Package 'projoint'

May 9, 2026

Type Package

Title Conjoint Analysis with Reliability Correction and Visualization

Version 1.1.1

Date 2026-02-22

Maintainer Yusaku Horiuchi <yusaku.horiuchi@gmail.com>

Author Yusaku Horiuchi [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0295-4089>),
Aaron Kaufman [aut] (ORCID: <https://orcid.org/0000-0003-3688-0428>),
Gary King [aut] (ORCID: <https://orcid.org/0000-0002-5327-7631>)

Description Provides tools for analyzing data generated from conjoint survey experiments, a method widely used in the social sciences for studying multidimensional preferences. The package implements estimation of marginal means (MMs) and average marginal component effects (AMCEs), with corrections for measurement error. Methods include profile-level and choice-level estimators, bias correction using intra-respondent reliability (IRR), and visualization utilities. For details on the methodology, see Clayton, Horiuchi, Kaufman, King, and Komisarchik (2025) <https://gking.harvard.edu/conjointE>.

Imports dplyr (>= 1.1.2), readr, rlang, tidyr, stringr, tidyselect, estimatr, ggthemes, ggplot2, stats, tibble, methods, MASS, forcats, scales

Suggests tidyverse, knitr, rmarkdown, downloadthis, patchwork, testthat (>= 3.0.0), DiagrammeR

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

URL https://yhoriuchi.github.io/projoint/

Depends R (>= 4.1.0)

LazyData false

BugReports https://github.com/yhoriuchi/projoint/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2026-02-22 23:00:02 UTC

Contents

`exampleData1` *Projoint Example Data Set 1: Building Conjoint with a Repeated, Flipped Task*

Description

A cleaned Qualtrics export from a conjoint study that compares two potential new building developments. Each respondent completed 8 standard tasks as well as a repeated version of the first task (flipped), which can be used to calculate intra-respondent reliability (response instability).

Usage

```
data(exampleData1)
```

Format

A data frame with 400 rows and 185 columns. Contains survey responses including demographic information, outcome choices, and conjoint attribute values identified by `K-*-*` variable names.

Examples

```
# Load the dataset
data(exampleData1)

# Inspect the first few rows
head(exampleData1)

# Number of rows and columns
dim(exampleData1)

# Display column names (truncated here)
names(exampleData1)[1:10]
```

`exampleData1_labelled_tibble`

Projoint Example Data Set 1: "Labelled Tibble"

Description

A cleaned tibble where each attribute corresponds to a separate column with a descriptive attribute name. The unit of observation is each of two profiles in each task for each respondent.

Usage

```
data(exampleData1_labelled_tibble)
```

Format

A tibble with 6,400 rows and 14 columns. Contains survey responses including outcome choices and conjoint attribute values (columns typically named with a 'K-*-*' convention).

Details

This dataset is intended for illustrating reading, reshaping, and analysis workflows in **projoint**. Column names are compatible with `reshape_projoint()`.

Source

Qualtrics survey on Prolific; see Clayton et al. replication materials.

Examples

```
# Load the data
data(exampleData1_labelled_tibble)

# Basic inspection (fast and always runnable)
head(exampleData1_labelled_tibble)
dim(exampleData1_labelled_tibble)

# Optional: quick structure peek (names only)
names(exampleData1_labelled_tibble)
```

---

exampleData2 *Projoint Example Data Set 2: Building Conjoint with a Repeated, Non-Flipped Task*

Description

A cleaned Qualtrics export from a conjoint study that compares two potential new building developments. Each respondent completed 8 standard tasks as well as a repeated version of the first task, where the profiles were presented in the same left-right positions (not flipped).

Usage

```
data(exampleData2)
```

Format

A data frame with 400 rows and 185 columns. Contains survey responses including demographic information, outcome choices, and conjoint attribute values identified by `K-*-*` variable names.

Examples

```
# Load the dataset
data(exampleData2)

# Inspect the first few rows
head(exampleData2)

# Number of rows and columns
dim(exampleData2)

# Display first 10 column names
names(exampleData2)[1:10]
```

exampleData3 *Projoint Example Data Set 3: Building Conjoint with No Repeated Task*

Description

A cleaned Qualtrics export from a conjoint study that compares two potential new building developments. Each respondent completed 8 standard tasks only; no repeated tasks are included in this dataset.

