

Package ‘insurancerating’

September 1, 2026

Type Package

Title Actuarial Tools for Insurance Pricing Models

Version 0.8.2

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](https://doi.org/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, partykit, rlang, scales, scam, stringr, tibble

Depends R (>= 4.1.0)

Suggests classInt, DBI, dbplyr, duckdb, ggbeeswarm, ggrepel, gt, spelling, knitr, rmarkdown, testthat

Language en-US

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

NeedsCompilation no

Repository CRAN

Date/Publication 2026-09-01 11:50:02 UTC

Contents

active_rows_by_date	3
add_portfolio_experience	5
add_prediction	7
add_rebasing	10
add_relativities	12
add_restriction	15
add_shrinkage	20
add_smoothing	22
add_tariff_segments	27
assess_excess_threshold	29
as_gt	32
audit_refinement	36
autoplot.bootstrap_performance	38
autoplot.check_residuals	39
autoplot.factor_analysis	40
autoplot.rating_refinement	44
autoplot.rating_table	46
autoplot.tariff_segments	49
autoplot.truncated_severity	52
bootstrap_coefficients	54
bootstrap_performance	56
calibrate_model	58
check_overdispersion	59
check_residuals	61
derive_tariff_segments	62
edit_smoothing	66
extract_model_data	70
factor_analysis	72
fisher_classify	74
fit_truncated_severity	75
merge_date_ranges	77
merge_date_ranges_db	80
model_performance	81
MTPL	83
MTPL2	83
outlier_histogram	84
plot_severity_distribution	86
premium_change	88
prepare_refinement	91
rating_grid	93
rating_grid_db	98
rating_table	100
redistribute_excess_loss	105
refit	114
relativity_specification	116
rgammat	118

<i>active_rows_by_date</i>	3
risk_factor_gam	119
rlnormt	121
rmse	122
set_reference_level	123
split_periods_to_months	125
Index	128

<i>active_rows_by_date</i>	<i>Match event dates to active portfolio periods</i>
----------------------------	--

Description

Match event dates, such as claim dates or inspection dates, to policy records that were active when the event occurred. The matched result contains the portfolio characteristics and coverage information applicable on each event date.

Usage

```
active_rows_by_date(
  portfolio,
  dates,
  period_start,
  period_end,
  date,
  by = NULL,
  unmatched = c("drop", "keep"),
  multiple_matches = c("all", "first", "last"),
  nomatch = NULL,
  mult = NULL
)
```

Arguments

portfolio	A <code>data.frame</code> or <code>data.table</code> with portfolio rows and active date intervals.
dates	A <code>data.frame</code> or <code>data.table</code> 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.
unmatched	Character string. Use "drop" to omit event dates for which no active portfolio row is found, or "keep" to retain them with missing portfolio information. The default is "drop".

multiple_matches	Character string controlling events that match multiple active portfolio rows. Use "all" to retain every match, or "first" or "last" to retain one matching row. The default is "all".
nomatch, mult	Deprecated technical argument names. Use unmatched and multiple_matches instead.

Details

Claim and event files often contain an event date and policy identifier but not the rating factors used at that point in time. The function performs an interval match between those events and the portfolio history. Supplying a policy identifier through `by` prevents an event from matching active periods belonging to another policy.

This is a temporal matching operation rather than a portfolio reduction. See [merge_date_ranges\(\)](#) for consolidating connected coverage periods and [split_periods_to_months\(\)](#) for expanding periods into monthly records.

Multiple matches can be valid when one event relates to several concurrently active coverages. They can also reveal overlapping or duplicated policy periods. Use `multiple_matches = "all"` when every active record is relevant; use "first" or "last" only when the source system defines which record should take precedence.

With `unmatched = "drop"`, events outside every applicable coverage period are omitted. With `unmatched = "keep"`, they remain visible with missing portfolio fields. Retaining unmatched events is generally preferable during data-quality review because it makes gaps in the policy history explicit.

The interval join is performed internally with `data.table::foverlaps()` on local copies. Neither input is modified by reference. The output follows the original order of dates and is always returned as a regular `data.frame`.

Value

A regular `data.frame` containing the event records and the portfolio information active on their dates. Event order is preserved. Depending on `multiple_matches`, one event can produce more than one output row.

Author(s)

Martin Haringa

See Also

[split_periods_to_months\(\)](#), [merge_date_ranges\(\)](#), [rating_grid\(\)](#)

Examples

```
portfolio <- data.frame(
  policy_id = c("P001", "P001", "P002"),
  coverage_start = as.Date(c("2024-01-01", "2025-01-01", "2025-01-01")),
  coverage_end = as.Date(c("2024-12-31", "2025-12-31", "2025-12-31")),
  sector = c("Retail", "Industry", "Services"),
```

```

    insured_amount = c(500000, 750000, 300000),
    earned_premium = c(900, 1250, 650)
  )

  claims <- data.frame(
    claim_id = c("C001", "C002", "C003"),
    policy_id = c("P001", "P001", "P002"),
    claim_date = as.Date(c("2024-06-15", "2025-08-10", "2026-01-10")),
    claim_amount = c(12000, 45000, 8000)
  )

  # Attach the policy characteristics that applied on each claim date.
  active_rows_by_date(
    portfolio,
    claims,
    period_start = "coverage_start",
    period_end = "coverage_end",
    date = "claim_date",
    by = "policy_id"
  )

  # Keep claims outside the available policy history for data-quality review.
  active_rows_by_date(
    portfolio,
    claims,
    period_start = "coverage_start",
    period_end = "coverage_end",
    date = "claim_date",
    by = "policy_id",
    unmatched = "keep"
  )

```

add_portfolio_experience

Add portfolio experience to a rating table

Description

`add_portfolio_experience()` enriches a `rating_table()` object with observed portfolio experience. When data is supplied, observed experience is calculated automatically for all risk factors in the rating table, unless `risk_factors` is specified. Existing `factor_analysis()` results can also be supplied through `observed`.

This makes it possible to compare fitted GLM relativities with observed portfolio patterns in `autoplot.rating_table()`. The full observed output is stored on the rating table, so `autoplot.rating_table()` can later switch between metrics such as "frequency", "average_severity" and "risk_premium" without recalculating the summaries.

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_portfolio_experience(x, ...)

## S3 method for class 'rating_table'
add_portfolio_experience(
  x,
  observed = NULL,
  data = NULL,
  risk_factors = NULL,
  claim_count = NULL,
  exposure = NULL,
  claim_amount = NULL,
  metric = NULL,
  label = "Observed experience",
  color = NULL,
  scale = c("reference", "mean"),
  experience = NULL,
  ...
)
```

Arguments

x	A rating_table object returned by rating_table() .
...	Unused.
observed	Optional factor_analysis() object or list of factor_analysis() objects. If supplied, these observed summaries are attached directly.
data	Optional data.frame. If observed = NULL, observed experience is calculated from this data.
risk_factors	Optional character vector. Risk factors for which observed experience should be calculated. If NULL, all risk factors in the rating table are used.
claim_count	Optional character string. Claim count column used by factor_analysis() .
exposure	Optional character string. Exposure column used by factor_analysis() .
claim_amount	Optional character string. Claim amount column used by factor_analysis() .
metric	Optional character string. Default observed metric to plot. Common choices are "frequency", "severity"/"average_severity" and "risk_premium". The metric can also be overridden in autoplot.rating_table() .
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".
experience	Deprecated alias for observed.

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
)

rating_table(model, model_data = df, exposure = "exposure") |>
  add_portfolio_experience(
    data = df,
    claim_count = "nclaims",
    exposure = "exposure"
  ) |>
  autoplot(risk_factors = "area", metric = "frequency")

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

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

add_prediction

Add model predictions to a pricing data set

Description

add_prediction() adds predictions from one or more fitted glm models to a data frame.

In pricing workflows, this is often used to bring count and severity model output together on the same portfolio. For example, an expected claim count can be normalised by exposure and multiplied by an expected average claim amount to calculate a risk premium per exposure unit.

The function is deliberately small: it does not refit models or decide how predictions should be combined. It only adds model predictions, and optionally confidence intervals, using clear output column names.

Usage

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

Arguments

<code>data</code>	A data.frame containing the new data for which predictions should be generated.
<code>...</code>	One or more fitted model objects of class "glm".
<code>predictions</code>	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.
<code>prefix</code>	Character. Prefix used for automatically generated prediction column names. Default is "pred".
<code>confidence</code>	Logical. If TRUE, add confidence intervals for predictions. Default is FALSE.
<code>interval_names</code>	Character vector of length two. Names appended to the prediction column name for lower and upper confidence interval bounds. Default is c("lower", "upper").
<code>alpha</code>	Numeric between 0 and 1. Controls the miscoverage level for interval estimates. Default is 0.10, corresponding to a 90% confidence interval.
<code>var</code>	Deprecated. Use predictions instead.
<code>conf_int</code>	Deprecated. Use confidence instead.

Details

Predictions are calculated on the response scale using `stats::predict(..., type = "response")`. For GLMs with a log link, such as Poisson count models or Gamma severity models, the added columns are already on the original response scale. For a Poisson claim-count model containing an exposure offset, this is the expected claim count for the supplied exposure, not frequency per exposure unit. Divide by exposure when a rate is required.

If `confidence = TRUE`, lower and upper confidence interval columns are added next to each prediction column. The default interval suffixes are "lower" and "upper".

Predictions containing missing values are retained. If one or more NA predictions are produced, the function issues a warning with the affected prediction columns and number of missing predictions. This is typically caused by missing predictor values in data or by predictor values outside the domain supported by the fitted model.

Value

A data.frame containing the original data and additional columns for model predictions. If confidence = TRUE, confidence interval columns are added as well.

Author(s)

Martin Haringa

Examples

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

# Add the expected claim count for each record's exposure
mtpl_pred <- add_prediction(
  MTPL,
  mod1,
  predictions = "expected_claim_count"
)

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

# Combine frequency and severity predictions into a risk premium
freq <- glm(nclaims ~ bm + zip,
            data = MTPL,
            offset = log(exposure),
            family = poisson())

severity_data <- MTPL[MTPL$nclaims > 0 & MTPL$amount > 0, ]
severity_data$average_claim_amount <-
  severity_data$amount / severity_data$nclaims

sev <- glm(average_claim_amount ~ bm + zip,
            data = severity_data,
            weights = nclaims,
            family = Gamma(link = "log"))

pricing <- add_prediction(
  MTPL,
  freq,
  sev,
  predictions = c("expected_claim_count", "expected_average_severity")
)
```

```
pricing$claim_frequency <-
  pricing$expected_claim_count / pricing$exposure
pricing$expected_loss <-
  pricing$expected_claim_count * pricing$expected_average_severity
pricing$risk_premium <-
  pricing$claim_frequency * pricing$expected_average_severity
```

add_rebasing

Rebase categorical tariff relativities to a reference level

Description

Rescale the current relativities of one categorical risk factor so that a selected level has relativity 1. `add_rebasing()` is an ordered refinement step: it uses the relativities available at that point in the workflow and retains all ratios between levels.

