Package 'insurancerating'

June 2, 2026

Type Package

Title Actuarial Tools for Insurance Pricing Models

Version 0.8.0

Author Martin Haringa [aut, cre]

Maintainer Martin Haringa <mtharinga@gmail.com>

BugReports https://github.com/MHaringa/insurancerating/issues

Description Provides actuarial tools and building blocks for analysing, modelling, refining, and validating insurance rating models. Designed to support common GLM-based pricing tasks and the translation of statistical model output into practical tariff structures. The package supports the construction of insurance tariff classes using a data-driven approach, based on the methodology of Antonio and Valdez (2012) <doi:10.1007/s10182-011-0152-7>.

License GPL (>= 2)

URL https://mharinga.github.io/insurancerating/, https://github.com/MHaringa/insurancerating

Encoding UTF-8

LazyData true

Imports ciTools, data.table, DHARMa, dplyr, evtree, fitdistrplus, ggplot2, lifecycle, lubridate, mgcv, patchwork, rlang, scales, scam, stringr

Depends R (>= 4.1.0)

Suggests classInt, ggbeeswarm, ggrepel, spelling, knitr, rmarkdown, testthat

Language en-US

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Repository CRAN

Date/Publication 2026-06-02 12:40:02 UTC

Contents

`active_rows_by_date` *Find active portfolio rows for event dates*

Description

Matches event dates, such as claim dates or portfolio snapshot dates, to the rows that were active in the portfolio on those dates.

Usage

```
active_rows_by_date(
  portfolio,
  dates,
  period_start,
  period_end,
  date,
  by = NULL,
  nomatch = NULL,
  mult = "all"
)
```

Arguments

`portfolio`	A `data.frame` or `data.table` with portfolio rows and active date intervals.
`dates`	A `data.frame` or `data.table` with event or snapshot dates.
`period_start`	Character string. Name of the portfolio column with period start dates.
`period_end`	Character string. Name of the portfolio column with period end dates.
`date`	Character string. Name of the date column in `dates`.
`by`	Character vector with additional columns used to match `portfolio` and `dates`, for example policy number or claim identifier.
`nomatch`	When a row (with interval say, `[a,b]`) in x has no match in y, nomatch=NA means NA is returned for y's non-by.y columns for that row of x. nomatch=NULL (default) means no rows will be returned for that row of x.
`mult`	When multiple rows in y match to the row in x, `mult` controls which values are returned - "all" (default), "first" or "last".

Details

This is useful when claim records or other dated events need the rating factors, premium, exposure, or policy attributes that were active at the event date. The function performs an interval join between event dates and portfolio coverage periods, optionally within matching identifiers such as a policy number.

Value

An object with the same class as `portfolio`.

Author(s)

Martin Haringa

Examples

```
library(lubridate)
portfolio <- data.frame(
begin1 = ymd(c("2014-01-01", "2014-01-01")),
end = ymd(c("2014-03-14", "2014-05-10")),
termination = ymd(c("2014-03-14", "2014-05-10")),
exposure = c(0.2025, 0.3583),
premium =  c(125, 150),
car_type = c("BMW", "TESLA"))

## Find active rows on different dates
dates0 <- data.frame(active_date = seq(ymd("2014-01-01"), ymd("2014-05-01"),
by = "months"))
active_rows_by_date(
  portfolio,
  dates0,
  period_start = "begin1",
  period_end = "end",
  date = "active_date"
)

## With extra identifiers (merge claim date with time interval in portfolio)
claim_dates <- data.frame(claim_date = ymd("2014-01-01"),
car_type = c("BMW", "VOLVO"))

### Only rows are returned that can be matched
active_rows_by_date(
  portfolio,
  claim_dates,
  period_start = "begin1",
  period_end = "end",
  date = "claim_date",
  by = "car_type"
)

### When row cannot be matched, NA is returned for that row
active_rows_by_date(
  portfolio,
  claim_dates,
  period_start = "begin1",
  period_end = "end",
  date = "claim_date",
  by = "car_type",
  nomatch = NA
```

```
)
```

`add_observed_experience` *Add observed portfolio experience to a rating table*

Description

Attach the output of `factor_analysis()` to a `rating_table()` object so it can be shown in `autoplot.rating_table()`. This is useful when you want to compare fitted GLM relativities with the observed portfolio pattern for the same rating factor.

The observed metric is scaled before plotting. With `scale = "reference"` the metric is divided by the observed value of the model reference level. If a clear reference level cannot be found, the metric is scaled to its mean. With `scale = "mean"`, the metric is always scaled to its mean.

Usage

```
add_observed_experience(
  object,
  experience,
  metric = "risk_premium",
  label = "Observed experience",
  color = NULL,
  scale = c("reference", "mean")
)
```

Arguments

`object`	A `rating_table` object returned by `rating_table()`.
`experience`	A `factor_analysis` object returned by `factor_analysis()`.
`metric`	Character; metric from `experience` to plot. Common choices are `"frequency"`, `"average_severity"`, `"risk_premium"`, `"loss_ratio"` and `"average_premium"`, depending on which columns were supplied to `factor_analysis()`.
`label`	Character; legend label for the observed experience line.
`color`	Optional line color. If `NULL`, the internal risk premium color is used.
`scale`	Character; scaling applied before plotting. One of `"reference"` or `"mean"`.

Value

A `rating_table` object with observed portfolio experience attached.

Author(s)

Martin Haringa

Examples

```
df <- MTPL2
df$area <- as.factor(df$area)

model <- glm(
  nclaims ~ area + offset(log(exposure)),
  family = poisson(),
  data = df
)

observed <- factor_analysis(
  df,
  risk_factors = "area",
  claim_count = "nclaims",
  exposure = "exposure"
)

rating_table(model, model_data = df, exposure = "exposure") |>
  add_observed_experience(observed, metric = "frequency") |>
  autoplot(risk_factors = "area")
```

`add_prediction` *Add Model Predictions to a Data Frame*

Description

Adds predictions (and optionally confidence intervals) from one or more `glm` models to a data frame.

Usage

```
add_prediction(
  data,
  ...,
  predictions = NULL,
  prefix = "pred",
  confidence = FALSE,
  interval_names = c("lower", "upper"),
  alpha = 0.1,
  var = NULL,
  conf_int = NULL
)
```

Arguments

`data`	A `data.frame` containing the new data for which predictions should be generated.
`...`	One or more fitted model objects of class `"glm"`.

predictions	Optional character vector giving names for the new prediction columns. Must have the same length as the number of models supplied. If NULL (default), names are generated automatically using prefix, the model response, and the model object name.
prefix	Character. Prefix used for automatically generated prediction column names. Default is "pred".
confidence	Logical. If TRUE, add confidence intervals for predictions. Default is FALSE.
interval_names	Character vector of length two. Names appended to the prediction column name for lower and upper confidence interval bounds. Default is c("lower", "upper").
alpha	Numeric between 0 and 1. Controls the miscoverage level for interval estimates. Default is 0.10, corresponding to a 90% confidence interval.
var	Deprecated. Use predictions instead.
conf_int	Deprecated. Use confidence instead.

Value

A data.frame containing the original data along with additional columns for model predictions (and confidence intervals if requested).

Author(s)

Martin Haringa

Examples

```
mod1 <- glm(nclaims ~ age_policyholder,
            data = MTPL,
            offset = log(exposure),
            family = poisson())

# Add predicted values
mtpl_pred <- add_prediction(MTPL, mod1)

# Add predicted values with confidence bounds
mtpl_pred_ci <- add_prediction(MTPL, mod1, confidence = TRUE)
```

---

add_relativities *Add expert-based relativities to a refinement workflow*

Description

Splits an existing model variable into more detailed tariff segments using supplied relativities. This is useful when the GLM is fitted on a coarser rating factor for credibility or stability, but the final tariff needs a more detailed split that is based on portfolio exposure, expert judgement or externally agreed relativities.

Usage

```
add_relativities(
  model,
  model_variable,
  split_variable,
  relativities,
  exposure,
  normalize = TRUE
)
```

Arguments

`model`	Object of class `rating_refinement`, usually created with `prepare_refinement()`.
`model_variable`	Character string. Existing variable in the GLM. Levels of this variable can be split into more detailed tariff segments.
`split_variable`	Character string. More granular portfolio variable that defines the detailed groups inside `model_variable`.
`relativities`	Named list of data frames, usually created with `relativities()` and `split_level()`.
`exposure`	Character string. Exposure column used for weighting and, when requested, normalisation.
`normalize`	Logical. If `TRUE`, normalise the supplied relativities by exposure within each split model level.

Details

`model_variable` is the variable already used in the GLM. `split_variable` is the more detailed variable in the portfolio data that will be used to split one or more levels of `model_variable`. The `relativities` argument should be a named list describing those splits, usually built with `relativities()` and `split_level()`.

The step is stored on the `rating_refinement` object and is applied when `refit()` is called. When `normalize = TRUE`, the supplied relativities are normalised using exposure so that the refined split keeps the original level effect on average. This helps prevent an expert split from unintentionally changing the total premium level for the original model group.

When to use

`add_relativities()` is intended for refinement within an already reasonably homogeneous GLM segment. It redistributes an existing coefficient across sublevels using exposure-weighted relativities, while preserving the overall level of the original coefficient. This is useful for mild heterogeneity, commercial refinement, monotonic tariff differentiation, or expert-based segmentation within a stable risk group where the original GLM coefficient is broadly representative.

Limitations

The method is not a substitute for creating a separate risk segment when the original GLM coefficient is itself distorted. For example, suppose a broad industry segment contains many relatively stable businesses, but a few chemical companies drive most of the losses while representing little exposure. The fitted industry coefficient may then be dominated by the chemical companies'

experience. Applying exposure-weighted relativities inside that segment may barely reduce the coefficient for the large exposure group, because the original coefficient is already pulled upward by the outlier subgroup.

In that situation it is often better to create a separate GLM factor level, derive a separate tariff segment, or apply explicit segmentation or acceptation rules, instead of relying only on `add_relativities()`.

Value

Object of class `rating_refinement`.

Author(s)

Martin Haringa

Examples

```
portfolio <- data.frame(
  claims = c(1, 2, 1, 3, 2, 4),
  exposure = rep(1, 6),
  construction = factor(c("residential", "commercial", "residential",
                          "commercial", "residential", "commercial")),
  construction_detail = factor(c("flat", "shop", "house",
                                 "office", "flat", "shop"))
)

model <- glm(
  claims ~ construction + offset(log(exposure)),
  family = poisson(),
  data = portfolio
)

relativities <- relativities(
  split_level(
    "residential",
    new_levels = c("flat", "house"),
    relativities = c(0.95, 1.05)
  )
)

refined <- prepare_refinement(model, data = portfolio) |>
  add_relativities(
    model_variable = "construction",
    split_variable = "construction_detail",
    relativities = relativities,
    exposure = "exposure"
  )
```

`add_restriction` *Add coefficient restrictions to a refinement workflow*

Description

Fixes selected model levels to user-supplied relativities in a refinement workflow. This is useful when the fitted GLM coefficients need to be adjusted before the final tariff is refitted, for example to apply expert judgement, enforce a business rule, remove an implausible local effect, or make a tariff structure easier to explain.

Usage

```
add_restriction(model, restrictions)
```

Arguments

`model`	Object of class `rating_refinement`, usually created with `prepare_refinement()`.
`restrictions`	Data frame with exactly two columns. The first column must have the same name as the model variable to restrict and contains the levels to adjust. The second column contains the replacement relativities. Levels that are not supplied are filled with the currently fitted GLM relativities.

Details

`add_restriction()` stores a restriction step on a `rating_refinement` object. It does not refit the GLM immediately. The restrictions are applied when `refit()` is called.

The `restrictions` data frame identifies the model variable to restrict by its first column. The second column contains the relativities that should be used for those levels in the refined model. New code should use this function after `prepare_refinement()`; the deprecated `restrict_coef()` wrapper is only kept for backwards compatibility.

The restriction table may contain all levels of the model variable, or only the levels that need a manual adjustment. If only a subset is supplied, the missing levels are automatically filled with their current fitted GLM relativities. This makes it possible to fix one level explicitly while keeping the other levels at their already estimated values.

Value

Object of class `rating_refinement`.

Author(s)

Martin Haringa

Examples

```
portfolio <- data.frame(
  claims = c(1, 2, 1, 3, 2, 4),
  exposure = rep(1, 6),
  postal_area = factor(c("A", "B", "C", "A", "B", "C"))
)

model <- glm(
  claims ~ postal_area + offset(log(exposure)),
  family = poisson(),
  data = portfolio
)

restrictions <- data.frame(
  postal_area = "C",
  relativity = 1.10
)

refined <- prepare_refinement(model, data = portfolio) |>
  add_restriction(restrictions)
```

---

add_smoothing *Add smoothing to a refinement workflow*

Description

Replaces a grouped or binned model effect by a smoother tariff curve in a refinement workflow. This is commonly used for numeric rating factors such as age, vehicle age, insured value or bonus-malus years, where a raw GLM factor can be too jagged for a stable and explainable tariff.