Usage

```
data(exampleData3)
```

Format

A data frame with 400 rows and 184 columns. Contains survey responses including demographic information, outcome choices, and conjoint attribute values identified by `K-*-*` variable names.

Examples

```
# Load the dataset
data(exampleData3)

# Inspect the first few rows
head(exampleData3)

# Number of rows and columns
dim(exampleData3)

# Display first 10 column names
names(exampleData3)[1:10]
```

make_projoint_data *Make a* `projoint_data` *object from a labelled tibble*

Description

Converts a labelled tibble/data frame (one column per attribute) into an object of class `projoint_data` that downstream functions (e.g., `projoint`) can consume. The unit of observation should be each of two profiles in each task for each respondent.

Usage

```
make_projoint_data(
  .dataframe,
  .attribute_vars,
  .id_var = "id",
  .task_var = "task",
  .profile_var = "profile",
  .selected_var = "selected",
  .selected_repeated_var = NULL,
  .fill = FALSE
)
```

Arguments

.dataframe	A data frame (or tibble). One row per profile per task per respondent.
.attribute_vars	Character vector of attribute column names.
.id_var	Column name (character) with respondent IDs. Default "id".
.task_var	Column name (character) with task numbers. Default "task".
.profile_var	Column name (character) with profile numbers. Default "profile".
.selected_var	Column name (character) with the binary choice for each task (values in {0,1}). Default "selected".
.selected_repeated_var	Optional column name (character) with the binary choice for the repeated task. Default NULL.
.fill	Logical. If TRUE, uses repeated-task agreement to *fill* missing agreement values for non-repeated tasks (assumes IRR is independent of table content). If unsure, prefer the default FALSE.

Value

A projoint_data object (a list-like object containing a labels tibble and a data tibble) ready to pass to projoint and related functions.

Examples

```
# Example: build a projoint_data object from the labelled-tibble example
data(exampleData1_labelled_tibble)

att_cols <- c("School Quality", "Violent Crime Rate (Vs National Rate)",
              "Racial Composition", "Housing Cost",
              "Presidential Vote (2020)",
              "Total Daily Driving Time for Commuting and Errands",
              "Type of Place")

pj_dat <- make_projoint_data(
  .dataframe            = exampleData1_labelled_tibble,
  .attribute_vars       = att_cols,
```

```
  .id_var                = "id",
  .task_var              = "task",
  .profile_var           = "profile",
  .selected_var          = "selected",
  .selected_repeated_var = "selected_repeated",
  .fill                  = FALSE
)

class(pj_dat)
# [1] "projoint_data"
```

`out1_arranged` *Example Output: Manually Rearranged Labels*

Description

A toy `projoint_data` object showing what happens after labels have been manually rearranged (e.g., via `save_labels` / `read_labels`).

Usage

```
data(out1_arranged)
```

Format

An object of class `projoint_data`. Contains two elements:

- `$labels`: reordered attribute–level mapping
- `$data`: the corresponding profile-level dataset

Details

Useful for illustrating workflows without requiring users to re-run the reordering themselves.

`plot.projoint_results` *Plot method for* `projoint_results`

Description

Creates publication-ready plots from a `projoint_results` object produced by `projoint`. Supports both profile-level and choice-level analyses, with plotting options tailored to each structure.

Usage

```
## S3 method for class 'projoint_results'
plot(
  x,
  .estimates = "corrected",
  .by_var = FALSE,
  .labels = NULL,
  .base_size = 12,
  .base_family = "",
  .type = c("bar", "pointrange"),
  .show_attribute = TRUE,
  .remove_xaxis = FALSE,
  .xlim = c(0, 1),
  .plot.margin = c(0, 3, 0, 3),
  ...
)
```

