
convert_dtc_to_dtm("2019-07-18")
```

---

convert_na_to_blanks *Convert NAs Into Blank Strings*

Description

Turn NAs to blank strings .

Usage

```
convert_na_to_blanks(x)

## Default S3 method:
convert_na_to_blanks(x)

## S3 method for class 'character'
convert_na_to_blanks(x)

## S3 method for class 'list'
convert_na_to_blanks(x)

## S3 method for class 'data.frame'
convert_na_to_blanks(x)
```

Arguments

x	Any R object
	Default value none

Details

The default methods simply returns its input unchanged. The `character` method turns every instance of `NA_character_` or `NA` into `""` while preserving *all* attributes. When given a data frame as input the function keeps all non-character columns as is and applies the just described logic to `character` all attributes such as labels are preserved.

Value

An object of the same class as the input

See Also

Utilities for Formatting Observations: `convert_blanks_to_na()`, `yn_to_numeric()`

Examples

```
library(tibble)

convert_na_to_blanks(c("a", "b", NA, "d", NA))

df <- tribble(
  ~USUBJID,   ~RFICDTC,
  "1001", "2000-01-01",
  "1002", "2001-01-01",
  "1003",           NA
)
print(df)
convert_na_to_blanks(df)
```

---

convert_xxtpt_to_hours

Convert `XXTPT` *Strings to Hours*

Description

[Experimental]

Converts CDISC timepoint strings (e.g., `PCTPT`, `VSTPT`, `EGTPT`, `ISTPT`, `LBTPT`) into numeric hours for analysis. The function handles common dose-centric formats including pre-dose, post-dose (hours/minutes), days, time ranges, and treatment-related time markers.

Usage

```
convert_xxtpt_to_hours(
  xxtpt,
  treatment_duration = 0,
  range_method = "midpoint"
)
```

Arguments

xxtpt	A character vector of timepoint descriptions from SDTM --TPT variables (e.g., PCTPT, VSTPT, EGTPT, ISTPT, LBTPT). Can contain NA values. **Permitted values** character vector **Default value** none
treatment_duration	Numeric value(s) specifying the duration of treatment in hours. Used to convert "EOI/EOT" (End of Infusion/Treatment) patterns and patterns describing time after end of treatment. Must be non-negative. Can be either: • A single value (used for all timepoints), or • A vector of the same length as xxtpt (one value per timepoint) Default is 0 hours (for instantaneous treatments like oral medications). **Permitted values** numeric scalar or vector (non-negative) **Default value** 0
range_method	Method for converting time ranges to single values. Options are "midpoint" (default), "start", or "end". For example, "0-6h" with midpoint returns 3, with start returns 0, with end returns 6. **Permitted values** character scalar ("midpoint", "start", or "end") **Default value** "midpoint"

Details

The function recognizes the following patterns (all case-insensitive):

Special Cases:

- "Screening" -> 0
- "Pre-dose", "Predose", "Pre-treatment", "Pre-infusion", "Pre-inf", "Before", "Infusion", "0H" -> 0
- "EOI", "EOT", "End of Infusion", "End of Treatment", "After End of Infusion", "After End of Treatment" -> treatment_duration (default: 0)
- "Morning", "Evening" -> NA_real_
- Unrecognized values -> NA_real_

Time Ranges: Time ranges are converted based on the range_method parameter:

- "0-6h Post-dose" with range_method = "midpoint" (default) -> 3
- "0-6h Post-dose" with range_method = "start" -> 0

- "0-6h Post-dose" with range_method = "end" -> 6
- "0-4H PRIOR START OF INFUSION" with midpoint -> -2 (negative for prior)
- "8-16H POST START OF INFUSION" with midpoint -> 12
- "0-4H AFTER EOI" with midpoint and treatment_duration=1 -> 3 (1 + 2)
- "0-4H EOT" with midpoint and treatment_duration=0 -> 2
- "4-8H AFTER END OF INFUSION" with midpoint and treatment_duration=1 -> 7 (1 + 6)
- "4-8H POST INFUSION" with midpoint and treatment_duration=1 -> 7 (1 + 6)
- "4-8H POST-INF" with midpoint and treatment_duration=1 -> 7 (1 + 6)

Time-based Conversions:

- **Days**: "Day 1" -> 24, "2D" -> 48, "30 DAYS AFTER LAST" -> 720 (requires unit indicator; bare numbers like "2" return NA)
- **Hours + Minutes**: "1H30M" -> 1.5
- **Hours**: "2 hours" -> 2, "1 HOUR POST" -> 1
- **Minutes**: "30M" -> 0.5, "30 MIN POST" -> 0.5
- **Predose**: "5 MIN PREDOSE" -> -0.0833, "5 MIN PRE-DOSE" -> -0.0833
- **Before treatment**: "5 MIN BEFORE" -> -0.0833
- **Post EOI/EOT**: "1 HOUR POST EOI" -> treatment_duration + 1, "24 HR POST INF" -> treatment_duration + 24, "24 HR POST-INF" -> treatment_duration + 24, "1 HOUR AFTER EOT" -> treatment_duration + 1
- **After end**: "30MIN AFTER END OF INFUSION" -> treatment_duration + 0.5
- **Start of infusion/treatment**: "8H PRIOR START OF INFUSION" -> -8, "8H BEFORE START OF TREATMENT" -> -8
- **Pre EOI/EOT**: "10MIN PRE EOI" -> treatment_duration - 1/6, "10MIN BEFORE EOT" -> treatment_duration - 1/6

Supported Unit Formats:

- Hours: H, h, HR, hr, HOUR, hour (with optional plurals)
- Minutes: M, m, MIN, min, MINUTE, minute (with optional plurals)
- Days: D, d, DAY, day (with optional plurals)
- Flexible whitespace and optional "Post-dose", "POST", "After last" suffixes
- Hyphens in compound terms: "PRE-DOSE", "POST-INF", "POST-INFUSION"

Understanding POST/AFTER Patterns:

It's important to distinguish between patterns relative to treatment **start** versus treatment **end**:

- **Relative to START** (treatment_duration NOT added):
  - "1H POST", "1H AFTER", "30M POST" -> Time from dose/treatment start
  - These patterns assume treatment starts at time 0
  - Example: "1H POST" -> 1 hour (regardless of treatment_duration)
- **Relative to END** (treatment_duration IS added):

- "1H POST EOI", "1H AFTER EOT", "1H POST INFUSION" -> Time from treatment end
- These patterns account for when treatment ends (start + duration)
- Example: "1H POST EOI" with treatment_duration=2 -> 3 hours (2 + 1)

This distinction follows standard pharmacokinetic conventions where "post-dose" refers to time from treatment initiation, while "post end of infusion" refers to time from treatment completion.

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Vectorized Treatment Duration:

When treatment_duration is a vector, each timepoint uses its corresponding treatment duration value. This is useful when different records have different treatment durations (e.g., different infusion lengths).

Value

A numeric vector of timepoints in hours. Returns NA_real_ for:

- Input NA values
- Unrecognized timepoint formats
- Non-time descriptors (e.g., "Morning", "Evening")

Returns numeric(0) for empty input.

Examples

Basic timepoint patterns:

Convert basic dose-centric patterns to hours

```
convert_xxtpt_to_hours(c(
  "Screening",
  "Pre-dose",
  "Pre-treatment",
  "Before",
  "30M",
  "1H",
  "2H POSTDOSE",
  "Day 1"
))
#> [1]  0.0  0.0  0.0  0.0  0.5  1.0  2.0 24.0
```

Predose and before patterns:

Convert predose/before patterns that return negative times

```
convert_xxtpt_to_hours(c("5 MIN PREDOSE", "5 MIN PRE-DOSE", "1 HOUR BEFORE"))
#> [1] -0.08333333 -0.08333333 -1.00000000
```

Treatment-related patterns (oral medications):

With default treatment_duration = 0 for oral medications

```
convert_xxtpt_to_hours(c(
  "EOT",
  "1 HOUR POST EOT",
  "1 HOUR AFTER EOT",
  "After End of Treatment"
))
#> [1] 0 1 1 0
```

Infusion-related patterns:

With treatment_duration = 1 hour for IV infusions

```
convert_xxtpt_to_hours(
  c(
    "EOI",
    "1 HOUR POST EOI",
    "24 HR POST INF",
    "24 HR POST-INF",
    "30MIN AFTER END OF INFUSION",
    "8H PRIOR START OF INFUSION",
    "10MIN PRE EOI"
  ),
  treatment_duration = 1
)
#> [1]  1.0000000  2.0000000 25.0000000 25.0000000  1.5000000 -8.0000000  0.8333333
```

Vectorized treatment duration:

Different treatment durations per timepoint

```
convert_xxtpt_to_hours(
  c("EOI", "1 HOUR POST EOI", "EOI", "1 HOUR POST EOI"),
  treatment_duration = c(1, 1, 2, 2)
)
#> [1] 1 2 2 3
```

Time ranges with midpoint method:

Default midpoint method for ranges

```
convert_xxtpt_to_hours(c(
  "0-6h Post-dose",
  "0-4H PRIOR START OF INFUSION",
  "8-16H POST START OF INFUSION"
))
#> [1]  3 -2 12
```

Time ranges with custom methods:

Specify start or end method for ranges

```
convert_xxtpt_to_hours("0-6h Post-dose", range_method = "end")
#> [1] 6
convert_xxtpt_to_hours("0-6h Post-dose", range_method = "start")
#> [1] 0
```

Ranges relative to EOI/EOT:

Time ranges after end of infusion/treatment

```
convert_xxtpt_to_hours(
  c(
    "0-4H AFTER EOI",
    "0-4H POST EOI",
    "4-8H AFTER END OF INFUSION",
    "4-8H AFTER EOT",
    "4-8H POST INFUSION",
    "4-8H POST-INF"
  ),
  treatment_duration = 1
)
#> [1] 3 3 7 7 7 7
```

POST vs POST EOI distinction:

Difference between POST (from start) and POST EOI (from end)

```
convert_xxtpt_to_hours(
  c("Pre-dose", "1H POST", "2H POST", "4H POST"),
  treatment_duration = 2
)
#> [1] 0 1 2 4

convert_xxtpt_to_hours(
  c("Pre-dose", "EOI", "1H POST EOI", "2H POST EOI"),
  treatment_duration = 2
)
#> [1] 0 2 3 4

convert_xxtpt_to_hours(
  c("1H POST", "1H POST EOI", "1H POST INFUSION"),
  treatment_duration = 2
)
#> [1] 1 3 3
```

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), impute_dtc_dt(), impute_dtc_dtm()

---

country_code_lookup *Country Code Lookup*

Description

These pre-defined country codes are sourced from ISO 3166 Standards. See also Wikipedia.

Usage

```
country_code_lookup
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 249 rows and 3 columns.

Details

`country_code` is the 3-letter ISO 3166-1 county code commonly found in the ADSL `COUNTRY` variable. `country_name` is the country long name corresponding to to the 3-letter code. `country_number` is the numeric code corresponding to an alphabetic sorting of the 3-letter codes.

To see the entire table in the console, run `print(country_code_lookup)`.

See Also

dose_freq_lookup

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, dose_freq_lookup

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

# Create reference dataset for periods
adsl <- tribble(
  ~USUBJID, ~SEX, ~COUNTRY,
  "ST01-01", "F", "AUT",
  "ST01-02", "M", "MWI",
  "ST01-03", "F", "GBR",
  "ST01-04", "M", "CHE",
  "ST01-05", "M", "NOR",
  "ST01-06", "F", "JPN",
  "ST01-07", "F", "USA"
)

adsl %>%
  derive_vars_merged(
    dataset_add = country_code_lookup,
    new_vars = exprs(COUNTRYN = country_number, COUNTRYL = country_name),
    by_vars = exprs(COUNTRY = country_code)
  )
```

count_vals *Count Number of Observations Where a Variable Equals a Value*

Description

Count number of observations where a variable equals a value.

Usage

```
count_vals(var, val)
```

Arguments

var	A vector **Default value** none
val	A value **Default value** none

See Also

Utilities for Filtering Observations: filter_exist(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), max_cond(), min_cond()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(admiral)
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "NE",
  "1",      4,        "CR",
  "1",      5,        "NE",
  "2",      1,        "CR",
  "2",      2,        "PR",
  "2",      3,        "CR",
  "3",      1,        "CR",
  "4",      1,        "CR",
  "4",      2,        "NE",
  "4",      3,        "NE",
  "4",      4,        "CR",
  "4",      5,        "PR"
)

# add variable providing the number of NEs for each subject
group_by(data, USUBJID) %>%
  mutate(nr_nes = count_vals(var = AVALC, val = "NE"))
```

`create_period_dataset` *Create a Reference Dataset for Subperiods, Periods, or Phases*

Description

The function creates a reference dataset for subperiods, periods, or phases from the ADSL dataset. The reference dataset can be used to derive subperiod, period, or phase variables like ASPER, ASPRSDT, ASPREDT, APERIOD, APERSDT, APEREDT, TRTA, APHASEN, PHSDTM, PHEDTM, ... in OCCDS and BDS datasets.

Usage

```
create_period_dataset(
  dataset,
  new_vars,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

`dataset`	Input dataset The variables specified by the `new_vars` and `subject_keys` arguments are expected to be in the dataset. For each element of `new_vars` at least one variable of the form of the right hand side value must be available in the dataset. **Default value** none
`new_vars`	New variables A named list of variables like `exprs(PHSDT = PHwSDT, PHEDT = PHwEDT, APHASE = APHASEw)` is expected. The left hand side of the elements defines a variable of the output dataset, the right hand side defines the source variables from the ADSL dataset in CDISC notation. If the lower case letter "w" is used it refers to a phase variable, if the lower case letters "xx" are used it refers to a period variable, and if both "xx" and "w" are used it refers to a subperiod variable. Only one type must be used, e.g., all right hand side values must refer to period variables. It is not allowed to mix for example period and subperiod variables. If period *and* subperiod variables are required, separate reference datasets must be created. **Default value** none
`subject_keys`	Variables to uniquely identify a subject A list of expressions where the expressions are symbols as returned by `exprs()` is expected. **Default value** `get_admiral_option("subject_keys")`

Details

For each subject and each subperiod/period/phase where at least one of the source variable is not NA an observation is added to the output dataset.

Depending on the type of the source variable (subperiod, period, or phase) the variable ASPER, APERIOD, or APHASEN is added and set to the number of the subperiod, period, or phase.

The variables specified for new_vars (left hand side) are added to the output dataset and set to the value of the source variable (right hand side).

Value

A period reference dataset (see "Details" section)

See Also

derive_vars_period()

Creating auxiliary datasets: consolidate_metadata(), create_query_data(), create_single_dose_dataset()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

# Create reference dataset for periods
adsl <- tribble(
  ~USUBJID, ~AP01SDT,     ~AP01EDT,     ~AP02SDT,     ~AP02EDT,     ~TRT01A, ~TRT02A,
  "1",      "2021-01-04", "2021-02-06", "2021-02-07", "2021-03-07", "A",     "B",
  "2",      "2021-02-02", "2021-03-02", "2021-03-03", "2021-04-01", "B",     "A",
) %>%
  mutate(
    across(matches("AP\\d\\d[ES]DT"), ymd)
  ) %>%
  mutate(
    STUDYID = "xyz"
  )

create_period_dataset(
  adsl,
  new_vars = exprs(APERSDT = APxxSDT, APEREDT = APxxEDT, TRTA = TRTxxA)
)

# Create reference dataset for phases
adsl <- tribble(
  ~USUBJID, ~PH1SDT,     ~PH1EDT,     ~PH2SDT,     ~PH2EDT,     ~APHASE1,    ~APHASE2,
  "1",      "2021-01-04", "2021-02-06", "2021-02-07", "2021-03-07", "TREATMENT", "FUP",
  "2",      "2021-02-02", "2021-03-02", NA,           NA,           "TREATMENT", NA
) %>%
  mutate(
    across(matches("PH\\d[ES]DT"), ymd)
  ) %>%
```

```
  mutate(
    STUDYID = "xyz"
  )

create_period_dataset(
  adsl,
  new_vars = exprs(PHSDT = PHwSDT, PHEDT = PHwEDT, APHASE = APHASEw)
)

# Create reference datasets for subperiods
adsl <- tribble(
  ~USUBJID, ~P01S1SDT,   ~P01S1EDT,   ~P01S2SDT,   ~P01S2EDT,   ~P02S1SDT,   ~P02S1EDT,
  "1",    "2021-01-04", "2021-01-19", "2021-01-20", "2021-02-06", "2021-02-07", "2021-03-07",
  "2",     "2021-02-02", "2021-03-02", NA,         NA,         "2021-03-03", "2021-04-01"
) %>%
  mutate(
    across(matches("P\\d\\dS\\d[ES]DT"), ymd)
  ) %>%
  mutate(
    STUDYID = "xyz"
  )

create_period_dataset(
  adsl,
  new_vars = exprs(ASPRSDT = PxxSwSDT, ASPREDT = PxxSwEDT)
)
```

`create_query_data` *Creates a queries dataset as input dataset to the* `dataset_queries` *argument in* `derive_vars_query()`

Description

Creates a queries dataset as input dataset to the `dataset_queries` argument in the `derive_vars_query()` function as defined in the `vignette("queries_dataset")`.

Usage

```
create_query_data(queries, version = NULL, get_terms_fun = NULL)
```

Arguments

`queries`	List of queries A list of `query()` objects is expected. **Default value** none
`version`	Dictionary version The dictionary version used for coding the terms should be specified. If any of the queries is a basket (SMQ, SDG, ...) or a customized query including a basket, the parameter needs to be specified.

Permitted values A character string (the expected format is company-specific)

Default value NULL

get_terms_fun Function which returns the terms

For each query specified for the queries parameter referring to a basket (i.e., those where the definition field is set to a basket_select() object or a list which contains at least one basket_select() object) the specified function is called to retrieve the terms defining the query. This function is not provided by admiral as it is company specific, i.e., it has to be implemented at company level.

The function must return a dataset with all the terms defining the basket. The output dataset must contain the following variables.

- SRCVAR: the variable to be used for defining a term of the basket, e.g., AEDECOD
- TERMCHAR: the name of the term if the variable SRCVAR is referring to is character
- TERMNUM the numeric id of the term if the variable SRCVAR is referring to is numeric
- GRPNAME: the name of the basket. The values must be the same for all observations.

The function must provide the following parameters

- basket_select: A basket_select() object.
- version: The dictionary version. The value specified for the version in the create_query_data() call is passed to this parameter.
- keep_id: If set to TRUE, the output dataset must contain the GRPID variable. The variable must be set to the numeric id of the basket.
- temp_env: A temporary environment is passed to this parameter. It can be used to store data which is used for all baskets in the create_query_data() call. For example if SMQs need to be read from a database all SMQs can be read and stored in the environment when the first SMQ is handled. For the other SMQs the terms can be retrieved from the environment instead of accessing the database again.

Default value NULL

Details

For each query() object listed in the queries argument, the terms belonging to the query (SRCVAR, TERMCHAR, TERMNUM) are determined with respect to the definition field of the query: if the definition field of the query() object is

- a basket_select() object, the terms are read from the basket database by calling the function specified for the get_terms_fun parameter.
- a data frame, the terms stored in the data frame are used.
- a list of data frames and basket_select() objects, all terms from the data frames and all terms read from the basket database referenced by the basket_select() objects are collated.

The following variables (as described in vignette("queries_dataset") are created:

- PREFIX: Prefix of the variables to be created by derive_vars_query() as specified by the prefix element.
- GRPNAME: Name of the query as specified by the name element.
- GRPID: Id of the query as specified by the id element. If the id element is not specified for a query, the variable is set to NA. If the id element is not specified for any query, the variable is not created.
- SCOPE: scope of the query as specified by the scope element of the basket_select() object. For queries not defined by a basket_select() object, the variable is set to NA. If none of the queries is defined by a basket_select() object, the variable is not created.
- SCOPEN: numeric scope of the query. It is set to 1 if the scope is broad. Otherwise it is set to 2. If the add_scope_num element equals FALSE, the variable is set to NA. If the add_scope_num element equals FALSE for all baskets or none of the queries is an basket , the variable is not created.
- SRCVAR: Name of the variable used to identify the terms.
- TERMCHAR: Value of the term variable if it is a character variable.
- TERMNUM: Value of the term variable if it is a numeric variable.
- VERSION: Set to the value of the version argument. If it is not specified, the variable is not created.

Value

A dataset to be used as input dataset to the dataset_queries argument in derive_vars_query()

See Also

derive_vars_query(), query(), basket_select(), vignette("queries_dataset")

Creating auxiliary datasets: consolidate_metadata(), create_period_dataset(), create_single_dose_dataset()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(pharmaversesdtm)
library(admiral)

# creating a query dataset for a customized query
cqterms <- tribble(
  ~TERMCHAR, ~TERMNUM,
  "APPLICATION SITE ERYTHEMA", 10003041L,
  "APPLICATION SITE PRURITUS", 10003053L
) %>%
  mutate(SRCVAR = "AEDECOD")

cq <- query(
  prefix = "CQ01",
  name = "Application Site Issues",
  definition = cqterms
)
```

```
create_query_data(queries = list(cq))

# create a query dataset for SMQs
pregsmq <- query(
  prefix = "SMQ02",
  id = auto,
  definition = basket_select(
    name = "Pregnancy and neonatal topics (SMQ)",
    scope = "NARROW",
    type = "smq"
  )
)

bilismq <- query(
  prefix = "SMQ04",
  definition = basket_select(
    id = 20000121L,
    scope = "BROAD",
    type = "smq"
  )
)

# The get_terms function from pharmaversesdtm is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(pregsmq, bilismq),
  get_terms_fun = pharmaversesdtm:::get_terms,
  version = "20.1"
)

# create a query dataset for SDGs
sdg <- query(
  prefix = "SDG01",
  id = auto,
  definition = basket_select(
    name = "5-aminosalicylates for ulcerative colitis",
    scope = NA_character_,
    type = "sdg"
  )
)

# The get_terms function from pharmaversesdtm is used for this example.
# In a real application a company-specific function must be used.
create_query_data(
  queries = list(sdg),
  get_terms_fun = pharmaversesdtm:::get_terms,
  version = "2019-09"
)

# creating a query dataset for a customized query including SMQs
# The get_terms function from pharmaversesdtm is used for this example.
# In a real application a company-specific function must be used.
```

```
create_query_data(
  queries = list(
    query(
      prefix = "CQ03",
      name = "Special issues of interest",
      definition = list(
        basket_select(
          name = "Pregnancy and neonatal topics (SMQ)",
          scope = "NARROW",
          type = "smq"
        ),
        cqterms
      )
    )
  ),
  get_terms_fun = pharmaversesdtm:::get_terms,
  version = "20.1"
)
```

`create_single_dose_dataset` *Create dataset of single doses*

Description

Derives dataset of single dose from aggregate dose information. This may be necessary when e.g. calculating last dose before an adverse event in `ADAE` or deriving a total dose parameter in `ADEX` when `EXDOSFRQ != ONCE`.

Usage

```
create_single_dose_dataset(
  dataset,
  dose_freq = EXDOSFRQ,
  start_date = ASTDT,
  start_datetime = NULL,
  end_date = AENDT,
  end_datetime = NULL,
  lookup_table = dose_freq_lookup,
  lookup_column = CDISC_VALUE,
  nominal_time = NULL,
 keep_source_vars = expr_c(get_admiral_option("subject_keys"), dose_freq, start_date,
    start_datetime, end_date, end_datetime)
)
```

Arguments

`dataset`	Input dataset

The variables specified by the dose_freq, start_date, and end_date arguments are expected to be in the dataset.

Default value none

dose_freq — The dose frequency

The aggregate dosing frequency used for multiple doses in a row.

Permitted values defined by lookup table.

Default value EXDOSFRQ

start_date — The start date

A date object is expected. This object cannot contain NA values.

Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

Default value ASTDT

start_datetime — The start date-time

A date-time object is expected. This object cannot contain NA values.

Refer to derive_vars_dtm() to impute and derive a date-time from a date character vector to a date object.

If the input dataset contains frequencies which refer to DOSE_WINDOW equals "HOUR" or "MINUTE", the parameter must be specified.

Default value NULL

end_date — The end date

A date or date-time object is expected. This object cannot contain NA values.

Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

Default value AENDT

end_datetime — The end date-time

A date-time object is expected. This object cannot contain NA values.

Refer to derive_vars_dtm() to impute and derive a date-time from a date character vector to a date object.

If the input dataset contains frequencies which refer to DOSE_WINDOW equals "HOUR" or "MINUTE", the parameter must be specified.

Default value NULL

lookup_table — The dose frequency value lookup table

The table used to look up dose_freq values and determine the appropriate multiplier to be used for row generation. If a lookup table other than the default is used, it must have columns DOSE_WINDOW, DOSE_COUNT, and CONVERSION_FACTOR. The default table dose_freq_lookup is described in detail here.

Permitted Values for DOSE_WINDOW: "MINUTE", "HOUR", "DAY", "WEEK", "MONTH", "YEAR"

Default value dose_freq_lookup

lookup_column — The dose frequency value column in the lookup table

The column of lookup_table.

Default value CDISC_VALUE

nominal_time
: The nominal relative time from first dose (NFRLT)

  Used for PK analysis, this will be in hours and should be 0 for the first dose. It can be derived as (VISITDY - 1) * 24 for example. This will be expanded as the single dose dataset is created. For example an EXDOFRQ of "QD" will result in the nominal_time being incremented by 24 hours for each expanded record.

  The value can be NULL if not needed.

  Default value NULL

keep_source_vars
: List of variables to be retained from source dataset

  This parameter can be specified if additional information is required in the output dataset. For example EXTRT for studies with more than one drug.

  Default value expr_c(get_admiral_option("subject_keys"), dose_freq, start_date,start_datetime, end_date, end_datetime)

Details

Each aggregate dose row is split into multiple rows which each represent a single dose.The number of completed dose periods between start_date or start_datetime and end_date or end_datetime is calculated with compute_duration and multiplied by DOSE_COUNT. For DOSE_WINDOW values of "WEEK", "MONTH", and "YEAR", CONVERSION_FACTOR is used to convert into days the time object to be added to start_date.

Observations with dose frequency "ONCE" are copied to the output dataset unchanged.

Value

The input dataset with a single dose per row.

See Also

Creating auxiliary datasets: consolidate_metadata(), create_period_dataset(), create_query_data()

Examples

```
# Example with default lookup

library(lubridate)
library(stringr)
library(tibble)
library(dplyr)

data <- tribble(
  ~STUDYID, ~USUBJID, ~EXDOSFRQ, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
  "STUDY01", "P01", "Q2D", ymd("2021-01-01"), ymd_hms("2021-01-01 10:30:00"),
  ymd("2021-01-07"), ymd_hms("2021-01-07 11:30:00"),
  "STUDY01", "P01", "Q3D", ymd("2021-01-08"), ymd_hms("2021-01-08 12:00:00"),
  ymd("2021-01-14"), ymd_hms("2021-01-14 14:00:00"),
  "STUDY01", "P01", "EVERY 2 WEEKS", ymd("2021-01-15"), ymd_hms("2021-01-15 09:57:00"),
  ymd("2021-01-29"), ymd_hms("2021-01-29 10:57:00")
```

```
)

create_single_dose_dataset(data)

# Example with custom lookup

custom_lookup <- tribble(
  ~Value,   ~DOSE_COUNT, ~DOSE_WINDOW, ~CONVERSION_FACTOR,
  "Q30MIN", (1 / 30),    "MINUTE",                     1,
  "Q90MIN", (1 / 90),    "MINUTE",                     1
)

data <- tribble(
  ~STUDYID, ~USUBJID, ~EXDOSFRQ, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
  "STUDY01", "P01", "Q30MIN", ymd("2021-01-01"), ymd_hms("2021-01-01T06:00:00"),
  ymd("2021-01-01"), ymd_hms("2021-01-01T07:00:00"),
  "STUDY02", "P02", "Q90MIN", ymd("2021-01-01"), ymd_hms("2021-01-01T06:00:00"),
  ymd("2021-01-01"), ymd_hms("2021-01-01T09:00:00")
)

create_single_dose_dataset(data,
  lookup_table = custom_lookup,
  lookup_column = Value,
  start_datetime = ASTDTM,
  end_datetime = AENDTM
)
# Example with nominal time

data <- tribble(
  ~STUDYID, ~USUBJID, ~EXDOSFRQ, ~NFRLT, ~ASTDT, ~ASTDTM, ~AENDT, ~AENDTM,
  "STUDY01", "P01", "BID", 0, ymd("2021-01-01"), ymd_hms("2021-01-01 08:00:00"),
  ymd("2021-01-07"), ymd_hms("2021-01-07 20:00:00"),
  "STUDY01", "P01", "BID", 168, ymd("2021-01-08"), ymd_hms("2021-01-08 08:00:00"),
  ymd("2021-01-14"), ymd_hms("2021-01-14 20:00:00"),
  "STUDY01", "P01", "BID", 336, ymd("2021-01-15"), ymd_hms("2021-01-15 08:00:00"),
  ymd("2021-01-29"), ymd_hms("2021-01-29 20:00:00")
)

create_single_dose_dataset(data,
  dose_freq = EXDOSFRQ,
  start_date = ASTDT,
  start_datetime = ASTDTM,
  end_date = AENDT,
  end_datetime = AENDTM,
  lookup_table = dose_freq_lookup,
  lookup_column = CDISC_VALUE,
  nominal_time = NFRLT,
  keep_source_vars = exprs(
    USUBJID, EXDOSFRQ, ASTDT, ASTDTM, AENDT, AENDTM, NFRLT
  )
)

# Example - derive a single dose dataset with imputations
```

```
# For either single drug administration records, or multiple drug administration
# records covering a range of dates, fill-in of missing treatment end datetime
# `EXENDTC` by substitution with an acceptable alternate, for example date of
# death, date of datacut may be required. This example shows the
# maximum possible number of single dose records to be derived. The example
# requires the date of datacut `DCUTDT` to be specified correctly, or
# if not appropriate to use `DCUTDT` as missing treatment end data and missing
# treatment end datetime could set equal to treatment start date and treatment
# start datetime. ADSL variables `DTHDT` and `DCUTDT` are preferred for
# imputation use.
#
# All available trial treatments are included, allowing multiple different
# last dose variables to be created in for example `use_ad_template("ADAE")`
# if required.

adsl <- tribble(
  ~STUDYID, ~USUBJID, ~DTHDT,
  "01", "1211", ymd("2013-01-14"),
  "01", "1083", ymd("2013-08-02"),
  "01", "1445", ymd("2014-11-01"),
  "01", "1015", NA,
  "01", "1023", NA
)

ex <- tribble(
  ~STUDYID, ~USUBJID, ~EXSEQ, ~EXTRT, ~EXDOSE, ~EXDOSU, ~EXDOSFRQ, ~EXSTDTC, ~EXENDTC,
  "01", "1015", 1, "PLAC", 0, "mg", "QD", "2014-01-02", "2014-01-16",
  "01", "1015", 2, "PLAC", 0, "mg", "QD", "2014-06-17", "2014-06-18",
  "01", "1015", 3, "PLAC", 0, "mg", "QD", "2014-06-19", NA_character_,
  "01", "1023", 1, "PLAC", 0, "mg", "QD", "2012-08-05", "2012-08-27",
  "01", "1023", 2, "PLAC", 0, "mg", "QD", "2012-08-28", "2012-09-01",
  "01", "1211", 1, "XANO", 54, "mg", "QD", "2012-11-15", "2012-11-28",
  "01", "1211", 2, "XANO", 54, "mg", "QD", "2012-11-29", NA_character_,
  "01", "1445", 1, "PLAC", 0, "mg", "QD", "2014-05-11", "2014-05-25",
  "01", "1445", 2, "PLAC", 0, "mg", "QD", "2014-05-26", "2014-11-01",
  "01", "1083", 1, "PLAC", 0, "mg", "QD", "2013-07-22", "2013-08-01"
)

adsl_death <- adsl %>%
  mutate(
    DTHDTM = convert_date_to_dtm(DTHDT),
    # Remove `DCUT` setup line below if ADSL `DCUTDT` is populated.
    DCUTDT = convert_dtc_to_dt("2015-03-06"), # Example only, enter date.
    DCUTDTM = convert_date_to_dtm(DCUTDT)
  )

# Select valid dose records, non-missing `EXSTDTC` and `EXDOSE`.
ex_mod <- ex %>%
  filter(!is.na(EXSTDTC) & !is.na(EXDOSE)) %>%
  derive_vars_merged(adsl_death, by_vars = get_admiral_option("subject_keys")) %>%
  # Example, set up missing `EXDOSFRQ` as QD daily dosing regime.
  # Replace with study dosing regime per trial treatment.
```

```
  mutate(EXDOSFRQ = if_else(is.na(EXDOSFRQ), "QD", EXDOSFRQ)) %>%
  # Create EXxxDTM variables and replace missing `EXENDTM`.
  derive_vars_dtm(
    dtc = EXSTDTC,
    new_vars_prefix = "EXST",
    date_imputation = "first",
    time_imputation = "first",
    flag_imputation = "none",
  ) %>%
  derive_vars_dtm_to_dt(exprs(EXSTDTM)) %>%
  derive_vars_dtm(
    dtc = EXENDTC,
    new_vars_prefix = "EXEN",
    # Maximum imputed treatment end date must not be not greater than
    # date of death or after the datacut date.
    max_dates = exprs(DTHDTM, DCUTDTM),
    date_imputation = "last",
    time_imputation = "last",
    flag_imputation = "none",
    highest_imputation = "Y",
  ) %>%
  derive_vars_dtm_to_dt(exprs(EXENDTM)) %>%
  # Select only unique values.
  # Removes duplicated records before final step.
  distinct(
    STUDYID, USUBJID, EXTRT, EXDOSE, EXDOSFRQ, DCUTDT, DTHDT, EXSTDT,
    EXSTDTM, EXENDT, EXENDTM, EXSTDTC, EXENDTC
  )

create_single_dose_dataset(
  ex_mod,
  start_date = EXSTDT,
  start_datetime = EXSTDTM,
  end_date = EXENDT,
  end_datetime = EXENDTM,
  keep_source_vars = exprs(
    STUDYID, USUBJID, EXTRT, EXDOSE, EXDOSFRQ,
    DCUTDT, EXSTDT, EXSTDTM, EXENDT, EXENDTM, EXSTDTC, EXENDTC
  )
)
```

---

date_source *Create a* date_source *object*

Description

[Deprecated] The `date_source()` function has been deprecated in favor of `event()`.

Create a `date_source` object as input for `derive_var_extreme_dt()` and `derive_var_extreme_dtm()`.

Usage

```
date_source(dataset_name, filter = NULL, date, set_values_to = NULL)
```

Arguments

dataset_name	The name of the dataset, i.e. a string, used to search for the date. **Default value** none
filter	An unquoted condition for filtering dataset. **Default value** NULL
date	A variable or an expression providing a date. A date or a datetime can be specified. An unquoted symbol or expression is expected. **Default value** none
set_values_to	Variables to be set **Default value** NULL

Value

An object of class date_source.

See Also

derive_var_extreme_dtm(), derive_var_extreme_dt()

Other deprecated: call_user_fun(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

Examples

```
# treatment end date from ADSL
trt_end_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDT
)

# lab date from LB where assessment was taken, i.e. not "NOT DONE"
lb_date <- date_source(
  dataset_name = "lb",
  filter = LBSTAT != "NOT DONE" | is.na(LBSTAT),
  date = convert_dtc_to_dt(LBDTC)
)

# death date from ADSL including traceability variables
death_date <- date_source(
  dataset_name = "adsl",
  date = DTHDT,
  set_values_to = exprs(
    LALVDOM = "ADSL",
```

```
    LALVVAR = "DTHDT"
  )
)
```

---

death_event *Pre-Defined Time-to-Event Source Objects*

Description

These pre-defined tte_source objects can be used as input to derive_param_tte().

Usage

```
death_event

lastalive_censor

ae_event

ae_ser_event

ae_gr1_event

ae_gr2_event

ae_gr3_event

ae_gr4_event

ae_gr5_event

ae_gr35_event

ae_sev_event

ae_wd_event
```

Details

To see the definition of the various objects simply print the object in the R console, e.g. print(death_event). For details of how to use these objects please refer to derive_param_tte().

See Also

derive_param_tte(), tte_source(), event_source(), censor_source()

Source Objects: basket_select(), censor_source(), event(), event_joined(), event_source(), flag_event(), query(), records_source(), tte_source()

Examples

```
# This shows the definition of all pre-defined `tte_source` objects that ship
# with {admiral}
for (obj in list_tte_source_objects()$object) {
  cat(obj, "\n")
  print(get(obj))
  cat("\n")
}
```

`default_qtc_paramcd` *Get Default Parameter Code for Corrected QT*

Description

Get Default Parameter Code for Corrected QT

Usage

```
default_qtc_paramcd(method)
```

Arguments

`method`	Method used to QT correction **Permitted values** `"Bazett"`, `"Fridericia"`, `"Sagie"` **Default value** none

Value

`"QTCBR"` if `method` is `"Bazett"`, `"QTCFR"` if it's `"Fridericia"` or `"QTLCR"` if it's `"Sagie"`. An error otherwise.

See Also

`derive_param_qtc()`

BDS-Findings Functions for adding Parameters/Records: `derive_basetype_records()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

Examples

```
default_qtc_paramcd("Sagie")
```

derivation_slice *Create a* derivation_slice *Object*

Description

Create a derivation_slice object as input for slice_derivation().

Usage

```
derivation_slice(filter, args = NULL)
```

Arguments

filter	An unquoted condition for defining the observations of the slice **Default value** none
args	Arguments of the derivation to be used for the slice A params() object is expected. **Default value** NULL

Value

An object of class derivation_slice

See Also

slice_derivation(), params()

Higher Order Functions: call_derivation(), restrict_derivation(), slice_derivation()

derivation_basetype_records *Derive Basetype Variable*

Description

Baseline Type BASETYPE is needed when there is more than one definition of baseline for a given Analysis Parameter PARAM in the same dataset. For a given parameter, if Baseline Value BASE or BASEC are derived and there is more than one definition of baseline, then BASETYPE must be non-null on all records of any type for that parameter where either BASE or BASEC are also non-null. Each value of BASETYPE refers to a definition of baseline that characterizes the value of BASE on that row. Please see section 4.2.1.6 of the ADaM Implementation Guide, version 1.3 for further background.

Usage

```
derive_basetype_records(dataset, basetypes)
```

Arguments

dataset	Input dataset The variables specified by the basetypes argument are expected to be in the dataset. **Default value** none
basetypes	A *named* list of expressions created using the rlang::exprs() function The names corresponds to the values of the newly created BASETYPE variables and the expressions are used to subset the input dataset. **Default value** none

Details

Adds the BASETYPE variable to a dataset and duplicates records based upon the provided conditions.

For each element of basetypes the input dataset is subset based upon the provided expression and the BASETYPE variable is set to the name of the expression. Then, all subsets are stacked. Records which do not match any condition are kept and BASETYPE is set to NA.

Value

The input dataset with variable BASETYPE added

Examples

Add records for different baseline types (basetypes):

The basetypes argument is a named list of expressions where each name becomes a value of BASETYPE and each expression defines which records receive that value. A record can match multiple expressions and will be duplicated once for each matching BASETYPE. In this example, records for subject P01 show the duplication across both baseline types.

Records that do not match any condition in basetypes are kept in the output dataset with BASETYPE set to NA. In this example, SCREENING records do not match any of the basetypes conditions and are therefore retained with BASETYPE = NA.

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

bds <- tribble(
  ~USUBJID, ~EPOCH,          ~PARAMCD,  ~ASEQ, ~AVAL,
  "P01",    "SCREENING",     "PARAM01",     1,  10.2,
  "P01",    "RUN-IN",        "PARAM01",     2,  10.0,
  "P01",    "RUN-IN",        "PARAM01",     3,   9.8,
  "P01",    "DOUBLE-BLIND",  "PARAM01",     4,   9.2,
  "P01",    "DOUBLE-BLIND",  "PARAM01",     5,  10.1,
  "P02",    "SCREENING",     "PARAM01",     1,  12.2,
  "P02",    "RUN-IN",        "PARAM01",     2,  12.1,
  "P02",    "DOUBLE-BLIND",  "PARAM01",     3,  10.2
)
```

```
derive_basetype_records(
  dataset = bds,
  basetypes = exprs(
    "RUN-IN" = EPOCH %in% c("RUN-IN", "DOUBLE-BLIND"),
    "DOUBLE-BLIND" = EPOCH == "DOUBLE-BLIND"
  )
)
#> # A tibble: 11 × 6
#>    USUBJID EPOCH        PARAMCD  ASEQ  AVAL BASETYPE
#>    <chr>   <chr>        <chr>   <dbl> <dbl> <chr>
#>  1 P01     SCREENING    PARAM01     1  10.2 <NA>
#>  2 P02     SCREENING    PARAM01     1  12.2 <NA>
#>  3 P01     RUN-IN       PARAM01     2  10   RUN-IN
#>  4 P01     RUN-IN       PARAM01     3   9.8 RUN-IN
#>  5 P01     DOUBLE-BLIND PARAM01     4   9.2 RUN-IN
#>  6 P01     DOUBLE-BLIND PARAM01     5  10.1 RUN-IN
#>  7 P02     RUN-IN       PARAM01     2  12.1 RUN-IN
#>  8 P02     DOUBLE-BLIND PARAM01     3  10.2 RUN-IN
#>  9 P01     DOUBLE-BLIND PARAM01     4   9.2 DOUBLE-BLIND
#> 10 P01     DOUBLE-BLIND PARAM01     5  10.1 DOUBLE-BLIND
#> 11 P02     DOUBLE-BLIND PARAM01     3  10.2 DOUBLE-BLIND
```

Include all records for multiple baseline type derivations (`basetypes = TRUE`):

When all parameter records need to be included for multiple baseline type derivations (such as `"LAST"` and `"WORST"`), set each expression in `basetypes` to `TRUE`. This duplicates every record once for each named baseline type.

```
bds <- tribble(
  ~USUBJID, ~EPOCH,          ~PARAMCD,  ~ASEQ, ~AVAL,
  "P01",    "RUN-IN",        "PARAM01",     1,  10.0,
  "P01",    "RUN-IN",        "PARAM01",     2,   9.8,
  "P01",    "DOUBLE-BLIND",  "PARAM01",     3,   9.2,
  "P01",    "DOUBLE-BLIND",  "PARAM01",     4,  10.1
)

derive_basetype_records(
  dataset = bds,
  basetypes = exprs(
    "LAST" = TRUE,
    "WORST" = TRUE
  )
)
#> # A tibble: 8 × 6
#>   USUBJID EPOCH        PARAMCD  ASEQ  AVAL BASETYPE
#>   <chr>   <chr>        <chr>   <dbl> <dbl> <chr>
#> 1 P01     RUN-IN       PARAM01     1  10   LAST
#> 2 P01     RUN-IN       PARAM01     2   9.8 LAST
#> 3 P01     DOUBLE-BLIND PARAM01     3   9.2 LAST
#> 4 P01     DOUBLE-BLIND PARAM01     4  10.1 LAST
```

```
#> 5 P01     RUN-IN       PARAM01     1  10   WORST
#> 6 P01     RUN-IN       PARAM01     2   9.8 WORST
#> 7 P01     DOUBLE-BLIND PARAM01     3   9.2 WORST
#> 8 P01     DOUBLE-BLIND PARAM01     4  10.1 WORST
```

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

---

derive_expected_records

Derive Expected Records

Description

Add expected records as new observations for each 'by group' when the dataset contains missing observations.

Usage

```
derive_expected_records(
  dataset,
  dataset_ref,
  by_vars = NULL,
  set_values_to = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the dataset_ref and by_vars arguments are expected to be in the dataset. **Default value** none
dataset_ref	Expected observations dataset Data frame with the expected observations, e.g., all the expected combinations of PARAMCD, PARAM, AVISIT, AVISITN, ... **Default value** none
by_vars	Grouping variables For each group defined by by_vars those observations from dataset_ref are added to the output dataset which do not have a corresponding observation in the input dataset.

Default value NULL

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations.

A list of variable name-value pairs is expected.

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, NA, or expressions, e.g., `exprs(PARAMCD = "TDOSE", PARCAT1 = "OVERALL")`.

Default value NULL

Details

For each group (the variables specified in the `by_vars` parameter), those records from `dataset_ref` that are missing in the input dataset are added to the output dataset.

Value

The input dataset with the missed expected observations added for each `by_vars`. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars` or `set_values_to`.

See Also

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_basetype_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

Examples

```
library(tibble)

adqs <- tribble(
  ~USUBJID, ~PARAMCD, ~AVISITN, ~AVISIT, ~AVAL,
  "1",      "a",             1, "WEEK 1",   10,
  "1",      "b",             1, "WEEK 1",   11,
  "2",      "a",             2, "WEEK 2",   12,
  "2",      "b",             2, "WEEK 2",   14
)

# Example 1. visit variables are parameter independent
parm_visit_ref <- tribble(
  ~AVISITN, ~AVISIT,
  1,        "WEEK 1",
  2,        "WEEK 2"
)

derive_expected_records(
  dataset = adqs,
  dataset_ref = parm_visit_ref,
```

```
  by_vars = exprs(USUBJID, PARAMCD),
  set_values_to = exprs(DTYPE = "DERIVED")
)

# Example 2. visit variables are parameter dependent
parm_visit_ref <- tribble(
  ~PARAMCD, ~AVISITN, ~AVISIT,
  "a",             1, "WEEK 1",
  "a",             2, "WEEK 2",
  "b",             1, "WEEK 1"
)

derive_expected_records(
  dataset = adqs,
  dataset_ref = parm_visit_ref,
  by_vars = exprs(USUBJID, PARAMCD),
  set_values_to = exprs(DTYPE = "DERIVED")
)
```

`derive_extreme_event` *Add the Worst or Best Observation for Each By Group as New Records*

Description

Add the first available record from `events` for each by group as new records, all variables of the selected observation are kept. It can be used for selecting the extreme observation from a series of user-defined events. This distinguishes `derive_extreme_event()` from `derive_extreme_records()`, where extreme records are derived based on certain order of existing variables.

Usage

```
derive_extreme_event(
  dataset = NULL,
  by_vars,
  events,
  tmp_event_nr_var = NULL,
  order,
  mode,
  source_datasets = NULL,
  check_type = "warning",
  set_values_to = NULL,
  keep_source_vars = exprs(everything())
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `order` arguments are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`by_vars` Grouping variables

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`events` Conditions and new values defining events

A list of `event()` or `event_joined()` objects is expected. Only observations listed in the `events` are considered for deriving extreme event. If multiple records meet the filter `condition`, take the first record sorted by `order`. The data is grouped by `by_vars`, i.e., summary functions like `all()` or `any()` can be used in `condition`.

For `event_joined()` events the observations are selected by calling `filter_joined()`. The `condition` field is passed to the `filter_join` argument.

Permitted values an `event()` or `event_joined()` object

Default value none

`tmp_event_nr_var`

Temporary event number variable

The specified variable is added to all source datasets and is set to the number of the event before selecting the records of the event.

It can be used in `order` to determine which record should be used if records from more than one event are selected.

The variable is not included in the output dataset.

Permitted values an unquoted symbol, e.g., `AVAL`

Default value `NULL`

`order` Sort order

If a particular event from `events` has more than one observation, within the event and by group, the records are ordered by the specified order.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`mode` Selection mode (first or last)

If a particular event from `events` has more than one observation, `"first"`/`"last"` is used to select the first/last record of this type of event sorting by `order`.

Permitted values `"first"`, `"last"`

Default value none

`source_datasets`

Source datasets

A named list of datasets is expected. The `dataset_name` field of `event()` and `event_joined()` refers to the dataset provided in the list.

Permitted values named list of datasets, e.g., `list(adsl = adsl, ae = ae)`

Default value NULL

check_type

Check uniqueness?

If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values "none", "message", "warning", "error"

Default value "warning"

set_values_to

Variables to be set

The specified variables are set to the specified values for the new observations.

Set a list of variables to some specified value for the new records

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression, or 'NA".

For example:

```
set_values_to = exprs(
  AVISIT = "LAST VALUE",
  DTYPE = "LOV"
)
```

Permitted values list of named expressions created by exprs(), e.g., exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))

Default value NULL

keep_source_vars

Variables to keep from the source dataset

For each event the specified variables are kept from the selected observations. The variables specified for by_vars and created by set_values_to are always kept. The keep_source_vars field of the event will take precedence over the value of the keep_source_vars argument.

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value exprs(everything())

Details

1. For each event select the observations to consider:
   (a) If the event is of class event, the observations of the source dataset are restricted by condition and then the first or last (mode) observation per by group (by_vars) is selected.
   If the event is of class event_joined, filter_joined() is called to select the observations.
   (b) The variables specified by the set_values_to field of the event are added to the selected observations.
   (c) The variable specified for tmp_event_nr_var is added and set to the number of the event.

(d) Only the variables specified for the keep_source_vars field of the event, and the by variables (by_vars) and the variables created by set_values_to are kept. If keep_source_vars = NULL is used for an event in derive_extreme_event() the value of the keep_source_vars argument of derive_extreme_event() is used.

2. All selected observations are bound together.
3. For each group (with respect to the variables specified for the by_vars parameter) the first or last observation (with respect to the order specified for the order parameter and the mode specified for the mode parameter) is selected.
4. The variables specified by the set_values_to parameter are added to the selected observations.
5. The observations are added to input dataset.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option save_memory to TRUE (see set_admiral_options()). This reduces the memory consumption but increases the run-time.

Value

The input dataset with the best or worst observation of each by group added as new observations.

Examples

Add a new record for the worst observation using event() objects:

For each subject, the observation containing the worst sleeping problem (if any exist) should be identified and added as a new record, retaining all variables from the original observation. If multiple occurrences of the worst sleeping problem occur, or no sleeping problems, then take the observation occurring at the latest day.

- The groups for which new records are added are specified by the by_vars argument. Here for each *subject* a record should be added. Thus by_vars = exprs(STUDYID, USUBJID) is specified.
- The sets of possible sleeping problems are passed through the events argument as event() objects. Each event contains a condition which may or may not be satisfied by each record (or possibly a group of records) within the input dataset dataset. Summary functions such as any() and all() are often handy to use within conditions, as is done here for the third event, which checks that the subject had no sleeping issues. The final event uses a catch-all condition = TRUE to ensure all subjects have a new record derived. Note that in this example, as no condition involves analysis of **cross-comparison values of within records**, it is sufficient to use event() objects rather than event_joined() - see the next example for a more complex condition.
- If any subject has one or more records satisfying the conditions from events, we can select just one record using the order argument. In this example, the first argument passed to order is event_nr, which is a temporary variable created through the tmp_event_nr_var argument, which numbers the events consecutively. Since mode = "first", we only consider the first event for which a condition is satisfied. Within that event, we consider only the observation with the latest day, because the second argument for the order is desc(ADY).
- Once a record is identified as satisfying an event's condition, a new observation is created by the following process:

1. the selected record is copied,
2. the variables specified in the event's set_values_to (here, AVAL and AVALC) are created/updated,
3. the variables specified in keep_source_vars (here, ADY does due to the use of the tidyselect expression everything()) (plus by_vars and the variables from set_values_to) are kept,
4. the variables specified in the global set_values_to (here, PARAM and PARAMCD) are created/updated.

```
library(tibble, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)

adqs1 <- tribble(
  ~USUBJID, ~PARAMCD,          ~AVALC,         ~ADY,
  "1",      "NO SLEEP",        "N",               1,
  "1",      "WAKE UP 3X",      "N",               2,
  "2",      "NO SLEEP",        "N",               1,
  "2",      "WAKE UP 3X",      "Y",               2,
  "2",      "WAKE UP 3X",      "Y",               3,
  "3",      "NO SLEEP",        NA_character_,     1
) %>%
mutate(STUDYID = "AB42")

derive_extreme_event(
  adqs1,
  by_vars = exprs(STUDYID, USUBJID),
  events = list(
    event(
      condition = PARAMCD == "NO SLEEP" & AVALC == "Y",
      set_values_to = exprs(AVALC = "No sleep", AVAL = 1)
    ),
    event(
      condition = PARAMCD == "WAKE UP 3X" & AVALC == "Y",
      set_values_to = exprs(AVALC = "Waking up three times", AVAL = 2)
    ),
    event(
      condition = all(AVALC == "N"),
      set_values_to = exprs(AVALC = "No sleeping problems", AVAL = 3)
    ),
    event(
      condition = TRUE,
      set_values_to = exprs(AVALC = "Missing", AVAL = 99)
    )
  ),
  tmp_event_nr_var = event_nr,
  order = exprs(event_nr, desc(ADY)),
  mode = "first",
  set_values_to = exprs(
```

```
    PARAMCD = "WSP",
    PARAM = "Worst Sleeping Problem"
  ),
  keep_source_vars = exprs(everything())
) %>%
select(-STUDYID)
#> # A tibble: 9 × 6
#>   USUBJID PARAMCD    AVALC                   ADY  AVAL PARAM
#>   <chr>   <chr>      <chr>                 <dbl> <dbl> <chr>
#> 1 1       NO SLEEP   N                         1    NA <NA>
#> 2 1       WAKE UP 3X N                         2    NA <NA>
#> 3 2       NO SLEEP   N                         1    NA <NA>
#> 4 2       WAKE UP 3X Y                         2    NA <NA>
#> 5 2       WAKE UP 3X Y                         3    NA <NA>
#> 6 3       NO SLEEP   <NA>                      1    NA <NA>
#> 7 1       WSP        No sleeping problems      2     3 Worst Sleeping Problem
#> 8 2       WSP        Waking up three times     3     2 Worst Sleeping Problem
#> 9 3       WSP        Missing                   1    99 Worst Sleeping Problem
```

Events based on comparison across records (`event_joined()`):

We'll now extend the above example. Specifically, we consider a new possible worst sleeping problem, namely if a subject experiences no sleep on consecutive days.

- The "consecutive days" portion of the condition requires records to be compared with each other. This is done by using an `event_joined()` object, specifically by passing `dataset_name = adqs2` to it so that the `adqs2` dataset is joined onto itself. The `condition` now checks for two no sleep records, and crucially compares the `ADY` values to see if they differ by one day. The `.join` syntax distinguishes between the `ADY` value of the parent and joined datasets. As the condition involves `AVALC`, `PARAMCD` and `ADY`, we specify these variables with `join_vars`, and finally, because we wish to compare all records with each other, we select `join_type = "all"`.

```
adqs2 <- tribble(
   ~USUBJID, ~PARAMCD,     ~AVALC, ~ADY,
   "4",      "WAKE UP",    "N",    1,
   "4",      "NO SLEEP",   "Y",    2,
   "4",      "NO SLEEP",   "Y",    3,
   "5",      "NO SLEEP",   "N",    1,
   "5",      "NO SLEEP",   "Y",    2,
   "5",      "WAKE UP 3X", "Y",    3,
   "5",      "NO SLEEP",   "Y",    4
) %>%
mutate(STUDYID = "AB42")

derive_extreme_event(
  adqs2,
  by_vars = exprs(STUDYID, USUBJID),
  events = list(
    event_joined(
      join_vars = exprs(AVALC, PARAMCD, ADY),
```

```
      join_type = "all",
      condition = PARAMCD == "NO SLEEP" & AVALC == "Y" &
        PARAMCD.join == "NO SLEEP" & AVALC.join == "Y" &
        ADY == ADY.join + 1,
      set_values_to = exprs(AVALC = "No sleep two nights in a row", AVAL = 0)
    ),
    event(
      condition = PARAMCD == "NO SLEEP" & AVALC == "Y",
      set_values_to = exprs(AVALC = "No sleep", AVAL = 1)
    ),
    event(
      condition = PARAMCD == "WAKE UP 3X" & AVALC == "Y",
      set_values_to = exprs(AVALC = "Waking up three times", AVAL = 2)
    ),
    event(
      condition = all(AVALC == "N"),
      set_values_to = exprs(
        AVALC = "No sleeping problems", AVAL = 3
      )
    ),
    event(
      condition = TRUE,
      set_values_to = exprs(AVALC = "Missing", AVAL = 99)
    )
  ),
  tmp_event_nr_var = event_nr,
  order = exprs(event_nr, desc(ADY)),
  mode = "first",
  set_values_to = exprs(
    PARAMCD = "WSP",
    PARAM = "Worst Sleeping Problem"
  ),
  keep_source_vars = exprs(everything())
) %>%
select(-STUDYID)
#> # A tibble: 9 × 6
#>   USUBJID PARAMCD    AVALC                          ADY  AVAL PARAM
#>   <chr>   <chr>      <chr>                        <dbl> <dbl> <chr>
#> 1 4       WAKE UP    N                                1    NA <NA>
#> 2 4       NO SLEEP   Y                                2    NA <NA>
#> 3 4       NO SLEEP   Y                                3    NA <NA>
#> 4 5       NO SLEEP   N                                1    NA <NA>
#> 5 5       NO SLEEP   Y                                2    NA <NA>
#> 6 5       WAKE UP 3X Y                                3    NA <NA>
#> 7 5       NO SLEEP   Y                                4    NA <NA>
#> 8 4      WSP      No sleep two nights in a row    3    0 Worst Sleeping Pr. . .
#> 9 5       WSP       No sleep                      4    1 Worst Sleeping Pr. . .
```

Events across records by record:

In the previous example, the new parameter was derived for each subject, i.e., all records of a subject were summarized. However, there are cases where we may want to derive a new parameter by record, but where the condition for deriving the parameter for a given record may involve comparison with other records.

For example, we want to derive a new parameter for each visit indicating response at this visit and the next one. For this we need to specify the `by_vars` argument in `event_joined()` which overwrites the value specified in the `derive_extreme_event()` call. There `by_vars = exprs(USUBJID, AVISITN)` is used because we want to add new records for each subject and visit. In `event_joined()`, `by_vars = exprs(USUBJID)` is used because we want to join the records by subject only.

The `tmp_obs_nr_var` argument is specified to create a variable which numbers the records within each subject. This variable is then used in the `condition` argument to ensure that the current record is compared with the next one. This ensures that missing visits like for subject 2 are handled correctly.

```
adbds <- tribble(
  ~USUBJID, ~AVISITN,  ~AVALC,
  "1",             1,  "Y",
  "1",             2,  "N",
  "1",             3,  "Y",
  "1",             4,  "Y",
  "1",             5,  "Y",
  "2",             1,  "Y",
  "2",             3,  "Y",
) %>%
  mutate(PARAMCD = "RESP")

derive_extreme_event(
  adbds,
  by_vars = exprs(USUBJID, AVISITN),
  source_datasets = list(adbds = adbds),
  tmp_event_nr_var = event_nr,
  order = exprs(event_nr),
  mode = "first",
  events = list(
    event_joined(
      dataset_name = "adbds",
      by_vars = exprs(USUBJID),
      order = exprs(AVISITN),
      join_vars = exprs(AVALC),
      join_type = "after",
      tmp_obs_nr_var = obs_nr,
      condition = AVALC == "Y" & AVALC.join == "Y" & obs_nr.join == obs_nr + 1,
      set_values_to = exprs(AVALC = "Y")
    ),
    event(
      dataset_name = "adbds",
      set_values_to = exprs(AVALC = "N")
    )
  ),
```

```
  set_values_to = exprs(PARAMCD = "CONFRESP")
)
#> # A tibble: 14 × 4
#>    USUBJID AVISITN AVALC PARAMCD
#>    <chr>     <dbl> <chr> <chr>
#>  1 1             1 Y     RESP
#>  2 1             2 N     RESP
#>  3 1             3 Y     RESP
#>  4 1             4 Y     RESP
#>  5 1             5 Y     RESP
#>  6 2             1 Y     RESP
#>  7 2             3 Y     RESP
#>  8 1             1 N     CONFRESP
#>  9 1             2 N     CONFRESP
#> 10 1             3 Y     CONFRESP
#> 11 1             4 Y     CONFRESP
#> 12 1             5 N     CONFRESP
#> 13 2             1 Y     CONFRESP
#> 14 2             3 N     CONFRESP
```

Restricting source data before join:

Sometimes it is useful to restrict the source data of some events before the join. For example, in the following example, records with missing results should be ignored for the confirmation.

```
adbds <- tribble(
  ~USUBJID, ~AVISITN,  ~AVALC,
  "1",             1,  "Y",
  "1",             2,  "N",
  "1",             3,  "Y",
  "1",             4,  "Y",
  "1",             5,  "Y",
  "2",             1,  "Y",
  "2",             2,  NA,
  "2",             3,  "Y"
) %>%
  mutate(PARAMCD = "RESP")

derive_extreme_event(
  adbds,
  by_vars = exprs(USUBJID, AVISITN),
  source_datasets = list(adbds = adbds),
  tmp_event_nr_var = event_nr,
  order = exprs(event_nr),
  mode = "first",
  events = list(
    event_joined(
      dataset_name = "adbds",
      filter_source = !is.na(AVALC),
      by_vars = exprs(USUBJID),
```

```
      order = exprs(AVISITN),
      join_vars = exprs(AVALC),
      join_type = "after",
      tmp_obs_nr_var = tmp_obs_nr,
    condition = AVALC == "Y" & AVALC.join == "Y" & tmp_obs_nr.join == tmp_obs_nr + 1,
      set_values_to = exprs(AVALC = "Y")
    ),
    event(
      dataset_name = "adbds",
      set_values_to = exprs(AVALC = "N")
    )
  ),
  set_values_to = exprs(
    PARAMCD = "CONFRESP"
  )
)
#> # A tibble: 16 × 4
#>    USUBJID AVISITN AVALC PARAMCD
#>    <chr>     <dbl> <chr> <chr>
#>  1 1             1 Y     RESP
#>  2 1             2 N     RESP
#>  3 1             3 Y     RESP
#>  4 1             4 Y     RESP
#>  5 1             5 Y     RESP
#>  6 2             1 Y     RESP
#>  7 2             2 <NA>  RESP
#>  8 2             3 Y     RESP
#>  9 1             1 N     CONFRESP
#> 10 1             2 N     CONFRESP
#> 11 1             3 Y     CONFRESP
#> 12 1             4 Y     CONFRESP
#> 13 1             5 N     CONFRESP
#> 14 2             1 Y     CONFRESP
#> 15 2             2 N     CONFRESP
#> 16 2             3 N     CONFRESP
```

Note that for subject 2 the results at visit 1 is considered as confirmed because the missing result at visit 2 is ignored.

Specifying different arguments across `event()` objects:

Here we consider a Hy's Law use case. We are interested in knowing whether a subject's Alkaline Phosphatase has ever been above twice the upper limit of normal range. If so, i.e. if `CRIT1FL` is `Y`, we are interested in the record for the first time this occurs, and if not, we wish to retain the last record. As such, for this case now we need to vary our usage of the `mode` argument dependent on the `event()`.

- In the first `event()`, since we simply seek the first time that `CRIT1FL` is `"Y"`, it's enough to specify the `condition`, because we inherit `order` and `mode` from the main `derive_extreme_event()` call here which will automatically select the first occurrence by `AVISITN`.

- In the second event(), we select the last record among the full set of records where CRIT1FL are all "N" by additionally specifying mode = "last" within the event().
- Note now the usage of keep_source_vars = exprs(AVISITN) rather than everything() as in the previous example. This is done to ensure CRIT1 and CRIT1FL are not populated for the new records.

```
adhy <- tribble(
  ~USUBJID, ~AVISITN,              ~CRIT1, ~CRIT1FL,
  "1",             1, "ALT > 2 times ULN", "N",
  "1",             2, "ALT > 2 times ULN", "N",
  "2",             1, "ALT > 2 times ULN", "N",
  "2",             2, "ALT > 2 times ULN", "Y",
  "2",             3, "ALT > 2 times ULN", "N",
  "2",             4, "ALT > 2 times ULN", "Y"
) %>%
  mutate(
    PARAMCD = "ALT",
    PARAM = "ALT (U/L)",
    STUDYID = "AB42"
  )

derive_extreme_event(
  adhy,
  by_vars = exprs(STUDYID, USUBJID),
  events = list(
    event(
      condition = CRIT1FL == "Y",
      set_values_to = exprs(AVALC = "Y")
    ),
    event(
      condition = CRIT1FL == "N",
      mode = "last",
      set_values_to = exprs(AVALC = "N")
    )
  ),
  tmp_event_nr_var = event_nr,
  order = exprs(event_nr, AVISITN),
  mode = "first",
  keep_source_vars = exprs(AVISITN),
  set_values_to = exprs(
    PARAMCD = "ALT2",
    PARAM = "ALT > 2 times ULN"
  )
) %>%
  select(-STUDYID)
#> # A tibble: 8 × 7
#>   USUBJID AVISITN CRIT1             CRIT1FL PARAMCD PARAM             AVALC
#>   <chr>     <dbl> <chr>             <chr>   <chr>   <chr>             <chr>
#> 1 1             1 ALT > 2 times ULN N       ALT     ALT (U/L)         <NA>
```

```
#> 2 1             2 ALT > 2 times ULN N       ALT     ALT (U/L)          <NA>
#> 3 2             1 ALT > 2 times ULN N       ALT     ALT (U/L)          <NA>
#> 4 2             2 ALT > 2 times ULN Y       ALT     ALT (U/L)          <NA>
#> 5 2             3 ALT > 2 times ULN N       ALT     ALT (U/L)          <NA>
#> 6 2             4 ALT > 2 times ULN Y       ALT     ALT (U/L)          <NA>
#> 7 1             2 <NA>              <NA>    ALT2    ALT > 2 times ULN N
#> 8 2             2 <NA>              <NA>    ALT2    ALT > 2 times ULN Y
```

A more complex example: Confirmed Best Overall Response (`first/last_cond_upper`, `join_type`, `source_datasets`):

The final example showcases a use of `derive_extreme_event()` to calculate the Confirmed Best Overall Response (CBOR) in an `ADRS` dataset, as is common in many oncology trials. This example builds on all the previous ones and thus assumes a baseline level of confidence with `derive_extreme_event()`.

The following `ADSL` and `ADRS` datasets will be used throughout:

```
adsl <- tribble(
  ~USUBJID, ~TRTSDTC,
  "1",      "2020-01-01",
  "2",      "2019-12-12",
  "3",      "2019-11-11",
  "4",      "2019-12-30",
  "5",      "2020-01-01",
  "6",      "2020-02-02",
  "7",      "2020-02-02",
  "8",      "2020-02-01"
) %>%
mutate(
  TRTSDT = ymd(TRTSDTC),
  STUDYID = "AB42"
)

adrs <- tribble(
  ~USUBJID, ~ADTC,        ~AVALC,
  "1",      "2020-01-01", "PR",
  "1",      "2020-02-01", "CR",
  "1",      "2020-02-16", "NE",
  "1",      "2020-03-01", "CR",
  "1",      "2020-04-01", "SD",
  "2",      "2020-01-01", "SD",
  "2",      "2020-02-01", "PR",
  "2",      "2020-03-01", "SD",
  "2",      "2020-03-13", "CR",
  "4",      "2020-01-01", "PR",
  "4",      "2020-03-01", "NE",
  "4",      "2020-04-01", "NE",
  "4",      "2020-05-01", "PR",
  "5",      "2020-01-01", "PR",
  "5",      "2020-01-10", "PR",
```

```
  "5",      "2020-01-20", "PR",
  "6",      "2020-02-06", "PR",
  "6",      "2020-02-16", "CR",
  "6",      "2020-03-30", "PR",
  "7",      "2020-02-06", "PR",
  "7",      "2020-02-16", "CR",
  "7",      "2020-04-01", "NE",
  "8",      "2020-02-16", "PD"
) %>%
  mutate(
    ADT = ymd(ADTC),
    STUDYID = "AB42",
    PARAMCD = "OVR",
    PARAM = "Overall Response by Investigator"
  ) %>%
  derive_vars_merged(
    dataset_add = adsl,
    by_vars = exprs(STUDYID, USUBJID),
    new_vars = exprs(TRTSDT)
  )
```

Since the CBOR derivation contains multiple complex parts, it's convenient to make use of the `description` argument within each event object to describe what condition is being checked.

- For the Confirmed Response (CR), for each `"CR"` record in the original `ADRS` dataset that will be identified by the first part of the `condition` argument (`AVALC == "CR"`), we need to use the `first_cond_upper` argument to limit the group of observations to consider alongside it. Namely, we need to look up to and including the second CR (`AVALC.join == "CR"`) over 28 days from the first one (`ADT.join >= ADT + 28`). The observations satisfying `first_cond_upper` then form part of our "join group", meaning that the remaining portions of `condition` which reference joined variables are limited to this group. In particular, within `condition` we use `all()` to check that all observations are either `"CR"` or `"NE"`, and `count_vals()` to ensure at most one is `"NE"`.
  Note that the selection of `join_type = "after"` is critical here, due to the fact that the restriction implied by `join_type` is applied before the one implied by `first_cond_upper`. Picking the first subject (who was correctly identified as a confirmed responder) as an example, selecting `join_type = "all"` instead of `"after"` would mean the first `"PR"` record from `"2020-01-01"` would also be considered when evaluating the `all(AVALC.join %in% c("CR", "NE"))` portion of `condition`. In turn, the condition would not be satisfied anymore, and in this case, following the later event logic shows the subject would be considered a partial responder instead.
- The Partial Response (PR), is very similar; with the difference being that the first portion of `condition` now references `"PR"` and `first_cond_upper` accepts a confirmatory `"PR"` or `"CR"` 28 days later. Note that now we must add `"PR"` as an option within the `all()` condition to account for confirmatory `"PR"`s.
- The Stable Disease (SD), Progressive Disease (PD) and Not Evaluable (NE) events are simpler and just require `event()` calls.
- Finally, we use a catch-all `event()` with `condition = TRUE` and `dataset_name = "adsl"` to identify those subjects who do not appear in `ADRS` and list their CBOR as `"MISSING"`. Note

here the fact that dataset_name is set to "adsl", which is a new source dataset. As such it's important in the main derive_extreme_event() call to list adsl as another source dataset with source_datasets = list(adsl = adsl).

```
derive_extreme_event(
  adrs,
  by_vars = exprs(STUDYID, USUBJID),
  tmp_event_nr_var = event_nr,
  order = exprs(event_nr, ADT),
  mode = "first",
  source_datasets = list(adsl = adsl),
  events = list(
    event_joined(
      description = paste(
        "CR needs to be confirmed by a second CR at least 28 days later",
        "at most one NE is acceptable between the two assessments"
      ),
      join_vars = exprs(AVALC, ADT),
      join_type = "after",
      first_cond_upper = AVALC.join == "CR" & ADT.join >= ADT + 28,
      condition = AVALC == "CR" &
        all(AVALC.join %in% c("CR", "NE")) &
        count_vals(var = AVALC.join, val = "NE") <= 1,
      set_values_to = exprs(AVALC = "CR")
    ),
    event_joined(
      description = paste(
       "PR needs to be confirmed by a second CR or PR at least 28 days later,",
        "at most one NE is acceptable between the two assessments"
      ),
      join_vars = exprs(AVALC, ADT),
      join_type = "after",
      first_cond_upper = AVALC.join %in% c("CR", "PR") & ADT.join >= ADT + 28,
      condition = AVALC == "PR" &
        all(AVALC.join %in% c("CR", "PR", "NE")) &
        count_vals(var = AVALC.join, val = "NE") <= 1,
      set_values_to = exprs(AVALC = "PR")
    ),
    event(
      description = paste(
        "CR, PR, or SD are considered as SD if occurring at least 28",
        "after treatment start"
      ),
      condition = AVALC %in% c("CR", "PR", "SD") & ADT >= TRTSDT + 28,
      set_values_to = exprs(AVALC = "SD")
    ),
    event(
      condition = AVALC == "PD",
      set_values_to = exprs(AVALC = "PD")
```

```
    ),
    event(
      condition = AVALC %in% c("CR", "PR", "SD", "NE"),
      set_values_to = exprs(AVALC = "NE")
    ),
    event(
    description = "Set response to MISSING for patients without records in ADRS",
      dataset_name = "adsl",
      condition = TRUE,
      set_values_to = exprs(AVALC = "MISSING"),
      keep_source_vars = exprs(TRTSDT)
    )
  ),
  set_values_to = exprs(
    PARAMCD = "CBOR",
    PARAM = "Best Confirmed Overall Response by Investigator"
  )
) %>%
  filter(PARAMCD == "CBOR") %>%
  select(-STUDYID, -ADTC)
#> # A tibble: 8 × 6
#>   USUBJID AVALC   ADT        PARAMCD PARAM                              TRTSDT
#>   <chr>   <chr>   <date>     <chr>   <chr>                              <date>
#> 1 1       CR      2020-02-01 CBOR    Best Confirmed Overall Response. . . 2020-01-01
#> 2 2       SD      2020-02-01 CBOR    Best Confirmed Overall Response. . . 2019-12-12
#> 3 3       MISSING NA         CBOR    Best Confirmed Overall Response. . . 2019-11-11
#> 4 4       SD      2020-05-01 CBOR    Best Confirmed Overall Response. . . 2019-12-30
#> 5 5       NE      2020-01-01 CBOR    Best Confirmed Overall Response. . . 2020-01-01
#> 6 6       PR      2020-02-06 CBOR    Best Confirmed Overall Response. . . 2020-02-02
#> 7 7       NE      2020-02-06 CBOR    Best Confirmed Overall Response. . . 2020-02-02
#> 8 8       PD      2020-02-16 CBOR    Best Confirmed Overall Response. . . 2020-02-01
```

Further examples:

Equivalent examples for using thecheck_type argument can be found in `derive_extreme_records()`.

See Also

`event()`, `event_joined()`, `derive_vars_extreme_event()`

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_basetype_records()`, `derive_expected_records()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

derive_extreme_records *Add the First or Last Observation for Each By Group as New Records*

Description

Add the first or last observation for each by group as new observations. The new observations can be selected from the additional dataset. This function can be used for adding the maximum or minimum value as a separate visit. All variables of the selected observation are kept. This distinguishes `derive_extreme_records()` from `derive_summary_records()`, where only the by variables are populated for the new records.

Usage

```
derive_extreme_records(
  dataset = NULL,
  dataset_add,
  dataset_ref = NULL,
  by_vars = NULL,
  order = NULL,
  mode = NULL,
  filter_add = NULL,
  check_type = "warning",
  exist_flag = NULL,
  true_value = "Y",
  false_value = NA_character_,
  keep_source_vars = exprs(everything()),
  set_values_to,
  missing_values = NULL
)
```

Arguments

`dataset` Input dataset

If the argument is not specified (or set to `NULL`), a new dataset is created. Otherwise, the new records are appended to the specified dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value `NULL`

`dataset_add` Additional dataset

The additional dataset, which determines the by groups returned in the input dataset, based on the groups that exist in this dataset after being subset by `filter_add`.

The variables specified in the `by_vars` and `filter_add` parameters are expected in this dataset. If `mode` and `order` are specified, the first or last observation within each by group, defined by `by_vars`, is selected.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`dataset_ref` Reference dataset

The variables specified for `by_vars` are expected. For each observation of the specified dataset a new observation is added to the input dataset.

For records which are added from `dataset_ref` because there are no records in `dataset_add` for the by group only those variables are kept which are also in `dataset_add` (and are included in `keep_source_vars`).

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value `NULL`

`by_vars` Grouping variables

If `dataset_ref` is specified, this argument must be specified.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`order` Sort order

Within each by group the observations are ordered by the specified order.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`mode` Selection mode (first or last)

If `"first"` is specified, the first observation of each by group is added to the input dataset. If `"last"` is specified, the last observation of each by group is added to the input dataset.

Permitted values `"first"`, `"last"`

Default value `NULL`

`filter_add` Filter for additional dataset (`dataset_add`)

Only observations in `dataset_add` fulfilling the specified condition are considered.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`check_type` Check uniqueness?

If `"warning"` or `"error"` is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order.

Permitted values `"none"`, `"message"`, `"warning"`, `"error"`

Default value `"warning"`

`exist_flag` Existence flag

The specified variable is added to the output dataset.

For by groups with at least one observation in the additional dataset (`dataset_add`) `exist_flag` is set to the value specified by the `true_value` argument.

For all other by groups `exist_flag` is set to the value specified by the `false_value` argument.

Permitted values Variable name

Default value NULL

true_value True value

For new observations selected from the additional dataset (dataset_add), exist_flag is set to the specified value.

Permitted values a character scalar, i.e., a character vector of length one

Default value "Y"

false_value False value

For new observations not selected from the additional dataset (dataset_add), exist_flag is set to the specified value.

Permitted values a character scalar, i.e., a character vector of length one

Default value NA_character_

keep_source_vars

Variables to be kept in the new records

A named list or tidyselect expressions created by exprs() defining the variables to be kept for the new records. The variables specified for by_vars and set_values_to need not be specified here as they are kept automatically.

Permitted values list of variables or tidyselect expressions created by exprs(), e.g., exprs(DTHDT, starts_with("AST")) or exprs(everything)

Default value exprs(everything())

set_values_to Variables to be set

The specified variables are set to the specified values for the new observations.

Set a list of variables to some specified value for the new records

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA.

For example:

```
set_values_to = exprs(
  PARAMCD = "WOBS",
  PARAM = "Worst Observations"
)
```

Permitted values list of named expressions created by exprs(), e.g., exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))

Default value none

missing_values Values for missing records

For observations of the reference dataset (dataset_ref) which do not have a matching record in dataset_add (with respect to by_vars and after applying filter_add), the specified variables are set to the specified values for the new observations.

Permitted values list of named expressions created by exprs(), e.g., exprs(CUMDOSA = sum(AVAL, na.rm = TRUE), AVALU = "ml")

Default value NULL

Details

1. The additional dataset (dataset_add) is restricted as specified by the filter_add argument.
2. For each group (with respect to the variables specified for the by_vars argument) the first or last observation (with respect to the order specified for the order argument and the mode specified for the mode argument) is selected.
3. If dataset_ref is specified, observations which are in dataset_ref but not in the selected records are added. Variables that are common across dataset_ref, dataset_add and keep_source_vars() are also populated for the new observations. If missing_values is specified, the specified values are set for these observations.
4. The variables specified by the set_values_to argument are added to the selected observations.
5. The variables specified by the keep_source_vars argument are selected along with the variables specified in by_vars and set_values_to arguments.
6. The observations are added to input dataset (dataset). If no input dataset is provided, a new dataset is created.

Value

The input dataset with the first or last observation of each by group added as new observations.

Examples

Add last/first record as new record:

For each subject the last record should be added as a new visit.

- The source dataset for the new records is specified by the dataset_add argument. Here it is the same as the input dataset.
- The groups for which new records are added are specified by the by_vars argument. Here for each *subject* a record should be added. Thus by_vars = exprs(USUBJID) is specified.
- As there are multiple records per subject, the mode and order arguments are specified to request that the *last* record is selected when the records are sorted by visit (AVISITN). The records are sorted by each by group (by_vars) separately, i.e., it is not necessary to add the variables from by_vars to order.
- To avoid duplicates in the output dataset the set_values_to argument is specified to set the visit (AVISIT) to a special value for the *new* records.

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)

adlb <- tribble(
  ~USUBJID, ~AVISITN, ~AVAL,
  "1",      1,          113,
  "1",      2,          111,
  "2",      1,          101,
  "2",      2,           NA,
  "3",      1,           NA,
```

```
)

derive_extreme_records(
  adlb,
  dataset_add = adlb,
  by_vars = exprs(USUBJID),
  order = exprs(AVISITN),
  mode = "last",
  set_values_to = exprs(
    AVISITN = 99
  )
)
#> # A tibble: 8 × 3
#>   USUBJID AVISITN  AVAL
#>   <chr>     <dbl> <dbl>
#> 1 1             1   113
#> 2 1             2   111
#> 3 2             1   101
#> 4 2             2    NA
#> 5 3             1    NA
#> 6 1            99   111
#> 7 2            99    NA
#> 8 3            99    NA
```

Restricting source records (`filter_add`)**:**

The source records can be restricted by the `filter_add` argument, e.g., to exclude visits with missing analysis value from selecting for the last visit record:

```
derive_extreme_records(
  adlb,
  dataset_add = adlb,
  filter_add = !is.na(AVAL),
  by_vars = exprs(USUBJID),
  order = exprs(AVISITN),
  mode = "last",
  set_values_to = exprs(
    AVISITN = 99
  )
)
#> # A tibble: 7 × 3
#>   USUBJID AVISITN  AVAL
#>   <chr>     <dbl> <dbl>
#> 1 1             1   113
#> 2 1             2   111
#> 3 2             1   101
#> 4 2             2    NA
#> 5 3             1    NA
#> 6 1            99   111
#> 7 2            99   101
```

Please note that new records are added only for subjects in the *restricted* source data. Therefore no new record is added for subject `"3"`.

Adding records for groups not in source (`dataset_ref`):

Adding records for groups which are not in the source data can be achieved by specifying a reference dataset by the `dataset_ref` argument. For example, specifying the input dataset for `dataset_ref` below ensures that new records are added also for subject without a valid analysis value:

```
derive_extreme_records(
  adlb,
  dataset_add = adlb,
  filter_add = !is.na(AVAL),
  dataset_ref = adlb,
  by_vars = exprs(USUBJID),
  order = exprs(AVISITN),
  mode = "last",
  set_values_to = exprs(
    AVISITN = 99
  )
)
#> # A tibble: 8 × 3
#>   USUBJID AVISITN  AVAL
#>   <chr>     <dbl> <dbl>
#> 1 1             1   113
#> 2 1             2   111
#> 3 2             1   101
#> 4 2             2    NA
#> 5 3             1    NA
#> 6 1            99   111
#> 7 2            99   101
#> 8 3            99    NA
```

Setting values for missing groups (`missing_values`):

When using `dataset_ref`, groups without matching records in `dataset_add` get new records with `NA` values by default. The `missing_values` argument allows setting specific values for these records. In the example records for the last non-missing value are added. For subjects without assessments the value is set to the mean analysis value across all records.

```
mean_value <- mean(adlb$AVAL, na.rm = TRUE)

derive_extreme_records(
  adlb,
  dataset_add = adlb,
  filter_add = !is.na(AVAL),
  dataset_ref = adlb,
  by_vars = exprs(USUBJID),
  order = exprs(AVISITN),
  mode = "last",
```

```
  set_values_to = exprs(
    AVISITN = 99
  ),
  missing_values = exprs(
    AVAL = !!mean_value,
    DTYPE = "MOV"
  )
)
#> # A tibble: 8 × 4
#>   USUBJID AVISITN  AVAL DTYPE
#>   <chr>     <dbl> <dbl> <chr>
#> 1 1             1  113  <NA>
#> 2 1             2  111  <NA>
#> 3 2             1  101  <NA>
#> 4 2             2   NA  <NA>
#> 5 3             1   NA  <NA>
#> 6 1            99  111  <NA>
#> 7 2            99  101  <NA>
#> 8 3            99  108. MOV
```

Selecting variables for new records (`keep_source_vars`)**:**

Which variables from the source dataset are kept for the new records can be specified by the `keep_source_vars` argument. Variables specified by `by_vars` or `set_values_to` don't need to be added to `keep_source_vars` as these are always kept.

```
adlb <- tribble(
  ~USUBJID, ~AVISIT,  ~AVAL, ~LBSEQ,
  "1",      "WEEK 1",   123,      1,
  "1",      "WEEK 2",   101,      2,
  "2",      "WEEK 1",    99,      1,
  "2",      "WEEK 2",   110,      2,
  "2",      "WEEK 3",    93,      3
)

derive_extreme_records(
  dataset_add = adlb,
  filter_add = !is.na(AVAL),
  by_vars = exprs(USUBJID),
  order = exprs(AVAL),
  mode = "first",
  keep_source_vars = exprs(AVAL),
  set_values_to = exprs(
    AVISIT = "MINIMUM"
  )
)
#> # A tibble: 2 × 3
#>   USUBJID AVISIT   AVAL
#>   <chr>   <chr>   <dbl>
#> 1 1       MINIMUM   101
```

```
#> 2 2       MINIMUM    93
```

Handling duplicates (`check_type`)**:**
The source records are checked regarding duplicates with respect to `by_vars` and `order`. By default, a warning is issued if any duplicates are found.

```
adlb <- tribble(
  ~USUBJID, ~AVISIT,  ~AVAL,
  "1",      "WEEK 1",   123,
  "1",      "WEEK 2",   123,
  "2",      "WEEK 1",    99,
  "2",      "WEEK 2",   110,
  "2",      "WEEK 3",    93,
)

derive_extreme_records(
  dataset_add = adlb,
  filter_add = !is.na(AVAL),
  by_vars = exprs(USUBJID),
  order = exprs(AVAL),
  mode = "first",
  set_values_to = exprs(
    AVISIT = "MINIMUM"
  )
)
#> # A tibble: 2 × 3
#>   USUBJID AVISIT   AVAL
#>   <chr>   <chr>   <dbl>
#> 1 1       MINIMUM   123
#> 2 2       MINIMUM    93
#> Warning: Dataset contains duplicate records with respect to `USUBJID` and `AVAL`
#> i Run `admiral::get_duplicates_dataset()` to access the duplicate records
```

For investigating the issue, the dataset of the duplicate source records can be obtained by calling `get_duplicates_dataset()`:

```
get_duplicates_dataset()
#> Duplicate records with respect to `USUBJID` and `AVAL`.
#> # A tibble: 2 × 3
#>   USUBJID  AVAL AVISIT
#> * <chr>   <dbl> <chr>
#> 1 1         123 WEEK 1
#> 2 1         123 WEEK 2
```

Common options to solve the issue are:

- Restricting the source records by specifying/updating the `filter_add` argument.
- Specifying additional variables for `order`.
- Setting `check_type = "none"` to ignore any duplicates.

In this example it doesn't matter which of the records with the minimum value is chosen because it doesn't affect the output dataset. Thus the third option is used:

```
derive_extreme_records(
  dataset_add = adlb,
  filter_add = !is.na(AVAL),
  by_vars = exprs(USUBJID),
  order = exprs(AVAL),
  mode = "first",
  check_type = "none",
  set_values_to = exprs(
    AVISIT = "MINIMUM"
  )
)
#> # A tibble: 2 × 3
#>   USUBJID AVISIT   AVAL
#>   <chr>   <chr>   <dbl>
#> 1 1       MINIMUM   123
#> 2 2       MINIMUM    93
```

Flagging existence of source records (`exist_flag`, `true_value`, `false_value`):

If the existence of a source record should be flagged, the `exist_flag` argument can be specified. The specified variable is set to `true_value` if a source record exists. Otherwise, it is set to `false_value`.

The `dataset_ref` argument should be specified as otherwise *all* new records originate from `dataset_add`, i.e., `exist_flag` would be set to `true_value` for all records.

```
adsl <- tribble(
  ~USUBJID, ~DTHDT,
  "1",      ymd("2022-05-13"),
  "2",      ymd(""),
  "3",      ymd("")
)

derive_extreme_records(
  dataset_ref = adsl,
  dataset_add = adsl,
  by_vars = exprs(USUBJID),
  filter_add = !is.na(DTHDT),
  exist_flag = AVALC,
  true_value = "Y",
  false_value = "N",
  set_values_to = exprs(
    PARAMCD = "DEATH",
    ADT = DTHDT
  )
)
#> # A tibble: 3 × 5
#>   USUBJID PARAMCD ADT        DTHDT      AVALC
#>   <chr>   <chr>   <date>     <date>     <chr>
#> 1 1       DEATH   2022-05-13 2022-05-13 Y
#> 2 2       DEATH   NA         NA         N
```

```
#> 3 3       DEATH   NA          NA          N
```

Derive `DTYPE = "LOV"`**:**

For each subject and parameter the last valid assessment (with respect to `AVISITN` and `LBSEQ`) should be selected and added as a new record to the input dataset. For the new records set `AVISIT = "PBL LAST"`, `AVISITN = 99`, and `DTYPE = "LOV"`.