Usage

```
add_smoothing(
  model,
  model_variable = NULL,
  source_variable = NULL,
  degree = NULL,
  breaks = NULL,
  smoothing = "spline",
  k = NULL,
  weights = NULL,
  tariff_class = NULL,
  rating_variable = NULL,
  x_cut = NULL,
  x_org = NULL
)
```

Arguments

model	Object of class rating_refinement, usually created with prepare_refinement().
model_variable	Character string. Existing grouped or binned variable in the GLM. This is the model term that will be replaced by a smoothed tariff factor.
source_variable	Character string. Original numeric portfolio variable underlying model_variable.
degree	Optional single whole number. Polynomial degree, used by polynomial smoothing methods.
breaks	Numeric vector with the tariff segment boundaries to use after smoothing. Values must be finite and strictly increasing.
smoothing	Character string with the smoothing method, for example "spline" when supported by the fitted workflow.
k	Optional single positive whole number. Number of basis functions for smoothing methods that use a basis dimension.
weights	Optional character string. Weights column, usually exposure.
tariff_class, rating_variable	Deprecated. Use model_variable and source_variable instead.
x_cut, x_org	Deprecated. Use model_variable and source_variable instead.

Details

add_smoothing() stores a smoothing step on a rating_refinement object. The original GLM contains model_variable, usually a factor or grouped tariff segment. The smoother is fitted against source_variable, the original numeric portfolio variable behind those groups. The smoothed result is then converted back to tariff segments using breaks and applied when refit() is called.

This makes the intended API explicit: first prepare the model with prepare_refinement(), then add a smoothing step, optionally adjust it with edit_smoothing(), and finally call refit(). The deprecated smooth_coef() wrapper is only kept for backwards compatibility.

Value

Object of class rating_refinement.

Author(s)

Martin Haringa

Examples

```
## Not run:
library(dplyr)

age_policyholder_frequency <- risk_factor_gam(
  data = MTPL,
  claim_count = "nclaims",
  risk_factor = "age_policyholder",
  exposure = "exposure"
```

```
)

age_segments_freq <- derive_tariff_segments(age_policyholder_frequency)

dat <- MTPL |>
  add_tariff_segments(age_segments_freq, name = "age_policyholder_freq_cat") |>
  mutate(across(where(is.character), as.factor)) |>
  mutate(across(where(is.factor), ~ set_reference_level(., exposure)))

freq <- glm(
  nclaims ~ bm + age_policyholder_freq_cat,
  offset = log(exposure),
  family = poisson(),
  data = dat
)

sev <- glm(
  amount ~ zip,
  weights = nclaims,
  family = Gamma(link = "log"),
  data = dat |> filter(amount > 0)
)

premium_df <- dat |>
  add_prediction(freq, sev) |>
  mutate(premium = pred_nclaims_freq * pred_amount_sev)

burn_unrestricted <- glm(
  premium ~ zip + bm + age_policyholder_freq_cat,
  weights = exposure,
  family = Gamma(link = "log"),
  data = premium_df
)

ref <- prepare_refinement(burn_unrestricted) |>
  add_smoothing(
    model_variable = "age_policyholder_freq_cat",
    source_variable = "age_policyholder",
    breaks = seq(18, 95, 5),
    weights = "exposure"
  )

## End(Not run)
```

`add_tariff_segments` *Add derived tariff segments to portfolio data*

Description

Adds the tariff segments derived by `derive_tariff_segments()` as a new factor column to a portfolio data set. This is the recommended way to attach derived tariff segments to the same portfolio rows that were used to fit the risk factor GAM.

Usage

```
add_tariff_segments(data, segments, name = NULL, overwrite = FALSE)
```

Arguments

`data`	A data frame to which the tariff segments should be added.
`segments`	Object of class `"tariff_segments"`, produced by `derive_tariff_segments()`. Old `"tariff_classes"` objects are accepted for backward compatibility.
`name`	Character string. Name of the new output column. If `NULL`, the name is based on the risk factor name, for example `"age_policyholder_segment"`.
`overwrite`	Logical. If `FALSE`, the function stops when `name` already exists in `data`.

Value

A data frame with the derived tariff segment column added.

Author(s)

Martin Haringa

Examples

```
## Not run:
age_segments <- risk_factor_gam(
  MTPL,
  risk_factor = "age_policyholder",
  claim_count = "nclaims",
  exposure = "exposure"
) |>
  derive_tariff_segments()

MTPL |>
  add_tariff_segments(age_segments, name = "age_policyholder_segment")

## End(Not run)
```

`allocate_excess_loss` *Allocate excess loss to a pricing portfolio*

Description

Large claims can distort risk-factor relativities and create unstable premiums. `allocate_excess_loss()` redistributes historical excess losses across a portfolio in a controlled and transparent way.

The function is typically used after `calculate_excess_loss()`. The base premium can be modelled on capped claim amounts, while the excess part of large claims is allocated back to the portfolio as an additional loading.

Usage

```
allocate_excess_loss(
  data,
  excess_amount,
  allocation_weight,
  risk_factor = NULL,
  allocation_subset = NULL,
  allocation = c("portfolio", "risk_factor", "partial"),
  credibility = NULL,
  credibility_basis = c("claims", "excess_claims", "allocation_weight"),
  credibility_threshold = 50,
  credibility_scale = 1,
  method = c("observed", "bootstrap"),
  n_bootstrap = 1000,
  bootstrap_seed = NULL,
  severity_noise = c("none", "lognormal", "normal"),
  severity_noise_sd = 0.25,
  preserve_total_excess = TRUE
)
```

Arguments

`data`	A data.frame, typically the output of `calculate_excess_loss()`.
`excess_amount`	Character string. Column containing the excess claim amount to allocate.
`allocation_weight`	Character string. Column used as allocation weight, typically exposure, premium, insured value or another earned unit.
`risk_factor`	Optional character string. Risk-factor column used for `allocation = "risk_factor"` or `allocation = "partial"`.
`allocation_subset`	Optional character string. Logical column indicating which rows participate in the allocation. If `NULL`, all rows are included.
`allocation`	Character string. One of `"portfolio"`, `"risk_factor"` or `"partial"`.

`credibility` Optional numeric scalar between 0 and 1. Used directly when `allocation = "partial"`. If `NULL`, credibility is calculated from `credibility_basis` and `credibility_threshold`.

`credibility_basis`
Character string. Experience basis used when `credibility = NULL`: `"claims"`, `"excess_claims"` or `"allocation_weight"`.

`credibility_threshold`
Positive numeric scalar. Amount of experience required to reach 50 percent credibility.

`credibility_scale`
Positive numeric scalar. Multiplies the derived or supplied credibility before it is capped between 0 and 1.

`method` Character string. Either `"observed"` or `"bootstrap"`.

`n_bootstrap` Positive whole number. Number of bootstrap samples.

`bootstrap_seed` Optional integer seed for reproducible bootstrap allocation.

`severity_noise` Character string. One of `"none"`, `"lognormal"` or `"normal"`.

`severity_noise_sd`
Non-negative numeric scalar controlling severity variation in bootstrap samples.

`preserve_total_excess`
Logical. If `TRUE`, the final allocation is rescaled so that the total allocated excess loss equals the total excess loss being allocated.

Details

Allocation methods:

The `allocation` argument determines how the excess burden is shared.

- `"portfolio"`: excess losses are pooled across the entire portfolio and redistributed using the specified allocation weight. The excess burden is shared by all included risks regardless of their risk-factor level.
  This provides the most stable excess loading and is often appropriate when excess losses are infrequent, highly volatile or considered a portfolio- wide risk rather than a risk-factor-specific characteristic.
- `"risk_factor"`: excess losses are allocated separately for each risk-factor level. The excess burden observed within a group is spread across all risks in that group and is not shared with other groups.
  This produces the strongest link between excess loadings and observed group experience, but can lead to volatile results when excess losses are rare.
- `"partial"`: excess losses are allocated using a credibility-weighted combination of portfolio and risk-factor experience. Risk-factor levels with more credible experience receive excess loadings that more closely reflect their own observed excess-loss burden, while less credible groups are pooled more strongly towards the portfolio average.
  This approach typically provides a good balance between pricing stability and risk differentiation and is therefore often the preferred choice in practical rating applications.

Credibility weighting:

For `allocation = "partial"`, excess losses are allocated using a credibility-weighted blend of portfolio and risk-factor experience.
The allocated loading is calculated as:

$$loading_g = Z_g \cdot loading_g^{risk\ factor} + (1 - Z_g) \cdot loading^{portfolio}$$

where `Z_g` represents the credibility assigned to the risk-factor-level experience.
If `credibility` is supplied, the same credibility is applied to all risk-factor levels.
If `credibility = NULL`, credibility is determined separately for each risk-factor level based on the selected `credibility_basis`:

$$Z_g = \frac{n_g}{n_g + credibility\_threshold}$$

where `n_g` is determined by `credibility_basis`:

- `"claims"`: total number of claims in the risk-factor level.
- `"excess_claims"`: number of claims with positive excess loss.
- `"allocation_weight"`: total allocation weight in the risk-factor level.

`credibility_threshold` represents the amount of experience required to reach 50 percent credibility.
Example:
A sector with 20 claims and `credibility_threshold = 50` receives:

$$Z = \frac{20}{20 + 50} = 0.29$$

Therefore 29% of the excess loading is based on the sector's own excess-loss experience and 71% is based on the portfolio-wide excess loading.
For example, with `credibility_threshold = 50`, a group with:

- 10 claims receives 17% credibility;
- 50 claims receives 50% credibility;
- 100 claims receives 67% credibility;
- 200 claims receives 80% credibility.

The final credibility is scaled using `credibility_scale` and then capped between 0 and 1:

$$Z_g = min(1, max(0, Z_g \cdot credibility\_scale))$$

Higher values of `credibility_threshold` or lower values of `credibility_scale` pool more strongly towards the portfolio loading.

Bootstrap allocation:
With `method = "observed"`, the function allocates the historically observed excess loss.
With `method = "bootstrap"`, the function repeatedly resamples observed positive excess claim amounts. This provides a pragmatic estimate of excess-loss volatility and the resulting uncertainty in excess loadings.
The approach is intended as a practical pricing approximation rather than a formal extreme value model.
The bootstrap affects both the total excess burden and the distribution of excess loss across risk-factor levels. Use `bootstrap_seed` to make bootstrap results reproducible.

Severity noise:

`severity_noise` can only be used with `method = "bootstrap"`.

If `severity_noise = "none"`, bootstrap samples reuse the observed excess claim amounts.

If `severity_noise = "lognormal"`, sampled excess claims are multiplied by lognormal noise. This is usually the most natural option for large claims, because claim amounts remain positive and variation is multiplicative.

If `severity_noise = "normal"`, additive normal noise is applied. This may be useful for experimentation, but is generally less natural for large positive claim amounts.

`severity_noise_sd` controls the amount of additional severity variation. As a rough guide:

- `0.10` provides limited variation;
- `0.25` provides moderate variation;
- `0.50` provides substantial variation.

Preserving the total excess loss:

If `preserve_total_excess = TRUE`, the final allocation is rescaled so that the sum of allocated excess loss equals the total excess loss being allocated.

This ensures that credibility blending, bootstrap sampling or other allocation choices do not unintentionally increase or decrease the total excess burden.

Typical pricing workflow:

A common workflow is:

1. Use `calculate_excess_loss()` to separate capped and excess losses.
2. Model the base premium using capped claim amounts.
3. Allocate the excess-loss burden using `allocate_excess_loss()`.
4. Add the resulting excess loading back to the technical premium using `apply_excess_loading()`.

This approach prevents a small number of large claims from distorting risk-factor relativities while still ensuring that the excess-loss burden is reflected in the final premium.

Value

An object of class `"excess_loss_allocation"`.

Author(s)

Martin Haringa

Examples

```
claims <- data.frame(
  sector = rep(c("Industry", "Retail"), each = 4),
  claim_amount = c(
    1000, 120000, 30000, 8000,
    2000, 150000, 40000, 6000
  ),
  earned_exposure = rep(1, 8)
)
```

```
decomposed <- calculate_excess_loss(
  claims,
  claim_amount = "claim_amount",
  threshold = 100000
)

# Pool all excess losses across the portfolio
portfolio_allocation <- allocate_excess_loss(
  decomposed,
  excess_amount = "excess_claim_amount",
  allocation_weight = "earned_exposure",
  allocation = "portfolio"
)

# Allocate excess losses separately by sector
sector_allocation <- allocate_excess_loss(
  decomposed,
  excess_amount = "excess_claim_amount",
  allocation_weight = "earned_exposure",
  risk_factor = "sector",
  allocation = "risk_factor"
)

# Blend sector and portfolio experience using credibility
partial_allocation <- allocate_excess_loss(
  decomposed,
  excess_amount = "excess_claim_amount",
  allocation_weight = "earned_exposure",
  risk_factor = "sector",
  allocation = "partial",
  credibility_basis = "claims",
  credibility_threshold = 50
)

summary(partial_allocation)
```

`apply_excess_loading` *Apply excess loading to a pricing portfolio*

Description

Apply an allocated excess-loss loading to a portfolio data set.