Usage

```
add_rebasing(model, model_variable, reference_level = NULL, weights = NULL)
```

Arguments

<code>model</code>	A <code>rating_refinement</code> object created with <code>prepare_refinement()</code> . Rebasing uses the current relativities at this point in the ordered workflow.
<code>model_variable</code>	Character string naming the categorical risk factor to rebase. This may identify an original GLM factor or a tariff factor created by an earlier <code>add_relativities()</code> , <code>add_restriction()</code> or <code>add_shrinkage()</code> step.
<code>reference_level</code>	Optional single character value naming the level that should receive relativity 1. When <code>NULL</code> , the level with the largest aggregated weight is selected automatically.
<code>weights</code>	<code>NULL</code> or a character string naming a numeric, non-negative column in the refinement data. The weights are used only when <code>reference_level = NULL</code> . <code>NULL</code> derives the basis from explicit model weights or a simple exposure offset.

Details

Rebasing changes the numerical reference of a tariff factor, but does not change its relative differentiation. If the current relativity of reference level j is r_j , every level is transformed as

$$r_i^{new} = \frac{r_i}{r_j}.$$

The selected reference level therefore becomes 1, while the ratio between any two levels remains unchanged. For example, relativities 0.8, 1.0 and 1.2 rebased to the first level become 1.0, 1.25 and 1.5. This is different from `add_shrinkage()`, which deliberately reduces the spread between levels.

Selecting the reference level:

Supply `reference_level` when the tariff has an established reference class or when governance requires a particular level to remain at 1. If `reference_level = NULL`, the level with the largest aggregated weight is selected. Ties are resolved by the order of the current factor levels.

With `weights = NULL`, explicit GLM weights are used when available; otherwise a single offset of the form `log(column)` is used. This commonly selects claim count for a weighted severity GLM and exposure for a frequency or risk-premium GLM. An explicit numeric column can be supplied when another portfolio basis is required. `weights` is ignored when `reference_level` is supplied because no automatic selection is then needed.

Position in the refinement workflow:

Rebasing is generally applied after the step that creates the final tariff levels. For example, `add_relativities()` may replace a broad level by several sublevels; `add_rebasing()` can then select one of those resulting sublevels as the new reference. It can also follow `add_shrinkage()` when the shrunked relativities should be reported relative to an established level.

`set_reference_level()` serves a different purpose. It changes the contrast reference of a factor before fitting a GLM. `add_rebasing()` rescales current tariff relativities inside an existing refinement specification. The refinement step and selected reference are retained for review by `summary()` and `audit_refinement()`.

Value

A `rating_refinement` object containing an ordered rebasing step. The step stores the original and rebased relativities, the selected reference level, its original relativity, the selection method and, when applicable, the aggregated level weights. The GLM is fitted only when `refit()` is called.

Author(s)

Martin Haringa

See Also

`prepare_refinement()`, `set_reference_level()`, `add_relativities()`, `add_shrinkage()`, `add_restriction()`, `refit()`, `audit_refinement()`

Examples

```
portfolio <- data.frame(
  claims = c(1, 2, 1, 3, 2, 4, 1, 5),
  exposure = c(1, 1, 1, 1, 2, 1, 1, 1),
  sector = factor(rep(c("Industry", "Office", "Retail", "Transport"), 2))
)

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

# Keep Office as the explicit tariff reference after shrinkage.
```

```

refinement <- prepare_refinement(model, data = portfolio) |>
  add_shrinkage(
    model_variable = "sector",
    credibility = 0.9,
    weights = "exposure"
  ) |>
  add_rebasing(
    model_variable = "sector",
    reference_level = "Office"
  )

summary(refinement)
refined_model <- refit(refinement)
rating_table(refined_model)

# Omitting reference_level selects the level with the largest exposure.
exposure_reference <- prepare_refinement(model, data = portfolio) |>
  add_rebasing(
    model_variable = "sector",
    weights = "exposure"
  )

```

add_relativities

Add sublevel relativities to a refinement workflow

Description

Divide one or more levels of an existing GLM risk factor into more detailed tariff levels using supplied relativities. This can be appropriate when the GLM is estimated on a coarser factor for statistical stability, while a documented actuarial segmentation is required within sufficiently homogeneous model levels.

Usage

```

add_relativities(
  model,
  model_variable,
  split_variable,
  relativities,
  exposure,
  normalize = TRUE,
  output_variable = paste0(model_variable, "_refined")
)

```

Arguments

model	Object of class <code>rating_refinement</code> , created with <code>prepare_refinement()</code> . A fitted GLM, including a model returned by <code>refit()</code> , is not accepted directly; retain and modify the corresponding refinement specification instead.
-------	--

model_variable	Character string. Existing variable in the GLM, or a restricted version created by an earlier <code>add_restriction()</code> step. Levels of the underlying model variable can be split into more detailed tariff segments. When an earlier restriction exists, its coefficients are used automatically.
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 <code>relativities()</code> and <code>split_level()</code> .
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.
output_variable	Character string naming the resulting hybrid tariff factor. The default appends <code>_refined</code> to model_variable. A more application-specific name, such as <code>sbi_tariff_segment</code> , can make the intended tariff use clearer in model output and reporting. The name must not overwrite an existing column in the refinement data.

Details

`add_relativities()` stores a relativity step on a `rating_refinement` object. It does not alter the fitted GLM immediately. The split is evaluated in the recorded step order and applied when `refit()` is called.

`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()`. `output_variable` names the resulting hybrid tariff factor: levels included in `relativities` are represented by their detailed `split_variable` level, while all other levels retain their `model_variable` level.

Levels of `model_variable` that are not included in `relativities` retain their existing model coefficient. In `rating_table()`, exposure for these retained levels is aggregated from `model_variable`, while exposure for the newly split levels is aggregated from `split_variable` within the specified parent model level. Omitting a model level from `relativities` therefore means that the level remains unsplit; it is not treated as an incomplete specification.

`add_relativities()` validates the supplied sublevel names against the observed values of `split_variable` before storing the refinement step. A misspelled or incorrectly spaced category or sublevel therefore produces an immediate error, with a suggestion when a closely matching observed value is available. It also verifies that each sublevel occurs within its specified parent category of `model_variable`.

When `normalize = TRUE`, the supplied relativities are normalised using exposure so that their exposure-weighted mean equals one within the split model level. They then redistribute the existing model coefficient across the sublevels without changing its exposure-weighted average. With `normalize = FALSE`, the supplied relativities are applied directly.

Step order and restrictions:

If `model_variable` was restricted in an earlier `add_restriction()` step, the restricted coefficients are automatically used as the basis for the derived relativities. The user can continue to supply the original model variable; no additional argument is needed. Supplying the restricted

variable explicitly gives the same coefficient basis and does not apply the restriction a second time. Refinement steps are order-dependent, so a restriction added after `add_relactivities()` does not affect an earlier relativity step. Once the restricted coefficients have been used to derive the final split, `rating_table()` reports `output_variable` as the tariff factor and does not also show the intermediate restricted variable.

Conversely, `add_restriction()` can be called after `add_relactivities()` to adjust selected levels of the derived `output_variable`. The output variable is then recognised as an existing refinement factor; users do not need to set `allow_new_risk_factors = TRUE`. Levels omitted from the restriction table are fixed at the relativities calculated by this step.

Appropriate use:

`add_relactivities()` 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 when normalisation is used. Appropriate applications include mild residual heterogeneity, monotonic tariff differentiation and expert-based segmentation within a stable risk group where the original GLM coefficient remains 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 acceptance rules, instead of relying only on `add_relactivities()`.

Value

A `rating_refinement` object containing the stored relativity specification. The pricing GLM is not fitted again until `refit()` is called.

Author(s)

Martin Haringa

See Also

`prepare_refinement()`, `relativities()`, `split_level()`, `add_restriction()`, `add_shrinkage()`, `add_rebasing()`, `add_smoothing()`, `refit()`, `rating_table()`

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)
    ),
    split_level(
      "commercial",
      new_levels = c("shop", "office"),
      relativities = c(1.10, 0.90)
    )
  )

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

  # A subsequent restriction can revise one derived level. The remaining
  # tariff-segment levels are fixed at the relativities calculated above.
  refined <- refined |>
    add_restriction(data.frame(
      construction_tariff_segment = "flat",
      construction_tariff_segment_restricted = 1.00
    ))

  refined_model <- refit(refined)
  rating_table(refined_model, exposure = FALSE)

```

add_restriction

Add coefficient restrictions to a refinement workflow

Description

Fix selected risk-factor levels at user-supplied relativities before the refined pricing GLM is fitted. This can be appropriate when sampling variation produces an implausible local effect, when an

actuarial assumption is supported by additional information, or when a documented tariff constraint must be applied consistently.

Usage