```
adlb <- tribble(
  ~USUBJID, ~AVISIT,    ~AVISITN, ~PARAMCD, ~AVAL, ~LBSEQ,
  "1",      "BASELINE",        1, "ABC",      120,      1,
  "1",      "WEEK 1",          2, "ABC",      113,      2,
  "1",      "WEEK 1",          2, "ABC",      117,      3,
  "2",      "BASELINE",        1, "ABC",      101,      1,
  "2",      "WEEK 1",          2, "ABC",      101,      2,
  "2",      "WEEK 2",          3, "ABC",       95,      3,
  "1",      "BASELINE",        1, "DEF",       17,      1,
  "1",      "WEEK 1",          2, "DEF",       NA,      2,
  "1",      "WEEK 1",          2, "DEF",       13,      3,
  "2",      "BASELINE",        1, "DEF",        9,      1,
  "2",      "WEEK 1",          2, "DEF",       10,      2,
  "2",      "WEEK 2",          3, "DEF",       12,      3
) %>%
mutate(STUDYID = "XYZ", .before = USUBJID)

derive_extreme_records(
  adlb,
  dataset_add = adlb,
  filter_add = !is.na(AVAL) & AVISIT != "BASELINE",
  by_vars = exprs(!!!get_admiral_option("subject_keys"), PARAMCD),
  order = exprs(AVISITN, LBSEQ),
  mode = "last",
  set_values_to = exprs(
    AVISIT = "PBL LAST",
    AVISITN = 99,
    DTYPE = "LOV"
  )
)
#> # A tibble: 16 × 8
#>    STUDYID USUBJID AVISIT   AVISITN PARAMCD  AVAL LBSEQ DTYPE
#>    <chr>   <chr>   <chr>      <dbl> <chr>   <dbl> <dbl> <chr>
#>  1 XYZ     1       BASELINE       1 ABC       120     1 <NA>
#>  2 XYZ     1       WEEK 1         2 ABC       113     2 <NA>
#>  3 XYZ     1       WEEK 1         2 ABC       117     3 <NA>
#>  4 XYZ     2       BASELINE       1 ABC       101     1 <NA>
#>  5 XYZ     2       WEEK 1         2 ABC       101     2 <NA>
#>  6 XYZ     2       WEEK 2         3 ABC        95     3 <NA>
#>  7 XYZ     1       BASELINE       1 DEF        17     1 <NA>
#>  8 XYZ     1       WEEK 1         2 DEF        NA     2 <NA>
#>  9 XYZ     1       WEEK 1         2 DEF        13     3 <NA>
```

```
#> 10 XYZ     2       BASELINE       1 DEF         9     1 <NA>
#> 11 XYZ     2       WEEK 1         2 DEF        10     2 <NA>
#> 12 XYZ     2       WEEK 2         3 DEF        12     3 <NA>
#> 13 XYZ     1       PBL LAST      99 ABC       117     3 LOV
#> 14 XYZ     1       PBL LAST      99 DEF        13     3 LOV
#> 15 XYZ     2       PBL LAST      99 ABC        95     3 LOV
#> 16 XYZ     2       PBL LAST      99 DEF        12     3 LOV
```

Derive `DTYPE = "MINIMUM"`**:**

For each subject and parameter the record with the minimum analysis value should be selected and added as a new record to the input dataset. If there are multiple records meeting the minimum, the first record with respect to `AVISIT` and `LBSEQ` should be selected. For the new records set `AVISIT = "PBL MIN"`, `AVISITN = 97`, and `DTYPE = "MINIMUM"`.

```
derive_extreme_records(
  adlb,
  dataset_add = adlb,
  filter_add = !is.na(AVAL) & AVISIT != "BASELINE",
  by_vars = exprs(!!!get_admiral_option("subject_keys"), PARAMCD),
  order = exprs(AVAL, AVISITN, LBSEQ),
  mode = "first",
  set_values_to = exprs(
    AVISIT = "PBL MIN",
    AVISITN = 97,
    DTYPE = "MINIMUM"
  )
)
#> # A tibble: 16 × 8
#>    STUDYID USUBJID AVISIT   AVISITN PARAMCD  AVAL LBSEQ DTYPE
#>    <chr>   <chr>   <chr>      <dbl> <chr>   <dbl> <dbl> <chr>
#>  1 XYZ     1       BASELINE       1 ABC       120     1 <NA>
#>  2 XYZ     1       WEEK 1         2 ABC       113     2 <NA>
#>  3 XYZ     1       WEEK 1         2 ABC       117     3 <NA>
#>  4 XYZ     2       BASELINE       1 ABC       101     1 <NA>
#>  5 XYZ     2       WEEK 1         2 ABC       101     2 <NA>
#>  6 XYZ     2       WEEK 2         3 ABC        95     3 <NA>
#>  7 XYZ     1       BASELINE       1 DEF        17     1 <NA>
#>  8 XYZ     1       WEEK 1         2 DEF        NA     2 <NA>
#>  9 XYZ     1       WEEK 1         2 DEF        13     3 <NA>
#> 10 XYZ     2       BASELINE       1 DEF         9     1 <NA>
#> 11 XYZ     2       WEEK 1         2 DEF        10     2 <NA>
#> 12 XYZ     2       WEEK 2         3 DEF        12     3 <NA>
#> 13 XYZ     1       PBL MIN       97 ABC       113     2 MINIMUM
#> 14 XYZ     1       PBL MIN       97 DEF        13     3 MINIMUM
#> 15 XYZ     2       PBL MIN       97 ABC        95     3 MINIMUM
#> 16 XYZ     2       PBL MIN       97 DEF        10     2 MINIMUM
```

Derive `DTYPE = "MAXIMUM"`**:**

For each subject and parameter the record with the maximum analysis value should be selected and added as a new record to the input dataset. If there are multiple records meeting the maximum,

the first record with respect to AVISIT and LBSEQ should be selected. For the new records set AVISIT = "PBL MAX", AVISITN = 98, and DTYPE = "MAXIMUM".

```
derive_extreme_records(
  adlb,
  dataset_add = adlb,
  filter_add = !is.na(AVAL) & AVISIT != "BASELINE",
  by_vars = exprs(!!!get_admiral_option("subject_keys"), PARAMCD),
  order = exprs(desc(AVAL), AVISITN, LBSEQ),
  mode = "first",
  set_values_to = exprs(
    AVISIT = "PBL MAX",
    AVISITN = 99,
    DTYPE = "MAXIMUM"
  )
)
#> # A tibble: 16 × 8
#>    STUDYID USUBJID AVISIT   AVISITN PARAMCD  AVAL LBSEQ DTYPE
#>    <chr>   <chr>   <chr>      <dbl> <chr>   <dbl> <dbl> <chr>
#>  1 XYZ     1       BASELINE       1 ABC       120     1 <NA>
#>  2 XYZ     1       WEEK 1         2 ABC       113     2 <NA>
#>  3 XYZ     1       WEEK 1         2 ABC       117     3 <NA>
#>  4 XYZ     2       BASELINE       1 ABC       101     1 <NA>
#>  5 XYZ     2       WEEK 1         2 ABC       101     2 <NA>
#>  6 XYZ     2       WEEK 2         3 ABC        95     3 <NA>
#>  7 XYZ     1       BASELINE       1 DEF        17     1 <NA>
#>  8 XYZ     1       WEEK 1         2 DEF        NA     2 <NA>
#>  9 XYZ     1       WEEK 1         2 DEF        13     3 <NA>
#> 10 XYZ     2       BASELINE       1 DEF         9     1 <NA>
#> 11 XYZ     2       WEEK 1         2 DEF        10     2 <NA>
#> 12 XYZ     2       WEEK 2         3 DEF        12     3 <NA>
#> 13 XYZ     1       PBL MAX       99 ABC       117     3 MAXIMUM
#> 14 XYZ     1       PBL MAX       99 DEF        13     3 MAXIMUM
#> 15 XYZ     2       PBL MAX       99 ABC       101     2 MAXIMUM
#> 16 XYZ     2       PBL MAX       99 DEF        12     3 MAXIMUM
```

Derive DTYPE = "WOC" **or** DTYPE = "BOC"**:**

For each subject and parameter the record with the worst analysis value should be selected and added as a new record to the input dataset. The worst value is either the minimum or maximum value depending on the parameter. If there are multiple records meeting the worst value, the first record with respect to AVISIT and LBSEQ should be selected. For the new records set AVISIT = "PBL WORST", AVISITN = 96, and DTYPE = "WOC".

Here the maximum is considered worst for PARAMCD = "ABC" and the minimum for PARAMCD = "DEF".

```
derive_extreme_records(
  adlb,
  dataset_add = adlb,
  filter_add = !is.na(AVAL) & AVISIT != "BASELINE",
```

```
  by_vars = exprs(!!!get_admiral_option("subject_keys"), PARAMCD),
  order = exprs(
    if_else(PARAMCD == "ABC", desc(AVAL), AVAL),
    AVISITN, LBSEQ
  ),
  mode = "first",
  set_values_to = exprs(
    AVISIT = "PBL WORST",
    AVISITN = 96,
    DTYPE = "WOC"
  )
)
#> # A tibble: 16 × 8
#>    STUDYID USUBJID AVISIT    AVISITN PARAMCD  AVAL LBSEQ DTYPE
#>    <chr>   <chr>   <chr>       <dbl> <chr>   <dbl> <dbl> <chr>
#>  1 XYZ     1       BASELINE        1 ABC       120     1 <NA>
#>  2 XYZ     1       WEEK 1          2 ABC       113     2 <NA>
#>  3 XYZ     1       WEEK 1          2 ABC       117     3 <NA>
#>  4 XYZ     2       BASELINE        1 ABC       101     1 <NA>
#>  5 XYZ     2       WEEK 1          2 ABC       101     2 <NA>
#>  6 XYZ     2       WEEK 2          3 ABC        95     3 <NA>
#>  7 XYZ     1       BASELINE        1 DEF        17     1 <NA>
#>  8 XYZ     1       WEEK 1          2 DEF        NA     2 <NA>
#>  9 XYZ     1       WEEK 1          2 DEF        13     3 <NA>
#> 10 XYZ     2       BASELINE        1 DEF         9     1 <NA>
#> 11 XYZ     2       WEEK 1          2 DEF        10     2 <NA>
#> 12 XYZ     2       WEEK 2          3 DEF        12     3 <NA>
#> 13 XYZ     1       PBL WORST      96 ABC       117     3 WOC
#> 14 XYZ     1       PBL WORST      96 DEF        13     3 WOC
#> 15 XYZ     2       PBL WORST      96 ABC       101     2 WOC
#> 16 XYZ     2       PBL WORST      96 DEF        10     2 WOC
```

Derive a parameter for the first disease progression (PD):

For each subject in the ADSL dataset a new parameter should be added to the input dataset which indicates whether disease progression (PD) occurred (set AVALC = "Y", AVAL = 1) or not (set AVALC = "N", AVAL = 0). For the new parameter set PARAMCD = "PD" and PARAM = "Disease Progression".

```
adsl <- tribble(
  ~USUBJID, ~DTHDT,
  "1",      ymd("2022-05-13"),
  "2",      ymd(""),
  "3",      ymd("")
) %>%
  mutate(STUDYID = "XX1234")

adrs <- tribble(
  ~USUBJID, ~RSDTC,       ~AVALC, ~AVAL,
  "1",      "2020-01-02", "PR",       2,
  "1",      "2020-02-01", "CR",       1,
```

```
  "1",      "2020-03-01", "CR",       1,
  "2",      "2021-06-15", "SD",       3,
  "2",      "2021-07-16", "PD",       4,
  "2",      "2021-09-14", "PD",       4
) %>%
  mutate(
    STUDYID = "XX1234", .before = USUBJID
  ) %>%
  mutate(
    ADT = ymd(RSDTC),
    PARAMCD = "OVR",
    PARAM = "Overall Response",
    .after = RSDTC
  )

derive_extreme_records(
  adrs,
  dataset_ref = adsl,
  dataset_add = adrs,
  by_vars = get_admiral_option("subject_keys"),
  filter_add = PARAMCD == "OVR" & AVALC == "PD",
  order = exprs(ADT),
  exist_flag = AVALC,
  true_value = "Y",
  false_value = "N",
  mode = "first",
  set_values_to = exprs(
    PARAMCD = "PD",
    PARAM = "Disease Progression",
    AVAL = yn_to_numeric(AVALC),
  )
)
#> # A tibble: 9 × 8
#>   STUDYID USUBJID RSDTC      ADT        PARAMCD PARAM              AVALC  AVAL
#>   <chr>   <chr>   <chr>      <date>     <chr>   <chr>              <chr> <dbl>
#> 1 XX1234  1       2020-01-02 2020-01-02 OVR     Overall Response   PR        2
#> 2 XX1234  1       2020-02-01 2020-02-01 OVR     Overall Response   CR        1
#> 3 XX1234  1       2020-03-01 2020-03-01 OVR     Overall Response   CR        1
#> 4 XX1234  2       2021-06-15 2021-06-15 OVR     Overall Response   SD        3
#> 5 XX1234  2       2021-07-16 2021-07-16 OVR     Overall Response   PD        4
#> 6 XX1234  2       2021-09-14 2021-09-14 OVR     Overall Response   PD        4
#> 7 XX1234  2       2021-07-16 2021-07-16 PD      Disease Progression Y        1
#> 8 XX1234  1       <NA>       NA         PD      Disease Progression N        0
#> 9 XX1234  3       <NA>       NA         PD      Disease Progression N        0
```

Derive parameter indicating death:

For each subject in the ADSL dataset a new parameter should be created which indicates whether the subject died (set AVALC = "Y", AVAL = 1) or not (set AVALC = "N", AVAL = 0). For the new parameter set PARAMCD = "DEATH", PARAM = "Death", and ADT to the date of death (DTHDT).

```
derive_extreme_records(
  dataset_ref = adsl,
  dataset_add = adsl,
  by_vars = exprs(STUDYID, USUBJID),
  filter_add = !is.na(DTHDT),
  exist_flag = AVALC,
  true_value = "Y",
  false_value = "N",
  mode = "first",
  keep_source_vars = exprs(AVALC),
  set_values_to = exprs(
    PARAMCD = "DEATH",
    PARAM = "Death",
    ADT = DTHDT
  )
)
#> # A tibble: 3 × 6
#>   STUDYID USUBJID PARAMCD PARAM ADT        AVALC
#>   <chr>   <chr>   <chr>   <chr> <date>     <chr>
#> 1 XX1234  1       DEATH   Death 2022-05-13 Y
#> 2 XX1234  2       DEATH   Death NA         N
#> 3 XX1234  3       DEATH   Death NA         N
```

The keep_source_vars argument is specified to avoid that all ADSL variables (like DTHDT) are copied to the parameter.

See Also

derive_summary_records()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

---

derive_locf_records *Derive LOCF (Last Observation Carried Forward) Records*

Description

Adds LOCF records as new observations for each 'by group' when the dataset does not contain observations for missed visits/time points and when analysis value is missing.

Usage

```
derive_locf_records(
  dataset,
```

```
  dataset_ref,
  by_vars,
  id_vars_ref = NULL,
  analysis_var = AVAL,
  imputation = "add",
  order,
  keep_vars = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars, analysis_var, order, and keep_vars arguments are expected to be in the dataset. **Default value** none
dataset_ref	Expected observations dataset Data frame with all the combinations of PARAMCD, PARAM, AVISIT, AVISITN, ... which are expected in the dataset is expected. **Default value** none
by_vars	Grouping variables For each group defined by by_vars those observations from dataset_ref are added to the output dataset which do not have a corresponding observation in the input dataset or for which analysis_var is NA for the corresponding observation in the input dataset. **Default value** none
id_vars_ref	Grouping variables in expected observations dataset The variables to group by in dataset_ref when determining which observations should be added to the input dataset. **Default value** All the variables in dataset_ref
analysis_var	Analysis variable. **Permitted values** a variable **Default value** AVAL
imputation	Select the mode of imputation: add: Keep all original records and add imputed records for missing timepoints and missing analysis_var values from dataset_ref. update: Update records with missing analysis_var and add imputed records for missing timepoints from dataset_ref. update_add: Keep all original records, update records with missing analysis_var and add imputed records for missing timepoints from dataset_ref. **Permitted values** One of these 3 values: "add", "update", "update_add" **Default value** "add"

`order` Sort order

The dataset is sorted by `order` before carrying the last observation forward (e.g. `AVAL`) within each `by_vars`.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

Default value none

`keep_vars` Variables that need carrying the last observation forward

Keep variables that need carrying the last observation forward other than `analysis_var` (e.g., `PARAMN`, `VISITNUM`). If by default `NULL`, only variables specified in `by_vars` and `analysis_var` will be populated in the newly created records.

Default value `NULL`

Details

For each group (with respect to the variables specified for the by_vars parameter) those observations from `dataset_ref` are added to the output dataset

- which do not have a corresponding observation in the input dataset or
- for which `analysis_var` is `NA` for the corresponding observation in the input dataset.

  For the new observations, `analysis_var` is set to the non-missing `analysis_var` of the previous observation in the input dataset (when sorted by `order`) and `DTYPE` is set to "LOCF".

  The `imputation` argument decides whether to update the existing observation when `analysis_var` is `NA` (`"update"` and `"update_add"`), or to add a new observation from `dataset_ref` instead (`"add"`).

Value

The input dataset with the new "LOCF" observations added for each `by_vars`, based on the value passed to the `imputation` argument.

Examples

Add records for missing analysis variable using reference dataset:

Imputed records should be added for missing timepoints and for missing `analysis_var` (from `dataset_ref`), while retaining all original records.

- The reference dataset for the imputed records is specified by the `dataset_add` argument. It should contain all expected combinations of variables. In this case, `advs_expected_obsv` is created by `crossing()` datasets `paramcd` and `avisit`, which includes all combinations of PARAMCD, AVISITN, and AVISIT.
- The groups for which new records are added are specified by the `by_vars` argument. Here, one record should be added for each *subject* and *parameter*. Therefore, `by_vars = exprs(STUDYID, USUBJID, PARAMCD)` is specified.
- The imputation method is specified using the `imputation` argument. In this case, records with missing analysis values *add* records from `dataset_ref` after the data are sorted by the variables in `by_vars` and by visit (`AVISITN` and `AVISIT`), as specified in the `order` argument.
- Variables other than `analysis_var` and `by_vars` that require LOCF (Last-Observation- Carried-Forward handling (in this case, `PARAMN`) are specified in the `keep_vars` argument.

```
library(dplyr)
library(tibble)
library(tidyr)

advs <- tribble(
  ~STUDYID,  ~USUBJID,     ~VSSEQ, ~PARAMCD, ~PARAMN, ~AVAL, ~AVISITN, ~AVISIT,
  "CDISC01", "01-701-1015",     1, "PULSE",       1,    65,       0, "BASELINE",
  "CDISC01", "01-701-1015",     2, "DIABP",       2,    79,       0, "BASELINE",
  "CDISC01", "01-701-1015",     3, "DIABP",       2,    80,       2, "WEEK 2",
  "CDISC01", "01-701-1015",     4, "DIABP",       2,    NA,       4, "WEEK 4",
  "CDISC01", "01-701-1015",     5, "DIABP",       2,    NA,       6, "WEEK 6",
  "CDISC01", "01-701-1015",     6, "SYSBP",       3,   130,       0, "BASELINE",
  "CDISC01", "01-701-1015",     7, "SYSBP",       3,   132,       2, "WEEK 2"
)

paramcd <- tribble(
~PARAMCD,
"PULSE",
"DIABP",
"SYSBP"
)

avisit <- tribble(
~AVISITN, ~AVISIT,
0, "BASELINE",
2, "WEEK 2",
4, "WEEK 4",
6, "WEEK 6"
)

advs_expected_obsv <- paramcd %>%
crossing(avisit)

derive_locf_records(
  dataset = advs,
  dataset_ref = advs_expected_obsv,
  by_vars = exprs(STUDYID, USUBJID, PARAMCD),
  imputation = "add",
  order = exprs(AVISITN, AVISIT),
  keep_vars = exprs(PARAMN)
) |>
  arrange(USUBJID, PARAMCD, AVISIT)
#> # A tibble: 14 × 9
#>    STUDYID USUBJID     VSSEQ PARAMCD PARAMN  AVAL AVISITN AVISIT   DTYPE
#>    <chr>   <chr>       <dbl> <chr>    <dbl> <dbl>   <dbl> <chr>    <chr>
#>  1 CDISC01 01-701-1015     2 DIABP        2    79       0 BASELINE <NA>
#>  2 CDISC01 01-701-1015     3 DIABP        2    80       2 WEEK 2   <NA>
#>  3 CDISC01 01-701-1015    NA DIABP        2    80       4 WEEK 4   LOCF
```

```
#>  4 CDISC01 01-701-1015     4 DIABP        2    NA       4 WEEK 4   <NA>
#>  5 CDISC01 01-701-1015    NA DIABP        2    80       6 WEEK 6   LOCF
#>  6 CDISC01 01-701-1015     5 DIABP        2    NA       6 WEEK 6   <NA>
#>  7 CDISC01 01-701-1015     1 PULSE        1    65       0 BASELINE <NA>
#>  8 CDISC01 01-701-1015    NA PULSE        1    65       2 WEEK 2   LOCF
#>  9 CDISC01 01-701-1015    NA PULSE        1    65       4 WEEK 4   LOCF
#> 10 CDISC01 01-701-1015    NA PULSE        1    65       6 WEEK 6   LOCF
#> 11 CDISC01 01-701-1015     6 SYSBP        3   130       0 BASELINE <NA>
#> 12 CDISC01 01-701-1015     7 SYSBP        3   132       2 WEEK 2   <NA>
#> 13 CDISC01 01-701-1015    NA SYSBP        3   132       4 WEEK 4   LOCF
#> 14 CDISC01 01-701-1015    NA SYSBP        3   132       6 WEEK 6   LOCF
```

Update records for missing analysis variable:

When the `imputation` mode is set to *update*, missing `analysis_var` values are updated using values from the last record after the dataset is sorted by `by_vars` and `order`. Imputed records are added for missing timepoints (from `dataset_ref`).

```
derive_locf_records(
  dataset = advs,
  dataset_ref = advs_expected_obsv,
  by_vars = exprs(STUDYID, USUBJID, PARAMCD),
  imputation = "update",
  order = exprs(AVISITN, AVISIT),
) |>
  arrange(USUBJID, PARAMCD, AVISIT)
#> # A tibble: 12 × 9
#>    STUDYID USUBJID     VSSEQ PARAMCD PARAMN  AVAL AVISITN AVISIT   DTYPE
#>    <chr>   <chr>       <dbl> <chr>    <dbl> <dbl>   <dbl> <chr>    <chr>
#>  1 CDISC01 01-701-1015     2 DIABP        2    79       0 BASELINE <NA>
#>  2 CDISC01 01-701-1015     3 DIABP        2    80       2 WEEK 2   <NA>
#>  3 CDISC01 01-701-1015     4 DIABP        2    80       4 WEEK 4   LOCF
#>  4 CDISC01 01-701-1015     5 DIABP        2    80       6 WEEK 6   LOCF
#>  5 CDISC01 01-701-1015     1 PULSE        1    65       0 BASELINE <NA>
#>  6 CDISC01 01-701-1015    NA PULSE       NA    65       2 WEEK 2   LOCF
#>  7 CDISC01 01-701-1015    NA PULSE       NA    65       4 WEEK 4   LOCF
#>  8 CDISC01 01-701-1015    NA PULSE       NA    65       6 WEEK 6   LOCF
#>  9 CDISC01 01-701-1015     6 SYSBP        3   130       0 BASELINE <NA>
#> 10 CDISC01 01-701-1015     7 SYSBP        3   132       2 WEEK 2   <NA>
#> 11 CDISC01 01-701-1015    NA SYSBP       NA   132       4 WEEK 4   LOCF
#> 12 CDISC01 01-701-1015    NA SYSBP       NA   132       6 WEEK 6   LOCF
```

Update records for missing analysis variable while keeping the original records:

When the `imputation` mode is set to *update_add*, the missing `analysis_var` values are updated using values from the last record after the dataset is sorted by `by_vars` and `order`. The updated values are added as new records, while the original records with missing `analysis_var` are retained. Imputed records are added for missing timepoints (from `dataset_ref`).

```
derive_locf_records(
  dataset = advs,
```

```
  dataset_ref = advs_expected_obsv,
  by_vars = exprs(STUDYID, USUBJID, PARAMCD),
  imputation = "update_add",
  order = exprs(AVISITN, AVISIT),
) |>
  arrange(USUBJID, PARAMCD, AVISIT)
#> # A tibble: 14 × 9
#>    STUDYID USUBJID     VSSEQ PARAMCD PARAMN  AVAL AVISITN AVISIT   DTYPE
#>    <chr>   <chr>       <dbl> <chr>    <dbl> <dbl>   <dbl> <chr>    <chr>
#>  1 CDISC01 01-701-1015     2 DIABP        2    79       0 BASELINE <NA>
#>  2 CDISC01 01-701-1015     3 DIABP        2    80       2 WEEK 2   <NA>
#>  3 CDISC01 01-701-1015     4 DIABP        2    80       4 WEEK 4   LOCF
#>  4 CDISC01 01-701-1015     4 DIABP        2    NA       4 WEEK 4   <NA>
#>  5 CDISC01 01-701-1015     5 DIABP        2    80       6 WEEK 6   LOCF
#>  6 CDISC01 01-701-1015     5 DIABP        2    NA       6 WEEK 6   <NA>
#>  7 CDISC01 01-701-1015     1 PULSE        1    65       0 BASELINE <NA>
#>  8 CDISC01 01-701-1015    NA PULSE       NA    65       2 WEEK 2   LOCF
#>  9 CDISC01 01-701-1015    NA PULSE       NA    65       4 WEEK 4   LOCF
#> 10 CDISC01 01-701-1015    NA PULSE       NA    65       6 WEEK 6   LOCF
#> 11 CDISC01 01-701-1015     6 SYSBP        3   130       0 BASELINE <NA>
#> 12 CDISC01 01-701-1015     7 SYSBP        3   132       2 WEEK 2   <NA>
#> 13 CDISC01 01-701-1015    NA SYSBP       NA   132       4 WEEK 4   LOCF
#> 14 CDISC01 01-701-1015    NA SYSBP       NA   132       6 WEEK 6   LOCF
```

Author(s)

G Gayatri

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

---

derive_param_bmi *Adds a Parameter for BMI*

Description

Adds a record for BMI/Body Mass Index using Weight and Height each by group (e.g., subject and visit) where the source parameters are available.

Note: This is a wrapper function for the more generic derive_param_computed().

Usage

```
derive_param_bmi(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "BMI"),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  get_unit_expr,
  filter = NULL,
  constant_by_vars = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. PARAMCD, and AVAL are expected as well. The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by weight_code and height_code. **Default value** none
by_vars	Grouping variables For each group defined by by_vars an observation is added to the output dataset. Only variables specified in by_vars will be populated in the newly created records. **Permitted values** list of variables created by exprs(), e.g., exprs(USUBJID, VISIT) **Default value** none
set_values_to	Variables to be set The specified variables are set to the specified values for the new observations. For example exprs(PARAMCD = "MAP") defines the parameter code for the new parameter. **Permitted values** List of variable-value pairs **Default value** exprs(PARAMCD = "MAP")
weight_code	WEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the WEIGHT. It is expected that WEIGHT is measured in kg **Permitted values** character value **Default value** "WEIGHT"
height_code	HEIGHT parameter code The observations where PARAMCD equals the specified value are considered as the HEIGHT. It is expected that HEIGHT is measured in cm **Permitted values** logical scalar

Default value "HEIGHT"

get_unit_expr An expression providing the unit of the parameter

The result is used to check the units of the input parameters.

Permitted values An expression which is evaluable in the input dataset and results in a character value

Default value none

filter Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

constant_by_vars

By variables for when HEIGHT is constant

When HEIGHT is constant, the HEIGHT parameters (measured only once) are merged to the other parameters using the specified variables.

If height is constant (e.g. only measured once at screening or baseline) then use constant_by_vars to select the subject-level variable to merge on (e.g. USUBJID). This will produce BMI at all visits where weight is measured. Otherwise it will only be calculated at visits with both height and weight collected.

Default value NULL

Details

The analysis value of the new parameter is derived as

$$BMI = \frac{WEIGHT}{HEIGHT^2}$$

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in by_vars.

See Also

compute_bmi()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```
# Example 1: Derive BMI where height is measured only once using constant_by_vars
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVISIT,
```

```
  "01-701-1015", "HEIGHT", "Height (cm)", 147, "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.0, "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)", 54.4, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 53.1, "WEEK 2",
  "01-701-1028", "HEIGHT", "Height (cm)", 163, "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)", 78.5, "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.3, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 80.7, "WEEK 2"
)

derive_param_bmi(
  advs,
  by_vars = exprs(USUBJID, AVISIT),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  set_values_to = exprs(
    PARAMCD = "BMI",
    PARAM = "Body Mass Index (kg/m^2)"
  ),
  get_unit_expr = extract_unit(PARAM),
  constant_by_vars = exprs(USUBJID)
)

# Example 2: Derive BMI where height is measured only once and keep only one record
# where both height and weight are measured.
derive_param_bmi(
  advs,
  by_vars = exprs(USUBJID, AVISIT),
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  set_values_to = exprs(
    PARAMCD = "BMI",
    PARAM = "Body Mass Index (kg/m^2)"
  ),
  get_unit_expr = extract_unit(PARAM)
)

# Example 3: Pediatric study where height and weight are measured multiple times
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-101-1001", "HEIGHT", "Height (cm)", 47.1, "BASELINE",
  "01-101-1001", "HEIGHT", "Height (cm)", 59.1, "WEEK 12",
  "01-101-1001", "HEIGHT", "Height (cm)", 64.7, "WEEK 24",
  "01-101-1001", "HEIGHT", "Height (cm)", 68.2, "WEEK 48",
  "01-101-1001", "WEIGHT", "Weight (kg)", 2.6, "BASELINE",
  "01-101-1001", "WEIGHT", "Weight (kg)", 5.3, "WEEK 12",
  "01-101-1001", "WEIGHT", "Weight (kg)", 6.7, "WEEK 24",
  "01-101-1001", "WEIGHT", "Weight (kg)", 7.4, "WEEK 48",
)

derive_param_bmi(
  advs,
  by_vars = exprs(USUBJID, VISIT),
```

```
  weight_code = "WEIGHT",
  height_code = "HEIGHT",
  set_values_to = exprs(
    PARAMCD = "BMI",
    PARAM = "Body Mass Index (kg/m^2)"
  ),
  get_unit_expr = extract_unit(PARAM)
)
```

`derive_param_bsa` *Adds a Parameter for BSA (Body Surface Area) Using the Specified Method*

Description

Adds a record for BSA (Body Surface Area) using the specified derivation method for each by group (e.g., subject and visit) where the source parameters are available.

Note: This is a wrapper function for the more generic `derive_param_computed()`.

Usage

```
derive_param_bsa(
  dataset,
  by_vars,
  method,
  set_values_to = exprs(PARAMCD = "BSA"),
  height_code = "HEIGHT",
  weight_code = "WEIGHT",
  get_unit_expr,
  filter = NULL,
  constant_by_vars = NULL
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well.

The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `HEIGHT` and `WEIGHT`.

Default value none

`by_vars` Grouping variables

For each group defined by `by_vars` an observation is added to the output dataset. Only variables specified in `by_vars` will be populated in the newly created records.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`method`

Derivation method to use. Note that `HEIGHT` is expected in cm and `WEIGHT` is expected in kg:

Mosteller: `sqrt(height * weight / 3600)`

DuBois-DuBois: `0.20247 * (height/100) ^ 0.725 * weight ^ 0.425`

Haycock: `0.024265 * height ^ 0.3964 * weight ^ 0.5378`

Gehan-George: `0.0235 * height ^ 0.42246 * weight ^ 0.51456`

Boyd: `0.0003207 * (height ^ 0.3) * (1000 * weight) ^ (0.7285 - (0.0188 * log10(1000 * weight)))`

Fujimoto: `0.008883 * height ^ 0.663 * weight ^ 0.444`

Takahira: `0.007241 * height ^ 0.725 * weight ^ 0.425`

Permitted values character value

Default value none

`set_values_to`

Variables to be set

The specified variables are set to the specified values for the new observations. For example `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.

Permitted values List of variable-value pairs

Default value `exprs(PARAMCD = "MAP")`

`height_code`

HEIGHT parameter code

The observations where `PARAMCD` equals the specified value are considered as the HEIGHT assessments. It is expected that HEIGHT is measured in cm.

Permitted values character value

Default value `"HEIGHT"`

`weight_code`

WEIGHT parameter code

The observations where `PARAMCD` equals the specified value are considered as the WEIGHT assessments. It is expected that WEIGHT is measured in kg.

Permitted values character value

Default value `"WEIGHT"`

`get_unit_expr`

An expression providing the unit of the parameter

The result is used to check the units of the input parameters.

Permitted values An expression which is evaluable in the input dataset and results in a character value

Default value none

`filter`

Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

constant_by_vars

By variables for when HEIGHT is constant

When HEIGHT is constant, the HEIGHT parameters (measured only once) are merged to the other parameters using the specified variables.

If height is constant (e.g. only measured once at screening or baseline) then use constant_by_vars to select the subject-level variable to merge on (e.g. USUBJID). This will produce BSA at all visits where weight is measured. Otherwise it will only be calculated at visits with both height and weight collected.

Default value NULL

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in by_vars.

See Also

compute_bsa()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```
library(tibble)

# Example 1: Derive BSA where height is measured only once using constant_by_vars
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-701-1015", "HEIGHT", "Height (cm)", 170, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 75, "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)", 78, "MONTH 1",
  "01-701-1015", "WEIGHT", "Weight (kg)", 80, "MONTH 2",
  "01-701-1028", "HEIGHT", "Height (cm)", 185, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 90, "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)", 88, "MONTH 1",
  "01-701-1028", "WEIGHT", "Weight (kg)", 85, "MONTH 2",
)

derive_param_bsa(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  method = "Mosteller",
  set_values_to = exprs(
    PARAMCD = "BSA",
    PARAM = "Body Surface Area (m^2)"
  ),
  get_unit_expr = extract_unit(PARAM),
```

```
  constant_by_vars = exprs(USUBJID)
)

derive_param_bsa(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  method = "Fujimoto",
  set_values_to = exprs(
    PARAMCD = "BSA",
    PARAM = "Body Surface Area (m^2)"
  ),
  get_unit_expr = extract_unit(PARAM),
  constant_by_vars = exprs(USUBJID)
)

# Example 2: Derive BSA where height is measured only once and keep only one record
# where both height and weight are measured.

derive_param_bsa(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  method = "Mosteller",
  set_values_to = exprs(
    PARAMCD = "BSA",
    PARAM = "Body Surface Area (m^2)"
  ),
  get_unit_expr = extract_unit(PARAM)
)

# Example 3: Pediatric study where height and weight are measured multiple times
advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-101-1001", "HEIGHT", "Height (cm)", 47.1, "BASELINE",
  "01-101-1001", "HEIGHT", "Height (cm)", 59.1, "WEEK 12",
  "01-101-1001", "HEIGHT", "Height (cm)", 64.7, "WEEK 24",
  "01-101-1001", "HEIGHT", "Height (cm)", 68.2, "WEEK 48",
  "01-101-1001", "WEIGHT", "Weight (kg)", 2.6, "BASELINE",
  "01-101-1001", "WEIGHT", "Weight (kg)", 5.3, "WEEK 12",
  "01-101-1001", "WEIGHT", "Weight (kg)", 6.7, "WEEK 24",
  "01-101-1001", "WEIGHT", "Weight (kg)", 7.4, "WEEK 48",
)
derive_param_bsa(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  method = "Mosteller",
  set_values_to = exprs(
    PARAMCD = "BSA",
    PARAM = "Body Surface Area (m^2)"
  ),
  get_unit_expr = extract_unit(PARAM)
)
```

`derive_param_computed` *Adds a Parameter Computed from the Analysis Value of Other Parameters*

Description

Adds a parameter computed from the analysis value of other parameters. It is expected that the analysis value of the new parameter is defined by an expression using the analysis values of other parameters, such as addition/sum, subtraction/difference, multiplication/product, division/ratio, exponentiation/logarithm, or by formula.

For example mean arterial pressure (MAP) can be derived from systolic (SYSBP) and diastolic blood pressure (DIABP) with the formula

$$MAP = \frac{SYSBP + 2DIABP}{3}$$

Usage

```
derive_param_computed(
  dataset = NULL,
  dataset_add = NULL,
  by_vars,
  parameters,
  set_values_to,
  filter = NULL,
  constant_by_vars = NULL,
  constant_parameters = NULL,
  keep_nas = FALSE
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD` is expected as well. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `parameters`. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** `NULL`
`dataset_add`	Additional dataset The variables specified by the `by_vars` parameter are expected. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the additional dataset after restricting it to the parameters specified by `parameters`. If the argument is specified, the observations of the additional dataset are considered in addition to the observations from the input dataset (`dataset` restricted by `filter`).

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value `NULL`

`by_vars` Grouping variables

For each group defined by `by_vars` an observation is added to the output dataset. Only variables specified in `by_vars` will be populated in the newly created records.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`parameters` Required parameter codes

It is expected that all parameter codes (`PARAMCD`) which are required to derive the new parameter are specified for this parameter or the `constant_parameters` parameter.

If observations should be considered which do not have a parameter code, e.g., if an SDTM dataset is used, temporary parameter codes can be derived by specifying a list of expressions. The name of the element defines the temporary parameter code and the expression the condition for selecting the records. For example `parameters = exprs(HGHT = VSTESTCD == "HEIGHT")` selects the observations with `VSTESTCD == "HEIGHT"` from the input data (`dataset` and `dataset_add`), sets `PARAMCD = "HGHT"` for these observations, and adds them to the observations to consider.

Unnamed elements in the list of expressions are considered as parameter codes. For example, `parameters = exprs(WEIGHT, HGHT = VSTESTCD == "HEIGHT")` uses the parameter code `"WEIGHT"` and creates a temporary parameter code `"HGHT"`.

Permitted values A character vector of `PARAMCD` values or a list of expressions

Default value none

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations. The values of variables of the parameters specified by `parameters` can be accessed using `<variable name>.<parameter code>`. For example

```
exprs(
  AVAL = (AVAL.SYSBP + 2 * AVAL.DIABP) / 3,
  PARAMCD = "MAP"
)
```

defines the analysis value and parameter code for the new parameter.

Variable names in the expression must not contain more than one dot.

Note that `dplyr` helper functions such as `dplyr::starts_with()` should be avoided unless the list of variable-value pairs is clearly specified in a statement via the `set_values_to` argument.

Permitted values list of named expressions created by `exprs()`, e.g., `exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))`

Default value none

filter
: Filter condition

  The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

  Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

  Default value `NULL`

constant_by_vars
: By variables for constant parameters

  The constant parameters (parameters that are measured only once) are merged to the other parameters using the specified variables. (Refer to Example 2)

  Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

  Default value `NULL`

constant_parameters
: Required constant parameter codes

  It is expected that all the parameter codes (`PARAMCD`) which are required to derive the new parameter and are measured only once are specified here. For example if BMI should be derived and height is measured only once while weight is measured at each visit. Height could be specified in the `constant_parameters` parameter. (Refer to Example 2)

  If observations should be considered which do not have a parameter code, e.g., if an SDTM dataset is used, temporary parameter codes can be derived by specifying a list of expressions. The name of the element defines the temporary parameter code and the expression the condition for selecting the records. For example `constant_parameters = exprs(HGHT = VSTESTCD == "HEIGHT")` selects the observations with `VSTESTCD == "HEIGHT"` from the input data (`dataset` and `dataset_add`), sets `PARAMCD = "HGHT"` for these observations, and adds them to the observations to consider.

  Unnamed elements in the list of expressions are considered as parameter codes. For example, `constant_parameters = exprs(WEIGHT, HGHT = VSTESTCD == "HEIGHT")` uses the parameter code `"WEIGHT"` and creates a temporary parameter code `"HGHT"`.

  Permitted values A character vector of `PARAMCD` values or a list of expressions

  Default value `NULL`

keep_nas
: Keep observations with `NA`s

  If the argument is set to `TRUE`, observations are added even if some of the values contributing to the computed value are `NA` (see Example 1b).

  If the argument is set to a list of variables, observations are added even if some of specified variables are `NA` (see Example 1c).

  Permitted values `TRUE`, `FALSE`, or a list of variables created by `exprs()` e.g. `exprs(ADTF, ATMF)`

  Default value `FALSE`

Details

For each group (with respect to the variables specified for the `by_vars` parameter) an observation is added to the output dataset if the filtered input dataset (`dataset`) or the additional dataset

(dataset_add) contains exactly one observation for each parameter code specified for parameters and all contributing values like AVAL.SYSBP are not NA. The keep_nas can be used to specify variables for which NAs are acceptable. See also Example 1b and 1c.

For the new observations the variables specified for set_values_to are set to the provided values. The values of the other variables of the input dataset are set to NA.

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in by_vars.

Examples

Example 1 - Data setup:

Examples 1a, 1b, and 1c use the following ADVS data.

```
ADVS <- tribble(
  ~USUBJID,      ~PARAMCD, ~PARAM,                              ~AVAL, ~VISIT,
  "01-701-1015", "DIABP",  "Diastolic Blood Pressure (mmHg)",    51, "BASELINE",
  "01-701-1015", "DIABP",  "Diastolic Blood Pressure (mmHg)",    50, "WEEK 2",
  "01-701-1015", "SYSBP",  "Systolic Blood Pressure (mmHg)",   121, "BASELINE",
  "01-701-1015", "SYSBP",  "Systolic Blood Pressure (mmHg)",    121, "WEEK 2",
  "01-701-1028", "DIABP",  "Diastolic Blood Pressure (mmHg)",   79, "BASELINE",
  "01-701-1028", "DIABP",  "Diastolic Blood Pressure (mmHg)",    80, "WEEK 2",
  "01-701-1028", "SYSBP",  "Systolic Blood Pressure (mmHg)",   130, "BASELINE",
  "01-701-1028", "SYSBP",  "Systolic Blood Pressure (mmHg)",     NA, "WEEK 2"
) %>%
  mutate(
    AVALU = "mmHg",
    ADT = case_when(
      VISIT == "BASELINE" ~ as.Date("2024-01-10"),
      VISIT == "WEEK 2" ~ as.Date("2024-01-24")
    ),
    ADTF = NA_character_
  )
```

Example 1a - Adding a parameter computed from a formula (parameters, set_values_to):

Derive mean arterial pressure (MAP) from systolic (SYSBP) and diastolic blood pressure (DIABP).

- Here, for each USUBJID and VISIT group (specified in by_vars), an observation is added to the output dataset when the filtered input dataset (dataset) contains exactly one observation for each parameter code specified for parameters and all contributing values (e.g., AVAL.SYSBP and AVAL.DIABP) are not NA. Indeed, patient 01-701-1028 does not get a "WEEK 2"-derived record as AVAL is NA for their "WEEK 2" systolic blood pressure.

```
derive_param_computed(
  ADVS,
  by_vars = exprs(USUBJID, VISIT),
  parameters = c("SYSBP", "DIABP"),
```

```
  set_values_to = exprs(
    AVAL = (AVAL.SYSBP + 2 * AVAL.DIABP) / 3,
    PARAMCD = "MAP",
    PARAM = "Mean Arterial Pressure (mmHg)",
    AVALU = "mmHg",
    ADT = ADT.SYSBP
  )
) %>%
select(-PARAM)
#> # A tibble: 11 × 7
#>    USUBJID     PARAMCD  AVAL VISIT    AVALU ADT        ADTF
#>    <chr>       <chr>   <dbl> <chr>    <chr> <date>     <chr>
#>  1 01-701-1015 DIABP    51   BASELINE mmHg  2024-01-10 <NA>
#>  2 01-701-1015 DIABP    50   WEEK 2   mmHg  2024-01-24 <NA>
#>  3 01-701-1015 SYSBP   121   BASELINE mmHg  2024-01-10 <NA>
#>  4 01-701-1015 SYSBP   121   WEEK 2   mmHg  2024-01-24 <NA>
#>  5 01-701-1028 DIABP    79   BASELINE mmHg  2024-01-10 <NA>
#>  6 01-701-1028 DIABP    80   WEEK 2   mmHg  2024-01-24 <NA>
#>  7 01-701-1028 SYSBP   130   BASELINE mmHg  2024-01-10 <NA>
#>  8 01-701-1028 SYSBP    NA   WEEK 2   mmHg  2024-01-24 <NA>
#>  9 01-701-1015 MAP      74.3 BASELINE mmHg  2024-01-10 <NA>
#> 10 01-701-1015 MAP      73.7 WEEK 2   mmHg  2024-01-24 <NA>
#> 11 01-701-1028 MAP      96   BASELINE mmHg  2024-01-10 <NA>
```

Example 1b - Keeping missing values for any source variables (`keep_nas = TRUE`):

Use option `keep_nas = TRUE` to derive MAP in the case where some/all values of a variable used in the computation are missing.

- Note that observations will be added here even if some of the values contributing to the computed values are `NA`. In particular, patient `01-701-1028` does get a `"WEEK 2"`-derived record as compared to Example 1a, but with `AVAL = NA`.

```
derive_param_computed(
  ADVS,
  by_vars = exprs(USUBJID, VISIT),
  parameters = c("SYSBP", "DIABP"),
  set_values_to = exprs(
    AVAL = (AVAL.SYSBP + 2 * AVAL.DIABP) / 3,
    PARAMCD = "MAP",
    PARAM = "Mean Arterial Pressure (mmHg)",
    AVALU = "mmHg",
    ADT = ADT.SYSBP,
    ADTF = ADTF.SYSBP
  ),
  keep_nas = TRUE
)%>%
select(-PARAM)
#> # A tibble: 12 × 7
#>    USUBJID     PARAMCD  AVAL VISIT    AVALU ADT        ADTF
#>    <chr>       <chr>   <dbl> <chr>    <chr> <date>     <chr>
```

```
#>  1 01-701-1015 DIABP    51   BASELINE mmHg  2024-01-10 <NA>
#>  2 01-701-1015 DIABP    50   WEEK 2   mmHg  2024-01-24 <NA>
#>  3 01-701-1015 SYSBP   121   BASELINE mmHg  2024-01-10 <NA>
#>  4 01-701-1015 SYSBP   121   WEEK 2   mmHg  2024-01-24 <NA>
#>  5 01-701-1028 DIABP    79   BASELINE mmHg  2024-01-10 <NA>
#>  6 01-701-1028 DIABP    80   WEEK 2   mmHg  2024-01-24 <NA>
#>  7 01-701-1028 SYSBP   130   BASELINE mmHg  2024-01-10 <NA>
#>  8 01-701-1028 SYSBP    NA   WEEK 2   mmHg  2024-01-24 <NA>
#>  9 01-701-1015 MAP      74.3 BASELINE mmHg  2024-01-10 <NA>
#> 10 01-701-1015 MAP      73.7 WEEK 2   mmHg  2024-01-24 <NA>
#> 11 01-701-1028 MAP      96   BASELINE mmHg  2024-01-10 <NA>
#> 12 01-701-1028 MAP      NA   WEEK 2   mmHg  2024-01-24 <NA>
```

Example 1c - Keeping missing values for some source variables (`keep_nas = exprs()`):

Use option `keep_nas = exprs(ADTF)` to derive MAP in the case where some/all values of a variable used in the computation are missing but keeping `NA` values of `ADTF`.

- This is subtly distinct from Examples 1a and 1b. In 1a, we do not get new derived records if any of the source records have a value of `NA` for a variable that is included in `set_values_to`. In 1b, we do the opposite and allow the creation of new records regardless of how many `NA`s we encounter in the source variables.
- Here, we want to disregard `NA` values but only from the variables that are specified via `keep_na_values`.
- This is important because we have added `ADTF` in `set_values_to`, but all values of this variable are `NA`. As such, in order to get any derived records at all, but continue not getting one when `AVAL` is `NA` in any of the source records, (see patient `"01-701-1028"` again), we specify `keep_nas = exprs(ADTF)`.

```
derive_param_computed(
  ADVS,
  by_vars = exprs(USUBJID, VISIT),
  parameters = c("SYSBP", "DIABP"),
  set_values_to = exprs(
    AVAL = (AVAL.SYSBP + 2 * AVAL.DIABP) / 3,
    PARAMCD = "MAP",
    PARAM = "Mean Arterial Pressure (mmHg)",
    AVALU = "mmHg",
    ADT = ADT.SYSBP,
    ADTF = ADTF.SYSBP
  ),
  keep_nas = exprs(ADTF)
)
#> # A tibble: 11 × 8
#>    USUBJID     PARAMCD PARAM                  AVAL VISIT AVALU ADT        ADTF
#>    <chr>       <chr>   <chr>                 <dbl> <chr> <chr> <date>     <chr>
#>  1 01-701-1015 DIABP   Diastolic Blood Press. . .  51  BASE. . . mmHg  2024-01-10 <NA>
#>  2 01-701-1015 DIABP   Diastolic Blood Press. . .  50  WEEK. . . mmHg  2024-01-24 <NA>
#>  3 01-701-1015 SYSBP   Systolic Blood Pressu. . . 121  BASE. . . mmHg  2024-01-10 <NA>
#>  4 01-701-1015 SYSBP   Systolic Blood Pressu. . . 121  WEEK. . . mmHg  2024-01-24 <NA>
```

```
#>  5 01-701-1028 DIABP  Diastolic Blood Press. . .  79  BASE. . . mmHg  2024-01-10 <NA>
#>  6 01-701-1028 DIABP  Diastolic Blood Press. . .  80  WEEK. . . mmHg  2024-01-24 <NA>
#>  7 01-701-1028 SYSBP  Systolic Blood Pressu. . . 130  BASE. . . mmHg  2024-01-10 <NA>
#>  8 01-701-1028 SYSBP  Systolic Blood Pressu. . .  NA  WEEK. . . mmHg  2024-01-24 <NA>
#>  9 01-701-1015 MAP    Mean Arterial Pressur. . .  74.3 BASE. . . mmHg  2024-01-10 <NA>
#> 10 01-701-1015 MAP    Mean Arterial Pressur. . .  73.7 WEEK. . . mmHg  2024-01-24 <NA>
#> 11 01-701-1028 MAP    Mean Arterial Pressur. . .  96  BASE. . . mmHg  2024-01-10 <NA>
```

Example 2 - Derivations using parameters measured only once (`constant_parameters` and `constant_by_vars`):

Derive BMI where `HEIGHT` is measured only once.

- In the above examples, for each parameter specified in the `parameters` argument, we expect one record per by group, where the by group is specified in `by_vars`. However, if a parameter is only measured once, it can be specified in `constant_parameters` instead.
- A modified by group still needs to be provided for the constant parameters. This can be done via `constant_by_vars`.
- See the example below, where weight is measured for each patient at each visit (`by_vars = exprs(USUBJID, VISIT)`), while height is measured for each patient only at the first visit (`constant_parameters = "HEIGHT"`, `constant_by_vars = exprs(USUBJID)`).

```
ADVS <- tribble(
  ~USUBJID,      ~PARAMCD, ~PARAM,         ~AVAL, ~AVALU, ~VISIT,
  "01-701-1015", "HEIGHT", "Height (cm)", 147.0, "cm",   "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)",  54.0, "kg",   "SCREENING",
  "01-701-1015", "WEIGHT", "Weight (kg)",  54.4, "kg",   "BASELINE",
  "01-701-1015", "WEIGHT", "Weight (kg)",  53.1, "kg",   "WEEK 2",
  "01-701-1028", "HEIGHT", "Height (cm)", 163.0, "cm",   "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)",  78.5, "kg",   "SCREENING",
  "01-701-1028", "WEIGHT", "Weight (kg)",  80.3, "kg",   "BASELINE",
  "01-701-1028", "WEIGHT", "Weight (kg)",  80.7, "kg",   "WEEK 2"
)

derive_param_computed(
  ADVS,
  by_vars = exprs(USUBJID, VISIT),
  parameters = "WEIGHT",
  set_values_to = exprs(
    AVAL = AVAL.WEIGHT / (AVAL.HEIGHT / 100)^2,
    PARAMCD = "BMI",
    PARAM = "Body Mass Index (kg/m^2)",
    AVALU = "kg/m^2"
  ),
  constant_parameters = c("HEIGHT"),
  constant_by_vars = exprs(USUBJID)
)
#> # A tibble: 14 × 6
#>    USUBJID     PARAMCD PARAM                    AVAL AVALU  VISIT
#>    <chr>       <chr>   <chr>                   <dbl> <chr>  <chr>
```

```
#>  1 01-701-1015 HEIGHT  Height (cm)              147   cm     SCREENING
#>  2 01-701-1015 WEIGHT  Weight (kg)               54   kg     SCREENING
#>  3 01-701-1015 WEIGHT  Weight (kg)               54.4 kg     BASELINE
#>  4 01-701-1015 WEIGHT  Weight (kg)               53.1 kg     WEEK 2
#>  5 01-701-1028 HEIGHT  Height (cm)              163   cm     SCREENING
#>  6 01-701-1028 WEIGHT  Weight (kg)               78.5 kg     SCREENING
#>  7 01-701-1028 WEIGHT  Weight (kg)               80.3 kg     BASELINE
#>  8 01-701-1028 WEIGHT  Weight (kg)               80.7 kg     WEEK 2
#>  9 01-701-1015 BMI     Body Mass Index (kg/m^2)  25.0 kg/m^2 SCREENING
#> 10 01-701-1015 BMI     Body Mass Index (kg/m^2)  25.2 kg/m^2 BASELINE
#> 11 01-701-1015 BMI     Body Mass Index (kg/m^2)  24.6 kg/m^2 WEEK 2
#> 12 01-701-1028 BMI     Body Mass Index (kg/m^2)  29.5 kg/m^2 SCREENING
#> 13 01-701-1028 BMI     Body Mass Index (kg/m^2)  30.2 kg/m^2 BASELINE
#> 14 01-701-1028 BMI     Body Mass Index (kg/m^2)  30.4 kg/m^2 WEEK 2
```

Example 3 - Derivations including data from an additional dataset (`dataset_add`) and non-`AVAL` variables:

Use data from an additional dataset and other variables than `AVAL`.

- In this example, the dataset specified via `dataset_add` (e.g., `QS`) is an SDTM dataset. There is no parameter code in the dataset.
- The `parameters` argument is therefore used to specify a list of expressions to derive temporary parameter codes.
- Then, `set_values_to` is used to specify the values for the new observations of each variable, and variable-value pairs from both datasets are referenced via `exprs()`.

```
QS <- tribble(
  ~USUBJID, ~AVISIT,   ~QSTESTCD, ~QSORRES, ~QSSTRESN,
  "1",      "WEEK 2",  "CHSF112", NA,              1,
  "1",      "WEEK 2",  "CHSF113", "Yes",          NA,
  "1",      "WEEK 2",  "CHSF114", NA,              1,
  "1",      "WEEK 4",  "CHSF112", NA,              2,
  "1",      "WEEK 4",  "CHSF113", "No",           NA,
  "1",      "WEEK 4",  "CHSF114", NA,              1
)

ADCHSF <- tribble(
  ~USUBJID, ~AVISIT,  ~PARAMCD, ~QSSTRESN, ~AVAL,
  "1",      "WEEK 2", "CHSF12", 1,             6,
  "1",      "WEEK 2", "CHSF14", 1,             6,
  "1",      "WEEK 4", "CHSF12", 2,            12,
  "1",      "WEEK 4", "CHSF14", 1,             6
) %>%
  mutate(QSORRES = NA_character_)

derive_param_computed(
  ADCHSF,
  dataset_add = QS,
  by_vars = exprs(USUBJID, AVISIT),
```

```
  parameters = exprs(CHSF12, CHSF13 = QSTESTCD %in% c("CHSF113"), CHSF14),
  set_values_to = exprs(
    AVAL = case_when(
      QSORRES.CHSF13 == "Not applicable" ~ 0,
      QSORRES.CHSF13 == "Yes" ~ 38,
      QSORRES.CHSF13 == "No" ~ if_else(
        QSSTRESN.CHSF12 > QSSTRESN.CHSF14,
        25,
        0
      )
    ),
    PARAMCD = "CHSF13"
  )
)
#> # A tibble: 6 × 6
#>   USUBJID AVISIT PARAMCD QSSTRESN  AVAL QSORRES
#>   <chr>   <chr>  <chr>      <dbl> <dbl> <chr>
#> 1 1       WEEK 2 CHSF12         1     6 <NA>
#> 2 1       WEEK 2 CHSF14         1     6 <NA>
#> 3 1       WEEK 4 CHSF12         2    12 <NA>
#> 4 1       WEEK 4 CHSF14         1     6 <NA>
#> 5 1       WEEK 2 CHSF13        NA    38 <NA>
#> 6 1       WEEK 4 CHSF13        NA    25 <NA>
```

Example 4 - Computing more than one variable:

Specify more than one variable-value pair via `set_values_to`.

- In this example, the values of `AVALC`, `ADTM`, `ADTF`, `PARAMCD`, and `PARAM` are determined via distinctly defined analysis values and parameter codes.
- This is different from Example 3 as more than one variable is derived.

```
ADLB_TBILIALK <- tribble(
  ~USUBJID, ~PARAMCD, ~AVALC, ~ADTM,        ~ADTF,
  "1",      "ALK2",   "Y",    "2021-05-13", NA_character_,
  "1",      "TBILI2", "Y",    "2021-06-30", "D",
  "2",      "ALK2",   "Y",    "2021-12-31", "M",
  "2",      "TBILI2", "N",    "2021-11-11", NA_character_,
  "3",      "ALK2",   "N",    "2021-04-03", NA_character_,
  "3",      "TBILI2", "N",    "2021-04-04", NA_character_
) %>%
  mutate(ADTM = ymd(ADTM))

derive_param_computed(
  dataset_add = ADLB_TBILIALK,
  by_vars = exprs(USUBJID),
  parameters = c("ALK2", "TBILI2"),
  set_values_to = exprs(
    AVALC = if_else(AVALC.TBILI2 == "Y" & AVALC.ALK2 == "Y", "Y", "N"),
    ADTM = pmax(ADTM.TBILI2, ADTM.ALK2),
    ADTF = if_else(ADTM == ADTM.TBILI2, ADTF.TBILI2, ADTF.ALK2),
```

```
    PARAMCD = "TB2AK2",
    PARAM = "TBILI > 2 times ULN and ALKPH <= 2 times ULN"
  ),
  keep_nas = TRUE
)
#> # A tibble: 3 × 6
#>   USUBJID AVALC ADTM       ADTF  PARAMCD PARAM
#>   <chr>   <chr> <date>     <chr> <chr>   <chr>
#> 1 1       Y     2021-06-30 D     TB2AK2  TBILI > 2 times ULN and ALKPH <= 2 tim. . .
#> 2 2       N     2021-12-31 M     TB2AK2  TBILI > 2 times ULN and ALKPH <= 2 tim. . .
#> 3 3       N     2021-04-04 <NA>  TB2AK2  TBILI > 2 times ULN and ALKPH <= 2 tim. . .
```

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

---

derive_param_doseint *Adds a Parameter for Dose Intensity*

Description

Adds a record for the dose intensity for each by group (e.g., subject and visit) where the source parameters are available.

Note: This is a wrapper function for the more generic derive_param_computed().

The analysis value of the new parameter is derived as Total Dose / Planned Dose * 100

Usage

```
derive_param_doseint(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "TNDOSINT"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE",
  zero_doses = "Inf",
  filter = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. PARAMCD, and AVAL are expected as well.

The variable specified by by_vars and PARAMCD must be a unique key of the input dataset after restricting it by the filter condition (filter parameter) and to the parameters specified by tadm_code and padm_code.

Default value none

by_vars

Grouping variables

Only variables specified in by_vars will be populated in the newly created records.

Default value none

set_values_to

Variables to be set

The specified variables are set to the specified values for the new observations. For example exprs(PARAMCD = "MAP") defines the parameter code for the new parameter.

Permitted values List of variable-value pairs

Default value exprs(PARAMCD = "MAP")

tadm_code

Total Doses Administered parameter code

The observations where PARAMCD equals the specified value are considered as the total dose administered. The AVAL associated with this PARAMCD will be the numerator of the dose intensity calculation.

Permitted values character value

Default value "TNDOSE"

tpadm_code

Total Doses Planned parameter code

The observations where PARAMCD equals the specified value are considered as the total planned dose. The AVAL associated with this PARAMCD will be the denominator of the dose intensity calculation.

Permitted values character value

Default value "TSNDOSE"

zero_doses

Flag indicating logic for handling 0 planned or administered doses for a by_vars group

Permitted values Inf, 100

No record is returned if either the planned (tpadm_code) or administered (tadm_code) AVAL are NA. No record is returned is a record does not exist for both tadm_code and tpadm_code for the specified by_var.

If zero_doses = Inf:

1. If the planned dose (tpadm_code) is 0 and administered dose (tadm_code) is 0, NaN is returned.
2. If the planned dose (tpadm_code) is 0 and the administered dose (tadm_code) is > 0, Inf is returned.

If zero_doses = 100 :

1. If the planned dose (tpadm_code) is 0 and administered dose (tadm_code) is 0, 0 is returned.
2. If the planned dose (tpadm_code) is 0 and the administered dose (tadm_code) is > 0, 100 is returned.

Default value "Inf"

filter — Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

Value

The input dataset with the new parameter rows added. Note, a variable will only be populated in the new parameter rows if it is specified in by_vars.

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```
library(tibble)
library(lubridate, warn.conflicts = FALSE)

adex <- tribble(
  ~USUBJID, ~PARAMCD, ~VISIT, ~ANL01FL, ~ASTDT, ~AENDT, ~AVAL,
  "P001", "TNDOSE", "V1", "Y", ymd("2020-01-01"), ymd("2020-01-30"), 59,
  "P001", "TSNDOSE", "V1", "Y", ymd("2020-01-01"), ymd("2020-02-01"), 96,
  "P001", "TNDOSE", "V2", "Y", ymd("2020-02-01"), ymd("2020-03-15"), 88,
  "P001", "TSNDOSE", "V2", "Y", ymd("2020-02-05"), ymd("2020-03-01"), 88,
  "P002", "TNDOSE", "V1", "Y", ymd("2021-01-01"), ymd("2021-01-30"), 0,
  "P002", "TSNDOSE", "V1", "Y", ymd("2021-01-01"), ymd("2021-02-01"), 0,
  "P002", "TNDOSE", "V2", "Y", ymd("2021-02-01"), ymd("2021-03-15"), 52,
  "P002", "TSNDOSE", "V2", "Y", ymd("2021-02-05"), ymd("2021-03-01"), 0
)

derive_param_doseint(
  adex,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(PARAMCD = "TNDOSINT"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE"
)

derive_param_doseint(
  adex,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(PARAMCD = "TDOSINT2"),
  tadm_code = "TNDOSE",
  tpadm_code = "TSNDOSE",
```

```
  zero_doses = "100"
)
```

---

derive_param_exist_flag *Add an Existence Flag Parameter*

Description

Add a new parameter indicating that a certain event exists in a dataset. AVALC and AVAL indicate if an event occurred or not. For example, the function can derive a parameter indicating if there is measurable disease at baseline.

Usage

```
derive_param_exist_flag(
  dataset = NULL,
  dataset_ref,
  dataset_add,
  condition,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_,
  filter_add = NULL,
  by_vars = get_admiral_option("subject_keys"),
  set_values_to
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. PARAMCD is expected as well. **Default value** NULL
dataset_ref	Reference dataset, e.g., ADSL The variables specified in by_vars are expected. For each group (as defined by by_vars) from the specified dataset (dataset_ref), the existence flag is calculated and added as a new observation to the input datasets (dataset). **Default value** none
dataset_add	Additional dataset The variables specified by the by_vars parameter are expected. This dataset is used to check if an event occurred or not. Any observation in the dataset fulfilling the event condition (condition) is considered as an event. **Default value** none

condition — Event condition

The condition is evaluated at the additional dataset (dataset_add).

For all groups where it evaluates as TRUE at least once AVALC is set to the true value (true_value) for the new observations.

For all groups where it evaluates as FALSE or NA for all observations AVALC is set to the false value (false_value).

For all groups not present in the additional dataset AVALC is set to the missing value (missing_value).

Default value none

true_value — True value

For all groups with at least one observations in the additional dataset (dataset_add) fulfilling the event condition (condition), AVALC is set to the specified value (true_value).

Permitted values A character scalar

Default value "Y"

false_value — False value

For all groups with at least one observations in the additional dataset (dataset_add) but none of them is fulfilling the event condition (condition), AVALC is set to the specified value (false_value).

Permitted values A character scalar

Default value NA_character_

missing_value — Values used for missing information

For all groups without an observation in the additional dataset (dataset_add), AVALC is set to the specified value (missing_value).

Permitted values A character scalar

Default value NA_character_

filter_add — Filter for additional data

Only observations fulfilling the specified condition are taken into account for flagging. If the parameter is not specified, all observations are considered.

Permitted values a condition

Default value NULL

by_vars — Grouping variables

Default value get_admiral_option("subject_keys")

set_values_to — Variables to set

A named list returned by exprs() defining the variables to be set for the new parameter, e.g. exprs(PARAMCD = "MDIS", PARAM = "Measurable Disease at Baseline") is expected. The values must be symbols, character strings, numeric values, NA, or expressions.

Default value none

Details

1. The additional dataset (dataset_add) is restricted to the observations matching the filter_add condition.
2. For each group in dataset_ref a new observation is created.
   - The AVALC variable is added and set to the true value (true_value) if for the group at least one observation exists in the (restricted) additional dataset where the condition evaluates to TRUE.
   - It is set to the false value (false_value) if for the group at least one observation exists and for all observations the condition evaluates to FALSE or NA.
   - Otherwise, it is set to the missing value (missing_value), i.e., for those groups not in dataset_add.
3. The variables specified by the set_values_to parameter are added to the new observations.
4. The new observations are added to input dataset.

Value

The input dataset with a new parameter indicating if an event occurred (AVALC and the variables specified by by_vars and set_value_to are populated for the new parameter).

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

# Derive a new parameter for measurable disease at baseline
adsl <- tribble(
  ~USUBJID,
  "1",
  "2",
  "3"
) %>%
  mutate(STUDYID = "XX1234")

tu <- tribble(
  ~USUBJID, ~VISIT,      ~TUSTRESC,
  "1",      "SCREENING", "TARGET",
  "1",      "WEEK 1",    "TARGET",
  "1",      "WEEK 5",    "TARGET",
  "1",      "WEEK 9",    "NON-TARGET",
  "2",      "SCREENING", "NON-TARGET",
```

```
  "2",      "SCREENING", "NON-TARGET"
) %>%
  mutate(
    STUDYID = "XX1234",
    TUTESTCD = "TUMIDENT"
  )

derive_param_exist_flag(
  dataset_ref = adsl,
  dataset_add = tu,
  filter_add = TUTESTCD == "TUMIDENT" & VISIT == "SCREENING",
  condition = TUSTRESC == "TARGET",
  false_value = "N",
  missing_value = "N",
  set_values_to = exprs(
    AVAL = yn_to_numeric(AVALC),
    PARAMCD = "MDIS",
    PARAM = "Measurable Disease at Baseline"
  )
)
```

`derive_param_exposure` *Add an Aggregated Parameter and Derive the Associated Start and End Dates*

Description

Add a record computed from the aggregated analysis value of another parameter and compute the start (`ASTDT(M)`)and end date (`AENDT(M)`) as the minimum and maximum date by `by_vars`.

Usage

```
derive_param_exposure(
  dataset = NULL,
  dataset_add,
  by_vars,
  input_code,
  filter_add = NULL,
  set_values_to = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. **Default value** `NULL`

dataset_add
: Additional dataset

  The variables specified for by_vars, analysis_var, PARAMCD, alongside either ASTDTM and AENDTM or ASTDT and AENDT are also expected. Observations from the specified dataset are going to be used to calculate and added as new records to the input dataset (dataset).

  Default value none

by_vars
: Grouping variables

  For each group defined by by_vars an observation is added to the output dataset. Only variables specified in by_vars will be populated in the newly created records.

  Default value none

input_code
: Required parameter code

  The observations where PARAMCD equals the specified value are considered to compute the summary record.

  Permitted values A character of PARAMCD value

  Default value none

filter_add
: Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within by_vars as this will help when an aggregating, lagging, or ranking function is involved.

  For example,

  - filter_add = (AVAL > mean(AVAL, na.rm = TRUE)) will filter all AVAL values greater than mean of AVAL with in by_vars.
  - filter_add = (dplyr::n() > 2) will filter n count of by_vars greater than 2.

  Default value NULL

set_values_to
: Variable-value pairs

  Set a list of variables to some specified value for the new observation(s)

  - LHS refer to a variable. It is expected that at least PARAMCD is defined.
  - RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, NA, or an expression. (e.g. exprs(PARAMCD = "TDOSE",PARCAT1 = "OVERALL")).

  Permitted values List of variable-value pairs

  Default value NULL

Details

For each group (with respect to the variables specified for the by_vars parameter), an observation is added to the output dataset and the defined values are set to the defined variables

Value

The input dataset with a new record added for each group (as defined by by_vars parameter). That is, a variable will only be populated in this new record if it is specified in by_vars. For each new record,

- set_values_to lists each specified variable and computes its value,
- the variable(s) specified on the LHS of set_values_to are set to their paired value (RHS). In addition, the start and end date are computed as the minimum/maximum dates by by_vars.

If the input datasets contains

- both AxxDTM and AxxDT then all ASTDTM,AENDTM, ASTDT, AENDT are computed
- only AxxDTM then ASTDTM,AENDTM are computed
- only AxxDT then ASTDT,AENDT are computed.

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)
library(stringr, warn.conflicts = FALSE)
adex <- tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~AVALC, ~VISIT, ~ASTDT, ~AENDT,
  "1015", "DOSE", 80, NA_character_, "BASELINE", ymd("2014-01-02"), ymd("2014-01-16"),
  "1015", "DOSE", 85, NA_character_, "WEEK 2", ymd("2014-01-17"), ymd("2014-06-18"),
  "1015", "DOSE", 82, NA_character_, "WEEK 24", ymd("2014-06-19"), ymd("2014-07-02"),
  "1015", "ADJ", NA, NA_character_, "BASELINE", ymd("2014-01-02"), ymd("2014-01-16"),
  "1015", "ADJ", NA, NA_character_, "WEEK 2", ymd("2014-01-17"), ymd("2014-06-18"),
  "1015", "ADJ", NA, NA_character_, "WEEK 24", ymd("2014-06-19"), ymd("2014-07-02"),
  "1017", "DOSE", 80, NA_character_, "BASELINE", ymd("2014-01-05"), ymd("2014-01-19"),
  "1017", "DOSE", 50, NA_character_, "WEEK 2", ymd("2014-01-20"), ymd("2014-05-10"),
  "1017", "DOSE", 65, NA_character_, "WEEK 24", ymd("2014-05-10"), ymd("2014-07-02"),
  "1017", "ADJ", NA, NA_character_, "BASELINE", ymd("2014-01-05"), ymd("2014-01-19"),
  "1017", "ADJ", NA, "ADVERSE EVENT", "WEEK 2", ymd("2014-01-20"), ymd("2014-05-10"),
  "1017", "ADJ", NA, NA_character_, "WEEK 24", ymd("2014-05-10"), ymd("2014-07-02")
) %>%
 mutate(ASTDTM = ymd_hms(paste(ASTDT, "00:00:00")), AENDTM = ymd_hms(paste(AENDT, "00:00:00")))

# Cumulative dose
adex %>%
  derive_param_exposure(
    dataset_add = adex,
    by_vars = exprs(USUBJID),
    set_values_to = exprs(
      PARAMCD = "TDOSE",
      PARCAT1 = "OVERALL",
      AVAL = sum(AVAL, na.rm = TRUE)
    ),
```

```
    input_code = "DOSE"
  ) %>%
  select(-ASTDTM, -AENDTM)

# average dose in w2-24
adex %>%
  derive_param_exposure(
    dataset_add = adex,
    by_vars = exprs(USUBJID),
    filter_add = VISIT %in% c("WEEK 2", "WEEK 24"),
    set_values_to = exprs(
      PARAMCD = "AVDW224",
      PARCAT1 = "WEEK2-24",
      AVAL = mean(AVAL, na.rm = TRUE)
    ),
    input_code = "DOSE"
  ) %>%
  select(-ASTDTM, -AENDTM)

# Any dose adjustment?
adex %>%
  derive_param_exposure(
    dataset_add = adex,
    by_vars = exprs(USUBJID),
    set_values_to = exprs(
      PARAMCD = "TADJ",
      PARCAT1 = "OVERALL",
      AVALC = if_else(sum(!is.na(AVALC)) > 0, "Y", NA_character_)
    ),
    input_code = "ADJ"
  ) %>%
  select(-ASTDTM, -AENDTM)
```

`derive_param_extreme_record` *Adds a Parameter Based on First or Last Record from Multiple Sources*

Description

[Deprecated] The `derive_param_extreme_record()` function has been deprecated in favor of `derive_extreme_event()`.

Generates parameter based on the first or last observation from multiple source datasets, based on user-defined filter, order and by group criteria. All variables of the selected observation are kept.

Usage

```
derive_param_extreme_record(
  dataset = NULL,
```

```
  sources,
  source_datasets,
  by_vars = NULL,
  order,
  mode,
  set_values_to
)
```

Arguments

dataset	Input dataset **Default value** NULL
sources	Sources A list of records_source() objects is expected. **Default value** none
source_datasets	Source datasets A named list of datasets is expected. The dataset_name field of records_source() refers to the dataset provided in the list. The variables specified by the order and the by_vars arguments are expected after applying new_vars. **Default value** none
by_vars	Grouping variables If the argument is specified, for each by group the observations are selected separately. **Default value** NULL
order	Sort order If the argument is set to a non-null value, for each by group the first or last observation from the source datasets is selected with respect to the specified order. Variables created via new_vars e.g., imputed date variables, can be specified as well (see examples below). Please note that NA is considered as the last value. I.e., if a order variable is NA and mode = "last", this observation is chosen while for mode = "first" the observation is chosen only if there are no observations where the variable is not NA. **Permitted values** list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) **Default value** none
mode	Selection mode (first or last) If "first" is specified, for each by group the first observation with respect to order is included in the output dataset. If "last" is specified, the last observation is included in the output dataset. **Permitted values** "first", "last" **Default value** none

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations.

A list of variable name-value pairs is expected.

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value or `NA`, e.g., `exprs(PARAMCD = "PD", PARAM = "First Progressive Disease")`.

Default value none

Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the `filter` element are selected.
2. Variables specified by `new_vars` are created for each source dataset.
3. The first or last observation (with respect to the `order` variable) for each by group (specified by `by_vars`) from multiple sources is selected and added to the input dataset.

Value

The input dataset with the first or last observation of each by group added as new observations.

See Also

Other deprecated: `call_user_fun()`, `date_source()`, `derive_var_dthcaus()`, `derive_var_extreme_dt()`, `derive_var_extreme_dtm()`, `derive_var_merged_summary()`, `dthcaus_source()`, `get_summary_records()`

Examples

```
aevent_samp <- tibble::tribble(
  ~USUBJID, ~PARAMCD,                        ~PARAM,     ~RSSTDTC,
  "1",           "PD",  "First Progressive Disease", "2022-04-01",
  "2",           "PD",  "First Progressive Disease", "2021-04-01",
  "3",           "PD",  "First Progressive Disease", "2023-04-01"
)

cm <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~CMDECOD,     ~CMSTDTC,
  "1001",         "1",    "ACT", "2021-12-25"
)

pr <- tibble::tribble(
  ~STUDYID, ~USUBJID, ~PRDECOD,     ~PRSTDTC,
  "1001",         "1",    "ACS", "2021-12-27",
  "1001",         "2",    "ACS", "2020-12-25",
  "1001",         "3",    "ACS", "2022-12-25",
)
derive_param_extreme_record(
  dataset = aevent_samp,
```

```
  sources = list(
    records_source(
      dataset_name = "cm",
      filter = CMDECOD == "ACT",
      new_vars = exprs(
        ADT = convert_dtc_to_dt(CMSTDTC),
        AVALC = CMDECOD
      )
    ),
    records_source(
      dataset_name = "pr",
      filter = PRDECOD == "ACS",
      new_vars = exprs(
        ADT = convert_dtc_to_dt(PRSTDTC),
        AVALC = PRDECOD
      )
    )
  ),
  source_datasets = list(cm = cm, pr = pr),
  by_vars = exprs(USUBJID),
  order = exprs(ADT),
  mode = "first",
  set_values_to = exprs(
    PARAMCD = "FIRSTACT",
    PARAM = "First Anti-Cancer Therapy"
  )
)
```

`derive_param_framingham` *Adds a Parameter for Framingham Heart Study Cardiovascular Disease 10-Year Risk Score*

Description

Adds a record for framingham score (FCVD101) for each by group (e.g., subject and visit) where the source parameters are available.

Usage

```
derive_param_framingham(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "FCVD101"),
  sysbp_code = "SYSBP",
  chol_code = "CHOL",
  cholhdl_code = "CHOLHDL",
  age = AGE,
  sex = SEX,
```

```
  smokefl = SMOKEFL,
  diabetfl = DIABETFL,
  trthypfl = TRTHYPFL,
  get_unit_expr,
  filter = NULL
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well.

The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `sysbp_code`, `chol_code` and `hdl_code`.

Default value none

`by_vars` Grouping variables

Only variables specified in `by_vars` will be populated in the newly created records.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations. For example `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.

Permitted values List of variable-value pairs

Default value `exprs(PARAMCD = "MAP")`

`sysbp_code` Systolic blood pressure parameter code

The observations where `PARAMCD` equals the specified value are considered as the systolic blood pressure assessments.

Permitted values a character scalar, i.e., a character vector of length one

Default value `"SYSBP"`

`chol_code` Total serum cholesterol code

The observations where `PARAMCD` equals the specified value are considered as the total cholesterol assessments. This must be measured in mg/dL.

Permitted values a character scalar, i.e., a character vector of length one

Default value `"CHOL"`

`cholhdl_code` HDL serum cholesterol code

The observations where `PARAMCD` equals the specified value are considered as the HDL cholesterol assessments. This must be measured in mg/dL.

Permitted values a character scalar, i.e., a character vector of length one

Default value `"CHOLHDL"`

`age` Subject age

A variable containing the subject's age.

Permitted values A numeric variable name that refers to a subject age column of the input dataset

Default value `AGE`

`sex` Subject sex

A variable containing the subject's sex.

Permitted values A character variable name that refers to a subject sex column of the input dataset

Default value `SEX`

`smokefl` Smoking status flag

A flag indicating smoking status.

Permitted values A character variable name that refers to a smoking status column of the input dataset.

Default value `SMOKEFL`

`diabetfl` Diabetic flag

A flag indicating diabetic status.

Permitted values A character variable name that refers to a diabetic status column of the input dataset

Default value `DIABETFL`

`trthypfl` Treated with hypertension medication flag

A flag indicating if a subject was treated with hypertension medication.

Permitted values A character variable name that refers to a column that indicates whether a subject is treated for high blood pressure

Default value `TRTHYPFL`

`get_unit_expr` An expression providing the unit of the parameter

The result is used to check the units of the input parameters.

Permitted values An expression which is evaluable in the input dataset and results in a character value

Default value none

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

Details

The values of `age`, `sex`, `smokefl`, `diabetfl` and `trthypfl` will be added to the `by_vars` list. The predicted probability of having cardiovascular disease (CVD) within 10-years according to Framingham formula. See AHA Journal article General Cardiovascular Risk Profile for Use in Primary Care for reference.

For Women:

Factor	Amount
Age	2.32888
Total Chol	1.20904
HDL Chol	-0.70833
Sys BP	2.76157
Sys BP + Hypertension Meds	2.82263
Smoker	0.52873
Non-Smoker	0
Diabetic	0.69154
Not Diabetic	0
Average Risk	26.1931
Risk Period	0.95012

For Men:

Factor	Amount
Age	3.06117
Total Chol	1.12370
HDL Chol	-0.93263
Sys BP	1.93303
Sys BP + Hypertension Meds	2.99881
Smoker	.65451
Non-Smoker	0
Diabetic	0.57367
Not Diabetic	0
Average Risk	23.9802
Risk Period	0.88936

The equation for calculating risk:

$$RiskFactors = (log(Age) * AgeFactor) + (log(TotalChol) * TotalCholFactor) + (log(CholHDL) * CholHDLFactor)$$

$$Risk = 100 * (1 - RiskPeriodFactor^{RiskFactors})$$

Value

The input dataset with the new parameter added

See Also

compute_framingham()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```
library(tibble)

adcvrisk <- tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU,
  ~VISIT, ~AGE, ~SEX, ~SMOKEFL, ~DIABETFL, ~TRTHYPFL,
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121,
  "mmHg", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 115,
  "mmHg", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1015", "CHOL", "Total Cholesterol (mg/dL)", 216.16,
  "mg/dL", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "CHOL", "Total Cholesterol (mg/dL)", 210.78,
  "mg/dL", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1015", "CHOLHDL", "Cholesterol/HDL-Cholesterol (mg/dL)", 54.91,
  "mg/dL", "BASELINE", 44, "F", "N", "N", "N",
  "01-701-1015", "CHOLHDL", "Cholesterol/HDL-Cholesterol (mg/dL)", 26.72,
  "mg/dL", "WEEK 2", 44, "F", "N", "N", "Y",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 119,
  "mmHg", "BASELINE", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 101,
  "mmHg", "WEEK 2", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "CHOL", "Total Cholesterol (mg/dL)", 292.01,
  "mg/dL", "BASELINE", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "CHOL", "Total Cholesterol (mg/dL)", 246.73,
  "mg/dL", "WEEK 2", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "CHOLHDL", "Cholesterol/HDL-Cholesterol (mg/dL)", 65.55,
  "mg/dL", "BASELINE", 55, "M", "Y", "Y", "Y",
  "01-701-1028", "CHOLHDL", "Cholesterol/HDL-Cholesterol (mg/dL)", 44.62,
  "mg/dL", "WEEK 2", 55, "M", "Y", "Y", "Y"
)


adcvrisk %>%
  derive_param_framingham(
    by_vars = exprs(USUBJID, VISIT),
    set_values_to = exprs(
      PARAMCD = "FCVD101",
      PARAM = "FCVD1-Framingham CVD 10-Year Risk Score (%)"
    ),
    get_unit_expr = AVALU
  )

derive_param_framingham(
  adcvrisk,
```

```
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(
    PARAMCD = "FCVD101",
    PARAM = "FCVD1-Framingham CVD 10-Year Risk Score (%)"
  ),
  get_unit_expr = extract_unit(PARAM)
)
```

`derive_param_map` *Adds a Parameter for Mean Arterial Pressure*

Description

Adds a record for mean arterial pressure (MAP) for each by group (e.g., subject and visit) where the source parameters are available.

Note: This is a wrapper function for the more generic `derive_param_computed()`.

Usage

```
derive_param_map(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "MAP"),
  sysbp_code = "SYSBP",
  diabp_code = "DIABP",
  hr_code = NULL,
  get_unit_expr,
  filter = NULL
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well.

The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` parameter) and to the parameters specified by `sysbp_code`, `diabp_code` and `hr_code`.

Default value none

`by_vars` Grouping variables

For each group defined by `by_vars` an observation is added to the output dataset. Only variables specified in `by_vars` will be populated in the newly created records.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations. For example `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.

Permitted values List of variable-value pairs

Default value `exprs(PARAMCD = "MAP")`

`sysbp_code` Systolic blood pressure parameter code

The observations where `PARAMCD` equals the specified value are considered as the systolic blood pressure assessments.

Permitted values character value

Default value `"SYSBP"`

`diabp_code` Diastolic blood pressure parameter code

The observations where `PARAMCD` equals the specified value are considered as the diastolic blood pressure assessments.

Permitted values character value

Default value `"DIABP"`

`hr_code` Heart rate parameter code

The observations where `PARAMCD` equals the specified value are considered as the heart rate assessments.

Permitted values character value

Default value `NULL`

`get_unit_expr` An expression providing the unit of the parameter

The result is used to check the units of the input parameters.

Permitted values An expression which is evaluable in the input dataset and results in a character value

Default value none

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

Details

The analysis value of the new parameter is derived as

$$\frac{2DIABP + SYSBP}{3}$$

if it is based on diastolic and systolic blood pressure and

$$DIABP + 0.01e^{4.14 - \frac{40.74}{HR}}(SYSBP - DIABP)$$

if it is based on diastolic, systolic blood pressure, and heart rate.

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in by_vars.

See Also

compute_map()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs(), derive_summary_records()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

advs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~VISIT,
  "01-701-1015", "PULSE", "Pulse (beats/min)", 59, "BASELINE",
  "01-701-1015", "PULSE", "Pulse (beats/min)", 61, "WEEK 2",
  "01-701-1015", "DIABP", "Diastolic Blood Pressure (mmHg)", 51, "BASELINE",
  "01-701-1015", "DIABP", "Diastolic Blood Pressure (mmHg)", 50, "WEEK 2",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121, "BASELINE",
  "01-701-1015", "SYSBP", "Systolic Blood Pressure (mmHg)", 121, "WEEK 2",
  "01-701-1028", "PULSE", "Pulse (beats/min)", 62, "BASELINE",
  "01-701-1028", "PULSE", "Pulse (beats/min)", 77, "WEEK 2",
  "01-701-1028", "DIABP", "Diastolic Blood Pressure (mmHg)", 79, "BASELINE",
  "01-701-1028", "DIABP", "Diastolic Blood Pressure (mmHg)", 80, "WEEK 2",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 130, "BASELINE",
  "01-701-1028", "SYSBP", "Systolic Blood Pressure (mmHg)", 132, "WEEK 2"
)

# Derive MAP based on diastolic and systolic blood pressure
advs %>%
  derive_param_map(
    by_vars = exprs(USUBJID, VISIT),
    set_values_to = exprs(
      PARAMCD = "MAP",
      PARAM = "Mean Arterial Pressure (mmHg)"
    ),
    get_unit_expr = extract_unit(PARAM)
  ) %>%
  filter(PARAMCD != "PULSE")

# Derive MAP based on diastolic and systolic blood pressure and heart rate
derive_param_map(
  advs,
  by_vars = exprs(USUBJID, VISIT),
  hr_code = "PULSE",
```

```
  set_values_to = exprs(
    PARAMCD = "MAP",
    PARAM = "Mean Arterial Pressure (mmHg)"
  ),
  get_unit_expr = extract_unit(PARAM)
)
```

`derive_param_qtc` *Adds a Parameter for Corrected QT (an ECG measurement)*

Description

Adds a record for corrected QT using either Bazett's, Fridericia's or Sagie's formula for each by group (e.g., subject and visit) where the source parameters are available.

Note: This is a wrapper function for the more generic `derive_param_computed()`.

Usage

```
derive_param_qtc(
  dataset,
  by_vars,
  method,
  set_values_to = default_qtc_paramcd(method),
  qt_code = "QT",
  rr_code = "RR",
  get_unit_expr,
  filter = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `get_unit_expr` arguments are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` argument) and to the parameters specified by `qt_code` and `rr_code`. **Default value** none
`by_vars`	Grouping variables Only variables specified in `by_vars` will be populated in the newly created records. **Permitted values** list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)` **Default value** none

`method` Method used to QT correction

See `compute_qtc()` for details.

Permitted values `"Bazett"`, `"Fridericia"`, `"Sagie"`

Default value none

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations. For example `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.

Permitted values List of variable-value pairs

Default value `exprs(PARAMCD = "MAP")`

`qt_code` QT parameter code

The observations where `PARAMCD` equals the specified value are considered as the QT interval assessments. It is expected that QT is measured in ms or msec.

Permitted values character value

Default value `"QT"`

`rr_code` RR parameter code

The observations where `PARAMCD` equals the specified value are considered as the RR interval assessments. It is expected that RR is measured in ms or msec.

Permitted values character value

Default value `"RR"`

`get_unit_expr` An expression providing the unit of the parameter

The result is used to check the units of the input parameters.

Permitted values An expression which is evaluable in the input dataset and results in a character value

Default value none

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

See Also

`compute_qtc()`

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_basetype_records()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_rr()`, `derive_param_wbc_abs()`, `derive_summary_records()`

Examples

```
library(tibble)

adeg <- tribble(
  ~USUBJID,      ~PARAMCD, ~PARAM,                    ~AVAL, ~AVALU,      ~VISIT,
  "01-701-1015", "HR",     "Heart Rate (beats/min)", 70.14, "beats/min", "BASELINE",
  "01-701-1015", "QT",     "QT Duration (ms)",         370, "ms",        "WEEK 2",
  "01-701-1015", "HR",     "Heart Rate (beats/min)", 62.66, "beats/min", "WEEK 1",
  "01-701-1015", "RR",     "RR Duration (ms)",         710, "ms",        "WEEK 2",
  "01-701-1028", "HR",     "Heart Rate (beats/min)", 85.45, "beats/min", "BASELINE",
  "01-701-1028", "QT",     "QT Duration (ms)",         480, "ms",        "WEEK 2",
  "01-701-1028", "QT",     "QT Duration (ms)",         350, "ms",        "WEEK 3",
  "01-701-1028", "HR",     "Heart Rate (beats/min)", 56.54, "beats/min", "WEEK 3",
  "01-701-1028", "RR",     "RR Duration (ms)",         842, "ms",        "WEEK 2"
)

derive_param_qtc(
  adeg,
  by_vars = exprs(USUBJID, VISIT),
  method = "Bazett",
  set_values_to = exprs(
    PARAMCD = "QTCBR",
    PARAM = "QTcB - Bazett's Correction Formula Rederived (ms)",
    AVALU = "ms"
  ),
  get_unit_expr = AVALU
)

derive_param_qtc(
  adeg,
  by_vars = exprs(USUBJID, VISIT),
  method = "Fridericia",
  set_values_to = exprs(
    PARAMCD = "QTCFR",
    PARAM = "QTcF - Fridericia's Correction Formula Rederived (ms)",
    AVALU = "ms"
  ),
  get_unit_expr = extract_unit(PARAM)
)

derive_param_qtc(
  adeg,
  by_vars = exprs(USUBJID, VISIT),
  method = "Sagie",
  set_values_to = exprs(
    PARAMCD = "QTLCR",
    PARAM = "QTlc - Sagie's Correction Formula Rederived (ms)",
    AVALU = "ms"
  ),
  get_unit_expr = extract_unit(PARAM)
)
```

derive_param_rr *Adds a Parameter for Derived RR (an ECG measurement)*

Description

Adds a record for derived RR based on heart rate for each by group (e.g., subject and visit) where the source parameters are available.