Usage

```
apply_excess_loading(
  data,
  allocation,
  base_premium = "base_premium",
```

```
  allocated_excess_loss = NULL,
  allocated_loading = NULL,
  weight = NULL,
  output = c("premium", "rate")
)
```

Arguments

data	A data.frame containing the base premium or base rate.
allocation	An object returned by allocate_excess_loss().
base_premium	Character string. Column containing the base premium amount or base rate before the excess loading is added.
allocated_excess_loss	Optional character string. Column in allocation$data containing the allocated excess-loss amount in monetary terms. If NULL, allocated_excess_loss is used.
allocated_loading	Optional character string. Column in allocation$data containing the allocated excess loading per unit of allocation weight. If NULL, allocated_loading is used.
weight	Optional character string. Weight column used to convert between premium amounts and rates when output = "rate".
output	Character string. Use "premium" to return premium amounts or "rate" to return rates per unit of weight.

Details

apply_excess_loading() is the final step in the excess-loss pricing workflow. It does not cap claims, estimate excess losses or allocate the excess burden. Instead, it takes the output of allocate_excess_loss() and adds the allocated excess component back to the base premium or base rate.

The function is typically used after the base premium has been modelled on capped claim amounts. The excess loading then ensures that the cost of claims above the selected threshold is still reflected in the final technical premium.

Premium output:

With output = "premium", the function adds the allocated excess loss in monetary terms to the base premium:

$$loaded\_premium = base\_premium + allocated\_excess\_loss$$

allocated_excess_loss is the row-level monetary amount of excess loss allocated to each risk.

Rate output:

With output = "rate", the function adds the allocated excess loading per unit of weight to the base rate:

$$loaded\_rate = base\_rate + allocated\_loading$$

Use this option when the base value represents a rate per exposure, premium unit, insured value or other allocation weight.

If the input column supplied through `base_premium` contains premium amounts rather than rates, the function first converts the base premium to a rate:

$$base\_rate = \frac{base\_premium}{weight}$$

Interpretation of allocation columns:

`allocated_excess_loss` represents the monetary excess-loss burden allocated to a row.

`allocated_loading` represents the excess loading per unit of allocation weight.

In other words:

$$allocated\_excess\_loss = allocated\_loading \cdot weight$$

This distinction is important when moving between premium amounts and rates.

Typical pricing workflow:

A common workflow is:

1. Use `calculate_excess_loss()` to separate capped and excess losses.
2. Model the base premium using capped claim amounts.
3. Allocate the excess-loss burden using `allocate_excess_loss()`.
4. Use `apply_excess_loading()` to add the allocated excess component back to the base premium or base rate.

This produces a final technical premium that reflects both the modelled capped loss cost and the separately allocated excess-loss burden.

Value

A data.frame. With `output = "premium"`, the result contains `base_premium`, `allocated_excess_loss`, `allocated_loading`, `excess_loading` and `loaded_premium`. With `output = "rate"`, the result contains `base_rate`, `allocated_loading` and `loaded_rate`.

Author(s)

Martin Haringa

Examples

```
claims <- data.frame(
  sector = rep(c("Industry", "Retail"), each = 4),
  claim_amount = c(
    1000, 120000, 30000, 8000,
    2000, 150000, 40000, 6000
  ),
  earned_exposure = rep(1, 8)
)

decomposed <- calculate_excess_loss(
```

```
  claims,
  claim_amount = "claim_amount",
  threshold = 100000
)

decomposed$base_premium <- 500

allocation <- allocate_excess_loss(
  decomposed,
  excess_amount = "excess_claim_amount",
  allocation_weight = "earned_exposure"
)

apply_excess_loading(
  decomposed,
  allocation,
  base_premium = "base_premium"
)

apply_excess_loading(
  decomposed,
  allocation,
  base_premium = "base_premium",
  weight = "earned_exposure",
  output = "rate"
)
```

`assess_excess_threshold` *Assess possible excess-loss thresholds*

Description

Compare candidate thresholds for capped severity and large-loss pricing work.

`assess_excess_threshold()` is a diagnostic helper. It does not choose a threshold automatically. It shows how many claims and how much historical claim cost sit above candidate thresholds, and how much pure premium would remain after capping claims at each threshold.

Use this before `calculate_excess_loss()` to understand the effect of the threshold on the portfolio. The output is useful for tariff notes, pricing reviews and governance discussions around capped severity models.

Usage

```
assess_excess_threshold(
  data,
  claim_amount,
  thresholds,
```

```
  exposure = NULL,
  group = NULL
)
```

Arguments

data	A data.frame with claim-level observations.
claim_amount	Character string. Claim amount column.
thresholds	Numeric vector of candidate thresholds.
exposure	Optional character string. Exposure column. If supplied, pure premium before and after capping is calculated.
group	Optional character string. Grouping column used to assess thresholds by segment.

Value

A data.frame with class "excess_threshold_assessment".

Author(s)

Martin Haringa

Examples

```
claims <- data.frame(
  sector = rep(c("Industry", "Retail"), each = 5),
  claim_amount = c(1000, 25000, 120000, 50000, 175000,
                   2000, 40000, 90000, 150000, 300000),
  earned_exposure = rep(1, 10)
)

thresholds <- assess_excess_threshold(
  data = claims,
  claim_amount = "claim_amount",
  thresholds = c(25000, 50000, 100000, 150000),
  exposure = "earned_exposure",
  group = "sector"
)

autoplot(thresholds, y = "premium_impact")
```

`autoplot.bootstrap_performance` *Autoplot for bootstrap_performance objects*

Description

`autoplot()` method for objects created by `bootstrap_performance()`. Produces a histogram and density plot of the bootstrapped RMSE values, with the RMSE of the original fitted model shown as a dashed vertical line. Optionally, 95% quantile bounds are shown as dotted vertical lines.

Usage

```
## S3 method for class 'bootstrap_performance'
autoplot(object, fill = "#E6E6E6", color = NA, ...)
```

Arguments

`object`	An object of class `"bootstrap_performance"`, produced by `bootstrap_performance()`.
`fill`	Fill color of the histogram bars. Default = `"#E6E6E6"`.
`color`	Border color of the histogram bars. Default = `NA`, which removes bar borders.
`...`	Additional arguments passed to `ggplot2::autoplot()`.

Value

A ggplot2::ggplot object.

Author(s)

Martin Haringa

Examples

```
## Not run:
mod1 <- glm(nclaims ~ age_policyholder, data = MTPL,
            offset = log(exposure), family = poisson())
x <- bootstrap_performance(mod1, MTPL, n_resamples = 100,
                           show_progress = FALSE)
autoplot(x)

## End(Not run)
```

autoplot.check_residuals

Autoplot for check_residuals objects

Description

`autoplot()` method for objects created by `check_residuals()`. Produces a simulation-based uniform QQ-plot of the residuals, with the Kolmogorov-Smirnov p-value shown in the subtitle. Optionally prints a message about whether deviations are detected.

Usage

```
## S3 method for class 'check_residuals'
autoplot(object, show_message = TRUE, max_points = 1000, ...)
```

Arguments

`object`	An object of class `"check_residuals"`, produced by `check_residuals()`.
`show_message`	Logical. If TRUE (default), prints a short message based on the p-value from the KS test.
`max_points`	Maximum number of QQ-plot points to display. If the residual check contains more points, an evenly spaced subset is shown. Use `Inf` to plot all points.
`...`	Additional arguments passed to `ggplot2::autoplot()`.

Value

A ggplot2::ggplot object.

Author(s)

Martin Haringa

autoplot.excess_loss_allocation

Plot an excess-loss allocation

Description

Visualise the allocated excess loading, allocated excess loss or credibility by allocation group.

Usage

```
## S3 method for class 'excess_loss_allocation'
autoplot(
  object,
  y = c("allocated_loading", "allocated_excess_loss", "credibility"),
  top_n = NULL,
  show_labels = FALSE,
  ...
)
```

Arguments

object	An object returned by allocate_excess_loss().
y	Character. Measure to plot on the y-axis.
top_n	Optional positive whole number. If supplied, only the largest top_n groups by y are shown.
show_labels	Logical. If TRUE, add direct value labels to the bars.
...	Unused.

Value

A ggplot object.

Author(s)

Martin Haringa

---

autoplot.excess_threshold_assessment

Plot an excess threshold assessment

Description

Visualise one diagnostic from an object returned by assess_excess_threshold(). The plot helps compare how candidate thresholds affect excess loss, excess claim counts or pure-premium impact.

Usage

```
## S3 method for class 'excess_threshold_assessment'
autoplot(
  object,
  y = c("premium_impact", "excess_loss", "n_excess_claims", "excess_loss_ratio"),
  ...
)
```

Arguments

object	An object returned by assess_excess_threshold().
y	Character. Measure to plot on the y-axis.
...	Unused.

Value

A ggplot object.

Author(s)

Martin Haringa

---

autoplot.factor_analysis

Automatically create a ggplot for objects obtained from factor analysis

---

Description

Takes an object produced by factor_analysis() or univariate() (deprecated NSE interface) and plots the available statistics.

Usage

```
## S3 method for class 'factor_analysis'
autoplot(
  object,
  metrics = NULL,
  ncol = 1,
  show_exposure = TRUE,
  show_exposure_labels = TRUE,
  sort_by_exposure = FALSE,
  level_order = NULL,
  decimal_mark = ",",
  line_color = NULL,
  bar_fill = NULL,
  label_width = 50,
  flip_bars = FALSE,
  show_total = FALSE,
  total_color = NULL,
  total_name = NULL,
  rotate_angle = NULL,
  custom_theme = NULL,
  remove_underscores = FALSE,
  compact_x_axis = TRUE,
  show_plots = NULL,
```

```
  background = NULL,
  labels = NULL,
  sort = NULL,
  sort_manual = NULL,
  dec.mark = NULL,
  color = NULL,
  color_bg = NULL,
  coord_flip = NULL,
  remove_x_elements = NULL,
  ...
)
```

Arguments

object	A factor_analysis or univariate object produced by factor_analysis() or univariate().
metrics	Numeric or character vector specifying which metrics to plot (default is all available metrics). The numeric positions are: • 1. Frequency (nclaims / exposure) • 2. Average severity (severity / nclaims) • 3. Risk premium (severity / exposure) • 4. Loss ratio (severity / premium) • 5. Average premium (premium / exposure) • 6. Exposure • 7. Severity • 8. Number of claims • 9. Premium Character values can be "frequency", "average_severity", "risk_premium", "loss_ratio", "average_premium", "exposure", "claim_amount", "claim_count", and "premium".
ncol	Number of columns in output (default = 1).
show_exposure	Show exposure as background bars behind line plots (default = TRUE).
show_exposure_labels	Show labels with the exposure bars (default = TRUE).
sort_by_exposure	Sort risk factor levels into descending order by exposure (default = FALSE).
level_order	Custom order for risk factor levels; character vector (default = NULL).
decimal_mark	Decimal mark; defaults to ",".
line_color	Optional override for line/point color. If NULL (default), colors are taken from the internal palette. If specified, the chosen color is applied to all line-based plots.
bar_fill	Optional override for background bar color. If NULL (default), the background color is taken from the internal palette. If specified, the chosen color is applied to all background bars.

label_width — Width of labels on the x-axis (default = 10).

flip_bars — Logical. If TRUE, flip cartesian coordinates for bar plots (metrics 6 to 9). This option does not affect the line-based plots for metrics 1 to 5.

show_total — Show line for total if by is used (default = FALSE).

total_color — Color for total line (default = "black").

total_name — Legend name for total line (default = NULL).

rotate_angle — Numeric value for angle of labels on the x-axis (degrees).

custom_theme — List with customized theme options.

remove_underscores — Logical; remove underscores from labels (default = FALSE).

compact_x_axis — Logical. When TRUE and ncol == 1, x-axis components are removed from all plots except the last one. The following elements are suppressed:

- axis.title.x
- axis.text.x
- axis.ticks.x

This prevents duplicated x-axes in vertically stacked patchwork plots. Defaults to TRUE.

show_plots — Deprecated. Use metrics instead.

background — Deprecated alias for show_exposure.

labels — Deprecated alias for show_exposure_labels.

sort — Deprecated alias for sort_by_exposure.

sort_manual — Deprecated alias for level_order.

dec.mark — Deprecated alias for decimal_mark.

color — Deprecated alias for line_color.

color_bg — Deprecated alias for bar_fill.

coord_flip — Deprecated alias for flip_bars.

remove_x_elements — Deprecated alias for compact_x_axis.

... — Other plotting parameters.

Value

A ggplot2 object.