```
add_restriction(
  model,
  restrictions,
  allow_new_levels = TRUE,
  allow_new_risk_factors = FALSE,
  replaces = NULL
)
```

Arguments

model	Object of class <code>rating_refinement</code> , created with <code>prepare_refinement()</code> . A fitted GLM, including a model returned by <code>refit()</code> , is not accepted directly; retain and modify the corresponding refinement specification instead.
restrictions	Data frame with exactly two columns. The first column must have the same name as the risk factor to restrict and contains the levels to adjust. This can also be the <code>output_variable</code> from an earlier <code>add_relativities()</code> step. The second column contains the replacement relativities. Levels that are not supplied are fixed at their current effective relativities.
allow_new_levels	Logical. If TRUE (default), restrictions may contain levels that were not observed in the model data. Their supplied relativities are treated as explicit tariff assumptions rather than model estimates. If FALSE, an unknown level results in an error.
allow_new_risk_factors	Logical. If FALSE (default), the first column of restrictions must identify a variable included in the fitted GLM or a tariff factor created by an earlier refinement step. Set this to TRUE to add an external variable that is present in the refinement data but absent from both the model and preceding refinement steps. All observed levels must then have supplied relativities, which are treated as fixed tariff assumptions.
replaces	NULL (default) or a character string naming an existing standalone model term that the new fixed risk factor replaces. During <code>refit()</code> , this term is removed before the restricted relativity column is added. Supplying <code>replaces</code> also provides the explicit opt-in required for a new risk factor; <code>allow_new_risk_factors = TRUE</code> is then unnecessary. Existing terms used in transformations or interactions cannot be replaced through this argument.

Details

`add_restriction()` stores a restriction step on a `rating_refinement` object. It does not alter the fitted GLM immediately. The restriction is evaluated in the recorded step order and applied when `refit()` is called. Retain the refinement object when reviewing or revising the specification.

The restrictions data frame identifies the risk factor to restrict by its first column. This may be a variable from the original GLM or a tariff factor created by an earlier refinement step. The second column contains the relativities used for those levels in the refined model.

Actuarial interpretation:

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 effective relativities at that point in the refinement workflow. These may be the original fitted GLM relativities or values produced by preceding refinement steps. This makes it possible to change one level explicitly while fixing all other levels at their current values.

Levels that were not observed when the GLM was fitted can also be supplied. Such a level has no coefficient estimate from the model data. Its relativity is therefore an explicit tariff assumption, for example based on expert judgement, external experience or a planned extension of the tariff. Existing levels that are not supplied remain fixed at their fitted relativities.

With `allow_new_levels = TRUE`, which is the default, these new tariff levels are retained in the refinement metadata and subsequently shown by `rating_table()`. An informational message identifies every newly added level, its supplied relativity and the fact that it was not observed in the model data. Set `allow_new_levels = FALSE` when the restriction table should be checked strictly against the levels observed by the fitted model, for example to detect spelling errors in level names.

When a newly supplied level is written as a numeric interval, the function checks it against the existing interval levels. A warning is issued if the new interval overlaps the current classification. The level is still retained because overlapping labels may occasionally be intentional, but they do not form an unambiguous tariff partition. To replace an interval classification, first add the complete new classification as a separate column in the refinement data, use that column as the first column of restrictions, and identify the old model term with `replaces`.

A variable that is present in the refinement data but was not included in the fitted GLM can be added with `allow_new_risk_factors = TRUE`. In that case all observed levels must have a supplied relativity. The new factor is applied as a fixed tariff factor during `refit()`; its effects are not estimated from the model data. This can be appropriate when an external classification or expert assumption must be incorporated, such as a hail zone derived from geographic information. `allow_new_risk_factors` does not create the portfolio variable itself. The refinement data must already contain a column assigning every observation to a level. This is required to apply the supplied relativities to individual records.

Replacing an existing model variable:

A new fixed tariff factor can either supplement the fitted GLM or replace an existing model variable. Supply `replaces` when the new factor represents an alternative tariff classification for an effect already present in the model. During `refit()`, the named existing term is removed and the supplied fixed relativities are inserted in its place. With `replaces = NULL`, the new factor is added alongside the existing model terms, which preserves the previous behaviour.

Supplying `replaces` is itself an explicit request to add the new risk factor, so `allow_new_risk_factors = TRUE` does not also need to be supplied. The replacement relationship is retained in the ordered refinement metadata and is shown by `print()`, `summary()` and `audit_refinement()`. This makes clear that the new factor substitutes for an earlier model effect rather than adding further multiplicative differentiation.

`replaces` is intentionally limited to a standalone main-effect term in the current refinement formula. A variable used in an interaction or transformed expression cannot be removed unambiguously through this argument. Such model structures should be revised explicitly before the

refinement is prepared. This argument is therefore not a general-purpose facility for deleting model terms.

Updating an existing restriction:

A later call to `add_restriction()` for the same risk factor and the same restricted model variable updates the restriction already stored in the refinement. Relativities supplied in the later call replace the previously stored values for those levels. Restrictions for levels that are not supplied again are retained.

The existing and new values are first combined and the resulting restriction table is then validated as one specification. This is useful when an actuarial assumption is revised during model refinement: only the affected levels need to be supplied again, while the remaining tariff assumptions stay unchanged. The restriction step keeps its original position in the workflow, so subsequent steps such as `add_relativities()` use the revised restricted coefficients.

The second column must retain the same name when an existing restriction is updated, because that name identifies the restricted model variable used by `refit()`. A message reports levels whose previously supplied relativity is changed.

Restricting a factor created by `add_relativities()`:

An `output_variable` introduced by an earlier `add_relativities()` step is already part of the ordered refinement specification. It is therefore not treated as a new external risk factor and does not require `allow_new_risk_factors = TRUE`. `add_restriction()` identifies the preceding relativity step from its stored metadata and replaces the corresponding derived tariff effect during `refit()`.

When only one level of such a refined variable is supplied, that level receives the new relativity and every other level is fixed at the relativity produced by `add_relativities()`. Mathematically, the resulting restriction therefore covers all current levels. Only the explicitly supplied level changes. This is useful when actuarial review supports a local adjustment but the remaining expert split should not be re-estimated.

Refinement order remains material. A restriction added after `add_relativities()` operates on the derived split relativities. A restriction added before `add_relativities()` instead changes the coefficient basis from which the split is derived.

Value

A `rating_refinement` object containing the stored restriction specification. The pricing GLM is not fitted again until `refit()` is called.

Author(s)

Martin Haringa

See Also

`prepare_refinement()`, `add_smoothing()`, `add_shrinkage()`, `add_rebasing()`, `add_relativities()`, `refit()`, `rating_table()`

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("C", "D"),
  relativity = c(1.10, 1.20)
)

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

# Postal area D was not observed in the portfolio. Its relativity is an
# explicit tariff assumption and becomes available after refitting.
refined_model <- refit(refined)
rating_table(refined_model, exposure = FALSE)

# A factor absent from the fitted GLM can replace an existing model term.
# The portfolio must already assign every observation to a hail zone.
portfolio$hail_zone <- factor(c("low", "high", "low", "high", "low", "high"))
hail_restrictions <- data.frame(
  hail_zone = c("low", "high"),
  hail_relativity = c(1.00, 1.20)
)

prepare_refinement(model, data = portfolio) |>
  add_restriction(
    hail_restrictions,
    replaces = "postal_area"
  )
# During refit(), hail_zone replaces postal_area rather than supplementing it.

# Without `replaces`, a new fixed factor supplements the existing terms.
# A later actuarial review changes only the relativity for the low hail zone.
# The high-zone relativity remains 1.20 and the existing step is updated.
revised_hail_restrictions <- data.frame(
  hail_zone = "low",
  hail_relativity = 1.10
)

hail_refinement <- prepare_refinement(model, data = portfolio) |>
  add_restriction(
    hail_restrictions,

```

```

    allow_new_risk_factors = TRUE
  ) |>
  add_restriction(revised_hail_restrictions)

  refit(hail_refinement)

```

add_shrinkage

Shrink categorical tariff relativities towards a common level

Description

Reduce differences between the relativities of one categorical risk factor before the refined GLM is fitted. `add_shrinkage()` combines each current relativity with a central level on the logarithmic scale. Extreme relativities move further in absolute terms, while their ordering is retained.

Usage

```
add_shrinkage(model, model_variable, credibility = 0.9, weights = NULL)
```

Arguments

model	A <code>rating_refinement</code> object created with <code>prepare_refinement()</code> . Shrinkage is applied to the current relativities at this point in the ordered refinement workflow.
model_variable	Character string naming the categorical risk factor to shrink. This may also identify a tariff factor created by an earlier <code>add_relativities()</code> step. After <code>add_restriction()</code> , use the first column of its restriction table; the second column is resolved internally.
credibility	Numeric scalar between 0 and 1. This is the weight given to the current risk-factor relativity. The remaining weight is assigned to the common centre. The default 0.9 retains 90 percent of the current logarithmic effect.
weights	NULL, "equal", or a character string naming a numeric, non-negative column in the refinement data. NULL derives the weighting basis from explicit model weights or a simple exposure offset. "equal" gives every level equal weight.

Details

Shrinkage can be used when the direction of a fitted risk-factor pattern is credible, but the difference between its highest and lowest relativities is considered too large for the available experience or the intended tariff. It is a structured actuarial adjustment rather than a new statistical fit.

For level i , the unnormalised adjusted relativity is

$$\tilde{r}_i = \exp\{Z \log(r_i) + (1 - Z) \log(c)\},$$

where r_i is the current relativity, Z is credibility, and c is the weighted geometric centre. A credibility of 1 leaves the relativities unchanged. A credibility of 0 removes the differences between levels.

The adjusted relativities are subsequently rescaled so that their weighted arithmetic mean equals the weighted arithmetic mean before shrinkage. With portfolio weights such as exposure or claim count, this prevents shrinkage itself from changing the weighted level of the risk factor. The final GLM refit may still change the intercept or other fitted quantities; use `audit_refinement()` to assess that combined portfolio effect.

Weight selection:

`weights = NULL` first uses explicit GLM weights when these were supplied during model fitting. Otherwise, a single column in an offset of the form `log(column)` is used. This commonly selects claim count for a weighted severity GLM and exposure for a frequency or risk-premium GLM. If neither source is unambiguous, the function asks for an explicit choice.

Set `weights` to a column name to control the basis directly. For example, exposure is generally appropriate for frequency or risk-premium relativities, while claim count is generally appropriate for severity relativities. Set `weights = "equal"` to give every risk-factor level the same weight. In that case the equal-level mean is preserved, which does not necessarily preserve the level of the observed portfolio.

Interpretation:

credibility is a user-supplied refinement parameter. It should not be interpreted as an automatically estimated Buhlmann or Buhlmann-Straub credibility factor. Its value should be supported by portfolio stability, validation over time and the intended degree of tariff differentiation. The selected value and weighting basis are retained in the refinement specification and shown by `summary()`.

Following a restriction:

When shrinkage follows `add_restriction()`, `model_variable` remains the first column of the restriction table: the categorical risk factor whose levels are being adjusted. The second column is a numeric implementation column containing the fixed relativities used during `refit()`; it is not a separate categorical risk factor. `add_shrinkage()` resolves that internal column automatically from the stored restriction metadata.

Value

A `rating_refinement` object containing an ordered shrinkage step. The returned object stores the original and adjusted relativities, level weights, inferred weight source and normalization information. The GLM is fitted only when `refit()` is called.

Author(s)

Martin Haringa

See Also

`prepare_refinement()`, `add_smoothing()`, `add_restriction()`, `add_rebasing()`, `add_relativities()`, `refit()`, `audit_refinement()`

Examples

```

portfolio <- data.frame(
  claims = c(1, 2, 1, 3, 2, 4, 1, 5),
  exposure = c(1, 1, 1, 1, 2, 1, 1, 1),
  sector = factor(rep(c("Industry", "Office", "Retail", "Transport"), 2))
)

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

refinement <- prepare_refinement(model, data = portfolio) |>
  add_shrinkage(
    model_variable = "sector",
    credibility = 0.9,
    weights = "exposure"
  )

summary(refinement)
refined_model <- refit(refinement)
rating_table(refined_model)

# Use equal level weights explicitly when portfolio weighting is not wanted.
equal_level_refinement <- prepare_refinement(model, data = portfolio) |>
  add_shrinkage(
    model_variable = "sector",
    credibility = 0.8,
    weights = "equal"
  )

```

add_smoothing

Smooth grouped tariff relativities in a refinement workflow

Description

Replace independently estimated relativities of an ordered, grouped model variable with a smooth tariff curve. This can reduce sampling variation between adjacent levels of risk factors such as age, vehicle age, insured value or bonus-malus years while retaining the broad effect estimated by the GLM.

Usage

```

add_smoothing(
  model,
  model_variable = NULL,
  source_variable = NULL,

```

```

    breaks,
    smoothing = "spline",
    k = NULL,
    degree = NULL,
    weights = NULL,
    tariff_class = NULL,
    rating_variable = NULL,
    x_cut = NULL,
    x_org = NULL
)

```

Arguments

model	Object of class <code>rating_refinement</code> , created with <code>prepare_refinement()</code> . A fitted GLM, including a model returned by <code>refit()</code> , is not accepted directly; retain and modify the corresponding refinement specification instead.
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. The column must not contain missing values; remove or impute missing values before adding the smoothing step.
source_variable	Character string. Original numeric portfolio variable underlying <code>model_variable</code> . Its name is also used for the resulting smoothed tariff variable. The column must contain only finite, non-missing numeric values.
breaks	Numeric vector with the tariff segment boundaries to use after smoothing. These boundaries determine the final tariff segmentation, not the number of portfolio observations used to estimate the curve. Values must be finite, strictly increasing and cover every observed value of <code>source_variable</code> . A boundary may extend beyond the interval range represented by <code>model_variable</code> , for example to create a rounded final tariff class. This does not itself produce a message. A short message is shown only when a representative point of a new interval lies outside the fitted GLM range and its relativity therefore requires extrapolation. This argument is required.
smoothing	Character string selecting the smoothing method. Available values are "spline" (default), "poly", "gam", "increasing", "decreasing", "convex", "concave", "increasing_convex", "increasing_concave", "decreasing_convex" and "decreasing_concave". The former short SCOP codes remain accepted as compatibility aliases. See Details for the statistical interpretation and shape restrictions.
k	Optional single positive whole number. Basis dimension for smoothing methods "spline", "gam", "increasing", "decreasing", "convex", "concave" and the combined direction-curvature methods. It sets the maximum flexibility available to the smooth and is not necessarily equal to its estimated effective degrees of freedom. NULL uses the smaller of 10 and the number of unique grouped model points. At least three unique grouped values are required. The basis dimension cannot exceed the number of unique grouped covariate values available for fitting.

degree	Optional single whole number. Polynomial degree, used only by smoothing = "poly". The degree must be feasible for the number of unique grouped model points.
weights	Optional character string. Numeric volume column, usually exposure, used to weight the grouped GLM relativities during smoothing.
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 specification on a `rating_refinement` object. It does not alter the fitted GLM immediately. The smoothing is evaluated in the recorded step order and applied when `refit()` is called. The original GLM contains `model_variable`, usually a factor created by grouping a continuous risk factor. `source_variable` identifies the original numeric variable represented by those groups.