Note: This is a wrapper function for the more generic `derive_param_computed()`.

The analysis value of the new parameter is derived as

$$\frac{60000}{HR}$$

Usage

```
derive_param_rr(
  dataset,
  by_vars,
  set_values_to = exprs(PARAMCD = "RRR"),
  hr_code = "HR",
  get_unit_expr,
  filter = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. `PARAMCD`, and `AVAL` are expected as well. The variable specified by `by_vars` and `PARAMCD` must be a unique key of the input dataset after restricting it by the filter condition (`filter` argument) and to the parameters specified by `hr_code`. **Default value** none
`by_vars`	Grouping variables For each group defined by `by_vars` an observation is added to the output dataset. Only variables specified in `by_vars` will be populated in the newly created records. **Permitted values** list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)` **Default value** none
`set_values_to`	Variables to be set The specified variables are set to the specified values for the new observations. For example `exprs(PARAMCD = "MAP")` defines the parameter code for the new parameter.

Permitted values List of variable-value pairs

Default value `exprs(PARAMCD = "MAP")`

`hr_code` HR parameter code

The observations where `PARAMCD` equals the specified value are considered as the heart rate assessments.

Permitted values character value

Default value `"HR"`

`get_unit_expr` An expression providing the unit of the parameter

The result is used to check the units of the input parameters.

Permitted values An expression which is evaluable in the input dataset and results in a character value

Default value none

`filter` Filter condition

The specified condition is applied to the input dataset before deriving the new parameter, i.e., only observations fulfilling the condition are taken into account.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

Value

The input dataset with the new parameter added. Note, a variable will only be populated in the new parameter rows if it is specified in `by_vars`.

See Also

`compute_rr()`

BDS-Findings Functions for adding Parameters/Records: `default_qtc_paramcd()`, `derive_basetype_records()`, `derive_expected_records()`, `derive_extreme_event()`, `derive_extreme_records()`, `derive_locf_records()`, `derive_param_bmi()`, `derive_param_bsa()`, `derive_param_computed()`, `derive_param_doseint()`, `derive_param_exist_flag()`, `derive_param_exposure()`, `derive_param_framingham()`, `derive_param_map()`, `derive_param_qtc()`, `derive_param_wbc_abs()`, `derive_summary_records()`

Examples

```
library(tibble)

adeg <- tribble(
  ~USUBJID, ~PARAMCD, ~PARAM, ~AVAL, ~AVALU, ~VISIT,
  "01-701-1015", "HR", "Heart Rate", 70.14, "beats/min", "BASELINE",
  "01-701-1015", "QT", "QT Duration", 370, "ms", "WEEK 2",
  "01-701-1015", "HR", "Heart Rate", 62.66, "beats/min", "WEEK 1",
  "01-701-1015", "RR", "RR Duration", 710, "ms", "WEEK 2",
  "01-701-1028", "HR", "Heart Rate", 85.45, "beats/min", "BASELINE",
  "01-701-1028", "QT", "QT Duration", 480, "ms", "WEEK 2",
  "01-701-1028", "QT", "QT Duration", 350, "ms", "WEEK 3",
  "01-701-1028", "HR", "Heart Rate", 56.54, "beats/min", "WEEK 3",
```

```
  "01-701-1028", "RR", "RR Duration", 842, "ms", "WEEK 2"
)

derive_param_rr(
  adeg,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(
    PARAMCD = "RRR",
    PARAM = "RR Duration Rederived (ms)",
    AVALU = "ms"
  ),
  get_unit_expr = AVALU
)
```

`derive_param_tte` *Derive a Time-to-Event Parameter*

Description

Add a time-to-event parameter to the input dataset.

Usage

```
derive_param_tte(
  dataset = NULL,
  dataset_adsl,
  source_datasets,
  by_vars = NULL,
  start_date = TRTSDT,
  end_dates = NULL,
  event_conditions,
  censor_conditions = NULL,
  event_type = "negative",
  create_datetime = FALSE,
  set_values_to,
  subject_keys = get_admiral_option("subject_keys"),
  check_type = "warning"
)
```

Arguments

`dataset`	Input dataset `PARAMCD` is expected. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** `NULL`
`dataset_adsl`	ADSL input dataset The variables specified for `start_date`, and `subject_keys` are expected.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`source_datasets`

Source datasets

A named list of datasets is expected. The `dataset_name` field of `tte_source()` refers to the dataset provided in the list.

Permitted values named list of datasets, e.g., `list(adsl = adsl, ae = ae)`

Default value none

`by_vars` By variables

If the parameter is specified, separate time to event parameters are derived for each by group.

The by variables must be in at least one of the source datasets. Each source dataset must contain either all by variables or none of the by variables.

The by variables are not included in the output dataset.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`start_date` Time to event origin date

The variable `STARTDT` is set to the specified date. The value is taken from the ADSL dataset.

If the event or censoring date is before the origin date, `ADT` is set to the origin date.

Permitted values a date or datetime variable

Default value `TRTSDT`

`end_dates` Time to event end date(s)

A list of `censor_source()` objects is expected. Each date restricts the observation period for time-to-event analysis. For each subject the earliest date across all `end_dates` is used as the end date for that subject. The records defined by `event_conditions` and `censor_conditions` are restricted to dates before or equal to the selected end date.

This argument should be specified if there is more than one reason for stopping the observation of a subject, e.g., end of study, death, or intercurrent events like start of new drug. If there is only one reason for stopping the observation, it is sufficient to just include this as a censoring condition in `censor_conditions`.

See the Differentiating censoring reasons and Differentiating censoring date description examples to see how the values (`set_value_to`) set by end dates interact with the values set by the censoring conditions.

Permitted values a list of source objects, e.g., `list(pd, death)`

Default value `NULL`

`event_conditions`

Sources and conditions defining events

A list of `event_source()` objects is expected.

Permitted values a list of source objects, e.g., `list(pd, death)`

Default value none

censor_conditions

Sources and conditions defining censorings

A list of censor_source() objects is expected. Each record defined by the censor_conditions should be a possible censoring time, i.e., at this time it should be known that the event of interest has not yet occurred. Assessment where it is not known whether the event of interest has occurred or not should not be included as censoring times, e.g., when an assessment was not evaluable.

It is acceptable to include records with event as long as these are also included in event_conditions because events take precedence over censorings.

Permitted values a list of source objects, e.g., list(pd, death)

Default value NULL

event_type

Type of event

For events that are considered unfavorable, e.g., adverse events, progression, worsening, etc., the value should be "negative" and for events that are considered favorable, e.g., response to treatment, improvement, etc., the value should be "positive".

If event_type is specified as "positive", the objects specified for end_dates are added to the censoring conditions (censor_conditions). I.e., if a subject is censored, it is censored at the earliest date provided by end_dates.

Permitted values "negative", "positive"

Default value "negative"

create_datetime

Create datetime variables?

If set to TRUE, variables ADTM and STARTDTM are created. Otherwise, variables ADT and STARTDT are created.

Permitted values TRUE, FALSE

Default value FALSE

set_values_to

Variables to set

A named list returned by exprs() defining the variables to be set for the new parameter, e.g. exprs(PARAMCD = "OS", PARAM = "Overall Survival") is expected. The values must be symbols, character strings, numeric values, expressions, or NA.

Permitted values list of named expressions created by exprs(), e.g., exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))

Default value none

subject_keys

Variables to uniquely identify a subject

A list of symbols created using exprs() is expected.

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value get_admiral_option("subject_keys")

check_type — Check uniqueness

If "warning", "message", or "error" is specified, the specified message is issued if the observations of the source datasets are not unique with respect to the by variables and the date and order specified in the event_source() and censor_source() objects.

Permitted values "none", "message", "warning", "error"

Default value "warning"

Details

The following steps are performed to create the observations of the new parameter:

Deriving the events:

1. For each event source dataset the observations as specified by the filter element are selected. If the end_dates argument is specified, records after the first of the end dates are excluded. Then for each subject the first observation (with respect to date and order) is selected.
2. The ADT variable is set to the variable specified by the date element. If the date variable is a datetime variable, only the datepart is copied.
3. The CNSR variable is added and set to the censor element.
4. The variables specified by the set_values_to element are added.
5. The selected observations of all event source datasets are combined into a single dataset.
6. For each subject the first observation (with respect to the ADT/ADTM variable) from the single dataset is selected. If there is more than one event with the same date, the first event with respect to the order of events in event_conditions is selected.

Deriving the censoring observations:

1. For each censoring source dataset:
   (a) The observations as specified by the filter element are selected.
   (b) If the end_dates argument is specified, records after the first of the end dates are excluded and the variables defined by the set_values_to element of the first end date are added.
   (c) If event_type = "positive" and the end_dates argument is specified, new records with the first end date and the variables defined by the set_values_to element are added. (These will be selected in the next step, i.e., for positive events the first end date is used as censoring date.)
   (d) Then for each subject the last observation (with respect to date and order) is selected.
2. The ADT variable is set to the variable specified by the date element. If the date variable is a datetime variable, only the datepart is copied.
3. The CNSR variable is added. If the end_dates argument is specified and the consider_end_dates element is TRUE, it is set to the censor value of the first of the end dates. Otherwise, it set to the censor element.
4. The variables specified by the set_values_to element are added.
5. The selected observations of all censoring source datasets are combined into a single dataset.

6. For each subject the last observation (with respect to the ADT/ADTM variable) from the single dataset is selected. If there is more than one censoring with the same date, the last censoring with respect to the order of censorings in censor_conditions is selected.

For each subject (as defined by the subject_keys parameter) an observation is selected. If an event is available, the event observation is selected. Otherwise the censoring observation is selected.

Finally:

1. The variable specified for start_date is joined from the ADSL dataset. Only subjects in both datasets are kept, i.e., subjects with both an event or censoring and an observation in dataset_adsl.
2. The variables as defined by the set_values_to parameter are added.
3. The ADT/ADTM variable is set to the maximum of ADT/ADTM and STARTDT/STARTDTM (depending on the create_datetime parameter).
4. The new observations are added to the output dataset.

Value

The input dataset with the new parameter added

Examples

Add a basic time to event parameter:

For each subject the time to first adverse event should be created as a parameter.

- The event source object is created using event_source() and the date is set to adverse event start date.
- The censor source object is created using censor_source() and the date is set to end of study date.
- The event and censor source objects are then passed to derive_param_tte() to derive the time to event parameter with the provided parameter descriptions (PARAMCD and PARAM).
- Note the values of the censor variable (CNSR) that are derived below, where the first subject has an event and the second does not.

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)

adsl <- tribble(
  ~USUBJID, ~TRTSDT,           ~EOSDT,            ~NEWDRGDT,
  "01",     ymd("2020-12-06"), ymd("2021-03-06"), NA,
  "02",     ymd("2021-01-16"), ymd("2021-02-03"), ymd("2021-01-03")
) %>%
  mutate(STUDYID = "AB42")

adae <- tribble(
  ~USUBJID, ~ASTDT,            ~AESEQ, ~AEDECOD,
  "01",     ymd("2021-01-03"),      1, "Flu",
  "01",     ymd("2021-03-04"),      2, "Cough",
```

```
  "01",     ymd("2021-03-05"),      3, "Cough"
) %>%
  mutate(STUDYID = "AB42")

ttae <- event_source(
  dataset_name = "adae",
  date = ASTDT,
  set_values_to = exprs(
    EVNTDESC = "AE",
    SRCDOM = "ADAE",
    SRCVAR = "ASTDT",
    SRCSEQ = AESEQ
  )
)

eos <- censor_source(
  dataset_name = "adsl",
  date = EOSDT,
  set_values_to = exprs(
    EVNTDESC = "END OF STUDY",
    SRCDOM = "ADSL",
    SRCVAR = "EOSDT"
  )
)

derive_param_tte(
  dataset_adsl = adsl,
  event_conditions = list(ttae),
  censor_conditions = list(eos),
  source_datasets = list(adsl = adsl, adae = adae),
  set_values_to = exprs(
    PARAMCD = "TTAE",
    PARAM = "Time to First Adverse Event"
  )
) %>%
  select(USUBJID, STARTDT, PARAMCD, PARAM, ADT, CNSR, SRCSEQ)
#> # A tibble: 2 × 7
#>   USUBJID STARTDT    PARAMCD PARAM                     ADT        CNSR SRCSEQ
#>   <chr>   <date>     <chr>   <chr>                     <date>    <int>  <dbl>
#> 1 01      2020-12-06 TTAE    Time to First Adverse Event 2021-01-03    0      1
#> 2 02      2021-01-16 TTAE    Time to First Adverse Event 2021-02-03    1     NA
```

Adding a by variable (`by_vars`):

By variables can be added using the `by_vars` argument, e.g., now for each subject the time to first occurrence of each adverse event preferred term (`AEDECOD`) should be created as parameters.

Please note that CDISC requires separate parameters (`PARAMCD`, `PARAM`) for the by groups. Therefore the variables specified for the `by_vars` parameter are not included in the output dataset. The `PARAMCD` variable should be specified for the `set_value_to` parameter using an expression on the

right hand side which results in a unique value for each by group. If the values of the by variables should be included in the output dataset, they can be stored in PARCATy variables.

```
derive_param_tte(
  dataset_adsl = adsl,
  by_vars = exprs(AEDECOD),
  event_conditions = list(ttae),
  censor_conditions = list(eos),
  source_datasets = list(adsl = adsl, adae = adae),
  set_values_to = exprs(
    PARAMCD = paste0("TTAE", as.numeric(as.factor(AEDECOD))),
    PARAM = paste("Time to First", AEDECOD, "Adverse Event")
  )
) %>%
  select(USUBJID, STARTDT, PARAMCD, PARAM, ADT, CNSR, SRCSEQ)
#> # A tibble: 4 × 7
#>   USUBJID STARTDT    PARAMCD PARAM                     ADT        CNSR SRCSEQ
#>   <chr>   <date>     <chr>   <chr>                     <date>    <int>  <dbl>
#> 1 01    2020-12-06 TTAE1   Time to First Cough Advers. . .  2021-03-04    0     2
#> 2 01    2020-12-06 TTAE2   Time to First Flu Adverse . . .  2021-01-03    0     1
#> 3 02    2021-01-16 TTAE1   Time to First Cough Advers. . .  2021-02-03    1    NA
#> 4 02    2021-01-16 TTAE2   Time to First Flu Adverse . . .  2021-02-03    1    NA
```

Handling duplicates (check_type)**:**

The source records are checked regarding duplicates with respect to the by variables and the date and order specified in the source objects. By default, a warning is issued if any duplicates are found. Note here how after creating a new adverse event dataset containing a duplicate date for "Cough", it was then passed to the function using the source_datasets argument - where you see below adae = adae_dup.

```
adae_dup <- tribble(
  ~USUBJID, ~ASTDT,            ~AESEQ, ~AEDECOD, ~AESER,
  "01",     ymd("2021-01-03"),      1, "Flu",    "Y",
  "01",     ymd("2021-03-04"),      2, "Cough",  "N",
  "01",     ymd("2021-03-04"),      3, "Cough",  "Y"
) %>%
  mutate(STUDYID = "AB42")

derive_param_tte(
  dataset_adsl = adsl,
  by_vars = exprs(AEDECOD),
  start_date = TRTSDT,
  source_datasets = list(adsl = adsl, adae = adae_dup),
  event_conditions = list(ttae),
  censor_conditions = list(eos),
  set_values_to = exprs(
    PARAMCD = paste0("TTAE", as.numeric(as.factor(AEDECOD))),
    PARAM = paste("Time to First", AEDECOD, "Adverse Event")
  )
```

```
)
#> # A tibble: 4 × 11
#>   USUBJID STUDYID EVNTDESC     SRCDOM SRCVAR SRCSEQ  CNSR ADT        STARTDT
#>   <chr>   <chr>   <chr>        <chr>  <chr>   <dbl> <int> <date>     <date>
#> 1 01      AB42    AE           ADAE   ASTDT       2     0 2021-03-04 2020-12-06
#> 2 01      AB42    AE           ADAE   ASTDT       1     0 2021-01-03 2020-12-06
#> 3 02      AB42    END OF STUDY ADSL   EOSDT      NA     1 2021-02-03 2021-01-16
#> 4 02      AB42    END OF STUDY ADSL   EOSDT      NA     1 2021-02-03 2021-01-16
#> # i 2 more variables: PARAMCD <chr>, PARAM <chr>
#> Warning: Dataset "adae" contains duplicate records with respect to `STUDYID`, `USUBJID`,
#> `AEDECOD`, and `ASTDT`
#> i Run `admiral::get_duplicates_dataset()` to access the duplicate records
```

For investigating the issue, the dataset of the duplicate source records can be obtained by calling `get_duplicates_dataset()`:

```
get_duplicates_dataset()
#> Duplicate records with respect to `STUDYID`, `USUBJID`, `AEDECOD`, and `ASTDT`.
#> # A tibble: 2 × 6
#>   STUDYID USUBJID AEDECOD ASTDT      AESEQ AESER
#> * <chr>   <chr>   <chr>   <date>     <dbl> <chr>
#> 1 AB42    01      Cough   2021-03-04     2 N
#> 2 AB42    01      Cough   2021-03-04     3 Y
```

Common options to solve the issue:

- Restricting the source records by specifying/updating the `filter` argument in the `event_source()`/`censor_source()` calls.
- Specifying additional variables for `order` in the `event_source()`/`censor_source()` calls.
- Setting `check_type = "none"` in the `derive_param_tte()` call to ignore any duplicates.

In this example it does not have significant impact which record is chosen as the dates are the same so the time to event derivation will be the same, but it does impact `SRCSEQ` in the output dataset, so here the second option is used. Note here how you can also define source objects from within the `derive_param_tte()` function call itself.

```
derive_param_tte(
  dataset_adsl = adsl,
  by_vars = exprs(AEDECOD),
  start_date = TRTSDT,
  source_datasets = list(adsl = adsl, adae = adae_dup),
  event_conditions = list(event_source(
    dataset_name = "adae",
    date = ASTDT,
    set_values_to = exprs(
      EVNTDESC = "AE",
      SRCDOM = "ADAE",
      SRCVAR = "ASTDT",
      SRCSEQ = AESEQ
    ),
    order = exprs(AESEQ)
```

```
  )),
  censor_conditions = list(eos),
  set_values_to = exprs(
    PARAMCD = paste0("TTAE", as.numeric(as.factor(AEDECOD))),
    PARAM = paste("Time to First", AEDECOD, "Adverse Event")
  )
) %>%
  select(USUBJID, STARTDT, PARAMCD, PARAM, ADT, CNSR, SRCSEQ)
#> # A tibble: 4 × 7
#>   USUBJID STARTDT    PARAMCD PARAM                     ADT        CNSR SRCSEQ
#>   <chr>   <date>     <chr>   <chr>                     <date>    <int>  <dbl>
#> 1 01    2020-12-06 TTAE1  Time to First Cough Advers. . . 2021-03-04    0     2
#> 2 01    2020-12-06 TTAE2  Time to First Flu Adverse . . . 2021-01-03    0     1
#> 3 02    2021-01-16 TTAE1  Time to First Cough Advers. . . 2021-02-03    1    NA
#> 4 02    2021-01-16 TTAE2  Time to First Flu Adverse . . . 2021-02-03    1    NA
```

Filtering source records (`filter`)**:**

The first option from above could have been achieved using `filter`, for example here only using serious adverse events.

```
derive_param_tte(
  dataset_adsl = adsl,
  by_vars = exprs(AEDECOD),
  start_date = TRTSDT,
  source_datasets = list(adsl = adsl, adae = adae_dup),
  event_conditions = list(event_source(
    dataset_name = "adae",
    filter = AESER == "Y",
    date = ASTDT,
    set_values_to = exprs(
      EVNTDESC = "Serious AE",
      SRCDOM = "ADAE",
      SRCVAR = "ASTDT",
      SRCSEQ = AESEQ
    )
  )),
  censor_conditions = list(eos),
  set_values_to = exprs(
    PARAMCD = paste0("TTSAE", as.numeric(as.factor(AEDECOD))),
    PARAM = paste("Time to First Serious", AEDECOD, "Adverse Event")
  )
) %>%
  select(USUBJID, STARTDT, PARAMCD, PARAM, ADT, CNSR, SRCSEQ)
#> # A tibble: 4 × 7
#>   USUBJID STARTDT    PARAMCD PARAM                     ADT        CNSR SRCSEQ
#>   <chr>   <date>     <chr>   <chr>                     <date>    <int>  <dbl>
#> 1 01    2020-12-06 TTSAE1  Time to First Serious Coug. . . 2021-03-04    0     3
#> 2 01    2020-12-06 TTSAE2  Time to First Serious Flu . . . 2021-01-03    0     1
#> 3 02    2021-01-16 TTSAE1  Time to First Serious Coug. . . 2021-02-03    1    NA
```

```
#> 4 02    2021-01-16 TTSAE2  Time to First Serious Flu . . . 2021-02-03    1    NA
```

Using multiple event/censor conditions (`event_conditions` **/**`censor_conditions`**):**
In the above examples, we only have a single event and single censor condition. Here, we now consider multiple conditions for each passed using `event_conditions` and `censor_conditions`. For the event we are going to use first AE and additionally check a lab condition, and for the censor we'll add in treatment start date in case end of study date was ever missing.

```
adlb <- tribble(
  ~USUBJID, ~ADT,               ~PARAMCD, ~ANRIND,
  "01",     ymd("2020-12-22"), "HGB",    "LOW"
) %>%
  mutate(STUDYID = "AB42")

low_hgb <- event_source(
  dataset_name = "adlb",
  filter = PARAMCD == "HGB" & ANRIND == "LOW",
  date = ADT,
  set_values_to = exprs(
    EVNTDESC = "POSSIBLE ANEMIA",
    SRCDOM = "ADLB",
    SRCVAR = "ADT"
  )
)

trt_start <- censor_source(
  dataset_name = "adsl",
  date = TRTSDT,
  set_values_to = exprs(
    EVNTDESC = "TREATMENT START",
    SRCDOM = "ADSL",
    SRCVAR = "TRTSDT"
  )
)

derive_param_tte(
  dataset_adsl = adsl,
  event_conditions = list(ttae, low_hgb),
  censor_conditions = list(eos, trt_start),
  source_datasets = list(adsl = adsl, adae = adae, adlb = adlb),
  set_values_to = exprs(
    PARAMCD = "TTAELB",
    PARAM = "Time to First Adverse Event or Possible Anemia (Labs)"
  )
) %>%
  select(USUBJID, STARTDT, PARAMCD, PARAM, ADT, CNSR, SRCSEQ)
#> # A tibble: 2 × 7
#>   USUBJID STARTDT    PARAMCD PARAM                    ADT        CNSR SRCSEQ
#>   <chr>   <date>     <chr>   <chr>                    <date>    <int>  <dbl>
```

```
#> 1 01    2020-12-06 TTAELB  Time to First Adverse Even. . . 2020-12-22   0    NA
#> 2 02    2021-01-16 TTAELB  Time to First Adverse Even. . . 2021-02-03   1    NA
```

Note above how the earliest event date is always taken and the latest censor date.

End of the observation period (`end_dates`)**:**

The `end_dates` argument can be used to specify the end of the observation period if there is more than one date which restricts the observation period, e.g., end of study date and intercurrent events like new drug date. The earliest date is used as the end of the observation period and events/censorings occurring after this date are not considered.

In the example two `censor_source()` objects are defined, `eos` and `newdrg`, for end of study date and new drug date, respectively, and then passed to the `end_dates` argument.

```
adsl <- tribble(
  ~USUBJID, ~TRTSDT,           ~EOSDT,            ~NEWDRGDT,
  "01",     ymd("2020-12-06"), ymd("2021-03-06"), NA,
  "02",     ymd("2021-01-16"), ymd("2021-04-03"), ymd("2021-03-21"),
  "03",     ymd("2021-02-01"), NA,                NA,
  "04",     ymd("2021-03-10"), NA,                NA
) %>%
  mutate(STUDYID = "AB42")

adqs <- tribble(
  ~USUBJID, ~ADT,              ~CHG,
  "01",     ymd("2021-01-03"),    5,
  "01",     ymd("2021-02-03"),   -2,
  "01",     ymd("2021-03-01"),   NA,
  "01",     ymd("2021-03-07"),   10,
  "02",     ymd("2021-01-03"),    4,
  "02",     ymd("2021-02-03"),   -1,
  "02",     ymd("2021-04-01"),  -12,
  "03",     ymd("2021-02-15"),    3,
  "03",     ymd("2021-03-15"),  -15
) %>%
  mutate(STUDYID = "AB42")

eos <- censor_source(
  dataset_name = "adsl",
  date = EOSDT
)

newdrg <- censor_source(
  dataset_name = "adsl",
  date = NEWDRGDT
)

# Note to user: The source function has changed.
worsening <- event_source(
  dataset_name = "adqs",
```

```
  date = ADT,
  filter = CHG <= -10
)

valid_assessment <- censor_source(
  dataset_name = "adqs",
  date = ADT,
  filter = !is.na(CHG)
)

no_assessment <- censor_source(
  dataset_name = "adsl",
  date = TRTSDT
)

derive_param_tte(
  dataset_adsl = adsl,
  source_datasets = list(adsl = adsl, adqs = adqs),
  start_date = TRTSDT,
  end_dates = list(eos, newdrg),
  event_conditions = list(worsening),
  censor_conditions = list(valid_assessment, no_assessment),
  set_values_to = exprs(PARAMCD = "TTWORSE")
) %>%
  select(-STUDYID, -PARAMCD) %>%
  derive_vars_merged(
    dataset_add = adsl,
    by_vars = exprs(USUBJID),
    new_vars = exprs(EOSDT, NEWDRGDT)
  )
#> # A tibble: 4 × 6
#>   USUBJID ADT         CNSR STARTDT    EOSDT      NEWDRGDT
#>   <chr>   <date>     <int> <date>     <date>     <date>
#> 1 01      2021-02-03     1 2020-12-06 2021-03-06 NA
#> 2 02      2021-02-03     1 2021-01-16 2021-04-03 2021-03-21
#> 3 03      2021-03-15     0 2021-02-01 NA         NA
#> 4 04      2021-03-10     1 2021-03-10 NA         NA
```

Please note that:

- subject `02` has no event because the assessment with `CHG = -12` was excluded as it is after the start of a new drug.
- subject `01` and `02` are censored at the last valid assessment before the end of the observation period.

Positive event (`event_type`):

If positive events like response or improvement are analyzed, `event_type = "positive"` should be used. Subjects without events are censored at the end of the observation period (defined by `end_dates`) instead of the last assessment. For positive events this is the more conservative approach.

```
adsl <- tribble(
  ~USUBJID, ~TRTSDT,           ~EOSDT,             ~NEWDRGDT,
  "01",     ymd("2020-12-06"), ymd("2021-03-06"), NA,
  "02",     ymd("2021-01-16"), ymd("2021-04-03"), ymd("2021-03-21"),
  "03",     ymd("2021-02-01"), NA,                NA
) %>%
  mutate(STUDYID = "AB42")

adqs <- tribble(
  ~USUBJID, ~ADT,              ~CHG,
  "01",     ymd("2021-01-03"),    5,
  "01",     ymd("2021-02-03"),   -2,
  "01",     ymd("2021-03-01"),   NA,
  "01",     ymd("2021-03-07"),   10,
  "02",     ymd("2021-01-03"),    4,
  "02",     ymd("2021-02-03"),   -1,
  "02",     ymd("2021-04-01"),  -12,
  "03",     ymd("2021-02-15"),    3,
  "03",     ymd("2021-03-15"),   15
) %>%
  mutate(STUDYID = "AB42")

eos <- censor_source(
  dataset_name = "adsl",
  date = EOSDT
)

newdrg <- censor_source(
  dataset_name = "adsl",
  date = NEWDRGDT
)

improvement <- event_source(
  dataset_name = "adqs",
  date = ADT,
  filter = CHG >= 10
)

valid_assessment <- censor_source(
  dataset_name = "adqs",
  date = ADT,
  filter = !is.na(CHG)
)

derive_param_tte(
  dataset_adsl = adsl,
  source_datasets = list(adsl = adsl, adqs = adqs),
  start_date = TRTSDT,
```

```
  end_dates = list(eos, newdrg),
  event_conditions = list(improvement),
  censor_conditions = list(valid_assessment),
  event_type = "positive",
  set_values_to = exprs(PARAMCD = "TTIMPROV")
) %>%
  select(-STUDYID) %>%
  derive_vars_merged(
    dataset_add = adsl,
    by_vars = exprs(USUBJID),
    new_vars = exprs(EOSDT, NEWDRGDT)
  )
#> # A tibble: 3 × 7
#>   USUBJID ADT         CNSR STARTDT    PARAMCD  EOSDT      NEWDRGDT
#>   <chr>   <date>     <int> <date>     <chr>    <date>     <date>
#> 1 01      2021-03-06     1 2020-12-06 TTIMPROV 2021-03-06 NA
#> 2 02      2021-03-21     1 2021-01-16 TTIMPROV 2021-04-03 2021-03-21
#> 3 03      2021-03-15     0 2021-02-01 TTIMPROV NA         NA
```

Please note that subject `01` and `02` are censored at the end of the observation period instead of at the last assessment.

Differentiating censoring reasons:

There are three ADaM variables which allow to differentiate censoring reasons:

- `CNSR`: different values `>1` can be used to differentiate censoring reasons, e.g., `1` for end of study, `2` for new drug, etc.
- `EVNTDESC`: description of the event or censoring, e.g., `"END OF STUDY"`.
- `CNSDTDSC`: description of the date used for censoring when different from the censoring event, e.g., `"LAST ASSESSMENT"` if the censoring event is the end of the study but the censoring date is not the end of study date but the last assessment date before the end of the study.

In the example five censoring events are considered:

- end of study (`eos`, `no_worsening`)
- start of a new drug (`newdrg`, `no_worsening`)
- no post-baseline assessments (`no_post_baseline`)
- no baseline assessment (`no_baseline`)
- no assessments (`no_assessments`)

In the example data, the first five subjects have the five censoring events, respectively, and the sixth subject has an event.

```
adsl <- tribble(
~USUBJID, ~TRTSDT,           ~EOSDT,            ~NEWDRGDT,
"01",     ymd("2020-12-06"), ymd("2021-03-06"), NA,
"02",     ymd("2021-01-16"), ymd("2021-04-03"), ymd("2021-03-21"),
"03",     ymd("2021-03-10"), NA,                NA,
"04",     ymd("2021-04-02"), NA,                NA,
"05",     ymd("2021-05-09"), NA,                NA,
"06",     ymd("2021-02-01"), NA,                NA
```

```
) %>%
  mutate(STUDYID = "AB42")

adqs <- tribble(
  ~USUBJID, ~ADT,              ~CHG, ~ABLFL,
  "01",     ymd("2021-12-06"),    0, "Y",
  "01",     ymd("2021-02-03"),   -2, NA,
  "01",     ymd("2021-03-01"),   NA, NA,
  "01",     ymd("2021-03-07"),   10, NA,
  "02",     ymd("2021-01-16"),    0, "Y",
  "02",     ymd("2021-02-03"),   -1, NA,
  "02",     ymd("2021-04-01"),  -12, NA,
  "03",     ymd("2021-03-20"),   NA, NA,
  "03",     ymd("2021-04-07"),   NA, NA,
  "04",     ymd("2021-04-02"),    0, "Y",
  "06",     ymd("2021-02-01"),    0, "Y",
  "06",     ymd("2021-03-15"),  -15, NA
) %>%
  mutate(STUDYID = "AB42") %>%
  derive_vars_merged(
    dataset_add = adsl,
    by_vars = exprs(USUBJID),
    new_vars = exprs(TRTSDT)
  )
```

The eos and newdrg censoring events define the end dates.

```
eos <- censor_source(
  dataset_name = "adsl",
  date = EOSDT,
  censor = 1,
  set_values_to = exprs(
    EVNTDESC = "END OF STUDY"
  )
)

newdrg <- censor_source(
  dataset_name = "adsl",
  date = NEWDRGDT,
  censor = 2,
  set_values_to = exprs(
    EVNTDESC = "NEW DRUG"
  )
)
```

The worsening and valid_assessment events define which assessments are events and which are valid assessments, respectively. The EVNTDESC variable is not set by valid_assessment as its value is retrieved from the eos or newdrg censoring events.

```
worsening <- event_source(
```

```
  dataset_name = "adqs",
  date = ADT,
  filter = CHG <= -10,
  set_values_to = exprs(
    EVNTDESC = "WORSENING",
    SRCDOM = "ADQS",
    SRCVAR = "ADT"
  )
)

valid_assessment <- censor_source(
  dataset_name = "adqs",
  date = ADT,
  filter = !is.na(CHG),
  set_values_to = exprs(
    CNSDTDSC = "LAST ASSESSMENT",
    SRCDOM = "ADQS",
    SRCVAR = "ADT"
  )
)
```

The no_baseline, no_post_baseline, and no_assessment censoring events are used to censor subjects without valid assessments at day one (TRTSDT). Three events are defined to distinguish the reasons for not having valid assessments. The consider_end_dates = FALSE argument is used for these censoring events to avoid that the ENVTDESC and CNSR value from the end date (eos and newdrg) is used.

```
no_baseline <- censor_source(
  dataset_name = "adqs",
  date = TRTSDT,
  censor = 3,
  filter = is.na(ABLFL),
  order = exprs(ADT),
  consider_end_dates = FALSE,
  set_values_to = exprs(
    EVNTDESC = "NO BASELINE ASSESSMENT",
    CNSDTDSC = "TREATMENT START",
    SRCDOM = "ADQS",
    SRCVAR = "TRTSDT"
  )
)

no_post_baseline <- censor_source(
  dataset_name = "adqs",
  date = TRTSDT,
  censor = 4,
  filter = ABLFL == "Y",
  order = exprs(ADT),
  consider_end_dates = FALSE,
```

```
  set_values_to = exprs(
    EVNTDESC = "NO POST-BASELINE ASSESSMENT",
    CNSDTDSC = "TREATMENT START",
    SRCDOM = "ADQS",
    SRCVAR = "TRTSDT"
  )
)

no_assessment <- censor_source(
  dataset_name = "adsl",
  date = TRTSDT,
  censor = 5,
  consider_end_dates = FALSE,
  set_values_to = exprs(
    EVNTDESC = "NO ASSESSMENTS",
    CNSDTDSC = "TREATMENT START",
    SRCDOM = "ADSL",
    SRCVAR = "TRTSDT"
  )
)
```

In the derive_param_tte() function call, the order of the censoring events in the censor_conditions argument is important. For censoring events the records with the last date is selected. If there are multiple records with the same last date, then the last record in the order specified in the censor_conditions argument is selected. The events no_assessment, no_post_baseline, and no_baseline all use TRTSDT as date, i.e., the date is the same for them. Specifying no_assessment before no_post_baseline ensures that if a subject has no post-baseline assessments the record from no_post_baseline is used, i.e., EVNTDESC is set to "NO POST-BASELINE ASSESSMENT".

```
derive_param_tte(
  dataset_adsl = adsl,
  source_datasets = list(adsl = adsl, adqs = adqs),
  start_date = TRTSDT,
  end_dates = list(eos, newdrg),
  event_conditions = list(worsening),
 censor_conditions = list(valid_assessment, no_assessment, no_post_baseline, no_baseline),
  set_values_to = exprs(PARAMCD = "TTWORSE")
) %>%
  select(-STUDYID, -PARAMCD)
#> # A tibble: 6 × 8
#>   USUBJID ADT        EVNTDESC            SRCDOM SRCVAR  CNSR CNSDTDSC STARTDT
#>   <chr>   <date>     <chr>               <chr>  <chr>  <int> <chr>    <date>
#> 1 01     2021-02-03 END OF STUDY       ADQS   ADT        1 LAST AS. . . 2020-12-06
#> 2 02     2021-02-03 NEW DRUG           ADQS   ADT        2 LAST AS. . . 2021-01-16
#> 3 03     2021-03-10 NO BASELINE ASSESS. . . ADQS   TRTSDT     3 TREATME. . . 2021-03-10
#> 4 04     2021-04-02 NO POST-BASELINE A. . . ADQS   TRTSDT     4 TREATME. . . 2021-04-02
#> 5 05     2021-05-09 NO ASSESSMENTS     ADSL   TRTSDT     5 TREATME. . . 2021-05-09
#> 6 06      2021-03-15 WORSENING           ADQS   ADT        0 <NA>     2021-02-01
```

Differentiating censoring date description:

In this example, the event description (EVNTDESC) and the censoring date description (CNSDTDSC) should distinguish between the different censoring reasons: end of study, new drug, or just last assessment. If a variable like CNSDTDSC is set in both the end dates and the censoring condition, the latter overwrites the former. However, the `coalesce()` function can be used to avoid this.

```
adsl <- tribble(
~USUBJID, ~TRTSDT,           ~EOSDT,            ~NEWDRGDT,
"01",     ymd("2020-12-06"), ymd("2021-03-06"), NA,
"02",     ymd("2021-01-16"), ymd("2021-04-03"), ymd("2021-03-21"),
"03",     ymd("2021-03-10"), NA,                NA,
"04",     ymd("2021-04-02"), NA,                NA,
"05",     ymd("2021-05-09"), ymd("2021-07-30"), NA
) %>%
  mutate(STUDYID = "AB42")

adqs <- tribble(
  ~USUBJID, ~ADT,              ~CHG,
  "01",     ymd("2021-02-03"),   -2,
  "01",     ymd("2021-03-01"),   NA,
  "01",     ymd("2021-03-07"),   10,
  "02",     ymd("2021-02-03"),   -1,
  "02",     ymd("2021-04-01"),  -12,
  "03",     ymd("2021-03-20"),    2,
  "03",     ymd("2021-04-07"),    5,
  "04",     ymd("2021-04-15"),  -15,
  "05",     ymd("2021-06-01"),  -13
) %>%
  mutate(STUDYID = "AB42") %>%
  derive_vars_merged(
    dataset_add = adsl,
    by_vars = exprs(USUBJID),
    new_vars = exprs(TRTSDT)
  )
```

Here two end dates are defined which set both EVNTDESC and CNSDTDSC.

```
eos <- censor_source(
  dataset_name = "adsl",
  date = EOSDT,
  set_values_to = exprs(
    EVNTDESC = "END OF STUDY",
    CNSDTDSC = "LAST QA BEFORE EOS"
  )
)

newdrg <- censor_source(
  dataset_name = "adsl",
  date = NEWDRGDT,
  set_values_to = exprs(
    EVNTDESC = "NEW DRUG",
```

```
    CNSDTDSC = "LAST QA BEFORE NEW DRUG"
  )
)
```

For the censoring condition (`valid_assessment`) the `coalesce()` function is used to set the descriptions only if they are not already set by the end dates.

```
valid_assessment <- censor_source(
  dataset_name = "adqs",
  date = ADT,
  filter = !is.na(CHG),
  set_values_to = exprs(
    EVNTDESC = coalesce(EVNTDESC, "NO WORSENING"),
    CNSDTDSC = coalesce(CNSDTDSC, "LAST QA"),
    SRCDOM = "ADQS",
    SRCVAR = "ADT"
  )
)

worsening <- event_source(
  dataset_name = "adqs",
  date = ADT,
  filter = CHG <= -10,
  set_values_to = exprs(
    EVNTDESC = "WORSENING",
    SRCDOM = "ADQS",
    SRCVAR = "ADT"
  )
)

derive_param_tte(
  dataset_adsl = adsl,
  source_datasets = list(adsl = adsl, adqs = adqs),
  start_date = TRTSDT,
  end_dates = list(eos, newdrg),
  event_conditions = list(worsening),
  censor_conditions = list(valid_assessment),
  set_values_to = exprs(PARAMCD = "TTWORSE")
) %>%
  select(-STUDYID, -PARAMCD, -STARTDT, -SRCDOM, -SRCVAR)
#> # A tibble: 5 × 5
#>   USUBJID ADT        EVNTDESC      CNSR CNSDTDSC
#>   <chr>   <date>     <chr>        <int> <chr>
#> 1 01      2021-02-03 END OF STUDY     1 LAST QA BEFORE EOS
#> 2 02      2021-02-03 NEW DRUG         1 LAST QA BEFORE NEW DRUG
#> 3 03      2021-04-07 NO WORSENING     1 LAST QA
#> 4 04      2021-04-15 WORSENING        0 <NA>
#> 5 05      2021-06-01 WORSENING        0 <NA>
```

For subjects `01` and `02` the descriptions from the end dates are used. As subject `03` has no end

dates, the descriptions from the censoring condition (`valid_assessments`) are used.

Overall survival time to event parameter:
In oncology trials, this is commonly derived as time from randomization date to death. For those without event, they are censored at the last date they are known to be alive.

- The start date is set using `start_date` argument, now that we need to use different to the default.
- In this example, datetime was needed, which can be achieved by setting `create_datetime` argument to `TRUE`.

```
adsl <- tribble(
 ~USUBJID, ~RANDDTM,                ~LSALVDTM,               ~DTHDTM,                ~DTHFL,
 "01",    ymd_hms("2020-10-03 00:00:00"), ymd_hms("2022-12-15 23:59:59"), NA,                    NA,
 "02",    ymd_hms("2021-01-23 00:00:00"), ymd_hms("2021-02-03 19:45:59"), ymd_hms("2021-02-03 19:45:
) %>%
  mutate(STUDYID = "AB42")

# derive overall survival parameter
death <- event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDTM,
  set_values_to = exprs(
    EVNTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDTM"
  )
)

last_alive <- censor_source(
  dataset_name = "adsl",
  date = LSALVDTM,
  set_values_to = exprs(
    EVNTDESC = "LAST DATE KNOWN ALIVE",
    SRCDOM = "ADSL",
    SRCVAR = "LSALVDTM"
  )
)

derive_param_tte(
  dataset_adsl = adsl,
  start_date = RANDDTM,
  event_conditions = list(death),
  censor_conditions = list(last_alive),
  create_datetime = TRUE,
  source_datasets = list(adsl = adsl),
  set_values_to = exprs(
    PARAMCD = "OS",
    PARAM = "Overall Survival"
```

```
  )
) %>%
  select(USUBJID, STARTDTM, PARAMCD, PARAM, ADTM, CNSR)
#> # A tibble: 2 × 6
#>   USUBJID STARTDTM           PARAMCD PARAM           ADTM                CNSR
#>   <chr>   <dttm>             <chr>   <chr>           <dttm>             <int>
#> 1 01      2020-10-03 00:00:00 OS      Overall Survival 2022-12-15 23:59:59    1
#> 2 02      2021-01-23 00:00:00 OS      Overall Survival 2021-02-03 19:45:59    0
```

Duration of response time to event parameter:

In oncology trials, this is commonly derived as time from response until progression or death, or if neither have occurred then censor at last tumor assessment visit date. It is only relevant for subjects with a response. Note how only observations for subjects in `dataset_adsl` have the new parameter created, so see below how this is filtered only on responders.

```
adsl_resp <- tribble(
  ~USUBJID, ~DTHFL, ~DTHDT,             ~RSPDT,
  "01",     "Y",    ymd("2021-06-12"), ymd("2021-03-04"),
  "02",     "N",    NA,                NA,
  "03",     "Y",    ymd("2021-08-21"), NA,
  "04",     "N",    NA,                ymd("2021-04-14")
) %>%
  mutate(STUDYID = "AB42")

adrs <- tribble(
  ~USUBJID, ~AVALC, ~ADT,              ~ASEQ,
  "01",     "SD",   ymd("2021-01-03"), 1,
  "01",     "PR",   ymd("2021-03-04"), 2,
  "01",     "PD",   ymd("2021-05-05"), 3,
  "02",     "PD",   ymd("2021-02-03"), 1,
  "04",     "SD",   ymd("2021-02-13"), 1,
  "04",     "PR",   ymd("2021-04-14"), 2,
  "04",     "CR",   ymd("2021-05-15"), 3
) %>%
  mutate(STUDYID = "AB42", PARAMCD = "OVR")

pd <- event_source(
  dataset_name = "adrs",
  filter = AVALC == "PD",
  date = ADT,
  set_values_to = exprs(
    EVENTDESC = "PD",
    SRCDOM = "ADRS",
    SRCVAR = "ADTM",
    SRCSEQ = ASEQ
  )
)

death <- event_source(
```

```
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,
  set_values_to = exprs(
    EVENTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDT"
  )
)

last_visit <- censor_source(
  dataset_name = "adrs",
  date = ADT,
  set_values_to = exprs(
    EVENTDESC = "LAST TUMOR ASSESSMENT",
    SRCDOM = "ADRS",
    SRCVAR = "ADTM",
    SRCSEQ = ASEQ
  )
)

derive_param_tte(
  dataset_adsl = filter(adsl_resp, !is.na(RSPDT)),
  start_date = RSPDT,
  event_conditions = list(pd, death),
  censor_conditions = list(last_visit),
  source_datasets = list(adsl = adsl_resp, adrs = adrs),
  set_values_to = exprs(
    PARAMCD = "DURRSP",
    PARAM = "Duration of Response"
  )
) %>%
  select(USUBJID, STARTDT, PARAMCD, PARAM, ADT, CNSR, SRCSEQ)
#> # A tibble: 2 × 7
#>   USUBJID STARTDT    PARAMCD PARAM                ADT         CNSR SRCSEQ
#>   <chr>   <date>     <chr>   <chr>                <date>     <int>  <dbl>
#> 1 01      2021-03-04 DURRSP  Duration of Response 2021-05-05     0      3
#> 2 04      2021-04-14 DURRSP  Duration of Response 2021-05-15     1      3
```

Further examples:

Further example usages of this function can be found in the `vignette("bds_tte")` and `vignette("tte_analyses")`.

See Also

`event_source()`, `censor_source()`

derive_param_wbc_abs *Add a parameter for lab differentials converted to absolute values*

Description

Add a parameter by converting lab differentials from fraction or percentage to absolute values

Usage

```
derive_param_wbc_abs(
  dataset,
  by_vars,
  set_values_to,
  get_unit_expr,
  wbc_unit = "10^9/L",
  wbc_code = "WBC",
  diff_code,
  diff_type = "fraction"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. PARAMCD, and AVAL are expected as well. The variable specified by by_vars and PARAMCD must be a unique key of the input dataset, and to the parameters specified by wbc_code and diff_code. **Default value** none
by_vars	Grouping variables **Default value** none
set_values_to	Variables to set A named list returned by exprs() defining the variables to be set for the new parameter, e.g. exprs(PARAMCD = "LYMPH", PARAM = "Lymphocytes Abs (10^9/L)") is expected. **Default value** none
get_unit_expr	An expression providing the unit of the parameter The result is used to check the units of the input parameters. **Permitted values** a variable containing unit from the input dataset, or a function call, for example, get_unit_expr = extract_unit(PARAM). **Default value** none
wbc_unit	A string containing the required unit of the WBC parameter **Default value** "10^9/L"

wbc_code — White Blood Cell (WBC) parameter

The observations where PARAMCD equals the specified value are considered as the WBC absolute results to use for converting the differentials.

Permitted values character value

Default value "WBC"

diff_code — white blood differential parameter

The observations where PARAMCD equals the specified value are considered as the white blood differential lab results in fraction or percentage value to be converted into absolute value.

Default value none

diff_type — A string specifying the type of differential

Permitted values "percent", "fraction"

Default value "fraction"

Details

If diff_type is "percent", the analysis value of the new parameter is derived as

$$\frac{WhiteBloodCellCount * PercentageValue}{100}$$

If diff_type is "fraction", the analysis value of the new parameter is derived as

$$WhiteBloodCellCount * FractionValue$$

New records are created for each group of records (grouped by by_vars) if 1) the white blood cell component in absolute value is not already available from the input dataset, and 2) the white blood cell absolute value (identified by wbc_code) and the white blood cell differential (identified by diff_code) are both present.

Value

The input dataset with the new parameter added

See Also

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_summary_records()

Examples

```
library(tibble)

test_lb <- tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~PARAM, ~VISIT,
```

```
  "P01", "WBC", 33, "Leukocyte Count (10^9/L)", "CYCLE 1 DAY 1",
  "P01", "WBC", 38, "Leukocyte Count (10^9/L)", "CYCLE 2 DAY 1",
  "P01", "LYMLE", 0.90, "Lymphocytes (fraction of 1)", "CYCLE 1 DAY 1",
  "P01", "LYMLE", 0.70, "Lymphocytes (fraction of 1)", "CYCLE 2 DAY 1",
  "P01", "ALB", 36, "Albumin (g/dL)", "CYCLE 2 DAY 1",
  "P02", "WBC", 33, "Leukocyte Count (10^9/L)", "CYCLE 1 DAY 1",
  "P02", "LYMPH", 29, "Lymphocytes Abs (10^9/L)", "CYCLE 1 DAY 1",
  "P02", "LYMLE", 0.87, "Lymphocytes (fraction of 1)", "CYCLE 1 DAY 1",
  "P03", "LYMLE", 0.89, "Lymphocytes (fraction of 1)", "CYCLE 1 DAY 1"
)

derive_param_wbc_abs(
  dataset = test_lb,
  by_vars = exprs(USUBJID, VISIT),
  set_values_to = exprs(
    PARAMCD = "LYMPH",
    PARAM = "Lymphocytes Abs (10^9/L)",
    DTYPE = "CALCULATION"
  ),
  get_unit_expr = extract_unit(PARAM),
  wbc_code = "WBC",
  diff_code = "LYMLE",
  diff_type = "fraction"
)
```

`derive_summary_records` *Add New Records Within By Groups Using Aggregation Functions*

Description

It is not uncommon to have an analysis need whereby one needs to derive an analysis value (`AVAL`) from multiple records. The ADaM basic dataset structure variable `DTYPE` is available to indicate when a new derived records has been added to a dataset, if the derivation deviates from the standard derivation of the parameter.

Usage

```
derive_summary_records(
  dataset = NULL,
  dataset_add,
  dataset_ref = NULL,
  by_vars,
  filter_add = NULL,
  constant_values = NULL,
  set_values_to,
  missing_values = NULL
)
```

Arguments

`dataset` Input dataset

If the argument is not specified (or set to `NULL`), a new dataset is created. Otherwise, the new records are appended to the specified dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value `NULL`

`dataset_add` Additional dataset

The variables specified for `by_vars` are expected. Observations from the specified dataset are going to be used to calculate and added as new records to the input dataset (`dataset`).

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`dataset_ref` Reference dataset

The variables specified for `by_vars` are expected. For each observation of the specified dataset a new observation is added to the input dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value `NULL`

`by_vars` Grouping variables

Variables to consider for generation of groupwise summary records. Providing the names of variables in `exprs()` will create a groupwise summary and generate summary records for the specified groups.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`filter_add` Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within `by_vars` as this will help when an aggregating, lagging, or ranking function is involved.

For example,

- `filter_add = (AVAL > mean(AVAL, na.rm = TRUE))` will filter all `AVAL` values greater than mean of `AVAL` with in `by_vars`.
- `filter_add = (dplyr::n() > 2)` will filter n count of `by_vars` greater than 2.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`constant_values` Constant variables to set

The specified variables are set to the specified values for all new summary records, including those with data in `dataset_add` and those with no data imputed using `dataset_ref` and `missing_values`.

Set a list of variables to some specified value for the new records

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be an expression.

Permitted values list of named expressions created by exprs(), e.g., exprs(CUMDOSA = sum(AVAL, na.rm = TRUE), AVALU = "ml")

Default value NULL

set_values_to Variables to be set

The specified variables are set to the specified values for the new observations.

Set a list of variables to some specified value for the new records

- LHS refers to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for by_vars. Any expression on the RHS must result in a single value per by group.

For example:

```
set_values_to = exprs(
  AVAL = sum(AVAL),
  DTYPE = "AVERAGE",
)
```

Permitted values list of named expressions created by exprs(), e.g., exprs(CUMDOSA = sum(AVAL, na.rm = TRUE), AVALU = "ml")

Default value none

missing_values Values for missing records

For observations of the reference dataset (dataset_ref) which do not have a matching record in dataset_add (with respect to by_vars and after applying filter_add), the specified variables are set to the specified values for the new observations.

Permitted values list of named expressions created by exprs(), e.g., exprs(CUMDOSA = sum(AVAL, na.rm = TRUE), AVALU = "ml")

Default value NULL

Details

For the newly derived records, only variables specified within by_vars or set_values_to will be populated. All other variables will be set to NA.

Value

A data frame with derived records appended to original dataset.

Examples

Data setup:

The following examples use the ECG dataset below as a basis.

```
library(tibble, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)
```

```
adeg <- tribble(
  ~USUBJID,   ~PARAM,             ~AVISIT,    ~EGDTC,             ~AVAL,
  "XYZ-1001", "QTcF Int. (msec)", "Baseline", "2016-02-24T07:50", 385,
  "XYZ-1001", "QTcF Int. (msec)", "Baseline", "2016-02-24T07:52", 399,
  "XYZ-1001", "QTcF Int. (msec)", "Baseline", "2016-02-24T07:56", 396,
  "XYZ-1001", "QTcF Int. (msec)", "Visit 2",  "2016-03-08T09:48", 393,
  "XYZ-1001", "QTcF Int. (msec)", "Visit 2",  "2016-03-08T09:51", 388,
  "XYZ-1001", "QTcF Int. (msec)", "Visit 3",  "2016-03-22T10:48", 394,
  "XYZ-1001", "QTcF Int. (msec)", "Visit 3",  "2016-03-22T10:51", 402,
  "XYZ-1002", "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 399,
  "XYZ-1002", "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 200,
  "XYZ-1002", "QTcF Int. (msec)", "Baseline", "2016-02-22T08:01", 392,
  "XYZ-1002", "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:53", 414,
  "XYZ-1002", "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:56", 402
) %>%
  mutate(ADTM = convert_dtc_to_dtm(EGDTC))
```

Summarize one or more variables using summary functions:

A derived record is generated for each subject, containing the mean of the triplicate ECG interval values (AVAL) and the latest measurement's time (ADTM) by using summary functions within the set_values_to argument.

```
derive_summary_records(
  adeg,
  dataset_add = adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    ADTM = max(ADTM),
    DTYPE = "AVERAGE"
  )
) %>%
  arrange(USUBJID, AVISIT)
#> # A tibble: 17 × 7
#>    USUBJID  PARAM           AVISIT   EGDTC       AVAL ADTM                DTYPE
#>    <chr>    <chr>           <chr>    <chr>      <dbl> <dttm>              <chr>
#>  1 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  385  2016-02-24 07:50:00 <NA>
#>  2 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-24 07:52:00 <NA>
#>  3 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  396  2016-02-24 07:56:00 <NA>
#>  4 XYZ-1001 QTcF Int. (msec) Baseline <NA>      393. 2016-02-24 07:56:00 AVER. . .
#>  5 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  393  2016-03-08 09:48:00 <NA>
#>  6 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  388  2016-03-08 09:51:00 <NA>
#>  7 XYZ-1001 QTcF Int. (msec) Visit 2  <NA>      390. 2016-03-08 09:51:00 AVER. . .
#>  8 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  394  2016-03-22 10:48:00 <NA>
#>  9 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-22 10:51:00 <NA>
#> 10 XYZ-1001 QTcF Int. (msec) Visit 3  <NA>      398  2016-03-22 10:51:00 AVER. . .
#> 11 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-22 07:58:00 <NA>
#> 12 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  200  2016-02-22 07:58:00 <NA>
#> 13 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  392  2016-02-22 08:01:00 <NA>
```

```
#> 14 XYZ-1002 QTcF Int. (msec) Baseline <NA>      330. 2016-02-22 08:01:00 AVER. . .
#> 15 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  414  2016-03-24 10:53:00 <NA>
#> 16 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-24 10:56:00 <NA>
#> 17 XYZ-1002 QTcF Int. (msec) Visit 3  <NA>      408  2016-03-24 10:56:00 AVER. . .
```

Functions such as `all()` and `any()` are also often useful when creating summary records. For instance, the above example can be extended to flag which derived records were affected by outliers. Note that the outlier flag is created before `AVAL` is set for the summary record. Otherwise, referencing `AVAL` later on would pick up the `AVAL` from the summary record rather than the source records.

```
derive_summary_records(
  adeg,
  dataset_add = adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    OUTLIEFL = if_else(any(AVAL >= 500 | AVAL <= 300), "Y", "N"),
    AVAL = mean(AVAL, na.rm = TRUE),
    ADTM = max(ADTM),
    DTYPE = "AVERAGE"
  )
) %>%
  arrange(USUBJID, AVISIT)
#> # A tibble: 17 × 8
#>    USUBJID  PARAM         AVISIT EGDTC  AVAL ADTM              OUTLIEFL DTYPE
#>    <chr>    <chr>         <chr>  <chr> <dbl> <dttm>            <chr>    <chr>
#>  1 XYZ-1001 QTcF Int. (ms. . . Basel. . . 2016. . .  385  2016-02-24 07:50:00 <NA>    <NA>
#>  2 XYZ-1001 QTcF Int. (ms. . . Basel. . . 2016. . .  399  2016-02-24 07:52:00 <NA>    <NA>
#>  3 XYZ-1001 QTcF Int. (ms. . . Basel. . . 2016. . .  396  2016-02-24 07:56:00 <NA>    <NA>
#>  4 XYZ-1001 QTcF Int. (ms. . . Basel. . . <NA>  393. 2016-02-24 07:56:00 N      AVER. . .
#>  5 XYZ-1001 QTcF Int. (ms. . . Visit. . . 2016. . .  393  2016-03-08 09:48:00 <NA>    <NA>
#>  6 XYZ-1001 QTcF Int. (ms. . . Visit. . . 2016. . .  388  2016-03-08 09:51:00 <NA>    <NA>
#>  7 XYZ-1001 QTcF Int. (ms. . . Visit. . . <NA>  390. 2016-03-08 09:51:00 N      AVER. . .
#>  8 XYZ-1001 QTcF Int. (ms. . . Visit. . . 2016. . .  394  2016-03-22 10:48:00 <NA>    <NA>
#>  9 XYZ-1001 QTcF Int. (ms. . . Visit. . . 2016. . .  402  2016-03-22 10:51:00 <NA>    <NA>
#> 10 XYZ-1001 QTcF Int. (ms. . . Visit. . . <NA>  398  2016-03-22 10:51:00 N      AVER. . .
#> 11 XYZ-1002 QTcF Int. (ms. . . Basel. . . 2016. . .  399  2016-02-22 07:58:00 <NA>    <NA>
#> 12 XYZ-1002 QTcF Int. (ms. . . Basel. . . 2016. . .  200  2016-02-22 07:58:00 <NA>    <NA>
#> 13 XYZ-1002 QTcF Int. (ms. . . Basel. . . 2016. . .  392  2016-02-22 08:01:00 <NA>    <NA>
#> 14 XYZ-1002 QTcF Int. (ms. . . Basel. . . <NA>  330. 2016-02-22 08:01:00 Y      AVER. . .
#> 15 XYZ-1002 QTcF Int. (ms. . . Visit. . . 2016. . .  414  2016-03-24 10:53:00 <NA>    <NA>
#> 16 XYZ-1002 QTcF Int. (ms. . . Visit. . . 2016. . .  402  2016-03-24 10:56:00 <NA>    <NA>
#> 17 XYZ-1002 QTcF Int. (ms. . . Visit. . . <NA>  408  2016-03-24 10:56:00 N      AVER. . .
```

Restricting source records (`filter_add`):

The `filter_add` argument can be used to restrict the records that are being summarized. For instance, the mean of the triplicates above can be computed only for the baseline records by passing `filter_add = AVISIT == "Baseline"`.

```
derive_summary_records(
```

```
  adeg,
  dataset_add = adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  filter_add = AVISIT == "Baseline",
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    DTYPE = "AVERAGE"
  )
) %>%
  arrange(USUBJID, AVISIT)
#> # A tibble: 14 × 7
#>    USUBJID  PARAM           AVISIT   EGDTC      AVAL ADTM                DTYPE
#>    <chr>    <chr>           <chr>    <chr>     <dbl> <dttm>              <chr>
#>  1 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  385  2016-02-24 07:50:00 <NA>
#>  2 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-24 07:52:00 <NA>
#>  3 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  396  2016-02-24 07:56:00 <NA>
#>  4 XYZ-1001 QTcF Int. (msec) Baseline <NA>      393. NA              AVER. . .
#>  5 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  393  2016-03-08 09:48:00 <NA>
#>  6 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  388  2016-03-08 09:51:00 <NA>
#>  7 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  394  2016-03-22 10:48:00 <NA>
#>  8 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-22 10:51:00 <NA>
#>  9 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-22 07:58:00 <NA>
#> 10 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  200  2016-02-22 07:58:00 <NA>
#> 11 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  392  2016-02-22 08:01:00 <NA>
#> 12 XYZ-1002 QTcF Int. (msec) Baseline <NA>      330. NA              AVER. . .
#> 13 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  414  2016-03-24 10:53:00 <NA>
#> 14 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-24 10:56:00 <NA>
```

Summary functions can also be used within `filter_add` to filter based on conditions applied to the whole of the by group specified in `by_vars`. For instance, the mean of the triplicates can be computed only for by groups which do indeed contain three records by passing `filter_add = n() > 2`.

```
derive_summary_records(
  adeg,
  dataset_add = adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  filter_add = n() > 2,
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    DTYPE = "AVERAGE"
  )
) %>%
  arrange(USUBJID, AVISIT)
#> # A tibble: 14 × 7
#>    USUBJID  PARAM           AVISIT   EGDTC      AVAL ADTM                DTYPE
#>    <chr>    <chr>           <chr>    <chr>     <dbl> <dttm>              <chr>
#>  1 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  385  2016-02-24 07:50:00 <NA>
#>  2 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-24 07:52:00 <NA>
```

```
#>  3 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  396  2016-02-24 07:56:00 <NA>
#>  4 XYZ-1001 QTcF Int. (msec) Baseline <NA>      393. NA              AVER. . .
#>  5 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  393  2016-03-08 09:48:00 <NA>
#>  6 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  388  2016-03-08 09:51:00 <NA>
#>  7 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  394  2016-03-22 10:48:00 <NA>
#>  8 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-22 10:51:00 <NA>
#>  9 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-22 07:58:00 <NA>
#> 10 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  200  2016-02-22 07:58:00 <NA>
#> 11 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  392  2016-02-22 08:01:00 <NA>
#> 12 XYZ-1002 QTcF Int. (msec) Baseline <NA>      330. NA              AVER. . .
#> 13 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  414  2016-03-24 10:53:00 <NA>
#> 14 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-24 10:56:00 <NA>
```

Adding records for groups not in source (`dataset_ref` **and** `missing_values`)**:**

Adding records for groups which are not in the source data can be achieved by specifying a reference dataset in the `dataset_ref` argument. For example, specifying the input dataset `adeg_allparamvis` (containing an extra `"Visit 2"` for patient `1002`) ensures a summary record is derived for that visit as well. For these records, the values of the analysis variables to be populated should be specified within the `missing_values` argument. Here, `DTYPE = "PHANTOM"` was chosen as `AVAL` is set to missing.

```
adeg_allparamvis <- tribble(
  ~USUBJID,   ~PARAM,             ~AVISIT,
  "XYZ-1001", "QTcF Int. (msec)", "Baseline",
  "XYZ-1001", "QTcF Int. (msec)", "Visit 2",
  "XYZ-1001", "QTcF Int. (msec)", "Visit 3",
  "XYZ-1002", "QTcF Int. (msec)", "Baseline",
  "XYZ-1002", "QTcF Int. (msec)", "Visit 2",
  "XYZ-1002", "QTcF Int. (msec)", "Visit 3"
)

derive_summary_records(
  adeg,
  dataset_add = adeg,
  dataset_ref = adeg_allparamvis,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    ADTM = max(ADTM),
    DTYPE = "AVERAGE"
  ),
  missing_values = exprs(
    AVAL = NA,
    ADTM = NA,
    DTYPE = "PHANTOM"
  )
) %>%
  arrange(USUBJID, AVISIT)
#> # A tibble: 18 × 7
```

```
#>    USUBJID  PARAM           AVISIT   EGDTC       AVAL ADTM                DTYPE
#>    <chr>    <chr>           <chr>    <chr>      <dbl> <dttm>              <chr>
#>  1 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  385  2016-02-24 07:50:00 <NA>
#>  2 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-24 07:52:00 <NA>
#>  3 XYZ-1001 QTcF Int. (msec) Baseline 2016-02-2. . .  396  2016-02-24 07:56:00 <NA>
#>  4 XYZ-1001 QTcF Int. (msec) Baseline <NA>      393. 2016-02-24 07:56:00 AVER. . .
#>  5 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  393  2016-03-08 09:48:00 <NA>
#>  6 XYZ-1001 QTcF Int. (msec) Visit 2  2016-03-0. . .  388  2016-03-08 09:51:00 <NA>
#>  7 XYZ-1001 QTcF Int. (msec) Visit 2  <NA>      390. 2016-03-08 09:51:00 AVER. . .
#>  8 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  394  2016-03-22 10:48:00 <NA>
#>  9 XYZ-1001 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-22 10:51:00 <NA>
#> 10 XYZ-1001 QTcF Int. (msec) Visit 3  <NA>      398  2016-03-22 10:51:00 AVER. . .
#> 11 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  399  2016-02-22 07:58:00 <NA>
#> 12 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  200  2016-02-22 07:58:00 <NA>
#> 13 XYZ-1002 QTcF Int. (msec) Baseline 2016-02-2. . .  392  2016-02-22 08:01:00 <NA>
#> 14 XYZ-1002 QTcF Int. (msec) Baseline <NA>      330. 2016-02-22 08:01:00 AVER. . .
#> 15 XYZ-1002 QTcF Int. (msec) Visit 2  <NA>       NA  NA              PHAN. . .
#> 16 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  414  2016-03-24 10:53:00 <NA>
#> 17 XYZ-1002 QTcF Int. (msec) Visit 3  2016-03-2. . .  402  2016-03-24 10:56:00 <NA>
#> 18 XYZ-1002 QTcF Int. (msec) Visit 3  <NA>      408  2016-03-24 10:56:00 AVER. . .
```

Filtering out missing summary records:

It may be helpful to print out missing records to determine missingness of data. Using the example above, the data can be filtered to include `DTYPE = "PHANTOM"`.

```
adeg_allparamvis <- tribble(
  ~USUBJID,   ~PARAM,             ~AVISIT,
  "XYZ-1001", "QTcF Int. (msec)", "Baseline",
  "XYZ-1001", "QTcF Int. (msec)", "Visit 2",
  "XYZ-1001", "QTcF Int. (msec)", "Visit 3",
  "XYZ-1002", "QTcF Int. (msec)", "Baseline",
  "XYZ-1002", "QTcF Int. (msec)", "Visit 2",
  "XYZ-1002", "QTcF Int. (msec)", "Visit 3"
)

derive_summary_records(
  adeg,
  dataset_add = adeg,
  dataset_ref = adeg_allparamvis,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    ADTM = max(ADTM),
    DTYPE = "AVERAGE"
  ),
  missing_values = exprs(
    AVAL = NA,
    ADTM = NA,
    DTYPE = "PHANTOM"
```

```
  )
) %>%
  arrange(USUBJID, AVISIT) %>%
  filter(DTYPE == "PHANTOM")
#> # A tibble: 1 × 7
#>   USUBJID  PARAM            AVISIT  EGDTC  AVAL ADTM   DTYPE
#>   <chr>    <chr>            <chr>   <chr> <dbl> <dttm> <chr>
#> 1 XYZ-1002 QTcF Int. (msec) Visit 2 <NA>     NA NA     PHANTOM
```

Add constant values to derived and missing summary records:

The `constant_values` argument allows you to assign fixed, common values to all summary records generated by the function. This is particularly useful when you need to populate new information for observations derived from the `dataset_add` as well as for new records created for subjects present in `dataset_ref` but missing in `dataset_add`.

For example, if `ADSL` contains two subjects ("1" and "2"), but `ADAE` only has adverse event information for "Subject 1", `derive_summary_records` will:

1. Create a summary record for "Subject 1" based on `ADAE`.
2. Identify "Subject 2" (from `ADSL`) as having no corresponding records in `ADAE` and create a new record for it.

The `constant_values` argument ensures that all these generated summary records (for both Subject 1 and the newly created record for Subject 2) receive the same `PARAMCD`, `PARAM`, and `PARCAT1` values. Additionally, `missing_values` is used to specifically set `AVAL` to 0 for "Subject 2" when no adverse events are found.

```
library(tibble)

adsl <- tibble(USUBJID = c("1", "2"))

adae <- tribble(
 ~USUBJID, ~AEDECOD,
 "1",      "Illness",
 "1",      "Pain"
)

derive_summary_records(
  dataset_add = adae,
  dataset_ref = adsl,
  by_vars = exprs(USUBJID),
  constant_values = exprs(
    PARAMCD = "AECOUNT",
    PARAM = "Number of adverse events",
    PARCAT1 = "Adverse events"
  ),
  set_values_to = exprs(
    AVAL = n_distinct(AEDECOD),
    SRCDOM = "ADAE"
  ),
  missing_values = exprs(
```

```
    AVAL = 0
  )
)
#> # A tibble: 2 × 6
#>   USUBJID  AVAL SRCDOM PARAMCD PARAM                    PARCAT1
#>   <chr>   <dbl> <chr>  <chr>   <chr>                    <chr>
#> 1 1           2 ADAE   AECOUNT Number of adverse events Adverse events
#> 2 2           0 <NA>   AECOUNT Number of adverse events Adverse events
```

See Also

derive_vars_merged_summary()

BDS-Findings Functions for adding Parameters/Records: default_qtc_paramcd(), derive_basetype_records(), derive_expected_records(), derive_extreme_event(), derive_extreme_records(), derive_locf_records(), derive_param_bmi(), derive_param_bsa(), derive_param_computed(), derive_param_doseint(), derive_param_exist_flag(), derive_param_exposure(), derive_param_framingham(), derive_param_map(), derive_param_qtc(), derive_param_rr(), derive_param_wbc_abs()

---

derive_vars_aage *Derive Analysis Age*

Description

Derives analysis age (AAGE) and analysis age unit (AAGEU).

Note: This is a wrapper function for the more generic derive_vars_duration().

Usage

```
derive_vars_aage(
  dataset,
  start_date = BRTHDT,
  end_date = RANDDT,
  age_unit = "YEARS",
  type = "interval"
)
```

Arguments

dataset	Input dataset The variables specified by the start_date and end_date arguments are expected to be in the dataset. **Default value** none
start_date	The start date A date or date-time object is expected. Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

Default value BRTHDT

end_date — The end date

A date or date-time object is expected.

Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

Default value RANDDT

age_unit — Age unit

The age is derived in the specified unit

Permitted values The values are considered case-insensitive.
For years: "year", "years", "yr", "yrs", "y"
For months: "month", "months", "mo", "mos"
For weeks: "week", "weeks", "wk", "wks", "w"
For days: "day", "days", "d"
For hours: "hour", "hours", "hr", "hrs", "h"
For minutes: "minute", "minutes", "min", "mins"
For seconds: "second", "seconds", "sec", "secs", "s"

Default value "YEARS"

type — lubridate duration type

See below for details.

Default: "interval"

Permitted Values: "duration", "interval"

Default value "interval"

Details

The duration is derived as time from start to end date in the specified output unit. If the end date is before the start date, the duration is negative. The start and end date variable must be present in the specified input dataset.

The lubridate package calculates two types of spans between two dates: duration and interval. While these calculations are largely the same, when the unit of the time period is month or year the result can be slightly different.

The difference arises from the ambiguity in the length of "1 month" or "1 year". Months may have 31, 30, 28, or 29 days, and years are 365 days and 366 during leap years. Durations and intervals help solve the ambiguity in these measures.

The **interval** between 2000-02-01 and 2000-03-01 is 1 (i.e. one month). The **duration** between these two dates is 0.95, which accounts for the fact that the year 2000 is a leap year, February has 29 days, and the average month length is 30.4375, i.e. 29 / 30.4375 = 0.95.

For additional details, review the lubridate time span reference page.

Value

The input dataset with AAGE and AAGEU added

See Also

derive_vars_duration()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_vars_extreme_event(), derive_vars_period()

Examples

```
library(tibble)
library(lubridate)

data <- tribble(
  ~BRTHDT, ~RANDDT,
  ymd("1984-09-06"), ymd("2020-02-24")
)

derive_vars_aage(data)
```

---

derive_vars_atc *Derive ATC Class Variables*

Description

Add Anatomical Therapeutic Chemical class variables from FACM to ADCM.

Note: This is a wrapper function for the more generic derive_vars_transposed().

Usage

```
derive_vars_atc(
  dataset,
  dataset_facm,
  by_vars = exprs(!!!get_admiral_option("subject_keys"), CMREFID = FAREFID),
  id_vars = NULL,
  value_var = FASTRESC
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. **Default value** none
dataset_facm	FACM dataset The variables specified by the by_vars, id_vars, and value_var arguments and FATESTCD are required. The variables by_vars, id_vars, and FATESTCD must be a unique key. **Default value** none

by_vars — Grouping variables

Keys used to merge dataset_facm with dataset.

Default value exprs(!!!get_admiral_option("subject_keys"), CMREFID = FAREFID)

id_vars — ID variables

Variables (excluding by_vars) that uniquely identify each observation in dataset_merge.

Default value NULL

value_var — The variable of dataset_facm containing the values of the transposed variables

Default value FASTRESC

Value

The input dataset with ATC variables added

See Also

derive_vars_transposed()

OCCDS Functions: derive_var_trtemfl(), derive_vars_query()

Examples

```
library(tibble)

cm <- tribble(
  ~STUDYID,  ~USUBJID,       ~CMGRPID, ~CMREFID,  ~CMDECOD,
  "STUDY01", "BP40257-1001", "14",     "1192056", "PARACETAMOL",
  "STUDY01", "BP40257-1001", "18",     "2007001", "SOLUMEDROL",
  "STUDY01", "BP40257-1002", "19",     "2791596", "SPIRONOLACTONE"
)
facm <- tribble(
  ~STUDYID,  ~USUBJID,       ~FAGRPID, ~FAREFID,  ~FATESTCD,  ~FASTRESC,
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC1CD", "N",
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC2CD", "N02",
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC3CD", "N02B",
  "STUDY01", "BP40257-1001", "1",      "1192056", "CMATC4CD", "N02BE",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC1CD", "D",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC2CD", "D10",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC3CD", "D10A",
  "STUDY01", "BP40257-1001", "1",      "2007001", "CMATC4CD", "D10AA",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC1CD", "D",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC2CD", "D07",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC3CD", "D07A",
  "STUDY01", "BP40257-1001", "2",      "2007001", "CMATC4CD", "D07AA",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC1CD", "H",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC2CD", "H02",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC3CD", "H02A",
  "STUDY01", "BP40257-1001", "3",      "2007001", "CMATC4CD", "H02AB",
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC1CD", "C",
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC2CD", "C03",
```

```
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC3CD", "C03D",
  "STUDY01", "BP40257-1002", "1",      "2791596", "CMATC4CD", "C03DA"
)

derive_vars_atc(cm, facm, id_vars = exprs(FAGRPID))
```

`derive_vars_cat` *Derive Categorization Variables Like* `AVALCATy` *and* `AVALCAyN`

Description

Derive Categorization Variables Like `AVALCATy` and `AVALCAyN`

Usage

```
derive_vars_cat(dataset, definition, by_vars = NULL)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` and `definition` arguments are expected to be in the dataset.

Default value none

`definition` List of expressions created by `exprs()`. Must be in rectangular format and specified using the same syntax as when creating a `tibble` using the `tribble()` function. The `definition` object will be converted to a `tibble` using `tribble()` inside this function.

Must contain:

- the column `condition` which will be converted to a logical expression and will be used on the `dataset` input.
- at least one additional column with the new column name and the category value(s) used by the logical expression.
- the column specified in `by_vars` (if `by_vars` is specified)

e.g. if `by_vars` is not specified:

```
exprs(~condition,   ~AVALCAT1, ~AVALCA1N,
      AVAL >= 140, ">=140 cm",         1,
      AVAL < 140,   "<140 cm",         2)
```

e.g. if `by_vars` is specified as `exprs(VSTEST)`:

```
exprs(~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
      "Height", AVAL >= 140, ">=140 cm",         1,
      "Height",  AVAL < 140,  "<140 cm",         2)
```

Default value none

by_vars — list of expressions with one element. NULL by default. Allows for specifying by groups, e.g. exprs(PARAMCD). Variable must be present in both dataset and definition. The conditions in definition are applied only to those records that match by_vars. The categorization variables are set to NA for records not matching any of the by groups in definition.

Default value NULL

Details

If conditions are overlapping, the row order of definitions must be carefully considered. The **first** match will determine the category. i.e. if

AVAL = 155

and the definition is:

```
definition <- exprs(
  ~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
  "Height",  AVAL > 170,  ">170 cm",         1,
  "Height", AVAL <= 170, "<=170 cm",         2,
  "Height", AVAL <= 160, "<=160 cm",         3
)
```

then AVALCAT1 will be "<=170 cm", as this is the first match for AVAL. If you specify:

```
definition <- exprs(
  ~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
  "Height", AVAL <= 160, "<=160 cm",         3,
  "Height", AVAL <= 170, "<=170 cm",         2,
  "Height",  AVAL > 170,  ">170 cm",         1
)
```

Then AVAL <= 160 will lead to AVALCAT1 == "<=160 cm", AVAL in-between 160 and 170 will lead to AVALCAT1 == "<=170 cm", and AVAL <= 170 will lead to AVALCAT1 == ">170 cm".

However, we suggest to be more explicit when defining the condition, to avoid overlap. In this case, the middle condition should be: AVAL <= 170 & AVAL > 160

Value

The input dataset with the new variables defined in definition added

Examples

Data setup:

The following examples use the ADVS dataset below as a basis. It contains vital signs data with some missing values (NA) that will demonstrate how the function handles different scenarios.

```
library(dplyr)
library(tibble)

advs <- tibble::tribble(
  ~USUBJID,       ~VSTEST,  ~AVAL,
  "01-701-1015", "Height", 147.32,
  "01-701-1015", "Weight",  53.98,
  "01-701-1023", "Height", 162.56,
  "01-701-1023", "Weight",     NA,
  "01-701-1028", "Height",     NA,
  "01-701-1028", "Weight",     NA,
  "01-701-1033", "Height", 175.26,
  "01-701-1033", "Weight",  88.45
)
```

Derive categorization variables without `by_vars`**:**

In this example, we derive `AVALCAT1`, `AVALCA1N`, and `NEWCOL` without using `by_vars`. The conditions must include all necessary filtering logic, such as checking both `VSTEST` and `AVAL`. Records that don't match any condition will have `NA` values for the new variables.

```
definition <- exprs(
  ~condition,                         ~AVALCAT1, ~AVALCA1N,  ~NEWCOL,
  VSTEST == "Height" & AVAL > 160,   ">160 cm",         1, "extra1",
  VSTEST == "Height" & AVAL <= 160, "<=160 cm",         2, "extra2"
)

derive_vars_cat(
  dataset = advs,
  definition = definition
)
#> # A tibble: 8 × 6
#>   USUBJID     VSTEST  AVAL AVALCAT1 AVALCA1N NEWCOL
#>   <chr>       <chr>  <dbl> <chr>       <dbl> <chr>
#> 1 01-701-1015 Height 147.  <=160 cm        2 extra2
#> 2 01-701-1015 Weight  54.0 <NA>           NA <NA>
#> 3 01-701-1023 Height 163.  >160 cm         1 extra1
#> 4 01-701-1023 Weight  NA   <NA>           NA <NA>
#> 5 01-701-1028 Height  NA   <NA>           NA <NA>
#> 6 01-701-1028 Weight  NA   <NA>           NA <NA>
#> 7 01-701-1033 Height 175.  >160 cm         1 extra1
#> 8 01-701-1033 Weight  88.4 <NA>           NA <NA>
```

Derive categorization variables using `by_vars`**:**

When using `by_vars`, the conditions are automatically scoped to records matching each by group value. This simplifies the condition logic as you don't need to include the by variable in each condition. Here we derive categories for both Height and Weight measurements using `by_vars = exprs(VSTEST)`.

```
definition2 <- exprs(
```

```
  ~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
  "Height",  AVAL > 160,  ">160 cm",         1,
  "Height", AVAL <= 160, "<=160 cm",         2,
  "Weight",   AVAL > 70,   ">70 kg",         1,
  "Weight",  AVAL <= 70,  "<=70 kg",         2
)

derive_vars_cat(
  dataset = advs,
  definition = definition2,
  by_vars = exprs(VSTEST)
)
#> # A tibble: 8 × 5
#>   USUBJID     VSTEST  AVAL AVALCAT1 AVALCA1N
#>   <chr>       <chr>  <dbl> <chr>       <dbl>
#> 1 01-701-1015 Height 147.  <=160 cm        2
#> 2 01-701-1015 Weight  54.0 <=70 kg         2
#> 3 01-701-1023 Height 163.  >160 cm         1
#> 4 01-701-1023 Weight  NA   <NA>           NA
#> 5 01-701-1028 Height  NA   <NA>           NA
#> 6 01-701-1028 Weight  NA   <NA>           NA
#> 7 01-701-1033 Height 175.  >160 cm         1
#> 8 01-701-1033 Weight  88.4 >70 kg          1
```

Using multiple conditions with explicit ranges:

When you need more than two categories, you can define multiple conditions. It's best practice to make conditions mutually exclusive using explicit range definitions (e.g., `AVAL <= 170 & AVAL > 160`) to avoid ambiguity, even though the function uses first-match logic.

```
definition3 <- exprs(
  ~VSTEST,                ~condition,  ~AVALCAT1, ~AVALCA1N,
  "Height",                AVAL > 170,  ">170 cm",         1,
  "Height", AVAL <= 170 & AVAL > 160, "<=170 cm",         2,
  "Height",               AVAL <= 160, "<=160 cm",         3
)

derive_vars_cat(
  dataset = advs,
  definition = definition3,
  by_vars = exprs(VSTEST)
)
#> # A tibble: 8 × 5
#>   USUBJID     VSTEST  AVAL AVALCAT1 AVALCA1N
#>   <chr>       <chr>  <dbl> <chr>       <dbl>
#> 1 01-701-1015 Height 147.  <=160 cm        3
#> 2 01-701-1015 Weight  54.0 <NA>           NA
#> 3 01-701-1023 Height 163.  <=170 cm        2
#> 4 01-701-1023 Weight  NA   <NA>           NA
#> 5 01-701-1028 Height  NA   <NA>           NA
```

```
#> 6 01-701-1028 Weight  NA   <NA>           NA
#> 7 01-701-1033 Height 175.  >170 cm         1
#> 8 01-701-1033 Weight  88.4 <NA>           NA
```

Deriving categories based on reference ranges (MCRITy variables):

This example demonstrates deriving laboratory measurement criteria variables (MCRITyML and MCRITyMN). The conditions use reference range variables (like ANRHI) to create categories relative to normal ranges, which is common in laboratory data analysis.

```
adlb <- tibble::tribble(
  ~USUBJID,     ~PARAM, ~AVAL, ~AVALU,  ~ANRHI,
  "01-701-1015", "ALT",   150,  "U/L",      40,
  "01-701-1023", "ALT",    70,  "U/L",      40,
  "01-701-1036", "ALT",   130,  "U/L",      40,
  "01-701-1048", "ALT",    30,  "U/L",      40,
  "01-701-1015", "AST",    50,  "U/L",      35
)

definition_mcrit <- exprs(
  ~PARAM,                      ~condition,    ~MCRIT1ML, ~MCRIT1MN,
  "ALT",                    AVAL <= ANRHI,    "<=ANRHI",         1,
  "ALT", ANRHI < AVAL & AVAL <= 3 * ANRHI, ">1-3*ANRHI",         2,
  "ALT",                 3 * ANRHI < AVAL,   ">3*ANRHI",         3
)

adlb %>%
  derive_vars_cat(
    definition = definition_mcrit,
    by_vars = exprs(PARAM)
  )
#> # A tibble: 5 × 7
#>   USUBJID     PARAM  AVAL AVALU ANRHI MCRIT1ML   MCRIT1MN
#>   <chr>       <chr> <dbl> <chr> <dbl> <chr>          <dbl>
#> 1 01-701-1015 ALT     150 U/L      40 >3*ANRHI           3
#> 2 01-701-1023 ALT      70 U/L      40 >1-3*ANRHI         2
#> 3 01-701-1036 ALT     130 U/L      40 >3*ANRHI           3
#> 4 01-701-1048 ALT      30 U/L      40 <=ANRHI            1
#> 5 01-701-1015 AST      50 U/L      35 <NA>              NA
```

Handling missing values and partial by groups:

When using by_vars, records that don't match any by group in the definition will have NA for all derived variables. In this example, records with VSTEST == "Weight" will have NA values because only "Height" conditions are defined. Additionally, records with missing AVAL will result in NA for the categorization variables since conditions cannot be evaluated.

```
definition4 <- exprs(
  ~VSTEST,   ~condition,  ~AVALCAT1, ~AVALCA1N,
  "Height",  AVAL > 160,  ">160 cm",         1,
  "Height", AVAL <= 160, "<=160 cm",         2
```

```
)

derive_vars_cat(
  dataset = advs,
  definition = definition4,
  by_vars = exprs(VSTEST)
)
#> # A tibble: 8 × 5
#>   USUBJID     VSTEST  AVAL AVALCAT1 AVALCA1N
#>   <chr>       <chr>  <dbl> <chr>       <dbl>
#> 1 01-701-1015 Height 147.  <=160 cm        2
#> 2 01-701-1015 Weight  54.0 <NA>           NA
#> 3 01-701-1023 Height 163.  >160 cm         1
#> 4 01-701-1023 Weight  NA   <NA>           NA
#> 5 01-701-1028 Height  NA   <NA>           NA
#> 6 01-701-1028 Weight  NA   <NA>           NA
#> 7 01-701-1033 Height 175.  >160 cm         1
#> 8 01-701-1033 Weight  88.4 <NA>           NA
```

Deriving multiple categorization variables simultaneously:

You can derive any number of categorization variables in a single call. This example creates three different categorization schemes (AVALCAT1, AVALCAT2, and AVALCAT3) with their corresponding numeric flags, all from the same set of conditions.