Author(s)

Marc Haine, Martin Haringa

Examples

```
## --- New usage (SE, recommended) ---
x <- factor_analysis(MTPL2,
                     x = "area",
                     severity = "amount",
                     nclaims = "nclaims",
                     exposure = "exposure")
autoplot(x)

## --- Deprecated usage (NSE) ---
x_old <- univariate(MTPL2, x = area, severity = amount,
                    nclaims = nclaims, exposure = exposure)
autoplot(x_old)
```

`autoplot.rating_refinement` *Plot a model refinement step*

Description

Takes a `rating_refinement` object and plots one refinement step before `refit()` is called. This is useful for checking whether manual tariff restrictions, smoothing or expert-based relativities behave as intended before they are used in a refined pricing model.

For objects produced by `add_relativities()`, original levels that are split into new levels are removed from the connected original line and from the x-axis. Instead, the original level is shown as a horizontal blue segment spanning all child categories, with the original level label centred above the segment.

Usage

```
## S3 method for class 'rating_refinement'
autoplot(
  object,
  variable = NULL,
  step = NULL,
  remove_underscores = FALSE,
  rotate_angle = NULL,
  custom_theme = NULL,
  ...
)
```

Arguments

`object`	Object of class `rating_refinement`.
`variable`	Optional character string specifying the risk factor to plot. If `NULL` (default), all available variables in the refinement object are shown. If specified, only the selected risk factor is plotted.

`step` Optional integer specifying which refinement step to plot. This is mainly relevant when multiple refinement steps have been applied (e.g. multiple calls to `add_smoothing()`, `add_restriction()`, or `add_relativities()`).

- If `NULL` (default), the latest refinement step is shown.
- If specified, the corresponding step in the refinement sequence is used.

This makes it possible to inspect intermediate refinement stages before calling `refit()`.

`remove_underscores` Logical; if `TRUE`, underscores are replaced by spaces in the x-axis label. Default is `FALSE`.

`rotate_angle` Optional numeric value for the angle of x-axis labels.

`custom_theme` Optional list passed to `ggplot2::theme()`.

`...` Additional plotting arguments passed to ggplot2 geoms.

Value

A `ggplot2` object.

Author(s)

Martin Haringa

`autoplot.rating_table` *Plot risk factor effects from* `rating_table()` *results*

Description

Create a ggplot visualisation of a `rating_table` object produced by `rating_table()`. Estimates are plotted per risk factor, with optional exposure bars. Observed portfolio experience can be added first with `add_observed_experience()`.

When observed experience is attached, it is plotted as an additional line. The scaling is controlled by `add_observed_experience()`.

Usage

```
## S3 method for class 'rating_table'
autoplot(
  object,
  risk_factors = NULL,
  ncol = 1,
  show_exposure_labels = TRUE,
  decimal_mark = ",",
  y_label = "Relativity",
  bar_fill = NULL,
  model_color = NULL,
```

```
  use_linetype = FALSE,
  rotate_angle = NULL,
  custom_theme = NULL,
  remove_underscores = FALSE,
  labels = NULL,
  dec.mark = NULL,
  ylab = NULL,
  fill = NULL,
  color = NULL,
  linetype = NULL,
  ...
)
```

Arguments

object	A rating_table object returned by rating_table().
risk_factors	Character vector specifying which risk factors to plot. Defaults to all risk factors.
ncol	Number of columns in the patchwork layout. Default is 1.
show_exposure_labels	Logical; if TRUE, show exposure values as labels on the bars. Default is TRUE.
decimal_mark	Character; decimal separator, either "," (default) or ".".
y_label	Character; label for the y-axis. Default is "Relativity".
bar_fill	Fill color for the exposure bars. If NULL, taken from the internal palette.
model_color	Optional override for model line colors. If NULL, colors are taken from the internal discrete palette.
use_linetype	Logical; if TRUE, use different line types for models. Default is FALSE.
rotate_angle	Numeric value for angle of labels on the x-axis (degrees).
custom_theme	List with customised theme options.
remove_underscores	Logical; remove underscores from labels.
labels	Deprecated alias for show_exposure_labels.
dec.mark	Deprecated alias for decimal_mark.
ylab	Deprecated alias for y_label.
fill	Deprecated alias for bar_fill.
color	Deprecated alias for model_color.
linetype	Deprecated alias for use_linetype.
...	Additional arguments passed to ggplot2 layers.

Value

A ggplot/patchwork object.

`autoplot.riskfactor_gam` *Autoplot for GAM objects from* `risk_factor_gam()`

Description

Generates a `ggplot2` visualization of a fitted GAM created with `risk_factor_gam()`. The plot shows the fitted curve, and optionally confidence intervals and observed data points.

Usage

```
## S3 method for class 'riskfactor_gam'
autoplot(
  object,
  confidence = FALSE,
  color_gam = "steelblue",
  show_observations = FALSE,
  x_stepsize = NULL,
  size_points = 1,
  color_points = "black",
  rotate_labels = FALSE,
  remove_outliers = NULL,
  conf_int = NULL,
  ...
)
```

Arguments

`object`	An object of class `"riskfactor_gam"` returned by `risk_factor_gam()`.
`confidence`	Logical. If `TRUE`, add 95% confidence intervals around the fitted curve. Default is `FALSE`.
`color_gam`	Color for the fitted GAM line, specified by name (e.g., `"red"`) or hex code (e.g., `"#FF1234"`). Default is `"steelblue"`.
`show_observations`	Logical. If `TRUE`, add observed frequency/severity points corresponding to the underlying data.
`x_stepsize`	Numeric. Step size for tick marks on the x-axis. If `NULL`, breaks are determined automatically.
`size_points`	Numeric. Point size for observed data. Default is `1`.
`color_points`	Color for the observed data points. Default is `"black"`.
`rotate_labels`	Logical. If `TRUE`, rotate x-axis labels by 45 degrees to reduce overlap.
`remove_outliers`	Numeric. If specified, observations greater than this threshold are omitted from the plot.
`conf_int`	Deprecated. Use `confidence` instead.
`...`	Additional arguments passed to underlying `ggplot2` functions.

Value

A ggplot object representing the fitted GAM.

Author(s)

Martin Haringa

Examples

```
## Not run:
library(ggplot2)
fit <- risk_factor_gam(MTPL,
                       risk_factor = "age_policyholder",
                       claim_count = "nclaims",
                       exposure = "exposure")

autoplot(fit, show_observations = TRUE)

## End(Not run)
```

---

autoplot.tariff_segments

Autoplot for tariff segment objects

Description

`autoplot()` method for objects created by `derive_tariff_segments()`. Produces a `ggplot2::ggplot()` of the fitted GAM together with the derived tariff segment boundaries. Optionally, confidence intervals and observed data points can be added.

Usage

```
## S3 method for class 'tariff_segments'
autoplot(
  object,
  confidence = FALSE,
  color_gam = "steelblue",
  show_observations = FALSE,
  color_splits = "grey50",
  size_points = 1,
  color_points = "black",
  rotate_labels = FALSE,
  remove_outliers = NULL,
  conf_int = NULL,
  ...
)
```

Arguments

object	An object of class "tariff_segments", produced by derive_tariff_segments().
confidence	Logical, whether to plot 95% confidence intervals. Default = FALSE.
color_gam	Color of the fitted GAM line. Default = "steelblue".
show_observations	Logical, whether to add observed data points for each level of the risk factor. Default = FALSE.
color_splits	Color of the vertical split lines. Default = "grey50".
size_points	Numeric, size of points if show_observations = TRUE. Default = 1.
color_points	Color of observed points. Default = "black".
rotate_labels	Logical, whether to rotate x-axis labels by 45 degrees. Default = FALSE.
remove_outliers	Numeric, exclude observations above this value from the plot (helps with extreme outliers). Default = NULL.
conf_int	Deprecated. Use confidence instead.
...	Additional arguments passed to ggplot2::autoplot().

Value

A ggplot2::ggplot object.

Author(s)

Martin Haringa

---

autoplot.truncated_severity

Plot a fitted truncated severity distribution

Description

Creates a plot of the empirical cumulative distribution function (ECDF) of the observed truncated claim amounts together with the fitted truncated CDF.

Usage

```
## S3 method for class 'truncated_severity'
autoplot(
  object,
  ecdf_geom = c("point", "step"),
  x_label = NULL,
  y_label = NULL,
  y_limits = c(0, 1),
  x_limits = NULL,
```

```
  show_title = TRUE,
  digits = 2,
  truncation_digits = 2,
  geom_ecdf = NULL,
  xlab = NULL,
  ylab = NULL,
  ylim = NULL,
  xlim = NULL,
  print_title = NULL,
  print_dig = NULL,
  print_trunc = NULL,
  ...
)
```

Arguments

object	An object produced by fit_truncated_severity().
ecdf_geom	Character string indicating how to display the empirical CDF. Must be one of "point" or "step".
x_label	Title of the x axis. Defaults to "severity".
y_label	Title of the y axis. Defaults to "cumulative proportion".
y_limits	Numeric vector of length 2 specifying y-axis limits.
x_limits	Optional numeric vector of length 2 specifying x-axis limits.
show_title	Logical. If TRUE, print title and subtitle.
digits	Integer. Number of digits for parameter estimates in the subtitle.
truncation_digits	Integer. Number of digits used for truncation bounds.
geom_ecdf, xlab, ylab, ylim, xlim, print_title, print_dig, print_trunc	Deprecated argument names kept for backward compatibility.
...	Currently unused.

Details

The plot compares the empirical distribution of the observed, truncated claim severities with the fitted distribution conditional on the same truncation interval. This is a visual check of whether the selected severity distribution is plausible for the part of the portfolio that is actually observed.

Value

A ggplot2 object.

Author(s)

Martin Haringa

`bootstrap_performance` *Bootstrapped model performance*

Description

Generate repeated train/evaluation samples to compute model performance. Currently, the supported metric is root mean squared error (RMSE).

Usage

```
bootstrap_performance(
  model,
  data,
  n_resamples = 50,
  sample_fraction = 1,
  metric = "rmse",
  sampling = c("bootstrap", "split"),
  show_progress = TRUE,
  rmse_model = NULL,
  n = NULL,
  frac = NULL
)
```

Arguments

`model`	A fitted model object.
`data`	Data used to fit the model object.
`n_resamples`	Integer. Number of resampling replicates. Default = 50.
`sample_fraction`	Fraction of the data used in the training sample. Must be in `(0, 1]`. Default = 1.
`metric`	Character. Performance metric to compute. Currently only `"rmse"` is supported.
`sampling`	Character. Sampling scheme. `"bootstrap"` samples training rows with replacement and evaluates on out-of-bag rows when `sample_fraction < 1`. `"split"` samples training rows without replacement and evaluates on the remaining rows when `sample_fraction < 1`.
`show_progress`	Logical. Show progress bar during bootstrap iterations. Default = TRUE.
`rmse_model`	Optional numeric RMSE of the fitted (original) model. If NULL (default), it is computed automatically.
`n`, `frac`	Deprecated argument names. Use `n_resamples` and `sample_fraction` instead.

Details

To test the predictive stability of a fitted model it can be helpful to assess the variation in a performance metric. The variation is calculated by refitting the model on repeated samples and storing the resulting metric values.

- If `sample_fraction = 1`, the metric is evaluated on the sampled training data.
- If `sample_fraction < 1`, the metric is evaluated on rows that were not used for training.

Character columns and factor columns are converted to factors with levels taken from the full input data before resampling. For factor variables used in the model, the training sample is augmented when needed so every observed level is represented at least once. This prevents prediction failures when a level is present in the evaluation data but absent from a particular training sample.

Value

An object of class `"bootstrap_performance"`, which is a list with components:

rmse_bs Numeric vector with `n_resamples` bootstrap RMSE values.

rmse_mod Root mean squared error for the original fitted model.

metric Metric name.

sampling Sampling scheme.

Author(s)

Martin Haringa

Examples

```
## Not run:
mod1 <- glm(nclaims ~ age_policyholder, data = MTPL,
            offset = log(exposure), family = poisson())

# Use all records
x <- bootstrap_performance(mod1, MTPL, n_resamples = 80,
                           show_progress = FALSE)
print(x)
autoplot(x)

# Use 80% of records and evaluate on the remaining records
x_frac <- bootstrap_performance(mod1, MTPL, n_resamples = 50,
                                sample_fraction = .8, sampling = "split",
                                show_progress = FALSE)
autoplot(x_frac)

## End(Not run)
```

`calculate_excess_loss` *Decompose claim amounts into capped and excess parts*

Description

Large claims can distort risk-factor relativities and make pricing models unstable. `calculate_excess_loss()` separates each claim into a capped part and an excess part above a selected threshold.

Usage

```
calculate_excess_loss(data, claim_amount, threshold)
```

Arguments

`data`	A data.frame with claim-level observations.
`claim_amount`	Character string. Claim amount column.
`threshold`	Positive numeric scalar. Claims above this value contribute to the excess component. Claims below the threshold remain fully included in the capped claim amount.

Details

The capped claim amount can be used to model the base premium, while the excess component can be analysed, pooled or allocated separately. This allows the impact of large individual claims to be controlled without ignoring the associated cost.