The smoother is estimated from the fitted GLM relativities at the midpoint of each model interval. Consequently, the amount of information available to the smoother is primarily determined by the number of grouped model levels, rather than by the number of individual portfolio records. Exposure or another volume measure can be supplied through `weights` so that model levels with more portfolio volume have greater influence on the fitted curve.

The fitted curve is evaluated using breaks and converted back to a grouped tariff variable. The original model term is replaced by that smoothed tariff variable during refitting.

Actuarial interpretation:

Smoothing introduces a structural assumption: adjacent values of the source variable are expected to have related tariff effects. The selected method, basis dimension and breaks should therefore be assessed against exposure, observed experience, coefficient uncertainty and stability over time. A smooth curve should not be interpreted as evidence that the underlying risk relationship is itself known without uncertainty.

Smoothing methods:

The available methods represent different assumptions about the shape of the tariff effect:

- "spline" The general-purpose default. Fits an unconstrained penalized cubic regression spline. It is suitable when the tariff effect should be smooth but no monotonicity or curvature restriction is justified.
- "poly" Fits a global polynomial through the grouped GLM relativities. `degree` determines its order. A low degree gives a compact parametric trend; higher degrees can follow more local variation but may oscillate, particularly near the boundaries.
- "increasing" **and** "decreasing" Fit monotone smooths. These methods constrain the tariff effect to move in one direction, without imposing how quickly its slope changes. They are often the most directly interpretable constrained specifications when actuarial reasoning supports a consistently increasing or decreasing risk effect.
- "convex" **and** "concave" Constrain curvature but not direction. For a convex curve, the slope increases as the source variable increases; for a concave curve, the slope decreases. A convex curve may therefore be U-shaped and a concave curve may be inverted U-shaped. These are advanced choices when curvature itself has a defensible interpretation.

- "increasing_convex" **and** "increasing_concave" Fit increasing curves with an additional curvature constraint. An increasing convex effect rises at an increasing rate, for example when upper-tail risk causes marginal cost to accelerate. An increasing concave effect rises at a decreasing rate and gradually flattens, for example when risk cost rises with insured value but less than proportionally.
- "decreasing_convex" **and** "decreasing_concave" Fit decreasing curves with an additional curvature constraint. A decreasing convex effect becomes less steep and tends to flatten. A decreasing concave effect becomes progressively steeper.
- "gam" Fits an unconstrained thin-plate regression spline with `mgcv::gam()`. It is mainly intended as a flexible reference when comparing the general spline and shape-constrained specifications. It does not impose the actuarial shape assumptions represented by the constrained methods.

The shape-constrained methods are fitted with `scam::scam()`. Monotonicity concerns the direction of the effect, whereas convexity and concavity concern how its slope changes. In most tariff applications, a directional assumption is easier to substantiate than a curvature assumption. A constraint should reflect an actuarial or pricing assumption that is defensible for the risk factor; it should not be selected solely because it produces a smoother visual result. The combined monotonicity and curvature methods are advanced specifications and are most appropriate when both assumptions can be supported independently.

The former short codes "mpi", "mpd", "cx", "cv", "micx", "micv", "mdcx" and "mdcv" remain accepted as compatibility aliases. New code should use the readable method names above. Both forms produce the same smoothing specification.

Basis dimension and polynomial degree:

For "spline", "gam" and the shape-constrained methods, `k` specifies the basis dimension. It controls the maximum flexibility available to the smooth, but it is not the final effective degrees of freedom of the fitted curve. The estimated smoothing penalty can reduce the effective degrees of freedom below this maximum.

A smaller `k` restricts the curve to broad movements. A larger `k` permits more local variation, but requires enough distinct grouped covariate values and may be unstable when only a few tariff levels are available. If `k` is `NULL`, the function uses the smaller of 10 and the number of unique grouped model points. Spline, GAM and shape-constrained smoothing require at least three unique grouped values. The function checks this dimension before fitting and reports the observed number of unique values when the requested complexity is not feasible.

For "poly", `degree` has the corresponding complexity role. A polynomial of degree d requires at least $d + 1$ unique grouped values. When `degree` is omitted, the existing behaviour uses the highest degree supported by the grouped model points. In practice, an explicit low degree is generally preferable when a stable global trend is intended.

`degree` is only accepted for smoothing = "poly". Conversely, `k` is only accepted for "spline", "gam" and the shape-constrained methods. This separation prevents a complexity argument from being supplied but silently ignored.

The deprecated `smooth_coef()` wrapper remains available for backwards compatibility.

Value

An object of class `rating_refinement` containing the stored smoothing specification. The pricing GLM is not fitted again until `refit()` is called.

Author(s)

Martin Haringa

See Also

[prepare_refinement\(\)](#), [edit_smoothing\(\)](#), [add_restriction\(\)](#), [add_shrinkage\(\)](#), [add_rebasing\(\)](#), [add_relativities\(\)](#), [refit\(\)](#), [risk_factor_gam\(\)](#)

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,
  segmentation_penalty = 10,
  seed = 1
)

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 = c(seq(18, 93, 5), 95),
    smoothing = "spline",
    k = 6,
    weights = "exposure"
  )

  # When the tariff effect must not decrease, use the readable constrained
  # method name. The former value "mpi" remains accepted for compatibility.
  increasing_ref <- prepare_refinement(burn_unrestricted) |>
  add_smoothing(
    model_variable = "age_policyholder_freq_cat",
    source_variable = "age_policyholder",
    breaks = c(seq(18, 93, 5), 95),
    smoothing = "increasing",
    k = 6,
    weights = "exposure"
  )

  # Limit the visible range without changing the fitted smoothing curve.
  autoplot(ref, x_max = 80, y_max = 1.5)

## 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. The stored boundaries are applied to the original continuous risk-factor column, so the result does not depend on the row order used when the GAM was fitted.

The helper does not re-estimate the GAM or derive new boundaries. It can be used after filtering or reordering the original portfolio and on new data whose risk-factor values remain within the range used to derive the segmentation.

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 <code>derive_tariff_segments()</code> . 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.

Details

The risk-factor name and optional rounding increment are taken from segments. The risk-factor column in data must be numeric and contain only finite, non-missing values. Values outside the original segmentation range produce an error because their tariff treatment has not been supported by the fitted GAM. The resulting factor can be used in a GLM or retained as a candidate grouping for further actuarial review.

Value

A data frame with the derived tariff segment column added.

Author(s)

Martin Haringa

See Also

`risk_factor_gam()`, `derive_tariff_segments()`

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)
```

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, how many records contain claim amounts above candidate thresholds, how much historical claim cost sits above those thresholds, and how much risk premium would remain after capping claims at each threshold.

The function is intended for portfolio-level data as well as claim-level data. Portfolio-level data can include policies without claims, for example rows where `claim_count = 0` and the claim amount is zero. Use this before `redistribute_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 adjusted severity models.

Usage

```
assess_excess_threshold(
  data,
  claim_amount,
  thresholds,
  exposure = NULL,
  group = NULL,
  claim_count = NULL
)
```

Arguments

<code>data</code>	A data.frame with portfolio-level or claim-level observations. Portfolio-level data can include policies without claims.
<code>claim_amount</code>	Character string. Claim amount column.
<code>thresholds</code>	Numeric vector of candidate thresholds.
<code>exposure</code>	Optional character string. Exposure column. If supplied, risk premium before and after capping is calculated. The output column keeps this original name. If NULL, every record is counted as one exposure unit and the output contains an exposure column.
<code>group</code>	Optional character string. Grouping column used to assess thresholds by segment. The output column keeps this original name. If NULL, no grouping column is added.
<code>claim_count</code>	Optional character string. Claim-count column. If supplied, <code>n_claims</code> is calculated as the sum of this column. If NULL, records with <code>claim_amount > 0</code> are counted as one claim and records with <code>claim_amount == 0</code> as zero claims.

Details

The output can be used for two common follow-up analyses. First, aggregate the threshold assessment to portfolio level to calculate the average additional risk premium required to finance the excess layer. Second, after selecting a threshold, compare groups to see which parts of the portfolio benefit most from the excess protection.

Value

A data.frame with class "threshold_assessment" and columns:

group column The original grouping column, such as sector, if group is supplied. This is the first column when grouping is used.

threshold The excess threshold being assessed. Thresholds are shown in the same order as supplied in the thresholds argument.

exposure column The original exposure column, such as policy_years, if exposure is supplied. If exposure = NULL, this column is named exposure and counts records.

n_claims Total number of claims, calculated from claim_count or inferred from claim_amount > 0.

n_excess_records Number of records with claim_amount > threshold. This counts records, not individual claims.

total_loss Total claim amount before applying the threshold.

capped_loss Total claim amount retained below or at the threshold.

excess_loss Total claim amount above the threshold.

pure_premium_before Risk premium before capping: total_loss / exposure.

pure_premium_after Risk premium after capping: capped_loss / exposure.

premium_reduction pure_premium_before - pure_premium_after, equivalent to excess_loss / exposure. This is positive when applying the threshold reduces the retained risk premium.

premium_reduction_ratio premium_reduction / pure_premium_before. This is between 0 and 1 when pure_premium_before > 0; if pure_premium_before == 0, it is defined as 0.

Author(s)

Martin Haringa

Examples

```
portfolio <- data.frame(
  policy_id = 1:10,
  sector = rep(c("Industry", "Retail"), each = 5),
  claim_count = c(
    0, 1, 1, 1, 1,
    0, 1, 1, 1, 1
  ),
  claim_amount = c(
    0, 25000, 120000, 50000, 175000,
    0, 40000, 90000, 150000, 300000
  )
)
```

```

    ),
    policy_years = rep(1, 10)
  )

thresholds <- assess_excess_threshold(
  data = portfolio,
  claim_amount = "claim_amount",
  thresholds = c(25000, 50000, 100000, 150000),
  exposure = "policy_years",
  group = "sector",
  claim_count = "claim_count"
)

thresholds
if (requireNamespace("gt", quietly = TRUE)) {
  as_gt(thresholds)
}

# Calculate the average additional risk premium required to finance
# the excess portion of the claims.
thresholds |>
  dplyr::summarise(
    policy_years = sum(policy_years),
    excess_loss = sum(excess_loss),
    capped_loss = sum(capped_loss),
    extra_risk_premium = excess_loss / policy_years,
    risk_premium_increase = excess_loss / capped_loss,
    .by = "threshold"
  )

# After selecting a threshold, compare which groups benefit most
# from the excess protection.
selected_threshold <- thresholds |>
  dplyr::filter(threshold == 100000) |>
  dplyr::select(
    sector,
    threshold,
    policy_years,
    n_claims,
    n_excess_records,
    premium_reduction,
    premium_reduction_ratio
  ) |>
  dplyr::arrange(dplyr::desc(premium_reduction_ratio))

selected_threshold

# If claim_count is omitted, records with positive claim amounts are counted.
assess_excess_threshold(
  data = portfolio,
  claim_amount = "claim_amount",
  thresholds = 100000,
  exposure = "policy_years",

```

```

    group = "sector"
  )

```

as_gt

Convert an object to a gt table

Description

Generic presentation helper. Methods return a gt table for objects where a formatted reporting table is more useful than another plot.

Create a formatted gt table from an object returned by [assess_excess_threshold\(\)](#). The original object remains a regular data.frame subclass; as_gt() is only used when a presentation table is needed for a report, tariff note or pricing review.

Format the coefficient-level result from [bootstrap_coefficients\(\)](#) as a gt table. The table retains the requested link or exponentiated scale and shows how many bootstrap estimates were available for each coefficient.

Create a formatted gt table from an object returned by [rating_table\(\)](#). Risk factors are presented as row groups, while fitted model effects are shown as relativities or coefficients depending on the scale selected in [rating_table\(\)](#).

Usage

```

as_gt(x, ...)