Arguments

x	A `projoint_results` object (typically from `projoint`).
.estimates	Character: which estimates to plot. One of `"corrected"`, `"uncorrected"`, or `"both"` (for profile-level), and `"corrected"` or `"uncorrected"` (for choice-level). Default `"corrected"`.
.by_var	Logical (profile-level only). Whether to plot subgroup differences. Default `FALSE`.
.labels	Character vector of length 2 (choice-level only). Custom x-axis labels for bar/pointrange plots. If `NULL`, labels are taken from `x$labels`.
.base_size	Numeric. Base font size. Default `12`.
.base_family	Character. Base font family. Default `""` (system default).
.type	Character (choice-level only). One of `"bar"` or `"pointrange"`. Default `"bar"`.
.show_attribute	Logical (choice-level only). Show the attribute name as the title when both levels belong to the same attribute. Default `TRUE`.
.remove_xaxis	Logical (choice-level only). Remove x-axis line, ticks, and labels. Default `FALSE`.
.xlim	Numeric length-2 vector (choice-level only). X-axis limits. Default `c(0, 1)`.
.plot.margin	Numeric length-4 vector (choice-level only). Plot margins in cm: `c(top, left, bottom, right)`. Default `c(0, 3, 0, 3)`.
...	Additional arguments passed to downstream plotting helpers.

Details

For profile-level results, only `.by_var`, `.base_size`, and `.base_family` are relevant. For choice-level results, only `.type`, `.labels`, `.show_attribute`, `.remove_xaxis`, `.xlim`, and `.plot.margin` are relevant. Irrelevant arguments are ignored with a warning.

Value

A ggplot2 object.

See Also

projoint, plot_projoint_profile_level, plot_projoint_choice_level_mm

Examples

```
data(exampleData1)

# Two base tasks (1 & 2) + repeated of task 1 (last)
dat <- reshape_projoint(
  exampleData1,
  .outcomes = c("choice1", "choice2", "choice1_repeated_flipped")
)

# Build a valid QOI from the labels
att <- unique(dat$labels$attribute_id)[1]
levs <- subset(dat$labels, attribute_id == att)$level_id
lev_names <- sub(".*:", "", levs)

q <- set_qoi(
  .structure     = "choice_level",
  .estimand      = "mm",
  .att_choose    = att,
  .lev_choose    = lev_names[2],
  .att_notchoose = att,
  .lev_notchoose = lev_names[1]
)

fit <- projoint(dat, .qoi = q)

# Plot method
plot(fit)
```

---

plot.projoint_tau *Plot method for* projoint_tau

Description

Visualizes the estimated intra-respondent reliability (τ) produced by the extrapolation method and stored in a projoint_tau object.

Usage

```
## S3 method for class 'projoint_tau'
plot(x, ...)
```

Arguments

x	A projoint_tau object.
...	Optional arguments (currently unused).

Value

A ggplot2 object representing the IRR (τ) visualization. The plot is drawn for its side effect and also returned (invisibly).

Examples

```
# Estimate tau, then plot:
# dat <- reshape_projoint(exampleData1, .outcomes = c("choice1","choice2"))
# tau_fit <- projoint_tau(dat)  # or predict_tau(dat)
# p <- plot(tau_fit)            # also returns the ggplot object (invisibly)
```

---

predict_tau *Estimate intra-respondent reliability (tau) without a repeated task*

Description

Uses the extrapolation method to estimate intra-respondent reliability (IRR, τ) when your conjoint design does not include an explicit repeated task. The input is a projoint_data object (e.g., produced by reshape_projoint).

Usage

```
predict_tau(.data, .title = NULL)
```

Arguments

.data	A projoint_data object (from reshape_projoint).
.title	Optional character string used as the plot title prefix.

Details

The procedure constructs pairs of base tasks within respondent, computes the proportion of identical choices as a function of how many attributes differ between the two tasks, fits a weighted regression of agreement on the number of differing attributes, and extrapolates to zero differences to obtain $\hat{\tau}$.

Value

A projoint_tau object (list-like) with components:

- $irr: a tibble with columns x (number of differing attributes) and predicted (fitted agreement), including x = 0 which is the estimate of τ.
- $figure: a ggplot2 object visualizing observed agreement by x and the fitted line with the extrapolated point at x = 0.