The function is deliberately deterministic. It does not perform smoothing, credibility weighting, allocation or simulation. It simply decomposes each observed claim into:

$$claim\_amount = capped\_claim\_amount + excess\_claim\_amount$$

where:

$$excess\_claim\_amount = max(claim\_amount - threshold, 0)$$

and:

$$capped\_claim\_amount = min(claim\_amount, threshold)$$

The resulting excess component can subsequently be allocated using `allocate_excess_loss()` and added back to the technical premium using `apply_excess_loading()`.

Typical pricing workflow:

A common workflow is:

1. Select an excess threshold.
2. Split claims into capped and excess components.

3. Model frequency and severity using capped claim amounts.
4. Allocate the excess-loss burden separately.
5. Add the resulting excess loading back to the technical premium.

This approach reduces the influence of a small number of large claims on risk-factor relativities while ensuring that the total cost of excess losses remains reflected in the final premium.

Value

A data.frame with the original data and the columns `claim_amount`, `capped_claim_amount`, `excess_claim_amount` and `is_excess_claim`.

Author(s)

Martin Haringa

Examples

```
claims <- data.frame(
  claim_amount = c(1000, 120000, 30000)
)

calculate_excess_loss(
  claims,
  claim_amount = "claim_amount",
  threshold = 100000
)
```

---

check_overdispersion *Check overdispersion of a Poisson claim frequency model*

Description

Tests whether a fitted Poisson GLM shows overdispersion using Pearson's chi-squared statistic.

Usage

```
check_overdispersion(object)
```

Arguments

`object`	A fitted model of class `"glm"` with family Poisson.

Details

In Poisson claim frequency models, the variance is assumed to be equal to the mean. A dispersion ratio above 1 indicates that the observed variation is larger than expected under that assumption. In pricing work this can be a useful diagnostic signal for omitted heterogeneity, clustering, outliers, or model misspecification. It does not automatically mean that the model is unusable.

- A dispersion ratio close to 1 is broadly consistent with the Poisson variance assumption.
- A dispersion ratio above 1 suggests overdispersion.
- A p-value below 0.05 indicates statistically significant overdispersion.

Value

An object of class `"overdispersion_check"` and `"overdispersion"`, which is a list with elements:

pearson_chisq Pearson's chi-squared statistic.

dispersion_ratio Dispersion ratio, calculated as Pearson's chi-squared statistic divided by residual degrees of freedom.

residual_df Residual degrees of freedom.

p_value P-value from the chi-squared test.

For backwards compatibility the object also contains the aliases `chisq`, `ratio`, `rdf`, and `p`.

Author(s)

Martin Haringa

References

Bolker B. et al. (2017). GLMM FAQ See also: `performance::check_overdispersion()`.

Examples

```
x <- glm(nclaims ~ area, offset = log(exposure),
         family = poisson(), data = MTPL2)
check_overdispersion(x)
```

`check_residuals` *Check simulation-based model residuals*

Description

Checks whether a fitted model shows systematic residual deviations from the distribution implied by the model. The function uses simulation-based residuals from `DHARMa::simulateResiduals()`, which are especially useful for GLMs where classical residual plots can be hard to interpret.

Usage

```
check_residuals(object, n_simulations = 30)
```

Arguments

`object`	A fitted `"glm"` object supported by `DHARMa::simulateResiduals()`.
`n_simulations`	Number of simulations used to generate residuals. Must be a positive whole number. Default is 30.

Details

In insurance pricing, residual checks are used to assess whether a model is behaving consistently across the portfolio. For example, a Poisson frequency model may fit the average claim count well but still show structure in the residuals because of omitted rating factors, unmodelled heterogeneity, clustering, outliers, or an unsuitable distributional assumption.

DHARMa simulates new responses from the fitted model and compares the observed response with those simulations. The resulting scaled residuals are approximately uniformly distributed on `[0, 1]` when the model is correctly specified. This gives a common diagnostic scale for GLMs and related models, where raw residuals are otherwise difficult to compare across different fitted values, exposures, or expected claim amounts.

`check_residuals()` returns the scaled residuals, QQ-plot data, and a Kolmogorov-Smirnov p-value for a simple uniformity check. The p-value should be read as a diagnostic signal, not as a pricing decision rule. A low p-value indicates that the residual distribution differs from what the fitted model implies and that the model specification may need review.

Value

An object of class `"residual_check"` and `"check_residuals"`, which is a list with:

qq_data Data frame with theoretical quantiles (`x`) and observed scaled residuals (`y`).

scaled_residuals Numeric vector of DHARMa scaled residuals.

p_value P-value from a Kolmogorov-Smirnov test against `uniform(0, 1)`.

For backwards compatibility the object also contains the aliases `df` and `p.val`.

Author(s)

Martin Haringa

References

Dunn, K. P., & Smyth, G. K. (1996). Randomized quantile residuals. *JCGS*, 5, 1–10.

Gelman, A., & Hill, J. (2006). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Hartig, F. (2020). DHARMa: Residual Diagnostics for Hierarchical (Multi-Level / Mixed) Regression Models. R package version 0.3.0. https://CRAN.R-project.org/package=DHARMa

Examples

```
## Not run:
m1 <- glm(nclaims ~ area, offset = log(exposure),
          family = poisson(), data = MTPL2)
cr <- check_residuals(m1, n_simulations = 50)
autoplot(cr)

## End(Not run)
```

`derive_tariff_segments` *Derive insurance tariff segments*

Description

Derives data-driven tariff segments for a continuous risk factor from a fitted `"riskfactor_gam"` object produced by `risk_factor_gam()`. The segments help translate a smooth GAM response pattern into practical categorical rating factors for a GLM tariff.

Usage

```
derive_tariff_segments(
  object,
  complexity = 0,
  max_iterations = 10000,
  population_size = 200,
  seed = 1,
  alpha = NULL,
  niterations = NULL,
  ntrees = NULL
)
```

Arguments

`object`	An object of class `"riskfactor_gam"`, produced by `risk_factor_gam()`. Objects with the old `"fitgam"` class are still supported for backward compatibility.
`complexity`	Numeric. Controls the complexity penalty used when deriving segments. Higher values generally yield fewer tariff segments. Default = 0.

`max_iterations`	Integer. Maximum number of search iterations used by the underlying grouping algorithm. Default = 10000.
`population_size`	Integer. Number of candidate trees used by the underlying grouping algorithm. Default = 200.
`seed`	Integer, seed for the random number generator (for reproducibility).
`alpha`	Deprecated. Use `complexity` instead.
`niterations`	Deprecated. Use `max_iterations` instead.
`ntrees`	Deprecated. Use `population_size` instead.

Details

Evolutionary trees (via `evtree::evtree()`) are used as a technique to bin the fitted GAM object into candidate tariff segments. This method is based on the work by Henckaerts et al. (2018). See Grubinger et al. (2014) for details on the parameters controlling the evtree fit.

Value

A `list` of class `"tariff_segments"` with components:

gam_prediction Data frame with the fitted GAM curve.

risk_factor Name of the continuous risk factor.

model_type Model type: `"frequency"`, `"severity"`, or `"pure_premium"`.

classification_data Data frame used to derive the segments.

risk_factor_values Observed risk factor values in portfolio row order.

segment_boundaries Numeric vector with segment boundaries.

assigned_segments Factor with the tariff segment assigned to each observed risk factor value.

For backward compatibility, the old components `prediction`, `x`, `model`, `data`, `x_obs`, `splits`, `class_boundaries`, `assigned_groups`, and `tariff_classes` are also returned.

Author(s)

Martin Haringa

References

Antonio, K. and Valdez, E. A. (2012). Statistical concepts of a priori and a posteriori risk classification in insurance. *Advances in Statistical Analysis*, 96(2), 187–224. doi:10.1007/s1018201101527

Grubinger, T., Zeileis, A., and Pfeiffer, K.-P. (2014). *evtree: Evolutionary learning of globally optimal classification and regression trees in R*. Journal of Statistical Software, 61(1), 1–29. doi:10.18637/jss.v061.i01

Henckaerts, R., Antonio, K., Clijsters, M., & Verbelen, R. (2018). A data driven binning strategy for the construction of insurance tariff classes. *Scandinavian Actuarial Journal*, 2018(8), 681–705. doi:10.1080/03461238.2018.1429300

Wood, S.N. (2011). Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *JRSS B*, 73(1), 3–36. doi:10.1111/j.14679868.2010.00749.x

Examples

```
## Not run:
library(dplyr)

# Recommended new usage (SE)
age_segments <- risk_factor_gam(MTPL,
                                risk_factor = "age_policyholder",
                                claim_count = "nclaims",
                                exposure = "exposure") |>
  derive_tariff_segments()

MTPL |>
  add_tariff_segments(age_segments, name = "age_policyholder_segment")

## End(Not run)
```

`edit_smoothing` *Edit an existing smoothing step in a refinement workflow*

Description

Manually adjusts a smoothing step that was previously added with `add_smoothing()`. This is intended for actuarial review of a smoothed tariff curve, for example to flatten an unstable segment, align the end points of an interval, or add extra control points where expert judgement should guide the curve. The adjusted smoothing is applied when `refit()` is called.

Usage

```
edit_smoothing(
  model,
  model_variable = NULL,
  step = NULL,
  from,
  to,
  from_value = NULL,
  to_value = NULL,
  control_positions = NULL,
  control_values = NULL,
  allow_extrapolation = FALSE,
  extrapolation_step = NULL
)
```

Arguments

`model`	Object of class `rating_refinement`, usually created with `prepare_refinement()`. Legacy `smooth` and `restricted` objects are still accepted for backwards compatibility.

`model_variable` Character string. The `model_variable` of the smoothing step to edit. Required when more than one smoothing step exists and `step` is not supplied.

`step` Optional numeric index of the smoothing step to edit.

`from`, `to` Numeric values giving the start and end of the source-variable interval to modify.

`from_value`, `to_value`
Optional numeric values used to override the smoothed curve value at `from` and `to`.

`control_positions`, `control_values`
Optional numeric vectors of equal length. These define additional points that the edited smoothing curve should pass through.

`allow_extrapolation`
Logical. Whether edits may extend beyond the observed source-variable range.

`extrapolation_step`
Optional positive numeric scalar used to set the spacing of extra break points when extrapolation is allowed.

Details

Use `model_variable` or `step` to identify the smoothing step to edit. The interval from `from` to `to` defines the part of the source variable range that should be changed. `from_value` and `to_value` can be used to force the curve values at the interval boundaries. `control_positions` and `control_values` add additional points that the edited curve should follow inside the interval.

Value

Object of class `rating_refinement`.

Author(s)

Martin Haringa

Examples

```
set.seed(42)
driver_age <- rep(seq(20, 59), each = 4)
exposure <- rep(1, length(driver_age))
age_band <- cut(
  driver_age,
  breaks = c(18, 30, 40, 50, 60),
  include.lowest = TRUE
)
expected_claims <- exp(
  -1.7 + 0.018 * (driver_age - 20) + 0.0006 * (driver_age - 40)^2
)
portfolio <- data.frame(
  claims = rpois(length(driver_age), exposure * expected_claims),
  exposure = exposure,
  driver_age = driver_age,
  age_band = age_band
)
```

```
model <- glm(
  claims ~ age_band + offset(log(exposure)),
  family = poisson(),
  data = portfolio
)

refined <- prepare_refinement(model, data = portfolio) |>
  add_smoothing(
    model_variable = "age_band",
    source_variable = "driver_age",
    breaks = c(18, 30, 40, 50, 60),
    degree = 2,
    weights = "exposure"
  ) |>
  edit_smoothing(
    model_variable = "age_band",
    from = 30,
    to = 50,
    from_value = 1.00,
    to_value = 1.10,
    control_positions = c(40),
    control_values = c(1.05)
  )

refined_model <- refit(refined)
```

`extract_model_data` *Extract model data*

Description

[Experimental]

`extract_model_data()` retrieves the modelling data and metadata from fitted models. It works for objects of class `"glm"`, as well as objects produced by refitting procedures (`"refitsmooth"` or `"refitrestricted"`).

`model_data()` is kept as a deprecated compatibility wrapper.

Usage

```
extract_model_data(x)
```

Arguments

`x`	An object of class `"glm"`, `"refitsmooth"`, or `"refitrestricted"`.

Details

For GLM objects, the function returns the model data and attaches attributes with the response, rating factors, terms object, and any weights or offsets.

For refit objects, the function removes auxiliary columns used during smoothing or restriction and attaches attributes with rating factors, merged smooths, restrictions, and offsets.

Value

A `data.frame` of class `"model_data"` with additional attributes:

- `response` — response variable in the model;
- `rf` — names of risk factors in the model;
- `offweights` — weight and offset variables if present;
- `terms` — model terms object for plain GLMs;
- `mgd_rst`, `mgd_smt` — merged restrictions/smooths for refit objects;
- `new_nm`, `old_nm` — new and old column names for refit objects.