```
definition5 <- exprs(
 ~VSTEST,  ~condition,  ~AVALCAT1, ~AVALCA1N, ~AVALCAT2,    ~AVALCA2N, ~AVALCAT3,
 "Height",  AVAL > 160,  ">160 cm",       1,   "Tall",           1,   "Group A",
 "Height", AVAL <= 160, "<=160 cm",       2,  "Short",           2,   "Group B",
 "Weight",   AVAL > 70,   ">70 kg",       3,  "Heavy",           3,   "Group C",
 "Weight",  AVAL <= 70,  "<=70 kg",       4,  "Light",           4,   "Group D"
)

derive_vars_cat(
  dataset = advs,
  definition = definition5,
  by_vars = exprs(VSTEST)
)
#> # A tibble: 8 × 8
#>   USUBJID     VSTEST  AVAL AVALCAT1 AVALCA1N AVALCAT2 AVALCA2N AVALCAT3
#>   <chr>       <chr>  <dbl> <chr>       <dbl> <chr>       <dbl> <chr>
#> 1 01-701-1015 Height 147.  <=160 cm        2 Short           2 Group B
#> 2 01-701-1015 Weight  54.0 <=70 kg         4 Light           4 Group D
#> 3 01-701-1023 Height 163.  >160 cm         1 Tall            1 Group A
#> 4 01-701-1023 Weight  NA   <NA>           NA <NA>           NA <NA>
#> 5 01-701-1028 Height  NA   <NA>           NA <NA>           NA <NA>
#> 6 01-701-1028 Weight  NA   <NA>           NA <NA>           NA <NA>
#> 7 01-701-1033 Height 175.  >160 cm         1 Tall            1 Group A
#> 8 01-701-1033 Weight  88.4 >70 kg          3 Heavy           3 Group C
```

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sum derive_vars_transposed()

---

derive_vars_computed *Adds Variable(s) Computed from the Analysis Value of one or more Parameters*

Description

Adds Variable(s) computed from the analysis value of one or more parameters. It is expected that the value of the new variable is defined by an expression using the analysis values of other parameters, such as addition/sum, subtraction/difference, multiplication/product, division/ratio, exponentiation/logarithm, or by formula.

For example Body Mass Index at Baseline (BMIBL) in ADSL can be derived from of HEIGHT and WEIGHT parameters in ADVS.

Usage

```
derive_vars_computed(
  dataset,
  dataset_add,
  by_vars,
  parameters,
  new_vars,
  filter_add = NULL,
  constant_by_vars = NULL,
  constant_parameters = NULL
)
```

Arguments

dataset	The variables specified by the by_vars parameter are expected. **Default value** none
dataset_add	Additional dataset The variables specified by the by_vars parameter are expected. The variable specified by by_vars and PARAMCD must be a unique key of the additional dataset after restricting it by the filter condition (filter_add parameter) and to the parameters specified by parameters. **Default value** none

by_vars

Grouping variables

Grouping variables uniquely identifying a set of records for which new_vars are to be calculated.

Permitted values list of variables created by exprs()

Default value none

parameters

Required parameter codes

It is expected that all parameter codes (PARAMCD) which are required to derive the new variable are specified for this parameter or the constant_parameters parameter.

If observations should be considered which do not have a parameter code, e.g., if an SDTM dataset is used, temporary parameter codes can be derived by specifying a list of expressions. The name of the element defines the temporary parameter code and the expression defines the condition for selecting the records. For example, parameters = exprs(HGHT = VSTESTCD == "HEIGHT") selects the observations with VSTESTCD == "HEIGHT" from the input data (dataset and dataset_add), sets PARAMCD = "HGHT" for these observations, and adds them to the observations to consider.

Unnamed elements in the list of expressions are considered as parameter codes. For example, parameters = exprs(WEIGHT, HGHT = VSTESTCD == "HEIGHT") uses the parameter code "WEIGHT" and creates a temporary parameter code "HGHT".

Permitted values A character vector of PARAMCD values or a list of expressions

Default value none

new_vars

Name of the newly created variables

The specified variables are set to the specified values. The values of variables of the parameters specified by parameters can be accessed using <variable name>.<parameter code>. For example

```
exprs(
  BMIBL = (AVAL.WEIGHT / (AVAL.HEIGHT/100)^2)
)
```

defines the value for the new variable.

Variable names in the expression must not contain more than one dot.

Permitted values List of variable-value pairs

Default value none

filter_add

Filter condition of additional dataset

The specified condition is applied to the additional dataset before deriving the new variable, i.e., only observations fulfilling the condition are taken into account.

Permitted values a condition

Default value NULL

constant_by_vars

By variables for constant parameters

The constant parameters (parameters that are measured only once) are merged to the other parameters using the specified variables. (Refer to the Example)

Permitted values list of variables

Default value NULL

constant_parameters

Required constant parameter codes

It is expected that all the parameter codes (PARAMCD) which are required to derive the new variable and are measured only once are specified here. For example if BMI should be derived and height is measured only once while weight is measured at each visit. Height could be specified in the constant_parameters parameter. (Refer to the Example)

If observations should be considered which do not have a parameter code, e.g., if an SDTM dataset is used, temporary parameter codes can be derived by specifying a list of expressions. The name of the element defines the temporary parameter code and the expression defines the condition for selecting the records. For example constant_parameters = exprs(HGHT = VSTESTCD == "HEIGHT") selects the observations with VSTESTCD == "HEIGHT" from the input data (dataset and dataset_add), sets PARAMCD = "HGHT" for these observations, and adds them to the observations to consider.

Unnamed elements in the list of expressions are considered as parameter codes. For example, constant_parameters = exprs(WEIGHT, HGHT = VSTESTCD == "HEIGHT") uses the parameter code "WEIGHT" and creates a temporary parameter code "HGHT".

Permitted values A character vector of PARAMCD values or a list of expressions

Default value NULL

Details

For each group (with respect to the variables specified for the by_vars argument), the values of the new variables (new_vars) are computed based on the parameters in the additional dataset (dataset_add) and then the new variables are merged to the input dataset (dataset).

Value

The input dataset with the new variables added.

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

Examples

```
library(tibble)
library(dplyr)

# Example 1: Derive BMIBL
adsl <- tribble(
```

```
  ~STUDYID,   ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01", "01-1302",   61,   "YEARS",
  "PILOT01", "17-1344",   64,   "YEARS"
)

advs <- tribble(
  ~STUDYID,  ~USUBJID,  ~PARAMCD, ~PARAM,        ~VISIT,      ~AVAL, ~AVALU, ~ABLFL,
  "PILOT01", "01-1302", "HEIGHT", "Height (cm)", "SCREENING", 177.8, "cm",   "Y",
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "SCREENING", 81.19, "kg",   NA,
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "BASELINE",   82.1, "kg",   "Y",
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "WEEK 2",    81.19, "kg",   NA,
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "WEEK 4",    82.56, "kg",   NA,
  "PILOT01", "01-1302", "WEIGHT", "Weight (kg)", "WEEK 6",    80.74, "kg",   NA,
  "PILOT01", "17-1344", "HEIGHT", "Height (cm)", "SCREENING", 163.5, "cm",   "Y",
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "SCREENING", 58.06, "kg",   NA,
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "BASELINE",  58.06, "kg",   "Y",
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "WEEK 2",    58.97, "kg",   NA,
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "WEEK 4",    57.97, "kg",   NA,
  "PILOT01", "17-1344", "WEIGHT", "Weight (kg)", "WEEK 6",    58.97, "kg",   NA
)

derive_vars_computed(
  dataset = adsl,
  dataset_add = advs,
  by_vars = exprs(STUDYID, USUBJID),
  parameters = c("WEIGHT", "HEIGHT"),
  new_vars = exprs(BMIBL = compute_bmi(height = AVAL.HEIGHT, weight = AVAL.WEIGHT)),
  filter_add = ABLFL == "Y"
)
```

---

derive_vars_crit_flag *Derive Criterion Flag Variables* CRITy*,* CRITyFL*, and* CRITyFN

Description

The function derives ADaM compliant criterion flags, e.g., to facilitate subgroup analyses.

If a criterion flag can't be derived with this function, the derivation is not ADaM compliant. It helps to ensure that:

- the condition of the criterion depends only on variables of the same row,
- the CRITyFL is populated with valid values, i.e, either "Y" and NA or "Y", "N", and NA,
- the CRITy variable is populated correctly, i.e.,
  - set to a constant value within a parameter if CRITyFL is populated with "Y", "N", and NA and
  - set to a constant value within a parameter if the criterion condition is fulfilled and to NA otherwise if CRITyFL is populated with "Y", and NA

Usage

```
derive_vars_crit_flag(
  dataset,
  crit_nr = 1,
  condition,
  description,
  values_yn = FALSE,
  create_numeric_flag = FALSE
)
```

Arguments

dataset	Input dataset **Permitted values** a dataset, i.e., a data.frame or tibble **Default value** none
crit_nr	The criterion number, i.e., the y in CRITy **Permitted values** a positive integer, e.g. 2 or 5 **Default value** 1
condition	Condition for flagging records See description of the values_yn argument for details on how the CRITyFL variable is populated. **Permitted values** an unquoted condition, e.g., AVISIT == "BASELINE" **Default value** none
description	The description of the criterion The CRITy variable is set to the specified value. An expression can be specified to set the value depending on the parameter. Please note that the value must be constant within a parameter. **Permitted values** an unquoted expression which evaluates to a character (in dataset) **Default value** none
values_yn	Should "Y" and "N" be used for CRITyFL? If set to TRUE, the CRITyFL variable is set to "Y" if the condition (condition) evaluates to TRUE, it is set to "N" if the condition evaluate to FALSE, and to NA if it evaluates to NA. Otherwise, the CRITyFL variable is set to "Y" if the condition (condition) evaluates to TRUE, and to NA otherwise. **Permitted values** TRUE, FALSE **Default value** FALSE
create_numeric_flag	Create a numeric flag? If set to TRUE, the CRITyFN variable is created. It is set to 1 if CRITyFL == "Y", it set to 0 if CRITyFL == "N", and to NA otherwise. **Permitted values** TRUE, FALSE **Default value** FALSE

Value

The input dataset with the variables CRITy, CRITyFL, and optionally CRITyFN added.

Examples

Data setup:

The following examples use the BDS dataset below as a basis.

```
library(tibble, warn.conflicts = FALSE)

adbds <- tribble(
  ~PARAMCD, ~AVAL,
  "AST",    42,
  "AST",    52,
  "AST",    NA_real_,
  "ALT",    33,
  "ALT",    51
)
```

Creating a simple criterion flag with values "Y" and NA (condition, description):

The following call is a simple application of derive_vars_crit_flag() to derive a criterion flag/variable pair in a BDS dataset.

- The new variables are named CRIT1/CRIT1FL because the argument crit_nr has not been passed.
- Since the argument values_yn has also not been passed and thus is set to its default of FALSE, CRIT1FL is set to Y only if condition evaluates to TRUE. For example, in both the first and third records, where condition is respectively FALSE and NA, we set CRIT1FL = NA_character_. The fourth record also exhibits this behavior. Also, as per CDISC standards, in this case CRIT1 is populated only for records where condition evaluates to TRUE.

```
derive_vars_crit_flag(
  adbds,
  condition = AVAL > 50,
  description = "Absolute value > 50"
)
#> # A tibble: 5 × 4
#>   PARAMCD  AVAL CRIT1FL CRIT1
#>   <chr>   <dbl> <chr>   <chr>
#> 1 AST        42 <NA>    <NA>
#> 2 AST        52 Y       Absolute value > 50
#> 3 AST        NA <NA>    <NA>
#> 4 ALT        33 <NA>    <NA>
#> 5 ALT        51 Y       Absolute value > 50
```

The description argument also accepts expressions which depend on other variables in the input dataset. This can be useful to dynamically populate CRITx, for instance in the case below where we improve the CRIT1 text because the same flag/variable pair is actually being used for multiple parameters.

```
derive_vars_crit_flag(
  adbds,
  condition = AVAL > 50,
  description = paste(PARAMCD, "> 50"),
)
#> # A tibble: 5 × 4
#>   PARAMCD  AVAL CRIT1FL CRIT1
#>   <chr>   <dbl> <chr>   <chr>
#> 1 AST        42 <NA>    <NA>
#> 2 AST        52 Y       AST > 50
#> 3 AST        NA <NA>    <NA>
#> 4 ALT        33 <NA>    <NA>
#> 5 ALT        51 Y       ALT > 50
```

Creating a criterion flag with values "Y", "N" and NA (values_yn):

The next call builds on the previous example by using value_yn = TRUE to distinguish between the cases where condition is FALSE and those where it is not evaluable at all.

- As compared to the previous example, for the first record condition evaluates to FALSE and so we set CRIT1FL = "N", whereas for the third record, condition evaluates to NA because AVAL is missing and so we set CRIT1FL to NA.
- Note also that because we are using the values "Y", "N" and NA for the flag, as per CDISC standards CRIT1 is now populated for all records rather than just for the "Y" records.

```
derive_vars_crit_flag(
  adbds,
  condition = AVAL > 50,
  description = paste(PARAMCD, "> 50"),
  values_yn = TRUE
)
#> # A tibble: 5 × 4
#>   PARAMCD  AVAL CRIT1FL CRIT1
#>   <chr>   <dbl> <chr>   <chr>
#> 1 AST        42 N       AST > 50
#> 2 AST        52 Y       AST > 50
#> 3 AST        NA <NA>    AST > 50
#> 4 ALT        33 N       ALT > 50
#> 5 ALT        51 Y       ALT > 50
```

If the user wishes to set the criterion flag to "N" whenever the condition is not fulfilled, condition can be updated using an if_else call, where the third argument determines the behavior when the condition is not evaluable.

```
derive_vars_crit_flag(
  adbds,
  condition = if_else(AVAL > 50, TRUE, FALSE, FALSE),
  description = paste(PARAMCD, "> 50"),
  values_yn = TRUE
)
#> # A tibble: 5 × 4
```

```
#>   PARAMCD  AVAL CRIT1FL CRIT1
#>   <chr>   <dbl> <chr>   <chr>
#> 1 AST        42 N       AST > 50
#> 2 AST        52 Y       AST > 50
#> 3 AST        NA N       AST > 50
#> 4 ALT        33 N       ALT > 50
#> 5 ALT        51 Y       ALT > 50
```

Specifying the criterion variable/flag number and creating a numeric flag (`crit_nr`, `create_numeric_flag`).:

The user can manually specify the criterion variable/flag number to use to name `CRITy`/`CRITyFL` by passing the `crit_nr` argument. This may be necessary if, for instance, other criterion flags already exist in the input dataset.

The user can also choose to create an additional, equivalent numeric flag `CRITyFN` by setting `create_numeric_flag` to `TRUE`.

```
derive_vars_crit_flag(
  adbds,
  condition = AVAL > 50,
  description = paste(PARAMCD, "> 50"),
  values_yn = TRUE,
  crit_nr = 2,
  create_numeric_flag = TRUE
)
#> # A tibble: 5 × 5
#>   PARAMCD  AVAL CRIT2FL CRIT2    CRIT2FN
#>   <chr>   <dbl> <chr>   <chr>      <int>
#> 1 AST        42 N       AST > 50       0
#> 2 AST        52 Y       AST > 50       1
#> 3 AST        NA <NA>    AST > 50      NA
#> 4 ALT        33 N       ALT > 50       0
#> 5 ALT        51 Y       ALT > 50       1
```

See Also

BDS-Findings Functions that returns variable appended to dataset: `derive_var_analysis_ratio()`, `derive_var_anrind()`, `derive_var_atoxgr()`, `derive_var_atoxgr_dir()`, `derive_var_base()`, `derive_var_chg()`, `derive_var_nfrlt()`, `derive_var_ontrtfl()`, `derive_var_pchg()`, `derive_var_shift()`

`derive_vars_dt` *Derive/Impute a Date from a Character Date*

Description

Derive a date (`*DT`) from a character date (`--DTC`). The date can be imputed (see `date_imputation` argument) and the date imputation flag (`*DTF`) can be added.

Usage

```
derive_vars_dt(
  dataset,
  new_vars_prefix,
  dtc,
  highest_imputation = "n",
  date_imputation = "first",
  flag_imputation = "auto",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

`dataset`

Input dataset

The variables specified by the `dtc` argument are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`new_vars_prefix`

Prefix used for the output variable(s).

A character scalar is expected. For the date variable (`*DT`) is appended to the specified prefix and for the date imputation flag (`*DTF`), i.e., for `new_vars_prefix = "AST"` the variables `ASTDT` and `ASTDTF` are created.

Permitted values a character scalar, i.e., a character vector of length one

Default value none

`dtc`

The `--DTC` date to impute

A character date is expected in a format like `yyyy-mm-dd` or `yyyy-mm-ddThh:mm:ss`. Trailing components can be omitted and `-` is a valid "missing" value for any component.

Permitted values a character date variable

Default value none

`highest_imputation`

Highest imputation level

The `highest_imputation` argument controls which components of the `--DTC` value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, `NA` is returned. For example, for `highest_imputation = "D" "2020"` results in `NA` because the month is missing.

If `"n"` (none, lowest level) is specified no imputation is performed, i.e., if any component is missing, `NA` is returned.

If `"Y"` (year, highest level) is specified, `date_imputation` must be `"first"` or `"last"` and `min_dates` or `max_dates` must be specified respectively. Otherwise, an error is thrown.

Permitted values `"Y"` (year, highest level), `"M"` (month), `"D"` (day), `"n"` (none, lowest level)

Default value `"n"`

`date_imputation`

The value to impute the day/month when a datepart is missing.

A character value is expected.

- The`"first"` and `"last"` keywords allow imputation to the first/last day/month. They can also be used to impute the year if used in conjunction with the `min_dates` or `max_dates` arguments. Some examples of this are available here.
- When `highest_imputation` is `"M"` or `"D"`, the `"mid"` keyword can also be specified to impute missing components to the middle of the possible range:
  - If both month and day are missing, they are imputed as `"06-30"` (middle of the year).
  - If only day is missing, it is imputed as `"15"` (middle of the month).
- `"<dd>"` can be specified only if `highest_imputation = "D"`. Missing days are imputed by the specified day, e.g. `"10"` for the 10th day of the month. The specified day should be valid for all months as otherwise an error might be issued. For example, `date_imputation = "30"` results in an invalid date of "2024-02-30" for the partial date "2024-02".
- `"<mm>-<dd>"` can be specified only if `highest_imputation` is `"M"`, e.g. `"06-15"` for the 15th of June.

Permitted values a key-word, i.e. `"first"`, `"mid"`, `"last"`, or `"<mm>-<dd>"` or `"<dd>"`

Default value `"first"`

`flag_imputation`

Whether the date imputation flag must also be derived.

If `"auto"` is specified and `highest_imputation` argument is not `"n"`, then date imputation flag is derived.

If `"date"` is specified, then date imputation flag is derived.

If `"none"` is specified, then no date imputation flag is derived.

Please note that CDISC requirements dictate the need for a date imputation flag if any imputation is performed, so `flag_imputation = "none"` should only be used if the imputed variable is not part of the final ADaM dataset.

Permitted values `"auto"`, `"date"` or `"none"`

Default value `"auto"`

`min_dates` Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the `dtc` value are considered. The possible dates are defined by the missing parts of the `dtc` date (see example below). This ensures that the non-missing parts of the `dtc` date are not changed. A date or date-time object is expected. For example

```
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
   ymd_hms("2020-12-06T12:12:12"),
   ymd_hms("2020-11-11T11:11:11")
  ),
  highest_imputation = "M"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

max_dates — Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

preserve — Preserve day if month is missing and day is present

For example "2019---07" would return "2019-06-07 if preserve = TRUE (and date_imputation = "MID").

Permitted values TRUE, FALSE

Default value FALSE

Details

In {admiral} we don't allow users to pick any single part of the date/time to impute, we only enable to impute up to a highest level, i.e. you couldn't choose to say impute months, but not days.

The presence of a *DTF variable is checked and if it already exists in the input dataset, a warning is issued and *DTF will be overwritten.

Additionally, the function will throw an error if imputation rules cause an invalid date (e.g. "2020-02-31") to be generated. In this case, the user should adjust the imputation rules.

Value

The input dataset with the date *DT (and the date imputation flag *DTF if requested) added.

Examples

Derive a date variable without imputation:

In this example, we derive ASTDT from MHSTDTC with no imputation done for partial dates.

```
library(tibble)
library(lubridate)

mhdt <- tribble(
  ~MHSTDTC,
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
)

derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC
)
#> # A tibble: 7 × 2
#>   MHSTDTC               ASTDT
#>   <chr>                 <date>
#> 1 "2019-07-18T15:25:40" 2019-07-18
#> 2 "2019-07-18T15:25"    2019-07-18
#> 3 "2019-07-18"          2019-07-18
#> 4 "2019-02"             NA
#> 5 "2019"                NA
#> 6 "2019---07"           NA
#> 7 ""                    NA
```

Impute partial dates (highest_imputation):

Imputation is requested by the highest_imputation argument. Here highest_imputation = "M" for month imputation is used, i.e. the highest imputation done on a partial date is up to the month. By default, missing date components are imputed to the first day/month/year. A date imputation flag variable, ASTDTF, is automatically created. The flag variable indicates if imputation was done on the date.

```
derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  date_imputation = "first"
)
#> # A tibble: 7 × 3
```

```
#>   MHSTDTC               ASTDT      ASTDTF
#>   <chr>                 <date>     <chr>
#> 1 "2019-07-18T15:25:40" 2019-07-18 <NA>
#> 2 "2019-07-18T15:25"    2019-07-18 <NA>
#> 3 "2019-07-18"          2019-07-18 <NA>
#> 4 "2019-02"             2019-02-01 D
#> 5 "2019"                2019-01-01 M
#> 6 "2019---07"           2019-01-01 M
#> 7 ""                    NA         <NA>
```

It is also possible to just impute the day, with `highest_imputation = "D"`. Here dates with just a missing day have it imputed to the 10th of the month. Note that in this case care needs to be taken to ensure invalid dates are not created, e.g. `date_imputation = "30"` would create an invalid date of `"2020-02-30"` when trying to impute the day for `"2020-02"`.

```
derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "D",
  date_imputation = "10"
)
#> # A tibble: 7 × 3
#>   MHSTDTC               ASTDT      ASTDTF
#>   <chr>                 <date>     <chr>
#> 1 "2019-07-18T15:25:40" 2019-07-18 <NA>
#> 2 "2019-07-18T15:25"    2019-07-18 <NA>
#> 3 "2019-07-18"          2019-07-18 <NA>
#> 4 "2019-02"             2019-02-10 D
#> 5 "2019"                NA         <NA>
#> 6 "2019---07"           NA         <NA>
#> 7 ""                    NA         <NA>
```

Impute to the last day/month (`date_imputation = "last"`):

In this example, we derive `ADT` impute partial dates to last day/month, i.e. `date_imputation = "last"`.

```
qsdt <- tribble(
  ~QSDTC,
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
)

derive_vars_dt(
  qsdt,
```

```
  new_vars_prefix = "A",
  dtc = QSDTC,
  highest_imputation = "M",
  date_imputation = "last"
)
#> # A tibble: 7 × 3
#>   QSDTC                 ADT        ADTF
#>   <chr>                 <date>     <chr>
#> 1 "2019-07-18T15:25:40" 2019-07-18 <NA>
#> 2 "2019-07-18T15:25"    2019-07-18 <NA>
#> 3 "2019-07-18"          2019-07-18 <NA>
#> 4 "2019-02"             2019-02-28 D
#> 5 "2019"                2019-12-31 M
#> 6 "2019---07"           2019-12-31 M
#> 7 ""                    NA         <NA>
```

Impute to the middle (`date_imputation = "mid"`) and suppress imputation flag (`flag_imputation = "none"`):

In this example, we will derive `TRTSDT` with date imputation flag (`*DTF`) suppressed. Since `date_imputation = "mid"`, partial date imputation will be set to June 30th for missing month and 15th for missing day only. The `flag_imputation = "none"` call ensures no date imputation flag is created. In practice, as per CDISC requirements this option can only be selected if the imputed variable is not part of the final ADaM dataset.

```
exdt <- tribble(
  ~EXSTDTC,
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
)
derive_vars_dt(
  exdt,
  new_vars_prefix = "TRTS",
  dtc = EXSTDTC,
  highest_imputation = "M",
  date_imputation = "mid",
  flag_imputation = "none"
)
#> # A tibble: 7 × 2
#>   EXSTDTC               TRTSDT
#>   <chr>                 <date>
#> 1 "2019-07-18T15:25:40" 2019-07-18
#> 2 "2019-07-18T15:25"    2019-07-18
#> 3 "2019-07-18"          2019-07-18
#> 4 "2019-02"             2019-02-15
```

```
#> 5 "2019"                2019-06-30
#> 6 "2019---07"           2019-06-30
#> 7 ""                    NA
```

Impute to a specific date (`date_imputation = "04-06"`):

In this example, we derive `ASTDT` with specific date imputation, i.e. `date_imputation = "04-06"`. Note that day portion, `"-06"`, is used in the imputation of the record with `"2019-02"`.

```
derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  date_imputation = "04-06"
  )
#> # A tibble: 7 × 3
#>   MHSTDTC               ASTDT      ASTDTF
#>   <chr>                 <date>     <chr>
#> 1 "2019-07-18T15:25:40" 2019-07-18 <NA>
#> 2 "2019-07-18T15:25"    2019-07-18 <NA>
#> 3 "2019-07-18"          2019-07-18 <NA>
#> 4 "2019-02"             2019-02-06 D
#> 5 "2019"                2019-04-06 M
#> 6 "2019---07"           2019-04-06 M
#> 7 ""                    NA         <NA>
```

Applying a lower boundary to date imputation with (`min_dates`):

In this example, we derive `ASTDT` where `AESTDTC` is all partial dates in need of imputation. Using `min_dates = exprs(TRTSDTM)`, we are telling the function to apply the treatment start date (`TRTSDTM`) as a lower boundary for imputation via the `min_dates` argument. This means:

- For partial dates that could potentially include `TRTSDTM` (case 1 & 2), the imputed date is adjusted to `TRTSDTM`
- For partial dates that are entirely before `TRTSDTM` (case 3 & 4), standard imputation rules apply without adjustment
- For partial dates that are entirely after `TRTSDTM` (case 5), standard imputation rules apply

```
adae <- tribble(
  ~case, ~AESTDTC, ~TRTSDTM,
  1, "2020-12", ymd_hms("2020-12-06T12:12:12"),
  2, "2020", ymd_hms("2020-12-06T12:12:12"),
  3, "2020-11", ymd_hms("2020-12-06T12:12:12"),
  4, "2020-01", ymd_hms("2020-12-06T12:12:12"),
  5, "2021-01", ymd_hms("2020-12-06T12:12:12")
)

derive_vars_dt(
  adae,
  dtc = AESTDTC,
  new_vars_prefix = "AST",
```

```
  highest_imputation = "M",
  date_imputation = "first",
  min_dates = exprs(TRTSDTM)
  )
#> # A tibble: 5 × 5
#>    case AESTDTC TRTSDTM              ASTDT      ASTDTF
#>   <dbl> <chr>   <dttm>              <date>     <chr>
#> 1     1 2020-12 2020-12-06 12:12:12 2020-12-06 D
#> 2     2 2020    2020-12-06 12:12:12 2020-12-06 M
#> 3     3 2020-11 2020-12-06 12:12:12 2020-11-01 D
#> 4     4 2020-01 2020-12-06 12:12:12 2020-01-01 D
#> 5     5 2021-01 2020-12-06 12:12:12 2021-01-01 D
```

Applying an upper boundary to date imputation with (max_dates):

In this example, we derive ASTDT where AESTDTC is all partial dates in need of imputation. Using max_dates = exprs(TRTEDTM), we are telling the function to apply the treatment end date (TRTEDTM) as an upper boundary for imputation via the max_dates argument. This means:

- For partial dates that could potentially include TRTEDTM (case 1 & 2), the imputed date is adjusted to TRTEDTM
- For partial dates that are entirely before TRTEDTM (case 3 & 4), standard imputation rules apply without adjustment
- For partial dates that are entirely after TRTEDTM (case 5), standard imputation rules apply

```
adae <- tribble(
  ~case, ~AESTDTC, ~TRTSDTM, ~TRTEDTM,
  1, "2020-12", ymd_hms("2020-01-01T12:12:12"), ymd_hms("2020-12-20T23:59:59"),
  2, "2020", ymd_hms("2020-01-01T12:12:12"), ymd_hms("2020-12-20T23:59:59"),
  3, "2020-11", ymd_hms("2020-01-01T12:12:12"), ymd_hms("2020-12-20T23:59:59"),
  4, "2020-01", ymd_hms("2020-01-01T12:12:12"), ymd_hms("2020-12-20T23:59:59"),
  5, "2021-01", ymd_hms("2020-01-01T12:12:12"), ymd_hms("2020-12-20T23:59:59")
)

derive_vars_dt(
  adae,
  dtc = AESTDTC,
  new_vars_prefix = "AST",
  highest_imputation = "M",
  date_imputation = "last",
  max_dates = exprs(TRTEDTM)
)
#> # A tibble: 5 × 6
#>    case AESTDTC TRTSDTM              TRTEDTM             ASTDT      ASTDTF
#>   <dbl> <chr>   <dttm>              <dttm>              <date>     <chr>
#> 1     1 2020-12 2020-01-01 12:12:12 2020-12-20 23:59:59 2020-12-20 D
#> 2     2 2020    2020-01-01 12:12:12 2020-12-20 23:59:59 2020-12-20 M
#> 3     3 2020-11 2020-01-01 12:12:12 2020-12-20 23:59:59 2020-11-30 D
#> 4     4 2020-01 2020-01-01 12:12:12 2020-12-20 23:59:59 2020-01-31 D
#> 5     5 2021-01 2020-01-01 12:12:12 2020-12-20 23:59:59 2021-01-31 D
```

Preserve lower components if higher ones were imputed (`preserve`)**:**

The `preserve` argument can be used to "preserve" information from the partial dates. For example, `"2019---07"`, will be displayed as `"2019-06-07"` rather than `"2019-06-30"` with `preserve = TRUE` and `date_imputation = "mid"` .

```
derive_vars_dt(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  date_imputation = "mid",
  preserve = TRUE
)
#> # A tibble: 7 × 3
#>   MHSTDTC               ASTDT      ASTDTF
#>   <chr>                 <date>     <chr>
#> 1 "2019-07-18T15:25:40" 2019-07-18 <NA>
#> 2 "2019-07-18T15:25"    2019-07-18 <NA>
#> 3 "2019-07-18"          2019-07-18 <NA>
#> 4 "2019-02"             2019-02-15 D
#> 5 "2019"                2019-06-30 M
#> 6 "2019---07"           2019-06-07 M
#> 7 ""                    NA         <NA>
```

Further examples:

Further example usages of this function can be found in the `vignette("imputation")`.

See Also

`vignette("imputation")`

Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trtdurd()`, `derive_vars_dtm()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_duration()`, `derive_vars_dy()`

`derive_vars_dtm` *Derive/Impute a Datetime from a Character Date*

Description

Derive a datetime object (`*DTM`) from a character date (`--DTC`). The date and time can be imputed (see `date_imputation/time_imputation` arguments) and the date/time imputation flag (`*DTF`, `*TMF`) can be added.

Usage

```
derive_vars_dtm(
  dataset,
  new_vars_prefix,
  dtc,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  flag_imputation = "auto",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE,
  ignore_seconds_flag = TRUE
)
```

Arguments

`dataset`

Input dataset

The variables specified by the `dtc` argument are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`new_vars_prefix`

Prefix used for the output variable(s).

A character scalar is expected. For the date variable (`*DT`) is appended to the specified prefix, for the date imputation flag (`*DTF`), and for the time imputation flag (`*TMF`), i.e., for `new_vars_prefix = "AST"` the variables `ASTDT`, `ASTDTF`, and `ASTTMF` are created.

Permitted values a character scalar, i.e., a character vector of length one

Default value none

`dtc`

The `--DTC` date to impute

A character date is expected in a format like `yyyy-mm-dd` or `yyyy-mm-ddThh:mm:ss`. Trailing components can be omitted and `-` is a valid "missing" value for any component.

Permitted values a character date variable

Default value none

`highest_imputation`

Highest imputation level

The `highest_imputation` argument controls which components of the `--DTC` value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, `NA` is returned. For example, for `highest_imputation = "D" "2020"` results in `NA` because the month is missing.

If `"n"` is specified, no imputation is performed, i.e., if any component is missing, `NA` is returned.

If "Y" is specified, date_imputation should be "first" or "last" and min_dates or max_dates should be specified respectively. Otherwise, NA is returned if the year component is missing.

Permitted values "Y" (year, highest level), "M" (month), "D" (day), "h" (hour), "m" (minute), "s" (second), "n" (none, lowest level)

Default value "h"

date_imputation

The value to impute the day/month when a datepart is missing.

A character value is expected.

- The"first" and "last" keywords allow imputation to the first/last day/month. They can also be used to impute the year if used in conjunction with the min_dates or max_dates arguments. Some examples of this are available here.
- When highest_imputation is "M" or "D", the "mid" keyword can also be specified to impute missing components to the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).
- "<dd>" can be specified only if highest_imputation = "D". Missing days are imputed by the specified day, e.g. "10" for the 10th day of the month. The specified day should be valid for all months as otherwise an error might be issued. For example, date_imputation = "30" results in an invalid date of "2024-02-30" for the partial date "2024-02".
- "<mm>-<dd>" can be specified only if highest_imputation is "M", e.g. "06-15" for the 15th of June.

Permitted values a key-word, i.e. "first", "mid", "last", or "<mm>-<dd>" or "<dd>"

Default value "first"

time_imputation

The value to impute the time when a timepart is missing.

A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "first","last" to impute to the start/end of a day.

The argument is ignored if highest_imputation = "n".

Permitted values "first", "last", or user-defined

Default value "first"

flag_imputation

Whether the date/time imputation flag(s) must also be derived.

If "both" or "date" is specified, then date imputation flag is derived. If "auto" is specified and highest_imputation argument is greater than "h", then date imputation flag is derived.

If "both" or "time" is specified, then time imputation flag is derived. If "auto" is specified and highest_imputation argument is not "n", then time imputation flag is derived.

If "none" is specified, then no date or time imputation flag is derived.

Please note that CDISC requirements dictate the need for a date/time imputation flag if any imputation is performed, so flag_imputation = "none" should only be used if the imputed variable is not part of the final ADaM dataset.

Permitted values "auto", "date","time", "both" or "none"

Default value "auto"

min_dates — Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
   ymd_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  highest_imputation = "M"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

For date variables (not datetime) in the list the time is imputed to "00:00:00". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

max_dates — Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

For date variables (not datetime) in the list the time is imputed to "23:59:59". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

preserve — Preserve lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.

For example "2019---07" would return "2019-06-07 if preserve = TRUE (and date_imputation = "mid").

Permitted values TRUE, FALSE

Default value FALSE

ignore_seconds_flag

ADaM IG states that given SDTM (--DTC) variable, if only hours and minutes are ever collected, and seconds are imputed in (*DTM) as 00, then it is not necessary to set (*TMF) to "S".

By default it is assumed that no seconds are collected and *TMF shouldn't be set to "S". A user can set this to FALSE if seconds are collected.

The default value of ignore_seconds_flag is set to TRUE in admiral 1.4.0 and later.

Permitted values TRUE, FALSE

Default value TRUE

Details

In {admiral} we don't allow users to pick any single part of the date/time to impute, we only enable to impute up to a highest level, i.e. you couldn't choose to say impute months, but not days.

The presence of a *DTF variable is checked and the variable is not derived if it already exists in the input dataset. However, if *TMF already exists in the input dataset, a warning is issued and *TMF will be overwritten.

Additionally, the function will throw an error if imputation rules cause an invalid datetime (e.g. "2020-02-01T25:00:00") to be generated. In this case, the user should adjust the imputation rules.

Value

The input dataset with the datetime *DTM (and the date/time imputation flag *DTF, *TMF) added.

Examples

Derive a datetime variable imputing time:

In this example, we derive ASTDTM from MHSTDTC. Note that by default the function imputes missing time components to 00 but doesn't impute missing date components and automatically produces the time imputation flag (ASTTMF).

```
library(tibble)
library(lubridate)

mhdt <- tribble(
  ~MHSTDTC,
  "2019-07-18T15:25",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019---07",
  ""
```

```
)

derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC
)
#> # A tibble: 6 × 3
#>   MHSTDTC             ASTDTM              ASTTMF
#>   <chr>               <dttm>              <chr>
#> 1 "2019-07-18T15:25" 2019-07-18 15:25:00 <NA>
#> 2 "2019-07-18"       2019-07-18 00:00:00 H
#> 3 "2019-02"          NA                  <NA>
#> 4 "2019"             NA                  <NA>
#> 5 "2019---07"        NA                  <NA>
#> 6 ""                 NA                  <NA>
```

Impute to the latest (`date_imputation = "last"`):

In this example, we set `date_imputation = "last"` to get the last month/day for partial dates. We also set `time_imputation = "last"`. The function will use all or part of `23:59:59` for time imputation. Note that `highest_imputation` must be at least `"D"` to perform date imputation. Here we use `highest_imputation = "M"` to request imputation of month and day (and time). Also note that two flag variables are created. By default `ASTTMF` is set to `NA` if only seconds are imputed. Set `ignore_seconds_flag = FALSE` to have the `"S"` flag for `ASTTMF`.

```
derive_vars_dtm(
 mhdt,
 new_vars_prefix = "AST",
 dtc = MHSTDTC,
 date_imputation = "last",
 time_imputation = "last",
 highest_imputation = "M"
)
#> # A tibble: 6 × 4
#>   MHSTDTC             ASTDTM              ASTDTF ASTTMF
#>   <chr>               <dttm>              <chr>  <chr>
#> 1 "2019-07-18T15:25" 2019-07-18 15:25:59 <NA>   <NA>
#> 2 "2019-07-18"       2019-07-18 23:59:59 <NA>   H
#> 3 "2019-02"          2019-02-28 23:59:59 D      H
#> 4 "2019"             2019-12-31 23:59:59 M      H
#> 5 "2019---07"        2019-12-31 23:59:59 M      H
#> 6 ""                 NA                  <NA>   <NA>
```

Suppress imputation flags (`flag_imputation = "none"`):

In this example, we derive `ASTDTM` but suppress the `ASTTMF`. Note that function appends missing `"hh:mm:ss"` to `ASTDTM`. The `flag_imputation = "none"` call ensures no date/time imputation flag is created. In practice, as per CDISC requirements this option can only be selected if the imputed variable is not part of the final ADaM dataset.

```
derive_vars_dtm(
```

```
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  flag_imputation = "none"
)
#> # A tibble: 6 × 2
#>   MHSTDTC            ASTDTM
#>   <chr>              <dttm>
#> 1 "2019-07-18T15:25" 2019-07-18 15:25:00
#> 2 "2019-07-18"       2019-07-18 00:00:00
#> 3 "2019-02"          NA
#> 4 "2019"             NA
#> 5 "2019---07"        NA
#> 6 ""                 NA
```

Avoid imputation after specified datetimes (`max_dates`):

In this example, we derive `AENDTM` where AE end date is imputed to the last date. To ensure that the imputed date is not after the death or data cut off date we can set `max_dates = exprs(DTHDT, DCUTDT)`. Note two flag variables: `ASTDTF` and `ASTTMF` are created. Setting `highest_imputation = "Y"` will allow for the missing `AEENDTC` record to be imputed from `max_dates = exprs(DTHDT, DCUTDT)`.

```
adae <- tribble(
   ~AEENDTC,             ~DTHDT,           ~DCUTDT,
   "2020-12", ymd("2020-12-26"), ymd("2020-12-24"),
   "2020-11", ymd("2020-12-06"), ymd("2020-12-24"),
         "", ymd("2020-12-06"), ymd("2020-12-24"),
"2020-12-20", ymd("2020-12-06"), ymd("2020-12-24")
)

derive_vars_dtm(
  adae,
  dtc = AEENDTC,
  new_vars_prefix = "AEN",
  highest_imputation = "Y",
  date_imputation = "last",
  time_imputation = "last",
  max_dates = exprs(DTHDT, DCUTDT)
)
#> # A tibble: 4 × 6
#>   AEENDTC      DTHDT      DCUTDT     AENDTM              AENDTF AENTMF
#>   <chr>        <date>     <date>     <dttm>              <chr>  <chr>
#> 1 "2020-12"    2020-12-26 2020-12-24 2020-12-24 23:59:59 D      H
#> 2 "2020-11"    2020-12-06 2020-12-24 2020-11-30 23:59:59 D      H
#> 3 ""           2020-12-06 2020-12-24 2020-12-06 23:59:59 Y      H
#> 4 "2020-12-20" 2020-12-06 2020-12-24 2020-12-20 23:59:59 <NA>   H
```

Include `"S"` for time imputation flag (`ignore_seconds_flag`):

In this example, we set ignore_seconds_flag = FALSE to include S for seconds in the ASTTMF variable. The default value of ignore_seconds_flag is TRUE so the "S" is not normally displayed. The ADaM IG states that given SDTM (--DTC) variable, if only hours and minutes are ever collected, and seconds are imputed in (*DTM) as 00, then it is not necessary to set (*TMF) to "S".

```
mhdt <- tribble(
~MHSTDTC,
"2019-07-18T15:25",
"2019-07-18",
"2019-02",
"2019",
"2019---07",
""
)

derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  ignore_seconds_flag = FALSE
)
#> # A tibble: 6 × 4
#>   MHSTDTC             ASTDTM              ASTDTF ASTTMF
#>   <chr>               <dttm>              <chr>  <chr>
#> 1 "2019-07-18T15:25" 2019-07-18 15:25:00 <NA>   S
#> 2 "2019-07-18"       2019-07-18 00:00:00 <NA>   H
#> 3 "2019-02"          2019-02-01 00:00:00 D      H
#> 4 "2019"             2019-01-01 00:00:00 M      H
#> 5 "2019---07"        2019-01-01 00:00:00 M      H
#> 6 ""                 NA                  <NA>   <NA>
```

Preserve lower components if higher ones were imputed (preserve):

In this example, we impute dates as the middle month/day with date_imputation = "mid" and impute time as last (23:59:59) with time_imputation = "last". We use the preserve argument to "preserve" partial dates. For example, "2019---18T15:-:05", will be displayed as "2019-06-18 15:59:05" by setting preserve = TRUE.

```
mhdt <- tribble(
~MHSTDTC,
"2019-07-18T15:25",
"2019---18T15:-:05",
"2019-07-18",
"2019-02",
"2019",
"2019---07",
""
)
```

```
derive_vars_dtm(
  mhdt,
  new_vars_prefix = "AST",
  dtc = MHSTDTC,
  highest_imputation = "M",
  date_imputation = "mid",
  time_imputation = "last",
  preserve = TRUE,
  ignore_seconds_flag = FALSE
)
#> # A tibble: 7 × 4
#>   MHSTDTC             ASTDTM              ASTDTF ASTTMF
#>   <chr>               <dttm>              <chr>  <chr>
#> 1 "2019-07-18T15:25"  2019-07-18 15:25:59 <NA>   S
#> 2 "2019---18T15:-:05" 2019-06-18 15:59:05 M      M
#> 3 "2019-07-18"        2019-07-18 23:59:59 <NA>   H
#> 4 "2019-02"           2019-02-15 23:59:59 D      H
#> 5 "2019"              2019-06-30 23:59:59 M      H
#> 6 "2019---07"         2019-06-07 23:59:59 M      H
#> 7 ""                  NA                  <NA>   <NA>
```

Further examples:

Further example usages of this function can be found in the `vignette("imputation")`.

See Also

`vignette("imputation")`

Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trtdurd()`, `derive_vars_dt()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_duration()`, `derive_vars_dy()`

---

derive_vars_dtm_to_dt *Derive Date Variables from Datetime Variables*

Description

This function creates date(s) as output from datetime variable(s)

Usage

```
derive_vars_dtm_to_dt(dataset, source_vars)
```

Arguments

dataset	Input dataset The variables specified by the source_vars argument are expected to be in the dataset. **Default value** none
source_vars	A list of datetime variables created using exprs() from which dates are to be extracted **Default value** none

Value

A data frame containing the input dataset with the corresponding date (*DT) variable(s) of all datetime variables (*DTM) specified in source_vars.

See Also

Date/Time Derivation Functions that returns variable appended to dataset: derive_var_trtdurd(), derive_vars_dt(), derive_vars_dtm(), derive_vars_dtm_to_tm(), derive_vars_duration(), derive_vars_dy()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

adcm <- tribble(
  ~USUBJID, ~TRTSDTM,              ~ASTDTM,               ~AENDTM,
  "PAT01",  "2012-02-25 23:00:00", "2012-02-28 19:00:00", "2012-02-25 23:00:00",
  "PAT01",  NA,                    "2012-02-28 19:00:00", NA,
  "PAT01",  "2017-02-25 23:00:00", "2013-02-25 19:00:00", "2014-02-25 19:00:00",
  "PAT01",  "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-03-25 23:00:00",
  "PAT01",  "2017-02-25 16:00:00", "2017-02-25 14:00:00", "2017-04-29 14:00:00",
) %>%
  mutate(
    TRTSDTM = as_datetime(TRTSDTM),
    ASTDTM = as_datetime(ASTDTM),
    AENDTM = as_datetime(AENDTM)
  )

adcm %>%
  derive_vars_dtm_to_dt(exprs(TRTSDTM, ASTDTM, AENDTM)) %>%
  select(USUBJID, starts_with("TRT"), starts_with("AST"), starts_with("AEN"))
```

`derive_vars_dtm_to_tm` *Derive Time Variables from Datetime Variables*

Description

This function creates time variable(s) as output from datetime variable(s)

Usage

```
derive_vars_dtm_to_tm(dataset, source_vars)
```

Arguments

`dataset`	Input dataset The variables specified by the `source_vars` argument are expected to be in the dataset. **Default value** none
`source_vars`	A list of datetime variables created using `exprs()` from which time is to be extracted **Default value** none

Details

The names of the newly added variables are automatically set by replacing the `*DTM` suffix of the `source_vars` with `*TM`. The `*TM` variables are created using the `{hms}` package.

Value

A data frame containing the input dataset with the corresponding time (`*TM`) variable(s) of all datetime variables (`*DTM`) specified in `source_vars` with the correct name.

See Also

Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trtdurd()`, `derive_vars_dt()`, `derive_vars_dtm()`, `derive_vars_dtm_to_dt()`, `derive_vars_duration()`, `derive_vars_dy()`

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

adcm <- tribble(
  ~USUBJID, ~TRTSDTM, ~ASTDTM, ~AENDTM,
  "PAT01", "2012-02-25 23:41:10", "2012-02-28 19:03:00", "2013-02-25 23:32:16",
  "PAT01", "", "2012-02-28 19:00:00", "",
```

```
  "PAT01", "2017-02-25 23:00:02", "2013-02-25 19:00:15", "2014-02-25 19:00:56",
  "PAT01", "2017-02-25 16:00:00", "2017-02-25 14:25:00", "2017-03-25 23:00:00",
  "PAT01", "2017-02-25 16:05:17", "2017-02-25 14:20:00", "2018-04-29 14:06:45",
) %>%
  mutate(
    TRTSDTM = as_datetime(TRTSDTM),
    ASTDTM = as_datetime(ASTDTM),
    AENDTM = as_datetime(AENDTM)
  )

adcm %>%
  derive_vars_dtm_to_tm(exprs(TRTSDTM)) %>%
  select(USUBJID, starts_with("TRT"), everything())

adcm %>%
  derive_vars_dtm_to_tm(exprs(TRTSDTM, ASTDTM, AENDTM)) %>%
  select(USUBJID, starts_with("TRT"), starts_with("AS"), starts_with("AE"))
```

`derive_vars_duration` *Derive Duration*

Description

Derives duration between two dates, specified by the variables present in input dataset e.g., duration of adverse events, relative day, age, ...

Usage

```
derive_vars_duration(
  dataset,
  new_var,
  new_var_unit = NULL,
  start_date,
  end_date,
  in_unit = "days",
  out_unit = "DAYS",
  floor_in = TRUE,
  add_one = TRUE,
  trunc_out = FALSE,
  type = "duration"
)
```

Arguments

`dataset`	Input dataset The variables specified by the `start_date` and `end_date` arguments are expected to be in the dataset. **Default value** none

`new_var` Name of variable to create

Default value none

`new_var_unit` Name of the unit variable If the parameter is not specified, no variable for the unit is created.

Default value `NULL`

`start_date` The start date

A date or date-time object is expected.

Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

Refer to `convert_dtc_to_dt()` to obtain a vector of imputed dates.

Default value none

`end_date` The end date

A date or date-time object is expected.

Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

Refer to `convert_dtc_to_dt()` to obtain a vector of imputed dates.

Default value none

`in_unit` Input unit

See floor_in and add_one parameter for details.

Permitted Values (case-insensitive):

For years: `"year"`, `"years"`, `"yr"`, `"yrs"`, `"y"`

For months: `"month"`, `"months"`, `"mo"`, `"mos"`

For days: `"day"`, `"days"`, `"d"`

For hours: `"hour"`, `"hours"`, `"hr"`, `"hrs"`, `"h"`

For minutes: `"minute"`, `"minutes"`, `"min"`, `"mins"`

For seconds: `"second"`, `"seconds"`, `"sec"`, `"secs"`, `"s"`

Default value `"days"`

`out_unit` Output unit

The duration is derived in the specified unit

Permitted Values (case-insensitive):

For years: `"year"`, `"years"`, `"yr"`, `"yrs"`, `"y"`

For months: `"month"`, `"months"`, `"mo"`, `"mos"`

For weeks: `"week"`, `"weeks"`, `"wk"`, `"wks"`, `"w"`

For days: `"day"`, `"days"`, `"d"`

For hours: `"hour"`, `"hours"`, `"hr"`, `"hrs"`, `"h"`

For minutes: `"minute"`, `"minutes"`, `"min"`, `"mins"`

For seconds: `"second"`, `"seconds"`, `"sec"`, `"secs"`, `"s"`

Default value `"days"`

`floor_in` Round down input dates?

The input dates are round down with respect to the input unit, e.g., if the input unit is 'days', the time of the input dates is ignored.

Permitted values TRUE, FALSE
Default value TRUE

add_one — Add one input unit?

If the duration is non-negative, one input unit is added. i.e., the duration can not be zero.

Permitted values TRUE, FALSE
Default value TRUE

trunc_out — Return integer part

The fractional part of the duration (in output unit) is removed, i.e., the integer part is returned.

Permitted values TRUE, FALSE
Default value FALSE

type — lubridate duration type.

See below for details.

Permitted values "duration", "interval"
Default value "duration"

Details

The duration is derived as time from start to end date in the specified output unit. If the end date is before the start date, the duration is negative. The start and end date variable must be present in the specified input dataset.

The lubridate package calculates two types of spans between two dates: duration and interval. While these calculations are largely the same, when the unit of the time period is month or year the result can be slightly different.

The difference arises from the ambiguity in the length of "1 month" or "1 year". Months may have 31, 30, 28, or 29 days, and years are 365 days and 366 during leap years. Durations and intervals help solve the ambiguity in these measures.

The **interval** between 2000-02-01 and 2000-03-01 is 1 (i.e. one month). The **duration** between these two dates is 0.95, which accounts for the fact that the year 2000 is a leap year, February has 29 days, and the average month length is 30.4375, i.e. 29 / 30.4375 = 0.95.

For additional details, review the lubridate time span reference page.

Value

The input dataset with the duration and unit variable added

See Also

compute_duration()

Date/Time Derivation Functions that returns variable appended to dataset: derive_var_trtdurd(), derive_vars_dt(), derive_vars_dtm(), derive_vars_dtm_to_dt(), derive_vars_dtm_to_tm(), derive_vars_dy()

Examples

```
library(lubridate)
library(tibble)

# Derive age in years
data <- tribble(
  ~USUBJID, ~BRTHDT, ~RANDDT,
  "P01", ymd("1984-09-06"), ymd("2020-02-24"),
  "P02", ymd("1985-01-01"), NA,
  "P03", NA, ymd("2021-03-10"),
  "P04", NA, NA
)

derive_vars_duration(data,
  new_var = AAGE,
  new_var_unit = AAGEU,
  start_date = BRTHDT,
  end_date = RANDDT,
  out_unit = "years",
  add_one = FALSE,
  trunc_out = TRUE
)

# Derive adverse event duration in days
data <- tribble(
  ~USUBJID, ~ASTDT, ~AENDT,
  "P01", ymd("2021-03-05"), ymd("2021-03-02"),
  "P02", ymd("2019-09-18"), ymd("2019-09-18"),
  "P03", ymd("1985-01-01"), NA,
  "P04", NA, NA
)

derive_vars_duration(data,
  new_var = ADURN,
  new_var_unit = ADURU,
  start_date = ASTDT,
  end_date = AENDT,
  out_unit = "days"
)

# Derive adverse event duration in minutes
data <- tribble(
  ~USUBJID, ~ADTM, ~TRTSDTM,
  "P01", ymd_hms("2019-08-09T04:30:56"), ymd_hms("2019-08-09T05:00:00"),
  "P02", ymd_hms("2019-11-11T10:30:00"), ymd_hms("2019-11-11T11:30:00"),
  "P03", ymd_hms("2019-11-11T00:00:00"), ymd_hms("2019-11-11T04:00:00"),
  "P04", NA, ymd_hms("2019-11-11T12:34:56"),
)

derive_vars_duration(data,
  new_var = ADURN,
  new_var_unit = ADURU,
```

```
  start_date = ADTM,
  end_date = TRTSDTM,
  in_unit = "minutes",
  out_unit = "minutes",
  add_one = FALSE
)

# Derive adverse event start time since last dose in hours
data <- tribble(
  ~USUBJID, ~ASTDTM, ~LDOSEDTM,
  "P01", ymd_hms("2019-08-09T04:30:56"), ymd_hms("2019-08-08T10:05:00"),
  "P02", ymd_hms("2019-11-11T23:59:59"), ymd_hms("2019-10-11T11:37:00"),
  "P03", ymd_hms("2019-11-11T00:00:00"), ymd_hms("2019-11-10T23:59:59"),
  "P04", ymd_hms("2019-11-11T12:34:56"), NA,
  "P05", NA, ymd_hms("2019-09-28T12:34:56")
)
derive_vars_duration(
  data,
  new_var = LDRELTM,
  new_var_unit = LDRELTMU,
  start_date = LDOSEDTM,
  end_date = ASTDTM,
  in_unit = "hours",
  out_unit = "hours",
  add_one = FALSE
)
```

`derive_vars_dy` *Derive Relative Day Variables*

Description

Adds relative day variables (`*DY`) to the dataset, e.g., `ASTDY` and `AENDY`.

Usage

```
derive_vars_dy(dataset, reference_date, source_vars)
```

Arguments

`dataset`	Input dataset The variables specified by the `reference_date` and `source_vars` arguments are expected to be in the dataset. **Default value** none
`reference_date`	A date or date-time column, e.g., date of first treatment or date-time of last exposure to treatment. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

Default value none

source_vars — A list of datetime or date variables created using `exprs()` from which dates are to be extracted. This can either be a list of date(time) variables or named `*DY` variables and corresponding `*DT(M)` variables e.g. `exprs(TRTSDTM, ASTDTM, AENDT)` or `exprs(TRTSDT, ASTDTM, AENDT, DEATHDY = DTHDT)`. If the source variable does not end in `*DT(M)`, a name for the resulting `*DY` variable must be provided.

Default value none

Details

The relative day is derived as number of days from the reference date to the end date. If it is nonnegative, one is added. That is, the relative day of the reference date is 1. Unless a name is explicitly specified, the name of the resulting relative day variable is generated from the source variable name by replacing DT (or DTM as appropriate) with DY. In the ADaM as in the SDTM, there is no Day 0. If there is a need to create a relative day variable that includes Day 0, then its name must not end in DY.

Value

The input dataset with `*DY` corresponding to the `*DTM` or `*DT` source variable(s) added

See Also

Date/Time Derivation Functions that returns variable appended to dataset: `derive_var_trtdurd()`, `derive_vars_dt()`, `derive_vars_dtm()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_duration()`

Examples

```
library(tibble)
library(lubridate)
library(dplyr, warn.conflicts = FALSE)

datain <- tribble(
  ~TRTSDTM, ~ASTDTM, ~AENDT,
  "2014-01-17T23:59:59", "2014-01-18T13:09:O9", "2014-01-20"
) %>%
  mutate(
    TRTSDTM = as_datetime(TRTSDTM),
    ASTDTM = as_datetime(ASTDTM),
    AENDT = ymd(AENDT)
  )

derive_vars_dy(
  datain,
  reference_date = TRTSDTM,
  source_vars = exprs(TRTSDTM, ASTDTM, AENDT)
)
```

```
# specifying name of new variables
datain <- tribble(
  ~TRTSDT, ~DTHDT,
  "2014-01-17", "2014-02-01"
) %>%
  mutate(
    TRTSDT = ymd(TRTSDT),
    DTHDT = ymd(DTHDT)
  )

derive_vars_dy(
  datain,
  reference_date = TRTSDT,
  source_vars = exprs(TRTSDT, DEATHDY = DTHDT)
)
```

`derive_vars_extreme_event` *Add the Worst or Best Observation for Each By Group as New Variables*

Description

Add the first available record from `events` for each by group as new variables, all variables of the selected observation are kept. It can be used for selecting the extreme observation from a series of user-defined events.

Usage

```
derive_vars_extreme_event(
  dataset,
  by_vars,
  events,
  tmp_event_nr_var = NULL,
  order,
  mode,
  source_datasets = NULL,
  check_type = "warning",
  new_vars
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` and `order` arguments are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

by_vars
Grouping variables

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value NULL

events
Conditions and new values defining events

A list of event() or event_joined() objects is expected. Only observations listed in the events are considered for deriving extreme event. If multiple records meet the filter condition, take the first record sorted by order. The data is grouped by by_vars, i.e., summary functions like all() or any() can be used in condition.

For event_joined() events the observations are selected by calling filter_joined(). The condition field is passed to the filter_join argument.

Default value none

tmp_event_nr_var
Temporary event number variable

The specified variable is added to all source datasets and is set to the number of the event before selecting the records of the event.

It can be used in order to determine which record should be used if records from more than one event are selected.

The variable is not included in the output dataset.

Default value NULL

order
Sort order

If a particular event from events has more than one observation, within the event and by group, the records are ordered by the specified order.

For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

Permitted values list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL))

Default value none

mode
Selection mode (first or last)

If a particular event from events has more than one observation, "first"/"last" is used to select the first/last record of this type of event sorting by order.

Permitted values "first", "last"

Default value none

source_datasets
Source datasets

A named list of datasets is expected. The dataset_name field of event() and event_joined() refers to the dataset provided in the list.

Default value NULL

check_type
Check uniqueness?

If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values `"none"`, `"message"`, `"warning"`, `"error"`

Default value `"warning"`

`new_vars` Variables to add

The specified variables from the events are added to the output dataset. Variables can be renamed by naming the element, i.e., `new_vars = exprs(<new name> = <old name>)`.

Default value none

Details

1. For each event select the observations to consider:
   (a) If the event is of class `event`, the observations of the source dataset are restricted by `condition` and then the first or last (`mode`) observation per by group (`by_vars`) is selected.
   If the event is of class `event_joined`, `filter_joined()` is called to select the observations.
   (b) The variables specified by the `set_values_to` field of the event are added to the selected observations.
   (c) The variable specified for `tmp_event_nr_var` is added and set to the number of the event.
2. All selected observations are bound together.
3. For each group (with respect to the variables specified for the `by_vars` parameter) the first or last observation (with respect to the order specified for the `order` parameter and the mode specified for the `mode` parameter) is selected.
4. The variables specified by the `new_vars` parameter are added to the selected observations.
5. The variables are added to input dataset.

Value

The input dataset with the best or worst observation of each by group added as new variables.

See Also

`event()`, `event_joined()`, `derive_extreme_event()`

ADSL Functions that returns variable appended to dataset: `derive_var_age_years()`, `derive_vars_aage()`, `derive_vars_period()`

Examples

```
library(tibble)
library(dplyr)
library(lubridate)

adsl <- tribble(
  ~STUDYID, ~USUBJID, ~TRTEDT, ~DTHDT,
  "PILOT01", "01-1130", ymd("2014-08-16"), ymd("2014-09-13"),
  "PILOT01", "01-1133", ymd("2013-04-28"), ymd(""),
  "PILOT01", "01-1211", ymd("2013-01-12"), ymd(""),
  "PILOT01", "09-1081", ymd("2014-04-27"), ymd(""),
```

```
  "PILOT01", "09-1088", ymd("2014-10-09"), ymd("2014-11-01"),
)

lb <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~LBSEQ,             ~LBDTC,
  "PILOT01",    "LB", "01-1130",    219, "2014-06-07T13:20",
  "PILOT01",    "LB", "01-1130",    322, "2014-08-16T13:10",
  "PILOT01",    "LB", "01-1133",    268, "2013-04-18T15:30",
  "PILOT01",    "LB", "01-1133",    304, "2013-05-01T10:13",
  "PILOT01",    "LB", "01-1211",      8, "2012-10-30T14:26",
  "PILOT01",    "LB", "01-1211",    162, "2013-01-08T12:13",
  "PILOT01",    "LB", "09-1081",     47, "2014-02-01T10:55",
  "PILOT01",    "LB", "09-1081",    219, "2014-05-10T11:15",
  "PILOT01",    "LB", "09-1088",    283, "2014-09-27T12:13",
  "PILOT01",    "LB", "09-1088",    322, "2014-10-09T13:25"
) %>%
  mutate(
    ADT = convert_dtc_to_dt(LBDTC)
  )

derive_vars_extreme_event(
  adsl,
  by_vars = exprs(STUDYID, USUBJID),
  events = list(
    event(
      dataset_name = "adsl",
      condition = !is.na(DTHDT),
      set_values_to = exprs(LSTALVDT = DTHDT, DTHFL = "Y")
    ),
    event(
      dataset_name = "lb",
      condition = !is.na(ADT),
      order = exprs(ADT),
      mode = "last",
      set_values_to = exprs(LSTALVDT = ADT, DTHFL = "N")
    ),
    event(
      dataset_name = "adsl",
      condition = !is.na(TRTEDT),
      order = exprs(TRTEDT),
      mode = "last",
      set_values_to = exprs(LSTALVDT = TRTEDT, DTHFL = "N")
    )
  ),
  source_datasets = list(adsl = adsl, lb = lb),
  tmp_event_nr_var = event_nr,
  order = exprs(LSTALVDT, event_nr),
  mode = "last",
  new_vars = exprs(LSTALVDT, DTHFL)
)

# Derive DTHCAUS from AE and DS domain data
adsl <- tribble(
```

```
  ~STUDYID,  ~USUBJID,
  "STUDY01", "PAT01",
  "STUDY01", "PAT02",
  "STUDY01", "PAT03"
)
ae <- tribble(
  ~STUDYID, ~USUBJID, ~AESEQ, ~AEDECOD, ~AEOUT, ~AEDTHDTC,
  "STUDY01", "PAT01", 12, "SUDDEN DEATH", "FATAL", "2021-04-04",
  "STUDY01", "PAT01", 13, "CARDIAC ARREST", "FATAL", "2021-04-03",
)

ds <- tribble(
  ~STUDYID, ~USUBJID, ~DSSEQ, ~DSDECOD, ~DSTERM, ~DSSTDTC,
 "STUDY01", "PAT02", 1, "INFORMED CONSENT OBTAINED", "INFORMED CONSENT OBTAINED", "2021-04-03",
  "STUDY01", "PAT02", 2, "RANDOMIZATION", "RANDOMIZATION", "2021-04-11",
  "STUDY01", "PAT02", 3, "DEATH", "DEATH DUE TO PROGRESSION OF DISEASE", "2022-02-01",
  "STUDY01", "PAT03", 1, "DEATH", "POST STUDY REPORTING OF DEATH", "2022-03-03"
)

derive_vars_extreme_event(
  adsl,
  by_vars = exprs(STUDYID, USUBJID),
  events = list(
    event(
      dataset_name = "ae",
      condition = AEOUT == "FATAL",
      set_values_to = exprs(DTHCAUS = AEDECOD, DTHDT = convert_dtc_to_dt(AEDTHDTC)),
      order = exprs(DTHDT)
    ),
    event(
      dataset_name = "ds",
      condition = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
      set_values_to = exprs(DTHCAUS = DSTERM, DTHDT = convert_dtc_to_dt(DSSTDTC)),
      order = exprs(DTHDT)
    )
  ),
  source_datasets = list(ae = ae, ds = ds),
  tmp_event_nr_var = event_nr,
  order = exprs(DTHDT, event_nr),
  mode = "first",
  new_vars = exprs(DTHCAUS, DTHDT)
)
```

`derive_vars_joined` *Add Variables from an Additional Dataset Based on Conditions from Both Datasets*

Description

The function adds variables from an additional dataset to the input dataset. The selection of the observations from the additional dataset can depend on variables from both datasets. For example,

add the lowest value (nadir) before the current observation.

Usage

```
derive_vars_joined(
  dataset,
  dataset_add,
  by_vars = NULL,
  order = NULL,
  new_vars = NULL,
  tmp_obs_nr_var = NULL,
  join_vars = NULL,
  join_type,
  filter_add = NULL,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  filter_join = NULL,
  mode = NULL,
  exist_flag = NULL,
  true_value = "Y",
  false_value = NA_character_,
  missing_values = NULL,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. **Permitted values** a dataset, i.e., a data.frame or tibble **Default value** none
dataset_add	Additional dataset The variables specified by the by_vars, the new_vars, the join_vars, and the order argument are expected. **Permitted values** a dataset, i.e., a data.frame or tibble **Default value** none
by_vars	Grouping variables The two datasets are joined by the specified variables. Variables can be renamed by naming the element, i.e. by_vars = exprs(<name in input dataset> = <r similar to the dplyr joins. **Permitted values** list of variables created by exprs(), e.g., exprs(USUBJID, VISIT) **Default value** NULL

order

Sort order

If the argument is set to a non-null value, for each observation of the input dataset the first or last observation from the joined dataset is selected with respect to the specified order. The specified variables are expected in the additional dataset (dataset_add). If a variable is available in both dataset and dataset_add, the one from dataset_add is used for the sorting.

If an expression is named, e.g., exprs(EXSTDT = convert_dtc_to_dt(EXSTDTC), EXSEQ), a corresponding variable (EXSTDT) is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for join_vars and new_vars. The variable is not included in the output dataset.

For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value NULL

new_vars

Variables to add

The specified variables from the additional dataset are added to the output dataset. Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>).

For example new_vars = exprs(var1, var2) adds variables var1 and var2 from dataset_add to the input dataset.

And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2 from dataset_add and adds them to the input dataset renaming old_var2 to new_var2.

Values of the added variables can be modified by specifying an expression. For example, new_vars = LASTRSP = exprs(str_to_upper(AVALC)) adds the variable LASTRSP to the dataset and sets it to the upper case value of AVALC.

If the argument is not specified or set to NULL, all variables from the additional dataset (dataset_add) are added. In the case when a variable exists in both datasets, an error is issued to ensure the user either adds to by_vars, removes or renames.

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value NULL

tmp_obs_nr_var

Temporary observation number

The specified variable is added to the input dataset (dataset) and the additional dataset (dataset_add). It is set to the observation number with respect to order. For each by group (by_vars) the observation number starts with 1. If there is more than one record for specific values for by_vars and order, all records get the same observation number. By default, a warning (see check_type) is issued in this case. The variable can be used in the conditions (filter_join, first_cond_upper, first_cond_lower). It can also be used to select consecutive observations or the last observation.

The variable is not included in the output dataset. To include it specify it for new_vars.

Permitted values an unquoted symbol, e.g., AVAL

Default value NULL

join_vars

Variables to use from additional dataset

Any extra variables required from the additional dataset for `filter_join` should be specified for this argument. Variables specified for `new_vars` do not need to be repeated for `join_vars`. If a specified variable exists in both the input dataset and the additional dataset, the suffix ".join" is added to the variable from the additional dataset.

If an expression is named, e.g., `exprs(EXTDT = convert_dtc_to_dt(EXSTDTC))`, a corresponding variable is added to the additional dataset and can be used in the filter conditions (`filter_add`, `filter_join`) and for `new_vars`. The variable is not included in the output dataset.

The variables are not included in the output dataset.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

join_type

Observations to keep after joining

The argument determines which of the joined observations are kept with respect to the original observation. For example, if `join_type = "after"` is specified all observations after the original observations are kept.

For example for confirmed response or BOR in the oncology setting or confirmed deterioration in questionnaires the confirmatory assessment must be after the assessment. Thus `join_type = "after"` could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation. For example, to identify AEs occurring on or after seven days before a COVID AE. Thus `join_type = "all"` could be used.

Permitted values `"before"`, `"after"`, `"all"`

Default value none

filter_add

Filter for additional dataset (`dataset_add`)

Only observations from `dataset_add` fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by `order` or `new_vars` arguments can be used in the condition.

The condition can include summary functions like `all()` or `any()`. The additional dataset is grouped by the by variables (`by_vars`).

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

first_cond_lower

Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the last observation before the current observation where the specified condition is fulfilled up to the current observation. If the condition is not fulfilled for any of the other observations, no observations are considered.

This argument should be specified if `filter_join` contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For an example, see the "Examples" section below.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

first_cond_upper

Condition for selecting range of data (after)

If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the other observations, no observations are considered.

This argument should be specified if filter_join contains summary functions which should not apply to all observations but only up to the confirmation assessment. For an example, see the "Examples" section below.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

filter_join

Filter for the joined dataset

The specified condition is applied to the joined dataset. Therefore variables from both datasets dataset and dataset_add can be used.

Variables created by order or new_vars arguments can be used in the condition.

The condition can include summary functions like all() or any(). The joined dataset is grouped by the original observations.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

mode

Selection mode

Determines if the first or last observation is selected. If the order argument is specified, mode must be non-null.

If the order argument is not specified, the mode argument is ignored.

Permitted values "first", "last"

Default value NULL

exist_flag

Exist flag

If the argument is specified (e.g., exist_flag = FLAG), the specified variable (e.g., FLAG) is added to the input dataset. This variable will be the value provided in true_value for all selected records from dataset_add which are merged into the input dataset, and the value provided in false_value otherwise.

Permitted values an unquoted symbol, e.g., AVAL

Default value NULL

true_value

True value

The value for the specified variable exist_flag, applicable to the first or last observation (depending on the mode) of each by group.

Permitted values a character scalar, i.e., a character vector of length one

Default value "Y"

false_value

False value

The value for the specified variable exist_flag, NOT applicable to the first or last observation (depending on the mode) of each by group.

Permitted values a character scalar, i.e., a character vector of length one

Default value NA_character_

missing_values Values for non-matching observations

For observations of the input dataset (dataset) which do not have a matching observation in the additional dataset (dataset_add) the values of the specified variables are set to the specified value. Only variables specified for new_vars can be specified for missing_values.

Permitted values list of named expressions created by exprs(), e.g., exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))

Default value NULL

check_type Check uniqueness?

If "message", "warning" or "error" is specified, the specified message is issued if the observations of the (restricted) joined dataset are not unique with respect to the by variables and the order.

This argument is ignored if order is not specified. In this case an error is issued independent of check_type if the restricted joined dataset contains more than one observation for any of the observations of the input dataset.

Permitted values "none", "message", "warning", "error"

Default value "warning"

Details

1. The variables specified by order are added to the additional dataset (dataset_add).
2. The variables specified by join_vars are added to the additional dataset (dataset_add).
3. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
4. The input dataset and the (restricted) additional dataset are left joined by the grouping variables (by_vars). If no grouping variables are specified, a full join is performed.
5. If first_cond_lower is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where first_cond_lower is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

   If first_cond_upper is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where first_cond_upper is fulfilled (the observation fulfilling the condition is included). If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

   For an example, see the "Examples" section below.
6. The joined dataset is restricted by the filter_join condition.
7. If order is specified, for each observation of the input dataset the first or last observation (depending on mode) is selected.
8. The variables specified for new_vars are created (if requested) and merged to the input dataset. I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the joined dataset the new variables are set as specified by

missing_values (or to NA for variables not in missing_values). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option save_memory to TRUE (see set_admiral_options()). This reduces the memory consumption but increases the run-time.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variables specified for new_vars from the additional dataset (dataset_add).

Examples

Note on usage versus derive_vars_merged():

The question between using derive_vars_merged() or the more powerful derive_vars_joined() comes down to how you need to select the observations to be merged.

- If the observations from dataset_add to merge can be selected by a condition (filter_add) using *only* variables from dataset_add, then always use derive_vars_merged() as it requires less resources (time and memory). A common example of this would be a randomization date in ADSL, where you are simply merging on a date from DS according to a certain DSDECOD condition such as DSDECOD == "RANDOMIZATION".
- However, if the selection of the observations from dataset_add can depend on variables from *both* datasets, then use derive_vars_joined(). An example of this would be assigning period variables from ADSL to an ADAE, where you now need to check each adverse event start date against the period start and end dates to decide which period value to join.

Basic join based on a generic time window (filter_join):

Derive a visit based on where the study day falls according to a scheduled set of time windows.

- The filter_join argument here can check conditions using variables from both the dataset and dataset_add, so the study day is compared to the start and end of the time window.
- As no grouping variables are assigned using the by_vars argument, a full join is performed keeping all variables from dataset_add.

```
library(tibble)
library(lubridate)
library(dplyr, warn.conflicts = FALSE)
library(tidyr, warn.conflicts = FALSE)

adbds <- tribble(
  ~USUBJID, ~ADY, ~AVAL,
  "1",       -33,    11,
  "1",        -7,    10,
  "1",         1,    12,
  "1",         8,    12,
  "1",        15,     9,
  "1",        20,    14,
  "1",        24,    12,
  "2",        -1,    13,
```

```
  "2",        13,     8
) %>%
  mutate(STUDYID = "AB42")

windows <- tribble(
  ~AVISIT,    ~AWLO, ~AWHI,
  "BASELINE",   -30,     1,
  "WEEK 1",       2,     7,
  "WEEK 2",       8,    15,
  "WEEK 3",      16,    22,
  "WEEK 4",      23,    30
)

derive_vars_joined(
  adbds,
  dataset_add = windows,
  join_type = "all",
  filter_join = AWLO <= ADY & ADY <= AWHI
) %>%
  select(USUBJID, ADY, AWLO, AWHI, AVISIT)
#> # A tibble: 9 × 5
#>   USUBJID   ADY  AWLO  AWHI AVISIT
#>   <chr>   <dbl> <dbl> <dbl> <chr>
#> 1 1         -33    NA    NA <NA>
#> 2 1          -7   -30     1 BASELINE
#> 3 1           1   -30     1 BASELINE
#> 4 1           8     8    15 WEEK 2
#> 5 1          15     8    15 WEEK 2
#> 6 1          20    16    22 WEEK 3
#> 7 1          24    23    30 WEEK 4
#> 8 2          -1   -30     1 BASELINE
#> 9 2          13     8    15 WEEK 2
```

Join only the lowest/highest value occurring within a condition (`filter_join`, `order` and `mode`):

Derive the nadir value for each observation (i.e. the lowest value occurring before) by subject.

- Note how `dataset` and `dataset_add` are the same here, so we are joining a dataset with itself. This enables us to compare records within the dataset to each other.
- Now we use `by_vars` as we only want to perform the join by subject.
- To find the lowest value we use the `order` and `mode` arguments.
- We subsequently need to check `ADY` to only check assessments occurring before. As this is not included in `by_vars` or `order`, we have to ensure it also gets joined by adding to `join_vars`. Then in `filter_join` note how `ADY.join < ADY` is used as the same variable exists in both datasets, so the version from `dataset_add` has `.join` added.
- According to the `AVAL` sort order used there could be duplicates (e.g. see subject `"1"` records at day 1 and 8), but given we only need to join `AVAL` itself here it doesn't actually matter to us which exact record is taken. So, in this example, we silence the uniqueness check by using `check_type = "none"`.

```
derive_vars_joined(
  adbds,
  dataset_add = adbds,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(AVAL),
  new_vars = exprs(NADIR = AVAL),
  join_vars = exprs(ADY),
  join_type = "all",
  filter_join = ADY.join < ADY,
  mode = "first",
  check_type = "none"
) %>%
  select(USUBJID, ADY, AVAL, NADIR)
#> # A tibble: 9 × 4
#>   USUBJID   ADY  AVAL NADIR
#>   <chr>   <dbl> <dbl> <dbl>
#> 1 1         -33    11    NA
#> 2 1          -7    10    11
#> 3 1           1    12    10
#> 4 1           8    12    10
#> 5 1          15     9    10
#> 6 1          20    14     9
#> 7 1          24    12     9
#> 8 2          -1    13    NA
#> 9 2          13     8    13
```

Filtering which records are joined from the additional dataset (`filter_add`):
Imagine we wanted to achieve the same as above, but we now want to derive this allowing only post-baseline values to be possible for the nadir.

- The `filter_add` argument can be used here as we only need to restrict the source data from `dataset_add`.

```
derive_vars_joined(
  adbds,
  dataset_add = adbds,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(AVAL),
  new_vars = exprs(NADIR = AVAL),
  join_vars = exprs(ADY),
  join_type = "all",
  filter_add = ADY > 0,
  filter_join = ADY.join < ADY,
  mode = "first",
  check_type = "none"
) %>%
  select(USUBJID, ADY, AVAL, NADIR)
#> # A tibble: 9 × 4
#>   USUBJID   ADY  AVAL NADIR
#>   <chr>   <dbl> <dbl> <dbl>
```

```
#> 1 1          -33    11    NA
#> 2 1           -7    10    NA
#> 3 1            1    12    NA
#> 4 1            8    12    12
#> 5 1           15     9    12
#> 6 1           20    14     9
#> 7 1           24    12     9
#> 8 2           -1    13    NA
#> 9 2           13     8    NA
```

Combining all of the above examples:

Using all of the arguments demonstrated above, here is a more complex example to add to ADAE the highest hemoglobin value occurring within two weeks before each adverse event. Also join the day it occurred, taking the earliest occurrence if more than one assessment with the same value.

- Note how we used `mode = "last"` to get the highest lab value, but then as we wanted the earliest occurrence if more than one it means we need to add `desc(ADY)` to `order`. i.e. the last day when in descending order is the first.

```
adae <- tribble(
  ~USUBJID, ~ASTDY,
  "1",           3,
  "1",          22,
  "2",           2
) %>%
  mutate(STUDYID = "AB42")

adlb <- tribble(
  ~USUBJID, ~PARAMCD, ~ADY, ~AVAL,
  "1",      "HGB",       1,   8.5,
  "1",      "HGB",       3,   7.9,
  "1",      "HGB",       5,   8.9,
  "1",      "HGB",       8,   8.0,
  "1",      "HGB",       9,   8.0,
  "1",      "HGB",      16,   7.4,
  "1",      "ALB",       1,    42,
) %>%
  mutate(STUDYID = "AB42")

derive_vars_joined(
  adae,
  dataset_add = adlb,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(AVAL, desc(ADY)),
  new_vars = exprs(HGB_MAX = AVAL, HGB_DY = ADY),
  join_type = "all",
  filter_add = PARAMCD == "HGB",
  filter_join = ASTDY - 14 <= ADY & ADY <= ASTDY,
  mode = "last"
```

```
) %>%
  select(USUBJID, ASTDY, HGB_MAX, HGB_DY)
#> # A tibble: 3 × 4
#>   USUBJID ASTDY HGB_MAX HGB_DY
#>   <chr>   <dbl>   <dbl>  <dbl>
#> 1 1           3     8.5      1
#> 2 1          22     8        8
#> 3 2           2    NA       NA
```

Compute values in `new_vars` **and** `order`**:**

Add to `ADAE` the number of days since the last dose of treatment, plus 1 day. If the dose occurs on the same day as the AE then include it as the last dose.

- In the `new_vars` argument, other functions can be utilized to modify the joined values using variables from both `dataset` and `dataset_add`. For example, in the below case we want to calculate the number of days between the AE and the last dose using `compute_duration()`. This function includes the plus 1 day as default.
- Also note how in this example `EXSDT` is created via the `order` argument and then used for `new_vars`, `filter_add` and `filter_join`.
- The reason to use `join_type = "all"` here instead of `"before"` is that we want to include any dose occurring on the same day as the AE, hence the `filter_join = EXSDT <= ASTDT`. Whereas using `join_type = "before"` would have resulted in the condition `EXSDT < ASTDT`. See the next example instead for `join_type = "before"`.

```
adae <- tribble(
  ~USUBJID, ~ASTDT,
  "1",      "2020-02-02",
  "1",      "2020-02-04",
  "2",      "2021-01-08"
) %>%
  mutate(
    ASTDT = ymd(ASTDT),
    STUDYID = "AB42"
  )

ex <- tribble(
  ~USUBJID, ~EXSDTC,
  "1",      "2020-01-10",
  "1",      "2020-01",
  "1",      "2020-01-20",
  "1",      "2020-02-03",
  "2",      "2021-01-05"
) %>%
  mutate(STUDYID = "AB42")

derive_vars_joined(
  adae,
  dataset_add = ex,
  by_vars = exprs(STUDYID, USUBJID),
```

```
  order = exprs(EXSDT = convert_dtc_to_dt(EXSDTC)),
  join_type = "all",
  new_vars = exprs(LDRELD = compute_duration(
    start_date = EXSDT, end_date = ASTDT
  )),
  filter_add = !is.na(EXSDT),
  filter_join = EXSDT <= ASTDT,
  mode = "last"
) %>%
  select(USUBJID, ASTDT, LDRELD)
#> # A tibble: 3 × 3
#>   USUBJID ASTDT      LDRELD
#>   <chr>   <date>      <dbl>
#> 1 1       2020-02-02     14
#> 2 1       2020-02-04      2
#> 3 2       2021-01-08      4
```

Join records occurring before a condition (`join_type = "before"`):

In an arbitrary dataset where subjects have values of `"0"`, `"-"`, `"+"` or `"++"`, for any value of `"0"` derive the last occurring `"++"` day that occurs before the `"0"`.

- The `AVAL.join == "++"` in `filter_join`, along with `order` and `mode` taking the last day, identifies the target records to join from `dataset_add` for each observation of `dataset`.
- Then `join_type = "before"` is now used instead of `join_type = "all"`. This is because we only want to join the records occurring before the current observation in `dataset`. Including `AVAL == "0"` in `filter_join` ensures here that we only populate the new variable for records with `AVAL == "0"` in our `dataset`.

```
myd <- tribble(
  ~USUBJID, ~ADY, ~AVAL,
  "1",          1, "++",
  "1",          2, "-",
  "1",          3, "0",
  "1",          4, "+",
  "1",          5, "++",
  "1",          6, "-",
  "2",          1, "-",
  "2",          2, "++",
  "2",          3, "+",
  "2",          4, "0",
  "2",          5, "-",
  "2",          6, "++",
  "2",          7, "0"
) %>%
  mutate(STUDYID = "AB42")

derive_vars_joined(
  myd,
  dataset_add = myd,
  by_vars = exprs(STUDYID, USUBJID),
```

```
  order = exprs(ADY),
  mode = "last",
  new_vars = exprs(PREVPLDY = ADY),
  join_vars = exprs(AVAL),
  join_type = "before",
  filter_join = AVAL == "0" & AVAL.join == "++"
) %>%
  select(USUBJID, ADY, AVAL, PREVPLDY)
#> # A tibble: 13 × 4
#>    USUBJID   ADY AVAL  PREVPLDY
#>    <chr>   <dbl> <chr>    <dbl>
#>  1 1           1 ++          NA
#>  2 1           2 -           NA
#>  3 1           3 0            1
#>  4 1           4 +           NA
#>  5 1           5 ++          NA
#>  6 1           6 -           NA
#>  7 2           1 -           NA
#>  8 2           2 ++          NA
#>  9 2           3 +           NA
#> 10 2           4 0            2
#> 11 2           5 -           NA
#> 12 2           6 ++          NA
#> 13 2           7 0            6
```

Join records occurring before a condition and checking all values in between (`first_cond_lower`, `join_type` and `filter_join`):

In the same example as above, now additionally check that in between the `"++"` and the `"0"` all results must be either `"+"` or `"++"`.

- Firstly, `first_cond_lower = AVAL.join == "++"` is used so that for each observation of `dataset` the joined records from `dataset_add` are restricted to only include from the last occurring `"++"` before. This is necessary because of the use of a summary function in `filter_join` only on a subset of the joined observations as explained below.
- The `filter_join` condition used here now includes `all(AVAL.join %in% c("+", "++"))` to further restrict the joined records from `dataset_add` to only where all the values are either `"+"` or `"++"`.
- The `order` and `mode` arguments ensure only the day of the `"++"` value is joined. For example, for subject `"2"` it selects the day 2 record instead of day 3, by using `"first"`.

```
derive_vars_joined(
  myd,
  dataset_add = myd,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(ADY),
  mode = "first",
  new_vars = exprs(PREVPLDY = ADY),
  join_vars = exprs(AVAL),
  join_type = "before",
```

```
  first_cond_lower = AVAL.join == "++",
  filter_join = AVAL == "0" & all(AVAL.join %in% c("+", "++"))
) %>%
  select(USUBJID, ADY, AVAL, PREVPLDY)
#> # A tibble: 13 × 4
#>    USUBJID   ADY AVAL  PREVPLDY
#>    <chr>   <dbl> <chr>    <dbl>
#>  1 1           1 ++          NA
#>  2 1           2 -           NA
#>  3 1           3 0           NA
#>  4 1           4 +           NA
#>  5 1           5 ++          NA
#>  6 1           6 -           NA
#>  7 2           1 -           NA
#>  8 2           2 ++          NA
#>  9 2           3 +           NA
#> 10 2           4 0            2
#> 11 2           5 -           NA
#> 12 2           6 ++          NA
#> 13 2           7 0            6
```

Join records occurring after a condition checking all values in between (`first_cond_upper`, `join_type` and `filter_join`):

Similar to the above, now derive the first `"++"` day after any `"0"` where all results in between are either `"+"` or `"++"`.

- Note how the main difference here is the use of `join_type = "after"`, `mode = "last"` and the `first_cond_upper` argument, instead of `first_cond_lower`.

```
derive_vars_joined(
  myd,
  dataset_add = myd,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(ADY),
  mode = "last",
  new_vars = exprs(NEXTPLDY = ADY),
  join_vars = exprs(AVAL),
  join_type = "after",
  first_cond_upper = AVAL.join == "++",
  filter_join = AVAL == "0" & all(AVAL.join %in% c("+", "++"))
) %>%
  select(USUBJID, ADY, AVAL, NEXTPLDY)
#> # A tibble: 13 × 4
#>    USUBJID   ADY AVAL  NEXTPLDY
#>    <chr>   <dbl> <chr>    <dbl>
#>  1 1           1 ++          NA
#>  2 1           2 -           NA
#>  3 1           3 0            5
#>  4 1           4 +           NA
#>  5 1           5 ++          NA
```

```
#>  6 1           6 -           NA
#>  7 2           1 -           NA
#>  8 2           2 ++          NA
#>  9 2           3 +           NA
#> 10 2           4 0           NA
#> 11 2           5 -           NA
#> 12 2           6 ++          NA
#> 13 2           7 0           NA
```

Join a value from the next occurring record (`join_type = "after"`):

Add the value from the next occurring record as a new variable.

- The `join_type = "after"` here essentially acts as a lag to join variables from the next occurring record, and `mode = "first"` selects the first of these.

```
derive_vars_joined(
  myd,
  dataset_add = myd,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(ADY),
  mode = "first",
  new_vars = exprs(NEXTVAL = AVAL),
  join_vars = exprs(AVAL),
  join_type = "after"
) %>%
  select(USUBJID, ADY, AVAL, NEXTVAL)
#> # A tibble: 13 × 4
#>    USUBJID   ADY AVAL  NEXTVAL
#>    <chr>   <dbl> <chr> <chr>
#>  1 1           1 ++    -
#>  2 1           2 -     0
#>  3 1           3 0     +
#>  4 1           4 +     ++
#>  5 1           5 ++    -
#>  6 1           6 -     <NA>
#>  7 2           1 -     ++
#>  8 2           2 ++    +
#>  9 2           3 +     0
#> 10 2           4 0     -
#> 11 2           5 -     ++
#> 12 2           6 ++    0
#> 13 2           7 0     <NA>
```

Join records after a condition occurring in consecutive visits (`tmp_obs_nr_var`, `join_type` and `filter_join`):

Find the last occurring value on any of the next 3 unique visit days.

- The `tmp_obs_nr_var` argument can be useful as shown here to help pick out records happening before or after with respect to `order`, as you can see in the `filter_join`.

```
derive_vars_joined(
```

```
  myd,
  dataset_add = myd,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(ADY),
  mode = "last",
  new_vars = exprs(NEXTVAL = AVAL),
  tmp_obs_nr_var = tmp_obs_nr,
  join_vars = exprs(AVAL),
  join_type = "after",
  filter_join = tmp_obs_nr + 3 >= tmp_obs_nr.join
) %>%
  select(USUBJID, ADY, AVAL, NEXTVAL)
#> # A tibble: 13 × 4
#>    USUBJID   ADY AVAL  NEXTVAL
#>    <chr>   <dbl> <chr> <chr>
#>  1 1           1 ++    +
#>  2 1           2 -     ++
#>  3 1           3 0     -
#>  4 1           4 +     -
#>  5 1           5 ++    -
#>  6 1           6 -     <NA>
#>  7 2           1 -     0
#>  8 2           2 ++    -
#>  9 2           3 +     ++
#> 10 2           4 0     0
#> 11 2           5 -     0
#> 12 2           6 ++    0
#> 13 2           7 0     <NA>
```

Derive period variables (APERIOD, APERSDT, APEREDT):

Create a period reference dataset from ADSL and join this with ADAE to identify within which period each AE occurred.