See Also

plot.projoint_tau, summary.projoint_tau, reshape_projoint

Examples

```
# Example workflow:
data(exampleData1)
outcomes <- c(paste0("choice", 1:8), "choice1_repeated_flipped")

# Even if your real study lacks a repeated task, this shows the API:
pj <- reshape_projoint(exampleData1, outcomes, .repeated = TRUE)

tau_fit <- predict_tau(pj, .title = "IRR (tau): ")
# Inspect the extrapolated tau (row where x == 0)
tau_fit$irr[tau_fit$irr$x == 0, ]

# Plot (also available via plot(tau_fit))
print(tau_fit$figure)
```

---

print.projoint_data *Print a projoint_data object*

Description

Custom print method for objects of class `projoint_data`.

Usage

```
## S3 method for class 'projoint_data'
print(x, ...)
```

Arguments

`x`	A `projoint_data` object.
`...`	Additional arguments (currently unused).

Value

No return value, called for its side effect of printing a summary of the `projoint_data` object to the console.

Examples

```
data(exampleData1)
dat <- reshape_projoint(
  exampleData1,
  .outcomes = c("choice1", "choice2")
)
print(dat)
```

`print.projoint_results` *Print method for projoint_results*

Description

Custom print method for objects of class `projoint_results`.

Usage

```
## S3 method for class 'projoint_results'
print(x, ...)
```

Arguments

`x`	An object of class `projoint_results`.
`...`	Additional arguments (ignored).

Value

No return value, called for its side effect of printing a summary of the `projoint_results` object to the console.

Examples

```
data(exampleData1)
dat <- reshape_projoint(
  exampleData1,
  .outcomes = c("choice1", "choice2", "choice1_repeated_flipped")
)
att <- unique(dat$labels$attribute_id)[1]
levs <- subset(dat$labels, attribute_id == att)$level_id
lev_names <- sub(".*:", "", levs)
q <- set_qoi(
  .structure     = "choice_level",
  .estimand      = "mm",
  .att_choose    = att,
  .lev_choose    = lev_names[2],
  .att_notchoose = att,
```

```
  .lev_notchoose = lev_names[1]
)
fit <- projoint(dat, .qoi = q)
print(fit)
```

`print.projoint_tau` *Print method for projoint_tau objects*

Description

Custom print method for objects of class `projoint_tau`, typically created by `projoint` or related functions.

Usage

```
## S3 method for class 'projoint_tau'
print(x, ...)
```

Arguments

`x`	An object of class `projoint_tau`.
`...`	Additional arguments (currently unused).

Value

No return value; called for its side effect of printing a summary of the estimated intra-respondent reliability (τ).

Examples

```
toy_tau <- structure(
  list(irr = data.frame(predicted = 0.413, se = 0.02, n = 200)),
  class = "projoint_tau"
)
print(toy_tau)
```

`projoint` *Analyze a conjoint dataset with measurement-error correction*

Description

Computes marginal means (MMs) or average marginal component effects (AMCEs) with correction for intra-respondent reliability (IRR, τ). When a repeated task is present, IRR is estimated unless a fixed value is supplied. Results are returned in a structured object ready for plotting and summary.

Usage

```
projoint(
  .data,
  .qoi = NULL,
  .by_var = NULL,
  .structure = "choice_level",
  .estimand = "mm",
  .se_method = "analytical",
  .irr = NULL,
  .remove_ties = TRUE,
  .ignore_position = NULL,
  .n_sims = NULL,
  .n_boot = NULL,
  .weights_1 = NULL,
  .clusters_1 = NULL,
  .se_type_1 = NULL,
  .weights_2 = NULL,
  .clusters_2 = NULL,
  .se_type_2 = NULL,
  .auto_cluster = TRUE,
  .seed = NULL
)
```

Arguments

`.data`	A `projoint_data` created by `reshape_projoint` or `make_projoint_data`.
`.qoi`	Optional `projoint_qoi` describing the quantity of interest. If supplied, its fields override `.structure` and `.estimand`.
`.by_var`	Optional column name (character) for subgroup analysis; must be logical (TRUE/FALSE) or numeric/integer coded as 0/1. Only supported for `.structure == "profile_level"` (ignored otherwise).
`.structure`	Either `"profile_level"` or `"choice_level"` (default `"choice_level"`). Overridden by `.qoi$structure` if present.
`.estimand`	Either `"mm"` (marginal mean) or `"amce"` (average marginal component effect). Default `"mm"`. Overridden by `.qoi$estimand` if present.