Author(s)

Martin Haringa

Examples

```
## Not run:
library(insurancerating)

pmodel <- glm(
  breaks ~ wool + tension,
  data = warpbreaks,
  family = poisson(link = "log")
)

extract_model_data(pmodel)

## End(Not run)
```

`factor_analysis` *Factor analysis for discrete risk factors*

Description

Performs a factor analysis for discrete risk factors in an insurance portfolio. The following summary statistics are calculated:

- frequency = number of claims / exposure
- average severity = severity / number of claims
- risk premium = severity / exposure
- loss ratio = severity / premium
- average premium = premium / exposure

Usage

```
factor_analysis(
  data = NULL,
  risk_factors = NULL,
  claim_amount = NULL,
  claim_count = NULL,
  exposure = NULL,
  premium = NULL,
  group_by = NULL,
  df = NULL,
  x = NULL,
  severity = NULL,
  nclaims = NULL,
  by = NULL
)
```

Arguments

`data`	A `data.frame` with the insurance portfolio.
`risk_factors`	Character vector: column(s) in `data` with the risk factor(s).
`claim_amount`	Character, column in `data` with claim amounts (default = NULL).
`claim_count`	Character, column in `data` with number of claims (default = NULL).
`exposure`	Character, column in `data` with exposures (default = NULL).
`premium`	Character, column in `data` with premiums (default = NULL).
`group_by`	Character vector of column(s) in `data` to group by in addition to `risk_factors`.
`df`, `x`, `severity`, `nclaims`, `by`	Deprecated argument names. Use `data`, `risk_factors`, `claim_amount`, `claim_count`, and `group_by` instead.

Details

The function computes summary statistics for discrete risk factors.

- **Frequency**: number of claims / exposure
- **Average severity**: severity / number of claims

- **Risk premium**: severity / exposure
- **Loss ratio**: severity / premium
- **Average premium**: premium / exposure

If one or more input arguments are not specified, the related statistics are omitted from the results.

Migration from `univariate()`:

The function `univariate()` is deprecated as of version 0.8.0 and replaced by `factor_analysis()`. In addition to the name change, the interface has also changed:

- `univariate()` used **non-standard evaluation (NSE)**, so column names could be passed unquoted (e.g. `x = area`).
- `factor_analysis()` uses **standard evaluation (SE)**, so column names must be passed as character strings (e.g. `x = "area"`).

This makes the function easier to use in programmatic workflows.

`univariate()` is still available for backward compatibility but will emit a deprecation warning and will be removed in a future release.

Value

An object of class `"factor_analysis"` and `"univariate"` with summary statistics.

Author(s)

Martin Haringa

Examples

```
## --- New usage (SE) ---
factor_analysis(MTPL2,
                risk_factors = "area",
                claim_amount = "amount",
                claim_count = "nclaims",
                exposure = "exposure",
                premium = "premium")

## --- Deprecated usage (NSE) ---
univariate(MTPL2,
           x = area,
           severity = amount,
           nclaims = nclaims,
           exposure = exposure,
           premium = premium)
```

`fisher_classify` *Fisher's natural breaks classification*

Description

`fisher_classify()` is deprecated as of version 0.8.0 because Fisher-Jenks classification is not directly linked to the insurance rating workflow.

Classifies a continuous numeric vector into intervals using Fisher-Jenks natural breaks. Useful for choropleth mapping or other applications where grouped ranges are required.

Usage

```
fisher_classify(x, n = 7, dig.lab = NULL, diglab = NULL)
```

Arguments

`x`	A numeric vector to be classified.
`n`	Integer. Number of classes to generate (default = 7).
`dig.lab`	Integer. Number of significant digits to use for interval labels (default = 2).
`diglab`	Deprecated. Use `dig.lab` instead.

Details

The `"fisher"` style uses the algorithm proposed by Fisher (1958), commonly referred to as the Fisher-Jenks algorithm. This function is a thin wrapper around `classInt::classIntervals()`.

The argument `diglab` is deprecated and will be removed in a future version.

Value

A factor indicating the interval to which each element of `x` belongs.

Author(s)

Martin Haringa

References

Bivand, R. (2018). *classInt: Choose Univariate Class Intervals*. R package version 0.2-3. https://CRAN.R-project.org/package=classInt

Fisher, W. D. (1958). *On grouping for maximum homogeneity*. Journal of the American Statistical Association, 53, pp. 789–798. doi:10.1080/01621459.1958.10501479

Examples

```
set.seed(1)
x <- rnorm(100)
fisher_classify(x, n = 5)
```

`fit_truncated_severity` *Fit severity distributions to truncated claim data*

Description

[Experimental] Estimate an underlying claim severity distribution when the observed claims are truncated.

Usage

```
fit_truncated_severity(
  losses = NULL,
  distribution = c("gamma", "lognormal"),
  lower_truncation = NULL,
  upper_truncation = NULL,
  start_values = NULL,
  print_initial = TRUE,
  n_variants = 1,
  n_shape_grid = 8,
  n_scale_grid = 8,
  show_progress = FALSE,
  show_summary = TRUE,
  y = NULL,
  dist = NULL,
  left = NULL,
  right = NULL,
  start = NULL,
  trace = NULL,
  report = NULL
)
```

Arguments

`losses`	Numeric vector with observed claim severities.
`distribution`	Severity distribution to fit: `"gamma"` or `"lognormal"`.
`lower_truncation`	Numeric lower truncation point. Claims at or below this value are assumed not to be present in `losses`. Defaults to `0`.
`upper_truncation`	Numeric upper truncation point. Claims at or above this value are assumed not to be present in `losses`. Defaults to `Inf`.
`start_values`	Optional named list of starting values. If `NULL`, a multi-start strategy is used. For a gamma distribution use `shape` and `scale`; for a lognormal distribution use `meanlog` and `sdlog`.
`print_initial`	Deprecated logical retained for backward compatibility.

`n_variants`	Controls how many local variations around base starts are used.
`n_shape_grid`	Number of grid points for gamma shape.
`n_scale_grid`	Number of grid points for gamma scale.
`show_progress`	Logical. If `TRUE`, prints periodic progress during the fitting loop.
`show_summary`	Logical. If `TRUE`, prints a short summary at the end.
`y`, `dist`, `left`, `right`, `start`, `trace`, `report`	Deprecated argument names kept for backward compatibility.

Details

In insurance pricing, severity models are often fitted on claim amounts that are not observed over the full range of possible losses. Small claims may be absent because of a deductible, reporting threshold, or data extraction rule. Very large claims may be capped, excluded, or modelled separately as large losses. A standard gamma or lognormal fit on the remaining observed claims treats that truncated sample as if it were complete, which can bias the estimated severity distribution.

`fit_truncated_severity()` fits the distribution conditional on the claim being observed within the truncation interval. This means the fitted likelihood uses the density divided by the probability mass between `lower_truncation` and `upper_truncation`. The function is intended for truncation, where claims outside the interval are absent from the data. This differs from censoring, where claims outside a limit are still observed but their exact amount is not known.

Observed losses must lie strictly inside the truncation interval. Values outside the interval indicate that the bounds do not describe the data and therefore produce an error.

Value

An object of class `c("truncated_severity", "truncated_dist", "fitdist")`. The object contains the fitted distribution parameters from `fitdistrplus::fitdist()` and additional attributes:

truncated_vec The observed losses used for fitting.

lower_truncation, upper_truncation The truncation bounds.

fit_attempts Metadata for each attempted start combination.

n_attempts, n_success, n_failed Fit attempt counts.

best_attempt_index Index of the selected start combination.

Examples

```
## Not run:
observed <- MTPL2$amount[MTPL2$amount > 500 & MTPL2$amount < 10000]
fit <- fit_truncated_severity(
  losses = observed,
  distribution = "gamma",
  lower_truncation = 500,
  upper_truncation = 10000
)
autoplot(fit)

## End(Not run)
```

`merge_date_ranges` *Reduce portfolio periods by merging adjacent date ranges*

Description

Merges overlapping or nearly adjacent policy periods within portfolio groups.

Usage

```
merge_date_ranges(
  df,
  ...,
  period_start = NULL,
  period_end = NULL,
  group_by = NULL,
  aggregate_cols = NULL,
  aggregate_fun = "sum",
  merge_gap_days = 5,
  begin = NULL,
  end = NULL,
  agg_cols = NULL,
  agg = NULL,
  min.gapwidth = NULL
)
```

Arguments

`df`	A `data.frame` or `data.table`.
`period_start`	Character string. Name of the column with period start dates.
`period_end`	Character string. Name of the column with period end dates.
`group_by`	Character vector with columns that identify the portfolio entity or rating segment within which date ranges should be merged.
`aggregate_cols`	Character vector with numeric columns to aggregate over merged ranges, for example premium or exposure.
`aggregate_fun`	Aggregation function or function name. Defaults to `"sum"`.
`merge_gap_days`	Non-negative whole number. Ranges with a gap smaller than this number of days are merged. Defaults to 5.
`begin`, `end`, `...`, `agg_cols`, `agg`, `min.gapwidth`	Deprecated NSE argument names kept for backward compatibility.

Details

Insurance portfolio extracts often contain multiple rows for the same policy or risk because of renewals, endorsements, product changes, or short administrative gaps. Before calculating portfolio

in/outflow, active exposure windows, or policy counts, it can be useful to reduce those rows to stable coverage intervals.

`merge_date_ranges()` merges date ranges within each `group_by` combination. Ranges with a gap smaller than `merge_gap_days` are treated as one continuous interval. If `aggregate_cols` is supplied, those columns are aggregated over the merged interval.

Value

A `data.table` of class `"reduce"`, with attributes:

- `begin` — name of the period-start column
- `end` — name of the period-end column
- `cols` — grouping columns

Author(s)

Martin Haringa

Examples

```
portfolio <- data.frame(
  policy_nr   = rep("12345", 11),
  productgroup= rep("fire", 11),
  product     = rep("contents", 11),
  begin_dat   = as.Date(c(16709,16740,16801,17410,17440,17805,17897,
                          17956,17987,18017,18262), origin="1970-01-01"),
  end_dat     = as.Date(c(16739,16800,16831,17439,17531,17896,17955,
                          17986,18016,18261,18292), origin="1970-01-01"),
  premium     = c(89,58,83,73,69,94,91,97,57,65,55)
)

# Merge periods
pt1 <- merge_date_ranges(
  portfolio,
  period_start = "begin_dat",
  period_end = "end_dat",
  group_by = c("policy_nr", "productgroup", "product"),
  merge_gap_days = 5
)

# Aggregate per period
summary(pt1, period = "days", policy_nr, productgroup, product)

# Merge periods and sum premium per period
pt2 <- merge_date_ranges(
  portfolio,
  period_start = "begin_dat",
  period_end = "end_dat",
  group_by = c("policy_nr", "productgroup", "product"),
  aggregate_cols = "premium",
  merge_gap_days = 5
```

```
)

# Create summary with aggregation per week
summary(pt2, period = "weeks", policy_nr, productgroup, product)
```

`model_performance` *Performance of fitted GLMs*

Description

Computes model performance indices for one or more fitted GLMs.

Usage

```
model_performance(...)
```

Arguments

`...`	One or more objects of class `"glm"`.

Details

The following indices are reported:

AIC Akaike's Information Criterion.

BIC Bayesian Information Criterion.

RMSE Root mean squared error, computed from observed and predicted values.

This function is adapted from `performance::model_performance()`.

Value

A data frame of class `"model_performance"`, with columns:

Model Name of the model object as passed to the function.

AIC AIC value.

BIC BIC value.

RMSE Root mean squared error.

Author(s)

Martin Haringa

Examples

```
m1 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(),
          data = MTPL2)
m2 <- glm(nclaims ~ area + premium, offset = log(exposure), family = poisson(),
          data = MTPL2)
model_performance(m1, m2)
```

MTPL *Motor Third Party Liability (MTPL) portfolio*

Description

A dataset containing the characteristics of 30,000 policyholders in a Dutch Motor Third Party Liability (MTPL) insurance portfolio. Includes information on policyholder characteristics, vehicle attributes, and claims.

Usage

```
MTPL
```

Format

A data frame containing 30,000 rows and 7 variables:

age_policyholder Age of the policyholder (in years).

nclaims Number of claims.

exposure Exposure, expressed in years. For example, if a vehicle is insured from July 1, the exposure equals 0.5 for that year.

amount Claim severity (in euros).

power Engine power of the vehicle (in kilowatts).

bm Bonus-malus level (0–22). Higher levels indicate worse claim history.

zip Region indicator (0–3).

Author(s)

Martin Haringa

MTPL2 *Motor Third Party Liability (MTPL) portfolio (3,000 policyholders)*

Description

A dataset containing the characteristics of 3,000 policyholders in a Dutch Motor Third Party Liability (MTPL) insurance portfolio. Includes information on region, claims, exposure, and premium.

Usage

```
MTPL2
```

Format

A data frame containing 3,000 rows and 6 variables:

customer_id Unique customer identifier.

area Region where the customer lives (0–3).

nclaims Number of claims.

amount Claim severity (in euros).

exposure Exposure, expressed in years.

premium Earned premium.