```
adsl <- tribble(
  ~USUBJID, ~AP01SDT,     ~AP01EDT,     ~AP02SDT,     ~AP02EDT,
  "1",      "2021-01-04", "2021-02-06", "2021-02-07", "2021-03-07",
  "2",      "2021-02-02", "2021-03-02", "2021-03-03", "2021-04-01"
) %>%
  mutate(across(ends_with("DT"), ymd)) %>%
  mutate(STUDYID = "AB42")

period_ref <- create_period_dataset(
  adsl,
  new_vars = exprs(APERSDT = APxxSDT, APEREDT = APxxEDT)
)

period_ref
#> # A tibble: 4 × 5
#>   STUDYID USUBJID APERIOD APERSDT    APEREDT
```

```
#>   <chr>   <chr>     <int> <date>     <date>
#> 1 AB42    1             1 2021-01-04 2021-02-06
#> 2 AB42    1             2 2021-02-07 2021-03-07
#> 3 AB42    2             1 2021-02-02 2021-03-02
#> 4 AB42    2             2 2021-03-03 2021-04-01

adae <- tribble(
  ~USUBJID, ~ASTDT,
  "1",      "2021-01-01",
  "1",      "2021-01-05",
  "1",      "2021-02-05",
  "1",      "2021-03-05",
  "1",      "2021-04-05",
  "2",      "2021-02-15",
) %>%
  mutate(
    ASTDT = ymd(ASTDT),
    STUDYID = "AB42"
  )

derive_vars_joined(
  adae,
  dataset_add = period_ref,
  by_vars = exprs(STUDYID, USUBJID),
  join_vars = exprs(APERSDT, APEREDT),
  join_type = "all",
  filter_join = APERSDT <= ASTDT & ASTDT <= APEREDT
) %>%
  select(USUBJID, ASTDT, APERSDT, APEREDT, APERIOD)
#> # A tibble: 6 × 5
#>   USUBJID ASTDT      APERSDT    APEREDT    APERIOD
#>   <chr>   <date>     <date>     <date>       <int>
#> 1 1       2021-01-01 NA         NA              NA
#> 2 1       2021-01-05 2021-01-04 2021-02-06       1
#> 3 1       2021-02-05 2021-01-04 2021-02-06       1
#> 4 1       2021-03-05 2021-02-07 2021-03-07       2
#> 5 1       2021-04-05 NA         NA              NA
#> 6 2       2021-02-15 2021-02-02 2021-03-02       1
```

Further examples:

Further example usages of this function can be found in the `vignette("generic")`.

Equivalent examples for using the `exist_flag`, `true_value`, `false_value`, `missing_values` and `check_type` arguments can be found in `derive_vars_merged()`.

See Also

`derive_var_joined_exist_flag()`, `filter_joined()`

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_ef_msrc()`, `derive_var_merged_exist_flag()`,

derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sum derive_vars_transposed()

---

derive_vars_joined_summary

Summarize Variables from an Additional Dataset Based on Conditions from Both Datasets

---

Description

The function summarizes variables from an additional dataset and adds the summarized values as new variables to the input dataset. The selection of the observations from the additional dataset can depend on variables from both datasets. For example, all doses before the current observation can be selected and the sum be added to the input dataset.

Usage

```
derive_vars_joined_summary(
  dataset,
  dataset_add,
  by_vars = NULL,
  order = NULL,
  new_vars,
  tmp_obs_nr_var = NULL,
  join_vars = NULL,
  join_type,
  filter_add = NULL,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  filter_join = NULL,
  missing_values = NULL,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. **Permitted values** a dataset, i.e., a data.frame or tibble **Default value** none
dataset_add	Additional dataset The variables specified by the by_vars, the new_vars, the join_vars, and the order argument are expected. **Permitted values** a dataset, i.e., a data.frame or tibble

Default value none

by_vars

Grouping variables

The two datasets are joined by the specified variables.

Variables can be renamed by naming the element, i.e. by_vars = exprs(<name in input dataset> = <r
similar to the dplyr joins.

Permitted values list of (optionally named) variables created by exprs(), e.g., exprs(USUBJID, ADY = ASTDY)

Default value NULL

order

Sort order

The specified variables are used to determine the order of the records if first_cond_lower or first_cond_upper is specified or if join_type equals "before" or "after".

If an expression is named, e.g., exprs(EXSTDT = convert_dtc_to_dt(EXSTDTC), EXSEQ), a corresponding variable (EXSTDT) is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for join_vars and new_vars. The variable is not included in the output dataset.

For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

Permitted values list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) or NULL

Default value NULL

new_vars

Variables to add

The new variables can be defined by named expressions, i.e., new_vars = exprs(<new variable> = <val
The value must be defined such that it results in a single record per by group, e.g., by using a summary function like mean(), sum(), ...

Permitted values list of named expressions created by exprs(), e.g., exprs(CUMDOSA = sum(AVAL, na.rm = TRUE), AVALU = "ml")

Default value none

tmp_obs_nr_var

Temporary observation number

The specified variable is added to the input dataset (dataset) and the restricted additional dataset (dataset_add after applying filter_add). It is set to the observation number with respect to order. For each by group (by_vars) the observation number starts with 1. The variable can be used in the conditions (filter_join, first_cond_upper, first_cond_lower). It can also be used to select consecutive observations or the last observation.

The variable is not included in the output dataset. To include it specify it for new_vars.

Permitted values an unquoted symbol, e.g., AVAL

Default value NULL

join_vars

Variables to use from additional dataset

Any extra variables required from the additional dataset for filter_join should be specified for this argument. Variables specified for new_vars do not need to be repeated for join_vars. If a specified variable exists in both the input dataset and the additional dataset, the suffix ".join" is added to the variable from the additional dataset.

If an expression is named, e.g., exprs(EXSTDT = convert_dtc_to_dt(EXSTDTC)), a corresponding variable is added to the additional dataset and can be used in the filter conditions (filter_add, filter_join) and for new_vars.

The variables are not included in the output dataset.

Permitted values list of variables or named expressions created by exprs(), e.g., exprs(EXSTDY, EXSTDTM = convert_dtc_to_dtm(EXSTDTC))

Default value NULL

join_type

Observations to keep after joining

The argument determines which of the joined observations are kept with respect to the original observation. For example, if join_type = "after" is specified all observations after the original observations are kept.

Permitted values "before", "after", "all"

Default value none

filter_add

Filter for additional dataset (dataset_add)

Only observations from dataset_add fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by order or new_vars arguments can be used in the condition.

The condition can include summary functions like all() or any(). The additional dataset is grouped by the by variables (by_vars).

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

first_cond_lower

Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. If the condition is not fulfilled for any of the other observations, no observations are considered.

This argument should be specified if filter_join contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For an example see the last example below.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

first_cond_upper

Condition for selecting range of data (after)

If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the other observations, no observations are considered.

This argument should be specified if filter_join contains summary functions which should not apply to all observations but only up to the confirmation assessment. For an example see the last example below.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

filter_join — Filter for the joined dataset

The specified condition is applied to the joined dataset. Therefore variables from both datasets dataset and dataset_add can be used.

Variables created by order or new_vars arguments can be used in the condition.

The condition can include summary functions like all() or any(). The joined dataset is grouped by the original observations.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

missing_values — Values for non-matching observations

For observations of the input dataset (dataset) which do not have a matching observation in the additional dataset (dataset_add) the values of the specified variables are set to the specified value. Only variables specified for new_vars can be specified for missing_values.

Permitted values list of named expressions created by exprs(), e.g., exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))

Default value NULL

check_type — Check uniqueness?

If "message", "warning" or "error" is specified, the specified message is issued if the observations of the input dataset (dataset) or the restricted additional dataset (dataset_add after applying filter_add) are not unique with respect to the by variables and the order.

The uniqueness is checked only if tmp_obs_nr_var, first_cond_lower, or first_cond_upper is specified or join_type equals "before" or "after".

Permitted values "none", "message", "warning", "error"

Default value "warning"

Details

1. The variables specified by order are added to the additional dataset (dataset_add).
2. The variables specified by join_vars are added to the additional dataset (dataset_add).
3. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
4. The input dataset and the (restricted) additional dataset are left joined by the grouping variables (by_vars). If no grouping variables are specified, a full join is performed.
5. If first_cond_lower is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where first_cond_lower is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

   If first_cond_upper is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where first_cond_upper is fulfilled (the observation fulfilling the condition is included). If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

   For an example see the last example in the "Examples" section.

6. The joined dataset is restricted by the filter_join condition.
7. The variables specified for new_vars are created and merged to the input dataset. I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the joined dataset the new variables are set as specified by missing_values (or to NA for variables not in missing_values). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option save_memory to TRUE (see set_admiral_options()). This reduces the memory consumption but increases the run-time.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variables specified for new_vars derived from the additional dataset (dataset_add).

Examples

The examples focus on the functionality specific to this function. For examples of functionality common to all "joined" functions like filter_join, filter_add, join_vars, ... please see the examples of derive_vars_joined().

Derive cumulative dose before event (CUMDOSA):

Deriving the cumulative actual dose up to the day of the adverse event in the ADAE dataset.

- USUBJID is specified for by_vars to join the ADAE and the ADEX dataset by subject.
- filter_join is specified to restrict the ADEX dataset to the days up to the adverse event. ADY.join refers to the study day in ADEX.
- The new variable CUMDOSA is defined by the new_vars argument. It is set to the sum of AVAL.
- As ADY from ADEX is used in filter_join (but not in new_vars), it needs to be specified for join_vars.
- The join_type is set to "all" to consider all records in the joined dataset. join_type = "before" can't by used here because then doses at the same day as the adverse event would be excluded.

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

adex <- tribble(
  ~USUBJID, ~ADY, ~AVAL,
  "1",         1,    10,
  "1",         8,    20,
  "1",        15,    10,
  "2",         8,     5
)

adae <- tribble(
  ~USUBJID, ~ADY, ~AEDECOD,
  "1",         2, "Fatigue",
  "1",         9, "Influenza",
```

```
  "1",        15, "Theft",
  "1",        15, "Fatigue",
  "2",         4, "Parasomnia",
  "3",         2, "Truancy"
)

derive_vars_joined_summary(
  dataset = adae,
  dataset_add = adex,
  by_vars = exprs(USUBJID),
  filter_join = ADY.join <= ADY,
  join_type = "all",
  join_vars = exprs(ADY),
  new_vars = exprs(CUMDOSA = sum(AVAL, na.rm = TRUE))
)
#> # A tibble: 6 × 4
#>   USUBJID   ADY AEDECOD    CUMDOSA
#>   <chr>   <dbl> <chr>        <dbl>
#> 1 1           2 Fatigue         10
#> 2 1           9 Influenza       30
#> 3 1          15 Theft           40
#> 4 1          15 Fatigue         40
#> 5 2           4 Parasomnia      NA
#> 6 3           2 Truancy         NA
```

Define values for records without records in the additional dataset (`missing_values`)**:**
By default, the new variables are set to `NA` for records without matching records in the restricted additional dataset. This can be changed by specifying the `missing_values` argument.

```
derive_vars_joined_summary(
  dataset = adae,
  dataset_add = adex,
  by_vars = exprs(USUBJID),
  filter_join = ADY.join <= ADY,
  join_type = "all",
  join_vars = exprs(ADY),
  new_vars = exprs(CUMDOSE = sum(AVAL, na.rm = TRUE)),
  missing_values = exprs(CUMDOSE = 0)
)
#> # A tibble: 6 × 4
#>   USUBJID   ADY AEDECOD    CUMDOSE
#>   <chr>   <dbl> <chr>        <dbl>
#> 1 1           2 Fatigue         10
#> 2 1           9 Influenza       30
#> 3 1          15 Theft           40
#> 4 1          15 Fatigue         40
#> 5 2           4 Parasomnia       0
#> 6 3           2 Truancy          0
```

Selecting records (`join_type = "before"`, `join_type = "after"`)**:**

The join_type argument can be used to select records from the additional dataset. For example, if join_type = "before" is specified, only records before the current observation are selected. If join_type = "after" is specified, only records after the current observation are selected.

To illustrate this, a variable (SELECTED_DAYS) is derived which contains the selected days.

```
mydata <- tribble(
  ~DAY,
  1,
  2,
  3,
  4,
  5
)

derive_vars_joined_summary(
  mydata,
  dataset_add = mydata,
  order = exprs(DAY),
  join_type = "before",
  new_vars = exprs(SELECTED_DAYS = paste(DAY, collapse = ", "))
)
#> # A tibble: 5 × 2
#>     DAY SELECTED_DAYS
#>   <dbl> <chr>
#> 1     1 <NA>
#> 2     2 1
#> 3     3 1, 2
#> 4     4 1, 2, 3
#> 5     5 1, 2, 3, 4

derive_vars_joined_summary(
  mydata,
  dataset_add = mydata,
  order = exprs(DAY),
  join_type = "after",
  new_vars = exprs(SELECTED_DAYS = paste(DAY, collapse = ", "))
)
#> # A tibble: 5 × 2
#>     DAY SELECTED_DAYS
#>   <dbl> <chr>
#> 1     1 2, 3, 4, 5
#> 2     2 3, 4, 5
#> 3     3 4, 5
#> 4     4 5
#> 5     5 <NA>
```

Selecting records (first_cond_lower, first_cond_upper):

The first_cond_lower and first_cond_upper arguments can be used to restrict the joined dataset to a certain range of records. For example, if first_cond_lower is specified, the joined

dataset is restricted to the last observation before the current record where the condition is fulfilled. Please note:

- If the condition is not fulfilled for any of the records, no records are selected.
- The restriction implied by `join_type` is applied first.
- If a variable is contained in both `dataset` and `dataset_add` like `DAY` in the example below, `DAY` refers to the value from `dataset` and `DAY.join` to the value from `dataset_add`.

To illustrate this, a variable (`SELECTED_DAYS`) is derived which contains the selected days.

```
derive_vars_joined_summary(
  mydata,
  dataset_add = mydata,
  order = exprs(DAY),
  join_type = "before",
  first_cond_lower = DAY.join == 2,
  new_vars = exprs(SELECTED_DAYS = paste(sort(DAY), collapse = ", "))
)
#> # A tibble: 5 × 2
#>     DAY SELECTED_DAYS
#>   <dbl> <chr>
#> 1     1 <NA>
#> 2     2 <NA>
#> 3     3 2
#> 4     4 2, 3
#> 5     5 2, 3, 4

derive_vars_joined_summary(
  mydata,
  dataset_add = mydata,
  order = exprs(DAY),
  join_type = "after",
  first_cond_upper = DAY.join == 4,
  new_vars = exprs(SELECTED_DAYS = paste(DAY, collapse = ", "))
)
#> # A tibble: 5 × 2
#>     DAY SELECTED_DAYS
#>   <dbl> <chr>
#> 1     1 2, 3, 4
#> 2     2 3, 4
#> 3     3 4
#> 4     4 <NA>
#> 5     5 <NA>

derive_vars_joined_summary(
  mydata,
  dataset_add = mydata,
  order = exprs(DAY),
  join_type = "all",
  first_cond_lower = DAY.join == 2,
```

```
  first_cond_upper = DAY.join == 4,
  new_vars = exprs(SELECTED_DAYS = paste(sort(DAY), collapse = ", "))
)
#> # A tibble: 5 × 2
#>     DAY SELECTED_DAYS
#>   <dbl> <chr>
#> 1     1 2, 3, 4
#> 2     2 2, 3, 4
#> 3     3 2, 3, 4
#> 4     4 2, 3, 4
#> 5     5 2, 3, 4
```

Derive weekly score if enough assessments are available:

For each planned visit the average score within the week before the visit should be derived if at least three assessments are available.

Please note that the condition for the number of assessments is specified in `new_vars` and not in `filter_join`. This is because the number of assessments within the week before the visit should be counted but not the number of assessments available for the subject.

```
planned_visits <- tribble(
  ~AVISIT,  ~ADY,
  "WEEK 1",    8,
  "WEEK 4",   29,
  "WEEK 8",   57
  ) %>%
  mutate(USUBJID = "1", .before = AVISIT)

adqs <- tribble(
  ~ADY, ~AVAL,
     1,    10,
     2,    12,
     4,     9,
     5,     9,
     7,    10,
    25,    11,
    27,    10,
    29,    10,
    41,     8,
    42,     9,
    44,     5
) %>%
mutate(USUBJID = "1")

derive_vars_joined_summary(
  planned_visits,
  dataset_add = adqs,
  by_vars = exprs(USUBJID),
  filter_join = ADY - 7 <= ADY.join & ADY.join < ADY,
  join_type = "all",
```

```
  join_vars = exprs(ADY),
  new_vars = exprs(AVAL = if_else(n() >= 3, mean(AVAL, na.rm = TRUE), NA))
)
#> # A tibble: 3 × 4
#>   USUBJID AVISIT   ADY  AVAL
#>   <chr>   <chr>  <dbl> <dbl>
#> 1 1       WEEK 1     8    10
#> 2 1       WEEK 4    29    NA
#> 3 1       WEEK 8    57    NA
```

See Also

derive_vars_joined(), derive_vars_merged_summary(), derive_var_joined_exist_flag(), filter_joined()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_summary(), derive_vars_transposed()

---

derive_vars_merged *Add New Variable(s) to the Input Dataset Based on Variables from Another Dataset*

Description

Add new variable(s) to the input dataset based on variables from another dataset. The observations to merge can be selected by a condition (filter_add argument) and/or selecting the first or last observation for each by group (order and mode argument).

Usage

```
derive_vars_merged(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_vars = NULL,
  filter_add = NULL,
  mode = NULL,
  exist_flag = NULL,
  true_value = "Y",
  false_value = NA_character_,
  missing_values = NULL,
  check_type = "warning",
  duplicate_msg = NULL,
  relationship = NULL
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`dataset_add` Additional dataset

The variables specified by the `by_vars`, the `new_vars`, and the `order` argument are expected.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`by_vars` Grouping variables

The input dataset and the selected observations from the additional dataset are merged by the specified variables.

Variables can be renamed by naming the element, i.e. `by_vars = exprs(<name in input dataset> = <r`
similar to the `dplyr` joins.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`order` Sort order

If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.

Variables defined by the `new_vars` argument can be used in the sort order.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`new_vars` Variables to add

The specified variables from the additional dataset are added to the output dataset.
Variables can be renamed by naming the element, i.e., `new_vars = exprs(<new name> = <old name>)`.

For example `new_vars = exprs(var1, var2)` adds variables `var1` and `var2` from `dataset_add` to the input dataset.

And `new_vars = exprs(var1, new_var2 = old_var2)` takes `var1` and `old_var2` from `dataset_add` and adds them to the input dataset renaming `old_var2` to `new_var2`.

Values of the added variables can be modified by specifying an expression. For example, `new_vars = LASTRSP = exprs(str_to_upper(AVALC))` adds the variable `LASTRSP` to the dataset and sets it to the upper case value of `AVALC`.

If the argument is not specified or set to `NULL`, all variables from the additional dataset (`dataset_add`) are added. In the case when a variable exists in both datasets, an error is issued to ensure the user either adds to `by_vars`, removes or renames.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`filter_add` Filter for additional dataset (`dataset_add`)

Only observations fulfilling the specified condition are taken into account for merging. If the argument is not specified, all observations are considered.

Variables defined by the `new_vars` argument can be used in the filter condition.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`mode` Selection mode

Determines if the first or last observation is selected. If the `order` argument is specified, `mode` must be non-null.

If the `order` argument is not specified, the `mode` argument is ignored.

Permitted values `"first"`, `"last"`

Default value `NULL`

`exist_flag` Exist flag

If the argument is specified (e.g., `exist_flag = FLAG`), the specified variable (e.g., `FLAG`) is added to the input dataset. This variable will be the value provided in `true_value` for all selected records from `dataset_add` which are merged into the input dataset, and the value provided in `false_value` otherwise.

Permitted values an unquoted symbol, e.g., `AVAL`

Default value `NULL`

`true_value` True value

The value for the specified variable `exist_flag`, applicable to the first or last observation (depending on the mode) of each by group.

Permitted values a character scalar, i.e., a character vector of length one

Default value `"Y"`

`false_value` False value

The value for the specified variable `exist_flag`, NOT applicable to the first or last observation (depending on the mode) of each by group.

Permitted values a character scalar, i.e., a character vector of length one

Default value `NA_character_`

`missing_values` Values for non-matching observations

For observations of the input dataset (`dataset`) which do not have a matching observation in the additional dataset (`dataset_add`) the values of the specified variables are set to the specified value. Only variables specified for `new_vars` can be specified for `missing_values`.

Permitted values list of named expressions created by `exprs()`, e.g., `exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))`

Default value `NULL`

check_type — Check uniqueness?

If "warning", "message", or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order.

If the order argument is not specified, the check_type argument is ignored: if the observations of the (restricted) additional dataset are not unique with respect to the by variables, an error is issued.

Permitted values "none", "message", "warning", "error"

Default value "warning"

duplicate_msg — Message of unique check

If the uniqueness check fails, the specified message is displayed.

Permitted values a console message to be printed, e.g. "Attention" or for longer messages use paste("Line 1", "Line 2")

Default value
```
paste(
  "Dataset {.arg dataset_add} contains duplicate records with respect to",
  "{.var {vars2chr(by_vars)}}."
)
```

relationship — Expected merge-relationship between the by_vars variable(s) in dataset (input dataset) and the dataset_add (additional dataset) containing the additional new_vars.

This argument is passed to the dplyr::left_join() function. See https://dplyr.tidyverse.org/reference/mutate-joins.html#arguments for more details.

Permitted values "one-to-one", "many-to-one"

Default value NULL

Details

1. The new variables (new_vars) are added to the additional dataset (dataset_add).
2. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
3. If order is specified, for each by group the first or last observation (depending on mode) is selected.
4. The variables specified for new_vars are merged to the input dataset using left_join(). I.e., the output dataset contains all observations from the input dataset. For observations without a matching observation in the additional dataset the new variables are set as specified by missing_values (or to NA for variables not in missing_values). Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variables specified for new_vars from the additional dataset (dataset_add).

Examples

Note on usage versus `derive_vars_joined()`:

The question between using `derive_vars_merged()` or the more powerful `derive_vars_joined()` comes down to how you need to select the observations to be merged.

- If the observations from `dataset_add` to merge can be selected by a condition (`filter_add`) using *only* variables from `dataset_add`, then always use `derive_vars_merged()` as it requires less resources (time and memory). A common example of this would be a randomization date in `ADSL`, where you are simply merging on a date from `DS` according to a certain `DSDECOD` condition such as `DSDECOD == "RANDOMIZATION"`.
- However, if the selection of the observations from `dataset_add` can depend on variables from *both* datasets, then use `derive_vars_joined()`. An example of this would be assigning period variables from `ADSL` to an `ADAE`, where you now need to check each adverse event start date against the period start and end dates to decide which period value to join.

Basic merge of a full dataset:

Merge all demographic variables onto a vital signs dataset.

- The variable `DOMAIN` exists in both datasets so note the use of `select(dm, -DOMAIN)` in the `dataset_add` argument. Without this an error would be issued to notify the user.

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
  ~DOMAIN,  ~USUBJID, ~VSTESTCD, ~VISIT,      ~VSSTRESN, ~VSDTC,
  "VS",     "01",     "HEIGHT",  "SCREENING",     178.0, "2013-08-20",
  "VS",     "01",     "WEIGHT",  "SCREENING",      81.9, "2013-08-20",
  "VS",     "01",     "WEIGHT",  "BASELINE",       82.1, "2013-08-29",
  "VS",     "01",     "WEIGHT",  "WEEK 2",         81.9, "2013-09-15",
  "VS",     "01",     "WEIGHT",  "WEEK 4",         82.6, "2013-09-24",
  "VS",     "02",     "WEIGHT",  "BASELINE",       58.6, "2014-01-11"
) %>%
  mutate(STUDYID = "AB42")

dm <- tribble(
  ~DOMAIN, ~USUBJID, ~AGE, ~AGEU,
  "DM",    "01",       61, "YEARS",
  "DM",    "02",       64, "YEARS",
  "DM",    "03",       85, "YEARS"
) %>%
  mutate(STUDYID = "AB42")

derive_vars_merged(
  vs,
  dataset_add = select(dm, -DOMAIN),
  by_vars = exprs(STUDYID, USUBJID)
) %>%
  select(USUBJID, VSTESTCD, VISIT, VSSTRESN, AGE, AGEU)
#> # A tibble: 6 × 6
#>   USUBJID VSTESTCD VISIT     VSSTRESN   AGE AGEU
```

```
#>   <chr>   <chr>    <chr>        <dbl> <dbl> <chr>
#> 1 01      HEIGHT   SCREENING    178      61 YEARS
#> 2 01      WEIGHT   SCREENING     81.9    61 YEARS
#> 3 01      WEIGHT   BASELINE      82.1    61 YEARS
#> 4 01      WEIGHT   WEEK 2        81.9    61 YEARS
#> 5 01      WEIGHT   WEEK 4        82.6    61 YEARS
#> 6 02      WEIGHT   BASELINE      58.6    64 YEARS
```

Merge only the first/last value (`order` **and** `mode`)**:**

Merge the last occurring weight for each subject to the demographics dataset.

- To enable sorting by visit date `convert_dtc_to_dtm()` is used to convert to a datetime, within the `order` argument.
- Then the `mode` argument is set to `"last"` to ensure the last sorted value is taken. Be cautious if `NA` values are possible in the `order` variables - see Sort Order.
- The `filter_add` argument is used to restrict the vital signs records only to weight assessments.

```
derive_vars_merged(
  dm,
  dataset_add = vs,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(convert_dtc_to_dtm(VSDTC)),
  mode = "last",
  new_vars = exprs(LSTWT = VSSTRESN),
  filter_add = VSTESTCD == "WEIGHT"
) %>%
  select(USUBJID, AGE, AGEU, LSTWT)
#> # A tibble: 3 × 4
#>   USUBJID   AGE AGEU  LSTWT
#>   <chr>   <dbl> <chr> <dbl>
#> 1 01         61 YEARS  82.6
#> 2 02         64 YEARS  58.6
#> 3 03         85 YEARS  NA
```

Handling duplicates (`check_type`)**:**

The source records are checked regarding duplicates with respect to the by variables and the order specified. By default, a warning is issued if any duplicates are found. Note the results here with a new vital signs dataset containing a duplicate last weight assessment date.

```
vs_dup <- tribble(
  ~DOMAIN,  ~USUBJID, ~VSTESTCD, ~VISIT,      ~VSSTRESN, ~VSDTC,
  "VS",     "01",     "WEIGHT",  "WEEK 2",        81.1, "2013-09-24",
  "VS",     "01",     "WEIGHT",  "WEEK 4",        82.6, "2013-09-24"
) %>%
  mutate(STUDYID = "AB42")

derive_vars_merged(
  dm,
  dataset_add = vs_dup,
```

```
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(convert_dtc_to_dtm(VSDTC)),
  mode = "last",
  new_vars = exprs(LSTWT = VSSTRESN),
  filter_add = VSTESTCD == "WEIGHT"
) %>%
  select(USUBJID, AGE, AGEU, LSTWT)
#> # A tibble: 3 × 4
#>   USUBJID   AGE AGEU  LSTWT
#>   <chr>   <dbl> <chr> <dbl>
#> 1 01         61 YEARS  82.6
#> 2 02         64 YEARS  NA
#> 3 03         85 YEARS  NA
#> Warning: Dataset contains duplicate records with respect to `STUDYID`, `USUBJID`, and
#> `convert_dtc_to_dtm(VSDTC)`
#> i Run `admiral::get_duplicates_dataset()` to access the duplicate records
```

For investigating the issue, the dataset of the duplicate source records can be obtained by calling `get_duplicates_dataset()`:

```
get_duplicates_dataset()
#> Duplicate records with respect to `STUDYID`, `USUBJID`, and
#> `convert_dtc_to_dtm(VSDTC)`.
#> # A tibble: 2 × 9
#>  STUDYID USUBJID convert_dtc_to_dtm(VSDT. . . ¹ DOMAIN VSTESTCD VISIT VSSTRESN VSDTC
#> * <chr>   <chr>   <dttm>                  <chr>  <chr>    <chr>    <dbl> <chr>
#> 1 AB42   01    2013-09-24 00:00:00     VS    WEIGHT  WEEK. . .    81.1 2013. . .
#> 2 AB42   01    2013-09-24 00:00:00     VS    WEIGHT  WEEK. . .    82.6 2013. . .
#> # i abbreviated name: ¹`convert_dtc_to_dtm(VSDTC)`
#> # i 1 more variable: LSTWT <dbl>
```

Common options to solve the issue:

- Specifying additional variables for `order` - this is the most common approach, adding something like a sequence variable.
- Restricting the source records by specifying/updating the `filter_add` argument.
- Setting `check_type = "none"` to ignore any duplicates, but then in this case the last occurring record would be chosen according to the sort order of the input `dataset_add`. This is not often advisable, unless the order has no impact on the result, as the temporary sort order can be prone to variation across an ADaM script.

Modify values dependent on the merge (`new_vars` and `missing_values`):

For the last occurring weight for each subject, add a categorization of which visit it occurred at to the demographics dataset.

- In the `new_vars` argument, other functions can be utilized to modify the merged values. For example, in the below case we want to categorize the visit as `"BASELINE"` or `"POST-BASELINE"` using `if_else()`.
- The `missing_values` argument assigns a specific value for subjects with no matching observations - see subject `"03"` in the below example.

```
derive_vars_merged(
  dm,
  dataset_add = vs,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(convert_dtc_to_dtm(VSDTC)),
  mode = "last",
  new_vars = exprs(
    LSTWTCAT = if_else(VISIT == "BASELINE", "BASELINE", "POST-BASELINE")
  ),
  filter_add = VSTESTCD == "WEIGHT",
  missing_values = exprs(LSTWTCAT = "MISSING")
) %>%
  select(USUBJID, AGE, AGEU, LSTWTCAT)
#> # A tibble: 3 × 4
#>   USUBJID   AGE AGEU  LSTWTCAT
#>   <chr>   <dbl> <chr> <chr>
#> 1 01         61 YEARS POST-BASELINE
#> 2 02         64 YEARS BASELINE
#> 3 03         85 YEARS MISSING
```

Check existence of records to merge (`exist_flag`, `true_value` and `false_value`):

Similar to the above example, now we prefer to have a separate flag variable to show whether a selected record was merged.

- The name of the new variable is set with the `exist_flag` argument.
- The values of this new variable are assigned via the `true_value` and `false_value` arguments.

```
derive_vars_merged(
  dm,
  dataset_add = vs,
  by_vars = exprs(STUDYID, USUBJID),
  order = exprs(convert_dtc_to_dtm(VSDTC)),
  mode = "last",
  new_vars = exprs(
    LSTWTCAT = if_else(VISIT == "BASELINE", "BASELINE", "POST-BASELINE")
  ),
  filter_add = VSTESTCD == "WEIGHT",
  exist_flag = WTCHECK,
  true_value = "Y",
  false_value = "MISSING"
) %>%
  select(USUBJID, AGE, AGEU, LSTWTCAT, WTCHECK)
#> # A tibble: 3 × 5
#>   USUBJID   AGE AGEU  LSTWTCAT      WTCHECK
#>   <chr>   <dbl> <chr> <chr>         <chr>
#> 1 01         61 YEARS POST-BASELINE Y
#> 2 02         64 YEARS BASELINE      Y
#> 3 03         85 YEARS <NA>          MISSING
```

Creating more than one variable from the merge (`new_vars`)**:**
Derive treatment start datetime and associated imputation flags.

- In this example we first impute exposure datetime and associated flag variables as a separate first step to be used in the `order` argument.
- In the `new_vars` arguments, you can see how both datetime and the date and time imputation flags are all merged in one call.

```
ex <- tribble(
  ~DOMAIN, ~USUBJID, ~EXSTDTC,
  "EX",    "01",     "2013-08-29",
  "EX",    "01",     "2013-09-16",
  "EX",    "02",     "2014-01-11",
  "EX",    "02",     "2014-01-25"
) %>%
  mutate(STUDYID = "AB42")

ex_ext <- derive_vars_dtm(
  ex,
  dtc = EXSTDTC,
  new_vars_prefix = "EXST",
  highest_imputation = "M"
)

derive_vars_merged(
  dm,
  dataset_add = ex_ext,
  by_vars = exprs(STUDYID, USUBJID),
  new_vars = exprs(TRTSDTM = EXSTDTM, TRTSDTF = EXSTDTF, TRTSTMF = EXSTTMF),
  order = exprs(EXSTDTM),
  mode = "first"
) %>%
  select(USUBJID, TRTSDTM, TRTSDTF, TRTSTMF)
#> # A tibble: 3 × 4
#>   USUBJID TRTSDTM             TRTSDTF TRTSTMF
#>   <chr>   <dttm>              <chr>   <chr>
#> 1 01      2013-08-29 00:00:00 <NA>    H
#> 2 02      2014-01-11 00:00:00 <NA>    H
#> 3 03      NA                  <NA>    <NA>
```

Further examples:
Further example usages of this function can be found in the `vignette("generic")`.

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: `derive_var_extreme_flag()`, `derive_var_joined_exist_flag()`, `derive_var_merged_ef_msrc()`, `derive_var_merged_exist_flag()`, `derive_var_obs_number()`, `derive_var_relative_flag()`, `derive_vars_cat()`, `derive_vars_computed()`, `derive_vars_joined()`, `derive_vars_joined_summary()`, `derive_vars_merged_lookup()`, `derive_vars_merged_sur`
`derive_vars_transposed()`

derive_vars_merged_lookup
Merge Lookup Table with Source Dataset

Description

Merge user-defined lookup table with the input dataset. Optionally print a list of records from the input dataset that do not have corresponding mapping from the lookup table.

Usage

```
derive_vars_merged_lookup(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_vars = NULL,
  mode = NULL,
  filter_add = NULL,
  check_type = "warning",
  duplicate_msg = NULL,
  print_not_mapped = TRUE
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`dataset_add` Lookup table

The variables specified by the `by_vars` argument are expected.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`by_vars` Grouping variables

The input dataset and the selected observations from the additional dataset are merged by the specified variables.

Variables can be renamed by naming the element, i.e. `by_vars = exprs(<name in input dataset> = <r` similar to the `dplyr` joins.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

order

Sort order

If the argument is set to a non-null value, for each by group the first or last observation from the additional dataset is selected with respect to the specified order.

Variables defined by the new_vars argument can be used in the sort order.

For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value NULL

new_vars

Variables to add

The specified variables from the additional dataset are added to the output dataset. Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>).

For example new_vars = exprs(var1, var2) adds variables var1 and var2 from dataset_add to the input dataset.

And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2 from dataset_add and adds them to the input dataset renaming old_var2 to new_var2.

Values of the added variables can be modified by specifying an expression. For example, new_vars = LASTRSP = exprs(str_to_upper(AVALC)) adds the variable LASTRSP to the dataset and sets it to the upper case value of AVALC.

If the argument is not specified or set to NULL, all variables from the additional dataset (dataset_add) are added. In the case when a variable exists in both datasets, an error is issued to ensure the user either adds to by_vars, removes or renames.

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value NULL

mode

Selection mode

Determines if the first or last observation is selected. If the order argument is specified, mode must be non-null.

If the order argument is not specified, the mode argument is ignored.

Permitted values "first", "last"

Default value NULL

filter_add

Filter for additional dataset (dataset_add)

Only observations fulfilling the specified condition are taken into account for merging. If the argument is not specified, all observations are considered.

Variables defined by the new_vars argument can be used in the filter condition.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

check_type

Check uniqueness?

If "warning", "message", or "error" is specified, the specified message is issued if the observations of the (restricted) additional dataset are not unique with respect to the by variables and the order.

If the order argument is not specified, the check_type argument is ignored: if the observations of the (restricted) additional dataset are not unique with respect to the by variables, an error is issued.

Permitted values "none", "message", "warning", "error"

Default value "warning"

duplicate_msg Message of unique check

If the uniqueness check fails, the specified message is displayed.

Permitted values a console message to be printed, e.g. "Attention" or for longer messages use paste("Line 1", "Line 2")

Default value
```
paste(
  "Dataset {.arg dataset_add} contains duplicate records with respect to",
  "{.var {vars2chr(by_vars)}}."
)
```

print_not_mapped

Print a list of unique by_vars values that do not have corresponding records from the lookup table?

Permitted values TRUE, FALSE

Default value TRUE

Value

The output dataset contains all observations and variables of the input dataset, and add the variables specified in new_vars from the lookup table specified in dataset_add. Optionally prints a list of unique by_vars values that do not have corresponding records from the lookup table (by specifying print_not_mapped = TRUE).

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_summary(), derive_vars_transposed()

Examples

```
library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,        ~VISIT, ~VSTESTCD,       ~VSTEST,
  "PILOT01",    "VS", "01-1028",   "SCREENING",  "HEIGHT",      "Height",
  "PILOT01",    "VS", "01-1028",   "SCREENING",    "TEMP", "Temperature",
  "PILOT01",    "VS", "01-1028",    "BASELINE",    "TEMP", "Temperature",
  "PILOT01",    "VS", "01-1028",      "WEEK 4",    "TEMP", "Temperature",
  "PILOT01",    "VS", "01-1028", "SCREENING 1",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "01-1028",    "BASELINE",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "01-1028",      "WEEK 4",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "04-1325",   "SCREENING",  "HEIGHT",      "Height",
  "PILOT01",    "VS", "04-1325",   "SCREENING",    "TEMP", "Temperature",
```

```
  "PILOT01",    "VS", "04-1325",    "BASELINE",    "TEMP", "Temperature",
  "PILOT01",    "VS", "04-1325",      "WEEK 4",    "TEMP", "Temperature",
  "PILOT01",    "VS", "04-1325", "SCREENING 1",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "04-1325",    "BASELINE",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "04-1325",      "WEEK 4",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "10-1027",   "SCREENING",  "HEIGHT",      "Height",
  "PILOT01",    "VS", "10-1027",   "SCREENING",    "TEMP", "Temperature",
  "PILOT01",    "VS", "10-1027",    "BASELINE",    "TEMP", "Temperature",
  "PILOT01",    "VS", "10-1027",      "WEEK 4",    "TEMP", "Temperature",
  "PILOT01",    "VS", "10-1027", "SCREENING 1",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "10-1027",    "BASELINE",  "WEIGHT",      "Weight",
  "PILOT01",    "VS", "10-1027",      "WEEK 4",  "WEIGHT",      "Weight"
)

param_lookup <- tribble(
  ~VSTESTCD,                 ~VSTEST, ~PARAMCD,                       ~PARAM,
  "SYSBP", "Systolic Blood Pressure",  "SYSBP", "Syst Blood Pressure (mmHg)",
  "WEIGHT",                 "Weight", "WEIGHT",                "Weight (kg)",
  "HEIGHT",                 "Height", "HEIGHT",                "Height (cm)",
  "TEMP",              "Temperature",   "TEMP",            "Temperature (C)",
  "MAP",    "Mean Arterial Pressure",    "MAP",   "Mean Art Pressure (mmHg)",
  "BMI",            "Body Mass Index",    "BMI",    "Body Mass Index(kg/m^2)",
  "BSA",         "Body Surface Area",    "BSA",     "Body Surface Area(m^2)"
)

derive_vars_merged_lookup(
  dataset = vs,
  dataset_add = param_lookup,
  by_vars = exprs(VSTESTCD),
  new_vars = exprs(PARAMCD, PARAM),
  print_not_mapped = TRUE
)
```

`derive_vars_merged_summary` *Merge Summary Variables*

Description

Merge a summary variable from a dataset to the input dataset.

Usage

```
derive_vars_merged_summary(
  dataset,
  dataset_add,
  by_vars,
  new_vars = NULL,
  filter_add = NULL,
  missing_values = NULL
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`dataset_add` Additional dataset

The variables specified by the `by_vars` and the variables used on the left hand sides of the `new_vars` arguments are expected.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`by_vars` Grouping variables

The expressions on the left hand sides of `new_vars` are evaluated by the specified *variables*. Then the resulting values are merged to the input dataset (`dataset`) by the specified *variables*.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`new_vars` New variables to add

The specified variables are added to the input dataset.

A named list of expressions is expected:

- LHS refer to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for `by_vars`.

For example:

```
new_vars = exprs(
  DOSESUM = sum(AVAL),
  DOSEMEAN = mean(AVAL)
)
```

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`filter_add` Filter for additional dataset (`dataset_add`)

Only observations fulfilling the specified condition are taken into account for summarizing. If the argument is not specified, all observations are considered.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`missing_values` Values for non-matching observations

For observations of the input dataset (`dataset`) which do not have a matching observation in the additional dataset (`dataset_add`) the values of the specified variables are set to the specified value. Only variables specified for `new_vars` can be specified for `missing_values`.

Permitted values list of named expressions created by exprs(), e.g., exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))

Default value NULL

Details

1. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
2. The new variables (new_vars) are created for each by group (by_vars) in the additional dataset (dataset_add) by calling summarize(). I.e., all observations of a by group are summarized to a single observation.
3. The new variables are merged to the input dataset. For observations without a matching observation in the additional dataset the new variables are set to NA. Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variables specified for new_vars.

Examples

Data setup:

The following examples use the BDS dataset below as a basis.

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

adbds <- tribble(
  ~USUBJID,  ~AVISIT,  ~ASEQ, ~AVAL,
  "1",      "WEEK 1",      1,    10,
  "1",      "WEEK 1",      2,    NA,
  "1",      "WEEK 2",      3,    NA,
  "1",      "WEEK 3",      4,    42,
  "1",      "WEEK 4",      5,    12,
  "1",      "WEEK 4",      6,    12,
  "1",      "WEEK 4",      7,    15,
  "2",      "WEEK 1",      1,    21,
  "2",      "WEEK 4",      2,    22
)
```

Summarize one or more variables using summary functions (new_vars):

The new_vars argument specifies a named list of expressions where the right-hand side uses summary functions (e.g. mean(), sum(), max()) to aggregate values from dataset_add within each by group. Multiple summary variables can be added in a single call.

In the example below, the mean and sum of AVAL within each subject and visit are derived and merged back onto the input dataset:

```
derive_vars_merged_summary(
  adbds,
  dataset_add = adbds,
  by_vars = exprs(USUBJID, AVISIT),
  new_vars = exprs(
    MEANVIS = mean(AVAL, na.rm = TRUE),
    SUMVIS = sum(AVAL, na.rm = TRUE)
  )
)
#> # A tibble: 9 × 6
#>   USUBJID AVISIT  ASEQ  AVAL MEANVIS SUMVIS
#>   <chr>   <chr>  <dbl> <dbl>   <dbl>  <dbl>
#> 1 1       WEEK 1     1    10      10     10
#> 2 1       WEEK 1     2    NA      10     10
#> 3 1       WEEK 2     3    NA     NaN      0
#> 4 1       WEEK 3     4    42      42     42
#> 5 1       WEEK 4     5    12      13     39
#> 6 1       WEEK 4     6    12      13     39
#> 7 1       WEEK 4     7    15      13     39
#> 8 2       WEEK 1     1    21      21     21
#> 9 2       WEEK 4     2    22      22     22
```

In the example above, subject "1" at "WEEK 2" has only missing AVAL values, so MEANVIS is NaN (the result of mean(NA, na.rm = TRUE)) and SUMVIS is 0. Note that missing_values cannot be used here because the by group *exists* in dataset_add — it merely produces NaN. To coerce NaN to NA, apply a follow-up mutate() step:

```
derive_vars_merged_summary(
  adbds,
  dataset_add = adbds,
  by_vars = exprs(USUBJID, AVISIT),
  new_vars = exprs(
    MEANVIS = mean(AVAL, na.rm = TRUE),
    SUMVIS = sum(AVAL, na.rm = TRUE)
  )
) %>%
  mutate(MEANVIS = ifelse(is.nan(MEANVIS), NA_real_, MEANVIS))
#> # A tibble: 9 × 6
#>   USUBJID AVISIT  ASEQ  AVAL MEANVIS SUMVIS
#>   <chr>   <chr>  <dbl> <dbl>   <dbl>  <dbl>
#> 1 1       WEEK 1     1    10      10     10
#> 2 1       WEEK 1     2    NA      10     10
#> 3 1       WEEK 2     3    NA      NA      0
#> 4 1       WEEK 3     4    42      42     42
#> 5 1       WEEK 4     5    12      13     39
#> 6 1       WEEK 4     6    12      13     39
#> 7 1       WEEK 4     7    15      13     39
#> 8 2       WEEK 1     1    21      21     21
#> 9 2       WEEK 4     2    22      22     22
```

Subject "1" at "WEEK 2" now has MEANVIS = NA instead of NaN.

Restricting source records (filter_add):

The filter_add argument restricts the records in dataset_add that are used for the summarization. Only records satisfying the filter condition contribute to the summary values. This can be useful, for example, to compute a summary statistic based only on records before or after a certain time point.

In the following example, the mean of AVAL is computed only for records with a positive study day (ADY > 0), and the result is merged onto the ADSL-like dataset. Subject "2" has no records with ADY > 0, so MEANPBL is NA for that subject.

```
adsl <- tribble(
  ~USUBJID,
  "1",
  "2",
  "3"
)

adbds2 <- tribble(
  ~USUBJID, ~ADY, ~AVAL,
  "1",        -3,    10,
  "1",         2,    12,
  "1",         8,    15,
  "3",         4,    42
)

derive_vars_merged_summary(
  adsl,
  dataset_add = adbds2,
  by_vars = exprs(USUBJID),
  new_vars = exprs(MEANPBL = mean(AVAL, na.rm = TRUE)),
  filter_add = ADY > 0
)
#> # A tibble: 3 × 2
#>   USUBJID MEANPBL
#>   <chr>     <dbl>
#> 1 1          13.5
#> 2 2          NA
#> 3 3          42
```

Handling non-matching observations (missing_values):

By default, records in dataset with no matching by group in dataset_add receive NA for the new variables. The missing_values argument allows you to specify a different value for these non-matching records.

A natural use-case is counting observations per subject and defaulting to 0 (rather than NA) for subjects with no matching records. In the example below, the number of distinct post-baseline visits per subject is derived from adbds and merged onto adsl. Subject "3" has no records in adbds, so without missing_values the new variable would be NA; setting missing_values = exprs(NVIS = 0) makes the count meaningful for all subjects:

```
derive_vars_merged_summary(
  adsl,
  dataset_add = adbds,
  by_vars = exprs(USUBJID),
  new_vars = exprs(NVIS = n_distinct(AVISIT)),
  missing_values = exprs(NVIS = 0)
)
#> # A tibble: 3 × 2
#>   USUBJID  NVIS
#>   <chr>   <dbl>
#> 1 1           4
#> 2 2           2
#> 3 3           0
```

Renaming by variables (by_vars):

The by_vars argument supports renaming, using the syntax exprs(<left_name> = <right_name>), where <left_name> is the variable name in dataset and <right_name> is the corresponding variable in dataset_add. This is useful when the grouping variable has different names in the two datasets.

In the example below the input dataset uses AVISIT while the additional dataset uses VISIT for the same concept. The by_vars argument maps them together so the merge can proceed correctly:

```
adbds_renamed <- adbds %>% rename(VISIT = AVISIT)

derive_vars_merged_summary(
  adbds,
  dataset_add = adbds_renamed,
  by_vars = exprs(USUBJID, AVISIT = VISIT),
  new_vars = exprs(MEANVIS = mean(AVAL, na.rm = TRUE))
)
#> # A tibble: 9 × 5
#>   USUBJID AVISIT  ASEQ  AVAL MEANVIS
#>   <chr>   <chr>  <dbl> <dbl>   <dbl>
#> 1 1       WEEK 1     1    10      10
#> 2 1       WEEK 1     2    NA      10
#> 3 1       WEEK 2     3    NA     NaN
#> 4 1       WEEK 3     4    42      42
#> 5 1       WEEK 4     5    12      13
#> 6 1       WEEK 4     6    12      13
#> 7 1       WEEK 4     7    15      13
#> 8 2       WEEK 1     1    21      21
#> 9 2       WEEK 4     2    22      22
```

String aggregation:

Summary expressions are not restricted to numeric aggregations. Any expression that reduces a group to a single value is permitted. For example, paste(..., collapse = ", ") can be used to concatenate character values within a by group into a single string.

In the example below, the lesion identifiers observed at baseline for each subject are collected into a single comma-separated string and merged onto the ADSL dataset:

```
adtr <- tribble(
  ~USUBJID,     ~AVISIT, ~LESIONID,
  "1",        "BASELINE",  "INV-T1",
  "1",        "BASELINE",  "INV-T2",
  "1",        "BASELINE",  "INV-T3",
  "1",        "BASELINE",  "INV-T4",
  "1",          "WEEK 1",  "INV-T1",
  "1",          "WEEK 1",  "INV-T2",
  "1",          "WEEK 1",  "INV-T4",
  "2",        "BASELINE",  "INV-T1",
  "2",        "BASELINE",  "INV-T2",
  "2",        "BASELINE",  "INV-T3",
  "2",          "WEEK 1",  "INV-T1",
  "2",          "WEEK 1",  "INV-N1"
)

derive_vars_merged_summary(
  adsl,
  dataset_add = adtr,
  by_vars = exprs(USUBJID),
  filter_add = AVISIT == "BASELINE",
  new_vars = exprs(LESIONSBL = paste(LESIONID, collapse = ", "))
)
#> # A tibble: 3 × 2
#>   USUBJID LESIONSBL
#>   <chr>   <chr>
#> 1 1       INV-T1, INV-T2, INV-T3, INV-T4
#> 2 2       INV-T1, INV-T2, INV-T3
#> 3 3       <NA>
```

See Also

derive_summary_records(), get_summary_records()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_transposed()

---

derive_vars_period *Add Subperiod, Period, or Phase Variables to ADSL*

Description

The function adds subperiod, period, or phase variables like P01S1SDT, P01S2SDT, AP01SDTM, AP02SDTM, TRT01A, TRT02A, PH1SDT, PH2SDT, ... to the input dataset. The values of the variables are defined by a period reference dataset which has one observations per patient and subperiod, period, or phase.

Usage

```
derive_vars_period(
  dataset,
  dataset_ref,
  new_vars,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

dataset	Input dataset The variables specified by the subject_keys argument are expected to be in the dataset. **Default value** none
dataset_ref	Period reference dataset The variables specified by new_vars and subject_keys are expected. If subperiod variables are requested, APERIOD and ASPER are expected. If period variables are requested. APERIOD is expected. If phase variables are requested, APHASEN is expected. **Default value** none
new_vars	New variables A named list of variables like exprs(PHwSDT = PHSDT, PHwEDT = PHEDT, APHASEw = APHASE) is expected. The left hand side of the elements defines a set of variables (in CDISC notation) to be added to the output dataset. The right hand side defines the source variable from the period reference dataset. If the lower case letter "w" is used it refers to a phase variable, if the lower case letters "xx" are used it refers to a period variable, and if both "xx" and "w" are used it refers to a subperiod variable. Only one type must be used, e.g., all left hand side values must refer to period variables. It is not allowed to mix for example period and subperiod variables. If period *and* subperiod variables are required, separate calls must be used. **Default value** none
subject_keys	Variables to uniquely identify a subject A list of expressions where the expressions are symbols as returned by exprs() is expected. **Default value** get_admiral_option("subject_keys")

Details

For each subperiod/period/phase in the period reference dataset and each element in new_vars a variable (LHS value of new_vars) is added to the output dataset and set to the value of the source variable (RHS value of new_vars.

Value

The input dataset with subperiod/period/phase variables added (see "Details" section)

See Also

create_period_dataset()

ADSL Functions that returns variable appended to dataset: derive_var_age_years(), derive_vars_aage(), derive_vars_extreme_event()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

adsl <- tibble(STUDYID = "xyz", USUBJID = c("1", "2"))

# Add period variables to ADSL
period_ref <- tribble(
  ~USUBJID, ~APERIOD, ~APERSDT,     ~APEREDT,
  "1",             1, "2021-01-04", "2021-02-06",
  "1",             2, "2021-02-07", "2021-03-07",
  "2",             1, "2021-02-02", "2021-03-02",
  "2",             2, "2021-03-03", "2021-04-01"
) %>%
  mutate(
    STUDYID = "xyz",
    APERIOD = as.integer(APERIOD),
    across(matches("APER[ES]DT"), ymd)
  )

derive_vars_period(
  adsl,
  dataset_ref = period_ref,
  new_vars = exprs(APxxSDT = APERSDT, APxxEDT = APEREDT)
) %>%
  select(STUDYID, USUBJID, AP01SDT, AP01EDT, AP02SDT, AP02EDT)

# Add phase variables to ADSL
phase_ref <- tribble(
  ~USUBJID, ~APHASEN, ~PHSDT,       ~PHEDT,       ~APHASE,
  "1",             1, "2021-01-04", "2021-02-06", "TREATMENT",
  "1",             2, "2021-02-07", "2021-03-07", "FUP",
  "2",             1, "2021-02-02", "2021-03-02", "TREATMENT"
) %>%
  mutate(
    STUDYID = "xyz",
    APHASEN = as.integer(APHASEN),
    across(matches("PH[ES]DT"), ymd)
  )

derive_vars_period(
  adsl,
  dataset_ref = phase_ref,
  new_vars = exprs(PHwSDT = PHSDT, PHwEDT = PHEDT, APHASEw = APHASE)
) %>%
```

```
  select(STUDYID, USUBJID, PH1SDT, PH1EDT, PH2SDT, PH2EDT, APHASE1, APHASE2)

# Add subperiod variables to ADSL
subperiod_ref <- tribble(
  ~USUBJID, ~APERIOD, ~ASPER, ~ASPRSDT,     ~ASPREDT,
  "1",             1,      1, "2021-01-04", "2021-01-19",
  "1",             1,      2, "2021-01-20", "2021-02-06",
  "1",             2,      1, "2021-02-07", "2021-03-07",
  "2",             1,      1, "2021-02-02", "2021-03-02",
  "2",             2,      1, "2021-03-03", "2021-04-01"
) %>%
  mutate(
    STUDYID = "xyz",
    APERIOD = as.integer(APERIOD),
    ASPER = as.integer(ASPER),
    across(matches("ASPR[ES]DT"), ymd)
  )

derive_vars_period(
  adsl,
  dataset_ref = subperiod_ref,
  new_vars = exprs(PxxSwSDT = ASPRSDT, PxxSwEDT = ASPREDT)
) %>%
  select(STUDYID, USUBJID, P01S1SDT, P01S1EDT, P01S2SDT, P01S2EDT, P02S1SDT, P02S1EDT)
```

`derive_vars_query` *Derive Query Variables*

Description

Derive Query Variables

Usage

```
derive_vars_query(dataset, dataset_queries)
```

Arguments

`dataset`	Input dataset **Default value** none
`dataset_queries`	A dataset containing required columns `PREFIX`, `GRPNAME`, `SRCVAR`, `TERMCHAR` and/or `TERMNUM`, and optional columns `GRPID`, `SCOPE`, `SCOPEN`. `create_query_data()` can be used to create the dataset. **Default value** none

Details

This function can be used to derive CDISC variables such as SMQzzNAM, SMQzzCD, SMQzzSC, SMQzzSCN, and CQzzNAM in ADAE and ADMH, and variables such as SDGzzNAM, SDGzzCD, and SDGzzSC in ADCM. An example usage of this function can be found in the vignette("occds").

A query dataset is expected as an input to this function. See the vignette("queries_dataset") for descriptions, or call data("queries") for an example of a query dataset.

For each unique element in PREFIX, the corresponding "NAM" variable will be created. For each unique PREFIX, if GRPID is not "" or NA, then the corresponding "CD" variable is created; similarly, if SCOPE is not "" or NA, then the corresponding "SC" variable will be created; if SCOPEN is not "" or NA, then the corresponding "SCN" variable will be created.

For each record in dataset, the "NAM" variable takes the value of GRPNAME if the value of TERMCHAR or TERMNUM in dataset_queries matches the value of the respective SRCVAR in dataset. Note that TERMCHAR in dataset_queries dataset may be NA only when TERMNUM is non-NA and vice versa. The matching is case insensitive. The "CD", "SC", and "SCN" variables are derived accordingly based on GRPID, SCOPE, and SCOPEN respectively, whenever not missing.

Value

The input dataset with query variables derived.

See Also

create_query_data()

OCCDS Functions: derive_var_trtemfl(), derive_vars_atc()

Examples

```
library(tibble)
data("queries")
adae <- tribble(
  ~USUBJID, ~ASTDTM, ~AETERM, ~AESEQ, ~AEDECOD, ~AELLT, ~AELLTCD,
  "01", "2020-06-02 23:59:59", "ALANINE AMINOTRANSFERASE ABNORMAL",
  3, "Alanine aminotransferase abnormal", NA_character_, NA_integer_,
  "02", "2020-06-05 23:59:59", "BASEDOW'S DISEASE",
  5, "Basedow's disease", NA_character_, 1L,
  "03", "2020-06-07 23:59:59", "SOME TERM",
  2, "Some query", "Some term", NA_integer_,
  "05", "2020-06-09 23:59:59", "ALVEOLAR PROTEINOSIS",
  7, "Alveolar proteinosis", NA_character_, NA_integer_
)
derive_vars_query(adae, queries)
```

derive_vars_transposed *Derive Variables by Transposing and Merging a Second Dataset*

Description

Adds variables from a vertical dataset after transposing it into a wide one.

Usage

```
derive_vars_transposed(
  dataset,
  dataset_merge,
  by_vars,
  id_vars = NULL,
  key_var,
  value_var,
  filter = NULL,
  relationship = NULL
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. **Default value** none
dataset_merge	Dataset to transpose and merge The variables specified by the by_vars, id_vars, key_var and value_var arguments are expected. The variables by_vars, id_vars, key_var have to be a unique key. **Default value** none
by_vars	Grouping variables Keys used to merge dataset_merge with dataset. **Default value** none
id_vars	ID variables Variables (excluding by_vars and key_var) that uniquely identify each observation in dataset_merge. **Default value** NULL
key_var	The variable of dataset_merge containing the names of the transposed variables **Default value** none

value_var
: The variable of dataset_merge containing the values of the transposed variables

  Default value none

filter
: Expression used to restrict the records of dataset_merge prior to transposing

  Default value NULL

relationship
: Expected merge-relationship between the by_vars variable(s) in dataset and dataset_merge (after transposition)

  This argument is passed to the dplyr::left_join() function. See https://dplyr.tidyverse.org/reference/mutate-joins.html#arguments for more details.

  Permitted values "one-to-one", "one-to-many", "many-to-one", "many-to-many", NULL

  Default value NULL

Details

1. The records from the dataset to transpose and merge (dataset_merge) are restricted to those matching the filter condition, if provided.
2. The records from dataset_merge are checked to ensure they are uniquely identified using by_vars, id_vars and key_var.
3. dataset_merge is transposed (from "tall" to "wide"), with new variables added whose names come from key_var and values come from value_var.
4. The transposed dataset is merged with the input dataset using by_vars as keys. If a relationship has been provided, this merge must satisfy the relationship, otherwise an error is thrown.

Note that unlike other derive_vars_*() functions, the final step may cause new records to be added to the input dataset. The relationship argument can be specified to ensure this does not happen inadvertently.

Value

The input dataset with transposed variables from dataset_merge added

See Also

derive_vars_atc()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_summary()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
```

```
# Adding ATC classes to CM using FACM
cm <- tribble(
  ~USUBJID,       ~CMGRPID, ~CMREFID,  ~CMDECOD,
  "BP40257-1001", "14",     "1192056", "PARACETAMOL",
  "BP40257-1001", "18",     "2007001", "SOLUMEDROL",
  "BP40257-1002", "19",     "2791596", "SPIRONOLACTONE"
)
facm <- tribble(
  ~USUBJID,       ~FAGRPID, ~FAREFID,  ~FATESTCD,  ~FASTRESC,
  "BP40257-1001", "1",      "1192056", "CMATC1CD", "N",
  "BP40257-1001", "1",      "1192056", "CMATC2CD", "N02",
  "BP40257-1001", "1",      "1192056", "CMATC3CD", "N02B",
  "BP40257-1001", "1",      "1192056", "CMATC4CD", "N02BE",
  "BP40257-1001", "1",      "2007001", "CMATC1CD", "D",
  "BP40257-1001", "1",      "2007001", "CMATC2CD", "D10",
  "BP40257-1001", "1",      "2007001", "CMATC3CD", "D10A",
  "BP40257-1001", "1",      "2007001", "CMATC4CD", "D10AA",
  "BP40257-1001", "2",      "2007001", "CMATC1CD", "D",
  "BP40257-1001", "2",      "2007001", "CMATC2CD", "D07",
  "BP40257-1001", "2",      "2007001", "CMATC3CD", "D07A",
  "BP40257-1001", "2",      "2007001", "CMATC4CD", "D07AA",
  "BP40257-1001", "3",      "2007001", "CMATC1CD", "H",
  "BP40257-1001", "3",      "2007001", "CMATC2CD", "H02",
  "BP40257-1001", "3",      "2007001", "CMATC3CD", "H02A",
  "BP40257-1001", "3",      "2007001", "CMATC4CD", "H02AB",
  "BP40257-1002", "1",      "2791596", "CMATC1CD", "C",
  "BP40257-1002", "1",      "2791596", "CMATC2CD", "C03",
  "BP40257-1002", "1",      "2791596", "CMATC3CD", "C03D",
  "BP40257-1002", "1",      "2791596", "CMATC4CD", "C03DA"
)

cm %>%
  derive_vars_transposed(
    dataset_merge = facm,
    by_vars = exprs(USUBJID, CMREFID = FAREFID),
    id_vars = exprs(FAGRPID),
    key_var = FATESTCD,
    value_var = FASTRESC
  ) %>%
  select(USUBJID, CMDECOD, starts_with("CMATC"))

# Note: the `id_vars` argument here is needed to uniquely identify
# rows of dataset_merge and avoid duplicates-related errors.
# Compare the above call to when `id_vars = NULL`:

try(
  cm %>%
    derive_vars_transposed(
      dataset_merge = facm,
      by_vars = exprs(USUBJID, CMREFID = FAREFID),
      id_vars = NULL,
      key_var = FATESTCD,
      value_var = FASTRESC
```

```
    )
)
```

`derive_var_age_years` *Derive Age in Years*

Description

Converts the given age variable (`age_var`) to the unit 'years' from the current units given in the `age_var+U` variable or `age_unit` argument and stores in a new variable (`new_var`).

Usage

```
derive_var_age_years(dataset, age_var, age_unit = NULL, new_var)
```

Arguments

`dataset`	Input dataset The variables specified by the `age_var` argument are expected to be in the dataset. **Default value** none
`age_var`	Age variable. A numeric object is expected. **Default value** none
`age_unit`	Age unit. The `age_unit` argument is only expected when there is NOT a variable `age_var+U` in `dataset`. This gives the unit of the `age_var` variable and is used to convert AGE to 'years' so that grouping can occur. **Permitted values** 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds' **Default value** `NULL`
`new_var`	New age variable to be created in years. The returned values are doubles and NOT integers. ' **Default value** none

Details

This function is used to convert an age variable into the unit 'years' which can then be used to create age groups. The resulting column contains the equivalent years as a double. Note, underlying computations assume an equal number of days in each year (365.25).

Value

The input dataset (`dataset`) with `new_var` variable added in years.

See Also

derive_vars_duration()

ADSL Functions that returns variable appended to dataset: derive_vars_aage(), derive_vars_extreme_event(), derive_vars_period()

Examples

```
library(tibble)

# Derive age with age units specified
data <- tribble(
  ~AGE, ~AGEU,
  27, "days",
  24, "months",
  3, "years",
  4, "weeks",
  1, "years"
)

derive_var_age_years(data, AGE, new_var = AAGE)

# Derive age without age units variable specified
data <- tribble(
  ~AGE,
  12,
  24,
  36,
  48
)
derive_var_age_years(data, AGE, age_unit = "months", new_var = AAGE)
```

---

derive_var_analysis_ratio

Derive Ratio Variable

Description

Derives a ratio variable for a BDS dataset based on user specified variables.

Usage

```
derive_var_analysis_ratio(dataset, numer_var, denom_var, new_var = NULL)
```

Arguments

dataset	Input dataset The variables specified by the numer_var and denom_var arguments are expected to be in the dataset.

Default value none

`numer_var` Variable containing numeric values to be used in the numerator of the ratio calculation.

Default value none

`denom_var` Variable containing numeric values to be used in the denominator of the ratio calculation.

Default value none

`new_var` A user-defined variable that will be appended to the dataset. The default behavior will take the denominator variable and prefix it with `R2` and append to the dataset. Using this argument will override this default behavior.

Default is `NULL`.

Default value `NULL`

Details

A user wishing to calculate a Ratio to Baseline, `AVAL / BASE` will have returned a new variable `R2BASE` that will be appended to the input dataset. Ratio to Analysis Range Lower Limit `AVAL / ANRLO` will return a new variable `R2ANRLO`, and Ratio to Analysis Range Upper Limit `AVAL / ANRHI` will return a new variable `R2ANRLO`. Please note how the denominator variable has the prefix `R2----`. A user can override the default returned variables by using the `new_var` argument. Also, values of 0 in the denominator will return `NA` in the derivation.

Note that `R2AyHI` and `R2AyLO` can also be derived using this function.

Reference CDISC ADaM Implementation Guide Version 1.1 Section 3.3.4 Analysis Parameter Variables for BDS Datasets

Value

The input dataset with a ratio variable appended

See Also

BDS-Findings Functions that returns variable appended to dataset: `derive_var_anrind()`, `derive_var_atoxgr()`, `derive_var_atoxgr_dir()`, `derive_var_base()`, `derive_var_chg()`, `derive_var_nfrlt()`, `derive_var_ontrtfl()`, `derive_var_pchg()`, `derive_var_shift()`, `derive_vars_crit_flag()`

Examples

```
library(tibble)

data <- tribble(
  ~USUBJID, ~PARAMCD, ~SEQ, ~AVAL, ~BASE, ~ANRLO, ~ANRHI,
  "P01", "ALT", 1, 27, 27, 6, 34,
  "P01", "ALT", 2, 41, 27, 6, 34,
  "P01", "ALT", 3, 17, 27, 6, 34,
  "P02", "ALB", 1, 38, 38, 33, 49,
  "P02", "ALB", 2, 39, 38, 33, 49,
  "P02", "ALB", 3, 37, 38, 33, 49
```

```
)

# Returns "R2" prefixed variables
data %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = BASE) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRLO) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRHI)

# Returns user-defined variables
data %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = BASE, new_var = R01BASE) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRLO, new_var = R01ANRLO) %>%
  derive_var_analysis_ratio(numer_var = AVAL, denom_var = ANRHI, new_var = R01ANRHI)
```

`derive_var_anrind` *Derive Reference Range Indicator*

Description

Derive Reference Range Indicator

Usage

```
derive_var_anrind(
  dataset,
  signif_dig = get_admiral_option("signif_digits"),
  use_a1hia1lo = FALSE
)
```

Arguments

`dataset`	Input dataset `ANRLO`, `ANRHI`, and `AVAL` are expected and if `use_a1hia1lo` is set to `TRUE`, `A1LO` and `A1H1` are expected as well. **Default value** none
`signif_dig`	Number of significant digits to use when comparing values. Significant digits used to avoid floating point discrepancies when comparing numeric values. See blog: How admiral handles floating points **Default value** `get_admiral_option("signif_digits")`
`use_a1hia1lo`	A logical value indicating whether to use `A1H1` and `A1LO` in the derivation of `ANRIND`. **Default value** `FALSE`

Details

In the case that A1H1 and A1LO are to be used, ANRIND is set to:

- "NORMAL" if AVAL is greater or equal ANRLO and less than or equal ANRHI; or if AVAL is greater than or equal ANRLO and ANRHI is missing; or if AVAL is less than or equal ANRHI and ANRLO is missing
- "LOW" if AVAL is less than ANRLO and either A1LO is missing or AVAL is greater than or equal A1LO
- "HIGH" if AVAL is greater than ANRHI and either A1HI is missing or AVAL is less than or equal A1HI
- "LOW LOW" if AVAL is less than A1LO
- "HIGH HIGH" if AVAL is greater than A1HI

In the case that A1H1 and A1LO are not to be used, ANRIND is set to:

- "NORMAL" if AVAL is greater or equal ANRLO and less than or equal ANRHI; or if AVAL is greater than or equal ANRLO and ANRHI is missing; or if AVAL is less than or equal ANRHI and ANRLO is missing
- "LOW" if AVAL is less than ANRLO
- "HIGH" if AVAL is greater than ANRHI

Value

The input dataset with additional column ANRIND

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift(), derive_vars_crit_flag()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

vs <- tibble::tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ANRLO, ~ANRHI, ~A1LO, ~A1HI,
  "P01",       "PUL",    70,     60,    100,    40,   110,
  "P01",       "PUL",    57,     60,    100,    40,   110,
  "P01",       "PUL",    60,     60,    100,    40,   110,
  "P01",     "DIABP",   102,     60,     80,    40,    90,
  "P02",       "PUL",   109,     60,    100,    40,   110,
  "P02",       "PUL",   100,     60,    100,    40,   110,
  "P02",     "DIABP",    80,     60,     80,    40,    90,
  "P03",       "PUL",    39,     60,    100,    40,   110,
  "P03",       "PUL",    40,     60,    100,    40,   110
)
```

```
vs %>% derive_var_anrind(use_a1hia1lo = TRUE)
vs %>% derive_var_anrind(use_a1hia1lo = FALSE)
```

`derive_var_atoxgr` *Derive Lab High toxicity Grade 0 - 4 and Low Toxicity Grades 0 - (-4)*

Description

Derives character lab grade based on high and low severity/toxicity grade(s).

Usage

```
derive_var_atoxgr(
  dataset,
  lotox_description_var = ATOXDSCL,
  hitox_description_var = ATOXDSCH
)
```

Arguments

`dataset` Input dataset

The variables specified by the `lotox_description_var` and `hitox_description_var` arguments are expected to be in the dataset. `ATOXGRL`, and `ATOXGRH` are expected as well.

Default value none

`lotox_description_var`

Variable containing the toxicity grade description for low values, eg. "Anemia"

Default value `ATOXDSCL`

`hitox_description_var`

Variable containing the toxicity grade description for high values, eg. "Hemoglobin Increased".

Default value `ATOXDSCH`

Details

Created variable `ATOXGR` will contain values "-4", "-3", "-2", "-1" for low values and "1", "2", "3", "4" for high values, and will contain "0" if value is gradable and does not satisfy any of the criteria for high or low values. ATOXGR is set to missing if information not available to give a grade.

Function applies the following rules:

- High and low missing - overall missing
- Low grade not missing and > 0 - overall holds low grade
- High grade not missing and > 0 - overall holds high grade

- (Only high direction OR low direction is NORMAL) and high grade normal - overall NORMAL
- (Only low direction OR high direction is NORMAL) and low grade normal - overall NORMAL
- otherwise set to missing

Value

The input data set with the character variable added

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift(), derive_vars_crit_flag()

Examples

```
library(tibble)

adlb <- tribble(
  ~ATOXDSCL,          ~ATOXDSCH,        ~ATOXGRL,      ~ATOXGRH,
  "Hypoglycemia",     "Hyperglycemia",  NA_character_, "0",
  "Hypoglycemia",     "Hyperglycemia",  "0",           "1",
  "Hypoglycemia",     "Hyperglycemia",  "0",           "0",
  NA_character_,      "INR Increased",  NA_character_, "0",
  "Hypophosphatemia", NA_character_,    "1",           NA_character_
)

derive_var_atoxgr(adlb)
```

---

derive_var_atoxgr_dir *Derive Lab Toxicity Grade 0 - 4*

Description

Derives a character lab grade based on severity/toxicity criteria.

Usage

```
derive_var_atoxgr_dir(
  dataset,
  new_var,
  tox_description_var,
  meta_criteria,
  criteria_direction,
  abnormal_indicator = NULL,
  high_indicator = NULL,
```

```
  low_indicator = NULL,
  get_unit_expr,
  signif_dig = get_admiral_option("signif_digits")
)
```

Arguments

`dataset` Input dataset

The variables specified by the `tox_description_var` argument are expected to be in the dataset.

Default value none

`new_var` Name of the character grade variable to create, for example, `ATOXGRH` or `ATOXGRL`.

Default value none

`tox_description_var`

Variable containing the description of the grading criteria. For example: "Anemia" or "INR Increased".

Default value none

`meta_criteria` Metadata data set holding the criteria (normally a case statement)

Permitted values `atoxgr_criteria_ctcv4`, `atoxgr_criteria_ctcv5`, `atoxgr_criteria_ctcv6`, `atoxgr_criteria_daids`

- `atoxgr_criteria_ctcv4` implements Common Terminology Criteria for Adverse Events (CTCAE) v4.0
- `atoxgr_criteria_ctcv5` implements Common Terminology Criteria for Adverse Events (CTCAE) v5.0
- `atoxgr_criteria_ctcv6` implements Common Terminology Criteria for Adverse Events (CTCAE) v6.0
- `atoxgr_criteria_daids` implements Division of AIDS (DAIDS) Table for Grading the Severity of Adult and Pediatric Adverse Events

The metadata should have the following variables:

- `TERM`: variable to hold the term describing the criteria applied to a particular lab test, eg. "Anemia" or "INR Increased". Note: the variable is case insensitive.
- `DIRECTION`: variable to hold the direction of the abnormality of a particular lab test value. "L" is for LOW values, "H" is for HIGH values. Note: the variable is case insensitive.
- `UNIT_CHECK`: variable to hold unit of particular lab test. Used to check against input data if criteria is based on absolute values.
- `VAR_CHECK`: variable to hold comma separated list of variables used in criteria. Used to check against input data that variables exist.
- `GRADE_CRITERIA_CODE`: variable to hold code that creates grade based on defined criteria.
- `FILTER`: Required only for DAIDS grading, specifies `admiral` code to filter the lab data based on a subset of subjects (e.g. AGE > 18 YEARS)

Default value none

criteria_direction
: Direction (L= Low, H = High) of toxicity grade.

  Permitted values "L", "H"

  Default value none

abnormal_indicator
: **[Deprecated]** Please use low_indicator and high_indicator instead.

  Default value NULL

high_indicator
: Value in BNRIND derivation to indicate an abnormal high value. Usually "HIGH" for criteria_direction = "H".

  This is only required when meta_criteria = atoxgr_criteria_ctcv5 or meta_criteria = atoxgr_criteria_ctcv6 and BNRIND is a required variable. Currently, for terms "Alanine aminotransferase increased", "Aspartate aminotransferase increased", "Blood bilirubin increased" and "GGT increased" for both sets of criteria. Also, term "Alkaline phosphatase increased" for meta_criteria = atoxgr_criteria_ctcv5.

  Default value NULL

low_indicator
: Value in BNRIND derivation to indicate an abnormal low value. Usually "LOW" for criteria_direction = "L".

  This is only required when meta_criteria = atoxgr_criteria_ctcv6 and BNRIND is a required variable. Currently, only for term "Creatinine increased".

  Default value NULL

get_unit_expr
: An expression providing the unit of the parameter

  The result is used to check the units of the input parameters. Compared with UNIT_CHECK in metadata (see meta_criteria parameter).

  Permitted values A variable containing unit from the input dataset, or a function call, for example, get_unit_expr = extract_unit(PARAM).

  Default value none

signif_dig
: Number of significant digits to use when comparing a lab value against another value.

  Significant digits used to avoid floating point discrepancies when comparing numeric values. See blog: How admiral handles floating points

  Default value get_admiral_option("signif_digits")

Details

new_var is derived with values NA, "0", "1", "2", "3", "4", where "4" is the most severe grade

- "4" is where the lab value satisfies the criteria for grade 4.
- "3" is where the lab value satisfies the criteria for grade 3.
- "2" is where the lab value satisfies the criteria for grade 2.
- "1" is where the lab value satisfies the criteria for grade 1.
- "0" is where a grade can be derived and is not grade "1", "2", "3" or "4".
- NA is where a grade cannot be derived.

Value

The input dataset with the character variable added

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_base(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift(), derive_vars_crit_flag()

Examples

```
library(tibble)

data <- tribble(
  ~ATOXDSCL,                    ~AVAL, ~ANRLO, ~ANRHI, ~PARAM,
  "Hypoglycemia",               119,   4,      7,      "Glucose (mmol/L)",
  "Lymphocyte count decreased", 0.7,   1,      4,      "Lymphocytes Abs (10^9/L)",
  "Anemia",                     129,   120,    180,    "Hemoglobin (g/L)",
  "White blood cell decreased", 10,    5,      20,     "White blood cell (10^9/L)",
  "White blood cell decreased", 15,    5,      20,     "White blood cell (10^9/L)",
  "Anemia",                     140,   120,    180,    "Hemoglobin (g/L)"
)

derive_var_atoxgr_dir(data,
  new_var = ATOXGRL,
  tox_description_var = ATOXDSCL,
  meta_criteria = atoxgr_criteria_ctcv5,
  criteria_direction = "L",
  get_unit_expr = extract_unit(PARAM)
)

data <- tribble(
  ~ATOXDSCH,                    ~AVAL,  ~ANRLO,   ~ANRHI, ~PARAM,
  "CPK increased",              129,    0,        30,     "Creatine Kinase (U/L)",
  "Lymphocyte count increased", 4,      1,        4,      "Lymphocytes Abs (10^9/L)",
  "Lymphocyte count increased", 2,      1,        4,      "Lymphocytes Abs (10^9/L)",
  "CPK increased",              140,    120,      180,    "Creatine Kinase (U/L)"
)

derive_var_atoxgr_dir(data,
  new_var = ATOXGRH,
  tox_description_var = ATOXDSCH,
  meta_criteria = atoxgr_criteria_ctcv5,
  criteria_direction = "H",
  get_unit_expr = extract_unit(PARAM)
)
```

derive_var_base *Derive Baseline Variables*

Description

Derive baseline variables, e.g. BASE or BNRIND, in a BDS dataset.

Note: This is a wrapper function for the more generic derive_vars_merged().

Usage

```
derive_var_base(
  dataset,
  by_vars,
  source_var = AVAL,
  new_var = BASE,
  filter = ABLFL == "Y"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and source_var arguments are expected to be in the dataset. **Default value** none
by_vars	Grouping variables Grouping variables uniquely identifying a set of records for which to calculate new_var. **Default value** none
source_var	The column from which to extract the baseline value, e.g. AVAL **Default value** AVAL
new_var	The name of the newly created baseline column, e.g. BASE **Default value** BASE
filter	The condition used to filter dataset for baseline records. By default ABLFL == "Y" **Default value** ABLFL == "Y"

Details

For each by_vars group, the baseline record is identified by the condition specified in filter which defaults to ABLFL == "Y". Subsequently, every value of the new_var variable for the by_vars group is set to the value of the source_var variable of the baseline record. In case there are multiple baseline records within by_vars an error is issued.

Value

A new data.frame containing all records and variables of the input dataset plus the new_var variable

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift(), derive_vars_crit_flag()

Examples

```
library(tibble)

dataset <- tribble(
  ~STUDYID, ~USUBJID, ~PARAMCD, ~AVAL, ~AVALC,   ~AVISIT,    ~ABLFL, ~ANRIND,
  "TEST01", "PAT01", "PARAM01", 10.12, NA,       "Baseline", "Y",    "NORMAL",
  "TEST01", "PAT01", "PARAM01", 9.700, NA,       "Day 7",    NA,     "LOW",
  "TEST01", "PAT01", "PARAM01", 15.01, NA,       "Day 14",   NA,     "HIGH",
  "TEST01", "PAT01", "PARAM02", 8.350, NA,       "Baseline", "Y",    "LOW",
  "TEST01", "PAT01", "PARAM02",    NA, NA,       "Day 7",    NA,     NA,
  "TEST01", "PAT01", "PARAM02", 8.350, NA,       "Day 14",   NA,     "LOW",
  "TEST01", "PAT01", "PARAM03",    NA, "LOW",    "Baseline", "Y",    NA,
  "TEST01", "PAT01", "PARAM03",    NA, "LOW",    "Day 7",    NA,     NA,
  "TEST01", "PAT01", "PARAM03",    NA, "MEDIUM", "Day 14",   NA,     NA,
  "TEST01", "PAT01", "PARAM04",    NA, "HIGH",   "Baseline", "Y",    NA,
  "TEST01", "PAT01", "PARAM04",    NA, "HIGH",   "Day 7",    NA,     NA,
  "TEST01", "PAT01", "PARAM04",    NA, "MEDIUM", "Day 14",   NA,     NA
)

## Derive `BASE` variable from `AVAL`
derive_var_base(
  dataset,
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = AVAL,
  new_var = BASE
)

## Derive `BASEC` variable from `AVALC`
derive_var_base(
  dataset,
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = AVALC,
  new_var = BASEC
)

## Derive `BNRIND` variable from `ANRIND`
derive_var_base(
  dataset,
  by_vars = exprs(USUBJID, PARAMCD),
  source_var = ANRIND,
  new_var = BNRIND
```

```
)
```

`derive_var_chg` *Derive Change from Baseline*

Description

Derive change from baseline (`CHG`) in a BDS dataset

Usage

```
derive_var_chg(dataset)
```

Arguments

`dataset`	Input dataset `AVAL` and `BASE` are expected. **Default value** none

Details

Change from baseline is calculated by subtracting the baseline value from the analysis value.

Value

The input dataset with an additional column named `CHG`

See Also

BDS-Findings Functions that returns variable appended to dataset: `derive_var_analysis_ratio()`, `derive_var_anrind()`, `derive_var_atoxgr()`, `derive_var_atoxgr_dir()`, `derive_var_base()`, `derive_var_nfrlt()`, `derive_var_ontrtfl()`, `derive_var_pchg()`, `derive_var_shift()`, `derive_vars_crit_flag()`

Examples

```
library(tibble)

advs <- tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ABLFL, ~BASE,
  "P01",    "WEIGHT", 80,    "Y",    80,
  "P01",    "WEIGHT", 80.8,  NA,     80,
  "P01",    "WEIGHT", 81.4,  NA,     80,
  "P02",    "WEIGHT", 75.3,  "Y",    75.3,
  "P02",    "WEIGHT", 76,    NA,     75.3
)
derive_var_chg(advs)
```

`derive_var_dthcaus` *Derive Death Cause*

Description

[Deprecated] The `derive_var_dthcaus()` function has been deprecated in favor of `derive_vars_extreme_event()`.

Derive death cause (`DTHCAUS`) and add traceability variables if required.

Usage

```
derive_var_dthcaus(
  dataset,
  ...,
  source_datasets,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

`dataset`	Input dataset The variables specified by the `subject_keys` argument are expected to be in the dataset. **Default value** none
`...`	Objects of class "dthcaus_source" created by `dthcaus_source()`. **Default value** none
`source_datasets`	A named `list` containing datasets in which to search for the death cause **Default value** none
`subject_keys`	Variables to uniquely identify a subject A list of expressions where the expressions are symbols as returned by `exprs()` is expected. **Default value** `get_admiral_option("subject_keys")`

Details

This function derives `DTHCAUS` along with the user-defined traceability variables, if required. If a subject has death info from multiple sources, the one from the source with the earliest death date will be used. If dates are equivalent, the first source will be kept, so the user should provide the inputs in the preferred order.

Value

The input dataset with `DTHCAUS` variable added.

See Also

dthcaus_source()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

adsl <- tribble(
  ~STUDYID,  ~USUBJID,
  "STUDY01", "PAT01",
  "STUDY01", "PAT02",
  "STUDY01", "PAT03"
)
ae <- tribble(
  ~STUDYID,  ~USUBJID, ~AESEQ, ~AEDECOD,        ~AEOUT,  ~AEDTHDTC,
  "STUDY01", "PAT01",  12,     "SUDDEN DEATH", "FATAL", "2021-04-04"
)

ds <- tribble(
  ~STUDYID, ~USUBJID, ~DSSEQ, ~DSDECOD, ~DSTERM, ~DSSTDTC,
 "STUDY01", "PAT02", 1, "INFORMED CONSENT OBTAINED", "INFORMED CONSENT OBTAINED", "2021-04-03",
  "STUDY01", "PAT02", 2, "RANDOMIZATION", "RANDOMIZATION", "2021-04-11",
  "STUDY01", "PAT02", 3, "DEATH", "DEATH DUE TO PROGRESSION OF DISEASE", "2022-02-01",
  "STUDY01", "PAT03", 1, "DEATH", "POST STUDY REPORTING OF DEATH", "2022-03-03"
)

# Derive `DTHCAUS` only - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = convert_dtc_to_dt(AEDTHDTC),
  mode = "first",
  dthcaus = AEDECOD
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM
)

derive_var_dthcaus(adsl, src_ae, src_ds, source_datasets = list(ae = ae, ds = ds))

# Derive `DTHCAUS` and add traceability variables - for on-study deaths only
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
```

```
  date = convert_dtc_to_dt(AEDTHDTC),
  mode = "first",
  dthcaus = AEDECOD,
  set_values_to = exprs(DTHDOM = "AE", DTHSEQ = AESEQ)
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM,
  set_values_to = exprs(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

derive_var_dthcaus(adsl, src_ae, src_ds, source_datasets = list(ae = ae, ds = ds))

# Derive `DTHCAUS` as above - now including post-study deaths with different `DTHCAUS` value
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = convert_dtc_to_dt(AEDTHDTC),
  mode = "first",
  dthcaus = AEDECOD,
  set_values_to = exprs(DTHDOM = "AE", DTHSEQ = AESEQ)
)

ds <- mutate(
  ds,
  DSSTDT = convert_dtc_to_dt(DSSTDTC)
)

src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & grepl("DEATH DUE TO", DSTERM),
  date = DSSTDT,
  mode = "first",
  dthcaus = DSTERM,
  set_values_to = exprs(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

src_ds_post <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH" & DSTERM == "POST STUDY REPORTING OF DEATH",
  date = DSSTDT,
  mode = "first",
  dthcaus = "POST STUDY: UNKNOWN CAUSE",
  set_values_to = exprs(DTHDOM = "DS", DTHSEQ = DSSEQ)
)

derive_var_dthcaus(
  adsl,
  src_ae, src_ds, src_ds_post,
```

```
  source_datasets = list(ae = ae, ds = ds)
)
```

`derive_var_extreme_dt` *Derive First or Last Date from Multiple Sources*

Description

[Deprecated] The `derive_var_extreme_dt()` function has been deprecated in favor of `derive_vars_extreme_event()`.

Add the first or last date from multiple sources to the dataset, e.g., the last known alive date (LSTALVDT).

Note: This is a wrapper function for the function `derive_var_extreme_dtm()`.