## S3 method for class 'threshold_assessment'
as_gt(
  x,
  claims = TRUE,
  loss = FALSE,
  premium = TRUE,
  locale = "nl-NL",
  loss_decimals = 0,
  premium_decimals = 0,
  ratio_decimals = 1,
  color_last_column = TRUE,
  title = NULL,
  subtitle = NULL,
  ...
)

## S3 method for class 'bootstrap_coefficients'
as_gt(
  x,
  scale = c("link", "exponentiated", "relativity"),
  confidence = 0.95,

```

```

    interval = c("percentile", "normal"),
    locale = "nl-NL",
    estimate_decimals = 3,
    success_decimals = 1,
    title = NULL,
    subtitle = NULL,
    ...
)

## S3 method for class 'rating_table'
as_gt(
  x,
  significance = NULL,
  show_effect_spanner = NULL,
  model_labels = NULL,
  locale = "nl-NL",
  estimate_decimals = 3,
  exposure_decimals = 0,
  missing_text = "-",
  title = NULL,
  subtitle = NULL,
  ...
)

```

Arguments

x	A supported object to convert, such as a threshold_assessment, rating_table, bootstrap_coefficients or refinement_audit object.
...	Arguments passed to methods.
claims	Logical. If TRUE, include claim-count columns.
loss	Logical. If TRUE, include loss amount columns. The default is FALSE to keep the threshold comparison compact.
premium	Logical. If TRUE, include risk-premium and premium-reduction columns.
locale	Character. Locale used to format model effects and exposure, for example "nl-NL" or "en-US".
loss_decimals, premium_decimals, ratio_decimals	Non-negative whole numbers controlling displayed decimals for loss amounts, premium amounts and percentage ratios.
color_last_column	Logical. If TRUE, color the final displayed column from white to yellow so the highest values stand out in the presentation table.
title	Optional character. Table title. If NULL, no title is added.
subtitle	Optional character. Table subtitle. If NULL, no subtitle is added.
scale	Character string. "link" reports coefficients on their fitted GLM scale. "exponentiated" applies exp() to every original and bootstrap coefficient. "relativity" is an alias for "exponentiated"; this interpretation is most direct for a log-link

	GLM. For a logit-link model, exponentiated coefficients are odds ratios rather than response probabilities.
confidence	Numeric scalar between 0 and 1 giving the confidence level.
interval	Character string. "percentile" uses empirical bootstrap quantiles. "normal" uses the original estimate plus or minus a normal quantile times the bootstrap standard error.
estimate_decimals	Non-negative whole number. Number of decimals shown for fitted coefficients or relativities.
success_decimals	Non-negative whole number. Number of decimals for the success-rate percentage.
significance	Optional logical. If NULL, use the significance setting stored on x. If TRUE, append the stored significance stars to the model effects and add the significance-level note. If FALSE, show fitted effects without stars.
show_effect_spanner	Optional logical. If NULL, show the "Relativities" or "Coefficients" spanner when multiple models are present and omit it for a single model. Use TRUE or FALSE to override this behaviour.
model_labels	Optional character vector with display labels for the fitted models. By default, each model object name is used unchanged. An unnamed vector is matched to the model columns in their existing order. A named vector can map model object names to labels, for example <code>c(freq = "Frequency", sev = "Severity")</code> .
exposure_decimals	Non-negative whole number. Number of decimals shown for the exposure column, when available.
missing_text	Single character string used to display missing values. The default is an en dash ("\\u2013") so structural missing values, such as exposure for the intercept, are visually distinct from observed zero values.

Details

The first column of a `rating_table` identifies the model risk factor. `as_gt()` uses this column as `groupname_col` and sets `row_group_as_column = TRUE`. Levels belonging to the same risk factor are therefore kept together in a compact format suitable for a tariff note, model review or technical appendix.

Risk-factor and level order are taken directly from `rating_table()`. Use its `risk_factor_order`, `level_order`, `numeric_level_order`, `reference_first` and `order_model` arguments to determine the order before formatting the table.

With `significance = TRUE`, significance stars are appended to the fitted effects and the significance levels are shown below the table. This requires an object originally created with `rating_table(significance = TRUE)`, because p-value information is deliberately not retained when significance is disabled during table construction. Significance stars are a statistical diagnostic and should be interpreted together with exposure, effect size, model stability and actuarial relevance.

In the underlying `rating_table`, estimates and significance indicators are stored in separate `est_*` and `signif_*` columns. `as_gt()` merges each pair only for display. The estimates therefore remain numeric in the source object, including when several models are presented in one table.

Value

A gt_tbl object for supported methods.

Author(s)

Martin Haringa

Examples

```
portfolio <- data.frame(
  policy_id = 1:10,
  sector = rep(c("Industry", "Retail"), each = 5),
  claim_count = c(
    0, 1, 1, 1, 1,
    0, 1, 1, 1, 1
  ),
  claim_amount = c(
    0, 25000, 120000, 50000, 175000,
    0, 40000, 90000, 150000, 300000
  ),
  policy_years = rep(1, 10)
)

thresholds <- assess_excess_threshold(
  data = portfolio,
  claim_amount = "claim_amount",
  thresholds = c(25000, 50000, 100000, 150000),
  exposure = "policy_years",
  group = "sector",
  claim_count = "claim_count"
)

if (requireNamespace("gt", quietly = TRUE)) {
  as_gt(thresholds)
}

portfolio <- MTPL
portfolio$zip <- as.factor(portfolio$zip)

frequency_model <- glm(
  nclaims ~ bm + zip + offset(log(exposure)),
  family = poisson(),
  data = portfolio
)

fitted_tariff <- rating_table(
  frequency_model,
  model_data = portfolio,
  exposure = "exposure",
  significance = TRUE
)
```

```

if (requireNamespace("gt", quietly = TRUE)) {
  as_gt(fitted_tariff)
  as_gt(fitted_tariff, model_labels = "Frequency model")
  as_gt(fitted_tariff, significance = FALSE, locale = "en-US")
}

```

audit_refinement	<i>Audit the effect of a fitted model refinement</i>
------------------	--

Description

Compare an unrestricted GLM with the model returned by `refit()` on the same observed portfolio. The audit records the refinement specification and quantifies how the fitted response or fitted rate changes for the portfolio and for each final tariff-factor level.

Usage

```

audit_refinement(
  object,
  exposure = NULL,
  risk_factors = NULL,
  scale = c("auto", "response", "per_exposure"),
  metric = NULL
)

```

Arguments

object	A fitted model returned by <code>refit()</code> . Ordinary GLMs do not contain the stored baseline and refinement metadata required for the comparison.
exposure	Optional character string naming the exposure column. With <code>scale = "auto"</code> , the function attempts to infer a single exposure column from the original model offset. Supply this argument explicitly when that interpretation is ambiguous.
risk_factors	Optional character vector identifying final tariff factors for the level comparison. If <code>NULL</code> , the final factors reported by <code>rating_table()</code> and available in <code>rating_grid()</code> are used.
scale	Character string. <code>"per_exposure"</code> compares fitted values per unit of exposure. <code>"response"</code> compares predictions on the response scale. <code>"auto"</code> selects <code>"per_exposure"</code> when one exposure variable can be identified from the model offset and otherwise selects <code>"response"</code> .
metric	Optional character string used to describe the audited measure, for example <code>"risk_premium"</code> , <code>"frequency"</code> or <code>"average_severity"</code> . If <code>NULL</code> , the audit uses <code>"fitted_rate"</code> or <code>"fitted_response"</code> .

Details

Direct coefficient comparisons are generally not a sufficient refinement audit. A coefficient can change because the intercept or another model term changes, while the combined fitted value for a policy remains similar. `audit_refinement()` therefore compares predictions from the original and refined models on common observed model-point combinations.

The model points are obtained with `rating_grid()`. Portfolio and level results are weighted by the number of records or, when supplied, by exposure. With `scale = "per_exposure"`, predictions that include an exposure offset are divided by total exposure after aggregation. This gives an exposure-weighted fitted rate rather than an unweighted average over unique model points.

The resulting measure should be named according to the model being audited. For a frequency model it is normally a fitted frequency; for a severity model it is a fitted average severity; and for a direct risk-premium model it can be labelled `"risk_premium"`. A complete risk-premium comparison requires either a direct risk-premium model or an explicit combination of frequency and severity predictions.

Value

An object of class `refinement_audit`. The object contains package and model metadata, the ordered refinement steps, portfolio-level results, results by risk factor and level, and the model points used in the calculation. Use `summary.refinement_audit()` for a concise audit report, `as.data.frame()` for the level results and `as_gt()` for a formatted table.

Author(s)

Martin Haringa

See Also

`prepare_refinement()`, `refit()`, `rating_grid()`, `summary.rating_refinement()`

Examples

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

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

refinement <- prepare_refinement(base_model, data = portfolio) |>
  add_restriction(data.frame(
    risk_class = "B",
    risk_class_restricted = 1.15
  ))
```

```
summary(refinement)

refined_model <- refit(refinement)
audit <- audit_refinement(
  refined_model,
  exposure = "exposure",
  metric = "frequency"
)

summary(audit)
as.data.frame(audit)

if (requireNamespace("gt", quietly = TRUE)) {
  as_gt(audit)
}
```

```
autoplot.bootstrap_performance
```

Plot the resampled performance distribution

Description

Display the empirical distribution of resampled RMSE values from [bootstrap_performance\(\)](#). The histogram shows the observed resampling distribution, while the density curve provides a smooth visual summary.

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 colour of the histogram bars. Default is "#E6E6E6".
color	Border colour of the histogram bars. Default is NA, which removes bar borders.
...	Currently unused.

Details

The dashed orange line marks the RMSE of the original fitted model. The dotted grey lines mark the 2.5 and 97.5 percent empirical quantiles of the resampled values when these can be calculated. Their distance provides a practical indication of sampling variability; it is not a formal prediction interval for future portfolio performance.