Author(s)

Martin Haringa

outlier_histogram *Portfolio histogram with tail bins*

Description

Visualize the distribution of a numeric portfolio variable while keeping extreme tails readable.

Insurance portfolios often contain skewed variables such as claim amounts, premium, exposure, insured sums, deductibles, or fitted premiums. A few very large policies or claim events can stretch a regular histogram so much that the body of the portfolio becomes hard to inspect. `outlier_histogram()` keeps the main range visible and groups values below `lower` or above `upper` into dedicated tail bins.

The plot is useful for actuarial portfolio checks, data quality review, and model preparation: it helps show where most risks are concentrated while still making the presence of extreme observations explicit.

Usage

```
outlier_histogram(
  data,
  x,
  lower = NULL,
  upper = NULL,
  density = FALSE,
  bins = 30,
  bar_fill = "#E6E6E6",
  bar_color = "white",
  tail_fill = "#F28E2B",
  tail_color = "white",
  density_color = "#2C7FB8",
  left = NULL,
  right = NULL,
  line = NULL,
  fill = NULL,
  color = NULL,
  fill_outliers = NULL
)
```

Arguments

data	A data.frame containing the portfolio variable to inspect.
x	Character; numeric column in data to plot.
lower	Optional numeric lower threshold. Values below this threshold are grouped into one left-tail bin.
upper	Optional numeric upper threshold. Values above this threshold are grouped into one right-tail bin.
density	Logical. If TRUE, add a density line. Default = FALSE.
bins	Integer. Number of bins used for the displayed range. Default = 30.
bar_fill	Fill color for regular histogram bars.
bar_color	Border color for regular histogram bars.
tail_fill	Fill color for tail bins.
tail_color	Border color for tail bins.
density_color	Color for the optional density line.
left, right	Deprecated aliases for lower and upper.
line	Deprecated alias for density.
fill, color, fill_outliers	Deprecated aliases for bar_fill, bar_color, and tail_fill.

Details

This function is intended as an exploratory portfolio diagnostic. It does not remove or winsorize observations in `data`; it only groups tail values in the visual display. The labels on the tail bins show the original range captured by each tail bin.

The method for handling outlier bins is based on https://edwinth.github.io/blog/outlier-bin/.

Value

A ggplot2::ggplot object.

Author(s)

Martin Haringa

Examples

```
# Inspect the full premium distribution
outlier_histogram(MTPL2, "premium")

# Keep the portfolio body readable while showing both tails
outlier_histogram(MTPL2, "premium", lower = 30, upper = 120, bins = 30)
```

`plot_severity_distribution`

Exploratory severity diagnostics by category

Description

Visualise individual claim amounts overall or per risk factor.

Average claim amounts can be misleading because a small number of large losses may dominate the mean. `plot_severity_distribution()` shows the full claim amount distribution, usually on a log scale, together with mean and median claim amount markers. If `risk_factor` is supplied, the distribution is shown per level of that risk factor. If `risk_factor = NULL`, the function shows the overall claim amount distribution. This makes heavy tails, clusters of small claims, spread differences, extreme losses and distributional shape visible in a way that average severity alone cannot.

The function is intended for exploratory severity diagnostics in pricing analysis, portfolio diagnostics, tariff notes, exploratory segmentation analysis and severity model validation. It uses standard evaluation: pass column names as character strings through `claim_amount` and `risk_factor`.

If `threshold` is supplied, claims above the threshold are highlighted in `"firebrick"` and a dotted threshold line is added. Claims at or below the threshold remain light grey. Direct labels for the mean, median and optional threshold are added with `ggrepel` when `show_labels = TRUE`; `ggrepel` is a suggested package and is not imported as a hard dependency.

Usage

```
plot_severity_distribution(
  data,
  claim_amount,
  risk_factor = NULL,
  top_n = 10,
  min_claims = 20,
  sort = c("median", "mean", "n_claims"),
  threshold = NULL,
  mean = TRUE,
  median = TRUE,
  distribution = c("none", "half_violin", "violin"),
  point_method = c("quasirandom", "jitter", "none"),
  orientation = c("horizontal", "vertical"),
  log_scale = TRUE,
  boxplot = FALSE,
  boxplot_width = 0.06,
  show_labels = TRUE,
  all_claims_label = "All claims",
  mean_label = "Mean",
  median_label = "Median",
  threshold_label = "Threshold",
  x_label = NULL,
  y_label = NULL,
  point_alpha = 0.16,
  point_size = 0.75,
  point_width = 0.15
)
```

Arguments

data	A data.frame with claim-level observations.
claim_amount	Character string. Name of the claim amount column.
risk_factor	Optional character string. Name of the risk factor used to split the severity distribution. If NULL, the overall claim amount distribution is shown.
top_n	Positive whole number. Number of categories to keep after filtering and sorting.
min_claims	Positive whole number. Categories with fewer than this number of claim observations are removed.
sort	Character. Metric used to sort and select categories. One of "median", "mean" or "n_claims".
threshold	Optional numeric scalar. If supplied, claims above this threshold are highlighted and a dotted threshold line is shown.
mean	Logical. If TRUE, add a marker for the average claim amount.
median	Logical. If TRUE, add a marker for the median claim amount.
distribution	Character. Distribution layer. One of "none", "half_violin" or "violin". Default is "none".

point_method	Character. Point placement method. One of "quasirandom", "jitter" or "none".
orientation	Character. "horizontal" places claim amount on the x-axis and categories on the y-axis. "vertical" reverses this.
log_scale	Logical. If TRUE, use a log10 scale for claim amounts.
boxplot	Logical. If TRUE, add a small centred boxplot. Default is FALSE.
boxplot_width	Numeric scalar. Width of the optional boxplot. Smaller values keep the boxplot as a subtle summary layer behind the individual claim points.
show_labels	Logical. If TRUE, add direct labels for the mean, median and, when supplied, threshold. Requires the suggested package ggrepel.
all_claims_label	Character string used as the category label when risk_factor = NULL.
mean_label	Character string used for the direct mean marker label. Default is "Mean".
median_label	Character string used for the direct median marker label. Default is "Median".
threshold_label	Character string used for the optional threshold label.
x_label	Optional character string. X-axis label. If NULL, a default is chosen from claim_amount, risk_factor and orientation.
y_label	Optional character string. Y-axis label. If NULL, a default is chosen from claim_amount, risk_factor and orientation.
point_alpha	Numeric alpha for raw claim points.
point_size	Numeric point size for raw claim points.
point_width	Numeric spread for raw claim points.

Value

A ggplot object. The plot can be extended with regular ggplot2 syntax, for example + ggplot2::labs(caption = "...") or + ggplot2::theme(...).

Author(s)

Martin Haringa

Examples

```
x <- plot_severity_distribution(
  MTPL,
  claim_amount = "amount",
  risk_factor = "zip",
  top_n = 4,
  min_claims = 20,
  point_method = "jitter",
  show_labels = FALSE
)
print(x)
```

```
x_threshold <- plot_severity_distribution(
  MTPL,
  claim_amount = "amount",
  risk_factor = NULL,
  threshold = 10000,
  min_claims = 20,
  point_method = "jitter",
  show_labels = FALSE
)
```

`prepare_refinement` *Prepare a model refinement workflow*

Description

Start a refinement workflow for a fitted GLM. Refinement steps such as smoothing, restrictions and expert-based relativities can be added sequentially and are only applied once `refit()` is called.

Usage

```
prepare_refinement(model, data = NULL)
```

Arguments

`model`	Object of class `glm`.
`data`	Optional data.frame with the same rows and model variables as the fitted GLM. If `NULL`, the data are retrieved from the model object.

Value

Object of class `rating_refinement`.

`rating_grid` *Construct observed rating-grid points from model data or a data frame*

Description

`rating_grid()` constructs rating-grid points by collapsing rows with identical combinations of grouping variables to a single row.

The function returns only combinations that are actually observed in the input data. It does **not** create the full Cartesian product of all unique values. This keeps the output compact and suitable for model diagnostics, portfolio summaries, and prediction analysis.

When `x` is an object returned by `extract_model_data()`, the function uses the extracted model metadata to determine the grouping variables if `group_by` is not supplied. When `x` is a plain `data.frame`, it is recommended to supply `group_by` explicitly.

Usage

```
rating_grid(
  x,
  group_by = NULL,
  exposure = NULL,
  exposure_by = NULL,
  aggregate_cols = NULL,
  drop_na = FALSE,
  group_vars = NULL,
  agg_cols = NULL
)
```

Arguments

x	A data.frame, an object of class "model_data" returned by extract_model_data(), or a fitted model that can be passed to extract_model_data().
group_by	Optional character vector with the variables that define the rating-grid points. If NULL and x is a "model_data" object, the risk-factor variables stored in the object are used. If NULL and x is a plain data.frame, all columns except those listed in exposure, exposure_by, and aggregate_cols are used.
exposure	Optional character; name of the exposure column to aggregate.
exposure_by	Optional character; name of a column used to split exposure or counts, for example a year variable.
aggregate_cols	Optional character vector with additional numeric columns to aggregate using sum(na.rm = TRUE).
drop_na	Logical; if TRUE, rows with missing values in group_by are removed before aggregation. Default is FALSE.
group_vars, agg_cols	Deprecated argument names. Use group_by and aggregate_cols instead.

Details

The implementation uses base R only. Output is always a regular data.frame, not a tibble or data.table.

If exposure_by is supplied, exposure or row counts are split across levels of that variable and returned in wide format, for example "exposure_2020" or "count_2020".

For objects returned by extract_model_data(), refinement mappings are joined by their original factor column. They are not cross-joined onto every row.

Value

A data.frame with one row per observed rating-grid point.

Author(s)

Martin Haringa

Examples

```
## Not run:
rating_grid(mtcars, group_by = c("cyl", "vs"))

rating_grid(
  mtcars,
  group_by = c("cyl", "vs"),
  exposure = "disp",
  exposure_by = "gear",
  aggregate_cols = "mpg"
)

pmodel <- glm(
  breaks ~ wool + tension,
  data = warpbreaks,
  family = poisson(link = "log")
)

pmodel |>
  extract_model_data() |>
  rating_grid()

## End(Not run)
```

`rating_table` *Build rating tables from fitted pricing models*

Description

`rating_table()` extracts model coefficients in tariff-table form. It adds the reference level for factor variables, can exponentiate GLM coefficients into relativities, and can add exposure by risk-factor level when the model data are available.

`rating_table()` is intended for fitted models:

- plain `glm` objects
- models obtained after `refit()`
- models obtained after `refit_glm()`

For pre-refit objects (`rating_refinement`, `restricted`, `smooth`) use `print()`, `summary()` and `autoplot()` instead.

Usage

```
rating_table(
  ...,
  model_data = NULL,
  exposure = TRUE,
```

```
  exposure_output = NULL,
  exponentiate = TRUE,
  significance = FALSE,
  round_exposure = 0,
  exposure_name = NULL,
  signif_stars = NULL
)
```

Arguments

`...`	glm object(s) produced by `glm()`, `refit()` or `refit_glm()`
`model_data`	Optional data.frame used to create the model(s). If `NULL`, the function tries to use `model$data` for each supplied model.
`exposure`	Logical or character. If `TRUE` (default), exposure is added if it can be inferred from the model. If `FALSE`, no exposure is added. If a character string is supplied, it is interpreted as the exposure column name.
`exposure_output`	Optional name for the exposure column in the output. If `NULL`, the original exposure column name is used.
`exponentiate`	logical indicating whether or not to exponentiate the coefficient estimates. Defaults to TRUE.
`significance`	Logical; if `TRUE`, show significance stars for p-values.
`round_exposure`	number of digits for exposure (defaults to 0)
`exposure_name`	Deprecated. Use `exposure_output` instead.
`signif_stars`	Deprecated. Use `significance` instead.

Value

Object of class `"rating_table"` and legacy class `"riskfactor"`.

`refit` *Refit a prepared refinement workflow*

Description

Applies the refinement steps stored in a `rating_refinement` object and returns a refitted GLM. This is the final step in the refinement workflow after `prepare_refinement()`, `add_smoothing()`, `add_restriction()` or `add_relativities()` have been used to define the proposed tariff structure.

Usage

```
refit(object, intercept_only = FALSE, ...)
```

Arguments

object	Object of class rating_refinement, usually created with prepare_refinement().
intercept_only	Logical. If FALSE (default), fit the refined model with remaining model terms still free. If TRUE, keep remaining existing relativities fixed as offsets and estimate only the intercept.
...	Additional arguments passed to stats::glm(), such as control.

Details

Refinement steps are not applied to the fitted model immediately. They are collected on the rating_refinement object so they can be inspected first, for example with autoplot.rating_refinement(). refit() then applies the steps in order, updates the model formula and data, and calls stats::glm() with the original model family and any additional arguments passed through ....