.se_method	Standard-error method: "analytical" (default), "simulation", or "bootstrap".
.irr	Numeric or NULL. If NULL (default), IRR is estimated (when design allows); otherwise a fixed IRR value is used and IRR estimation is skipped.
.remove_ties	Logical; remove ties in choice data before estimation? Default TRUE.
.ignore_position	Logical; choice-level only. Ignore profile position (left/right)? Default TRUE.
.n_sims	Integer; required when .se_method = "simulation".
.n_boot	Integer; required when .se_method = "bootstrap".
.weights_1, .clusters_1, .se_type_1	Passed to lm_robust when estimating IRR.
.weights_2, .clusters_2, .se_type_2	Passed to lm_robust when estimating MMs/AMCEs.
.auto_cluster	Logical. If TRUE (default), automatically cluster on an id column when present and no .clusters_* are supplied. Auto-clustering only occurs when the corresponding .se_type_* is NULL.
.seed	Optional integer. Sets a temporary RNG seed for reproducible simulation/bootstrap inside the call; restores the previous RNG state on exit.

Details

Most users will pass a projoint_data object (from reshape_projoint or make_projoint_data). Advanced users may specify custom quantities via projoint_qoi; if provided, its structure and estimand override .structure and .estimand.

Valid se_type_* values depend on clustering:

- Without clusters: "classical", "HC0", "HC1", "HC2", "HC3"
- With clusters: "CR0", "CR1", "CR2", "stata", "none"

If NULL, *estimatr* defaults are used (HC2 when unclustered; CR2 when clustered).

Value

A projoint_results object (list-like) with components such as:

- $estimand, $structure, $se_method, $irr, $tau
- $labels: attribute/level mapping used in estimation
- $estimates: a data frame of estimates with columns like att_level_choose, att_level_notchoose (if choice-level), estimate, se / std.error, conf.low, conf.high, and an estimand label such as "mm_corrected" or "amce_uncorrected".

This object is suitable for downstream use in plot.projoint_results, summary.projoint_results, and related helpers.

See Also

reshape_projoint, projoint_qoi, plot.projoint_results, summary.projoint_results

Examples

```
# Prepare example data
data(exampleData1)
outcomes <- c(paste0("choice", 1:8), "choice1_repeated_flipped")
pj <- reshape_projoint(exampleData1, outcomes)

# Choice-level QoI based on pj$labels
att <- unique(pj$labels$attribute_id)[1]
lev_ids   <- pj$labels$level_id[pj$labels$attribute_id == att]
lev_names <- sub(".*:", "", lev_ids)

q <- set_qoi(
    .structure     = "choice_level",
    .estimand      = "mm",
    .att_choose    = att,
    .lev_choose    = lev_names[2],
    .att_notchoose = att,
    .lev_notchoose = lev_names[1]
  )

# Choice-level, marginal means (fast example: fix IRR)
fit_choice <- projoint(
  pj,
  .qoi = q,
  .structure = "choice_level",
  .estimand  = "mm",
  .irr       = 0.80,        # skip IRR estimation for a quick example
  .se_method = "analytical"
)
head(summary(fit_choice))

# Profile-level AMCEs
fit_profile <- projoint(
  pj,
  .structure = "profile_level",
  .estimand  = "amce",
  .se_method = "analytical"
)
# Plot using the S3 plot method
p <- plot(fit_profile, .estimates = "both")
print(p)
```

`read_labels` *Read and apply a reordered attribute/level mapping*

Description

Reads a CSV containing a revised ordering of attributes and levels and applies it to an existing `projoint_data` object. Typical workflow: first save the current labels to CSV (e.g., with

save_labels), manually reorder rows (and/or the attribute grouping) in the CSV, then call read_labels() to apply.

Usage

```
read_labels(.data, .filename)
```

Arguments

.data	A projoint_data object whose labels/data should be reordered.
.filename	Path to the revised labels CSV (originally produced from the package's labels).