Usage

```
derive_var_extreme_dt(
  dataset,
  new_var,
  ...,
  source_datasets,
  mode,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

`dataset`	Input dataset The variables specified by the `subject_keys` argument are expected to be in the dataset. **Default value** none
`new_var`	Name of variable to create **Default value** none
`...`	Source(s) of dates. One or more `date_source()` objects are expected. **Default value** none
`source_datasets`	A named `list` containing datasets in which to search for the first or last date **Default value** none
`mode`	Selection mode (first or last) If `"first"` is specified, the first date for each subject is selected. If `"last"` is specified, the last date for each subject is selected. **Permitted values** `"first"`, `"last"` **Default value** none
`subject_keys`	Variables to uniquely identify a subject A list of expressions where the expressions are symbols as returned by `exprs()` is expected. **Default value** `get_admiral_option("subject_keys")`

Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the `filter` element are selected and observations where `date` is `NA` are removed. Then for each patient the first or last observation (with respect to `date` and `mode`) is selected.
2. The new variable is set to the variable or expression specified by the `date` element.
3. The variables specified by the `set_values_to` element are added.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the new variable and `mode`) from the single dataset is selected and the new variable is merged to the input dataset.
6. The time part is removed from the new variable.

Value

The input dataset with the new variable added.

See Also

`date_source()`, `derive_var_extreme_dtm()`, `derive_vars_merged()`

Other deprecated: `call_user_fun()`, `date_source()`, `derive_param_extreme_record()`, `derive_var_dthcaus()`, `derive_var_extreme_dtm()`, `derive_var_merged_summary()`, `dthcaus_source()`, `get_summary_records()`

Examples

```
library(dplyr, warn.conflicts = FALSE)
ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AESEQ,     ~AESTDTC,     ~AEENDTC,
  "PILOT01",    "AE", "01-1130",      5, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      6, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      4, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      8, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      7, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      2, "2014-03-09", "2014-03-09",
  "PILOT01",    "AE", "01-1130",      1, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1130",      3, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1133",      1, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      3, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      2, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      4, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1211",      5, "2012-11-29",           NA,
  "PILOT01",    "AE", "01-1211",      1, "2012-11-16",           NA,
  "PILOT01",    "AE", "01-1211",      7, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      8, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      4, "2012-11-22",           NA,
  "PILOT01",    "AE", "01-1211",      2, "2012-11-21", "2012-11-21",
  "PILOT01",    "AE", "01-1211",      3, "2012-11-21",           NA,
  "PILOT01",    "AE", "01-1211",      6, "2012-12-09",           NA,
  "PILOT01",    "AE", "01-1211",      9, "2013-01-14", "2013-01-14",
```

```
  "PILOT01",    "AE", "09-1081",      2, "2014-05-01",          NA,
  "PILOT01",    "AE", "09-1081",      1, "2014-04-07",          NA,
  "PILOT01",    "AE", "09-1088",      1, "2014-05-08",          NA,
  "PILOT01",    "AE", "09-1088",      2, "2014-08-02",          NA
)

adsl <- tribble(
  ~STUDYID,   ~USUBJID,              ~TRTEDTM,      ~TRTEDT,
  "PILOT01", "01-1130", "2014-08-16 23:59:59", "2014-08-16",
  "PILOT01", "01-1133", "2013-04-28 23:59:59", "2013-04-28",
  "PILOT01", "01-1211", "2013-01-12 23:59:59", "2013-01-12",
  "PILOT01", "09-1081", "2014-04-27 23:59:59", "2014-04-27",
  "PILOT01", "09-1088", "2014-10-09 23:59:59", "2014-10-09"
) %>%
  mutate(
    across(TRTEDTM:TRTEDT, as.Date)
  )


lb <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~LBSEQ,             ~LBDTC,
  "PILOT01",    "LB", "01-1130",    219, "2014-06-07T13:20",
  "PILOT01",    "LB", "01-1130",    322, "2014-08-16T13:10",
  "PILOT01",    "LB", "01-1133",    268, "2013-04-18T15:30",
  "PILOT01",    "LB", "01-1133",    304, "2013-04-29T10:13",
  "PILOT01",    "LB", "01-1211",      8, "2012-10-30T14:26",
  "PILOT01",    "LB", "01-1211",    162, "2013-01-08T12:13",
  "PILOT01",    "LB", "09-1081",     47, "2014-02-01T10:55",
  "PILOT01",    "LB", "09-1081",    219, "2014-05-10T11:15",
  "PILOT01",    "LB", "09-1088",    283, "2014-09-27T12:13",
  "PILOT01",    "LB", "09-1088",    322, "2014-10-09T13:25"
)

dm <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",    "DM", "01-1130",   84, "YEARS",
  "PILOT01",    "DM", "01-1133",   81, "YEARS",
  "PILOT01",    "DM", "01-1211",   76, "YEARS",
  "PILOT01",    "DM", "09-1081",   86, "YEARS",
  "PILOT01",    "DM", "09-1088",   69, "YEARS"
)

ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AESTDTC, highest_imputation = "M")
)
ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AEENDTC, highest_imputation = "M")
)

ae_ext <- ae %>%
  derive_vars_dt(
```

```
    dtc = AESTDTC,
    new_vars_prefix = "AEST",
    highest_imputation = "M"
  ) %>%
  derive_vars_dt(
    dtc = AEENDTC,
    new_vars_prefix = "AEEN",
    highest_imputation = "M"
  )

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dt(LBDTC)
)

lb_ext <- derive_vars_dt(
  lb,
  dtc = LBDTC,
  new_vars_prefix = "LB"
)

adsl_date <- date_source(dataset_name = "adsl", date = TRTEDT)

dm %>%
  derive_var_extreme_dt(
    new_var = LSTALVDT,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDT)

# derive last alive date and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AESTDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dt(AEENDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
```

```
    LALVVAR = "AEENDTC"
  )
)

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dt(LBDTC),
  set_values_to = exprs(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

adsl_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDT,
  set_values_to = exprs(
    LALVDOM = "ADSL",
    LALVSEQ = NA_integer_,
    LALVVAR = "TRTEDT"
  )
)

dm %>%
  derive_var_extreme_dt(
    new_var = LSTALVDT,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDT, LALVDOM, LALVSEQ, LALVVAR)
```

`derive_var_extreme_dtm` *Derive First or Last Datetime from Multiple Sources*

Description

[Deprecated]

The `derive_var_extreme_dtm()` function has been deprecated in favor of `derive_vars_extreme_event()`.

Add the first or last datetime from multiple sources to the dataset, e.g., the last known alive datetime (`LSTALVDTM`).

Usage

```
derive_var_extreme_dtm(
  dataset,
  new_var,
  ...,
  source_datasets,
  mode,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

dataset	Input dataset The variables specified by the subject_keys argument are expected to be in the dataset. **Default value** none
new_var	Name of variable to create **Default value** none
...	Source(s) of dates. One or more date_source() objects are expected. **Default value** none
source_datasets	A named list containing datasets in which to search for the first or last date **Default value** none
mode	Selection mode (first or last) If "first" is specified, the first date for each subject is selected. If "last" is specified, the last date for each subject is selected. **Permitted values** "first", "last" **Default value** none
subject_keys	Variables to uniquely identify a subject A list of expressions where the expressions are symbols as returned by exprs() is expected. **Default value** get_admiral_option("subject_keys")

Details

The following steps are performed to create the output dataset:

1. For each source dataset the observations as specified by the filter element are selected and observations where date is NA are removed. Then for each patient the first or last observation (with respect to date and mode) is selected.
2. The new variable is set to the variable or expression specified by the date element. If this is a date variable (rather than datetime), then the time is imputed as "00:00:00".
3. The variables specified by the set_values_to element are added.
4. The selected observations of all source datasets are combined into a single dataset.
5. For each patient the first or last observation (with respect to the new variable and mode) from the single dataset is selected and the new variable is merged to the input dataset.

Value

The input dataset with the new variable added.

See Also

date_source(), derive_var_extreme_dt(), derive_vars_merged()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_merged_summary(), dthcaus_source(), get_summary_records()

Examples

```
library(dplyr, warn.conflicts = FALSE)
library(lubridate)
dm <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",    "DM", "01-1130",   84, "YEARS",
  "PILOT01",    "DM", "01-1133",   81, "YEARS",
  "PILOT01",    "DM", "01-1211",   76, "YEARS",
  "PILOT01",    "DM", "09-1081",   86, "YEARS",
  "PILOT01",    "DM", "09-1088",   69, "YEARS"
)
ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AESEQ,     ~AESTDTC,     ~AEENDTC,
  "PILOT01",    "AE", "01-1130",      5, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      6, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      4, "2014-05-09", "2014-05-09",
  "PILOT01",    "AE", "01-1130",      8, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      7, "2014-05-22",           NA,
  "PILOT01",    "AE", "01-1130",      2, "2014-03-09", "2014-03-09",
  "PILOT01",    "AE", "01-1130",      1, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1130",      3, "2014-03-09", "2014-03-16",
  "PILOT01",    "AE", "01-1133",      1, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      3, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      2, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1133",      4, "2012-12-27",           NA,
  "PILOT01",    "AE", "01-1211",      5, "2012-11-29",           NA,
  "PILOT01",    "AE", "01-1211",      1, "2012-11-16",           NA,
  "PILOT01",    "AE", "01-1211",      7, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      8, "2013-01-11",           NA,
  "PILOT01",    "AE", "01-1211",      4, "2012-11-22",           NA,
  "PILOT01",    "AE", "01-1211",      2, "2012-11-21", "2012-11-21",
  "PILOT01",    "AE", "01-1211",      3, "2012-11-21",           NA,
  "PILOT01",    "AE", "01-1211",      6, "2012-12-09",           NA,
  "PILOT01",    "AE", "01-1211",      9, "2013-01-14", "2013-01-14",
  "PILOT01",    "AE", "09-1081",      2, "2014-05-01",           NA,
  "PILOT01",    "AE", "09-1081",      1, "2014-04-07",           NA,
  "PILOT01",    "AE", "09-1088",      1, "2014-05-08",           NA,
  "PILOT01",    "AE", "09-1088",      2, "2014-08-02",           NA
)
lb <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~LBSEQ,             ~LBDTC,
```

```
  "PILOT01",    "LB", "01-1130",    219, "2014-06-07T13:20",
  "PILOT01",    "LB", "01-1130",    322, "2014-08-16T13:10",
  "PILOT01",    "LB", "01-1133",    268, "2013-04-18T15:30",
  "PILOT01",    "LB", "01-1133",    304, "2013-04-29T10:13",
  "PILOT01",    "LB", "01-1211",      8, "2012-10-30T14:26",
  "PILOT01",    "LB", "01-1211",    162, "2013-01-08T12:13",
  "PILOT01",    "LB", "09-1081",     47, "2014-02-01T10:55",
  "PILOT01",    "LB", "09-1081",    219, "2014-05-10T11:15",
  "PILOT01",    "LB", "09-1088",    283, "2014-09-27T12:13",
  "PILOT01",    "LB", "09-1088",    322, "2014-10-09T13:25"
)
adsl <- tribble(
  ~STUDYID,   ~USUBJID,              ~TRTEDTM,
  "PILOT01", "01-1130", "2014-08-16 23:59:59",
  "PILOT01", "01-1133", "2013-04-28 23:59:59",
  "PILOT01", "01-1211", "2013-01-12 23:59:59",
  "PILOT01", "09-1081", "2014-04-27 23:59:59",
  "PILOT01", "09-1088", "2014-10-09 23:59:59"
) %>%
  mutate(
    TRTEDTM = as_datetime(TRTEDTM)
  )

# derive last known alive datetime (LSTALVDTM)
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AESTDTC, highest_imputation = "M"),
)
ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AEENDTC, highest_imputation = "M"),
)

ae_ext <- ae %>%
  derive_vars_dtm(
    dtc = AESTDTC,
    new_vars_prefix = "AEST",
    highest_imputation = "M"
  ) %>%
  derive_vars_dtm(
    dtc = AEENDTC,
    new_vars_prefix = "AEEN",
    highest_imputation = "M"
  )

lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dtm(LBDTC),
)

lb_ext <- derive_vars_dtm(
  lb,
  dtc = LBDTC,
```

```
  new_vars_prefix = "LB"
)

adsl_date <- date_source(
  dataset_name = "adsl",
  date = TRTEDTM
)

dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM)

# derive last alive datetime and traceability variables
ae_start <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AESTDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AESTDTC"
  )
)

ae_end <- date_source(
  dataset_name = "ae",
  date = convert_dtc_to_dtm(AEENDTC, highest_imputation = "M"),
  set_values_to = exprs(
    LALVDOM = "AE",
    LALVSEQ = AESEQ,
    LALVVAR = "AEENDTC"
  )
)
lb_date <- date_source(
  dataset_name = "lb",
  date = convert_dtc_to_dtm(LBDTC),
  set_values_to = exprs(
    LALVDOM = "LB",
    LALVSEQ = LBSEQ,
    LALVVAR = "LBDTC"
  )
)

adsl_date <- date_source(
  dataset_name = "adsl",
```

```
  date = TRTEDTM,
  set_values_to = exprs(
    LALVDOM = "ADSL",
    LALVSEQ = NA_integer_,
    LALVVAR = "TRTEDTM"
  )
)

dm %>%
  derive_var_extreme_dtm(
    new_var = LSTALVDTM,
    ae_start, ae_end, lb_date, adsl_date,
    source_datasets = list(
      adsl = adsl,
      ae = ae_ext,
      lb = lb_ext
    ),
    mode = "last"
  ) %>%
  select(USUBJID, LSTALVDTM, LALVDOM, LALVSEQ, LALVVAR)
```

`derive_var_extreme_flag` *Add a Variable Flagging the First or Last Observation Within Each By Group*

Description

Add a variable flagging the first or last observation within each by group

Usage

```
derive_var_extreme_flag(
  dataset,
  by_vars,
  order,
  new_var,
  mode,
  true_value = "Y",
  false_value = NA_character_,
  flag_all = FALSE,
  check_type = "warning"
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble
Default value none

`by_vars` Grouping variables

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`
Default value none

`order` Sort order

The first or last observation is determined with respect to the specified order.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`
Default value none

`new_var` Variable to add

The specified variable is added to the output dataset. It is set to the value set in `true_value` for the first or last observation (depending on the mode) of each by group.

Permitted values an unquoted symbol, e.g., `AVAL`
Default value none

`mode` Flag mode

Determines of the first or last observation is flagged.

Permitted values `"first"`, `"last"`
Default value none

`true_value` True value

The value for the specified variable `new_var`, applicable to the first or last observation (depending on the mode) of each by group.

Permitted values a character scalar, i.e., a character vector of length one
Default value `"Y"`

`false_value` False value

The value for the specified variable `new_var`, NOT applicable to the first or last observation (depending on the mode) of each by group.

Permitted values a character scalar, i.e., a character vector of length one
Default value `NA_character_`

`flag_all` Flag setting

A logical value where if set to `TRUE`, all records are flagged and no error or warning is issued if the first or last record is not unique.

Permitted values `TRUE`, `FALSE`
Default value `FALSE`

`check_type` Check uniqueness?

If `"warning"` or `"error"` is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values `"none"`, `"message"`, `"warning"`, `"error"`
Default value `"warning"`

Details

For each group (with respect to the variables specified for the by_vars parameter), new_var is set to "Y" for the first or last observation (with respect to the order specified for the order parameter and the flag mode specified for the mode parameter). In the case where the user wants to flag multiple records of a grouping, for example records that all happen on the same visit and time, the argument flag_all can be set to TRUE. Otherwise, new_var is set to NA. Thus, the direction of "worst" is considered fixed for all parameters in the dataset depending on the order and the mode, i.e. for every parameter the first or last record will be flagged across the whole dataset.

Value

The input dataset with the new flag variable added

Examples

Data setup:

The following examples use the ADVS and ADAE datasets below as a basis.

```
library(tibble, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)
library(dplyr, warn.conflicts = FALSE)

advs <- tribble(
  ~USUBJID, ~PARAMCD,    ~AVISIT,          ~ADT, ~AVAL,
  "1015",   "TEMP",   "BASELINE",  "2021-04-27",  38.0,
  "1015",   "TEMP",   "BASELINE",  "2021-04-25",  39.0,
  "1015",   "TEMP",   "WEEK 2",    "2021-05-10",  37.5,
  "1015",   "WEIGHT", "SCREENING", "2021-04-19",  81.2,
  "1015",   "WEIGHT", "BASELINE",  "2021-04-25",  82.7,
  "1015",   "WEIGHT", "BASELINE",  "2021-04-27",  84.0,
  "1015",   "WEIGHT", "WEEK 2",    "2021-05-09",  82.5,
  "1023",   "TEMP",   "SCREENING", "2021-04-27",  38.0,
  "1023",   "TEMP",   "BASELINE",  "2021-04-28",  37.5,
  "1023",   "TEMP",   "BASELINE",  "2021-04-29",  37.5,
  "1023",   "TEMP",   "WEEK 1",    "2021-05-03",  37.0,
  "1023",   "WEIGHT", "SCREENING", "2021-04-27",  69.6,
  "1023",   "WEIGHT", "BASELINE",  "2021-04-29",  67.2,
  "1023",   "WEIGHT", "WEEK 1",    "2021-05-02",  65.9
) %>%
mutate(
  STUDYID = "AB123",
  ADT = ymd(ADT)
)

adae <- tribble(
  ~USUBJID,         ~AEBODSYS,    ~AEDECOD,   ~AESEV, ~AESTDY, ~AESEQ,
  "1015", "GENERAL DISORDERS",  "ERYTHEMA",   "MILD",       2,      1,
  "1015", "GENERAL DISORDERS",  "PRURITUS",   "MILD",       2,      2,
  "1015",      "GI DISORDERS", "DIARRHOEA",   "MILD",       8,      3,
```

```
  "1023", "CARDIAC DISORDERS",  "AV BLOCK",   "MILD",      22,      4,
  "1023",    "SKIN DISORDERS",  "ERYTHEMA",   "MILD",       3,      1,
  "1023",    "SKIN DISORDERS",  "ERYTHEMA", "SEVERE",       5,      2,
  "1023",    "SKIN DISORDERS",  "ERYTHEMA",   "MILD",       8,      3
) %>%
mutate(STUDYID = "AB123")
```

Flagging the first/last observation within a by group (`order`, `mode`):
A new variable is added for each subject to flag the last observation within a by group. Within each by group (specified by `by_vars`), the `order = exprs(ADT)` argument specifies we wish to sort the records by analysis date and then select the last one (`mode = "last"`). The name of the new variable is passed through the `new_var = LASTFL` call.

```
advs %>%
  derive_var_extreme_flag(
    by_vars = exprs(STUDYID, USUBJID, PARAMCD),
    order = exprs(ADT),
    new_var = LASTFL,
    mode = "last",
  ) %>%
  arrange(STUDYID, USUBJID, PARAMCD, ADT) %>%
  select(STUDYID, everything())
#> # A tibble: 14 × 7
#>    STUDYID USUBJID PARAMCD AVISIT    ADT         AVAL LASTFL
#>    <chr>   <chr>   <chr>   <chr>     <date>     <dbl> <chr>
#>  1 AB123   1015    TEMP    BASELINE  2021-04-25  39   <NA>
#>  2 AB123   1015    TEMP    BASELINE  2021-04-27  38   <NA>
#>  3 AB123   1015    TEMP    WEEK 2    2021-05-10  37.5 Y
#>  4 AB123   1015    WEIGHT  SCREENING 2021-04-19  81.2 <NA>
#>  5 AB123   1015    WEIGHT  BASELINE  2021-04-25  82.7 <NA>
#>  6 AB123   1015    WEIGHT  BASELINE  2021-04-27  84   <NA>
#>  7 AB123   1015    WEIGHT  WEEK 2    2021-05-09  82.5 Y
#>  8 AB123   1023    TEMP    SCREENING 2021-04-27  38   <NA>
#>  9 AB123   1023    TEMP    BASELINE  2021-04-28  37.5 <NA>
#> 10 AB123   1023    TEMP    BASELINE  2021-04-29  37.5 <NA>
#> 11 AB123   1023    TEMP    WEEK 1    2021-05-03  37   Y
#> 12 AB123   1023    WEIGHT  SCREENING 2021-04-27  69.6 <NA>
#> 13 AB123   1023    WEIGHT  BASELINE  2021-04-29  67.2 <NA>
#> 14 AB123   1023    WEIGHT  WEEK 1    2021-05-02  65.9 Y
```

Note here that a similar `FIRSTFL` variable could instead be derived simply by switching to `mode = "first"`. Alternatively, we could make use of `desc()` within the sorting specified by `order`:

```
advs %>%
  derive_var_extreme_flag(
    by_vars = exprs(STUDYID, USUBJID, PARAMCD),
    order = exprs(desc(ADT)),
    new_var = FIRSTFL,
    mode = "last",
  ) %>%
```

```
  arrange(STUDYID, USUBJID, PARAMCD, ADT) %>%
  select(STUDYID, everything())
#> # A tibble: 14 × 7
#>    STUDYID USUBJID PARAMCD AVISIT    ADT         AVAL FIRSTFL
#>    <chr>   <chr>   <chr>   <chr>     <date>     <dbl> <chr>
#>  1 AB123   1015    TEMP    BASELINE  2021-04-25  39   Y
#>  2 AB123   1015    TEMP    BASELINE  2021-04-27  38   <NA>
#>  3 AB123   1015    TEMP    WEEK 2    2021-05-10  37.5 <NA>
#>  4 AB123   1015    WEIGHT  SCREENING 2021-04-19  81.2 Y
#>  5 AB123   1015    WEIGHT  BASELINE  2021-04-25  82.7 <NA>
#>  6 AB123   1015    WEIGHT  BASELINE  2021-04-27  84   <NA>
#>  7 AB123   1015    WEIGHT  WEEK 2    2021-05-09  82.5 <NA>
#>  8 AB123   1023    TEMP    SCREENING 2021-04-27  38   Y
#>  9 AB123   1023    TEMP    BASELINE  2021-04-28  37.5 <NA>
#> 10 AB123   1023    TEMP    BASELINE  2021-04-29  37.5 <NA>
#> 11 AB123   1023    TEMP    WEEK 1    2021-05-03  37   <NA>
#> 12 AB123   1023    WEIGHT  SCREENING 2021-04-27  69.6 Y
#> 13 AB123   1023    WEIGHT  BASELINE  2021-04-29  67.2 <NA>
#> 14 AB123   1023    WEIGHT  WEEK 1    2021-05-02  65.9 <NA>
```

Modifying the flag values (`true_value`**,** `false_value`**):**

The previous example is now enhanced with custom values for the flag entries. Records which are flagged are filled with the contents of `true_value` and those which are not are filled with the contents of `false_value`. Note that these are normally preset to `"Y"` and `NA`, which is why they were not specified in the example above.

```
advs %>%
  derive_var_extreme_flag(
    by_vars = exprs(STUDYID, USUBJID, PARAMCD),
    order = exprs(ADT),
    new_var = LASTFL,
    mode = "last",
    true_value = "Yes",
    false_value = "No",
  ) %>%
  arrange(STUDYID, USUBJID, PARAMCD, ADT) %>%
  select(STUDYID, everything())
#> # A tibble: 14 × 7
#>    STUDYID USUBJID PARAMCD AVISIT    ADT         AVAL LASTFL
#>    <chr>   <chr>   <chr>   <chr>     <date>     <dbl> <chr>
#>  1 AB123   1015    TEMP    BASELINE  2021-04-25  39   No
#>  2 AB123   1015    TEMP    BASELINE  2021-04-27  38   No
#>  3 AB123   1015    TEMP    WEEK 2    2021-05-10  37.5 Yes
#>  4 AB123   1015    WEIGHT  SCREENING 2021-04-19  81.2 No
#>  5 AB123   1015    WEIGHT  BASELINE  2021-04-25  82.7 No
#>  6 AB123   1015    WEIGHT  BASELINE  2021-04-27  84   No
#>  7 AB123   1015    WEIGHT  WEEK 2    2021-05-09  82.5 Yes
#>  8 AB123   1023    TEMP    SCREENING 2021-04-27  38   No
#>  9 AB123   1023    TEMP    BASELINE  2021-04-28  37.5 No
```

```
#> 10 AB123   1023    TEMP    BASELINE  2021-04-29  37.5 No
#> 11 AB123   1023    TEMP    WEEK 1    2021-05-03  37   Yes
#> 12 AB123   1023    WEIGHT  SCREENING 2021-04-27  69.6 No
#> 13 AB123   1023    WEIGHT  BASELINE  2021-04-29  67.2 No
#> 14 AB123   1023    WEIGHT  WEEK 1    2021-05-02  65.9 Yes
```

Creating temporary variables for sorting (`check_type`):

In this example we wish to flag the first occurrence of the most severe AE within each subject. To ensure correct sorting of the severity values, `AESEV` must be pre-processed into a numeric variable `TEMP_AESEVN` which can then be passed inside `order`. Once again, to ensure we only flag the *first* occurrence, we specify `AESTDY` and `AESEQ` inside `order` as well.

```
adae %>%
  mutate(
    TEMP_AESEVN =
      as.integer(factor(AESEV, levels = c("SEVERE", "MODERATE", "MILD")))
  ) %>%
  derive_var_extreme_flag(
    new_var = AOCCIFL,
    by_vars = exprs(STUDYID, USUBJID),
    order = exprs(TEMP_AESEVN, AESTDY, AESEQ),
    mode = "first",
    check_type = "warning"
  ) %>%
  arrange(STUDYID, USUBJID, AESTDY, AESEQ) %>%
  select(STUDYID, USUBJID, AEDECOD, AESEV, AESTDY, AESEQ, AOCCIFL)
#> # A tibble: 7 × 7
#>   STUDYID USUBJID AEDECOD   AESEV  AESTDY AESEQ AOCCIFL
#>   <chr>   <chr>   <chr>     <chr>   <dbl> <dbl> <chr>
#> 1 AB123   1015    ERYTHEMA  MILD        2     1 Y
#> 2 AB123   1015    PRURITUS  MILD        2     2 <NA>
#> 3 AB123   1015    DIARRHOEA MILD        8     3 <NA>
#> 4 AB123   1023    ERYTHEMA  MILD        3     1 <NA>
#> 5 AB123   1023    ERYTHEMA  SEVERE      5     2 Y
#> 6 AB123   1023    ERYTHEMA  MILD        8     3 <NA>
#> 7 AB123   1023    AV BLOCK  MILD       22     4 <NA>
```

Note here that the presence of `AESEQ` as a sorting variable inside the `order` argument ensures that the combination of `by_vars` and `order` indexes unique records in the dataset. If this had been omitted, the choice of `check_type = "warning"` would have ensured that `derive_var_extreme_flag()` would throw a warning due to perceived duplicate records (in this case, the first two AEs for subject `"1015"`). If no sorting variables exist, or if these duplicates are acceptable, then the user can silence the warning with `check_type = "none"`. Alternatively, the warning can be upgraded to an error with `check_type = "error"`.

Flagging all records if multiple are identified (`flag_all`):

Revisiting the above example, if we instead wish to flag *all* AEs of the highest severity occurring on the earliest date, then we can use `flag_all = TRUE`. Note that we now also omit `AESEQ` from the `order` argument because we do not need to differentiate between two AEs occurring on the same day (e.g. for subject `"1015"`) as they are both flagged.

```
adae %>%
  mutate(
    TEMP_AESEVN =
      as.integer(factor(AESEV, levels = c("SEVERE", "MODERATE", "MILD")))
  ) %>%
  derive_var_extreme_flag(
    new_var = AOCCIFL,
    by_vars = exprs(STUDYID, USUBJID),
    order = exprs(TEMP_AESEVN, AESTDY),
    mode = "first",
    flag_all = TRUE
  ) %>%
  arrange(STUDYID, USUBJID, AESTDY, AESEQ) %>%
  select(STUDYID, USUBJID, AEDECOD, AESEV, AESTDY, AESEQ, AOCCIFL)
#> # A tibble: 7 × 7
#>   STUDYID USUBJID AEDECOD   AESEV  AESTDY AESEQ AOCCIFL
#>   <chr>   <chr>   <chr>     <chr>   <dbl> <dbl> <chr>
#> 1 AB123   1015    ERYTHEMA  MILD        2     1 Y
#> 2 AB123   1015    PRURITUS  MILD        2     2 Y
#> 3 AB123   1015    DIARRHOEA MILD        8     3 <NA>
#> 4 AB123   1023    ERYTHEMA  MILD        3     1 <NA>
#> 5 AB123   1023    ERYTHEMA  SEVERE      5     2 Y
#> 6 AB123   1023    ERYTHEMA  MILD        8     3 <NA>
#> 7 AB123   1023    AV BLOCK  MILD       22     4 <NA>
```

Deriving a baseline flag:

`derive_var_extreme_flag()` is very often used to derive the baseline flag `ABLFL`, so the following section contains various examples of this in action for the `ADVS` dataset. Note that for these derivations it is often convenient to leverage higher order functions such as `restrict_derivation()` and `slice_derivation()`. Please read the Higher Order Functions vignette, as well as their specific reference pages, to learn more.

To set the baseline flag for the last observation among those where `AVISIT = "BASELINE"`, we can use a similar call to the examples above but wrapping inside of `restrict_derivation()` and making use of the `filter` argument.

```
restrict_derivation(
  advs,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = exprs(USUBJID, PARAMCD),
    order = exprs(ADT),
    new_var = ABLFL,
    mode = "last"
  ),
  filter = AVISIT == "BASELINE"
) %>%
  arrange(STUDYID, USUBJID, PARAMCD, ADT) %>%
  select(STUDYID, everything())
#> # A tibble: 14 × 7
```

```
#>    STUDYID USUBJID PARAMCD AVISIT    ADT         AVAL ABLFL
#>    <chr>   <chr>   <chr>   <chr>     <date>     <dbl> <chr>
#>  1 AB123   1015    TEMP    BASELINE  2021-04-25  39   <NA>
#>  2 AB123   1015    TEMP    BASELINE  2021-04-27  38   Y
#>  3 AB123   1015    TEMP    WEEK 2    2021-05-10  37.5 <NA>
#>  4 AB123   1015    WEIGHT  SCREENING 2021-04-19  81.2 <NA>
#>  5 AB123   1015    WEIGHT  BASELINE  2021-04-25  82.7 <NA>
#>  6 AB123   1015    WEIGHT  BASELINE  2021-04-27  84   Y
#>  7 AB123   1015    WEIGHT  WEEK 2    2021-05-09  82.5 <NA>
#>  8 AB123   1023    TEMP    SCREENING 2021-04-27  38   <NA>
#>  9 AB123   1023    TEMP    BASELINE  2021-04-28  37.5 <NA>
#> 10 AB123   1023    TEMP    BASELINE  2021-04-29  37.5 Y
#> 11 AB123   1023    TEMP    WEEK 1    2021-05-03  37   <NA>
#> 12 AB123   1023    WEIGHT  SCREENING 2021-04-27  69.6 <NA>
#> 13 AB123   1023    WEIGHT  BASELINE  2021-04-29  67.2 Y
#> 14 AB123   1023    WEIGHT  WEEK 1    2021-05-02  65.9 <NA>
```

Alternatively, to set baseline as the lowest observation among those where `AVISIT = "BASELINE"` (selecting the latest if there are multiple) we can modify the `order` argument, ensuring to sort by descending `AVAL` before `ADT`. Note here the synergy between `desc()` and `mode`, because `mode = "last"` applies to both the ordering variables `AVAL` and `ADT` and so we need to reverse only the ordering of the former to ensure that the lowest value is selected but also that the latest one among multiple is preferred. This is relevant for subject `"1023"`'s temperature records.

```
restrict_derivation(
  advs,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = exprs(USUBJID, PARAMCD),
    order = exprs(desc(AVAL), ADT),
    new_var = ABLFL,
    mode = "last"
  ),
  filter = AVISIT == "BASELINE"
) %>%
  arrange(STUDYID, USUBJID, PARAMCD, ADT) %>%
  select(STUDYID, everything())
#> # A tibble: 14 × 7
#>    STUDYID USUBJID PARAMCD AVISIT    ADT         AVAL ABLFL
#>    <chr>   <chr>   <chr>   <chr>     <date>     <dbl> <chr>
#>  1 AB123   1015    TEMP    BASELINE  2021-04-25  39   <NA>
#>  2 AB123   1015    TEMP    BASELINE  2021-04-27  38   Y
#>  3 AB123   1015    TEMP    WEEK 2    2021-05-10  37.5 <NA>
#>  4 AB123   1015    WEIGHT  SCREENING 2021-04-19  81.2 <NA>
#>  5 AB123   1015    WEIGHT  BASELINE  2021-04-25  82.7 Y
#>  6 AB123   1015    WEIGHT  BASELINE  2021-04-27  84   <NA>
#>  7 AB123   1015    WEIGHT  WEEK 2    2021-05-09  82.5 <NA>
#>  8 AB123   1023    TEMP    SCREENING 2021-04-27  38   <NA>
#>  9 AB123   1023    TEMP    BASELINE  2021-04-28  37.5 <NA>
```

```
#> 10 AB123   1023    TEMP    BASELINE  2021-04-29  37.5 Y
#> 11 AB123   1023    TEMP    WEEK 1    2021-05-03  37   <NA>
#> 12 AB123   1023    WEIGHT  SCREENING 2021-04-27  69.6 <NA>
#> 13 AB123   1023    WEIGHT  BASELINE  2021-04-29  67.2 Y
#> 14 AB123   1023    WEIGHT  WEEK 1    2021-05-02  65.9 <NA>
```

In practice, baseline-setting may vary on a parameter by parameter basis, in which case `slice_derivation()` could be used in place of `restrict_derivation()`. In the example below, we set the baseline flag as follows: for temperature records, as the lowest value recorded at a baseline visit; for weight records, as the highest value recorded at a baseline visit. In both cases, we again select the latest observation if there are multiple.

```
slice_derivation(
  advs,
  derivation = derive_var_extreme_flag,
  args = params(
    by_vars = exprs(USUBJID, PARAMCD),
    mode = "last",
    new_var = ABLFL,
  ),
  derivation_slice(
    filter = AVISIT == "BASELINE" & PARAMCD == "TEMP",
    args = params(order = exprs(desc(AVAL), ADT))
  ),
  derivation_slice(
    filter = AVISIT == "BASELINE" & PARAMCD == "WEIGHT",
    args = params(order = exprs(AVAL, ADT))
  )
) %>%
  arrange(STUDYID, USUBJID, PARAMCD, ADT) %>%
  select(STUDYID, everything())
#> # A tibble: 14 × 7
#>    STUDYID USUBJID PARAMCD AVISIT    ADT         AVAL ABLFL
#>    <chr>   <chr>   <chr>   <chr>     <date>     <dbl> <chr>
#>  1 AB123   1015    TEMP    BASELINE  2021-04-25  39   <NA>
#>  2 AB123   1015    TEMP    BASELINE  2021-04-27  38   Y
#>  3 AB123   1015    TEMP    WEEK 2    2021-05-10  37.5 <NA>
#>  4 AB123   1015    WEIGHT  SCREENING 2021-04-19  81.2 <NA>
#>  5 AB123   1015    WEIGHT  BASELINE  2021-04-25  82.7 <NA>
#>  6 AB123   1015    WEIGHT  BASELINE  2021-04-27  84   Y
#>  7 AB123   1015    WEIGHT  WEEK 2    2021-05-09  82.5 <NA>
#>  8 AB123   1023    TEMP    SCREENING 2021-04-27  38   <NA>
#>  9 AB123   1023    TEMP    BASELINE  2021-04-28  37.5 <NA>
#> 10 AB123   1023    TEMP    BASELINE  2021-04-29  37.5 Y
#> 11 AB123   1023    TEMP    WEEK 1    2021-05-03  37   <NA>
#> 12 AB123   1023    WEIGHT  SCREENING 2021-04-27  69.6 <NA>
#> 13 AB123   1023    WEIGHT  BASELINE  2021-04-29  67.2 Y
#> 14 AB123   1023    WEIGHT  WEEK 1    2021-05-02  65.9 <NA>
```

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

`derive_var_joined_exist_flag` *Derives a Flag Based on an Existing Flag*

Description

Derive a flag which depends on other observations of the dataset. For example, flagging events which need to be confirmed by a second event.

Usage

```
derive_var_joined_exist_flag(
  dataset,
  dataset_add,
  by_vars,
  order = NULL,
  new_var,
  tmp_obs_nr_var = NULL,
  join_vars,
  join_type,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  filter_add = NULL,
  filter_join,
  true_value = "Y",
  false_value = NA_character_,
  check_type = "warning"
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `join_vars` arguments are expected to be in the dataset. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** none
`dataset_add`	Additional dataset The variables specified for `by_vars`, `join_vars`, and `order` are expected.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`by_vars` Grouping variables

The specified variables are used for joining the input dataset (`dataset`) with the additional dataset (`dataset_add`).

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`order` Order

The observations are ordered by the specified order if `join_type = "after"`, `join_type = "before"`, `first_cond_lower`, `first_cond_upper`, or `tmp_obs_nr_var` are specified.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`new_var` New variable

The specified variable is added to the input dataset.

Permitted values an unquoted symbol, e.g., `AVAL`

Default value none

`tmp_obs_nr_var` Temporary observation number

The specified variable is added to the input dataset (`dataset`) and the additional dataset (`dataset_add`). It is set to the observation number with respect to `order`. For each by group (`by_vars`) the observation number starts with `1`. If there is more than one record for specific values for `by_vars` and `order`, all records get the same observation number. By default, a warning (see `check_type`) is issued in this case. The variable can be used in the conditions (`filter_join`, `first_cond_upper`, `first_cond_lower`). It is not included in the output dataset. It can also be used to flag consecutive observations or the last observation (see last example below).

Permitted values an unquoted symbol, e.g., `AVAL`

Default value `NULL`

`join_vars` Variables to keep from joined dataset

The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to flag all observations with `AVALC == "Y"` and `AVALC == "Y"` for at least one subsequent visit `join_vars = exprs(AVALC, AVISITN)` and `filter_join = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join` could be specified.

The `*.join` variables are not included in the output dataset.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`join_type` Observations to keep after joining

The argument determines which of the joined observations are kept with respect to the original observation. For example, if `join_type = "after"` is specified all observations after the original observations are kept.

For example for confirmed response or BOR in the oncology setting or confirmed deterioration in questionnaires the confirmatory assessment must be after the assessment. Thus `join_type = "after"` could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation. For example, to identify AEs occurring on or after seven days before a COVID AE. Thus `join_type = "all"` could be used.

Permitted values `"before"`, `"after"`, `"all"`

Default value none

`first_cond_lower`

Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

This parameter should be specified if `filter_join` contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For an example see the last example below.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`first_cond_upper`

Condition for selecting range of data (after)

If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

This parameter should be specified if `filter_join` contains summary functions which should not apply to all observations but only up to the confirmation assessment. For an example see the third example below.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`filter_add` Filter for additional dataset (`dataset_add`)

Only observations from `dataset_add` fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by `order` or `new_vars` arguments can be used in the condition.

The condition can include summary functions like `all()` or `any()`. The additional dataset is grouped by the by variables (`by_vars`).

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

filter_join — Condition for selecting observations

The filter is applied to the joined dataset for flagging the confirmed observations. The condition can include summary functions like `all()` or `any()`. The joined dataset is grouped by the original observations. I.e., the summary function are applied to all observations up to the confirmation observation. For example, `filter_join = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) & count_vals(var = AVALC.join, val = "NE") <= 1` selects observations with response "CR" and for all observations up to the confirmation observation the response is "CR" or "NE" and there is at most one "NE".

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value none

true_value — Value of `new_var` for flagged observations

Permitted values a character scalar, i.e., a character vector of length one

Default value `"Y"`

false_value — Value of `new_var` for observations not flagged

Permitted values a character scalar, i.e., a character vector of length one

Default value `NA_character_`

check_type — Check uniqueness?

If `"message"`, `"warning"`, or `"error"` is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values `"none"`, `"message"`, `"warning"`, `"error"`

Default value `"warning"`

Details

An example usage might be flagging if a patient received two required medications within a certain timeframe of each other.

In the oncology setting, for example, the function could be used to flag if a response value can be confirmed by an other assessment. This is commonly used in endpoints such as best overall response.

The following steps are performed to produce the output dataset.

Step 1:

- The variables specified by `order` are added to the additional dataset (`dataset_add`).
- The variables specified by `join_vars` are added to the additional dataset (`dataset_add`).
- The records from the additional dataset (`dataset_add`) are restricted to those matching the `filter_add` condition.

The input dataset (`dataset`) is joined with the restricted additional dataset by the variables specified for `by_vars`. From the additional dataset only the variables specified for `join_vars` are kept. The suffix ".join" is added to those variables which also exist in the input dataset.

For example, for `by_vars = USUBJID`, `join_vars = exprs(AVISITN, AVALC)` and input dataset and additional dataset

```
# A tibble: 2 x 4
USUBJID AVISITN AVALC  AVAL
<chr>     <dbl> <chr> <dbl>
1             1 Y         1
1             2 N         0
```

the joined dataset is

```
A tibble: 4 x 6
USUBJID AVISITN AVALC  AVAL AVISITN.join AVALC.join
<chr>     <dbl> <chr> <dbl>        <dbl> <chr>
1             1 Y         1            1 Y
1             1 Y         1            2 N
1             2 N         0            1 Y
1             2 N         0            2 N
```

Step 2:

The joined dataset is restricted to observations with respect to `join_type` and `order`.

The dataset from the example in the previous step with `join_type = "after"` and `order = exprs(AVISITN)` is restricted to

```
A tibble: 4 x 6
USUBJID AVISITN AVALC  AVAL AVISITN.join AVALC.join
<chr>     <dbl> <chr> <dbl>        <dbl> <chr>
1             1 Y         1            2 N
```

Step 3:

If `first_cond_lower` is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where `first_cond_lower` is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

If `first_cond_upper` is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where `first_cond_upper` is fulfilled (the observation fulfilling the condition is included). If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

For examples see the "Examples" section.

Step 4:

The joined dataset is grouped by the observations from the input dataset and restricted to the observations fulfilling the condition specified by `filter_join`.

Step 5:

The first observation of each group is selected.

Step 6:

The variable specified by `new_var` is added to the input dataset. It is set to `true_value` for all observations which were selected in the previous step. For the other observations it is set to `false_value`.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option `save_memory` to `TRUE` (see `set_admiral_options()`). This reduces the memory consumption but increases the run-time.

Value

The input dataset with the variable specified by `new_var` added.

Examples

Flag records considering other records (`filter_join`, `join_vars`):

In this example, records with a duration longer than 30 and where a COVID AE (`ACOVFL == "Y"`) occurred before or up to seven days after the record should be flagged. The condition for flagging the records is specified by the `filter_join` argument. Variables from the other records are referenced by variable names with the suffix `.join`. These variables have to be specified for the `join_vars` argument. As records before *and* after the current record should be considered, `join_type = "all"` is specified.

```
library(tibble)

adae <- tribble(
  ~USUBJID, ~ADY, ~ACOVFL, ~ADURN,
  "1",         10, "N",          1,
  "1",         21, "N",         50,
  "1",         23, "Y",         14,
  "1",         32, "N",         31,
  "1",         42, "N",         20,
  "2",         11, "Y",         13,
  "2",         23, "N",          2,
  "3",         13, "Y",         12,
  "4",         14, "N",         32,
  "4",         21, "N",         41
)

derive_var_joined_exist_flag(
  adae,
  dataset_add = adae,
  new_var = ALCOVFL,
  by_vars = exprs(USUBJID),
  join_vars = exprs(ACOVFL, ADY),
  join_type = "all",
  filter_join = ADURN > 30 & ACOVFL.join == "Y" & ADY.join <= ADY + 7
)
#> # A tibble: 10 × 5
#>    USUBJID   ADY ACOVFL ADURN ALCOVFL
#>    <chr>   <dbl> <chr>  <dbl> <chr>
#>  1 1          10 N          1 <NA>
#>  2 1          21 N         50 Y
#>  3 1          23 Y         14 <NA>
#>  4 1          32 N         31 Y
#>  5 1          42 N         20 <NA>
#>  6 2          11 Y         13 <NA>
#>  7 2          23 N          2 <NA>
```

```
#>  8 3          13 Y         12 <NA>
#>  9 4          14 N         32 <NA>
#> 10 4          21 N         41 <NA>
```

Considering only records after the current one (`join_type = "after"`, `true_value`, `false_value`):

In this example, records with `AVALC == "Y"` and `AVALC == "Y"` at a subsequent visit should be flagged. `join_type = "after"` is specified to consider only records after the current one. Please note that the `order` argument must be specified, as otherwise it is not possible to determine which records are after the current record.

Please note that a numeric flag is created here by specifying the `true_value` and the `false_value` argument.

```
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "Y",
  "1",      2,        "N",
  "1",      3,        "Y",
  "1",      4,        "N",
  "2",      1,        "Y",
  "2",      2,        "N",
  "3",      1,        "Y",
  "4",      1,        "N",
  "4",      2,        "N",
)

derive_var_joined_exist_flag(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  new_var = CONFFLN,
  join_vars = exprs(AVALC, AVISITN),
  join_type = "after",
  order = exprs(AVISITN),
  filter_join = AVALC == "Y" & AVALC.join == "Y",
  true_value = 1,
  false_value = 0
)
#> # A tibble: 9 × 4
#>   USUBJID AVISITN AVALC CONFFLN
#>   <chr>     <dbl> <chr>   <dbl>
#> 1 1             1 Y           1
#> 2 1             2 N           0
#> 3 1             3 Y           0
#> 4 1             4 N           0
#> 5 2             1 Y           0
#> 6 2             2 N           0
#> 7 3             1 Y           0
#> 8 4             1 N           0
```

```
#> 9 4             2 N           0
```

Considering a range of records only (`first_cond_lower`, `first_cond_upper`):
Consider the following data.

```
myd <- tribble(
  ~subj, ~day, ~val,
  "1",      1, "++",
  "1",      2, "-",
  "1",      3, "0",
  "1",      4, "+",
  "1",      5, "++",
  "1",      6, "-",
  "2",      1, "-",
  "2",      2, "++",
  "2",      3, "+",
  "2",      4, "0",
  "2",      5, "-",
  "2",      6, "++"
)
```

To flag `"0"` where all results from the first `"++"` before the `"0"` up to the `"0"` (excluding the `"0"`) are `"+"` or `"++"` the `first_cond_lower` argument and `join_type = "before"` are specified.

```
derive_var_joined_exist_flag(
  myd,
  dataset_add = myd,
  by_vars = exprs(subj),
  order = exprs(day),
  new_var = flag,
  join_vars = exprs(val),
  join_type = "before",
  first_cond_lower = val.join == "++",
  filter_join = val == "0" & all(val.join %in% c("+", "++"))
)
#> # A tibble: 12 × 4
#>    subj    day val   flag
#>    <chr> <dbl> <chr> <chr>
#>  1 1         1 ++    <NA>
#>  2 1         2 -     <NA>
#>  3 1         3 0     <NA>
#>  4 1         4 +     <NA>
#>  5 1         5 ++    <NA>
#>  6 1         6 -     <NA>
#>  7 2         1 -     <NA>
#>  8 2         2 ++    <NA>
#>  9 2         3 +     <NA>
#> 10 2         4 0     Y
#> 11 2         5 -     <NA>
#> 12 2         6 ++    <NA>
```

To flag "0" where all results from the "0" (excluding the "0") up to the first "++" after the "0" are "+" or "++" the first_cond_upper argument and join_type = "after" are specified.

```
derive_var_joined_exist_flag(
  myd,
  dataset_add = myd,
  by_vars = exprs(subj),
  order = exprs(day),
  new_var = flag,
  join_vars = exprs(val),
  join_type = "after",
  first_cond_upper = val.join == "++",
  filter_join = val == "0" & all(val.join %in% c("+", "++"))
)
#> # A tibble: 12 × 4
#>    subj    day val   flag
#>    <chr> <dbl> <chr> <chr>
#>  1 1         1 ++    <NA>
#>  2 1         2 -     <NA>
#>  3 1         3 0     Y
#>  4 1         4 +     <NA>
#>  5 1         5 ++    <NA>
#>  6 1         6 -     <NA>
#>  7 2         1 -     <NA>
#>  8 2         2 ++    <NA>
#>  9 2         3 +     <NA>
#> 10 2         4 0     <NA>
#> 11 2         5 -     <NA>
#> 12 2         6 ++    <NA>
```

Considering only records up to a condition (first_cond_upper)**:**

In this example from deriving confirmed response in oncology, the records with

- AVALC == "CR",
- AVALC == "CR" at a subsequent visit,
- only "CR" or "NE" in between, and
- at most one "NE" in between

should be flagged. The other records to be considered are restricted to those up to the first occurrence of "CR" by specifying the first_cond_upper argument. The count_vals() function is used to count the "NE"s for the last condition.

```
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "NE",
  "1",      4,        "CR",
  "1",      5,        "NE",
  "2",      1,        "CR",
```

```
  "2",      2,        "PR",
  "2",      3,        "CR",
  "3",      1,        "CR",
  "4",      1,        "CR",
  "4",      2,        "NE",
  "4",      3,        "NE",
  "4",      4,        "CR",
  "4",      5,        "PR"
)

derive_var_joined_exist_flag(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  join_vars = exprs(AVALC),
  join_type = "after",
  order = exprs(AVISITN),
  new_var = CONFFL,
  first_cond_upper = AVALC.join == "CR",
  filter_join = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) &
    count_vals(var = AVALC.join, val = "NE") <= 1
)
#> # A tibble: 14 × 4
#>    USUBJID AVISITN AVALC CONFFL
#>    <chr>     <dbl> <chr> <chr>
#>  1 1             1 PR    <NA>
#>  2 1             2 CR    Y
#>  3 1             3 NE    <NA>
#>  4 1             4 CR    <NA>
#>  5 1             5 NE    <NA>
#>  6 2             1 CR    <NA>
#>  7 2             2 PR    <NA>
#>  8 2             3 CR    <NA>
#>  9 3             1 CR    <NA>
#> 10 4             1 CR    <NA>
#> 11 4             2 NE    <NA>
#> 12 4             3 NE    <NA>
#> 13 4             4 CR    <NA>
#> 14 4             5 PR    <NA>
```

Considering order of values (`min_cond()`, `max_cond()`)**:**

In this example from deriving confirmed response in oncology, records with

- `AVALC == "PR"`,
- `AVALC == "CR"` or `AVALC == "PR"` at a subsequent visit at least 20 days later,
- only `"CR"`, `"PR"`, or `"NE"` in between,
- at most one `"NE"` in between, and
- `"CR"` is not followed by `"PR"`

should be flagged. The last condition is realized by using min_cond() and max_cond(), ensuring that the first occurrence of "CR" is after the last occurrence of "PR". The second call to count_vals() in the condition is required to cover the case of no "CR"s (the min_cond() call returns NA then).

```
data <- tribble(
  ~USUBJID, ~ADY, ~AVALC,
  "1",         6, "PR",
  "1",        12, "CR",
  "1",        24, "NE",
  "1",        32, "CR",
  "1",        48, "PR",
  "2",         3, "PR",
  "2",        21, "CR",
  "2",        33, "PR",
  "3",        11, "PR",
  "4",         7, "PR",
  "4",        12, "NE",
  "4",        24, "NE",
  "4",        32, "PR",
  "4",        55, "PR"
)

derive_var_joined_exist_flag(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  join_vars = exprs(AVALC, ADY),
  join_type = "after",
  order = exprs(ADY),
  new_var = CONFFL,
  first_cond_upper = AVALC.join %in% c("CR", "PR") & ADY.join - ADY >= 20,
  filter_join = AVALC == "PR" &
    all(AVALC.join %in% c("CR", "PR", "NE")) &
    count_vals(var = AVALC.join, val = "NE") <= 1 &
    (
      min_cond(var = ADY.join, cond = AVALC.join == "CR") >
        max_cond(var = ADY.join, cond = AVALC.join == "PR") |
        count_vals(var = AVALC.join, val = "CR") == 0
    )
)
#> # A tibble: 14 × 4
#>    USUBJID   ADY AVALC CONFFL
#>    <chr>   <dbl> <chr> <chr>
#>  1 1           6 PR    <NA>
#>  2 1          12 CR    <NA>
#>  3 1          24 NE    <NA>
#>  4 1          32 CR    <NA>
#>  5 1          48 PR    <NA>
```

```
#>  6 2           3 PR    <NA>
#>  7 2          21 CR    <NA>
#>  8 2          33 PR    <NA>
#>  9 3          11 PR    <NA>
#> 10 4           7 PR    <NA>
#> 11 4          12 NE    <NA>
#> 12 4          24 NE    <NA>
#> 13 4          32 PR    Y
#> 14 4          55 PR    <NA>
```

Considering the order of records (`tmp_obs_nr_var`)**:**

In this example, the records with `CRIT1FL == "Y"` at two consecutive visits or at the last visit should be flagged. A temporary order variable is created by specifying the `tmp_obs_nr_var` argument. Then it is used in `filter_join`. The temporary variable doesn't need to be specified for `join_vars`.

```
data <- tribble(
  ~USUBJID, ~AVISITN, ~CRIT1FL,
  "1",      1,        "Y",
  "1",      2,        "N",
  "1",      3,        "Y",
  "1",      5,        "N",
  "2",      1,        "Y",
  "2",      3,        "Y",
  "2",      5,        "N",
  "3",      1,        "Y",
  "4",      1,        "Y",
  "4",      2,        "N",
)

derive_var_joined_exist_flag(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  new_var = CONFFL,
  tmp_obs_nr_var = tmp_obs_nr,
  join_vars = exprs(CRIT1FL),
  join_type = "all",
  order = exprs(AVISITN),
  filter_join = CRIT1FL == "Y" & CRIT1FL.join == "Y" &
    (tmp_obs_nr + 1 == tmp_obs_nr.join | tmp_obs_nr == max(tmp_obs_nr.join))
)
#> # A tibble: 10 × 4
#>    USUBJID AVISITN CRIT1FL CONFFL
#>    <chr>     <dbl> <chr>   <chr>
#>  1 1             1 Y       <NA>
#>  2 1             2 N       <NA>
#>  3 1             3 Y       <NA>
#>  4 1             5 N       <NA>
```

```
#>  5 2             1 Y       Y
#>  6 2             3 Y       <NA>
#>  7 2             5 N       <NA>
#>  8 3             1 Y       Y
#>  9 4             1 Y       <NA>
#> 10 4             2 N       <NA>
```

Flag each dose which is lower than the previous dose (`tmp_obs_nr_var`)**:**

```
ex <- tribble(
  ~USUBJID, ~EXSTDTM,           ~EXDOSE,
  "1",      "2024-01-01T08:00",       2,
  "1",      "2024-01-02T08:00",       4,
  "2",      "2024-01-01T08:30",       1,
  "2",      "2024-01-02T08:30",       4,
  "2",      "2024-01-03T08:30",       3,
  "2",      "2024-01-04T08:30",       2,
  "2",      "2024-01-05T08:30",       2
)

derive_var_joined_exist_flag(
  ex,
  dataset_add = ex,
  by_vars = exprs(USUBJID),
  order = exprs(EXSTDTM),
  new_var = DOSREDFL,
  tmp_obs_nr_var = tmp_dose_nr,
  join_vars = exprs(EXDOSE),
  join_type = "before",
  filter_join = (
    tmp_dose_nr == tmp_dose_nr.join + 1 # Look only at adjacent doses
    & EXDOSE > 0 & EXDOSE.join > 0 # Both doses are valid
    & EXDOSE < EXDOSE.join # Dose is lower than previous
  )
)
#> # A tibble: 7 × 4
#>   USUBJID EXSTDTM          EXDOSE DOSREDFL
#>   <chr>   <chr>             <dbl> <chr>
#> 1 1       2024-01-01T08:00      2 <NA>
#> 2 1       2024-01-02T08:00      4 <NA>
#> 3 2       2024-01-01T08:30      1 <NA>
#> 4 2       2024-01-02T08:30      4 <NA>
#> 5 2       2024-01-03T08:30      3 Y
#> 6 2       2024-01-04T08:30      2 Y
#> 7 2       2024-01-05T08:30      2 <NA>
```

Derive definitive deterioration flag:

In this example a definitive deterioration flag should be derived as any deterioration (`CHGCAT1 = "Worsened"`) by parameter that is not followed by a non-deterioration. Please note that `join_type = "after"` can't by used here, as otherwise the last record wouldn't be flagged.

```
adqs <- tribble(
  ~USUBJID, ~PARAMCD, ~ADY, ~CHGCAT1,
  "1",      "QS1",      10, "Improved",
  "1",      "QS1",      21, "Improved",
  "1",      "QS1",      23, "Improved",
  "1",      "QS2",      32, "Worsened",
  "1",      "QS2",      42, "Improved",
  "2",      "QS1",      11, "Worsened",
  "2",      "QS1",      24, "Worsened"
)

derive_var_joined_exist_flag(
  adqs,
  dataset_add = adqs,
  new_var = DDETERFL,
  by_vars = exprs(USUBJID, PARAMCD),
  join_vars = exprs(CHGCAT1, ADY),
  join_type = "all",
  filter_join = all(CHGCAT1.join == "Worsened" | ADY > ADY.join)
)
#> # A tibble: 7 × 5
#>   USUBJID PARAMCD   ADY CHGCAT1  DDETERFL
#>   <chr>   <chr>   <dbl> <chr>    <chr>
#> 1 1       QS1        10 Improved <NA>
#> 2 1       QS1        21 Improved <NA>
#> 3 1       QS1        23 Improved <NA>
#> 4 1       QS2        32 Worsened <NA>
#> 5 1       QS2        42 Improved <NA>
#> 6 2       QS1        11 Worsened Y
#> 7 2       QS1        24 Worsened Y
```

Handling duplicates (check_type)**:**

If the order argument is used, it is checked if the records are unique with respect to by_vars and order. Consider for example the derivation of CONFFL which flags records with AVALC == "Y" which are confirmed at a subsequent visit.

```
data <- tribble(
  ~USUBJID, ~AVISITN, ~ADY, ~AVALC,
  "1",      1,          1, "Y",
  "1",      2,          8, "N",
  "1",      3,         15, "Y",
  "1",      4,         22, "N",
  "2",      1,          1, "Y",
  "2",      2,          8, "Y",
  "2",      2,         10, "Y"
)

derive_var_joined_exist_flag(
  data,
```

```
  dataset_add = data,
  by_vars = exprs(USUBJID),
  new_var = CONFFL,
  join_vars = exprs(AVALC, AVISITN),
  join_type = "after",
  order = exprs(AVISITN),
  filter_join = AVALC == "Y" & AVALC.join == "Y"
)
#> # A tibble: 7 × 5
#>   USUBJID AVISITN   ADY AVALC CONFFL
#>   <chr>     <dbl> <dbl> <chr> <chr>
#> 1 1             1     1 Y     Y
#> 2 1             2     8 N     <NA>
#> 3 1             3    15 Y     <NA>
#> 4 1             4    22 N     <NA>
#> 5 2             1     1 Y     Y
#> 6 2             2     8 Y     <NA>
#> 7 2             2    10 Y     <NA>
#> Warning: Dataset `dataset` contains duplicate records with respect to `USUBJID` and
#> `AVISITN`
#> i Run `admiral::get_duplicates_dataset()` to access the duplicate records
#> Warning: Dataset `dataset_add` contains duplicate records with respect to `USUBJID` and
#> `AVISITN`
#> i Run `admiral::get_duplicates_dataset()` to access the duplicate records
```

The records for USUBJID == "2" are not unique with respect to USUBJID and AVISITN. Thus a warning is issued. The duplicates can be accessed by calling get_duplicates_dataset():

```
get_duplicates_dataset()
#> Duplicate records with respect to `USUBJID` and `AVISITN`.
#> # A tibble: 2 × 4
#>   USUBJID AVISITN   ADY AVALC
#> * <chr>     <dbl> <dbl> <chr>
#> 1 2             2     8 Y
#> 2 2             2    10 Y
```

In this example, confirmation is required at a subsequent *visit*. Please note that the first record for subject "2" at visit 2 is not flagged. Thus the warning can be suppressed by specifying check_type = "none".

```
derive_var_joined_exist_flag(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  new_var = CONFFL,
  join_vars = exprs(AVALC, AVISITN),
  join_type = "after",
  order = exprs(AVISITN),
  filter_join = AVALC == "Y" & AVALC.join == "Y",
  check_type = "none"
```

```
)
#> # A tibble: 7 × 5
#>   USUBJID AVISITN   ADY AVALC CONFFL
#>   <chr>     <dbl> <dbl> <chr> <chr>
#> 1 1             1     1 Y     Y
#> 2 1             2     8 N     <NA>
#> 3 1             3    15 Y     <NA>
#> 4 1             4    22 N     <NA>
#> 5 2             1     1 Y     Y
#> 6 2             2     8 Y     <NA>
#> 7 2             2    10 Y     <NA>
```

See Also

filter_joined(), derive_vars_joined()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur
derive_vars_transposed()

---

derive_var_merged_ef_msrc

Merge an Existence Flag From Multiple Sources

Description

Adds a flag variable to the input dataset which indicates if there exists at least one observation in one of the source datasets fulfilling a certain condition. For example, if a dose adjustment flag should be added to ADEX but the dose adjustment information is collected in different datasets, e.g., EX, EC, and FA.

Usage

```
derive_var_merged_ef_msrc(
  dataset,
  by_vars,
  flag_events,
  source_datasets,
  new_var,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_
)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`by_vars` Grouping variables

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`flag_events` Flag events

A list of `flag_event()` objects is expected. For each event the condition (`condition` field) is evaluated in the source dataset referenced by the `dataset_name` field. If it evaluates to `TRUE` at least once, the new variable is set to `true_value`.

Permitted values a list of `flag_event()` objects

Default value none

`source_datasets`

Source datasets

A named list of datasets is expected. The `dataset_name` field of `flag_event()` refers to the dataset provided in the list.

Permitted values named list of datasets, e.g., `list(adsl = adsl, ae = ae)`

Default value none

`new_var` New variable

The specified variable is added to the input dataset.

Permitted values an unquoted symbol, e.g., `AVAL`

Default value none

`true_value` True value

The new variable (`new_var`) is set to the specified value for all by groups for which at least one of the source object (`sources`) has the condition evaluate to `TRUE`.

The values of `true_value`, `false_value`, and `missing_value` must be of the same type.

Permitted values a character scalar, i.e., a character vector of length one

Default value `"Y"`

`false_value` False value

The new variable (`new_var`) is set to the specified value for all by groups which occur in at least one source (`sources`) but the condition never evaluates to `TRUE`.

The values of `true_value`, `false_value`, and `missing_value` must be of the same type.

Permitted values a character scalar, i.e., a character vector of length one

Default value `NA_character_`

`missing_value` Values used for missing information

The new variable is set to the specified value for all by groups without observations in any of the sources (`sources`).

The values of `true_value`, `false_value`, and `missing_value` must be of the same type.

Permitted values a character scalar, i.e., a character vector of length one

Default value `NA_character_`

Details

1. For each `flag_event()` object specified for `flag_events`: The condition (`condition`) is evaluated in the dataset referenced by `dataset_name`. If the `by_vars` field is specified the dataset is grouped by the specified variables for evaluating the condition. If named elements are used in `by_vars` like `by_vars = exprs(USUBJID, EXLNKID = ECLNKID)`, the variables are renamed after the evaluation. If the `by_vars` element is not specified, the observations are grouped by the variables specified for the `by_vars` argument.
2. The new variable (`new_var`) is added to the input dataset and set to the true value (`true_value`) if for the by group at least one condition evaluates to `TRUE` in one of the sources. It is set to the false value (`false_value`) if for the by group at least one observation exists and for all observations the condition evaluates to `FALSE` or `NA`. Otherwise, it is set to the missing value (`missing_value`).

Value

The output dataset contains all observations and variables of the input dataset and additionally the variable specified for `new_var`.

Examples

Data setup:

The following examples use the datasets below. `adsl` is the subject- level dataset onto which the flag is merged. `cm` contains concomitant medication records and `pr` contains procedure records — both are used as sources in the examples.

```
library(dplyr)

adsl <- tribble(
  ~USUBJID,
  "1",
  "2",
  "3",
  "4",
  "5"
)

cm <- tribble(
  ~USUBJID, ~CMCAT,         ~CMSEQ,
  "1",      "ANTI-CANCER",      1,
```

```
  "1",      "GENERAL",          2,
  "2",      "GENERAL",          1,
  "3",      "ANTI-CANCER",      1,
  "5",      "GENERAL",          1
)

# All records in PR are assumed to indicate cancer treatment
pr <- tribble(
  ~USUBJID, ~PRSEQ,
  "2",      1,
  "3",      1
)
```

Flagging from multiple sources (`flag_events`):

The `flag_events` argument takes a list of `flag_event()` objects, each pointing to a named source dataset and an optional `condition`. For a given by group, the new variable is set to `true_value` if the condition evaluates to `TRUE` at least once in **any** of the sources.

In the example below, an anti-cancer treatment flag `CANCTRFL` is derived from two sources:

- `cm`: flagged when `CMCAT == "ANTI-CANCER"`
- `pr`: all records qualify (no `condition` specified), so any subject with a procedure record is flagged

With the default `false_value = NA_character_` and `missing_value = NA_character_`, both subjects `"4"` and `"5"` receive `NA` — but for different reasons: subject `"5"` is present in `cm` but has no anti-cancer record (`false_value`), while subject `"4"` is absent from all sources (`missing_value`). See the next section to learn how to distinguish these two cases by setting `false_value` and `missing_value` to different values.

```
derive_var_merged_ef_msrc(
  adsl,
  by_vars = exprs(USUBJID),
  flag_events = list(
    flag_event(
      dataset_name = "cm",
      condition = CMCAT == "ANTI-CANCER"
    ),
    flag_event(
      dataset_name = "pr"
    )
  ),
  source_datasets = list(cm = cm, pr = pr),
  new_var = CANCTRFL
)
#> # A tibble: 5 × 2
#>   USUBJID CANCTRFL
#>   <chr>   <chr>
#> 1 1       Y
#> 2 2       Y
#> 3 3       Y
```

```
#> 4 4       <NA>
#> 5 5       <NA>
```

Controlling flag values (`true_value`, `false_value`, `missing_value`)**:**

By default `true_value = "Y"`, `false_value = NA_character_`, and `missing_value = NA_character_`. Setting them explicitly lets you distinguish three subject-level states:

- `true_value`: subject has at least one qualifying record in any source
- `false_value`: subject appears in at least one source, but no record meets the condition
- `missing_value`: subject has **no** records in any source

In the example below, a subject-level `ADSL` dataset is used together with dose adjustment sources (`adex`, `ec`, `fa`). This reveals all three cases in the output:

- Subjects `"1"` and `"3"`: dose adjustment found → `"Y"` via `true_value`
- Subject `"2"`: present in `adex` but no adjustment found → `"N"` via `false_value`
- Subject `"4"`: absent from all sources → `NA` via `missing_value`

```
adsl_ex <- tribble(
  ~USUBJID,
  "1",
  "2",
  "3",
  "4"
)

adex <- tribble(
  ~USUBJID, ~EXADJ,
  "1",      "DOSE REDUCED",
  "2",      NA_character_
)

ec <- tribble(
  ~USUBJID, ~ECADJ,
  "3",      "DOSE REDUCED"
)

fa <- tribble(
  ~USUBJID, ~FATESTCD, ~FAOBJ,             ~FASTRESC,
  "1",      "OCCUR",   "DOSE ADJUSTMENT", "Y"
)

derive_var_merged_ef_msrc(
  adsl_ex,
  by_vars = exprs(USUBJID),
  flag_events = list(
    flag_event(
      dataset_name = "ex",
      condition = !is.na(EXADJ)
    ),
```

```
    flag_event(
      dataset_name = "ec",
      condition = !is.na(ECADJ)
    ),
    flag_event(
      dataset_name = "fa",
    condition = FATESTCD == "OCCUR" & FAOBJ == "DOSE ADJUSTMENT" & FASTRESC == "Y"
    )
  ),
  source_datasets = list(ex = adex, ec = ec, fa = fa),
  new_var = DOSADJFL,
  true_value = "Y",
  false_value = "N",
  missing_value = NA_character_
)
#> # A tibble: 4 × 2
#>   USUBJID DOSADJFL
#>   <chr>   <chr>
#> 1 1       Y
#> 2 2       N
#> 3 3       Y
#> 4 4       <NA>
```

Per-source `by_vars` **renaming:**

When the grouping variable has a different name in a source dataset, the `by_vars` argument of `flag_event()` can be used to rename it using the `exprs(<target> = <source>)` syntax. This allows each source to use its own link variable while still merging correctly onto the input dataset.

In the example below, a dose adjustment flag `DOSADJFL` is derived for each exposure record in `adex`. The flag is set to `"Y"` if a dose adjustment is recorded in any of three sources:

- `ex`: directly via `EXADJ`
- `ec`: linked via `ECLNKID` (renamed to `EXLNKID` for the merge)
- `fa`: linked via `FALNKID` (renamed to `EXLNKID` for the merge)

```
adex <- tribble(
  ~USUBJID, ~EXLNKID, ~EXADJ,
  "1",      "1",      "AE",
  "1",      "2",      NA_character_,
  "1",      "3",      NA_character_,
  "2",      "1",      NA_character_,
  "3",      "1",      NA_character_
)

ec <- tribble(
  ~USUBJID, ~ECLNKID, ~ECADJ,
  "1",      "3",      "AE",
  "3",      "1",      NA_character_
)
```

```
fa <- tribble(
  ~USUBJID, ~FALNKID, ~FATESTCD, ~FAOBJ,             ~FASTRESC,
  "3",      "1",      "OCCUR",   "DOSE ADJUSTMENT", "Y"
)

derive_var_merged_ef_msrc(
  adex,
  by_vars = exprs(USUBJID, EXLNKID),
  flag_events = list(
    flag_event(
      dataset_name = "ex",
      condition = !is.na(EXADJ)
    ),
    flag_event(
      dataset_name = "ec",
      condition = !is.na(ECADJ),
      by_vars = exprs(USUBJID, EXLNKID = ECLNKID)
    ),
    flag_event(
      dataset_name = "fa",
    condition = FATESTCD == "OCCUR" & FAOBJ == "DOSE ADJUSTMENT" & FASTRESC == "Y",
      by_vars = exprs(USUBJID, EXLNKID = FALNKID)
    )
  ),
  source_datasets = list(ex = adex, ec = ec, fa = fa),
  new_var = DOSADJFL
)
#> # A tibble: 5 × 4
#>   USUBJID EXLNKID EXADJ DOSADJFL
#>   <chr>   <chr>   <chr> <chr>
#> 1 1       1       AE    Y
#> 2 1       2       <NA>  <NA>
#> 3 1       3       <NA>  Y
#> 4 2       1       <NA>  <NA>
#> 5 3       1       <NA>  Y
```

See Also

flag_event()

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

derive_var_merged_exist_flag
Merge an Existence Flag

Description

Adds a flag variable to the input dataset which indicates if there exists at least one observation in another dataset fulfilling a certain condition.

Note: This is a wrapper function for the more generic `derive_vars_merged()`.

Usage

```
derive_var_merged_exist_flag(
  dataset,
  dataset_add,
  by_vars,
  new_var,
  condition,
  true_value = "Y",
  false_value = NA_character_,
  missing_value = NA_character_,
  filter_add = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** none
`dataset_add`	Additional dataset The variables specified by the `by_vars` argument are expected. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** none
`by_vars`	Grouping variables **Permitted values** list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)` **Default value** none
`new_var`	New variable The specified variable is added to the input dataset. **Permitted values** an unquoted symbol, e.g., `AVAL` **Default value** none

condition
: Condition

  The condition is evaluated at the additional dataset (dataset_add). For all by groups where it evaluates as TRUE at least once the new variable is set to the true value (true_value). For all by groups where it evaluates as FALSE or NA for all observations the new variable is set to the false value (false_value). The new variable is set to the missing value (missing_value) for by groups not present in the additional dataset.

  Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

  Default value none

true_value
: True value

  Permitted values a character scalar, i.e., a character vector of length one

  Default value "Y"

false_value
: False value

  Permitted values a character scalar, i.e., a character vector of length one

  Default value NA_character_

missing_value
: Value used for missing information

  The new variable is set to the specified value for all by groups without observations in the additional dataset.

  Permitted values a character scalar, i.e., a character vector of length one

  Default value NA_character_

filter_add
: Filter for additional data

  Only observations fulfilling the specified condition are taken into account for flagging. If the argument is not specified, all observations are considered.

  Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

  Default value NULL

Details

1. The additional dataset is restricted to the observations matching the filter_add condition.
2. The new variable is added to the input dataset and set to the true value (true_value) if for the by group at least one observation exists in the (restricted) additional dataset where the condition evaluates to TRUE. It is set to the false value (false_value) if for the by group at least one observation exists and for all observations the condition evaluates to FALSE or NA. Otherwise, it is set to the missing value (missing_value).

Value

The output dataset contains all observations and variables of the input dataset and additionally the variable specified for new_var derived from the additional dataset (dataset_add).

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_obs_number(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sum derive_vars_transposed()

Examples

```
library(dplyr, warn.conflicts = FALSE)

dm <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",    "DM", "01-1028",   71, "YEARS",
  "PILOT01",    "DM", "04-1127",   84, "YEARS",
  "PILOT01",    "DM", "06-1049",   60, "YEARS"
)

ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,    ~AETERM,     ~AEREL,
  "PILOT01",    "AE", "01-1028", "ERYTHEMA", "POSSIBLE",
  "PILOT01",    "AE", "01-1028", "PRURITUS", "PROBABLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "POSSIBLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "PROBABLE"
)


derive_var_merged_exist_flag(
  dm,
  dataset_add = ae,
  by_vars = exprs(STUDYID, USUBJID),
  new_var = AERELFL,
  condition = AEREL == "PROBABLE"
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, AERELFL)

vs <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,      ~VISIT, ~VSTESTCD, ~VSSTRESN, ~VSBLFL,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "HEIGHT",     177.8,      NA,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "01-1028",  "BASELINE",  "WEIGHT",     99.34,     "Y",
  "PILOT01",    "VS", "01-1028",    "WEEK 4",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "04-1127", "SCREENING",  "HEIGHT",     165.1,      NA,
  "PILOT01",    "VS", "04-1127", "SCREENING",  "WEIGHT",     42.87,      NA,
  "PILOT01",    "VS", "04-1127",  "BASELINE",  "WEIGHT",     41.05,     "Y",
  "PILOT01",    "VS", "04-1127",    "WEEK 4",  "WEIGHT",     41.73,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "HEIGHT",    167.64,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "WEIGHT",     57.61,      NA,
  "PILOT01",    "VS", "06-1049",  "BASELINE",  "WEIGHT",     57.83,     "Y",
  "PILOT01",    "VS", "06-1049",    "WEEK 4",  "WEIGHT",     58.97,      NA
)
derive_var_merged_exist_flag(
  dm,
  dataset_add = vs,
  by_vars = exprs(STUDYID, USUBJID),
  filter_add = VSTESTCD == "WEIGHT" & VSBLFL == "Y",
  new_var = WTBLHIFL,
  condition = VSSTRESN > 90,
  false_value = "N",
  missing_value = "M"
```

```
) %>%
  select(STUDYID, USUBJID, AGE, AGEU, WTBLHIFL)
```

`derive_var_merged_summary` *Merge Summary Variables*

Description

[Deprecated] The `derive_var_merged_summary()` function has been deprecated in favor of `derive_vars_merged_summar`

Usage

```
derive_var_merged_summary(
  dataset,
  dataset_add,
  by_vars,
  new_vars = NULL,
  filter_add = NULL,
  missing_values = NULL
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` argument are expected to be in the dataset. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** none
`dataset_add`	Additional dataset The variables specified by the `by_vars` and the variables used on the left hand sides of the `new_vars` arguments are expected. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** none
`by_vars`	Grouping variables The expressions on the left hand sides of `new_vars` are evaluated by the specified *variables*. Then the resulting values are merged to the input dataset (`dataset`) by the specified *variables*. **Permitted values** list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)` **Default value** none
`new_vars`	New variables to add The specified variables are added to the input dataset. A named list of expressions is expected:

- LHS refer to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for by_vars. Any expression on the RHS must result in a single value per by group.

For example:

```
new_vars = exprs(
  DOSESUM = sum(AVAL),
  DOSEMEAN = mean(AVAL)
)
```

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value NULL

filter_add — Filter for additional dataset (dataset_add)

Only observations fulfilling the specified condition are taken into account for summarizing. If the argument is not specified, all observations are considered.

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

missing_values — Values for non-matching observations

For observations of the input dataset (dataset) which do not have a matching observation in the additional dataset (dataset_add) the values of the specified variables are set to the specified value. Only variables specified for new_vars can be specified for missing_values.

Permitted values list of named expressions created by exprs(), e.g., exprs(AVALC = VSSTRESC, AVAL = yn_to_numeric(AVALC))

Default value NULL

Details

1. The records from the additional dataset (dataset_add) are restricted to those matching the filter_add condition.
2. The new variables (new_vars) are created for each by group (by_vars) in the additional dataset (dataset_add) by calling summarize(). I.e., all observations of a by group are summarized to a single observation.
3. The new variables are merged to the input dataset. For observations without a matching observation in the additional dataset the new variables are set to NA. Observations in the additional dataset which have no matching observation in the input dataset are ignored.

Value

The output dataset contains all observations and variables of the input dataset and additionally the variables specified for new_vars.

See Also

derive_summary_records(), get_summary_records()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), dthcaus_source(), get_summary_records()

---

derive_var_nfrlt *Derive Nominal Relative Time from First Dose (NFRLT)*

Description

[Experimental]

Derives nominal/planned time from first dose in hours by combining visit day information with timepoint descriptions. The function converts timepoint strings to hours using `convert_xxtpt_to_hours()` and adds them to the day-based offset. Optionally creates a corresponding unit variable.

Usage

```
derive_var_nfrlt(
  dataset,
  new_var = NFRLT,
  new_var_unit = NULL,
  out_unit = "HOURS",
  tpt_var = NULL,
  visit_day,
  first_dose_day = 1,
  treatment_duration = 0,
  range_method = "midpoint",
  set_values_to_na = NULL
)
```

Arguments

`dataset`	Input dataset containing visit day variable and optionally timepoint variable. **Permitted values** A data frame or tibble **Default value** none
`new_var`	Name of the new variable to create (unquoted). Default is `NFRLT`. **Permitted values** Unquoted variable name **Default value** `NFRLT`
`new_var_unit`	Name of the unit variable to create (unquoted). If specified, a character variable will be created containing the unit of time exactly as provided in `out_unit`. Common CDISC variables are `FRLTU` (First Dose Relative Time Unit) or `RRLTU` (Reference Relative Time Unit). If not specified, no unit variable is created. **Permitted values** Unquoted variable name (optional) **Default value** `NULL`

`out_unit` Unit of time for the output variable. Options are:

- Days: "day", "days", "d"
- Hours: "hour", "hours", "hr", "hrs", "h" (default: "hours")
- Minutes: "minute", "minutes", "min", "mins"
- Weeks: "week", "weeks", "wk", "wks", "w"

Case-insensitive. The internal calculation is performed in hours, then converted to the specified unit. If `new_var_unit` is specified, it will contain the value exactly as provided by the user.

Permitted values Character scalar (see options above)

Default value `"HOURS"`

`tpt_var` Timepoint variable containing descriptions like "Pre-dose", "1H Post-dose", etc. (unquoted). If not provided or if the variable doesn't exist in the dataset, only the visit day offset is calculated (timepoint contribution is 0).

Permitted values Unquoted variable name (optional)

Default value `NULL`

`visit_day` Visit day variable (unquoted). This should be the planned/ nominal visit day (e.g., `VISITDY`). Records with `NA` in this variable will have NFRLT set to `NA`.

Permitted values Unquoted variable name

Default value none

`first_dose_day` The day number considered as the first dose day. Default is 1. For multiple-dose studies, this is typically Day 1.

Permitted values Numeric scalar (positive integer)

Default value `1`

`treatment_duration`

Duration of treatment in hours. Can be either:

- A numeric scalar (used for all records), or
- An unquoted variable name from the dataset (e.g., `EXDUR`) where each record can have a different treatment duration

Passed to `convert_xxtpt_to_hours()`. Must be non-negative. Default is 0 hours (for instantaneous treatments like oral medications).

Permitted values Numeric scalar or unquoted variable name (non-negative)

Default value `0`

`range_method` Method for converting time ranges to single values. Options are "midpoint" (default), "start", or "end". Passed to `convert_xxtpt_to_hours()`. For example, "0-6h" with midpoint returns 3, with start returns 0, with end returns 6.

Permitted values Character scalar ("midpoint", "start", or "end")

Default value `"midpoint"`

`set_values_to_na`

An optional condition that marks derived NFRLT values as `NA`. For example, `set_values_to_na = VISIT == "UNSCHEDULED"` will set NFRLT to `NA` for all unscheduled visits. Can use any variables in the dataset. When `new_var_unit` is specified, the unit variable will also be set to `NA` for these records.

Permitted values Condition (optional)

Default value `NULL`

Details

The nominal relative time is calculated as:

`NFRLT = (day_offset * 24 + timepoint_hours) * conversion_factor`

Where:

- `day_offset` is calculated from `visit_day` and `first_dose_day`, accounting for the absence of Day 0 in clinical trial convention
- `timepoint_hours` is derived from the timepoint description using `convert_xxtpt_to_hours()`, or 0 if `tpt_var` is not provided
- `conversion_factor` is:
  - 1 for "hours" (default)
  - 1/24 for "days"
  - 1/168 for "weeks" (1/24/7)
  - 60 for "minutes"

If `new_var_unit` is specified, a character variable is created containing the value of `out_unit` exactly as provided by the user. For example:

- `out_unit = "hours"` creates unit variable with value "hours"
- `out_unit = "HOURS"` creates unit variable with value "HOURS"
- `out_unit = "Days"` creates unit variable with value "Days"
- `NA` when the corresponding time value is `NA`

This matches the behavior of `derive_vars_duration()` and allows consistency when deriving multiple time variables.

Handling "No Day 0":

In clinical trials, day numbering typically follows the convention: ..., Day -2, Day -1, Day 1, Day 2, ... (no Day 0). This function accounts for this by adjusting the day offset when `visit_day` is negative and `first_dose_day` is positive.

For example, with `first_dose_day = 1` and different output units:

- Day -1, `out_unit = "hours"` -> -24 hours
- Day -1, `out_unit = "days"` -> -1 day
- Day -1, `out_unit = "weeks"` -> -0.1429 weeks
- Day -1, `out_unit = "minutes"` -> -1440 minutes
- Day -7 -> -168 hours, -7 days, -1 week, or -10080 minutes
- Day 1 -> 0 (in any unit, first dose day)
- Day 8 -> 168 hours, 7 days, 1 week, or 10080 minutes

With `first_dose_day = 7`:

- Day -1 -> -168 hours, -7 days, -1 week, or -10080 minutes
- Day 1 -> -144 hours, -6 days, -0.857 weeks, or -8640 minutes
- Day 6 -> -24 hours, -1 day, -0.143 weeks, or -1440 minutes

- Day 7 -> 0 (in any unit, first dose day)

Common Use Cases:

- **Single dose study**: Day 1 only, with samples at various timepoints (e.g., Pre-dose, 1H, 2H, 4H, 8H, 24H)
- **Multiple dose study**: Dosing on multiple days (e.g., Day 1, Day 8, Day 15) with samples around each dose
- **Screening visits**: Negative visit days (e.g., Day -14, Day -7) before first dose
- **Steady state study**: Multiple daily doses with sampling on specific days
- **Oral medications**: Use default `treatment_duration = 0` for instantaneous absorption
- **IV infusions**: Specify `treatment_duration` as infusion duration in hours (scalar) or as a variable name containing duration per record
- **Exposure records (EX)**: Can be called without `tpt_var` to derive NFRLT based only on visit day
- **Unscheduled visits**: Use `set_values_to_na` to set NFRLT to `NA` for unscheduled or early discontinuation visits
- **Variable treatment durations**: Use a variable name (e.g., `EXDUR`) when different subjects or visits have different treatment durations
- **Hours output**: Use `out_unit = "hours"` (default) for variables like `NFRLT` with `FRLTU`
- **Days output**: Use `out_unit = "days"` for variables like `NFRLTDY` with `FRLTU`
- **Weeks output**: Use `out_unit = "weeks"` for long-term studies with weekly dosing
- **Minutes output**: Use `out_unit = "minutes"` for very short-term PK studies or when minute precision is needed
- **CDISC compliance**: Use `new_var_unit = FRLTU` for first dose relative time or `new_var_unit = RRLTU` for reference relative time
- **Consistency with duration**: Use the same case for `out_unit` across `derive_vars_duration()` and `derive_var_nfrlt()` to ensure unit variables match

Important Notes:

- The function assumes `visit_day` represents the nominal/planned day, not the actual study day
- Day numbering follows clinical trial convention with no Day 0
- For timepoints that span multiple days (e.g., "24H Post-dose"), ensure `visit_day` is set to the day when the sample was taken. For example, if dosing occurs on Day 3, a "24H Post-dose" sample taken on Day 4 should have `visit_day = 4`.
- For crossover studies, consider deriving NFRLT separately per period
- `NA` values in `visit_day` will automatically result in `NA` for NFRLT (no need to use `set_values_to_na` for this case)
- `NA` values in `tpt_var` will result in `NA` for NFRLT
- `NA` values in the `treatment_duration` variable (if using a variable) will result in `NA` for NFRLT for those records

- Use `set_values_to_na` when you need to set NFRLT to `NA` based on other variables (e.g., `VISIT == "UNSCHEDULED"`), especially when `visit_day` is populated but should not be used for the NFRLT calculation
- If `tpt_var` is not provided or doesn't exist in the dataset, timepoint contribution is assumed to be 0 hours
- When using non-hour units, timepoint contributions are still calculated in hours first (e.g., "2H Post-dose" = 2 hours), then the entire result is converted to the specified unit
- The unit variable (if created) will contain the exact value provided in `out_unit`, preserving case and format

Setting Special Values:

If you need to set NFRLT to a specific value (e.g., 99999) for certain visits instead of `NA`, use `set_values_to_na` first to set them to `NA`, then use a subsequent `mutate()` call to replace those `NA` values:

```
dataset %>%
  derive_var_nfrlt(
    ...,
    set_values_to_na = VISIT == "UNSCHEDULED"
  ) %>%
  mutate(NFRLT = if_else(is.na(NFRLT) & VISIT == "UNSCHEDULED", 99999, NFRLT))
```

Value

The input dataset with the new nominal relative time variable added, and optionally the unit variable if `new_var_unit` is specified.

Examples

Single dose study:

Day 1 only with oral medication

```
library(dplyr)
library(tibble)

adpc <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    1,        "Pre-dose",
  "001",    1,        "1H Post-dose",
  "001",    1,        "2H Post-dose",
  "001",    1,        "4H Post-dose",
  "001",    1,        "24H Post-dose"
)

derive_var_nfrlt(
  adpc,
  new_var = NFRLT,
  tpt_var = PCTPT,
```

```
  visit_day = VISITDY
)
#> # A tibble: 5 × 4
#>   USUBJID VISITDY PCTPT         NFRLT
#>   <chr>     <dbl> <chr>         <dbl>
#> 1 001           1 Pre-dose          0
#> 2 001           1 1H Post-dose      1
#> 3 001           1 2H Post-dose      2
#> 4 001           1 4H Post-dose      4
#> 5 001           1 24H Post-dose    24
```

Single dose study with unit variable:

Creating NFRLT with FRLTU unit variable

```
derive_var_nfrlt(
  adpc,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY
)
#> # A tibble: 5 × 5
#>   USUBJID VISITDY PCTPT         NFRLT FRLTU
#>   <chr>     <dbl> <chr>         <dbl> <chr>
#> 1 001           1 Pre-dose          0 HOURS
#> 2 001           1 1H Post-dose      1 HOURS
#> 3 001           1 2H Post-dose      2 HOURS
#> 4 001           1 4H Post-dose      4 HOURS
#> 5 001           1 24H Post-dose    24 HOURS
```

Single dose study with different output units:

Deriving NFRLT in different time units with unit variables

```
adpc %>%
  derive_var_nfrlt(
    new_var = NFRLT,
    new_var_unit = FRLTU,
    out_unit = "HOURS",
    tpt_var = PCTPT,
    visit_day = VISITDY
  ) %>%
  derive_var_nfrlt(
    new_var = NFRLTDY,
    new_var_unit = FRLTDYU,
    out_unit = "days",
    tpt_var = PCTPT,
    visit_day = VISITDY
  )
#> # A tibble: 5 × 7
```

```
#>   USUBJID VISITDY PCTPT         NFRLT FRLTU NFRLTDY FRLTDYU
#>   <chr>     <dbl> <chr>         <dbl> <chr>   <dbl> <chr>
#> 1 001           1 Pre-dose          0 HOURS  0      days
#> 2 001           1 1H Post-dose      1 HOURS  0.0417 days
#> 3 001           1 2H Post-dose      2 HOURS  0.0833 days
#> 4 001           1 4H Post-dose      4 HOURS  0.167  days
#> 5 001           1 24H Post-dose    24 HOURS  1      days
```

Study with screening visits:

Handling negative visit days (no Day 0 in clinical trials)

```
adpc_screen <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    -14,      "Screening",
  "001",    -7,       "Pre-dose",
  "001",    -1,       "Pre-dose",
  "001",    1,        "Pre-dose",
  "001",    1,        "2H Post-dose"
)

derive_var_nfrlt(
  adpc_screen,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY
)
#> # A tibble: 5 × 5
#>   USUBJID VISITDY PCTPT        NFRLT FRLTU
#>   <chr>     <dbl> <chr>        <dbl> <chr>
#> 1 001         -14 Screening     -336 HOURS
#> 2 001          -7 Pre-dose      -168 HOURS
#> 3 001          -1 Pre-dose       -24 HOURS
#> 4 001           1 Pre-dose         0 HOURS
#> 5 001           1 2H Post-dose     2 HOURS
```

Multiple dose study:

Dosing on Days 1, 8, and 15

```
adpc_md <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    1,        "Pre-dose",
  "001",    1,        "2H Post-dose",
  "001",    8,        "Pre-dose",
  "001",    8,        "2H Post-dose",
  "001",    15,       "Pre-dose",
  "001",    15,       "2H Post-dose"
)
```

```
derive_var_nfrlt(
  adpc_md,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY
)
#> # A tibble: 6 × 5
#>   USUBJID VISITDY PCTPT        NFRLT FRLTU
#>   <chr>     <dbl> <chr>        <dbl> <chr>
#> 1 001           1 Pre-dose         0 HOURS
#> 2 001           1 2H Post-dose     2 HOURS
#> 3 001           8 Pre-dose       168 HOURS
#> 4 001           8 2H Post-dose   170 HOURS
#> 5 001          15 Pre-dose       336 HOURS
#> 6 001          15 2H Post-dose   338 HOURS
```

Multiple dose study with days output:

Deriving both NFRLT (hours) and NFRLTDY (days) with unit variables

```
adpc_md %>%
  derive_var_nfrlt(
    new_var = NFRLT,
    new_var_unit = FRLTU,
    tpt_var = PCTPT,
    visit_day = VISITDY
  ) %>%
  derive_var_nfrlt(
    new_var = NFRLTDY,
    new_var_unit = FRLTDYU,
    out_unit = "days",
    tpt_var = PCTPT,
    visit_day = VISITDY
  )
#> # A tibble: 6 × 7
#>   USUBJID VISITDY PCTPT        NFRLT FRLTU NFRLTDY FRLTDYU
#>   <chr>     <dbl> <chr>        <dbl> <chr>   <dbl> <chr>
#> 1 001           1 Pre-dose         0 HOURS  0      days
#> 2 001           1 2H Post-dose     2 HOURS  0.0833 days
#> 3 001           8 Pre-dose       168 HOURS  7      days
#> 4 001           8 2H Post-dose   170 HOURS  7.08   days
#> 5 001          15 Pre-dose       336 HOURS 14      days
#> 6 001          15 2H Post-dose   338 HOURS 14.1    days
```

Weekly dosing study:

Long-term study with weekly dosing, using weeks output