With intercept_only = FALSE, the refined GLM is fitted with the remaining free model terms that are still present after applying the refinement steps. With intercept_only = TRUE, remaining original model effects are fixed as offsets based on the existing fitted relativities. The refit then estimates only the intercept. This can be useful when the relative tariff structure should remain fixed and only the overall premium level should be recalibrated.

Value

A refitted object of class glm. The returned model also stores attributes used by rating_table() and rating_grid() to recognise refined rating factors, fixed relativities and smoothing metadata.

Author(s)

Martin Haringa

Examples

```
zip_df <- data.frame(
  zip = c(0, 1, 2, 3),
  zip_adj = c(0.8, 0.9, 1.0, 1.2)
)

model <- glm(
  nclaims ~ zip + offset(log(exposure)),
  family = poisson(),
  data = MTPL
)

refined_model <- prepare_refinement(model) |>
  add_restriction(zip_df) |>
  refit(intercept_only = TRUE)
```

relativities *Combine multiple level splits into relativities*

Description

Helper function to combine multiple level split definitions into a single named list suitable for use in `add_relativities()`.

Usage

```
relativities(...)
```

Arguments

`...`	One or more objects created by `split_level()`.

Value

A named list of data.frames suitable for the `relativities` argument in `add_relativities()`.

Examples

```
relativities(
  split_level("construction",
              c("residential", "commercial", "civil"),
              c(1.00, 1.10, 1.25))
)
```

rgammat *Generate random samples from a truncated gamma distribution*

Description

Generates random observations from a gamma distribution truncated to the interval $(lower, upper)$ using inverse transform sampling.

Usage

```
rgammat(n, shape, scale, lower, upper)
```

Arguments

`n`	Integer. Number of observations to generate.
`shape`	Numeric. Shape parameter of the gamma distribution.
`scale`	Numeric. Scale parameter of the gamma distribution.
`lower`	Numeric. Lower truncation bound.
`upper`	Numeric. Upper truncation bound.

Details

Random values are generated by sampling from a uniform distribution on the interval $[F(lower), F(upper)]$, where F is the CDF of the gamma distribution, and then applying the inverse CDF.

This approach ensures that the generated values follow the truncated distribution exactly.

The implementation is based on the inverse transform method as described in: https://www.r-bloggers.com/2020/08/generating-data-from-a-truncated-distribution/

Value

A numeric vector of length n containing random draws from the truncated gamma distribution.

Author(s)

Martin Haringa

`risk_factor_gam` *Fit a GAM for a continuous risk factor*

Description

Fits a generalized additive model (GAM) to a continuous risk factor in one of three insurance pricing contexts: claim frequency, claim severity, or pure premium. The fitted curve helps assess non-linear rating effects before a continuous variable is grouped into tariff segments or used in a GLM workflow.

Usage

```
risk_factor_gam(
  data,
  risk_factor = NULL,
  claim_count = NULL,
  exposure = NULL,
  claim_amount = NULL,
  pure_premium = NULL,
  model = "frequency",
  round_risk_factor = NULL,
  x = NULL,
  nclaims = NULL,
  amount = NULL,
  round_x = NULL
)
```

Arguments

data	A data.frame containing the insurance portfolio.
risk_factor	Character, name of column in data with the continuous risk factor.
claim_count	Character, name of column in data with the number of claims.
exposure	Character, name of column in data with the exposure.
claim_amount	(Optional) Character, column name in data with the claim amount. Required for model = "severity".
pure_premium	(Optional) Character, column name in data with the pure premium. Required for model = "pure_premium".
model	Character string specifying the model type. One of "frequency", "severity", or "pure_premium". Default is "frequency". The old value "burning" is deprecated and maps to "pure_premium".
round_risk_factor	(Optional) Numeric value to round the risk factor to a multiple of round_risk_factor. Can speed up fitting for factors with many distinct values.
x, nclaims, amount, round_x	Deprecated argument names. Use risk_factor, claim_count, claim_amount, and round_risk_factor instead.

Details

- **Frequency model**: Fits a Poisson GAM to the number of claims. The log of the exposure is used as an offset so the expected number of claims is proportional to exposure.
- **Severity model**: Fits a Gamma GAM with log link to the average claim size (total amount divided by number of claims). The number of claims is included as a weight.
- **Pure premium model**: Fits a Gamma GAM with log link to the pure premium (risk premium). Implemented by aggregating exposure-weighted pure premiums. The deprecated model value "burning" is still accepted for backward compatibility.

Migration from fit_gam():

The function fit_gam() is deprecated as of version 0.8.0 and replaced by risk_factor_gam(). In addition to the name change, the interface has also changed:

- fit_gam() used **non-standard evaluation (NSE)**, so column names could be passed unquoted (e.g. x = age_policyholder).
- risk_factor_gam() uses **standard evaluation (SE)**, so column names must be passed as character strings (e.g. risk_factor = "age_policyholder").

This makes the function easier to use in programmatic workflows.

riskfactor_gam() and fit_gam() are still available for backward compatibility but will emit deprecation warnings.

Value

A list of class "riskfactor_gam" with the following elements:

prediction	A data frame with predicted values and confidence intervals.

x	Name of the continuous risk factor.
model	The model type: "frequency", "severity", or "pure_premium".
data	Merged data frame with predictions and observed values.
x_obs	Observed values of the continuous risk factor.

Author(s)

Martin Haringa

References

Antonio, K. and Valdez, E. A. (2012). Statistical concepts of a priori and a posteriori risk classification in insurance. Advances in Statistical Analysis, 96(2):187–224.

Henckaerts, R., Antonio, K., Clijsters, M. and Verbelen, R. (2018). A data driven binning strategy for the construction of insurance tariff classes. Scandinavian Actuarial Journal, 2018:8, 681–705.

Wood, S.N. (2011). Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. Journal of the Royal Statistical Society (B) 73(1):3–36.

Examples

```
## --- Recommended new usage (SE) ---
# Column names must be passed as strings
risk_factor_gam(MTPL,
                risk_factor = "age_policyholder",
                claim_count = "nclaims",
                exposure = "exposure")

## --- Deprecated usage (NSE) ---
# This still works but will show a warning
fit_gam(MTPL,
        nclaims = nclaims,
        x = age_policyholder,
        exposure = exposure)
```

---

rlnormt *Generate random samples from a truncated lognormal distribution*

Description

Generates random observations from a lognormal distribution truncated to the interval $(lower, upper)$ using inverse transform sampling.

Usage

```
rlnormt(n, meanlog, sdlog, lower, upper)
```

Arguments

n	Integer. Number of observations to generate.
meanlog	Numeric. Mean of the underlying normal distribution.
sdlog	Numeric. Standard deviation of the underlying normal distribution.
lower	Numeric. Lower truncation bound.
upper	Numeric. Upper truncation bound.

Details

Random values are generated by sampling from a uniform distribution on the interval $[F(lower), F(upper)]$, where F is the CDF of the lognormal distribution, and then applying the inverse CDF.

This approach ensures that the generated values follow the truncated distribution exactly.

The implementation is based on the inverse transform method as described in: https://www.r-bloggers.com/2020/08/generating-data-from-a-truncated-distribution/

Value

A numeric vector of length n containing random draws from the truncated lognormal distribution.

Author(s)

Martin Haringa

---

rmse *Root Mean Squared Error (RMSE)*

Description

Computes the root mean squared error (RMSE) for a fitted model, defined as the square root of the mean of squared differences between predictions and observed values.

Usage

```
rmse(x, data = NULL)
```

Arguments

x	A fitted model object (e.g. of class "glm").
data	A data frame containing the variables used in the model. Required if not already stored in object.

Details

The RMSE indicates the absolute fit of the model to the data. It can be interpreted as the standard deviation of the unexplained variance, and is expressed in the same units as the response variable. Lower values indicate better model fit.

Value

A numeric value: the root mean squared error.

Author(s)

Martin Haringa

Examples

```
x <- glm(nclaims ~ area, offset = log(exposure),
         family = poisson(), data = MTPL2)
rmse(x, MTPL2)
```

`set_reference_level` *Set the reference level of a factor*

Description

Relevels a factor so that the selected category becomes the reference (first) level. By default, the reference level is chosen as the level with the largest total weight, for example the largest exposure in an insurance portfolio. Use `method = "manual"` with `reference_level` when a specific business category should be the reference level.

Usage

```
set_reference_level(
  x,
  weight = NULL,
  method = "largest_weight",
  reference_level = NULL
)
```

Arguments

`x`	A factor (unordered). Character vectors should be converted to factor before use.
`weight`	A numeric vector of the same length as `x`, typically representing exposure or frequency weights. Required when `method = "largest_weight"`.
`method`	Character. Method used to choose the reference level. Supported methods are `"largest_weight"` and `"manual"`.
`reference_level`	Character string with the level to use as reference when `method = "manual"`.

Value

A factor of the same length as `x`, with the selected reference level set as the first level.

Author(s)

Martin Haringa

References

Kaas, Rob & Goovaerts, Marc & Dhaene, Jan & Denuit, Michel. (2008). Modern Actuarial Risk Theory: Using R. doi:10.1007/9783540709985

Examples

```
## Not run:
library(dplyr)
df <- chickwts |>
mutate(across(where(is.character), as.factor)) |>
mutate(across(where(is.factor), ~set_reference_level(., weight)))

set_reference_level(df$feed, method = "manual", reference_level = "casein")

## End(Not run)
```

`split_level` *Define a level split with relativities*

Description

Helper function to define how one level of a risk factor should be split into sublevels with corresponding relativities. Intended for use inside `relativities()` and `add_relativities()`.

Usage

```
split_level(level, new_levels, relativities)
```

Arguments

`level`	Character string. Existing level of the risk factor to split.
`new_levels`	Character vector. Names of the new sublevels.
`relativities`	Numeric vector. Relativities corresponding to each sublevel. Must have the same length as `new_levels`.

Value

A named list of length 1, where the name is `level` and the value is a data.frame with columns `new_level` and `relativity`.

Examples

```
split_level(
  level = "construction",
  new_levels = c("residential", "commercial", "civil"),
  relativities = c(1.00, 1.10, 1.25)
)
```

`split_periods_to_months` *Split policy periods into monthly rows*

Description

Splits policy or exposure periods that cross calendar months into monthly rows. Numeric columns such as exposure or premium can be prorated over the resulting monthly rows.

This function uses **standard evaluation (SE)**: column names must be passed as **character strings** (e.g. `period_start = "begin_date"`). The older function `period_to_months()` used non-standard evaluation (NSE) and is deprecated as of version 0.8.0.

Usage

```
split_periods_to_months(
  df,
  period_start = NULL,
  period_end = NULL,
  prorate_cols = NULL,
  begin = NULL,
  end = NULL,
  cols = NULL
)
```

Arguments

`df`	A `data.frame` or `data.table`.
`period_start`	Character string. Name of the column with policy period start dates.
`period_end`	Character string. Name of the column with policy period end dates.
`prorate_cols`	Character vector with names of numeric columns to prorate over the monthly rows, for example exposure or premium.
`begin, end, cols`	Deprecated argument names kept for backward compatibility.

Details

Rating and monitoring work often needs exposure, premium, claim counts, or policy counts by calendar month. Source portfolios, however, usually contain policy periods that start and end on arbitrary dates. This helper expands those periods into monthly rows before modelling, reporting, or joining to monthly portfolio summaries.

Prorated columns are distributed according to the part of the policy period that falls in each monthly row. Full months receive weight 1; partial months use a 30-day month convention. The total value of each prorated column is preserved per original row.

Value

A `data.frame` with the same columns as in `df`, plus an `id` column.

Author(s)

Martin Haringa

Examples

```
library(lubridate)
portfolio <- data.frame(
  begin_date = ymd(c("2014-01-01", "2014-01-01")),
  end_date   = ymd(c("2014-03-14", "2014-05-10")),
  exposure   = c(0.2025, 0.3583),
  premium    = c(125, 150)
)

# New SE interface
split_periods_to_months(portfolio,
  period_start = "begin_date",
  period_end = "end_date",
  prorate_cols = c("premium", "exposure")
)

# Old NSE interface (deprecated)
## Not run:
period_to_months(portfolio, begin_date, end_date, premium, exposure)

## End(Not run)
```

---

split_relativities *Construct a relativities mapping for level splitting*

Description

Helper function to create a standardized data.frame defining relativities for sublevels within a risk factor level. This function is intended to be used as input for `add_relativities()`.

Usage

```
split_relativities(new_levels, relativities)
```

Arguments

`new_levels`	Character vector. Names of the new sublevels.
`relativities`	Numeric vector. Relativities corresponding to each sublevel. Must have the same length as `new_levels`.

Details

This function provides a convenient and safe way to construct the required input structure for `add_relativities()`. Each call defines how a single level of a risk factor is split into multiple sublevels with corresponding relativities.

Value

A data.frame with columns:

new_level Character. Name of the new sublevel.

relativity Numeric. Multiplicative factor relative to the base level.

Examples

```
split_relativities(
  new_levels = c("residential", "commercial", "civil"),
  relativities = c(1.00, 1.10, 1.25)
)
```

Index