Value

A projoint_data object with the same content as .data but with attributes and levels reordered to match the CSV. The returned object contains:

- $labels: a tibble with new attribute_id and level_id reflecting the chosen order
- $data: a tibble whose att* columns have been remapped to the new level_ids

See Also

save_labels, reshape_projoint

Examples

```
# Create a projoint_data object from the example dataset
data(exampleData1)
outcomes <- c(paste0("choice", 1:8), "choice1_repeated_flipped")
pj <- reshape_projoint(exampleData1, outcomes)

# Write current labels to a temporary CSV, adding an 'order' column
tmp <- tempfile(fileext = ".csv")
pj$labels |>
  dplyr::mutate(order = dplyr::row_number()) |>
  readr::write_csv(tmp)

# (User would reorder rows in 'tmp' manually; we just read it back)
pj_reordered <- read_labels(pj, tmp)

# Inspect the updated label order
head(pj_reordered$labels)
```

`read_Qualtrics` *Read and re-format a Qualtrics CSV (choice text)*

Description

Reads a CSV file exported from Qualtrics (with "Use choice text" enabled) and returns a data frame formatted for downstream processing with `reshape_projoint`.

Usage

```
read_Qualtrics(.file)
```

Arguments

`.file`	A character string giving the path to a Qualtrics CSV file.

Value

A data frame where column names are preserved from the Qualtrics export. The first two rows of Qualtrics metadata are skipped automatically.

See Also

`reshape_projoint`

Examples

```
# Write a tiny dummy Qualtrics-style CSV to a temp file
tmp <- tempfile(fileext = ".csv")
readr::write_csv(
  data.frame(Q1 = c("Choice Text", "Choice Text", "A", "B")),
  tmp
)
# Read it back in
df <- read_Qualtrics(tmp)
head(df)
```

`reshape_projoint` *Reshape survey response data for conjoint analysis (single task set)*

Description

Takes a wide survey data frame (e.g., from `read_Qualtrics`) and reshapes it so that each row corresponds to a single respondent–task–profile. Supports arbitrary ordering of base tasks and a single repeated task per respondent. The repeated base task is inferred from the first base outcome in `.outcomes`, and the repeated outcome must be the last element of `.outcomes`.

Usage

```
reshape_projoint(
  .dataframe,
  .outcomes,
  .choice_labels = c("A", "B"),
  .alphabet = "K",
  .idvar = "ResponseId",
  .repeated = TRUE,
  .flipped = TRUE,
  .covariates = NULL,
  .fill = FALSE
)
```

Arguments

`.dataframe`	A data frame, preferably from `read_Qualtrics`.
`.outcomes`	Character vector of outcome column names in the *asked order*. If a repeated task is used, its outcome must be the *last element*.
`.choice_labels`	Character vector (default `c("A","B")`) giving the two labels that appear at the end of the outcome strings.
`.alphabet`	Single character (default `"K"`) indicating the Qualtrics prefix.
`.idvar`	Character (default `"ResponseId"`) indicating the respondent id column.
`.repeated`	Logical (default `TRUE`) indicating whether a repeated task is present.
`.flipped`	Logical (default `TRUE`) indicating whether the repeated task flips profiles before agreement is computed.
`.covariates`	Optional character vector of respondent-level covariate column names to carry through.
`.fill`	Logical (default `FALSE`). If `TRUE`, fills `agree` within respondent across tasks as described under "Filling agreement".

Details

Scope and assumptions

- One set of conjoint tasks with exactly two profiles per task (profiles 1 and 2).
- For multi-set designs, call `reshape_projoint()` once per set and bind the results.

Expected input (Qualtrics K-codes)

- Wide columns named `K-<task>-<attribute>` (attribute names) and `K-<task>-<profile>-<attribute>` (level names), with `<task>` in `1..n` and `<profile>` in `1,2`.
- Rows with missing `K-1-1` are dropped as empty tables (server hiccup safeguard).

Outcome columns (`.outcomes`)

- List all choice variables in the *order asked*. If you include a repeated task, its outcome must be the *last element*.
- For base tasks (all but the last element), the function extracts the base task id by reading the *digits* in each outcome name (e.g., `"choice4"`, `"Q4"`, `"task04"` -> task 4).
- The set of base task ids extracted from `.outcomes` must exactly match the set of task ids present in the K-codes; otherwise an error is thrown.
- The repeated base task is inferred as the digits in the *first base outcome* (i.e., the first element of `.outcomes`, excluding the final repeated outcome).