Value

A [ggplot2::ggplot](#) object.

Author(s)

Martin Haringa

See Also[bootstrap_performance\(\)](#), [rmse\(\)](#)**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

Inspect simulation-based residual uniformity

Description

Plot the scaled residuals returned by [check_residuals\(\)](#) against their theoretical uniform quantiles. Systematic departures from the diagonal can indicate remaining structure, distributional mismatch or influential observations in a fitted pricing model.

Usage

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

Arguments

object	An object produced by check_residuals() .
show_message	Logical. If TRUE, print a concise interpretation of the Kolmogorov-Smirnov p-value.
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 display all points.
...	Currently unused.

Details

The subtitle reports the uniformity-test p-value stored in object. This p-value is a diagnostic signal rather than a stand-alone model acceptance criterion. The shape and location of deviations should be assessed together with exposure, fitted values and relevant risk-factor levels.

Reducing `max_points` affects only the displayed QQ points; it does not change the residual calculation or the reported test.

Value

A `ggplot2::ggplot` object.

Author(s)

Martin Haringa

See Also

`check_residuals()`, `check_overdispersion()`

`autoplot.factor_analysis`

Plot observed portfolio experience by risk factor

Description

Visualise the exposure and observed actuarial metrics calculated by `factor_analysis()`. The plots support assessment of portfolio composition, observed frequency, severity, risk premium and loss ratio across risk-factor levels.

The observed patterns are descriptive. They do not adjust for correlations with other risk factors and should therefore not be interpreted as multivariate tariff relativities.

Usage

```
## S3 method for class 'factor_analysis'
autoplot(
  object,
  metrics = NULL,
  ncol = 1,
  legend_position = c("right", "bottom", "top", "left", "none"),
  show_exposure = TRUE,
  show_exposure_labels = TRUE,
  sort_by_exposure = FALSE,
  level_order = NULL,
  decimal_mark = ",",
  line_color = NULL,
  bar_fill = NULL,
```

```

abbreviate_labels = TRUE,
label_width = 20,
label_abbreviations = NULL,
flipBars = 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 <code>factor_analysis()</code> or <code>univariate()</code> .
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 (claim_amount / claim_count) • 3. Risk premium (claim_amount / exposure) • 4. Loss ratio (claim_amount / 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	Positive whole number. Number of columns in the plot composition.
legend_position	Character string specifying the legend position. Supported values are "right", "bottom", "top", "left" and "none". A legend is present only when the

factor_analysis object was created with group_by, because the resulting group-specific series are distinguished by colour. Without group_by, colours are fixed plot styles and no legend is drawn; in that case this argument has no visible effect.

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.
abbreviate_labels	Logical. If TRUE, long risk-factor level labels are shortened to label_width characters. A shortened label ends in one period; for example, "Bouwnijverheid" becomes "Bouwn ." when label_width = 6. Only the displayed axis labels are changed.
label_width	Positive whole number of at least 2. Maximum number of characters in automatically shortened level labels.
label_abbreviations	Optional named character vector with explicit display labels, for example c("Bouwnijverheid" = "Bouwn .", "Onroerend goed" = "Onr . goed"). Explicit labels take precedence over automatic shortening.
flipBars	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.
...	Currently unused.

Details

For rate-based metrics, exposure can be shown as background bars so that observed level differences can be interpreted together with portfolio volume. Levels with little exposure or few claims can produce volatile observed metrics and should be assessed accordingly.

When the factor analysis contains one or more by variables, separate observed series are shown. `show_total = TRUE` adds the aggregate portfolio series for comparison. Sorting and manual level ordering affect only the presentation; the underlying summaries are unchanged.

A rate or ratio is not shown for a risk-factor level when its denominator is zero or missing. For example, average severity is undefined when claim count is zero, and loss ratio is undefined when premium is zero. `autoplot()` gives one combined warning identifying the affected metrics and levels. Other valid metrics and exposure information remain in the figure.

Value

A patchwork composition containing the requested ggplot panels.

Author(s)

Marc Haïne, Martin Haringa

See Also

[factor_analysis\(\)](#), [add_portfolio_experience\(\)](#)

Examples

```
# Plot observed frequency and risk premium
x <- factor_analysis(MTPL2,
  x = "area",
  severity = "amount",
  nclaims = "nclaims",
  exposure = "exposure")
autoplot(x, metrics = c("frequency", "risk_premium"))
```

autoplot.rating_refinement

Inspect a model refinement step

Description

Visualise one stored step of a `rating_refinement` specification before fitting the revised GLM with `refit()`. The plot compares the original fitted tariff effect with the smoothing, restriction, shrinkage, rebasing or sublevel relativity specification produced by the selected step.

Usage

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

Arguments

<code>object</code>	Object of class <code>rating_refinement</code> .
<code>variable</code>	Optional character string identifying the model or derived variable whose refinement step should be shown. For one smoothing lineage, the most recent smoothing or edit step is selected. An error is returned when no step matches or when matches belong to different refinements.
<code>step</code>	Optional positive integer identifying a step in the stored refinement sequence. This takes precedence over <code>variable</code> .

<code>x_max</code>	Optional single finite numeric value. Maximum value displayed on the x-axis of a smoothing plot. This changes only the visible plotting range; it does not remove observations, alter the fitted smoothing curve or affect <code>refit()</code> . It is useful when a small number of extreme values would otherwise compress the range containing most portfolio risks. For example, use <code>x_max = 1e7</code> to display insured values up to 10 million. This argument is only available for smoothing steps.
<code>y_max</code>	Optional single finite numeric value. Maximum relativity displayed on the y-axis of a smoothing plot. Like <code>x_max</code> , this changes only the visible plotting range and does not alter the smoothing fit, refinement data or <code>refit()</code> . This argument is only available for smoothing steps.
<code>show_initial_smoothing</code>	Logical. For a smoothing or smoothing-edit plot, whether to overlay the initial curve produced by the corresponding <code>add_smoothing()</code> step. The other smoothing line shows the cumulative curve at the selected step. Default is FALSE. This argument does not alter the refinement specification or <code>refit()</code> .
<code>show_segments</code>	Logical. For a smoothing or smoothing-edit relativity plot, whether to show the horizontal relativities and boundary points of the new tariff segments. Set this to FALSE to inspect the continuous smoothing curve without the segmented tariff representation. The original fitted model effects remain visible. Default is TRUE.
<code>remove_underscores</code>	Logical; if TRUE, underscores are replaced by spaces in the x-axis label. Default is FALSE.
<code>rotate_angle</code>	Optional numeric value for the angle of x-axis labels.
<code>custom_theme</code>	Optional list passed to <code>ggplot2::theme()</code> .
<code>...</code>	Additional plotting arguments passed to <code>ggplot2</code> geoms.

Details

Refinement steps are evaluated in their stored order up to and including the selected step. The plot is a diagnostic preview: it does not refit the GLM and does not modify the refinement specification.

If `step` is supplied, that position in the refinement sequence is shown. If only `variable` is supplied, the most recent step in one smoothing lineage is used. For other refinement types, exactly one stored step must match that variable. When neither is supplied, the object must contain exactly one refinement step. Otherwise the function asks the user to select a step explicitly.

Each `edit_smoothing()` call is stored as a separate workflow step. Selecting such a step shows the cumulative smoothing after all preceding edits up to that point. Set `show_initial_smoothing = TRUE` to add the curve produced by the corresponding `add_smoothing()` step before any edits were applied.

Actuarial interpretation:

The plot supports review of the proposed tariff structure before estimation. It can be used to assess the local shape and magnitude of a smoothing curve, the effect of fixed relativities, and the differentiation introduced within a broader GLM level. This visual assessment does not by itself establish statistical adequacy; claim volume, exposure, stability over time and model diagnostics should also be considered.

For a sublevel split created by `add_relativities()`, the original parent level is shown as a horizontal segment across its child levels. This makes the parent GLM effect and the proposed within-level differentiation directly comparable.

Value

A ggplot2 object.

Author(s)

Martin Haringa

See Also

`prepare_refinement()`, `add_smoothing()`, `edit_smoothing()`, `add_restriction()`, `add_shrinkage()`, `add_rebasing()`, `add_relativities()`, `refit()`

Examples

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

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

refinement <- prepare_refinement(model, data = portfolio) |>
  add_restriction(data.frame(
    risk_class = "C",
    risk_class_restricted = 1.10
  ))

autoplot(refinement)
```

autoplot.rating_table *Compare fitted risk-factor effects graphically*

Description

Plot the coefficients or relativities stored in a `rating_table()` object by risk factor. Multiple fitted models can be compared, exposure can be shown as background bars, and observed portfolio experience attached with `add_portfolio_experience()` can be added as a separate line.

Usage

```
## S3 method for class 'rating_table'
autoplot(
  object,
  risk_factors = NULL,
  metric = NULL,
  ncol = 1,
  legend_position = c("auto", "right", "bottom", "top", "left", "none"),
  show_exposure_labels = TRUE,
  decimal_mark = ",",
  y_label = "Relativity",
  bar_fill = NULL,
  model_color = NULL,
  use_linetype = FALSE,
  abbreviate_labels = TRUE,
  label_width = 20,
  label_abbreviations = NULL,
  rotate_angle = NULL,
  custom_theme = NULL,
  remove_underscores = FALSE,
  labels = NULL,
  dec.mark = NULL,
  ylab = NULL,
  fill = NULL,
  color = NULL,
  linetype = NULL,
  ...
)
```

Arguments

<code>object</code>	A "rating_table" object returned by <code>rating_table()</code> .
<code>risk_factors</code>	Optional character vector specifying the risk factors to plot. If NULL, all available risk factors are shown.
<code>metric</code>	Optional character string. Observed-experience metric to plot when observed experience has been attached with <code>add_portfolio_experience()</code> . Common choices are "frequency", "severity"/"average_severity" and "risk_premium".
<code>ncol</code>	Positive integer specifying the number of columns in the patchwork layout.
<code>legend_position</code>	Character string specifying the legend position. The default, "auto", hides the legend when only one fitted model is shown and no observed-experience line is present. It places the legend on the right when multiple fitted models or an observed-experience comparison are shown. Use "right", "bottom", "top", "left" or "none" to override this behaviour.
<code>show_exposure_labels</code>	Logical. If TRUE, print exposure values on the background bars.
<code>decimal_mark</code>	Character string, either ",", " or ". ", controlling number labels.