```
adpc_weekly <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
```

```
  "001",    1,        "Pre-dose",
  "001",    8,        "Pre-dose",
  "001",    15,       "Pre-dose",
  "001",    22,       "Pre-dose",
  "001",    29,       "Pre-dose"
)

derive_var_nfrlt(
  adpc_weekly,
  new_var = NFRLTWK,
  new_var_unit = FRLTU,
  out_unit = "weeks",
  tpt_var = PCTPT,
  visit_day = VISITDY
)
#> # A tibble: 5 × 5
#>   USUBJID VISITDY PCTPT    NFRLTWK FRLTU
#>   <chr>     <dbl> <chr>      <dbl> <chr>
#> 1 001           1 Pre-dose       0 weeks
#> 2 001           8 Pre-dose       1 weeks
#> 3 001          15 Pre-dose       2 weeks
#> 4 001          22 Pre-dose       3 weeks
#> 5 001          29 Pre-dose       4 weeks
```

Short-term PK study with minutes:

Very short timepoints requiring minute precision

```
adpc_short <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    1,        "Pre-dose",
  "001",    1,        "5 MIN POST",
  "001",    1,        "15 MIN POST",
  "001",    1,        "30 MIN POST",
  "001",    1,        "1H POST"
)

derive_var_nfrlt(
  adpc_short,
  new_var = NFRLTMIN,
  new_var_unit = FRLTU,
  out_unit = "minutes",
  tpt_var = PCTPT,
  visit_day = VISITDY
)
#> # A tibble: 5 × 5
#>   USUBJID VISITDY PCTPT       NFRLTMIN FRLTU
#>   <chr>     <dbl> <chr>          <dbl> <chr>
#> 1 001           1 Pre-dose           0 minutes
#> 2 001           1 5 MIN POST         5 minutes
```

```
#> 3 001           1 15 MIN POST       15 minutes
#> 4 001           1 30 MIN POST       30 minutes
#> 5 001           1 1H POST           60 minutes
```

Custom first dose day:

First dose on Day 7 instead of Day 1

```
adpc_day7 <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    -1,       "Pre-dose",
  "001",    1,        "Pre-dose",
  "001",    6,        "Pre-dose",
  "001",    7,        "Pre-dose",
  "001",    8,        "Pre-dose"
)

derive_var_nfrlt(
  adpc_day7,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY,
  first_dose_day = 7
)
#> # A tibble: 5 × 5
#>   USUBJID VISITDY PCTPT    NFRLT FRLTU
#>   <chr>     <dbl> <chr>    <dbl> <chr>
#> 1 001          -1 Pre-dose  -168 HOURS
#> 2 001           1 Pre-dose  -144 HOURS
#> 3 001           6 Pre-dose   -24 HOURS
#> 4 001           7 Pre-dose     0 HOURS
#> 5 001           8 Pre-dose    24 HOURS
```

IV infusion with scalar treatment duration:

2-hour infusion duration for all records

```
adpc_inf <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    1,        "Pre-dose",
  "001",    1,        "EOI",
  "001",    1,        "1H Post EOI",
  "001",    1,        "10MIN PRE EOI"
)

derive_var_nfrlt(
  adpc_inf,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
```

```
  visit_day = VISITDY,
  treatment_duration = 2
)
#> # A tibble: 4 × 5
#>   USUBJID VISITDY PCTPT         NFRLT FRLTU
#>   <chr>     <dbl> <chr>         <dbl> <chr>
#> 1 001           1 Pre-dose       0    HOURS
#> 2 001           1 EOI            2    HOURS
#> 3 001           1 1H Post EOI    3    HOURS
#> 4 001           1 10MIN PRE EOI  1.83 HOURS
```

Variable treatment duration:

Different treatment durations per subject using a variable

```
adpc_var_dur <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,           ~EXDUR,
  "001",    1,        "Pre-dose",       1,
  "001",    1,        "EOI",            1,
  "001",    1,        "1H POST EOI",    1,
  "002",    1,        "Pre-dose",       2,
  "002",    1,        "EOI",            2,
  "002",    1,        "1H POST EOI",    2
)

derive_var_nfrlt(
  adpc_var_dur,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY,
  treatment_duration = EXDUR
)
#> # A tibble: 6 × 6
#>   USUBJID VISITDY PCTPT       EXDUR NFRLT FRLTU
#>   <chr>     <dbl> <chr>       <dbl> <dbl> <chr>
#> 1 001           1 Pre-dose        1     0 HOURS
#> 2 001           1 EOI             1     1 HOURS
#> 3 001           1 1H POST EOI     1     2 HOURS
#> 4 002           1 Pre-dose        2     0 HOURS
#> 5 002           1 EOI             2     2 HOURS
#> 6 002           1 1H POST EOI     2     3 HOURS
```

Exposure records without timepoint variable:

Deriving NFRLT based only on visit day

```
ex <- tribble(
  ~USUBJID, ~VISITDY,
  "001",    1,
  "001",    8,
```

```
  "001",    15
)

derive_var_nfrlt(
  ex,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  visit_day = VISITDY
)
#> # A tibble: 3 × 4
#>   USUBJID VISITDY NFRLT FRLTU
#>   <chr>     <dbl> <dbl> <chr>
#> 1 001           1     0 HOURS
#> 2 001           8   168 HOURS
#> 3 001          15   336 HOURS
```

Exposure records with different output units:

Deriving NFRLT in hours, days, and weeks for exposure records

```
ex %>%
  derive_var_nfrlt(
    new_var = NFRLT,
    new_var_unit = FRLTU,
    visit_day = VISITDY
  ) %>%
  derive_var_nfrlt(
    new_var = NFRLTDY,
    new_var_unit = FRLTDYU,
    out_unit = "days",
    visit_day = VISITDY
  ) %>%
  derive_var_nfrlt(
    new_var = NFRLTWK,
    new_var_unit = FRLTWKU,
    out_unit = "weeks",
    visit_day = VISITDY
  )
#> # A tibble: 3 × 8
#>   USUBJID VISITDY NFRLT FRLTU NFRLTDY FRLTDYU NFRLTWK FRLTWKU
#>   <chr>     <dbl> <dbl> <chr>   <dbl> <chr>     <dbl> <chr>
#> 1 001           1     0 HOURS       0 days          0 weeks
#> 2 001           8   168 HOURS       7 days          1 weeks
#> 3 001          15   336 HOURS      14 days          2 weeks
```

Unscheduled visits:

Setting NFRLT to NA for unscheduled visits

```
adpc_unsched <- tribble(
  ~USUBJID, ~VISITDY, ~VISIT,        ~PCTPT,
```

```
  "001",    1,        "VISIT 1",     "Pre-dose",
  "001",    1,        "VISIT 1",     "2H Post-dose",
  "001",    NA_real_, "UNSCHEDULED", "Pre-dose",
  "001",    NA_real_, "UNSCHEDULED", "2H Post-dose"
)

derive_var_nfrlt(
  adpc_unsched,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY,
  set_values_to_na = VISIT == "UNSCHEDULED"
)
#> # A tibble: 4 × 6
#>   USUBJID VISITDY VISIT       PCTPT        NFRLT FRLTU
#>   <chr>     <dbl> <chr>       <chr>        <dbl> <chr>
#> 1 001           1 VISIT 1     Pre-dose         0 HOURS
#> 2 001           1 VISIT 1     2H Post-dose     2 HOURS
#> 3 001          NA UNSCHEDULED Pre-dose        NA <NA>
#> 4 001          NA UNSCHEDULED 2H Post-dose    NA <NA>
```

Early discontinuation visits:

Handling study drug early discontinuation

```
adpc_disc <- tribble(
  ~USUBJID, ~VISITDY, ~VISIT,                              ~PCTPT,
  "001",    1,        "VISIT 1",                           "Pre-dose",
  "001",    1,        "VISIT 1",                           "2H Post-dose",
  "001",    NA_real_, "STUDY DRUG EARLY DISCONTINUATION",  "Pre-dose"
)

derive_var_nfrlt(
  adpc_disc,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY,
  set_values_to_na = VISIT == "STUDY DRUG EARLY DISCONTINUATION"
)
#> # A tibble: 3 × 6
#>   USUBJID VISITDY VISIT                           PCTPT        NFRLT FRLTU
#>   <chr>     <dbl> <chr>                           <chr>        <dbl> <chr>
#> 1 001           1 VISIT 1                         Pre-dose         0 HOURS
#> 2 001           1 VISIT 1                         2H Post-dose     2 HOURS
#> 3 001          NA STUDY DRUG EARLY DISCONTINUATION Pre-dose        NA <NA>
```

Multiple exclusion criteria:

Excluding multiple visit types

```
adpc_multi <- tribble(
  ~USUBJID, ~VISITDY, ~VISIT,                              ~PCTPT,
  "001",    1,        "VISIT 1",                           "Pre-dose",
  "001",    NA_real_, "UNSCHEDULED",                       "Pre-dose",
  "001",    NA_real_, "STUDY DRUG EARLY DISCONTINUATION",  "Pre-dose"
)

derive_var_nfrlt(
  adpc_multi,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY,
  set_values_to_na = VISIT %in% c(
    "UNSCHEDULED",
    "STUDY DRUG EARLY DISCONTINUATION"
  )
)
#> # A tibble: 3 × 6
#>   USUBJID VISITDY VISIT                            PCTPT    NFRLT FRLTU
#>   <chr>     <dbl> <chr>                            <chr>    <dbl> <chr>
#> 1 001           1 VISIT 1                          Pre-dose     0 HOURS
#> 2 001          NA UNSCHEDULED                      Pre-dose    NA <NA>
#> 3 001          NA STUDY DRUG EARLY DISCONTINUATION Pre-dose    NA <NA>
```

Setting special values instead of NA:

Using mutate to set NFRLT to 99999 for unscheduled visits

```
adpc_unsched_value <- tribble(
  ~USUBJID, ~VISITDY, ~VISIT,        ~PCTPT,
  "001",    1,        "VISIT 1",     "Pre-dose",
  "001",    1,        "VISIT 1",     "2H Post-dose",
  "001",    NA_real_, "UNSCHEDULED", "Pre-dose",
  "001",    NA_real_, "UNSCHEDULED", "2H Post-dose"
)

adpc_unsched_value %>%
  derive_var_nfrlt(
    new_var = NFRLT,
    new_var_unit = FRLTU,
    tpt_var = PCTPT,
    visit_day = VISITDY,
    set_values_to_na = VISIT == "UNSCHEDULED"
  ) %>%
  mutate(
    NFRLT = if_else(is.na(NFRLT) & VISIT == "UNSCHEDULED", 99999, NFRLT),
   FRLTU = if_else(is.na(FRLTU) & VISIT == "UNSCHEDULED", NA_character_, FRLTU)
  )
#> # A tibble: 4 × 6
```

```
#>   USUBJID VISITDY VISIT       PCTPT        NFRLT FRLTU
#>   <chr>     <dbl> <chr>       <chr>        <dbl> <chr>
#> 1 001           1 VISIT 1     Pre-dose         0 HOURS
#> 2 001           1 VISIT 1     2H Post-dose     2 HOURS
#> 3 001          NA UNSCHEDULED Pre-dose     99999 <NA>
#> 4 001          NA UNSCHEDULED 2H Post-dose 99999 <NA>
```

Custom range method:

Using end of range instead of midpoint

```
adpc_range <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    1,        "Pre-dose",
  "001",    1,        "0-6h Post-dose"
)

derive_var_nfrlt(
  adpc_range,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY,
  range_method = "end"
)
#> # A tibble: 2 × 5
#>   USUBJID VISITDY PCTPT          NFRLT FRLTU
#>   <chr>     <dbl> <chr>          <dbl> <chr>
#> 1 001           1 Pre-dose           0 HOURS
#> 2 001           1 0-6h Post-dose     6 HOURS
```

Alternative terminology:

Using "Before" and "After" terminology

```
adpc_alt <- tribble(
  ~USUBJID, ~VISITDY, ~PCTPT,
  "001",    1,        "Before",
  "001",    1,        "1H After",
  "001",    1,        "2H After"
)

derive_var_nfrlt(
  adpc_alt,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY
)
#> # A tibble: 3 × 5
#>   USUBJID VISITDY PCTPT    NFRLT FRLTU
```

```
#>   <chr>     <dbl> <chr>    <dbl> <chr>
#> 1 001           1 Before       0 HOURS
#> 2 001           1 1H After     1 HOURS
#> 3 001           1 2H After     2 HOURS
```

Reference relative time with RRLTU:

Using RRLTU for reference relative time instead of first dose

```
derive_var_nfrlt(
  adpc,
  new_var = NRRLT,
  new_var_unit = RRLTU,
  tpt_var = PCTPT,
  visit_day = VISITDY,
  first_dose_day = 8
)
#> # A tibble: 5 × 5
#>   USUBJID VISITDY PCTPT         NRRLT RRLTU
#>   <chr>     <dbl> <chr>         <dbl> <chr>
#> 1 001           1 Pre-dose       -168 HOURS
#> 2 001           1 1H Post-dose   -167 HOURS
#> 3 001           1 2H Post-dose   -166 HOURS
#> 4 001           1 4H Post-dose   -164 HOURS
#> 5 001           1 24H Post-dose  -144 HOURS
```

Case sensitivity in out_unit:

Unit variable preserves the case provided in out_unit

```
derive_var_nfrlt(
  adpc,
  new_var = NFRLT,
  new_var_unit = FRLTU,
  out_unit = "HOURS",
  tpt_var = PCTPT,
  visit_day = VISITDY
)
#> # A tibble: 5 × 5
#>   USUBJID VISITDY PCTPT         NFRLT FRLTU
#>   <chr>     <dbl> <chr>         <dbl> <chr>
#> 1 001           1 Pre-dose          0 HOURS
#> 2 001           1 1H Post-dose      1 HOURS
#> 3 001           1 2H Post-dose      2 HOURS
#> 4 001           1 4H Post-dose      4 HOURS
#> 5 001           1 24H Post-dose    24 HOURS
```

See Also

convert_xxtpt_to_hours(), derive_vars_duration()

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_ontrtfl(), derive_var_pchg(), derive_var_shift(), derive_vars_crit_flag()

---

derive_var_obs_number *Adds a Variable Numbering the Observations Within Each By Group*

Description

Adds a variable numbering the observations within each by group

Usage

```
derive_var_obs_number(
  dataset,
  by_vars = NULL,
  order = NULL,
  new_var = ASEQ,
  check_type = "none"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset. **Default value** none
by_vars	Grouping variables **Default value** NULL
order	Sort order Within each by group the observations are ordered by the specified order. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic"). **Permitted values** list of variables or functions of variables **Default value** NULL
new_var	Name of variable to create The new variable is set to the observation number for each by group. The numbering starts with 1. **Permitted values** an unquoted symbol, e.g., AVAL **Default value** ASEQ
check_type	Check uniqueness? If "message", "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order. **Permitted values** "none", "message", "warning", "error" **Default value** "none"

Details

For each group (with respect to the variables specified for the by_vars parameter) the first or last observation (with respect to the order specified for the order parameter and the mode specified for the mode parameter) is included in the output dataset.

Value

A dataset containing all observations and variables of the input dataset and additionally the variable specified by the new_var parameter.

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_relative_flag(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

Examples

```
library(dplyr, warn.conflicts = FALSE)
vs <- tribble(
  ~STUDYID,  ~DOMAIN,      ~USUBJID, ~VSTESTCD, ~VISITNUM, ~VSTPTNUM,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         3,       815,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         3,       816,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         4,       815,
  "PILOT01",    "VS", "01-703-1182",   "DIABP",         4,       816,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         3,       815,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         3,       816,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         4,       815,
  "PILOT01",    "VS", "01-703-1182",   "PULSE",         4,       816,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         3,       815,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         3,       816,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         4,       815,
  "PILOT01",    "VS", "01-703-1182",   "SYSBP",         4,       816,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         3,       815,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         3,       816,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         4,       815,
  "PILOT01",    "VS", "01-716-1229",   "DIABP",         4,       816,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         3,       815,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         3,       816,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         4,       815,
  "PILOT01",    "VS", "01-716-1229",   "PULSE",         4,       816,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         3,       815,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         3,       816,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         4,       815,
  "PILOT01",    "VS", "01-716-1229",   "SYSBP",         4,       816
)
vs %>%
  derive_var_obs_number(
    by_vars = exprs(USUBJID, VSTESTCD),
```

```
    order = exprs(VISITNUM, desc(VSTPTNUM))
  )
```

`derive_var_ontrtfl` *Derive On-Treatment Flag Variable*

Description

Derive on-treatment flag (`ONTRTFL`) in an ADaM dataset with a single assessment date (e.g `ADT`) or event start and end dates (e.g. `ASTDT`/`AENDT`).

Usage

```
derive_var_ontrtfl(
  dataset,
  new_var = ONTRTFL,
  start_date,
  end_date = NULL,
  ref_start_date,
  ref_end_date = NULL,
  ref_end_window = 0,
  ignore_time_for_ref_end_date = TRUE,
  filter_pre_timepoint = NULL,
  span_period = FALSE
)
```

Arguments

`dataset`	Input dataset Required columns are `start_date`, `end_date`, `ref_start_date` and `ref_end_date`. **Default value** none
`new_var`	On-treatment flag variable name to be created. **Default value** `ONTRTFL`
`start_date`	The start date (e.g. `AESDT`) or assessment date (e.g. `ADT`) Required; A date or date-time object column is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object. **Default value** none
`end_date`	The end date of assessment/event (e.g. `AENDT`) A date or date-time object column is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object. Optional; Default is null. If the used and date value is missing on an observation, it is assumed the medication is ongoing and `ONTRTFL` is set to `"Y"`.

Default value NULL

ref_start_date The lower bound of the on-treatment period Required; A date or date-time object column is expected.

Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

Default value none

ref_end_date The upper bound of the on-treatment period A date or date-time object column is expected.

Refer to derive_vars_dt() to impute and derive a date from a date character vector to a date object.

If set to NULL, everything after ref_start_date will be considered on-treatment.

Default value NULL

ref_end_window A window to add to the upper bound ref_end_date measured in days (e.g. 7 if 7 days should be added to the upper bound)

Default value 0

ignore_time_for_ref_end_date

If the argument is set to TRUE, the time part is ignored for checking if the event occurred more than ref_end_window days after reference end date.

Permitted values TRUE, FALSE

Default value TRUE

filter_pre_timepoint

An expression to filter observations as not on-treatment when date = ref_start_date. For example, if observations where VSTPT = PRE should not be considered on-treatment when date = ref_start_date, filter_pre_timepoint should be used to denote when the on-treatment flag should be set to null. Optional; default is NULL.

Default value NULL

span_period A logical scalar. If TRUE, events that started prior to the ref_start_dateand are ongoing or end after the ref_start_date are flagged as "Y". Optional; default is FALSE.

Default value FALSE

Details

On-Treatment is calculated by determining whether the assessment date or start/stop dates fall between 2 dates. The following logic is used to assign on-treatment = "Y":

1. start_date is missing and ref_start_dateis non-missing
2. No timepoint filter is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date
3. A timepoint is provided (filter_pre_timepoint) and both start_date and ref_start_date are non-missing and start_date = ref_start_date and the filter provided in filter_pre_timepoint is not true.

4. ref_end_date is not provided and ref_start_date < start_date
5. ref_end_date is provided and ref_start_date < start_date <= ref_end_date + ref_end_window.

If the end_date is provided and the end_date < ref_start_date then the ONTRTFL is set to NULL.This would be applicable to cases where the start_date is missing and ONTRTFL has been assigned as "Y" above.

If the span_period is TRUE, this allows the user to assign ONTRTFL as "Y" to cases where the record started prior to the ref_start_date and was ongoing or ended after the ref_start_date.

Any date imputations needed should be done prior to calling this function.

Value

The input dataset with an additional column named ONTRTFL with a value of "Y" or NA

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_nfrlt(), derive_var_pchg(), derive_var_shift(), derive_vars_crit_flag()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate, warn.conflicts = FALSE)

advs <- tribble(
  ~USUBJID, ~ADT,              ~TRTSDT,           ~TRTEDT,
  "P01",    ymd("2020-02-24"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P02",    ymd("2020-01-01"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P03",    ymd("2019-12-31"), ymd("2020-01-01"), ymd("2020-03-01")
)
derive_var_ontrtfl(
  advs,
  start_date = ADT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT
)

advs <- tribble(
  ~USUBJID, ~ADT,              ~TRTSDT,           ~TRTEDT,
  "P01",    ymd("2020-07-01"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P02",    ymd("2020-04-30"), ymd("2020-01-01"), ymd("2020-03-01"),
  "P03",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01")
)
derive_var_ontrtfl(
  advs,
  start_date = ADT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT,
  ref_end_window = 60
```

```
)

advs <- tribble(
  ~USUBJID, ~ADTM,                      ~TRTSDTM,                   ~TRTEDTM,
  "P01",   ymd_hm("2020-01-02T12:00"), ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
  "P02",   ymd("2020-01-01"),       ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
  "P03",   ymd("2019-12-31"),       ymd_hm("2020-01-01T12:00"), ymd_hm("2020-03-01T12:00"),
) %>%
  mutate(TPT = c(NA, "PRE", NA))
derive_var_ontrtfl(
  advs,
  start_date = ADTM,
  ref_start_date = TRTSDTM,
  ref_end_date = TRTEDTM,
  filter_pre_timepoint = TPT == "PRE"
)

advs <- tribble(
  ~USUBJID, ~ASTDT,            ~TRTSDT,           ~TRTEDT,           ~AENDT,
  "P01",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-12-01"),
  "P02",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-03-15"),
  "P03",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), NA,
)
derive_var_ontrtfl(
  advs,
  start_date = ASTDT,
  end_date = AENDT,
  ref_start_date = TRTSDT,
  ref_end_date = TRTEDT,
  ref_end_window = 60,
  span_period = TRUE
)

advs <- tribble(
  ~USUBJID, ~ASTDT,            ~AP01SDT,          ~AP01EDT,          ~AENDT,
  "P01",    ymd("2020-03-15"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-12-01"),
  "P02",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), ymd("2020-03-15"),
  "P03",    ymd("2019-04-30"), ymd("2020-01-01"), ymd("2020-03-01"), NA,
)
derive_var_ontrtfl(
  advs,
  new_var = ONTR01FL,
  start_date = ASTDT,
  end_date = AENDT,
  ref_start_date = AP01SDT,
  ref_end_date = AP01EDT,
  span_period = TRUE
)
```

---

derive_var_pchg *Derive Percent Change from Baseline*

Description

Derive percent change from baseline (PCHG) in a BDS dataset

Usage

```
derive_var_pchg(dataset)
```

Arguments

dataset	Input dataset AVAL and BASE are expected. **Default value** none

Details

Percent change from baseline is calculated by dividing change from baseline by the absolute value of the baseline value and multiplying the result by 100.

Value

The input dataset with an additional column named PCHG

See Also

derive_var_chg()

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_shift(), derive_vars_crit_flag()

Examples

```
library(tibble)

advs <- tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ABLFL, ~BASE,
  "P01",    "WEIGHT", 80,    "Y",    80,
  "P01",    "WEIGHT", 80.8,  NA,     80,
  "P01",    "WEIGHT", 81.4,  NA,     80,
  "P02",    "WEIGHT", 75.3,  "Y",    75.3,
  "P02",    "WEIGHT", 76,    NA,     75.3
)
derive_var_pchg(advs)
```

derive_var_relative_flag *Flag Observations Before or After a Condition is Fulfilled*

Description

Flag all observations before or after the observation where a specified condition is fulfilled for each by group. For example, the function could be called to flag for each subject all observations before the first disease progression or to flag all AEs after a specific AE.

Usage

```
derive_var_relative_flag(
  dataset,
  by_vars,
  order,
  new_var,
  condition,
  mode,
  selection,
  inclusive,
  flag_no_ref_groups = TRUE,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset. **Default value** none
by_vars	Grouping variables **Default value** none
order	Sort order Within each by group the observations are ordered by the specified order. For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic"). **Permitted values** list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) **Default value** none
new_var	New variable The variable is added to the input dataset and set to "Y" for all observations before or after the condition is fulfilled. For all other observations it is set to NA. **Default value** none

condition — Condition for Reference Observation

The specified condition determines the reference observation. In the output dataset all observations before or after (selection argument) the reference observation are flagged.

Default value none

mode — Selection mode (first or last)

If "first" is specified, for each by group the observations before or after (selection argument) the observation where the condition (condition argument) is fulfilled the *first* time is flagged in the output dataset. If "last" is specified, for each by group the observations before or after (selection argument) the observation where the condition (condition argument) is fulfilled the *last* time is flagged in the output dataset.

Permitted values "first", "last"

Default value none

selection — Flag observations before or after the reference observation?

Permitted values "before", "after"

Default value none

inclusive — Flag the reference observation?

Permitted values TRUE, FALSE

Default value none

flag_no_ref_groups

Should by groups without reference observation be flagged?

Permitted values TRUE, FALSE

Default value TRUE

check_type — Check uniqueness?

If "warning" or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values "none", "warning", "error"

Default value "warning"

Details

For each by group (by_vars argument) the observations before or after (selection argument) the observations where the condition (condition argument) is fulfilled the first or last time (order argument and mode argument) is flagged in the output dataset.

Value

The input dataset with the new variable (new_var) added

See Also

General Derivation Functions for all ADaMs that returns variable appended to dataset: derive_var_extreme_flag(), derive_var_joined_exist_flag(), derive_var_merged_ef_msrc(), derive_var_merged_exist_flag(), derive_var_obs_number(), derive_vars_cat(), derive_vars_computed(), derive_vars_joined(), derive_vars_joined_summary(), derive_vars_merged(), derive_vars_merged_lookup(), derive_vars_merged_sur derive_vars_transposed()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)

# Flag all AEs after the first COVID AE
adae <- tribble(
  ~USUBJID, ~ASTDY, ~ACOVFL, ~AESEQ,
  "1",           2, NA,           1,
  "1",           5, "Y",          2,
  "1",           5, NA,           3,
  "1",          17, NA,           4,
  "1",          27, "Y",          5,
  "1",          32, NA,           6,
  "2",           8, NA,           1,
  "2",          11, NA,           2,
)

derive_var_relative_flag(
  adae,
  by_vars = exprs(USUBJID),
  order = exprs(ASTDY, AESEQ),
  new_var = PSTCOVFL,
  condition = ACOVFL == "Y",
  mode = "first",
  selection = "after",
  inclusive = FALSE,
  flag_no_ref_groups = FALSE
)

response <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      0,        "PR",
  "1",      1,        "CR",
  "1",      2,        "CR",
  "1",      3,        "SD",
  "1",      4,        "NE",
  "2",      0,        "SD",
  "2",      1,        "PD",
  "2",      2,        "PD",
  "3",      0,        "SD",
  "4",      0,        "SD",
  "4",      1,        "PR",
  "4",      2,        "PD",
  "4",      3,        "SD",
```

```
  "4",      4,        "PR"
)

# Flag observations up to first PD for each patient
response %>%
  derive_var_relative_flag(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    new_var = ANL02FL,
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )

# Flag observations up to first PD excluding baseline (AVISITN = 0) for each patient
response %>%
  restrict_derivation(
    derivation = derive_var_relative_flag,
    args = params(
      by_vars = exprs(USUBJID),
      order = exprs(AVISITN),
      new_var = ANL02FL,
      condition = AVALC == "PD",
      mode = "first",
      selection = "before",
      inclusive = TRUE
    ),
    filter = AVISITN > 0
  ) %>%
  arrange(USUBJID, AVISITN)
```

`derive_var_shift` *Derive Shift*

Description

Derives a character shift variable containing concatenated shift in values based on user-defined pairing, e.g., shift from baseline to analysis value, shift from baseline grade to analysis grade, ...

Usage

```
derive_var_shift(
  dataset,
  new_var,
  from_var,
  to_var,
  missing_value = "NULL",
  sep_val = " to "
)
```

Arguments

dataset	Input dataset The variables specified by the from_var and to_var arguments are expected to be in the dataset. **Default value** none
new_var	Name of the character shift variable to create. **Default value** none
from_var	Variable containing value to shift from. **Default value** none
to_var	Variable containing value to shift to. **Default value** none
missing_value	Character string to replace missing values in from_var or to_var. **Default value** "NULL"
sep_val	Character string to concatenate values of from_var and to_var. **Default value** " to "

Details

new_var is derived by concatenating the values of from_var to values of to_var (e.g. "NORMAL to HIGH"). When from_var or to_var has missing value, the missing value is replaced by missing_value (e.g. "NORMAL to NULL").

Value

The input dataset with the character shift variable added

See Also

BDS-Findings Functions that returns variable appended to dataset: derive_var_analysis_ratio(), derive_var_anrind(), derive_var_atoxgr(), derive_var_atoxgr_dir(), derive_var_base(), derive_var_chg(), derive_var_nfrlt(), derive_var_ontrtfl(), derive_var_pchg(), derive_vars_crit_flag()

Examples

```
library(tibble)

data <- tribble(
  ~USUBJID, ~PARAMCD, ~AVAL, ~ABLFL, ~BNRIND,  ~ANRIND,
  "P01",    "ALB",       33, "Y",    "LOW",    "LOW",
  "P01",    "ALB",       38, NA,     "LOW",    "NORMAL",
  "P01",    "ALB",       NA, NA,     "LOW",    NA,
  "P02",    "ALB",       37, "Y",    "NORMAL", "NORMAL",
  "P02",    "ALB",       49, NA,     "NORMAL", "HIGH",
  "P02",    "SODIUM",   147, "Y",    "HIGH",   "HIGH"
)
```

```
data %>%
  convert_blanks_to_na() %>%
  derive_var_shift(
    new_var = SHIFT1,
    from_var = BNRIND,
    to_var = ANRIND
  )

# or only populate post-baseline records
data %>%
  convert_blanks_to_na() %>%
  restrict_derivation(
    derivation = derive_var_shift,
    args = params(
      new_var = SHIFT1,
      from_var = BNRIND,
      to_var = ANRIND
    ),
    filter = is.na(ABLFL)
  )
```

`derive_var_trtdurd` *Derive Total Treatment Duration (Days)*

Description

Derives total treatment duration (days) (`TRTDURD`).

Note: This is a wrapper function for the more generic `derive_vars_duration()`.

Usage

```
derive_var_trtdurd(dataset, start_date = TRTSDT, end_date = TRTEDT)
```

Arguments

`dataset`	Input dataset The variables specified by the `start_date` and `end_date` arguments are expected to be in the dataset. **Default value** none
`start_date`	The start date A date or date-time object is expected. Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object. **Default value** `TRTSDT`

end_date — The end date

A date or date-time object is expected.

Refer to `derive_vars_dt()` to impute and derive a date from a date character vector to a date object.

Default value `TRTEDT`

Details

The total treatment duration is derived as the number of days from start to end date plus one.

Value

The input dataset with `TRTDURD` added

See Also

`derive_vars_duration()`

Date/Time Derivation Functions that returns variable appended to dataset: `derive_vars_dt()`, `derive_vars_dtm()`, `derive_vars_dtm_to_dt()`, `derive_vars_dtm_to_tm()`, `derive_vars_duration()`, `derive_vars_dy()`

Examples

```
library(tibble)
library(lubridate)

data <- tribble(
  ~TRTSDT, ~TRTEDT,
  ymd("2020-01-01"), ymd("2020-02-24")
)

derive_var_trtdurd(data)
```

---

derive_var_trtemfl *Derive Treatment-emergent Flag*

Description

Derive treatment emergent analysis flag (e.g., `TRTEMFL`).

Usage

```
derive_var_trtemfl(
  dataset,
  new_var = TRTEMFL,
  start_date = ASTDTM,
  end_date = AENDTM,
  trt_start_date = TRTSDTM,
```

```
  trt_end_date = NULL,
  end_window = NULL,
  ignore_time_for_trt_end = TRUE,
  initial_intensity = NULL,
  intensity = NULL,
  group_var = NULL,
  subject_keys = get_admiral_option("subject_keys")
)
```

Arguments

`dataset` Input dataset

The variables specified by `start_date`, `end_date`, `trt_start_date`, `trt_end_date`, `initial_intensity`, and `intensity` are expected.

Permitted values a dataset, i.e., a `data.frame` or tibble

Default value none

`new_var` New variable

Permitted values an unquoted symbol, e.g., `AVAL`

Default value `TRTEMFL`

`start_date` Event start date

Permitted values a date or datetime variable

Default value `ASTDTM`

`end_date` Event end date

Permitted values a date or datetime variable

Default value `AENDTM`

`trt_start_date` Treatment start date

Permitted values a date or datetime variable

Default value `TRTSDTM`

`trt_end_date` Treatment end date

Permitted values a date or datetime variable

Default value `NULL`

`end_window` If the argument is specified (in 'days'), events starting more than the specified number of days after end of treatment, are not flagged.

Permitted values a positive integer, e.g. `2` or `5`

Default value `NULL`

`ignore_time_for_trt_end`

If the argument is set to `TRUE`, the time part is ignored for checking if the event occurred more than `end_window` days after end of treatment.

Permitted values `TRUE`, `FALSE`

Default value `TRUE`

initial_intensity

Initial severity/intensity or toxicity

initial_intensity is ignored when group_var is specified.

If this argument is specified and group_var is NULL, events which start before treatment start and end after treatment start (or are ongoing) and worsened (i.e., the intensity is greater than the initial intensity), are flagged.

The values of the specified variable must be comparable with the usual comparison operators. I.e., if the intensity is greater than the initial intensity initial_intensity < intensity must evaluate to TRUE.

Permitted values an unquoted symbol, e.g., AVAL

Default value NULL

intensity

Severity/intensity or toxicity

If the argument is specified, events which start before treatment start and end after treatment start (or are ongoing) and worsened (i.e., the intensity is greater than the initial intensity), are flagged.

The values of the specified variable must be comparable with the usual comparison operators. I.e., if the intensity is greater than the initial intensity initial_intensity < intensity must evaluate to TRUE.

Permitted values an unquoted symbol, e.g., AVAL

Default value NULL

group_var

Grouping variable

If the argument is specified, it assumes that AEs are recorded as one episode of AE with multiple lines using a grouping variable.

Events starting during treatment or before treatment and worsening afterward are flagged. Once an AE record in a group is flagged, all subsequent records in the treatment window are flagged regardless of severity.

Permitted values an unquoted symbol, e.g., AVAL

Default value NULL

subject_keys

Variables to uniquely identify a subject.

This argument is only used when group_var is specified.

Permitted values list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value get_admiral_option("subject_keys")

Details

For the derivation of the new variable the following cases are considered in this order. The first case which applies, defines the value of the variable.

- *not treated*: If trt_start_date is NA, it is set to NA_character_.
- *event before treatment*: If end_date is before trt_start_date (and end_date is not NA), it is set to NA_character_.
- *no event date*: If start_date is NA, it is set to "Y" as in such cases it is usually considered more conservative to assume the event was treatment-emergent.

- *event started during treatment*:
  - if end_window is not specified: if start_date is on or after trt_start_date, it is set to "Y",
  - if end_window is specified: if start_date is on or after trt_start_date and start_date is on or before trt_end_date + end_window days, it is set to "Y",
- *event started before treatment and (possibly) worsened on treatment*:
  - if initial_intensity, intensity is specified and group_var is not specified: if initial_intensity < intensity and start_date is before trt_start_date and end_date is on or after trt_start_date or end_date is NA, it is set to "Y";
  - if group_var is specified: if intensity at treatment start < intensity and start_date is after trt_start_date and end_date is on or after trt_start_date or end_date is NA, it is set to "Y";
- Otherwise it is set to NA_character_.

The behavior of derive_var_trtemfl() is aligned with the proposed treatment-emergent AE assignment in the following PHUSE White Paper. See the final example in the examples section below.

Value

The input dataset with the variable specified by new_var added

Examples

Basic treatment-emergent flag:

Derive TRTEMFL without considering treatment end and worsening

- For this basic example, all we are using are AE start/end dates and comparing those against treatment start date.
- If the AE started on or after treatment then we flag as treatment-emergent (e.g. records 5-7).
- If missing AE start date then we flag as treatment-emergent as worst case (e.g. records 8, 11 and 13), unless we know that the AE end date was before treatment so we can rule out this being treatment-emergent (e.g. record 12).
- Any not treated subject would not get their AEs flagged as treatment-emergent (e.g. records 14-16).

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(lubridate)

adae <- tribble(
  ~USUBJID, ~ASTDT,            ~AENDT,            ~AEITOXGR, ~AETOXGR,
  # before treatment
  "1",      ymd("2021-12-13"), ymd("2021-12-15"), "1",       "1",
  "1",      ymd("2021-12-14"), ymd("2021-12-14"), "1",       "3",
  # starting before treatment and ending during treatment
  "1",      ymd("2021-12-30"), ymd("2022-01-14"), "1",       "3",
  "1",      ymd("2021-12-31"), ymd("2022-01-01"), "1",       "1",
```

```
  # starting during treatment
  "1",      ymd("2022-01-01"), ymd("2022-01-02"), "3",       "4",
  # after treatment
  "1",      ymd("2022-05-10"), ymd("2022-05-10"), "2",       "2",
  "1",      ymd("2022-05-11"), ymd("2022-05-11"), "2",       "2",
  # missing dates
  "1",      NA,                NA,                "3",       "4",
  "1",      ymd("2021-12-30"), NA,                "3",       "4",
  "1",      ymd("2021-12-31"), NA,                "3",       "3",
  "1",      NA,                ymd("2022-01-04"), "3",       "4",
  "1",      NA,                ymd("2021-12-24"), "3",       "4",
  "1",      NA,                ymd("2022-06-04"), "3",       "4",
  # without treatment
  "2",      NA,                ymd("2021-12-03"), "1",       "2",
  "2",      ymd("2021-12-01"), ymd("2021-12-03"), "1",       "2",
  "2",      ymd("2021-12-06"), NA,                "1",       "2"
) %>%
  mutate(
    STUDYID = "AB42",
    TRTSDT = if_else(USUBJID == "1", ymd("2022-01-01"), NA),
    TRTEDT = if_else(USUBJID == "1", ymd("2022-04-30"), NA)
  )

derive_var_trtemfl(
  adae,
  start_date = ASTDT,
  end_date = AENDT,
  trt_start_date = TRTSDT
) %>% select(USUBJID, TRTSDT, ASTDT, AENDT, TRTEMFL)
#> # A tibble: 16 × 5
#>    USUBJID TRTSDT     ASTDT      AENDT      TRTEMFL
#>    <chr>   <date>     <date>     <date>     <chr>
#>  1 1       2022-01-01 2021-12-13 2021-12-15 <NA>
#>  2 1       2022-01-01 2021-12-14 2021-12-14 <NA>
#>  3 1       2022-01-01 2021-12-30 2022-01-14 <NA>
#>  4 1       2022-01-01 2021-12-31 2022-01-01 <NA>
#>  5 1       2022-01-01 2022-01-01 2022-01-02 Y
#>  6 1       2022-01-01 2022-05-10 2022-05-10 Y
#>  7 1       2022-01-01 2022-05-11 2022-05-11 Y
#>  8 1       2022-01-01 NA         NA         Y
#>  9 1       2022-01-01 2021-12-30 NA         <NA>
#> 10 1       2022-01-01 2021-12-31 NA         <NA>
#> 11 1       2022-01-01 NA         2022-01-04 Y
#> 12 1       2022-01-01 NA         2021-12-24 <NA>
#> 13 1       2022-01-01 NA         2022-06-04 Y
#> 14 2       NA         NA         2021-12-03 <NA>
#> 15 2       NA         2021-12-01 2021-12-03 <NA>
#> 16 2       NA         2021-12-06 NA         <NA>
```

Considering treatment end date (`trt_end_date` and `end_window`):

Derive `TRTEMFL` taking a treatment end window into account

- In addition to the treatment-emergent checks explained in the above example, we now supply a treatment end date, `trt_end_date = TRTEDT` and an end window, `end_window = 10`. With these, any AE which started on or before treatment end date + 10 days is considered as treatment-emergent. Otherwise, those starting after the treatment end window are no longer flagged as treatment-emergent (e.g. record 7).

```
derive_var_trtemfl(
  adae,
  start_date = ASTDT,
  end_date = AENDT,
  trt_start_date = TRTSDT,
  trt_end_date = TRTEDT,
  end_window = 10
) %>% select(USUBJID, TRTSDT, TRTEDT, ASTDT, AENDT, TRTEMFL)
#> # A tibble: 16 × 6
#>    USUBJID TRTSDT     TRTEDT     ASTDT      AENDT      TRTEMFL
#>    <chr>   <date>     <date>     <date>     <date>     <chr>
#>  1 1       2022-01-01 2022-04-30 2021-12-13 2021-12-15 <NA>
#>  2 1       2022-01-01 2022-04-30 2021-12-14 2021-12-14 <NA>
#>  3 1       2022-01-01 2022-04-30 2021-12-30 2022-01-14 <NA>
#>  4 1       2022-01-01 2022-04-30 2021-12-31 2022-01-01 <NA>
#>  5 1       2022-01-01 2022-04-30 2022-01-01 2022-01-02 Y
#>  6 1       2022-01-01 2022-04-30 2022-05-10 2022-05-10 Y
#>  7 1       2022-01-01 2022-04-30 2022-05-11 2022-05-11 <NA>
#>  8 1       2022-01-01 2022-04-30 NA         NA         Y
#>  9 1       2022-01-01 2022-04-30 2021-12-30 NA         <NA>
#> 10 1       2022-01-01 2022-04-30 2021-12-31 NA         <NA>
#> 11 1       2022-01-01 2022-04-30 NA         2022-01-04 Y
#> 12 1       2022-01-01 2022-04-30 NA         2021-12-24 <NA>
#> 13 1       2022-01-01 2022-04-30 NA         2022-06-04 Y
#> 14 2       NA         NA         NA         2021-12-03 <NA>
#> 15 2       NA         NA         2021-12-01 2021-12-03 <NA>
#> 16 2       NA         NA         2021-12-06 NA         <NA>
```

Considering treatment worsening (`initial_intensity` and `intensity`):

Derive a new variable named `TRTEM2FL` taking worsening after treatment start into account

- We also now start look at changes in intensity following treatment start using the `initial_intensity` and `intensity` arguments. This only impacts AEs starting before treatment, and ending on or after treatment (or with missing AE end date). We can additionally consider treatment-emergence for an AE that was ongoing at the start of treatment which may have worsened as a result of treatment, i.e. the most extreme intensity is greater than the initial intensity (e.g. records 3 and 9).

```
derive_var_trtemfl(
  adae,
  new_var = TRTEM2FL,
  start_date = ASTDT,
```

```
  end_date = AENDT,
  trt_start_date = TRTSDT,
  trt_end_date = TRTEDT,
  end_window = 10,
  initial_intensity = AEITOXGR,
  intensity = AETOXGR
) %>% select(USUBJID, TRTSDT, ASTDT, AENDT, AEITOXGR, AETOXGR, TRTEM2FL)
#> # A tibble: 16 × 7
#>    USUBJID TRTSDT     ASTDT      AENDT      AEITOXGR AETOXGR TRTEM2FL
#>    <chr>   <date>     <date>     <date>     <chr>    <chr>   <chr>
#>  1 1       2022-01-01 2021-12-13 2021-12-15 1        1       <NA>
#>  2 1       2022-01-01 2021-12-14 2021-12-14 1        3       <NA>
#>  3 1       2022-01-01 2021-12-30 2022-01-14 1        3       Y
#>  4 1       2022-01-01 2021-12-31 2022-01-01 1        1       <NA>
#>  5 1       2022-01-01 2022-01-01 2022-01-02 3        4       Y
#>  6 1       2022-01-01 2022-05-10 2022-05-10 2        2       Y
#>  7 1       2022-01-01 2022-05-11 2022-05-11 2        2       <NA>
#>  8 1       2022-01-01 NA         NA         3        4       Y
#>  9 1       2022-01-01 2021-12-30 NA         3        4       Y
#> 10 1       2022-01-01 2021-12-31 NA         3        3       <NA>
#> 11 1       2022-01-01 NA         2022-01-04 3        4       Y
#> 12 1       2022-01-01 NA         2021-12-24 3        4       <NA>
#> 13 1       2022-01-01 NA         2022-06-04 3        4       Y
#> 14 2       NA         NA         2021-12-03 1        2       <NA>
#> 15 2       NA         2021-12-01 2021-12-03 1        2       <NA>
#> 16 2       NA         2021-12-06 NA         1        2       <NA>
```

Worsening when the same AE is collected over multiple records (`intensity` and `group_var`):

Derive `TRTEMFL` taking worsening after treatment into account within a grouping variable

- Firstly, to understand which records correspond to the same AE, we need to supply a grouping variable (`group_var`). Then this example works in a similar way to the above one, but here we don't have an initial intensity so we have to use the intensity of the AE at the time of treatment start. If an ongoing AE increases intensity after treatment start (i.e. worsens), then from that point on the records are considered treatment-emergent, unless after the treatment end window (e.g. records 4, 6 and 7).

```
adae2 <- tribble(
  ~USUBJID, ~ASTDT,            ~AENDT,            ~AETOXGR, ~AEGRPID,
  # ongoing AE where intensity drops after treatment start
  "1",      ymd("2021-12-31"), ymd("2022-01-01"), "3",      "1",
  "1",      ymd("2022-01-02"), ymd("2022-01-11"), "2",      "1",
  # ongoing AE where intensity increases after treatment start
  "1",      ymd("2021-12-31"), ymd("2022-01-01"), "1",      "2",
  "1",      ymd("2022-01-02"), ymd("2022-01-11"), "2",      "2",
  # ongoing AE where intensity increases after treatment start and then drops
  "1",      ymd("2021-12-31"), ymd("2022-01-01"), "1",      "3",
  "1",      ymd("2022-01-02"), ymd("2022-01-11"), "2",      "3",
  "1",      ymd("2022-01-12"), ymd("2022-01-15"), "1",      "3"
```

```
) %>%
  mutate(
    STUDYID = "AB42",
    TRTSDT = if_else(USUBJID == "1", ymd("2022-01-01"), NA),
    TRTEDT = if_else(USUBJID == "1", ymd("2022-04-30"), NA)
  )

derive_var_trtemfl(
  adae2,
  start_date = ASTDT,
  end_date = AENDT,
  trt_start_date = TRTSDT,
  trt_end_date = TRTEDT,
  end_window = 10,
  intensity = AETOXGR,
  group_var = AEGRPID
) %>% select(USUBJID, TRTSDT, ASTDT, AENDT, AETOXGR, AEGRPID, TRTEMFL)
#> # A tibble: 7 × 7
#>   USUBJID TRTSDT     ASTDT      AENDT      AETOXGR AEGRPID TRTEMFL
#>   <chr>   <date>     <date>     <date>     <chr>   <chr>   <chr>
#> 1 1       2022-01-01 2021-12-31 2022-01-01 3       1       <NA>
#> 2 1       2022-01-01 2022-01-02 2022-01-11 2       1       <NA>
#> 3 1       2022-01-01 2021-12-31 2022-01-01 1       2       <NA>
#> 4 1       2022-01-01 2022-01-02 2022-01-11 2       2       Y
#> 5 1       2022-01-01 2021-12-31 2022-01-01 1       3       <NA>
#> 6 1       2022-01-01 2022-01-02 2022-01-11 2       3       Y
#> 7 1       2022-01-01 2022-01-12 2022-01-15 1       3       Y
```

Further Examples from PHUSE White Paper:

Here we present more cases (some new, some similar to the examples above) which are aligned one-to-one with the scenarios in the PHUSE White Paper

```
adae3 <- tribble(
  ~USUBJID, ~TRTSDTM, ~TRTEDTM, ~ASTDTM, ~AENDTM, ~AEITOXGR, ~AETOXGR,
  # Patient 1: Pre-treatment AE
  "1", "2021-01-01", "2021-12-31", "2020-12-20", "2020-12-21", "2", "2",
  # Patient 2: On-treatment AE
  "2", "2021-01-01", "2021-12-31", "2021-12-20", "2021-12-21", "2", "2",
  # Patient 3: Pre-treatment AE, then on-treatment AE at same intensity
  "3", "2021-01-01", "2021-12-31", "2020-12-20", "2020-12-21", "2", "2",
  "3", "2021-01-01", "2021-12-31", "2021-12-20", "2021-12-21", "2", "2",
  # Patient 4: Pre-treatment AE, then on-treatment AE at wors. intensity
  "4", "2021-01-01", "2021-12-31", "2020-12-20", "2020-12-21", "2", "2",
  "4", "2021-01-01", "2021-12-31", "2021-12-20", "2021-12-21", "2", "3",
  # Patient 5: Pre-treatment AE, then on-treatment AE at impr. intensity
  "5", "2021-01-01", "2021-12-31", "2020-12-20", "2020-12-21", "2", "2",
  "5", "2021-01-01", "2021-12-31", "2021-12-20", "2021-12-21", "2", "1",
 # Patient 6: AE starting pre-treatment, continuing on-treatment, then 2nd AE at same intensity
  "6", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "2", "2",
```

```
  "6", "2021-01-01", "2021-12-31", "2021-12-20", "2021-12-21", "2", "2",
 # Patient 7: AE starting pre-treatment, continuing on-treatment, then 2nd AE at wors. intensity
  "7", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "2", "2",
  "7", "2021-01-01", "2021-12-31", "2021-12-20", "2021-12-21", "2", "3",
 # Patient 8: AE starting pre-treatment, continuing on-treatment, then 2nd AE at impr. intensity
  "8", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "2", "2",
  "8", "2021-01-01", "2021-12-31", "2021-12-20", "2021-12-21", "2", "1",
 # Patient 9: AE starting pre-treatment, continuing on-treatment, and no change in intensity
  "9", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "2", "2",
 # Patient 10: AE starting pre-treatment, continuing on-treatment, and wors. intensity
  "10", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "2", "4",
 # Patient 11: AE starting pre-treatment, continuing on-treatment, and impr. intensity
  "11", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "2", "1",
  # Patient 12: AE starting pre-treatment, worsening, then improving
  "12", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "3", "2",
  # Patient 13: AE starting pre-treatment, improving, then worsening
  "13", "2021-01-01", "2021-12-31", "2020-12-23", "2021-01-21", "1", "2",
) %>%
  mutate(
    ASTDTM = ymd(ASTDTM),
    AENDTM = ymd(AENDTM),
    TRTSDTM = ymd(TRTSDTM),
    TRTEDTM = ymd(TRTEDTM),
  )

derive_var_trtemfl(
  adae3,
  new_var = TRTEMFL,
  trt_end_date = TRTEDTM,
  end_window = 0,
  initial_intensity = AEITOXGR,
  intensity = AETOXGR,
  subject_keys = exprs(USUBJID)
) %>%
  select(USUBJID, TRTSDTM, TRTEDTM, ASTDTM, AENDTM, AEITOXGR, AETOXGR, TRTEMFL)
#> # A tibble: 19 × 8
#>    USUBJID TRTSDTM    TRTEDTM    ASTDTM     AENDTM     AEITOXGR AETOXGR TRTEMFL
#>     <chr>   <date>     <date>     <date>     <date>     <chr>    <chr>   <chr>
#>  1 1       2021-01-01 2021-12-31 2020-12-20 2020-12-21 2        2       <NA>
#>  2 2       2021-01-01 2021-12-31 2021-12-20 2021-12-21 2        2       Y
#>  3 3       2021-01-01 2021-12-31 2020-12-20 2020-12-21 2        2       <NA>
#>  4 3       2021-01-01 2021-12-31 2021-12-20 2021-12-21 2        2       Y
#>  5 4       2021-01-01 2021-12-31 2020-12-20 2020-12-21 2        2       <NA>
#>  6 4       2021-01-01 2021-12-31 2021-12-20 2021-12-21 2        3       Y
#>  7 5       2021-01-01 2021-12-31 2020-12-20 2020-12-21 2        2       <NA>
#>  8 5       2021-01-01 2021-12-31 2021-12-20 2021-12-21 2        1       Y
#>  9 6       2021-01-01 2021-12-31 2020-12-23 2021-01-21 2        2       <NA>
#> 10 6       2021-01-01 2021-12-31 2021-12-20 2021-12-21 2        2       Y
```

```
#> 11 7       2021-01-01 2021-12-31 2020-12-23 2021-01-21 2        2       <NA>
#> 12 7       2021-01-01 2021-12-31 2021-12-20 2021-12-21 2        3       Y
#> 13 8       2021-01-01 2021-12-31 2020-12-23 2021-01-21 2        2       <NA>
#> 14 8       2021-01-01 2021-12-31 2021-12-20 2021-12-21 2        1       Y
#> 15 9       2021-01-01 2021-12-31 2020-12-23 2021-01-21 2        2       <NA>
#> 16 10      2021-01-01 2021-12-31 2020-12-23 2021-01-21 2        4       Y
#> 17 11      2021-01-01 2021-12-31 2020-12-23 2021-01-21 2        1       <NA>
#> 18 12      2021-01-01 2021-12-31 2020-12-23 2021-01-21 3        2       <NA>
#> 19 13      2021-01-01 2021-12-31 2020-12-23 2021-01-21 1        2       Y
```

See Also

OCCDS Functions: `derive_vars_atc()`, `derive_vars_query()`

---

desc *dplyr desc*

Description

See `dplyr::desc` for details.

---

dose_freq_lookup *Pre-Defined Dose Frequencies*

Description

These pre-defined dose frequencies are sourced from CDISC. The number of rows to generate using `create_single_dose_dataset()` arguments `start_date` and `end_date` is derived from `DOSE_COUNT`, `DOSE_WINDOW`, and `CONVERSION_FACTOR` with appropriate functions from `lubridate`.

Usage

```
dose_freq_lookup
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 86 rows and 5 columns.

Details

NCI_CODE and CDISC_VALUE are included from the CDISC source for traceability.

DOSE_COUNT represents the number of doses received in one single unit of DOSE_WINDOW. For example, for CDISC_VALUE=="10 DAYS PER MONTH", DOSE_WINDOW=="MONTH" and DOSE_COUNT==10. Similarly, for CDISC_VALUE=="EVERY 2 WEEKS", DOSE_WINDOW=="WEEK" and DOSE_COUNT==0.5 (to yield one dose every two weeks).

CONVERSION_FACTOR is used to convert DOSE_WINDOW units "WEEK", "MONTH", and "YEAR" to the unit "DAY".

For example, for CDISC_VALUE=="10 DAYS PER MONTH", CONVERSION_FACTOR is 0.0329. One day of a month is assumed to be 1 / 30.4375 of a month (one day is assumed to be 1/365.25 of a year). Given only start_date and end_date in the aggregate dataset, CONVERSION_FACTOR is used to calculate specific dates forstart_date and end_date in the resulting single dose dataset for the doses that occur. In such cases, doses are assumed to occur at evenly spaced increments over the interval.

To see the entire table in the console, run print(dose_freq_lookup).

See Also

create_single_dose_dataset()

Other metadata: atoxgr_criteria_ctcv4, atoxgr_criteria_ctcv4_uscv, atoxgr_criteria_ctcv5, atoxgr_criteria_ctcv5_uscv, atoxgr_criteria_ctcv6, atoxgr_criteria_ctcv6_uscv, atoxgr_criteria_daids, atoxgr_criteria_daids_uscv, country_code_lookup

---

dthcaus_source *Create a* dthcaus_source *Object*

Description

[Deprecated] The dthcaus_source() function and dthcaus_source() have been deprecated in favor of event().

Usage

```
dthcaus_source(
  dataset_name,
  filter,
  date,
  order = NULL,
  mode = "first",
  dthcaus,
  set_values_to = NULL
)
```

Arguments

dataset_name	The name of the dataset, i.e. a string, used to search for the death cause. **Default value** none
filter	An expression used for filtering dataset. **Default value** none
date	A date or datetime variable or an expression to be used for sorting dataset. **Default value** none
order	Sort order Additional variables/expressions to be used for sorting the dataset. The dataset is ordered by date and order. Can be used to avoid duplicate record warning. **Permitted values** list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) or NULL **Default value** NULL
mode	One of "first" or "last". Either the "first" or "last" observation is preserved from the dataset which is ordered by date. **Default value** "first"
dthcaus	A variable name, an expression, or a string literal If a variable name is specified, e.g., AEDECOD, it is the variable in the source dataset to be used to assign values to DTHCAUS; if an expression, e.g., str_to_upper(AEDECOD), it is evaluated in the source dataset and the results is assigned to DTHCAUS; if a string literal, e.g. "Adverse Event", it is the fixed value to be assigned to DTHCAUS. **Default value** none
set_values_to	Variables to be set to trace the source dataset **Default value** NULL

Value

An object of class "dthcaus_source".

See Also

derive_var_dthcaus()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), get_summary_records()

Examples

```
# Deaths sourced from AE
src_ae <- dthcaus_source(
  dataset_name = "ae",
  filter = AEOUT == "FATAL",
  date = AEDTHDT,
```

```
  mode = "first",
  dthcaus = AEDECOD
)

# Deaths sourced from DS
src_ds <- dthcaus_source(
  dataset_name = "ds",
  filter = DSDECOD == "DEATH",
  date = convert_dtc_to_dt(DSSTDTC),
  mode = "first",
  dthcaus = DSTERM
)
```

`event` *Create an* `event` *Object*

Description

The `event` object is used to define events as input for the `derive_extreme_event()` and `derive_vars_extreme_event()` functions.

Usage

```
event(
  dataset_name = NULL,
  condition = NULL,
  mode = NULL,
  order = NULL,
  set_values_to = NULL,
  keep_source_vars = NULL,
  description = NULL
)
```

Arguments

`dataset_name`	Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for `source_datasets` in `derive_extreme_event()`. If the argument is not specified, the input dataset (`dataset`) of `derive_extreme_event()` is used. **Permitted values** a character scalar **Default value** `NULL`
`condition`	An unquoted condition for selecting the observations, which will contribute to the extreme event. If the condition contains summary functions like `all()`, they are evaluated for each by group separately. **Permitted values** an unquoted condition **Default value** `NULL`

mode — If specified, the first or last observation with respect to order is selected for each by group.

Permitted values "first", "last", NULL

Default value NULL

order — The specified variables or expressions are used to select the first or last observation if mode is specified.

For handling of NAs in sorting variables see the "Sort Order" section in vignette("generic").

Permitted values list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) or NULL

Default value NULL

set_values_to — A named list returned by exprs() defining the variables to be set for the event, e.g. exprs(PARAMCD = "WSP", PARAM = "Worst Sleeping Problems"). The values can be a symbol, a character string, a numeric value, NA or an expression.

Permitted values a named list of expressions, e.g., created by exprs()

Default value NULL

keep_source_vars

Variables to keep from the source dataset

The specified variables are kept for the selected observations. The variables specified for by_vars (of derive_extreme_event()) and created by set_values_to are always kept.

Permitted values A list of expressions where each element is a symbol or a tidyselect expression, e.g., exprs(VISIT, VISITNUM, starts_with("RS")).

Default value NULL

description — Description of the event

The description does not affect the derivations where the event is used. It is intended for documentation only.

Permitted values a character scalar

Default value NULL

Value

An object of class event

See Also

derive_extreme_event(), derive_vars_extreme_event(), event_joined()

Source Objects: basket_select(), censor_source(), death_event, event_joined(), event_source(), flag_event(), query(), records_source(), tte_source()

event_joined *Create a* event_joined *Object*

Description

The event_joined object is used to define events as input for the derive_extreme_event() and derive_vars_extreme_event() functions. This object should be used if the event does not depend on a single observation of the source dataset but on multiple observations. For example, if the event needs to be confirmed by a second observation of the source dataset.

The events are selected by calling filter_joined(). See its documentation for more details.

Usage

```
event_joined(
  dataset_name = NULL,
  filter_source = NULL,
  condition,
  by_vars = NULL,
  order = NULL,
  tmp_obs_nr_var = NULL,
  join_vars,
  join_type,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  set_values_to = NULL,
  keep_source_vars = NULL,
  description = NULL
)
```

Arguments

dataset_name	Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for source_datasets in derive_extreme_event(). If the argument is not specified, the input dataset (dataset) of derive_extreme_event() is used. **Permitted values** a character scalar **Default value** NULL
filter_source	A condition to restrict the source dataset before joining **Permitted values** an unquoted condition, e.g., AVISIT == "BASELINE" **Default value** NULL
condition	An unquoted condition for selecting the observations, which will contribute to the extreme event. The condition is applied to the joined dataset for selecting the confirmed observations. The condition can include summary functions like all() or any(). The

joined dataset is grouped by the original observations. I.e., the summary function are applied to all observations up to the confirmation observation. For example in the oncology setting when using this function for confirmed best overall response, `condition = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) & count_vals(var = AVALC.join, val = "NE") <= 1` selects observations with response "CR" and for all observations up to the confirmation observation the response is "CR" or "NE" and there is at most one "NE".

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value none

`by_vars` By variables

The specified variables are used to join the dataset with itself. If the argument is not specified (or set to `NULL`), the by variables specified for `derive_extreme_event()` are used.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

`order` If specified, the specified variables or expressions are used to select the first observation.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

Permitted values list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))` or `NULL`

Default value `NULL`

`tmp_obs_nr_var` Temporary observation number

The specified variable is added to the source dataset (`dataset_name`). It is set to the observation number with respect to `order`. For each by group (`by_vars`) the observation number starts with `1`. If there is more than one record for specific values for `by_vars` and `order`, all records get the same observation number. The variable can be used in the conditions (`filter_join`, `first_cond_upper`, `first_cond_lower`). It is not included in the output dataset. It can also be used to select events depending on consecutive observations or the last observation.

Permitted values an unquoted symbol, e.g., `AVAL`

Default value `NULL`

`join_vars` Variables to keep from joined dataset

The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix ".join". For example to select all observations with `AVALC == "Y"` and `AVALC == "Y"` for at least one subsequent visit `join_vars = exprs(AVALC, AVISITN)` and `condition = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join` could be specified.

The `*.join` variables are not included in the output dataset.

Permitted values a named list of expressions, e.g., created by `exprs()`

Default value none

join_type
: Observations to keep after joining

  The argument determines which of the joined observations are kept with respect to the original observation. For example, if join_type = "after" is specified all observations after the original observations are kept.

  Permitted values "before", "after", "all"

  Default value none

first_cond_lower
: Condition for selecting range of data (before)

  If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

  This parameter should be specified if condition contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation.

  Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

  Default value NULL

first_cond_upper
: Condition for selecting range of data (after)

  If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

  This parameter should be specified if condition contains summary functions which should not apply to all observations but only up to the confirmation assessment.

  Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

  Default value NULL

set_values_to
: A named list returned by exprs() defining the variables to be set for the event, e.g. exprs(PARAMCD = "WSP", PARAM = "Worst Sleeping Problems"). The values can be a symbol, a character string, a numeric value, NA or an expression.

  Permitted values a named list of expressions, e.g., created by exprs()

  Default value NULL

keep_source_vars
: Variables to keep from the source dataset

  The specified variables are kept for the selected observations. The variables specified for by_vars (of derive_extreme_event()) and created by set_values_to are always kept.

  Permitted values A list of expressions where each element is a symbol or a tidyselect expression, e.g., exprs(VISIT, VISITNUM, starts_with("RS")).

  Default value NULL

description
: Description of the event

  The description does not affect the derivations where the event is used. It is intended for documentation only.

Permitted values a character scalar
Default value NULL

Value

An object of class event_joined

See Also

derive_extreme_event(), derive_vars_extreme_event(), event()

Source Objects: basket_select(), censor_source(), death_event, event(), event_source(), flag_event(), query(), records_source(), tte_source()

Examples

```
library(tibble)
library(dplyr)
library(lubridate)
# Derive confirmed best overall response (using event_joined())
# CR - complete response, PR - partial response, SD - stable disease
# NE - not evaluable, PD - progressive disease
adsl <- tribble(
  ~USUBJID, ~TRTSDTC,
  "1",      "2020-01-01",
  "2",      "2019-12-12",
  "3",      "2019-11-11",
  "4",      "2019-12-30",
  "5",      "2020-01-01",
  "6",      "2020-02-02",
  "7",      "2020-02-02",
  "8",      "2020-02-01"
) %>%
  mutate(TRTSDT = ymd(TRTSDTC))

adrs <- tribble(
  ~USUBJID, ~ADTC,        ~AVALC,
  "1",      "2020-01-01", "PR",
  "1",      "2020-02-01", "CR",
  "1",      "2020-02-16", "NE",
  "1",      "2020-03-01", "CR",
  "1",      "2020-04-01", "SD",
  "2",      "2020-01-01", "SD",
  "2",      "2020-02-01", "PR",
  "2",      "2020-03-01", "SD",
  "2",      "2020-03-13", "CR",
  "4",      "2020-01-01", "PR",
  "4",      "2020-03-01", "NE",
  "4",      "2020-04-01", "NE",
  "4",      "2020-05-01", "PR",
  "5",      "2020-01-01", "PR",
  "5",      "2020-01-10", "PR",
  "5",      "2020-01-20", "PR",
```

```
  "6",      "2020-02-06", "PR",
  "6",      "2020-02-16", "CR",
  "6",      "2020-03-30", "PR",
  "7",      "2020-02-06", "PR",
  "7",      "2020-02-16", "CR",
  "7",      "2020-04-01", "NE",
  "8",      "2020-02-16", "PD"
) %>%
  mutate(
    ADT = ymd(ADTC),
    PARAMCD = "OVR",
    PARAM = "Overall Response by Investigator"
  ) %>%
  derive_vars_merged(
    dataset_add = adsl,
    by_vars = exprs(USUBJID),
    new_vars = exprs(TRTSDT)
  )

derive_extreme_event(
  adrs,
  by_vars = exprs(USUBJID),
  order = exprs(ADT),
  mode = "first",
  source_datasets = list(adsl = adsl),
  events = list(
    event_joined(
      description = paste(
        "CR needs to be confirmed by a second CR at least 28 days later",
        "at most one NE is acceptable between the two assessments"
      ),
      join_vars = exprs(AVALC, ADT),
      join_type = "after",
      first_cond_upper = AVALC.join == "CR" &
        ADT.join >= ADT + 28,
      condition = AVALC == "CR" &
        all(AVALC.join %in% c("CR", "NE")) &
        count_vals(var = AVALC.join, val = "NE") <= 1,
      set_values_to = exprs(
        AVALC = "CR"
      )
    ),
    event_joined(
      description = paste(
        "PR needs to be confirmed by a second CR or PR at least 28 days later,",
        "at most one NE is acceptable between the two assessments"
      ),
      join_vars = exprs(AVALC, ADT),
      join_type = "after",
      first_cond_upper = AVALC.join %in% c("CR", "PR") &
        ADT.join >= ADT + 28,
      condition = AVALC == "PR" &
        all(AVALC.join %in% c("CR", "PR", "NE")) &
```

```
      count_vals(var = AVALC.join, val = "NE") <= 1,
    set_values_to = exprs(
      AVALC = "PR"
    )
  ),
  event(
    description = paste(
      "CR, PR, or SD are considered as SD if occurring at least 28",
      "after treatment start"
    ),
    condition = AVALC %in% c("CR", "PR", "SD") & ADT >= TRTSDT + 28,
    set_values_to = exprs(
      AVALC = "SD"
    )
  ),
  event(
    condition = AVALC == "PD",
    set_values_to = exprs(
      AVALC = "PD"
    )
  ),
  event(
    condition = AVALC %in% c("CR", "PR", "SD", "NE"),
    set_values_to = exprs(
      AVALC = "NE"
    )
  ),
  event(
    description = "set response to MISSING for patients without records in ADRS",
    dataset_name = "adsl",
    condition = TRUE,
    set_values_to = exprs(
      AVALC = "MISSING"
    ),
    keep_source_vars = exprs(TRTSDT)
  )
),
set_values_to = exprs(
  PARAMCD = "CBOR",
  PARAM = "Best Confirmed Overall Response by Investigator"
)
) %>%
  filter(PARAMCD == "CBOR")
```

`event_source` *Create an* `event_source` *Object*

Description

`event_source` objects are used to define events as input for the `derive_param_tte()` function.

Note: This is a wrapper function for the more generic `tte_source()`.

Usage

```
event_source(
  dataset_name,
  filter = NULL,
  date,
  set_values_to = NULL,
  order = NULL
)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets parameter of derive_param_tte(). **Default value** none
filter	An unquoted condition for selecting the observations from dataset which are events or possible censoring time points. **Default value** NULL
date	A variable or expression providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol or expression is expected. Refer to derive_vars_dt() or convert_dtc_to_dt() to impute and derive a date from a date character vector to a date object. **Default value** none
set_values_to	A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA. **Default value** NULL
order	Sort order An optional named list returned by exprs() defining additional variables that the source dataset is sorted on after date. **Permitted values** list of variables created by exprs() e.g. exprs(ASEQ). **Default value** order

Value

An object of class event_source, inheriting from class tte_source

See Also

derive_param_tte(), censor_source()

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), flag_event(), query(), records_source(), tte_source()

Examples

```
# Death event

event_source(
  dataset_name = "adsl",
  filter = DTHFL == "Y",
  date = DTHDT,
  set_values_to = exprs(
    EVNTDESC = "DEATH",
    SRCDOM = "ADSL",
    SRCVAR = "DTHDT"
  )
)
```

---

example_qs *Example* QS *Dataset*

Description

An example QS dataset based on the examples from the CDISC ADaM Supplements Generalized Anxiety Disorder 7-Item Version 2 (GAD-7) and Geriatric Depression Scale Short Form (GDS-SF).

Usage

```
example_qs
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 161 rows and 11 columns.

Source

Created by (https://github.com/pharmaverse/admiral/blob/main/data-raw/create_example_qs.R)

See Also

Other datasets: admiral_adlb, admiral_adsl, queries, queries_mh

---

exprs *rlang exprs*

Description

See `rlang::exprs` for details.

extract_unit *Extract Unit From Parameter Description*

Description

Extract the unit of a parameter from a description like "Param (unit)".

Usage

```
extract_unit(x)
```

Arguments

x	A parameter description **Default value** none

Value

A string

See Also

Utilities used within Derivation functions: `get_flagged_records()`, `get_not_mapped()`, `get_vars_query()`

Examples

```
extract_unit("Height (cm)")

extract_unit("Diastolic Blood Pressure (mmHg)")
```

filter_exist *Returns records that fit into existing by groups in a filtered source dataset*

Description

Returns all records in the input dataset that belong to by groups that are present in a source dataset, after the source dataset is optionally filtered. For example, this could be used to return ADSL records for subjects that experienced a certain adverse event during the course of the study (as per records in ADAE).

Usage

```
filter_exist(dataset, dataset_add, by_vars, filter_add = NULL)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. **Default value** none
dataset_add	Source dataset The source dataset, which determines the by groups returned in the input dataset, based on the groups that exist in this dataset after being subset by filter_add. The variables specified in the by_vars and filter_add parameters are expected in this dataset. **Default value** none
by_vars	Grouping variables **Default value** none
filter_add	Filter for the source dataset The filter condition which will be used to subset the source dataset. Alternatively, if no filter condition is supplied, no subsetting of the source dataset will be performed. **Default value** NULL

Details

Returns the records in dataset which match an existing by group in dataset_add, after being filtered according to filter_add. If there are no by groups that exist in both datasets, an empty dataset will be returned.

Value

The records in the input dataset which are contained within an existing by group in the filtered source dataset.

See Also

Utilities for Filtering Observations: count_vals(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), max_cond(), min_cond()

Examples

```
# Get demographic information about subjects who have suffered from moderate or
# severe fatigue

library(tibble)

adsl <- tribble(
  ~USUBJID,      ~AGE, ~SEX,
  "01-701-1015", 63,   "F",
  "01-701-1034", 77,   "F",
```

```
  "01-701-1115", 84,   "M",
  "01-701-1146", 75,   "F",
  "01-701-1444", 63,   "M"
)

adae <- tribble(
  ~USUBJID,      ~AEDECOD,                    ~AESEV,     ~AESTDTC,
  "01-701-1015", "DIARRHOEA",                 "MODERATE", "2014-01-09",
  "01-701-1034", "FATIGUE",                   "SEVERE",   "2014-11-02",
  "01-701-1034", "APPLICATION SITE PRURITUS", "MODERATE", "2014-08-27",
  "01-701-1115", "FATIGUE",                   "MILD",     "2013-01-14",
  "01-701-1146", "FATIGUE",                   "MODERATE", "2013-06-03"
)

filter_exist(
  dataset = adsl,
  dataset_add = adae,
  by_vars = exprs(USUBJID),
  filter_add = AEDECOD == "FATIGUE" & AESEV %in% c("MODERATE", "SEVERE")
)
```

`filter_extreme` *Filter the First or Last Observation for Each By Group*

Description

Filters the first or last observation for each by group.

Usage

```
filter_extreme(dataset, by_vars = NULL, order, mode, check_type = "warning")
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `order` arguments are expected to be in the dataset. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** none
`by_vars`	Grouping variables **Permitted values** list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)` **Default value** `NULL`
`order`	Sort order Within each by group the observations are ordered by the specified order.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value none

`mode` Selection mode (first or last)

If `"first"` is specified, the first observation of each by group is included in the output dataset. If `"last"` is specified, the last observation of each by group is included in the output dataset.

Permitted values `"first"`, `"last"`

Default value none

`check_type` Check uniqueness?

If `"warning"` or `"error"` is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values `"none"`, `"message"`, `"warning"`, `"error"`

Default value `"warning"`

Details

For each group (with respect to the variables specified for the `by_vars` parameter) the first or last observation (with respect to the order specified for the `order` parameter and the mode specified for the `mode` parameter) is included in the output dataset.

Value

A dataset containing the first or last observation of each by group

See Also

Utilities for Filtering Observations: `count_vals()`, `filter_exist()`, `filter_joined()`, `filter_not_exist()`, `filter_relative()`, `max_cond()`, `min_cond()`

Examples

```
library(dplyr, warn.conflicts = FALSE)

ex <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~EXSEQ, ~EXDOSE,    ~EXTRT,
  "PILOT01",    "EX", "01-1442",      1,      54,    "XANO",
  "PILOT01",    "EX", "01-1442",      2,      54,    "XANO",
  "PILOT01",    "EX", "01-1442",      3,      54,    "XANO",
  "PILOT01",    "EX", "01-1444",      1,      54,    "XANO",
  "PILOT01",    "EX", "01-1444",      2,      81,    "XANO",
  "PILOT01",    "EX", "05-1382",      1,      54,    "XANO",
  "PILOT01",    "EX", "08-1213",      1,      54,    "XANO",
  "PILOT01",    "EX", "10-1053",      1,      54,    "XANO",
  "PILOT01",    "EX", "10-1053",      2,      54,    "XANO",
  "PILOT01",    "EX", "10-1183",      1,       0, "PLACEBO",
  "PILOT01",    "EX", "10-1183",      2,       0, "PLACEBO",
```

```
  "PILOT01",    "EX", "10-1183",      3,       0, "PLACEBO",
  "PILOT01",    "EX", "11-1036",      1,       0, "PLACEBO",
  "PILOT01",    "EX", "11-1036",      2,       0, "PLACEBO",
  "PILOT01",    "EX", "11-1036",      3,       0, "PLACEBO",
  "PILOT01",    "EX", "14-1425",      1,      54,    "XANO",
  "PILOT01",    "EX", "15-1319",      1,      54,    "XANO",
  "PILOT01",    "EX", "15-1319",      2,      81,    "XANO",
  "PILOT01",    "EX", "16-1151",      1,      54,    "XANO",
  "PILOT01",    "EX", "16-1151",      2,      54,    "XANO"
)


# Select first dose for each patient
ex %>%
  filter_extreme(
    by_vars = exprs(USUBJID),
    order = exprs(EXSEQ),
    mode = "first"
  ) %>%
  select(USUBJID, EXSEQ)

# Select highest dose for each patient on the active drug
ex %>%
  filter(EXTRT != "PLACEBO") %>%
  filter_extreme(
    by_vars = exprs(USUBJID),
    order = exprs(EXDOSE),
    mode = "last",
    check_type = "none"
  ) %>%
  select(USUBJID, EXTRT, EXDOSE)
```

`filter_joined` *Filter Observations Taking Other Observations into Account*

Description

The function filters observation using a condition taking other observations into account. For example, it could select all observations with `AVALC == "Y"` and `AVALC == "Y"` for at least one subsequent observation. The input dataset is joined with itself to enable conditions taking variables from both the current observation and the other observations into account. The suffix ".join" is added to the variables from the subsequent observations.

An example usage might be checking if a patient received two required medications within a certain timeframe of each other.

In the oncology setting, for example, we use such processing to check if a response value can be confirmed by a subsequent assessment. This is commonly used in endpoints such as best overall response.

Usage

```
filter_joined(
  dataset,
  dataset_add,
  by_vars,
  join_vars,
  join_type,
  first_cond_lower = NULL,
  first_cond_upper = NULL,
  order = NULL,
  tmp_obs_nr_var = NULL,
  filter_add = NULL,
  filter_join,
  check_type = "warning"
)
```

Arguments

dataset	Input dataset The variables specified by the by_vars and order arguments are expected to be in the dataset. **Permitted values** a dataset, i.e., a data.frame or tibble **Default value** none
dataset_add	Additional dataset The variables specified for by_vars, join_vars, and order are expected. **Permitted values** a dataset, i.e., a data.frame or tibble **Default value** none
by_vars	By variables The specified variables are used as by variables for joining the input dataset with itself. **Permitted values** list of variables created by exprs(), e.g., exprs(USUBJID, VISIT) **Default value** none
join_vars	Variables to keep from joined dataset The variables needed from the other observations should be specified for this parameter. The specified variables are added to the joined dataset with suffix .join. For example to select all observations with AVALC == "Y" and AVALC == "Y" for at least one subsequent visit join_vars = exprs(AVALC, AVISITN) and filter_join = AVALC == "Y" & AVALC.join == "Y" & AVISITN < AVISITN.join could be specified. The *.join variables are not included in the output dataset. The variable specified for tmp_obs_nr_var must not be included in join_vars. It is added automatically to the joined dataset with the suffix .join. **Permitted values** list of variables created by exprs(), e.g., exprs(USUBJID, VISIT)

Default value none

`join_type` Observations to keep after joining

The argument determines which of the joined observations are kept with respect to the original observation. For example, if `join_type = "after"` is specified all observations after the original observations are kept.

For example for confirmed response or BOR in the oncology setting or confirmed deterioration in questionnaires the confirmatory assessment must be after the assessment. Thus `join_type = "after"` could be used.

Whereas, sometimes you might allow for confirmatory observations to occur prior to the observation. For example, to identify AEs occurring on or after seven days before a COVID AE. Thus `join_type = "all"` could be used.

Permitted values `"before"`, `"after"`, `"all"`

Default value none

`first_cond_lower`

Condition for selecting range of data (before)

If this argument is specified, the other observations are restricted from the first observation before the current observation where the specified condition is fulfilled up to the current observation. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

This parameter should be specified if `filter_join` contains summary functions which should not apply to all observations but only from a certain observation before the current observation up to the current observation. For examples see the "Examples" section below.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`first_cond_upper`

Condition for selecting range of data (after)

If this argument is specified, the other observations are restricted up to the first observation where the specified condition is fulfilled. If the condition is not fulfilled for any of the other observations, no observations are considered, i.e., the observation is not flagged.

This parameter should be specified if `filter_join` contains summary functions which should not apply to all observations but only up to the confirmation assessment. For examples see the "Examples" section below.

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`order` Order

The observations are ordered by the specified order.

For handling of `NA`s in sorting variables see the "Sort Order" section in `vignette("generic")`.

Permitted values list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))` or `NULL`

Default value `NULL`

tmp_obs_nr_var Temporary observation number

The specified variable is added to the input dataset (dataset) and the additional dataset (dataset_add). It is set to the observation number with respect to order. For each by group (by_vars) the observation number starts with 1. If there is more than one record for specific values for by_vars and order, all records get the same observation number. By default, a warning (see check_type) is issued in this case. The variable can be used in the conditions (filter_join, first_cond_upper, first_cond_lower). It is not included in the output dataset. It can also be used to select consecutive observations or the last observation (see example below).

Permitted values an unquoted symbol, e.g., AVAL

Default value NULL

filter_add Filter for additional dataset (dataset_add)

Only observations from dataset_add fulfilling the specified condition are joined to the input dataset. If the argument is not specified, all observations are joined.

Variables created by the order argument can be used in the condition.

The condition can include summary functions. The additional dataset is grouped by the by variables (by_vars).

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value NULL

filter_join Condition for selecting observations

The filter is applied to the joined dataset for selecting the confirmed observations. The condition can include summary functions like all() or any(). The joined dataset is grouped by the original observations. I.e., the summary function are applied to all observations up to the confirmation observation. For example in the oncology setting when using this function for confirmed best overall response, filter_join = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) & count_vals(var = AVALC.join, val = "NE") <= 1 selects observations with response "CR" and for all observations up to the confirmation observation the response is "CR" or "NE" and there is at most one "NE".

Permitted values an unquoted condition, e.g., AVISIT == "BASELINE"

Default value none

check_type Check uniqueness?

If "message", "warning", or "error" is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values "none", "message", "warning", "error"

Default value "warning"

Details

The following steps are performed to produce the output dataset.

Step 1:

- The variables specified by order are added to the additional dataset (dataset_add).

- The variables specified by `join_vars` are added to the additional dataset (`dataset_add`).
- The records from the additional dataset (`dataset_add`) are restricted to those matching the `filter_add` condition.

Then the input dataset (`dataset`) is joined with the restricted additional dataset by the variables specified for `by_vars`. From the additional dataset only the variables specified for `join_vars` are kept. The suffix ".join" is added to those variables which are also present in the input dataset.

For example, for `by_vars` = `USUBJID`, `join_vars` = `exprs(AVISITN, AVALC)` and input dataset and additional dataset

```
# A tibble: 2 x 4
USUBJID AVISITN AVALC  AVAL
<chr>     <dbl> <chr> <dbl>
1             1 Y         1
1             2 N         0
```

the joined dataset is

```
A tibble: 4 x 6
USUBJID AVISITN AVALC  AVAL AVISITN.join AVALC.join
<chr>     <dbl> <chr> <dbl>        <dbl> <chr>
1             1 Y         1            1 Y
1             1 Y         1            2 N
1             2 N         0            1 Y
1             2 N         0            2 N
```

Step 2:

The joined dataset is restricted to observations with respect to `join_type` and `order`.

The dataset from the example in the previous step with `join_type` = `"after"` and `order` = `exprs(AVISITN)` is restricted to

```
A tibble: 4 x 6
USUBJID AVISITN AVALC  AVAL AVISITN.join AVALC.join
<chr>     <dbl> <chr> <dbl>        <dbl> <chr>
1             1 Y         1            2 N
```

Step 3:

If `first_cond_lower` is specified, for each observation of the input dataset the joined dataset is restricted to observations from the first observation where `first_cond_lower` is fulfilled (the observation fulfilling the condition is included) up to the observation of the input dataset. If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

If `first_cond_upper` is specified, for each observation of the input dataset the joined dataset is restricted to observations up to the first observation where `first_cond_upper` is fulfilled (the observation fulfilling the condition is included). If for an observation of the input dataset the condition is not fulfilled, the observation is removed.

For an example see the last example in the "Examples" section.

Step 4:

The joined dataset is grouped by the observations from the input dataset and restricted to the observations fulfilling the condition specified by `filter_join`.

Step 5:

The first observation of each group is selected and the `*.join` variables are dropped.

Note: This function creates temporary datasets which may be much bigger than the input datasets. If this causes memory issues, please try setting the admiral option `save_memory` to `TRUE` (see `set_admiral_options()`). This reduces the memory consumption but increases the run-time.

Value

A subset of the observations of the input dataset. All variables of the input dataset are included in the output dataset.

Examples

Filter records considering other records (`filter_join`, `join_vars`):

In this example, the input dataset should be restricted to records with a duration longer than 30 and where a COVID AE (`ACOVFL == "Y"`) occurred before or up to seven days after the record. The condition for restricting the records is specified by the `filter_join` argument. Variables from the other records are referenced by variable names with the suffix `.join`. These variables have to be specified for the `join_vars` argument. As records before *and* after the current record should be considered, `join_type = "all"` is specified.

```
library(tibble)

adae <- tribble(
  ~USUBJID, ~ADY, ~ACOVFL, ~ADURN,
  "1",        10, "N",          1,
  "1",        21, "N",         50,
  "1",        23, "Y",         14,
  "1",        32, "N",         31,
  "1",        42, "N",         20,
  "2",        11, "Y",         13,
  "2",        23, "N",          2,
  "3",        13, "Y",         12,
  "4",        14, "N",         32,
  "4",        21, "N",         41
)

filter_joined(
  adae,
  dataset_add = adae,
  by_vars = exprs(USUBJID),
  join_vars = exprs(ACOVFL, ADY),
  join_type = "all",
  filter_join = ADURN > 30 & ACOVFL.join == "Y" & ADY.join <= ADY + 7
)
#> # A tibble: 2 × 4
#>   USUBJID   ADY ACOVFL ADURN
#>   <chr>   <dbl> <chr>  <dbl>
#> 1 1          21 N         50
```

```
#> 2 1          32 N         31
```

Considering only records after the current one (`join_type = "after"`):

In this example, the input dataset is restricted to records with `AVALC == "Y"` and `AVALC == "Y"` at a subsequent visit. `join_type = "after"` is specified to consider only records after the current one. Please note that the `order` argument must be specified, as otherwise it is not possible to determine which records are after the current record.

```
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "Y",
  "1",      2,        "N",
  "1",      3,        "Y",
  "1",      4,        "N",
  "2",      1,        "Y",
  "2",      2,        "N",
  "3",      1,        "Y",
  "4",      1,        "N",
  "4",      2,        "N",
)

filter_joined(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  join_vars = exprs(AVALC, AVISITN),
  join_type = "after",
  order = exprs(AVISITN),
  filter_join = AVALC == "Y" & AVALC.join == "Y"
)
#> # A tibble: 1 × 3
#>   USUBJID AVISITN AVALC
#>   <chr>     <dbl> <chr>
#> 1 1             1 Y
```

Considering a range of records only (`first_cond_lower`, `first_cond_upper`):

Consider the following data.

```
myd <- tribble(
  ~subj, ~day, ~val,
  "1",      1, "++",
  "1",      2, "-",
  "1",      3, "0",
  "1",      4, "+",
  "1",      5, "++",
  "1",      6, "-",
  "2",      1, "-",
  "2",      2, "++",
  "2",      3, "+",
```

```
  "2",      4, "0",
  "2",      5, "-",
  "2",      6, "++"
)
```

To select `"0"` where all results from the first `"++"` before the `"0"` up to the `"0"` (excluding the `"0"`) are `"+"` or `"++"` the `first_cond_lower` argument and `join_type = "before"` are specified.

```
filter_joined(
  myd,
  dataset_add = myd,
  by_vars = exprs(subj),
  order = exprs(day),
  join_vars = exprs(val),
  join_type = "before",
  first_cond_lower = val.join == "++",
  filter_join = val == "0" & all(val.join %in% c("+", "++"))
)
#> # A tibble: 1 × 3
#>   subj    day val
#>   <chr> <dbl> <chr>
#> 1 2         4 0
```

To select `"0"` where all results from the `"0"` (excluding the `"0"`) up to the first `"++"` after the `"0"` are `"+"` or `"++"` the `first_cond_upper` argument and `join_type = "after"` are specified.

```
filter_joined(
  myd,
  dataset_add = myd,
  by_vars = exprs(subj),
  order = exprs(day),
  join_vars = exprs(val),
  join_type = "after",
  first_cond_upper = val.join == "++",
  filter_join = val == "0" & all(val.join %in% c("+", "++"))
)
#> # A tibble: 1 × 3
#>   subj    day val
#>   <chr> <dbl> <chr>
#> 1 1         3 0
```

Considering only records up to a condition (`first_cond_upper`):
In this example from deriving confirmed response in oncology, the records with

- `AVALC == "CR"`,
- `AVALC == "CR"` at a subsequent visit,
- only `"CR"` or `"NE"` in between, and
- at most one `"NE"` in between

should be selected. The other records to be considered are restricted to those up to the first occurrence of `"CR"` by specifying the `first_cond_upper` argument. The `count_vals()` function is used to count the `"NE"`s for the last condition.