Choice parsing

- The selected profile is parsed from the *last character* of each outcome string and matched to `.choice_labels`. Ensure outcomes end with these labels (e.g., `"Candidate A"/"Candidate B"`). If outcomes are numeric or differently formatted, pre-process or adjust `.choice_labels` accordingly.

Output

- A `projoint_data` object with:
  - `$labels`: map from human-readable `attribute/level` to stable ids (`attribute_id = "att1","att2",...`, `level_id = "attX:levelY"`).
  - `$data`: tibble with one row per `id`–`task`–`profile`, attribute columns (named `att*`) storing `level_id`, `selected` (1 if that profile was chosen; 0 otherwise), `agree` (1/0/NA for repeated-task agreement after flip logic), and any `.covariates`. `id` is coerced to character; attribute columns are factors.

Filling agreement

- If `.fill = TRUE`, `agree` is filled within respondent across tasks in task order, propagating the observed repeated-task agreement to all tasks for that respondent. This assumes IRR is respondent-specific and independent of table content.

Diagnostics

- `dplyr::count(reshaped$data, task, profile)` should show exactly two rows per task.
- If `pj_estimate()` later reports "No rows match the specified attribute/level", construct QoIs from `reshaped$labels` (use the exact `attX:levelY` ids).

Value

A `projoint_data` object with elements `$labels` and `$data`; see Details.

See Also

`make_projoint_data`, `projoint`

Examples

```
# Base tasks asked in numeric order; repeated task corresponds to task 1
data(exampleData1)
outcomes <- c(paste0("choice", 1:8), "choice1_repeated_flipped")
reshaped <- reshape_projoint(exampleData1, outcomes)
dplyr::count(reshaped$data, task, profile) # should be 2 per task
```

---

save_labels *Save attribute and level labels to a CSV file*

Description

Saves the attributes and levels (and their order) from a `projoint_data` object, as generated by `reshape_projoint`, to a CSV file. This enables manual reordering and later re-import via `read_labels`.

Usage

```
save_labels(.data, .filename)
```

Arguments

`.data`	A `projoint_data` object.
`.filename`	A character string giving the name of a CSV file to be written.

Value

No return value, called for side effects (writes a CSV file).

See Also

`read_labels`, `reshape_projoint`

Examples

```
library(projoint)
data(exampleData1)
reshaped <- reshape_projoint(
  exampleData1,
  .outcomes = c(paste0("choice", 1:8), "choice1_repeated_flipped")
)
tmpfile <- tempfile(fileext = ".csv")
save_labels(reshaped, tmpfile)
readLines(tmpfile, n = 5)  # show first few lines
```

`set_qoi` *Set the quantities of interest (QoIs)*

Description

Constructs a quantities-of-interest (QoI) specification for **projoint**. Use this to request specific estimands—marginal means (MMs) or average marginal component effects (AMCEs)—at either the choice- or profile-level, and to declare which attribute levels are compared (including baselines).

Usage

```
set_qoi(
  .structure = "choice_level",
  .estimand = "mm",
  .att_choose,
  .lev_choose,
  .att_notchoose = NULL,
  .lev_notchoose = NULL,
  .att_choose_b = NULL,
  .lev_choose_b = NULL,
  .att_notchoose_b = NULL,
  .lev_notchoose_b = NULL
)
```

Arguments

`.structure`	Either `"choice_level"` (default) or `"profile_level"`.
`.estimand`	Either `"mm"` for marginal means or `"amce"` for average marginal component effects.
`.att_choose`	Character scalar: the attribute (column) for the level(s) that are *chosen*.
`.lev_choose`	Character vector: the level id(s) for the *chosen* side. Length 1 for profile-level, $\geq$1 for choice-level.
`.att_notchoose`	Character scalar: the attribute (column) for the level(s) that are *not chosen*. Only used for `.structure == "choice_level"`.

`.lev_notchoose` Character vector: the level id(s) for the *not chosen* side. Length 1 for profile-level, $\geq$1 for choice-level. Only used for `.structure == "choice_level"`.

`.att_choose_b` Character scalar: *baseline* attribute for the chosen side when computing AMCEs.