<code>y_label</code>	Character string for the primary y-axis.
<code>bar_fill</code>	Optional colour for exposure bars. If NULL, the package palette is used.
<code>model_color</code>	Optional single colour overriding the model-line palette.
<code>use_linetype</code>	Logical. If TRUE, distinguish fitted models by line type as well as colour.
<code>abbreviate_labels</code>	Logical. If TRUE, long risk-factor level labels are shortened to <code>label_width</code> characters. A shortened label ends in one period; for example, "Bouwnijverheid" becomes "Bouwn." when <code>label_width = 6</code> . Only the displayed axis labels are changed.
<code>label_width</code>	Positive whole number of at least 2. Maximum number of characters in automatically shortened level labels.
<code>label_abbreviations</code>	Optional named character vector with explicit display labels, for example <code>c("Bouwnijverheid" = "Bouwn.", "Onroerend goed" = "Onr. goed")</code> . Explicit labels take precedence over automatic shortening.
<code>rotate_angle</code>	Optional numeric angle for risk-factor level labels.
<code>custom_theme</code>	Optional named list passed to <code>ggplot2::theme()</code> .
<code>remove_underscores</code>	Logical. If TRUE, replace underscores with spaces in risk-factor axis labels.
<code>labels</code>	Deprecated alias for <code>show_exposure_labels</code> .
<code>dec.mark</code>	Deprecated alias for <code>decimal_mark</code> .
<code>ylab</code>	Deprecated alias for <code>y_label</code> .
<code>fill</code>	Deprecated alias for <code>bar_fill</code> .
<code>color</code>	Deprecated alias for <code>model_color</code> .
<code>linetype</code>	Deprecated alias for <code>use_linetype</code> .
<code>...</code>	Additional arguments reserved for method compatibility.

Details

Plot contents:

One panel is produced for each selected risk factor. Model effects use the primary y-axis. When exposure is available, bars are rescaled to the plotting range and the original exposure scale is shown on the secondary y-axis. Panel and level order follow the input `rating_table()` object. This keeps the reference level and any explicit actuarial review order consistent between the data frame, `as_gt()` and the plot.

Observed experience is plotted only after it has been attached with `add_portfolio_experience()`. The selected metric is converted to the relative scale recorded in that object, using either the model reference level or the portfolio mean.

Actuarial interpretation:

The plot supports comparison of fitted tariff effects, portfolio volume and unadjusted observed experience. Differences between the observed and modelled lines may indicate portfolio-mix effects, sparse levels, model smoothing or genuine lack of fit. The chart does not separate these

explanations and should be reviewed together with claim counts, residual diagnostics and stability across periods.

When models are compared, the analyst should ensure that response definitions, link functions and relativity scales are sufficiently comparable. Exposure bars provide volume context but are not confidence intervals.

Value

A patchwork object containing one ggplot2 panel per selected risk factor.

Author(s)

Martin Haringa

See Also

[rating_table\(\)](#), [add_portfolio_experience\(\)](#), [factor_analysis\(\)](#), [as_gt.rating_table\(\)](#)

Examples

```
portfolio <- MTPL
portfolio$zip <- as.factor(portfolio$zip)

frequency <- glm(
  nclaims ~ bm + zip + offset(log(exposure)),
  family = poisson(),
  data = portfolio
)

effects <- rating_table(
  frequency,
  model_data = portfolio,
  exposure = "exposure"
)

autoplot(effects, risk_factors = "zip", show_exposure_labels = FALSE)
```

autoplot.tariff_segments

Inspect smooth risk-factor effects and tariff-segment boundaries

Description

Plot the smooth effect estimated by [risk_factor_gam\(\)](#) or inspect that same effect together with candidate boundaries returned by [derive_tariff_segments\(\)](#). Both methods use the same curve, confidence interval, observation and axis layers.

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,
  x_stepsize = NULL,
  show_segments = TRUE,
  ...
)

## 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

<code>object</code>	An object returned by <code>risk_factor_gam()</code> or <code>derive_tariff_segments()</code> .
<code>confidence</code>	Logical. If TRUE, add pointwise 95 percent confidence intervals where finite values are available.
<code>color_gam</code>	Colour for the fitted GAM line.
<code>show_observations</code>	Logical. If TRUE, add the aggregated observed experience used for fitting.
<code>color_splits</code>	Colour for segment boundaries. Used only for a <code>tariff_segments</code> object.
<code>size_points</code>	Numeric point size for observed experience.
<code>color_points</code>	Colour for observed experience.
<code>rotate_labels</code>	Logical. If TRUE, rotate x-axis labels by 45 degrees.

remove_outliers	Optional single numeric upper display limit for observed points. The fitted curve remains unchanged.
conf_int	Deprecated. Use confidence instead.
x_stepsize	Optional positive numeric step size for x-axis tick marks. If NULL, breaks are determined automatically.
show_segments	Logical. For a tariff_segments object, show the candidate segment boundaries when TRUE. Default is TRUE.
...	Additional arguments reserved for method compatibility.

Details

The fitted line is shown on its natural response scale: claim frequency, average severity or risk premium. Optional observed points represent portfolio experience aggregated at the continuous risk-factor values used for fitting.

For a tariff_segments object, vertical lines show the derived interval boundaries. These lines support actuarial review of where the continuous effect changes sufficiently to motivate a categorical tariff treatment. Set show_segments = FALSE to inspect only the underlying smooth curve.

Confidence intervals describe uncertainty in the fitted curve conditional on the selected GAM specification. They do not include uncertainty from model selection, omitted risk factors or future portfolio changes. Segment boundaries do not by themselves demonstrate that adjacent segments are statistically or commercially distinct. Exposure, claim volume, temporal stability and operational tariff constraints should be considered separately.

remove_outliers affects displayed observed points only. It does not remove observations from the fitted GAM or alter the prediction curve or segment boundaries.

Value

A ggplot2 object.

Author(s)

Martin Haringa

See Also

[risk_factor_gam\(\)](#), [derive_tariff_segments\(\)](#), [add_tariff_segments\(\)](#)

Examples

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

```
# Inspect the continuous effect before deriving tariff segments.
autoplot(fit, confidence = TRUE, show_observations = TRUE)

segments <- derive_tariff_segments(
  fit,
  segmentation_penalty = 10,
  seed = 1
)

# Inspect the same effect with the candidate segment boundaries.
autoplot(segments, confidence = TRUE, show_observations = TRUE)
autoplot(segments, show_segments = FALSE)

## End(Not run)
```

```
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.

The comparison assesses whether the fitted conditional severity distribution represents the shape of the observed claims within the same truncation interval.

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

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

Systematic separation between the empirical and fitted curves can indicate that the selected gamma or lognormal distribution does not adequately represent the observed severity shape. The plot does not assess the unobserved parts of the distribution outside the truncation bounds.

Value

A `ggplot2` object.

Author(s)

Martin Haringa

See Also

[fit_truncated_severity\(\)](#), [rlnormt\(\)](#), [rgammat\(\)](#)

bootstrap_coefficients

Assess GLM coefficient stability by portfolio-row bootstrap

Description

Refit a GLM on repeated bootstrap samples of the estimation portfolio and retain the coefficient estimates from every successful refit. The resulting distribution describes how sensitive individual model coefficients are to sampling variation in the observed portfolio.

Usage

```
bootstrap_coefficients(
  object,
  n_resamples = 500,
  seed = NULL,
  show_progress = interactive()
)
```

Arguments

object	A fitted glm object. Refined GLMs are accepted when their estimation data can be recovered from the model object.
n_resamples	Positive whole number. Number of bootstrap samples. Default is 500.
seed	Optional single numeric seed for reproducible resampling.
show_progress	Logical. If TRUE, display a text progress bar.

Details

Each resample contains the same number of portfolio rows as the original estimation data and is drawn with replacement. The function recovers these data from `object`; a separate data argument is deliberately not required. Rows omitted during the original model fit are excluded so the resampling population remains aligned with the fitted GLM.

Original factor levels, the model formula, offsets and model weights are retained during refitting. A factor level may nevertheless be absent from a particular bootstrap sample. Its coefficient can then be non-estimable and is stored as NA for that replicate.

A failed or non-converged GLM refit does not stop the procedure. The failed replicate is recorded and the function continues. After resampling, an informative message reports how many requested refits produced usable model objects. `summary.bootstrap_coefficients()` reports the number of finite estimates separately for each coefficient.

Actuarial interpretation:

The bootstrap distribution can identify tariff effects that are sensitive to the particular portfolio sample. Wide intervals, material bootstrap bias or a low number of estimable replicates often indicate sparse levels, correlated model terms or limited claim information. These diagnostics

should be considered alongside exposure, claim counts, coefficient interpretation and stability across calendar periods.

The row bootstrap represents sampling variation in the observed estimation portfolio. It does not include future trend, parameter uncertainty caused by model selection, structural changes in portfolio composition or dependence between repeated records for the same policy. Where such dependence is material, a cluster-level bootstrap would require a different resampling design.

Value

An object of class "bootstrap_coefficients". It contains the original coefficients, a coefficient matrix with one row per requested resample, indicators for successful model fits, recorded failure messages, and the resampling settings. Use `summary.bootstrap_coefficients()` for a coefficient-level data frame and `as_gt()` for a formatted table.

Author(s)

Martin Haringa

See Also

`summary.bootstrap_coefficients()`, `bootstrap_performance()`, `model_performance()`, `as_gt()`

Examples

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

boot <- bootstrap_coefficients(
  model,
  n_resamples = 25,
  seed = 123,
  show_progress = FALSE
)

summary(boot, scale = "link")
summary(boot, scale = "exponentiated")
summary(boot, scale = "relativity")

if (requireNamespace("gt", quietly = TRUE)) {
  as_gt(boot, scale = "relativity")
}
```

bootstrap_performance *Assess performance stability under repeated resampling*

Description

Refit a pricing model on repeated samples and record the resulting response-scale prediction error. The distribution of RMSE values describes how sensitive model performance is to changes in the observed portfolio sample.

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 that can be updated on resampled data.
data	Data frame containing the model response and predictors.
n_resamples	Positive whole number. Number of resampling replicates. Default is 50.
sample_fraction	Fraction of the data used in the training sample. Must be in (0, 1]. Default is 1.
metric	Character string. Performance metric to compute. Currently only "rmse" is supported.
sampling	Character string. 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 a progress bar during resampling. Default is TRUE.
rmse_model	Optional finite numeric RMSE for the original fitted model. If NULL, it is calculated from model and data.
n, frac	Deprecated argument names. Use n_resamples and sample_fraction instead.

Details

Resampling design:

With `sampling = "bootstrap"`, training rows are sampled with replacement. With `sampling = "split"`, they are sampled without replacement. When `sample_fraction < 1`, performance is evaluated on records not used for fitting. When `sample_fraction = 1`, performance is evaluated on the sampled training data and should be interpreted as an in-sample stability measure.

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.

Actuarial interpretation:

The resampled RMSE distribution is useful for comparing the stability of alternative frequency, severity or risk-premium specifications under repeated portfolio sampling. A narrow distribution indicates that the measured error is relatively insensitive to the sampled records; a wide distribution indicates greater sampling sensitivity.

This is an experience-based diagnostic and does not by itself represent the full uncertainty in future claims, trend, portfolio mix or model specification. Sparse factor levels are retained in training samples where necessary to avoid new-level prediction failures. That protection is useful operationally, but should be considered when interpreting the resampling design.