```
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "NE",
  "1",      4,        "CR",
  "1",      5,        "NE",
  "2",      1,        "CR",
  "2",      2,        "PR",
  "2",      3,        "CR",
  "3",      1,        "CR",
  "4",      1,        "CR",
  "4",      2,        "NE",
  "4",      3,        "NE",
  "4",      4,        "CR",
  "4",      5,        "PR"
)

filter_joined(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  join_vars = exprs(AVALC),
  join_type = "after",
  order = exprs(AVISITN),
  first_cond_upper = AVALC.join == "CR",
  filter_join = AVALC == "CR" & all(AVALC.join %in% c("CR", "NE")) &
    count_vals(var = AVALC.join, val = "NE") <= 1
)
#> # A tibble: 1 × 3
#>   USUBJID AVISITN AVALC
#>   <chr>     <dbl> <chr>
#> 1 1             2 CR
```

Considering order of values (`min_cond()`, `max_cond()`):

In this example from deriving confirmed response in oncology, records with

- `AVALC == "PR"`,
- `AVALC == "CR"` or `AVALC == "PR"` at a subsequent visit at least 20 days later,
- only `"CR"`, `"PR"`, or `"NE"` in between,
- at most one `"NE"` in between, and
- `"CR"` is not followed by `"PR"`

should be selected. The last condition is realized by using `min_cond()` and `max_cond()`, ensuring that the first occurrence of `"CR"` is after the last occurrence of `"PR"`. The second call to `count_vals()` in the condition is required to cover the case of no `"CR"`s (the `min_cond()` call returns `NA` then).

```
data <- tribble(
  ~USUBJID, ~ADY, ~AVALC,
```

```
  "1",          6, "PR",
  "1",         12, "CR",
  "1",         24, "NE",
  "1",         32, "CR",
  "1",         48, "PR",
  "2",          3, "PR",
  "2",         21, "CR",
  "2",         33, "PR",
  "3",         11, "PR",
  "4",          7, "PR",
  "4",         12, "NE",
  "4",         24, "NE",
  "4",         32, "PR",
  "4",         55, "PR"
)

filter_joined(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  join_vars = exprs(AVALC, ADY),
  join_type = "after",
  order = exprs(ADY),
  first_cond_upper = AVALC.join %in% c("CR", "PR") & ADY.join - ADY >= 20,
  filter_join = AVALC == "PR" &
    all(AVALC.join %in% c("CR", "PR", "NE")) &
    count_vals(var = AVALC.join, val = "NE") <= 1 &
    (
      min_cond(var = ADY.join, cond = AVALC.join == "CR") >
        max_cond(var = ADY.join, cond = AVALC.join == "PR") |
        count_vals(var = AVALC.join, val = "CR") == 0
    )
)
#> # A tibble: 1 × 3
#>   USUBJID   ADY AVALC
#>   <chr>   <dbl> <chr>
#> 1 4          32 PR
```

Considering the order of records (`tmp_obs_nr_var`)**:**

In this example, the records with `CRIT1FL == "Y"` at two consecutive visits or at the last visit should be selected. A temporary order variable is created by specifying the `tmp_obs_nr_var` argument. Then it is used in `filter_join`. The temporary variable doesn't need to be specified for `join_vars`.

```
data <- tribble(
  ~USUBJID, ~AVISITN, ~CRIT1FL,
  "1",      1,        "Y",
  "1",      2,        "N",
  "1",      3,        "Y",
```

```
  "1",      5,        "N",
  "2",      1,        "Y",
  "2",      3,        "Y",
  "2",      5,        "N",
  "3",      1,        "Y",
  "4",      1,        "Y",
  "4",      2,        "N",
)

filter_joined(
  data,
  dataset_add = data,
  by_vars = exprs(USUBJID),
  tmp_obs_nr_var = tmp_obs_nr,
  join_vars = exprs(CRIT1FL),
  join_type = "all",
  order = exprs(AVISITN),
  filter_join = CRIT1FL == "Y" & CRIT1FL.join == "Y" &
    (tmp_obs_nr + 1 == tmp_obs_nr.join | tmp_obs_nr == max(tmp_obs_nr.join))
)
#> # A tibble: 2 × 3
#>   USUBJID AVISITN CRIT1FL
#>   <chr>     <dbl> <chr>
#> 1 2             1 Y
#> 2 3             1 Y
```

See Also

count_vals(), min_cond(), max_cond()

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_not_exist(), filter_relative(), max_cond(), min_cond()

`filter_not_exist` *Returns records that don't fit into existing by groups in a filtered source dataset*

Description

Returns all records in the input dataset that belong to by groups that are not present in a source dataset, after the source dataset is optionally filtered. For example, this could be used to return ADSL records for subjects that didn't take certain concomitant medications during the course of the study (as per records in ADCM).

Usage

```
filter_not_exist(dataset, dataset_add, by_vars, filter_add = NULL)
```

Arguments

dataset	Input dataset The variables specified by the by_vars argument are expected to be in the dataset. **Default value** none
dataset_add	Source dataset The source dataset, which determines the by groups returned in the input dataset, based on the groups that don't exist in this dataset after being subset by filter_add. The variables specified in the by_vars and filter_add parameters are expected in this dataset. **Default value** none
by_vars	Grouping variables **Default value** none
filter_add	Filter for the source dataset The filter condition which will be used to subset the source dataset. Alternatively, if no filter condition is supplied, no subsetting of the source dataset will be performed. **Default value** NULL

Details

Returns the records in dataset which don't match any existing by groups in dataset_add, after being filtered according to filter_add. If all by groups that exist in dataset don't exist in dataset_add, an empty dataset will be returned.

Value

The records in the input dataset which are not contained within any existing by group in the filtered source dataset.

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_joined(), filter_relative(), max_cond(), min_cond()

Examples

```
# Get demographic information about subjects who didn't take vitamin supplements
# during the study

library(tibble)

adsl <- tribble(
  ~USUBJID,      ~AGE, ~SEX,
  "01-701-1015", 63,   "F",
  "01-701-1023", 64,   "M",
```

```
  "01-701-1034", 77,   "F",
  "01-701-1118", 52,   "M"
)

adcm <- tribble(
  ~USUBJID,      ~CMTRT,          ~CMSTDTC,
  "01-701-1015", "ASPIRIN",       "2013-05-14",
  "01-701-1023", "MYLANTA",       "2014-01-04",
  "01-701-1023", "CALCIUM",       "2014-02-25",
  "01-701-1034", "VITAMIN C",     "2013-12-12",
  "01-701-1034", "CALCIUM",       "2013-03-27",
  "01-701-1118", "MULTIVITAMIN", "2013-02-21"
)

filter_not_exist(
  dataset = adsl,
  dataset_add = adcm,
  by_vars = exprs(USUBJID),
  filter_add = str_detect(CMTRT, "VITAMIN")
)
```

`filter_relative` *Filter the Observations Before or After a Condition is Fulfilled*

Description

Filters the observations before or after the observation where a specified condition is fulfilled for each by group. For example, the function could be called to select for each subject all observations before the first disease progression.

Usage

```
filter_relative(
  dataset,
  by_vars,
  order,
  condition,
  mode,
  selection,
  inclusive,
  keep_no_ref_groups = TRUE,
  check_type = "warning"
)
```

Arguments

`dataset`	Input dataset The variables specified by the `by_vars` and `order` arguments are expected to be in the dataset.

Default value none

by_vars — Grouping variables

Default value none

order — Sort order

Within each by group the observations are ordered by the specified order.

For handling of NAs in sorting variables see the "Sort Order" section in `vignette("generic")`.

Permitted values list of expressions created by `exprs()`, e.g., `exprs(ADT, desc(AVAL))`

Default value none

condition — Condition for Reference Observation

The specified condition determines the reference observation. The output dataset contains all observations before or after (`selection` parameter) the reference observation.

Default value none

mode — Selection mode (first or last)

If `"first"` is specified, for each by group the observations before or after (`selection` parameter) the observation where the condition (`condition` parameter) is fulfilled the *first* time is included in the output dataset. If `"last"` is specified, for each by group the observations before or after (`selection` parameter) the observation where the condition (`condition` parameter) is fulfilled the *last* time is included in the output dataset.

Permitted values `"first"`, `"last"`

Default value none

selection — Select observations before or after the reference observation?

Permitted values `"before"`, `"after"`

Default value none

inclusive — Include the reference observation?

Permitted values `TRUE`, `FALSE`

Default value none

keep_no_ref_groups

Should by groups without reference observation be kept?

Permitted values `TRUE`, `FALSE`

Default value `TRUE`

check_type — Check uniqueness?

If `"warning"` or `"error"` is specified, the specified message is issued if the observations of the input dataset are not unique with respect to the by variables and the order.

Permitted values `"none"`, `"warning"`, `"error"`

Default value `"warning"`

Details

For each by group (by_vars parameter) the observations before or after (selection parameter) the observations where the condition (condition parameter) is fulfilled the first or last time (order parameter and mode parameter) is included in the output dataset.

Value

A dataset containing for each by group the observations before or after the observation where the condition was fulfilled the first or last time

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_joined(), filter_not_exist(), max_cond(), min_cond()

Examples

```
library(tibble)

response <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "CR",
  "1",      4,        "SD",
  "1",      5,        "NE",
  "2",      1,        "SD",
  "2",      2,        "PD",
  "2",      3,        "PD",
  "3",      1,        "SD",
  "4",      1,        "SD",
  "4",      2,        "PR",
  "4",      3,        "PD",
  "4",      4,        "SD",
  "4",      5,        "PR"
)

# Select observations up to first PD for each patient
response %>%
  filter_relative(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )

# Select observations after last CR, PR, or SD for each patient
response %>%
  filter_relative(
```

```
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC %in% c("CR", "PR", "SD"),
    mode = "last",
    selection = "after",
    inclusive = FALSE
  )

# Select observations from first response to first PD
response %>%
  filter_relative(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC %in% c("CR", "PR"),
    mode = "first",
    selection = "after",
    inclusive = TRUE,
    keep_no_ref_groups = FALSE
  ) %>%
  filter_relative(
    by_vars = exprs(USUBJID),
    order = exprs(AVISITN),
    condition = AVALC == "PD",
    mode = "first",
    selection = "before",
    inclusive = TRUE
  )
```

`flag_event` *Create a* `flag_event` *Object*

Description

The `flag_event` object is used to define events as input for the `derive_var_merged_ef_msrc()` function.

Usage

```
flag_event(dataset_name, condition = NULL, by_vars = NULL)
```

Arguments

`dataset_name`	Dataset name of the dataset to be used as input for the event. The name refers to the dataset specified for `source_datasets` in `derive_var_merged_ef_msrc()`. **Permitted values** a dataset, i.e., a `data.frame` or tibble **Default value** none
`condition`	Condition The condition is evaluated at the dataset referenced by `dataset_name`. For all by groups where it evaluates as `TRUE` at least once the new variable is set to the true value (`true_value`).

Permitted values an unquoted condition, e.g., `AVISIT == "BASELINE"`

Default value `NULL`

`by_vars` Grouping variables

If specified, the dataset is grouped by the specified variables before the condition is evaluated. If named elements are used in `by_vars` like `by_vars = exprs(USUBJID, EXLNKID = ECLNKID)`, the variables are renamed after the evaluation. If the `by_vars` element is not specified, the observations are grouped by the variables specified for the `by_vars` argument of `derive_var_merged_ef_msrc()`.

Permitted values list of variables created by `exprs()`, e.g., `exprs(USUBJID, VISIT)`

Default value `NULL`

See Also

`derive_var_merged_ef_msrc()`

Source Objects: `basket_select()`, `censor_source()`, `death_event`, `event()`, `event_joined()`, `event_source()`, `query()`, `records_source()`, `tte_source()`

`get_admiral_option` *Get the Value of an Admiral Option*

Description

Get the Value of an Admiral Option Which Can Be Modified for Advanced Users.

Usage

```
get_admiral_option(option)
```

Arguments

`option` A character scalar of commonly used admiral function inputs.

As of now, support only available for "subject_keys", "signif_digits", and "save_memory". See `set_admiral_options()` for a description of the options.

Default value none

Details

This function allows flexibility for function inputs that may need to be repeated multiple times in a script, such as `subject_keys`.

Value

The value of the specified option.

See Also

set_admiral_options(), derive_param_exist_flag(), derive_param_tte() derive_var_dthcaus(), derive_var_extreme_dtm(), derive_vars_period(), create_period_dataset()

Other admiral_options: set_admiral_options()

Examples

```
library(dplyr, warn.conflicts = FALSE)
dm <- tribble(
  ~STUDYID, ~DOMAIN,  ~USUBJID, ~AGE,   ~AGEU,
  "PILOT01",   "DM", "01-1302",   61, "YEARS",
  "PILOT01",   "DM", "17-1344",   64, "YEARS"
)

vs <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID, ~VSTESTCD,     ~VISIT,     ~VSTPT, ~VSSTRESN,
  "PILOT01",    "VS", "01-1302",   "DIABP", "BASELINE",    "LYING",        76,
  "PILOT01",    "VS", "01-1302",   "DIABP", "BASELINE", "STANDING",        87,
  "PILOT01",    "VS", "01-1302",   "DIABP",   "WEEK 2",    "LYING",        71,
  "PILOT01",    "VS", "01-1302",   "DIABP",   "WEEK 2", "STANDING",        79,
  "PILOT01",    "VS", "17-1344",   "DIABP", "BASELINE",    "LYING",        88,
  "PILOT01",    "VS", "17-1344",   "DIABP", "BASELINE", "STANDING",        86,
  "PILOT01",    "VS", "17-1344",   "DIABP",   "WEEK 2",    "LYING",        84,
  "PILOT01",    "VS", "17-1344",   "DIABP",   "WEEK 2", "STANDING",        82
)

# Merging all dm variables to vs
derive_vars_merged(
  vs,
  dataset_add = select(dm, -DOMAIN),
  by_vars = get_admiral_option("subject_keys")
)
```

---

get_duplicates_dataset

Get Duplicate Records that Led to a Prior Error

Description

Get Duplicate Records that Led to a Prior Error

Usage

```
get_duplicates_dataset()
```

Details

Many {admiral} function check that the input dataset contains only one record per by_vars group and throw an error otherwise. The get_duplicates_dataset() function allows one to retrieve the duplicate records that lead to an error.

Note that the function always returns the dataset of duplicates from the last error that has been thrown in the current R session. Thus, after restarting the R sessions get_duplicates_dataset() will return NULL and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

Value

A data.frame or NULL

See Also

Utilities for Dataset Checking: get_many_to_one_dataset(), get_one_to_many_dataset()

Examples

```
data(admiral_adsl)

# Duplicate the first record
adsl <- rbind(admiral_adsl[1L, ], admiral_adsl)

signal_duplicate_records(adsl, exprs(USUBJID), cnd_type = "warning")

get_duplicates_dataset()
```

---

get_flagged_records *Create an Existence Flag*

Description

Create a flag variable for the input dataset which indicates if there exists at least one observation in the input dataset fulfilling a certain condition.

Note: This is a helper function for derive_vars_merged_exist_flag() which inputs this result into derive_vars_merged().

Usage

```
get_flagged_records(dataset, new_var, condition, filter = NULL)
```

Arguments

`dataset`	Input dataset **Default value** none
`new_var`	New variable The specified variable is added to the input dataset. **Default value** none
`condition`	Condition The condition is evaluated at the dataset (`dataset`). For all rows where it evaluates as `TRUE` the new variable is set to `1` in the new column. Otherwise, it is set to `0`. **Default value** none
`filter`	Filter for additional data Only observations fulfilling the specified condition are taken into account for flagging. If the argument is not specified, all observations are considered. **Permitted values** a condition **Default value** `NULL`

Value

The output dataset is the input dataset filtered by the `filter` condition and with the variable specified for `new_var` representing a flag for the condition.

See Also

Utilities used within Derivation functions: `extract_unit()`, `get_not_mapped()`, `get_vars_query()`

Examples

```
library(dplyr, warn.conflicts = FALSE)


ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,    ~AETERM,     ~AEREL,
  "PILOT01",    "AE", "01-1028", "ERYTHEMA", "POSSIBLE",
  "PILOT01",    "AE", "01-1028", "PRURITUS", "PROBABLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "POSSIBLE",
  "PILOT01",    "AE", "06-1049",  "SYNCOPE", "PROBABLE"
)


get_flagged_records(
  dataset = ae,
  new_var = AERELFL,
  condition = AEREL == "PROBABLE"
) %>%
  select(STUDYID, USUBJID, AERELFL)
```

```
vs <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,      ~VISIT, ~VSTESTCD, ~VSSTRESN, ~VSBLFL,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "HEIGHT",     177.8,      NA,
  "PILOT01",    "VS", "01-1028", "SCREENING",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "01-1028",  "BASELINE",  "WEIGHT",     99.34,     "Y",
  "PILOT01",    "VS", "01-1028",    "WEEK 4",  "WEIGHT",     98.88,      NA,
  "PILOT01",    "VS", "04-1127", "SCREENING",  "HEIGHT",     165.1,      NA,
  "PILOT01",    "VS", "04-1127", "SCREENING",  "WEIGHT",     42.87,      NA,
  "PILOT01",    "VS", "04-1127",  "BASELINE",  "WEIGHT",     41.05,     "Y",
  "PILOT01",    "VS", "04-1127",    "WEEK 4",  "WEIGHT",     41.73,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "HEIGHT",    167.64,      NA,
  "PILOT01",    "VS", "06-1049", "SCREENING",  "WEIGHT",     57.61,      NA,
  "PILOT01",    "VS", "06-1049",  "BASELINE",  "WEIGHT",     57.83,     "Y",
  "PILOT01",    "VS", "06-1049",    "WEEK 4",  "WEIGHT",     58.97,      NA
)
get_flagged_records(
  dataset = vs,
  new_var = WTBLHIFL,
  condition = VSSTRESN > 90,
  filter = VSTESTCD == "WEIGHT" & VSBLFL == "Y"
) %>%
  select(STUDYID, USUBJID, WTBLHIFL)
```

`get_many_to_one_dataset` *Get Many to One Values that Led to a Prior Error*

Description

Get Many to One Values that Led to a Prior Error

Usage

```
get_many_to_one_dataset()
```

Details

If `assert_one_to_one()` detects an issue, the many to one values are stored in a dataset. This dataset can be retrieved by `get_many_to_one_dataset()`.

Note that the function always returns the many to one values from the last error that has been thrown in the current R session. Thus, after restarting the R sessions `get_many_to_one_dataset()` will return `NULL` and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

Value

A `data.frame` or `NULL`

See Also

Utilities for Dataset Checking: `get_duplicates_dataset()`, `get_one_to_many_dataset()`

Examples

```
library(admiraldev, warn.conflicts = FALSE)
data(admiral_adsl)

try(
  assert_one_to_one(admiral_adsl, exprs(SITEID), exprs(STUDYID))
)

get_many_to_one_dataset()
```

---

get_not_mapped *Get list of records not mapped from the lookup table.*

Description

Get list of records not mapped from the lookup table.

Usage

```
get_not_mapped()
```

Value

A `data.frame` or `NULL`

See Also

Utilities used within Derivation functions: `extract_unit()`, `get_flagged_records()`, `get_vars_query()`

---

get_one_to_many_dataset *Get One to Many Values that Led to a Prior Error*

Description

Get One to Many Values that Led to a Prior Error

Usage

```
get_one_to_many_dataset()
```

Details

If assert_one_to_one() detects an issue, the one to many values are stored in a dataset. This dataset can be retrieved by get_one_to_many_dataset().

Note that the function always returns the one to many values from the last error that has been thrown in the current R session. Thus, after restarting the R sessions get_one_to_many_dataset() will return NULL and after a second error has been thrown, the dataset of the first error can no longer be accessed (unless it has been saved in a variable).

Value

A data.frame or NULL

See Also

Utilities for Dataset Checking: get_duplicates_dataset(), get_many_to_one_dataset()

Examples

```
library(admiraldev, warn.conflicts = FALSE)
data(admiral_adsl)

try(
  assert_one_to_one(admiral_adsl, exprs(STUDYID), exprs(SITEID))
)

get_one_to_many_dataset()
```

---

get_summary_records *Create Summary Records*

Description

[Deprecated] The get_summary_records() has been deprecated in favor of derive_summary_records() (call it with the dataset_add argument and without the dataset argument).

It is not uncommon to have an analysis need whereby one needs to derive an analysis value (AVAL) from multiple records. The ADaM basic dataset structure variable DTYPE is available to indicate when a new derived records has been added to a dataset.

Usage

```
get_summary_records(dataset, by_vars, filter = NULL, set_values_to = NULL)
```

Arguments

`dataset` Input dataset

The variables specified by the `by_vars` argument are expected to be in the dataset.

Default value none

`by_vars` Grouping variables

Variables to consider for generation of groupwise summary records.

Default value none

`filter` Filter condition as logical expression to apply during summary calculation. By default, filtering expressions are computed within `by_vars` as this will help when an aggregating, lagging, or ranking function is involved.

For example,

- `filter_rows = (AVAL > mean(AVAL, na.rm = TRUE))` will filter all AVAL values greater than mean of AVAL with in `by_vars`.
- `filter_rows = (dplyr::n() > 2)` will filter n count of `by_vars` greater than 2.

Default value `NULL`

`set_values_to` Variables to be set

The specified variables are set to the specified values for the new observations.

Set a list of variables to some specified value for the new records

- LHS refer to a variable.
- RHS refers to the values to set to the variable. This can be a string, a symbol, a numeric value, an expression or NA. If summary functions are used, the values are summarized by the variables specified for `by_vars`.

For example:

```
set_values_to = exprs(
  AVAL = sum(AVAL),
  PARAMCD = "TDOSE",
  PARCAT1 = "OVERALL"
)
```

Default value `NULL`

Details

This function only creates derived observations and does not append them to the original dataset observations. If you would like to this instead, see the `derive_summary_records()` function.

Value

A data frame of derived records.

See Also

derive_summary_records(), derive_vars_merged_summary()

Other deprecated: call_user_fun(), date_source(), derive_param_extreme_record(), derive_var_dthcaus(), derive_var_extreme_dt(), derive_var_extreme_dtm(), derive_var_merged_summary(), dthcaus_source()

Examples

```
library(tibble)

adeg <- tribble(
  ~USUBJID,   ~EGSEQ, ~PARAM,             ~AVISIT,    ~EGDTC,             ~AVAL, ~TRTA,
  "XYZ-1001", 1,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:50", 385,  NA_character_,
  "XYZ-1001", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:52", 399,  NA_character_,
  "XYZ-1001", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:56", 396,  NA_character_,
  "XYZ-1001", 4,     "QTcF Int. (msec)", "Visit 2",  "2016-03-08T09:45", 384,   "Placebo",
  "XYZ-1001", 5,     "QTcF Int. (msec)", "Visit 2",  "2016-03-08T09:48", 393,   "Placebo",
  "XYZ-1001", 6,     "QTcF Int. (msec)", "Visit 2",  "2016-03-08T09:51", 388,   "Placebo",
  "XYZ-1001", 7,     "QTcF Int. (msec)", "Visit 3",  "2016-03-22T10:45", 385,   "Placebo",
  "XYZ-1001", 8,     "QTcF Int. (msec)", "Visit 3",  "2016-03-22T10:48", 394,   "Placebo",
  "XYZ-1001", 9,     "QTcF Int. (msec)", "Visit 3",  "2016-03-22T10:51", 402,   "Placebo",
  "XYZ-1002", 1,    "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 399,  NA_character_,
  "XYZ-1002", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 410,  NA_character_,
  "XYZ-1002", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-22T08:01", 392,  NA_character_,
  "XYZ-1002", 4,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:50", 401,  "Active 20mg",
  "XYZ-1002", 5,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:53", 407,  "Active 20mg",
  "XYZ-1002", 6,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:56", 400,  "Active 20mg",
  "XYZ-1002", 7,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:50", 412,  "Active 20mg",
  "XYZ-1002", 8,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:53", 414,  "Active 20mg",
  "XYZ-1002", 9,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:56", 402,  "Active 20mg"
)

# Summarize the average of the triplicate ECG interval values (AVAL)
get_summary_records(
  adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    DTYPE = "AVERAGE"
  )
)

# Derive more than one summary variable
get_summary_records(
  adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  set_values_to = exprs(
    AVAL = mean(AVAL),
    ASTDTM = min(convert_dtc_to_dtm(EGDTC)),
    AENDTM = max(convert_dtc_to_dtm(EGDTC)),
    DTYPE = "AVERAGE"
  )
)
```

```
# Sample ADEG dataset with triplicate record for only AVISIT = 'Baseline'
adeg <- tribble(
  ~USUBJID,   ~EGSEQ, ~PARAM,             ~AVISIT,    ~EGDTC,             ~AVAL, ~TRTA,
 "XYZ-1001", 1,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:50", 385,  NA_character_,
 "XYZ-1001", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:52", 399,  NA_character_,
 "XYZ-1001", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-24T07:56", 396,  NA_character_,
 "XYZ-1001", 4,     "QTcF Int. (msec)", "Visit 2",  "2016-03-08T09:48", 393,   "Placebo",
 "XYZ-1001", 5,     "QTcF Int. (msec)", "Visit 2",  "2016-03-08T09:51", 388,   "Placebo",
 "XYZ-1001", 6,     "QTcF Int. (msec)", "Visit 3",  "2016-03-22T10:48", 394,   "Placebo",
 "XYZ-1001", 7,     "QTcF Int. (msec)", "Visit 3",  "2016-03-22T10:51", 402,   "Placebo",
 "XYZ-1002", 1,    "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 399,  NA_character_,
 "XYZ-1002", 2,    "QTcF Int. (msec)", "Baseline", "2016-02-22T07:58", 410,  NA_character_,
 "XYZ-1002", 3,    "QTcF Int. (msec)", "Baseline", "2016-02-22T08:01", 392,  NA_character_,
 "XYZ-1002", 4,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:53", 407,  "Active 20mg",
 "XYZ-1002", 5,    "QTcF Int. (msec)", "Visit 2",  "2016-03-06T09:56", 400,  "Active 20mg",
 "XYZ-1002", 6,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:53", 414,  "Active 20mg",
 "XYZ-1002", 7,    "QTcF Int. (msec)", "Visit 3",  "2016-03-24T10:56", 402,  "Active 20mg"
)

# Compute the average of AVAL only if there are more than 2 records within the
# by group
get_summary_records(
  adeg,
  by_vars = exprs(USUBJID, PARAM, AVISIT),
  filter = n() > 2,
  set_values_to = exprs(
    AVAL = mean(AVAL, na.rm = TRUE),
    DTYPE = "AVERAGE"
  )
)
```

`get_vars_query` *Get Query Variables*

Description

Create a table for the input dataset which binds the necessary rows for a `derive_vars_query()` call with the relevant `SRCVAR`, `TERM_NAME_ID` and a temporary index if it is necessary

Note: This function is the first step performed in `derive_vars_query()` requested by some users to be present independently from it.

Usage

```
get_vars_query(dataset, dataset_queries)
```

Arguments

`dataset`	Input dataset

Default value none

dataset_queries

A dataset containing required columns PREFIX, GRPNAME, SRCVAR, TERMCHAR and/or TERMNUM, and optional columns GRPID, SCOPE, SCOPEN.

create_query_data() can be used to create the dataset.

Default value none

Details

This function can be used to derive CDISC variables such as SMQzzNAM, SMQzzCD, SMQzzSC, SMQzzSCN, and CQzzNAM in ADAE and ADMH, and variables such as SDGzzNAM, SDGzzCD, and SDGzzSC in ADCM. An example usage of this function can be found in the vignette("occds").

A query dataset is expected as an input to this function. See the vignette("queries_dataset") for descriptions, or call data("queries") for an example of a query dataset.

For each unique element in PREFIX, the corresponding "NAM" variable will be created. For each unique PREFIX, if GRPID is not "" or NA, then the corresponding "CD" variable is created; similarly, if SCOPE is not "" or NA, then the corresponding "SC" variable will be created; if SCOPEN is not "" or NA, then the corresponding "SCN" variable will be created.

For each record in dataset, the "NAM" variable takes the value of GRPNAME if the value of TERMCHAR or TERMNUM in dataset_queries matches the value of the respective SRCVAR in dataset. Note that TERMCHAR in dataset_queries dataset may be NA only when TERMNUM is non-NA and vice versa. The matching is case insensitive. The "CD", "SC", and "SCN" variables are derived accordingly based on GRPID, SCOPE, and SCOPEN respectively, whenever not missing.

Value

The processed query dataset with SRCVAR and TERM_NAME_ID so that that can be merged to the input dataset to execute the derivations outlined by dataset_queries.

See Also

create_query_data()

Utilities used within Derivation functions: extract_unit(), get_flagged_records(), get_not_mapped()

Examples

```
library(tibble)
data("queries")
adae <- tribble(
  ~USUBJID, ~ASTDTM, ~AETERM, ~AESEQ, ~AEDECOD, ~AELLT, ~AELLTCD,
  "01", "2020-06-02 23:59:59", "ALANINE AMINOTRANSFERASE ABNORMAL",
  3, "Alanine aminotransferase abnormal", NA_character_, NA_integer_,
  "02", "2020-06-05 23:59:59", "BASEDOW'S DISEASE",
  5, "Basedow's disease", NA_character_, 1L,
  "03", "2020-06-07 23:59:59", "SOME TERM",
  2, "Some query", "Some term", NA_integer_,
  "05", "2020-06-09 23:59:59", "ALVEOLAR PROTEINOSIS",
  7, "Alveolar proteinosis", NA_character_, NA_integer_
)
```

```
get_vars_query(adae, queries)
```

`impute_dtc_dt` *Impute Partial Date Portion of a* `--DTC` *Variable*

Description

Imputation partial date portion of a `--DTC` variable based on user input.

Usage

```
impute_dtc_dt(
  dtc,
  highest_imputation = "n",
  date_imputation = "first",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

`dtc`

The `--DTC` date to impute

A character date is expected in a format like `yyyy-mm-dd` or `yyyy-mm-ddThh:mm:ss`. Trailing components can be omitted and `-` is a valid "missing" value for any component.

Permitted values a character date variable

Default value none

`highest_imputation`

Highest imputation level

The `highest_imputation` argument controls which components of the `--DTC` value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, `NA` is returned. For example, for `highest_imputation = "D"` `"2020"` results in `NA` because the month is missing.

If `"n"` (none, lowest level) is specified no imputation is performed, i.e., if any component is missing, `NA` is returned.

If `"Y"` (year, highest level) is specified, `date_imputation` must be `"first"` or `"last"` and `min_dates` or `max_dates` must be specified respectively. Otherwise, an error is thrown.

Permitted values `"Y"` (year, highest level), `"M"` (month), `"D"` (day), `"n"` (none, lowest level)

Default value `"n"`

date_imputation

The value to impute the day/month when a datepart is missing.

A character value is expected.

- The"first" and "last" keywords allow imputation to the first/last day/month. They can also be used to impute the year if used in conjunction with the min_dates or max_dates arguments. Some examples of this are available here.
- When highest_imputation is "M" or "D", the "mid" keyword can also be specified to impute missing components to the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).
- "<dd>" can be specified only if highest_imputation = "D". Missing days are imputed by the specified day, e.g. "10" for the 10th day of the month. The specified day should be valid for all months as otherwise an error might be issued. For example, date_imputation = "30" results in an invalid date of "2024-02-30" for the partial date "2024-02".
- "<mm>-<dd>" can be specified only if highest_imputation is "M", e.g. "06-15" for the 15th of June.

Permitted values a key-word, i.e. "first", "mid", "last", or "<mm>-<dd>" or "<dd>"

Default value "first"

min_dates

Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
   ymd_hms("2020-12-06T12:12:12"),
   ymd_hms("2020-11-11T11:11:11")
  ),
  highest_imputation = "M"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

max_dates — Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

Permitted values a list of dates, e.g. `list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))`

Default value `NULL`

preserve — Preserve day if month is missing and day is present

For example `"2019---07"` would return `"2019-06-07"` if `preserve = TRUE` (and `date_imputation = "MID"`).

Permitted values `TRUE`, `FALSE`

Default value `FALSE`

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Additionally, the function will throw an error if imputation rules cause an invalid datetime (e.g. "2020-02-31") to be generated. In this case, the user should adjust the imputation rules.

Value

A character vector

See Also

Date/Time Computation Functions that returns a vector: `compute_age_years()`, `compute_dtf()`, `compute_duration()`, `compute_tmf()`, `convert_date_to_dtm()`, `convert_dtc_to_dt()`, `convert_dtc_to_dtm()`, `convert_xxtpt_to_hours()`, `impute_dtc_dtm()`

Examples

```
library(lubridate)

dates <- c(
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18T15",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019",
  "2019---07",
  ""
)

# No date imputation (highest_imputation defaulted to "n")
impute_dtc_dt(dtc = dates)
```

```
# Impute to first day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M"
)
# Same as above
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "01-01"
)

# Impute to last day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "last",
)

# Impute to mid day/month if date is partial
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "mid"
)

# Impute to a given day of the month if only day is missing
impute_dtc_dt(
  dtc = dates,
  highest_imputation = "D",
  date_imputation = "10"
)

# Impute a date and ensure that the imputed date is not before a list of
# minimum dates
impute_dtc_dt(
  "2020-12",
  min_dates = list(
    ymd("2020-12-06"),
    ymd("2020-11-11")
  ),
  highest_imputation = "M"
)

# Impute completely missing dates (only possible if min_dates or max_dates is specified)
impute_dtc_dt(
  c("2020-12", NA_character_),
  min_dates = list(
    ymd("2020-12-06", "2020-01-01"),
    ymd("2020-11-11", NA)
  ),
  highest_imputation = "Y"
)
```

impute_dtc_dtm *Impute Partial Date(-time) Portion of a* -DTC *Variable*

Description

Imputation partial date/time portion of a --DTC variable. based on user input.

Usage

```
impute_dtc_dtm(
  dtc,
  highest_imputation = "h",
  date_imputation = "first",
  time_imputation = "first",
  min_dates = NULL,
  max_dates = NULL,
  preserve = FALSE
)
```

Arguments

dtc

The --DTC date to impute

A character date is expected in a format like yyyy-mm-dd or yyyy-mm-ddThh:mm:ss. Trailing components can be omitted and - is a valid "missing" value for any component.

Permitted values a character date variable

Default value none

highest_imputation

Highest imputation level

The highest_imputation argument controls which components of the --DTC value are imputed if they are missing. All components up to the specified level are imputed.

If a component at a higher level than the highest imputation level is missing, NA is returned. For example, for highest_imputation = "D" "2020" results in NA because the month is missing.

If "n" is specified, no imputation is performed, i.e., if any component is missing, NA is returned.

If "Y" is specified, date_imputation should be "first" or "last" and min_dates or max_dates should be specified respectively. Otherwise, NA is returned if the year component is missing.

Permitted values "Y" (year, highest level), "M" (month), "D" (day), "h" (hour), "m" (minute), "s" (second), "n" (none, lowest level)

Default value "h"

date_imputation

The value to impute the day/month when a datepart is missing.

A character value is expected.

- The"first" and "last" keywords allow imputation to the first/last day/month. They can also be used to impute the year if used in conjunction with the min_dates or max_dates arguments. Some examples of this are available here.
- When highest_imputation is "M" or "D", the "mid" keyword can also be specified to impute missing components to the middle of the possible range:
  - If both month and day are missing, they are imputed as "06-30" (middle of the year).
  - If only day is missing, it is imputed as "15" (middle of the month).
- "<dd>" can be specified only if highest_imputation = "D". Missing days are imputed by the specified day, e.g. "10" for the 10th day of the month. The specified day should be valid for all months as otherwise an error might be issued. For example, date_imputation = "30" results in an invalid date of "2024-02-30" for the partial date "2024-02".
- "<mm>-<dd>" can be specified only if highest_imputation is "M", e.g. "06-15" for the 15th of June.

Permitted values a key-word, i.e. "first", "mid", "last", or "<mm>-<dd>" or "<dd>"

Default value "first"

time_imputation

The value to impute the time when a timepart is missing.

A character value is expected, either as a

- format with hour, min and sec specified as "hh:mm:ss": e.g. "00:00:00" for the start of the day,
- or as a keyword: "first","last" to impute to the start/end of a day.

The argument is ignored if highest_imputation = "n".

Permitted values "first", "last", or user-defined

Default value "first"

min_dates Minimum dates

A list of dates is expected. It is ensured that the imputed date is not before any of the specified dates, e.g., that the imputed adverse event start date is not before the first treatment date. Only dates which are in the range of possible dates of the dtc value are considered. The possible dates are defined by the missing parts of the dtc date (see example below). This ensures that the non-missing parts of the dtc date are not changed. A date or date-time object is expected. For example

```
impute_dtc_dtm(
  "2020-11",
  min_dates = list(
   ymd_hm("2020-12-06T12:12"),
   ymd_hm("2020-11-11T11:11")
  ),
  highest_imputation = "M"
)
```

returns "2020-11-11T11:11:11" because the possible dates for "2020-11" range from "2020-11-01T00:00:00" to "2020-11-30T23:59:59". Therefore "2020-12-06T12:12:12" is ignored. Returning "2020-12-06T12:12:12" would have changed the month although it is not missing (in the dtc date).

For date variables (not datetime) in the list the time is imputed to "00:00:00". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

max_dates — Maximum dates

A list of dates is expected. It is ensured that the imputed date is not after any of the specified dates, e.g., that the imputed date is not after the data cut off date. Only dates which are in the range of possible dates are considered. A date or date-time object is expected.

For date variables (not datetime) in the list the time is imputed to "23:59:59". Specifying date variables makes sense only if the date is imputed. If only time is imputed, date variables do not affect the result.

Permitted values a list of dates, e.g. list(ymd_hms("2021-07-01T04:03:01"), ymd_hms("2022-05-12T13:57:23"))

Default value NULL

preserve — Preserve lower level date/time part when higher order part is missing, e.g. preserve day if month is missing or preserve minute when hour is missing.

For example "2019---07" would return "2019-06-07 if preserve = TRUE (and date_imputation = "mid").

Permitted values TRUE, FALSE

Default value FALSE

Details

This is a vector-oriented helper and is not usually called directly on a data frame with %>%.

Value

A character vector

See Also

Date/Time Computation Functions that returns a vector: compute_age_years(), compute_dtf(), compute_duration(), compute_tmf(), convert_date_to_dtm(), convert_dtc_to_dt(), convert_dtc_to_dtm(), convert_xxtpt_to_hours(), impute_dtc_dt()

Examples

```
library(lubridate)

dates <- c(
```

```
  "2019-07-18T15:25:40",
  "2019-07-18T15:25",
  "2019-07-18T15",
  "2019-07-18",
  "2019-02",
  "2019",
  "2019",
  "2019---07",
  ""
)

# No date imputation (highest_imputation defaulted to "h")
# Missing time part imputed with 00:00:00 portion by default
impute_dtc_dtm(dtc = dates)

# No date imputation (highest_imputation defaulted to "h")
# Missing time part imputed with 23:59:59 portion
impute_dtc_dtm(
  dtc = dates,
  time_imputation = "23:59:59"
)

# Same as above
impute_dtc_dtm(
  dtc = dates,
  time_imputation = "last"
)

# Impute to first day/month if date is partial
# Missing time part imputed with 00:00:00 portion by default
impute_dtc_dtm(
  dtc = dates,
  highest_imputation = "M"
)
# same as above
impute_dtc_dtm(
  dtc = dates,
  highest_imputation = "M",
  date_imputation = "01-01"
)

# Impute to last day/month if date is partial
# Missing time part imputed with 23:59:59 portion
impute_dtc_dtm(
  dtc = dates,
  date_imputation = "last",
  time_imputation = "last"
)

# Impute to mid day/month if date is partial
# Missing time part imputed with 00:00:00 portion by default
impute_dtc_dtm(
  dtc = dates,
```

```
  highest_imputation = "M",
  date_imputation = "mid"
)

# Impute a date and ensure that the imputed date is not before a list of
# minimum dates
impute_dtc_dtm(
  "2020-12",
  min_dates = list(
    ymd_hm("2020-12-06T12:12"),
    ymd_hm("2020-11-11T11:11")
  ),
  highest_imputation = "M"
)

# Impute completely missing dates (only possible if min_dates or max_dates is specified)
impute_dtc_dtm(
  c("2020-12", NA_character_),
  min_dates = list(
    ymd_hm("2020-12-06T12:12", "2020-01-01T01:01"),
    ymd_hm("2020-11-11T11:11", NA)
  ),
  highest_imputation = "Y"
)
```

`list_all_templates` *List All Available ADaM Templates*

Description

List All Available ADaM Templates

Usage

```
list_all_templates(package = "admiral")
```

Arguments

package	The R package in which to look for templates. By default `"admiral"`.
	Default value `"admiral"`

Details

The function only lists the `.R` templates.

Value

A `character` vector of all available templates

See Also

Utilities used for examples and template scripts: use_ad_template()

Examples

```
list_all_templates()
```

`list_tte_source_objects`

List all `tte_source` *Objects Available in a Package*

Description

List all `tte_source` Objects Available in a Package

Usage

```
list_tte_source_objects(package = "admiral")
```

Arguments

package	The name of the package in which to search for `tte_source` objects **Default value** `"admiral"`

Value

A `data.frame` where each row corresponds to one `tte_source` object or `NULL` if `package` does not contain any `tte_source` objects

See Also

Other Advanced Functions: params()

Examples

```
list_tte_source_objects()
```

max_cond *Maximum Value on a Subset*

Description

The function derives the maximum value of a vector/column on a subset of entries/observations.

Usage

```
max_cond(var, cond)
```

Arguments

var	A vector
	Default value none
cond	A condition
	Default value none

See Also

Utilities for Filtering Observations: count_vals(), filter_exist(), filter_extreme(), filter_joined(), filter_not_exist(), filter_relative(), min_cond()

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(admiral)
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "NE",
  "1",      4,        "CR",
  "1",      5,        "NE",
  "2",      1,        "CR",
  "2",      2,        "PR",
  "2",      3,        "CR",
)

# In oncology setting, when needing to check the first time a patient had
# a Complete Response (CR) to compare to see if any Partial Response (PR)
# occurred after this add variable indicating if PR occurred after CR
group_by(data, USUBJID) %>% mutate(
  first_cr_vis = min_cond(var = AVISITN, cond = AVALC == "CR"),
  last_pr_vis = max_cond(var = AVISITN, cond = AVALC == "PR"),
  pr_after_cr = last_pr_vis > first_cr_vis
)
```

`min_cond` *Minimum Value on a Subset*

Description

The function derives the minimum value of a vector/column on a subset of entries/observations.

Usage

```
min_cond(var, cond)
```

Arguments

`var`	A vector
	Default value none
`cond`	A condition
	Default value none

See Also

Utilities for Filtering Observations: `count_vals()`, `filter_exist()`, `filter_extreme()`, `filter_joined()`, `filter_not_exist()`, `filter_relative()`, `max_cond()`

Examples

```
library(tibble)
library(dplyr, warn.conflicts = FALSE)
library(admiral)
data <- tribble(
  ~USUBJID, ~AVISITN, ~AVALC,
  "1",      1,        "PR",
  "1",      2,        "CR",
  "1",      3,        "NE",
  "1",      4,        "CR",
  "1",      5,        "NE",
  "2",      1,        "CR",
  "2",      2,        "PR",
  "2",      3,        "CR",
)

# In oncology setting, when needing to check the first time a patient had
# a Complete Response (CR) to compare to see if any Partial Response (PR)
# occurred after this add variable indicating if PR occurred after CR
group_by(data, USUBJID) %>% mutate(
  first_cr_vis = min_cond(var = AVISITN, cond = AVALC == "CR"),
  last_pr_vis = max_cond(var = AVISITN, cond = AVALC == "PR"),
  pr_after_cr = last_pr_vis > first_cr_vis
)
```

negate_vars *Negate List of Variables*

Description

The function adds a minus sign as prefix to each variable.

Usage

```
negate_vars(vars = NULL)
```

Arguments

vars	List of variables created by `exprs()`
	Default value `NULL`

Details

This is useful if a list of variables should be removed from a dataset, e.g., `select(!!!negate_vars(by_vars))` removes all by variables.

Value

A list of expressions

See Also

Utilities for working with quosures/list of expressions: `chr2vars()`

Examples

```
negate_vars(exprs(USUBJID, STUDYID))
```

params *Create a Set of Parameters*

Description

Create a set of variable parameters/function arguments to be used in `call_derivation()`.

Usage

```
params(...)
```

Arguments

...	One or more named arguments
	Default value none

Value

An object of class `params`

See Also

`call_derivation()`

Other Advanced Functions: `list_tte_source_objects()`

Examples

```
library(dplyr, warn.conflicts = FALSE)

adsl <- tribble(
  ~STUDYID,   ~USUBJID,      ~TRTSDT,      ~TRTEDT,
  "PILOT01", "01-1307",           NA,           NA,
  "PILOT01", "05-1377", "2014-01-04", "2014-01-25",
  "PILOT01", "06-1384", "2012-09-15", "2012-09-24",
  "PILOT01", "15-1085", "2013-02-16", "2013-08-18",
  "PILOT01", "16-1298", "2013-04-08", "2013-06-28"
) %>%
  mutate(
    across(TRTSDT:TRTEDT, as.Date)
  )

ae <- tribble(
  ~STUDYID,  ~DOMAIN,  ~USUBJID,     ~AESTDTC,     ~AEENDTC,
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-15", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "06-1384", "2012-09-23", "2012-09-29",
  "PILOT01",    "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-06-08", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06",
  "PILOT01",    "AE", "16-1298", "2013-04-22", "2013-07-06"
)

adae <- ae %>%
  select(USUBJID, AESTDTC, AEENDTC) %>%
```

```
  derive_vars_merged(
    dataset_add = adsl,
    new_vars = exprs(TRTSDT, TRTEDT),
    by_vars = exprs(USUBJID)
  )

## In order to derive both `ASTDT` and `AENDT` in `ADAE`, one can use `derive_vars_dt()`
adae %>%
  derive_vars_dt(
    new_vars_prefix = "AST",
    dtc = AESTDTC,
    date_imputation = "first",
    min_dates = exprs(TRTSDT),
    max_dates = exprs(TRTEDT)
  ) %>%
  derive_vars_dt(
    new_vars_prefix = "AEN",
    dtc = AEENDTC,
    date_imputation = "last",
    min_dates = exprs(TRTSDT),
    max_dates = exprs(TRTEDT)
  )


## While `derive_vars_dt()` can only add one variable at a time, using `call_derivation()`
## one can add multiple variables in one go.
## The function arguments which are different from a variable to another (e.g. `new_vars_prefix`,
## `dtc`, and `date_imputation`) are specified as a list of `params()` in the `variable_params`
## argument of `call_derivation()`. All other arguments which are common to all variables
## (e.g. `min_dates` and `max_dates`) are specified outside of `variable_params` (i.e. in `...`).
call_derivation(
  dataset = adae,
  derivation = derive_vars_dt,
  variable_params = list(
    params(dtc = AESTDTC, date_imputation = "first", new_vars_prefix = "AST"),
    params(dtc = AEENDTC, date_imputation = "last", new_vars_prefix = "AEN")
  ),
  min_dates = exprs(TRTSDT),
  max_dates = exprs(TRTEDT)
)

## The above call using `call_derivation()` is equivalent to the call using `derive_vars_dt()`
## to derive variables `ASTDT` and `AENDT` separately at the beginning.
```

---

queries *Queries Dataset*

Description

Queries Dataset

Usage

```
queries
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 15 rows and 8 columns.

Source

An example of standard query dataset to be used in deriving Standardized MedDRA Query variables in ADAE

See Also

Other datasets: admiral_adlb, admiral_adsl, example_qs, queries_mh

---

`queries_mh` *Queries MH Dataset*

Description

Queries MH Dataset

Usage

```
queries_mh
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 14 rows and 8 columns.

Source

An example of standard query MH dataset to be used in deriving Standardized MedDRA Query variables in ADMH

See Also

Other datasets: admiral_adlb, admiral_adsl, example_qs, queries

query *Create an* query *object*

Description

A query object defines a query, e.g., a Standard MedDRA Query (SMQ), a Standardized Drug Grouping (SDG), or a customized query (CQ). It is used as input to create_query_data().

Usage

```
query(prefix, name = auto, id = NULL, add_scope_num = FALSE, definition = NULL)
```

Arguments

prefix	The value is used to populate PREFIX in the output dataset of create_query_data(), e.g., "SMQ03" **Default value** none
name	The value is used to populate GRPNAME in the output dataset of create_query_data(). If the auto keyword is specified, the variable is set to the name of the query in the SMQ/SDG database. **Permitted values** A character scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an basket_select() object. **Default value** auto
id	The value is used to populate GRPID in the output dataset of create_query_data(). If the auto keyword is specified, the variable is set to the id of the query in the SMQ/SDG database. **Permitted values** A integer scalar or the auto keyword. The auto keyword is permitted only for queries which are defined by an basket_select() object. **Default value** NULL
add_scope_num	Determines if SCOPEN in the output dataset of create_query_data() is populated If the parameter is set to TRUE, the definition must be an basket_select() object. *Default*: FALSE **Permitted values** TRUE, FALSE **Default value** FALSE
definition	Definition of terms belonging to the query There are three different ways to define the terms: • An basket_select() object is specified to select a query from the SMQ database.

- A data frame with columns SRCVAR and TERMCHAR or TERMNUM can be specified to define the terms of a customized query. The SRCVAR should be set to the name of the variable which should be used to select the terms, e.g., "AEDECOD" or "AELLTCD". SRCVAR does not need to be constant within a query. For example a query can be based on AEDECOD and AELLT.
  If SRCVAR refers to a character variable, TERMCHAR should be set to the value the variable. If it refers to a numeric variable, TERMNUM should be set to the value of the variable. If only character variables or only numeric variables are used, TERMNUM or TERMCHAR respectively can be omitted.
- A list of data frames and basket_select() objects can be specified to define a customized query based on custom terms and SMQs. The data frames must have the same structure as described for the previous item.

Permitted values an basket_select() object, a data frame, or a list of data frames and basket_select() objects.

Default value NULL

Value

An object of class query.

See Also

create_query_data(), basket_select(), vignette("queries_dataset")

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), event_source(), flag_event(), records_source(), tte_source()

Examples

```
# create a query for an SMQ
library(tibble)
library(dplyr, warn.conflicts = FALSE)

# create a query for a SMQ
query(
  prefix = "SMQ02",
  id = auto,
  definition = basket_select(
    name = "Pregnancy and neonatal topics (SMQ)",
    scope = "NARROW",
    type = "smq"
  )
)

# create a query for an SDG
query(
  prefix = "SDG01",
  id = auto,
  definition = basket_select(
    name = "5-aminosalicylates for ulcerative colitis",
    scope = NA_character_,
```

```
    type = "sdg"
  )
)

# creating a query for a customized query
cqterms <- tribble(
  ~TERMCHAR, ~TERMNUM,
  "APPLICATION SITE ERYTHEMA", 10003041L,
  "APPLICATION SITE PRURITUS", 10003053L
) %>%
  mutate(SRCVAR = "AEDECOD")

query(
  prefix = "CQ01",
  name = "Application Site Issues",
  definition = cqterms
)

# creating a customized query based on SMQs and additional terms
query(
  prefix = "CQ03",
  name = "Special issues of interest",
  definition = list(
    cqterms,
    basket_select(
      name = "Pregnancy and neonatal topics (SMQ)",
      scope = "NARROW",
      type = "smq"
    ),
    basket_select(
      id = 8050L,
      scope = "BROAD",
      type = "smq"
    )
  )
)
```

`records_source` *Create a* `records_source` *Object*

Description

The `records_source` object is used to find extreme records of interest.

Usage

```
records_source(dataset_name, filter = NULL, new_vars)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets argument of derive_param_extreme_record(). **Default value** none
filter	An unquoted condition for selecting the observations from dataset. **Default value** NULL
new_vars	Variables to add The specified variables from the source datasets are added to the output dataset. Variables can be renamed by naming the element, i.e., new_vars = exprs(<new name> = <old name>). For example new_vars = exprs(var1, var2) adds variables var1 and var2 from to the input dataset. And new_vars = exprs(var1, new_var2 = old_var2) takes var1 and old_var2 from the source dataset and adds them to the input dataset renaming old_var2 to new_var2. Expressions can be used to create new variables (see for example new_vars argument in derive_vars_merged()). **Permitted values** list of expressions created by exprs(), e.g., exprs(ADT, desc(AVAL)) **Default value** none

Value

An object of class records_source

See Also

derive_param_extreme_record()

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), event_source(), flag_event(), query(), tte_source()

---

restrict_derivation *Execute a Derivation on a Subset of the Input Dataset*

Description

Execute a derivation on a subset of the input dataset.

Usage

```
restrict_derivation(dataset, derivation, args = NULL, filter)
```

Arguments

dataset	Input dataset **Default value** none
derivation	Derivation A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. All expected arguments for the derivation function must be provided through the params() objects passed to the args argument. **Default value** none
args	Arguments of the derivation A params() object is expected. **Default value** NULL
filter	Filter condition **Default value** none

Details

It is also possible to pass functions from outside the {admiral} package to restrict_derivation(), e.g. an extension package function, or dplyr::mutate(). The only requirement for a function being passed to derivation is that it must take a dataset as its first argument and return a dataset.

See Also

params() slice_derivation() call_derivation()

Higher Order Functions: call_derivation(), derivation_slice(), slice_derivation()

Examples

```
library(tibble)

adlb <- tribble(
  ~USUBJID, ~AVISITN, ~AVAL, ~ABLFL,
  "1",            -1,   113, NA_character_,
  "1",             0,   113, "Y",
  "1",             3,   117, NA_character_,
  "2",             0,    95, "Y",
  "3",             0,   111, "Y",
  "3",             1,   101, NA_character_,
  "3",             2,   123, NA_character_
)

# Derive BASE for post-baseline records only (derive_var_base() can not be used in this case
# as it requires the baseline observation to be in the input dataset)
restrict_derivation(
  adlb,
  derivation = derive_vars_merged,
```

```
  args = params(
    by_vars = exprs(USUBJID),
    dataset_add = adlb,
    filter_add = ABLFL == "Y",
    new_vars = exprs(BASE = AVAL)
  ),
  filter = AVISITN > 0
)

# Derive BASE for baseline and post-baseline records only
restrict_derivation(
  adlb,
  derivation = derive_var_base,
  args = params(
    by_vars = exprs(USUBJID)
  ),
  filter = AVISITN >= 0
) %>%
  # Derive CHG for post-baseline records only
  restrict_derivation(
    derivation = derive_var_chg,
    filter = AVISITN > 0
  )
```

`set_admiral_options` *Set the Value of admiral Options*

Description

Set the values of admiral options that can be modified for advanced users.

Usage

```
set_admiral_options(subject_keys, signif_digits, save_memory)
```

Arguments

`subject_keys`	Variables to uniquely identify a subject, defaults to `exprs(STUDYID, USUBJID)`. This option is used as default value for the `subject_keys` argument in all admiral functions. **Default value** none
`signif_digits`	Holds number of significant digits when comparing to numeric variables, defaults to `15`. This option is used as default value for the `signif_dig` argument in admiral functions `derive_var_atoxgr_dir()` and `derive_var_anrind()`. **Default value** none
`save_memory`	If set to `TRUE`, an alternative algorithm is used in the functions `derive_vars_joined()`, `derive_var_joined_exist_flag()`, `derive_extreme_event()`, and `filter_joined()` which requires less memory but more run-time. **Default value** none

Details

Modify an admiral option, e.g `subject_keys`, such that it automatically affects downstream function inputs where `get_admiral_option()` is called such as `derive_param_exist_flag()`.

Value

No return value, called for side effects.

See Also

`get_admiral_option()`, `derive_param_exist_flag()`,`derive_param_tte()`, `derive_var_dthcaus()`, `derive_var_extreme_dtm()`, `derive_vars_period()`, `create_period_dataset()`, `derive_var_atoxgr_dir()`, `derive_var_anrind()`

Other admiral_options: `get_admiral_option()`

Examples

```
library(lubridate)
library(dplyr, warn.conflicts = FALSE)
library(tibble)
set_admiral_options(subject_keys = exprs(STUDYID, USUBJID2))

# Derive a new parameter for measurable disease at baseline
adsl <- tribble(
  ~USUBJID2,
  "1",
  "2",
  "3"
) %>%
  mutate(STUDYID = "XX1234")

tu <- tribble(
  ~USUBJID2,      ~VISIT,    ~TUSTRESC,
  "1",        "SCREENING",     "TARGET",
  "1",           "WEEK 1",     "TARGET",
  "1",           "WEEK 5",     "TARGET",
  "1",           "WEEK 9", "NON-TARGET",
  "2",        "SCREENING", "NON-TARGET",
  "2",        "SCREENING", "NON-TARGET"
) %>%
  mutate(
    STUDYID = "XX1234",
    TUTESTCD = "TUMIDENT"
  )

derive_param_exist_flag(
  dataset_ref = adsl,
  dataset_add = tu,
  filter_add = TUTESTCD == "TUMIDENT" & VISIT == "SCREENING",
  condition = TUSTRESC == "TARGET",
  false_value = "N",
```

```
  missing_value = "N",
  set_values_to = exprs(
    PARAMCD = "MDIS",
    PARAM = "Measurable Disease at Baseline"
  )
)

set_admiral_options(signif_digits = 14)

# Derive ANRIND for ADVS
advs <- tribble(
  ~PARAMCD, ~AVAL, ~ANRLO, ~ANRHI,
  "DIABP",     59,     60,     80,
  "SYSBP",    120,     90,    130,
  "RESP",      21,      8,     20,
)

derive_var_anrind(advs)
```

`slice_derivation` *Execute a Derivation with Different Arguments for Subsets of the Input Dataset*

Description

The input dataset is split into slices (subsets) and for each slice the derivation is called separately. Some or all arguments of the derivation may vary depending on the slice.

Usage

```
slice_derivation(dataset, derivation, ..., args = NULL)
```

Arguments

`dataset`	Input dataset **Default value** none
`derivation`	Derivation A function that performs a specific derivation is expected. A derivation adds variables or observations to a dataset. The first argument of a derivation must expect a dataset and the derivation must return a dataset. All expected arguments for the derivation function must be provided through the `params()` object passed to the `args` argument or be provided in *every* `derivation_slice()`. **Default value** none
`...`	A `derivation_slice()` object is expected Each slice defines a subset of the input dataset and some of the parameters for the derivation. The derivation is called on the subset with the parameters specified

by the args parameter and the args field of the derivation_slice() object. If a parameter is specified for both, the value in derivation_slice() overwrites the one in args.

Default value none

args — Arguments of the derivation

A params() object is expected.

Default value NULL

Details

For each slice the derivation is called on the subset defined by the filter field of the derivation_slice() object and with the parameters specified by the args parameter and the args field of the derivation_slice() object. If a parameter is specified for both, the value in derivation_slice() overwrites the one in args.

- Observations that match with more than one slice are only considered for the first matching slice.
- The derivation is called for slices with no observations.
- Observations with no match to any of the slices are included in the output dataset but the derivation is not called for them.

It is also possible to pass functions from outside the {admiral} package to slice_derivation(), e.g. an extension package function, or dplyr::mutate(). The only requirement for a function being passed to derivation is that it must take a dataset as its first argument and return a dataset.

Value

The input dataset with the variables derived by the derivation added

See Also

params() restrict_derivation() call_derivation()

Higher Order Functions: call_derivation(), derivation_slice(), restrict_derivation()

Examples

```
library(tibble)
library(stringr)
advs <- tribble(
  ~USUBJID, ~VSDTC,       ~VSTPT,
  "1",      "2020-04-16", NA_character_,
  "1",      "2020-04-16", "BEFORE TREATMENT"
)

# For the second slice filter is set to TRUE. Thus derive_vars_dtm is called
# with time_imputation = "last" for all observations which do not match for the
# first slice.
slice_derivation(
  advs,
```

```
  derivation = derive_vars_dtm,
  args = params(
    dtc = VSDTC,
    new_vars_prefix = "A"
  ),
  derivation_slice(
    filter = str_detect(VSTPT, "PRE|BEFORE"),
    args = params(time_imputation = "first")
  ),
  derivation_slice(
    filter = TRUE,
    args = params(time_imputation = "last")
  )
)
```

`transform_range` *Transform Range*

Description

Transforms results from the source range to the target range. For example, for transforming source values 1, 2, 3, 4, 5 to 0, 25, 50, 75, 100.

Usage

```
transform_range(
  source,
  source_range,
  target_range,
  flip_direction = FALSE,
  outside_range = "NA"
)
```

Arguments

`source`	A vector of values to be transformed A numeric vector is expected. **Default value** none
`source_range`	The permitted source range A numeric vector containing two elements is expected, representing the lower and upper bounds of the permitted source range. **Default value** none
`target_range`	The target range A numeric vector containing two elements is expected, representing the lower and upper bounds of the target range.

Default value none

`flip_direction` Flip direction of the range?

The transformed values will be reversed within the target range, e.g. within the range 0 to 100, 25 would be reversed to 75.

Permitted values `TRUE`, `FALSE`

Default value `FALSE`

`outside_range` Handling of values outside the source range

Values outside the source range (`source_range`) are transformed to `NA`.

If `"warning"` or `"error"` is specified, a warning or error is issued if `source` includes any values outside the source range.

Permitted values `"NA"`, `"warning"`, `"error"`

Default value `"NA"`

Details

Returns the values of `source` linearly transformed from the source range (`source_range`) to the target range (`target_range`). Values outside the source range are set to `NA`.

Value

The source linearly transformed to the target range

See Also

BDS-Findings Functions that returns a vector: `compute_bmi()`, `compute_bsa()`, `compute_egfr()`, `compute_framingham()`, `compute_map()`, `compute_qtc()`, `compute_qual_imputation()`, `compute_qual_imputation_c`
`compute_rr()`, `compute_scale()`

Examples

```
transform_range(
  source = c(1, 4, 3, 6, 5),
  source_range = c(1, 5),
  target_range = c(0, 100)
)

transform_range(
  source = c(1, 4, 3, 6, 5),
  source_range = c(1, 5),
  target_range = c(0, 100),
  flip_direction = TRUE
)
```

tte_source *Create a* tte_source *Object*

Description

The tte_source object is used to define events and possible censorings.

Usage

```
tte_source(
  dataset_name,
  filter = NULL,
  date,
  censor = 0,
  set_values_to = NULL,
  order = order,
  consider_end_dates = TRUE
)
```

Arguments

dataset_name	The name of the source dataset The name refers to the dataset provided by the source_datasets parameter of derive_param_tte(). **Default value** none
filter	An unquoted condition for selecting the observations from dataset which are events or possible censoring time points. **Default value** NULL
date	A variable or expression providing the date of the event or censoring. A date, or a datetime can be specified. An unquoted symbol or expression is expected. Refer to derive_vars_dt() or convert_dtc_to_dt() to impute and derive a date from a date character vector to a date object. **Default value** none
censor	Censoring value CDISC strongly recommends using 0 for events and positive integers for censoring. **Default value** 0
set_values_to	A named list returned by exprs() defining the variables to be set for the event or censoring, e.g. exprs(EVENTDESC = "DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT"). The values must be a symbol, a character string, a numeric value, an expression, or NA. **Default value** NULL

order
: Sort order

  An optional named list returned by exprs() defining additional variables that the source dataset is sorted on after date.

  Permitted values list of variables created by exprs() e.g. exprs(ASEQ).

  Default value order

consider_end_dates
: Should end dates be considered?

  If end dates are considered, the records which are after the end date are ignored and the censor value specified for the end date takes precedence.

  Permitted values TRUE, FALSE

  Default value TRUE

Value

An object of class tte_source

See Also

derive_param_tte(), censor_source(), event_source()

Source Objects: basket_select(), censor_source(), death_event, event(), event_joined(), event_source(), flag_event(), query(), records_source()

---

use_ad_template *Open an ADaM Template Script*

Description

Open an ADaM Template Script

Usage

```
use_ad_template(
  adam_name = "adsl",
  save_path = paste0("./", adam_name, ".R"),
  package = "admiral",
  overwrite = FALSE
)
```

Arguments

adam_name
: An ADaM dataset name. You can use any of the available dataset names "ADAB", "ADAE", "ADCM", "ADEG", "ADEX", "ADLB", "ADLBHY", "ADMH", "ADPC", "ADPP", "ADPPK", "ADSL", "ADVS". The dataset name is case-insensitive. The default dataset name is "ADSL".

  Default value "adsl"

save_path	Path to save the script. **Default value** paste0("./", adam_name, ".R")
package	The R package in which to look for templates. By default "admiral". **Default value** "admiral"
overwrite	Whether to overwrite an existing file named save_path. **Default value** FALSE

Details

Running without any arguments such as use_ad_template() auto-generates adsl.R in the current path. Use list_all_templates() to discover which templates are available.

Value

No return values, called for side effects

See Also

Utilities used for examples and template scripts: list_all_templates()

Examples

```
if (interactive()) {
  use_ad_template("adsl")
}
```

---

yn_to_numeric *Map "Y" and "N" to Numeric Values*

Description

Map "Y" and "N" to numeric values.

Usage

```
yn_to_numeric(arg)
```

Arguments

arg	Character vector **Default value** none

Value

1 if arg equals "Y", 0 if arg equals "N", NA_real_ otherwise

See Also

Utilities for Formatting Observations: `convert_blanks_to_na()`, `convert_na_to_blanks()`

Examples

```
yn_to_numeric(c("Y", "N", NA_character_))
```

`%>%` *Pipe operator*

Description

See `magrittr::%>%` for more details.

Usage

```
lhs %>% rhs
```

Arguments

`lhs`	A value or the magrittr placeholder.
	Default value `NULL`
`rhs`	A function call using the magrittr semantics.
	Default value `NULL`

Index

∗ **admiral_options**
get_admiral_option, 424
set_admiral_options, 456
∗ **com_bds_findings**
compute_bmi, 24
compute_bsa, 25
compute_egfr, 30
compute_framingham, 33
compute_map, 36
compute_qtc, 37
compute_qual_imputation, 38
compute_qual_imputation_dec, 39
compute_rr, 40
compute_scale, 41
transform_range, 460
∗ **com_date_time**
compute_age_years, 22
compute_dtf, 26
compute_duration, 27
compute_tmf, 42
convert_date_to_dtm, 47
convert_dtc_to_dt, 50
convert_dtc_to_dtm, 52
convert_xxtpt_to_hours, 56
impute_dtc_dt, 435
impute_dtc_dtm, 439
∗ **create_aux**
consolidate_metadata, 44
create_period_dataset, 64
create_query_data, 66
create_single_dose_dataset, 70
∗ **datasets**
admiral_adlb, 5
admiral_adsl, 6
example_qs, 403
queries, 449
queries_mh, 450
∗ **deprecated**
call_user_fun, 19
date_source, 75
derive_param_extreme_record, 144
derive_var_dthcaus, 305
derive_var_extreme_dt, 308
derive_var_extreme_dtm, 312
derive_var_merged_summary, 351
dthcaus_source, 392
get_summary_records, 430
∗ **der_adsl**
derive_var_age_years, 292
derive_vars_aage, 193
derive_vars_extreme_event, 236
derive_vars_period, 284
∗ **der_bds_findings**
derive_var_analysis_ratio, 293
derive_var_anrind, 295
derive_var_atoxgr, 297
derive_var_atoxgr_dir, 298
derive_var_base, 302
derive_var_chg, 304
derive_var_nfrlt, 353
derive_var_ontrtfl, 371
derive_var_pchg, 374
derive_var_shift, 379
derive_vars_crit_flag, 206
∗ **der_date_time**
derive_var_trtdurd, 381
derive_vars_dt, 210
derive_vars_dtm, 219
derive_vars_dtm_to_dt, 227
derive_vars_dtm_to_tm, 229
derive_vars_duration, 230
derive_vars_dy, 234
∗ **der_gen**
derive_var_extreme_flag, 317
derive_var_joined_exist_flag, 326
derive_var_merged_ef_msrc, 341
derive_var_merged_exist_flag, 348
derive_var_obs_number, 369

derive_var_relative_flag, 376
derive_var_trtdurd, 381
derive_vars_cat, 197
derive_vars_computed, 203
derive_vars_dt, 210
derive_vars_dtm, 219
derive_vars_dtm_to_dt, 227
derive_vars_dtm_to_tm, 229
derive_vars_duration, 230
derive_vars_dy, 234
derive_vars_joined, 240
derive_vars_joined_summary, 257
derive_vars_merged, 266
derive_vars_merged_lookup, 275
derive_vars_merged_summary, 278
derive_vars_transposed, 289
* **der_occds**
derive_var_trtemfl, 382
derive_vars_atc, 195
derive_vars_query, 287
* **der_prm_bds_findings**
default_qtc_paramcd, 78
derive_basetype_records, 79
derive_expected_records, 82
derive_extreme_event, 84
derive_extreme_records, 99
derive_locf_records, 113
derive_param_bmi, 118
derive_param_bsa, 122
derive_param_computed, 126
derive_param_doseint, 135
derive_param_exist_flag, 138
derive_param_exposure, 141
derive_param_framingham, 147
derive_param_map, 152
derive_param_qtc, 155
derive_param_rr, 158
derive_param_wbc_abs, 182
derive_summary_records, 184
* **der_prm_tte**
derive_param_tte, 160
* **experimental**
convert_xxtpt_to_hours, 56
derive_var_nfrlt, 353
* **high_order_function**
call_derivation, 17
derivation_slice, 79
restrict_derivation, 454
slice_derivation, 458
* **metadata**
atoxgr_criteria_ctcv4, 6
atoxgr_criteria_ctcv4_uscv, 8
atoxgr_criteria_ctcv5, 9
atoxgr_criteria_ctcv5_uscv, 10
atoxgr_criteria_ctcv6, 11
atoxgr_criteria_ctcv6_uscv, 12
atoxgr_criteria_daids, 13
atoxgr_criteria_daids_uscv, 15
country_code_lookup, 61
dose_freq_lookup, 391
* **other_advanced**
list_tte_source_objects, 444
params, 447
* **reexport**
%>%, 465
desc, 391
exprs, 403
* **source_specifications**
basket_select, 16
censor_source, 20
death_event, 77
event, 394
event_joined, 396
event_source, 401
flag_event, 423
query, 451
records_source, 453
tte_source, 462
* **utils_ds_chk**
get_duplicates_dataset, 425
get_many_to_one_dataset, 428
get_one_to_many_dataset, 429
* **utils_examples**
list_all_templates, 443
use_ad_template, 463
* **utils_fil**
count_vals, 63
filter_exist, 404
filter_extreme, 406
filter_joined, 408
filter_not_exist, 418
filter_relative, 420
max_cond, 445
min_cond, 446
* **utils_fmt**
convert_blanks_to_na, 45

convert_na_to_blanks, 55
yn_to_numeric, 464
∗ **utils_help**
extract_unit, 404
get_flagged_records, 426
get_not_mapped, 429
get_vars_query, 433
∗ **utils_quo**
chr2vars, 22
negate_vars, 447
%>%, *465*, 465

admiral_adlb, 5, *6*, *403*, *450*
admiral_adsl, *6*, 6, *403*, *450*
ae_event *(*death_event*)*, 77
ae_gr1_event *(*death_event*)*, 77
ae_gr2_event *(*death_event*)*, 77
ae_gr35_event *(*death_event*)*, 77
ae_gr3_event *(*death_event*)*, 77
ae_gr4_event *(*death_event*)*, 77
ae_gr5_event *(*death_event*)*, 77
ae_ser_event *(*death_event*)*, 77
ae_sev_event *(*death_event*)*, 77
ae_wd_event *(*death_event*)*, 77
atoxgr_criteria_ctcv4, 6, *9–14*, *16*, *62*, *392*
atoxgr_criteria_ctcv4_uscv, *7*, 8, *10–14*, *16*, *62*, *392*
atoxgr_criteria_ctcv5, *7*, *9*, 9, *11–14*, *16*, *62*, *392*
atoxgr_criteria_ctcv5_uscv, *7*, *9*, *10*, 10, *12–14*, *16*, *62*, *392*
atoxgr_criteria_ctcv6, *7*, *9–11*, 11, *13*, *14*, *16*, *62*, *392*
atoxgr_criteria_ctcv6_uscv, *7*, *9–12*, 12, *14*, *16*, *62*, *392*
atoxgr_criteria_daids, *7*, *9–13*, 13, *16*, *62*, *392*
atoxgr_criteria_daids_uscv, *7*, *9–14*, 15, *62*, *392*

basket_select, 16
basket_select(), *21*, *68*, *77*, *395*, *399*, *402*, *424*, *452*, *454*, *463*

call_derivation, 17
call_derivation(), *18*, *79*, *447*, *448*, *455*, *459*
call_user_fun, 19
call_user_fun(), *76*, *146*, *306*, *309*, *314*, *353*, *393*, *432*
censor_source, 20
censor_source(), *17*, *77*, *181*, *395*, *399*, *402*, *424*, *452*, *454*, *463*
chr2vars, 22
chr2vars(), *447*
compute_age_years, 22
compute_age_years(), *27*, *29*, *43*, *49*, *52*, *55*, *61*, *437*, *441*
compute_bmi, 24
compute_bmi(), *26*, *32*, *35*, *36*, *38–40*, *42*, *120*, *461*
compute_bsa, 25
compute_bsa(), *24*, *32*, *35*, *36*, *38–40*, *42*, *124*, *461*
compute_dtf, 26
compute_dtf(), *23*, *29*, *43*, *49*, *52*, *55*, *61*, *437*, *441*
compute_duration, 27
compute_duration(), *23*, *27*, *43*, *49*, *52*, *55*, *61*, *232*, *437*, *441*
compute_egfr, 30
compute_egfr(), *24*, *26*, *35*, *36*, *38–40*, *42*, *461*
compute_framingham, 33
compute_framingham(), *24*, *26*, *32*, *36*, *38–40*, *42*, *150*, *461*
compute_map, 36
compute_map(), *24*, *26*, *32*, *35*, *38–40*, *42*, *154*, *461*
compute_qtc, 37
compute_qtc(), *24*, *26*, *32*, *35*, *36*, *39*, *40*, *42*, *156*, *461*
compute_qual_imputation, 38
compute_qual_imputation(), *24*, *26*, *32*, *35*, *36*, *38*, *40*, *42*, *461*
compute_qual_imputation_dec, 39
compute_qual_imputation_dec(), *24*, *26*, *32*, *35*, *36*, *38–40*, *42*, *461*
compute_rr, 40
compute_rr(), *24*, *26*, *32*, *35*, *36*, *38–40*, *42*, *159*, *461*
compute_scale, 41
compute_scale(), *24*, *26*, *32*, *35*, *36*, *38–40*, *461*
compute_tmf, 42
compute_tmf(), *23*, *27*, *29*, *49*, *52*, *55*, *61*,

437, *441*
consolidate_metadata, 44
consolidate_metadata(), *65*, *68*, *72*
convert_blanks_to_na, 45
convert_blanks_to_na(), *56*, *465*
convert_date_to_dtm, 47
convert_date_to_dtm(), *23*, *27*, *29*, *43*, *52*, *55*, *61*, *437*, *441*
convert_dtc_to_dt, 50
convert_dtc_to_dt(), *23*, *27*, *29*, *43*, *49*, *55*, *61*, *437*, *441*
convert_dtc_to_dtm, 52
convert_dtc_to_dtm(), *23*, *27*, *29*, *43*, *49*, *52*, *61*, *437*, *441*
convert_na_to_blanks, 55
convert_na_to_blanks(), *46*, *465*
convert_xxtpt_to_hours, 56
convert_xxtpt_to_hours(), *23*, *27*, *29*, *43*, *49*, *52*, *55*, *368*, *437*, *441*
count_vals, 63
count_vals(), *405*, *407*, *418, 419*, *422*, *445, 446*
country_code_lookup, *7*, *9–14*, *16*, 61, *392*
create_period_dataset, 64
create_period_dataset(), *45*, *68*, *72*, *286*, *425*, *457*
create_query_data, 66
create_query_data(), *17*, *45*, *65*, *72*, *288*, *434*, *452*
create_single_dose_dataset, 70
create_single_dose_dataset(), *45*, *65*, *68*, *392*

date_source, 75
date_source(), *20*, *146*, *306*, *309*, *314*, *353*, *393*, *432*
death_event, *17*, *21*, 77, *395*, *399*, *402*, *424*, *452*, *454*, *463*
default_qtc_paramcd, 78
default_qtc_paramcd(), *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derivation_slice, 79
derivation_slice(), *18*, *455*, *459*
derive_basetype_records, 79
derive_basetype_records(), *78*, *83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_expected_records, 82
derive_expected_records(), *78*, *82*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_extreme_event, 84
derive_extreme_event(), *78*, *82, 83*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*, *238*, *395*, *399*
derive_extreme_records, 99
derive_extreme_records(), *78*, *82, 83*, *98*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_locf_records, 113
derive_locf_records(), *78*, *82, 83*, *98*, *113*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_param_bmi, 118
derive_param_bmi(), *24*, *78*, *82, 83*, *98*, *113*, *118*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_param_bsa, 122
derive_param_bsa(), *26*, *78*, *82, 83*, *98*, *113*, *118*, *120*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_param_computed, 126
derive_param_computed(), *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_param_doseint, 135
derive_param_doseint(), *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *193*
derive_param_exist_flag, 138
derive_param_exist_flag(), *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *143*, *151*, *154*, *156*, *159*, *183*, *193*, *425*, *457*
derive_param_exposure, 141
derive_param_exposure(), *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *151*, *154*, *156*, *159*, *183*, *193*
derive_param_extreme_record, 144
derive_param_extreme_record(), *20*, *76*, *306*, *309*, *314*, *353*, *393*, *432*, *454*
derive_param_framingham, 147
derive_param_framingham(), *35*, *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *154*, *156*, *159*, *183*, *193*

derive_param_map, 152
derive_param_map(), *36*, *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *156*, *159*, *183*, *193*
derive_param_qtc, 155
derive_param_qtc(), *38*, *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *159*, *183*, *193*
derive_param_rr, 158
derive_param_rr(), *40*, *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *183*, *193*
derive_param_tte, 160
derive_param_tte(), *21*, *77*, *402*, *425*, *457*, *463*
derive_param_wbc_abs, 182
derive_param_wbc_abs(), *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *193*
derive_summary_records, 184
derive_summary_records(), *78*, *82, 83*, *98*, *113*, *118*, *120*, *124*, *135*, *137*, *140*, *143*, *151*, *154*, *156*, *159*, *183*, *284*, *353*, *432*
derive_var_age_years, 292
derive_var_age_years(), *195*, *238*, *286*
derive_var_analysis_ratio, 293
derive_var_analysis_ratio(), *210*, *296*, *298*, *301*, *303, 304*, *369*, *373*, *375*, *380*
derive_var_anrind, 295
derive_var_anrind(), *210*, *294*, *298*, *301*, *303, 304*, *369*, *373*, *375*, *380*, *457*
derive_var_atoxgr, 297
derive_var_atoxgr(), *210*, *294*, *296*, *301*, *303, 304*, *369*, *373*, *375*, *380*
derive_var_atoxgr_dir, 298
derive_var_atoxgr_dir(), *210*, *294*, *296*, *298*, *303, 304*, *369*, *373*, *375*, *380*, *457*
derive_var_base, 302
derive_var_base(), *210*, *294*, *296*, *298*, *301*, *304*, *369*, *373*, *375*, *380*
derive_var_chg, 304
derive_var_chg(), *210*, *294*, *296*, *298*, *301*, *303*, *369*, *373*, *375*, *380*
derive_var_dthcaus, 305
derive_var_dthcaus(), *20*, *76*, *146*, *309*, *314*, *353*, *393*, *425*, *432*, *457*
derive_var_extreme_dt, 308
derive_var_extreme_dt(), *20*, *76*, *146*, *306*, *314*, *353*, *393*, *432*
derive_var_extreme_dtm, 312
derive_var_extreme_dtm(), *20*, *76*, *146*, *306*, *309*, *353*, *393*, *425*, *432*, *457*
derive_var_extreme_flag, 317
derive_var_extreme_flag(), *203*, *205*, *256*, *266*, *274*, *277*, *284*, *290*, *341*, *347*, *349*, *370*, *378*
derive_var_joined_exist_flag, 326
derive_var_joined_exist_flag(), *203*, *205*, *256*, *266*, *274*, *277*, *284*, *290*, *326*, *347*, *349*, *370*, *378*
derive_var_merged_ef_msrc, 341
derive_var_merged_ef_msrc(), *203*, *205*, *256*, *266*, *274*, *277*, *284*, *290*, *326*, *341*, *349*, *370*, *378*, *424*
derive_var_merged_exist_flag, 348
derive_var_merged_exist_flag(), *203*, *205*, *256*, *266*, *274*, *277*, *284*, *290*, *326*, *341*, *347*, *370*, *378*
derive_var_merged_summary, 351
derive_var_merged_summary(), *20*, *76*, *146*, *306*, *309*, *314*, *393*, *432*
derive_var_nfrlt, 353
derive_var_nfrlt(), *210*, *294*, *296*, *298*, *301*, *303, 304*, *373*, *375*, *380*
derive_var_obs_number, 369
derive_var_obs_number(), *203*, *205*, *257*, *266*, *274*, *277*, *284*, *290*, *326*, *341*, *347*, *349*, *378*
derive_var_ontrtfl, 371
derive_var_ontrtfl(), *210*, *294*, *296*, *298*, *301*, *303, 304*, *369*, *375*, *380*
derive_var_pchg, 374
derive_var_pchg(), *210*, *294*, *296*, *298*, *301*, *303, 304*, *369*, *373*, *380*
derive_var_relative_flag, 376
derive_var_relative_flag(), *203*, *205*, *257*, *266*, *274*, *277*, *284*, *290*, *326*, *341*, *347*, *349*, *370*
derive_var_shift, 379
derive_var_shift(), *210*, *294*, *296*, *298*, *301*, *303, 304*, *369*, *373*, *375*
derive_var_trtdurd, 381
derive_var_trtdurd(), *219*, *227–229*, *232*,

235
derive_var_trtemfl, 382
derive_var_trtemfl(), *196*, *288*
derive_vars_aage, 193
derive_vars_aage(), *238*, *286*, *293*
derive_vars_atc, 195
derive_vars_atc(), *288*, *290*, *391*
derive_vars_cat, 197
derive_vars_cat(), *205*, *257*, *266*, *274*, *277*, *284*, *290*, *326*, *341*, *347*, *349*, *370*, *378*
derive_vars_computed, 203
derive_vars_computed(), *203*, *257*, *266*, *274*, *277*, *284*, *290*, *326*, *341*, *347*, *349*, *370*, *378*
derive_vars_crit_flag, 206
derive_vars_crit_flag(), *294*, *296*, *298*, *301*, *303, 304*, *369*, *373*, *375*, *380*
derive_vars_dt, 210
derive_vars_dt(), *227–229*, *232*, *235*, *382*
derive_vars_dtm, 219
derive_vars_dtm(), *219*, *228, 229*, *232*, *235*, *382*
derive_vars_dtm_to_dt, 227
derive_vars_dtm_to_dt(), *219*, *227*, *229*, *232*, *235*, *382*
derive_vars_dtm_to_tm, 229
derive_vars_dtm_to_tm(), *219*, *227, 228*, *232*, *235*, *382*
derive_vars_duration, 230
derive_vars_duration(), *29*, *195*, *219*, *227–229*, *235*, *293*, *368*, *382*
derive_vars_dy, 234
derive_vars_dy(), *219*, *227–229*, *232*, *382*
derive_vars_extreme_event, 236
derive_vars_extreme_event(), *98*, *195*, *286*, *293*, *395*, *399*
derive_vars_joined, 240
derive_vars_joined(), *203*, *205*, *261*, *266*, *274*, *277*, *284*, *290*, *326*, *341*, *347*, *349*, *370*, *378*
derive_vars_joined_summary, 257
derive_vars_joined_summary(), *203*, *205*, *257*, *274*, *277*, *284*, *290*, *326*, *341*, *347*, *349*, *370*, *378*
derive_vars_merged, 266
derive_vars_merged(), *203*, *205*, *257*, *266*, *277*, *284*, *290*, *309*, *314*, *326*, *341*, *347*, *349*, *370*, *378*
derive_vars_merged_lookup, 275
derive_vars_merged_lookup(), *203*, *205*, *257*, *266*, *274*, *284*, *290*, *326*, *341*, *347*, *349*, *370*, *378*
derive_vars_merged_summary, 278
derive_vars_merged_summary(), *193*, *203*, *205*, *257*, *266*, *274*, *277*, *290*, *326*, *341*, *347*, *349*, *370*, *378*, *432*
derive_vars_period, 284
derive_vars_period(), *65*, *195*, *238*, *293*, *425*, *457*
derive_vars_query, 287
derive_vars_query(), *68*, *196*, *391*
derive_vars_transposed, 289
derive_vars_transposed(), *196*, *203*, *205*, *257*, *266*, *274*, *277*, *284*, *326*, *341*, *347*, *349*, *370*, *378*
desc, *391*, 391
dose_freq_lookup, *7*, *9–14*, *16*, *62*, 391
dthcaus_source, 392
dthcaus_source(), *20*, *76*, *146*, *305, 306*, *309*, *314*, *353*, *432*

event, 394
event(), *17*, *21*, *77*, *98*, *238*, *399*, *402*, *424*, *452*, *454*, *463*
event_joined, 396
event_joined(), *17*, *21*, *77*, *98*, *238*, *395*, *402*, *424*, *452*, *454*, *463*
event_source, 401
event_source(), *17*, *21*, *77*, *181*, *395*, *399*, *424*, *452*, *454*, *463*
example_qs, *6*, 403, *450*
exprs, *403*, 403
exprs(), *22*, *185*
extract_unit, 404
extract_unit(), *427*, *429*, *434*

filter_exist, 404
filter_exist(), *63*, *407*, *418, 419*, *422*, *445, 446*
filter_extreme, 406
filter_extreme(), *63*, *405*, *418, 419*, *422*, *445, 446*
filter_joined, 408
filter_joined(), *63*, *256*, *266*, *341*, *405*, *407*, *419*, *422*, *445, 446*
filter_not_exist, 418

filter_not_exist(), *63*, *405*, *407*, *418*, *422*, *445*, *446*
filter_relative, 420
filter_relative(), *63*, *405*, *407*, *418*, *419*, *445*, *446*
flag_event, 423
flag_event(), *17*, *21*, *77*, *347*, *395*, *399*, *402*, *452*, *454*, *463*

get_admiral_option, 424
get_admiral_option(), *457*
get_duplicates_dataset, 425
get_duplicates_dataset(), *429*, *430*
get_flagged_records, 426
get_flagged_records(), *404*, *429*, *434*
get_many_to_one_dataset, 428
get_many_to_one_dataset(), *426*, *430*
get_not_mapped, 429
get_not_mapped(), *404*, *427*, *434*
get_one_to_many_dataset, 429
get_one_to_many_dataset(), *426*, *429*
get_summary_records, 430
get_summary_records(), *20*, *76*, *146*, *284*, *306*, *309*, *314*, *353*, *393*
get_vars_query, 433
get_vars_query(), *404*, *427*, *429*

here, *71*

impute_dtc_dt, 435
impute_dtc_dt(), *23*, *27*, *29*, *43*, *49*, *52*, *55*, *61*, *441*
impute_dtc_dtm, 439
impute_dtc_dtm(), *23*, *27*, *29*, *43*, *49*, *52*, *55*, *61*, *437*

lastalive_censor *(*death_event*)*, 77
list_all_templates, 443
list_all_templates(), *464*
list_tte_source_objects, 444
list_tte_source_objects(), *448*

max_cond, 445
max_cond(), *63*, *405*, *407*, *418*, *419*, *422*, *446*
min_cond, 446
min_cond(), *63*, *405*, *407*, *418*, *419*, *422*, *445*

negate_vars, 447
negate_vars(), *22*

params, 447
params(), *17*, *18*, *79*, *444*, *455*, *459*

queries, *6*, *403*, 449, *450*
queries_mh, *6*, *403*, *450*, 450
query, 451
query(), *17*, *21*, *68*, *77*, *395*, *399*, *402*, *424*, *454*, *463*

records_source, 453
records_source(), *17*, *21*, *77*, *395*, *399*, *402*, *424*, *452*, *463*
restrict_derivation, 454
restrict_derivation(), *18*, *79*, *459*

set_admiral_options, 456
set_admiral_options(), *425*
slice_derivation, 458
slice_derivation(), *18*, *79*, *455*

transform_range, 460
transform_range(), *24*, *26*, *32*, *35*, *36*, *38–40*, *42*
tte_source, 462
tte_source(), *17*, *21*, *77*, *395*, *399*, *402*, *424*, *452*, *454*

use_ad_template, 463
use_ad_template(), *444*

yn_to_numeric, 464
yn_to_numeric(), *46*, *56*