`.lev_choose_b` Character vector: *baseline* level id(s) for the chosen side when computing AMCEs. Length 1 for profile-level, $\geq$1 for choice-level.

`.att_notchoose_b` Character scalar: *baseline* attribute for the not-chosen side (choice-level only) when computing AMCEs.

`.lev_notchoose_b` Character vector: *baseline* level id(s) for the not-chosen side (choice-level only) when computing AMCEs.

Value

A `projoint_qoi` object (list-like) containing fields such as: `structure`, `estimand`, `attribute_of_interest`, `levels_of_interest`, and their baseline counterparts. This object can be supplied to downstream estimation helpers that accept a QoI spec.

Examples

```
# Specify a simple choice-level MM comparison for att1 levels:
q_mm <- set_qoi(
  .structure  = "choice_level",
  .estimand   = "mm",
  .att_choose = "att1",
  .lev_choose = c("att1:lev2"),
  .att_notchoose = "att1",
  .lev_notchoose = c("att1:lev1")
)
str(q_mm)

# Example AMCE with explicit baselines (profile-level):
q_amce <- set_qoi(
  .structure   = "profile_level",
  .estimand    = "amce",
  .att_choose  = "att2",
  .lev_choose  = "att2:lev3",
  .att_choose_b = "att2",
  .lev_choose_b = "att2:lev1"
)
str(q_amce)
```

`summary.projoint_data` *Summary for* `projoint_data`

Description

Custom summary method for objects of class `projoint_data`. Prints a brief overview of the main data and attribute-level labels contained in the object.

Usage

```
## S3 method for class 'projoint_data'
summary(object, ...)
```

Arguments

`object`	A `projoint_data` object.
`...`	Additional arguments (currently unused).

Value

No return value, called for its side effect of printing a summary of the `projoint_data` object to the console.

Examples

```
data(exampleData1)
dat <- reshape_projoint(
  exampleData1,
  .outcomes = c("choice1", "choice2")
)
summary(dat)
```

---

summary.projoint_results

Summary method for `projoint_results`

Description

Creates a concise tabular summary of a `projoint_results` object, including the chosen estimand, analysis structure, standard-error settings, and a data frame of estimates.

Usage

```
## S3 method for class 'projoint_results'
summary(object, ...)
```

Arguments

`object`	An object of class `projoint_results`.
`...`	Additional arguments (ignored).

Value

A data frame (often a tibble) summarizing the estimated effects. At minimum, it contains the columns produced in `object$estimates` (e.g., attribute/level identifiers and the point estimate with its standard error and confidence interval in columns such as `estimate`, `std.error`, `conf.low`, `conf.high`). This table is suitable for further processing or printing.

Examples

```
data(exampleData1)

# Reshape data for two base tasks + repeated (for IRR estimation)
dat <- reshape_projoint(
  exampleData1,
  .outcomes = c("choice1", "choice2", "choice1_repeated_flipped")
)

# Build a valid choice-level QoI
att <- unique(dat$labels$attribute_id)[1]
lev_ids   <- dat$labels$level_id[dat$labels$attribute_id == att]
lev_names <- sub(".*:", "", lev_ids)

q <- set_qoi(
  .structure     = "choice_level",
  .estimand      = "mm",
  .att_choose    = att,
  .lev_choose    = lev_names[2],
  .att_notchoose = att,
  .lev_notchoose = lev_names[1]
)

# Fit model
fit <- projoint(dat, .qoi = q)

# Get the tabular summary of estimates
tab <- summary(fit)
head(tab)
```

---

summary.projoint_tau *Summary method for projoint_tau objects*

Description

Custom summary method for objects of class `projoint_tau`, typically created by `projoint` or related functions. Summarizes intra-respondent reliability (IRR) estimates.

Usage

```
## S3 method for class 'projoint_tau'
summary(object, ...)
```

Arguments

`object`	An object of class `projoint_tau`.
`...`	Additional arguments (currently unused).

Value

A tibble (data frame) showing IRR estimates, typically by the number of differing attributes, as stored in `object$irr`.

Examples

```
toy_tau <- structure(
  list(irr = data.frame(predicted = 0.413, se = 0.02, n = 200)),
  class = "projoint_tau"
)
summary(toy_tau)
```

Index