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

See Also

`rmse()`, `model_performance()`, `autoplot.bootstrap_performance()`

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)
```

calibrate_model	<i>Calibrate the overall level of a refined pricing model</i>
-----------------	---

Description

Adjust the overall prediction level of a fitted model returned by `refit()` without re-estimating its relative tariff structure. Calibration is a final model-level operation: all refinement decisions must be completed before calling `calibrate_model()`.

Usage

```
calibrate_model(model, factor)
```

Arguments

model	A fitted refined GLM returned by <code>refit()</code> . It must inherit from <code>refitrestricted</code> or <code>refitsmooth</code> and use a log link.
factor	Positive finite numeric scalar. 1 retains the prediction level, values above 1 increase it, and values below 1 decrease it.

Details

For a refined GLM with a log link, calibration adds $\log(\text{factor})$ to the intercept. Consequently, every response-scale prediction is multiplied by `factor`, while all non-intercept coefficients and tariff relativities remain unchanged.

The returned object is a copied, internally consistent fitted model. Its coefficients, linear predictors, fitted values, working residuals, deviance and AIC are updated to the calibrated level. The original refined model is not modified. Calibration metadata store the factor, log shift, original and calibrated intercept, creation time and call.

Refinement and calibration:

Model refinement changes or constrains the relative tariff structure and is evaluated through `prepare_refinement()`, one or more `add_*()` operations, and `refit()`. Model calibration changes only the final overall level. A calibrated model cannot be calibrated again or used as the starting point for further refinement. Retain the `rating_refinement` specification and recalibrate a newly refitted model if earlier decisions need to be revised.

Value

A fitted glm that also inherits from calibrated_model. Attributes calibration_factor, calibration_original_intercept, calibration_intercept, calibration_log_shift, calibration_call and calibrated_at record the calibration.

See Also

[refit\(\)](#), [rating_table\(\)](#), [add_prediction\(\)](#), [audit_refinement\(\)](#)

Examples

```
restrictions <- data.frame(
  zip = c(0, 1, 2, 3),
  zip_restricted = c(0.90, 1.00, 1.05, 1.10)
)

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

mod_refined <- mod_initial |>
  prepare_refinement() |>
  add_restriction(restrictions) |>
  refit(intercept_only = TRUE)

mod_calibrated <- calibrate_model(mod_refined, factor = 1.05)

rating_table(mod_calibrated)

data_final <- mod_refined$data |>
  add_prediction(
    mod_calibrated,
    predictions = "net_risk_premium"
  )
```

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	<i>Check simulation-based model residuals</i>
-----------------	---

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

<code>object</code>	A fitted "glm" object supported by <code>DHARMA::simulateResiduals()</code> .
<code>n_simulations</code>	Number of simulations used to generate residuals. Must be a positive whole number. The default of 30 is intended for a quick check. For a more stable final assessment, 250 to 1,000 simulations will often be more suitable. More simulations increase the calculation time.

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.

A low p-value does not always mean that the model has an important pricing problem. In a large portfolio, even a small difference can produce a low p-value. The result should therefore be reviewed together with the QQ plot, the size of the difference and its relevance for the tariff.

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 = 250)
autoplot(cr)

## End(Not run)
```

derive_tariff_segments

Derive candidate tariff segments from a smooth risk-factor effect

Description

Approximate the smooth effect estimated by `risk_factor_gam()` with intervals for a continuous risk factor. The resulting boundaries provide a candidate categorical representation that can be inspected before inclusion in a pricing GLM or tariff structure.

Usage

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

Arguments

object	A "risk_factor_gam" object returned by <code>risk_factor_gam()</code> . Legacy "riskfactor_gam" and "fitgam" classes are accepted for compatibility.
segmentation_penalty	Non-negative numeric penalty on additional tree splits. Larger values generally favour fewer tariff segments. The default 0 retains the historical behaviour and applies no explicit split penalty; it can therefore produce a relatively detailed candidate segmentation. There is no universal actuarial value: compare candidate penalties and assess the resulting volume and stability by segment.
seed	Single finite whole number used to reproduce the evolutionary search.
max_iterations	Positive integer. Maximum number of evolutionary search iterations. This is an advanced algorithm-control parameter.
population_size	Positive integer. Number of candidate trees maintained during the evolutionary search. This is an advanced algorithm-control parameter.
complexity	Deprecated. Use segmentation_penalty instead.
alpha	Deprecated. Use segmentation_penalty instead.
niterations	Deprecated. Use max_iterations instead.
ntrees	Deprecated. Use population_size instead.

Details**Method:**

An evolutionary regression tree from `evtree::evtree()` is fitted to the predicted GAM effect over the distinct observed risk-factor values. The tree therefore approximates the estimated univariate curve; it is not fitted directly to individual claim outcomes or portfolio loss. Internal tree split points are translated into interval boundaries. If no internal split is supported by the fitted search, one interval spanning the observed range is returned.

The method follows the data-driven binning approach described by Henckaerts et al. (2018). `segmentation_penalty`, `population_size`, `max_iterations` and `seed` control the stochastic search rather than an actuarial minimum-volume rule. Reusing the same inputs and seed makes the result reproducible.

Each distinct observed risk-factor value has equal influence when the tree approximates the fitted curve. Exposure, claim count or another actuarial weight is deliberately not applied again in this step. The relevant portfolio information has already influenced the curve through the statistical specification used by `risk_factor_gam()`, such as the exposure offset in a frequency model, claim-count weights in a severity model or exposure weights in a risk-premium model. Applying a second weight during segmentation would introduce an additional portfolio-distribution choice after the GAM has been estimated.

Exposure and claim count remain available through `summary()`. They are diagnostics for assessing the support and practical stability of candidate segments, but they do not influence the estimated boundaries.

Actuarial interpretation:

The returned segments approximate the shape of the fitted univariate GAM; they are not automatically a final tariff classification. Before use in a multivariate model, the boundaries should

be assessed against exposure and claim volume, stability across periods, operational rounding and the interaction with other risk factors. Particular care is required for boundaries in sparsely populated tails.

`summary()` reports the number of portfolio records, number of distinct risk-factor values and available exposure and claim volume within each proposed segment. These diagnostics support actuarial review but do not constitute an automatic acceptance rule. Minimum-volume requirements and operational rounding should be selected with reference to portfolio size, model purpose and governance standards.

A staged GLM refinement workflow:

In practical pricing work, the candidate boundaries are often used to form an initial set of relatively broad model groups. The actuary reviews `summary()` and, where necessary, combines thinly populated segments or increases `segmentation_penalty` until the groups have sufficient exposure and claim information for stable estimation. The resulting factor can then be included in an unrestricted GLM.

This broad first-stage grouping avoids estimating a separate free GLM coefficient for every fine tariff interval when observations are unevenly distributed over the continuous risk factor. After fitting the GLM, `add_smoothing()` can use the broad model effect together with the original continuous variable to construct a regularised pattern over finer breaks. These finer breaks may reflect operational or commercial tariff boundaries, while their relativities remain linked through the smoothing specification rather than being estimated independently for every small segment.

The staged approach therefore separates statistical support from final tariff granularity: broad groups provide the information used by the GLM, while smoothing can translate that information into a finer and more regular tariff structure. Smoothing does not create additional observations, so the resulting classes should still be assessed for stability, extrapolation and commercial suitability. The first and last boundaries equal the observed range used by the GAM. Applying the segmentation to new data outside that range results in an informative error rather than silent extrapolation.

Use `autoplot.tariff_segments()` to compare the smooth curve and boundaries. Use `add_tariff_segments()` to apply the resulting boundaries to portfolio data using the original continuous risk factor.

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.

segment_summary Data frame with portfolio counts, distinct risk-factor values and the observed response components for each candidate segment. Use `summary()` as the public interface for this table.

segmentation_penalty Penalty applied to additional tree splits.

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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1111/j.14679868.2010.00749.x)

See Also

[risk_factor_gam\(\)](#), [autoplot.tariff_segments\(\)](#), [add_tariff_segments\(\)](#), [prepare_refinement\(\)](#), [add_smoothing\(\)](#)

Examples

```
## Not run:
age_segments <- risk_factor_gam(
  MTPL,
  risk_factor = "age_policyholder",
  claim_count = "nclaims",
  exposure = "exposure"
) |>
  derive_tariff_segments(
    segmentation_penalty = 10,
    seed = 1
  )

autoplot(age_segments, show_observations = TRUE)
summary(age_segments)

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

## End(Not run)
```

edit_smoothing	<i>Edit a smoothing curve in a refinement workflow</i>
----------------	--

Description

Modify a specified interval of a smoothing curve previously added with [add_smoothing\(\)](#). Use a relative adjustment when the existing shape is broadly appropriate, or explicit values and control points when the curve should follow known targets.

Usage

```
edit_smoothing(
  model,
  model_variable = NULL,
  step = NULL,
  from = NULL,
  to = NULL,
  from_value = NULL,
  to_value = NULL,
  control_positions = NULL,
  control_values = NULL,
  adjustment = NULL,
  slope_adjustment = 1,
  transition = NULL,
  allow_extrapolation = FALSE,
  extrapolation_step = NULL
)
```

Arguments

model	Object of class <code>rating_refinement</code> , created with prepare_refinement() and containing an existing smoothing step. Ordinary and refitted GLMs are not accepted directly. Legacy smooth and restricted objects are still accepted for backwards compatibility.
model_variable	Character string. The <code>model_variable</code> of the smoothing step to edit. Required when more than one smoothing step exists and <code>step</code> is not supplied.
step	Optional numeric index of the original smoothing step or one of its later edit steps. In both cases, the new edit is linked to the same original smoothing and appended after the existing workflow steps.
from, to	Optional numeric values giving the start and end of the source-variable interval to modify. For <code>adjustment</code> , either value may be omitted to use the beginning or end of the available smoothing range. For <code>slope_adjustment</code> , <code>from</code> is the required anchor and <code>to</code> must remain <code>NULL</code> . Explicit target-value and control-point edits require both values.
from_value, to_value	Optional numeric values used to override the smoothed curve value at <code>from</code> and <code>to</code> .

control_positions, control_values	Optional numeric vectors of equal length. These define additional points that the edited smoothing curve should pass through.
adjustment	Optional positive numeric scalar applied multiplicatively to the current smoothing within the selected interval. 1.05 requests an increase of up to 5 percent and 0.95 a decrease of up to 5 percent. With two boundaries, the default transition anchors the multiplier at 1 at from and to; a one-sided edit is anchored only at the supplied boundary.
slope_adjustment	Positive numeric scalar controlling the change in slope after from. The default 1 leaves the curve unchanged. Values above 1 strengthen the remaining change; values between 0 and 1 flatten it. This argument is available only in edit_smoothing().
transition	Optional character string controlling how adjustment connects to the unchanged smoothing. NULL inherits the original smoothing specification. "linear" gives continuous linear transitions and "step" permits immediate jumps. Smoothing methods accepted by add_smoothing() can be supplied as explicit structural overrides.
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

edit_smoothing() appends a separate, ordered edit step to a rating_refinement object. It does not alter the fitted GLM immediately. Repeated calls are cumulative: every new edit starts from the smoothing produced by preceding edits to the same add_smoothing() step. The selected cumulative curve is applied when refit() is called.

Use model_variable or step to identify the smoothing to edit. step may identify either its original add_smoothing() step or a later edit belonging to that smoothing. The interval from from to to defines the part of the source-variable range that should be changed. With adjustment, either boundary may be omitted. Supplying only from edits the curve from that value to the end of the smoothing range; supplying only to edits it from the beginning of the range to that value. adjustment multiplies the current smoothing within the selected range. For example, adjustment = 1.05 requests an increase of up to 5 percent relative to the existing smoothing.

With two boundaries, the multiplier is anchored at 1 at from and to and reaches the requested adjustment near the middle. With only from, it is anchored at 1 at from and moves towards the requested adjustment at the end of the range. With only to, it starts at the requested adjustment and reconnects to 1 at to. These one-sided forms are useful for refining a lower or upper tail without introducing a jump at the supplied boundary.

By default, transition = NULL inherits the smoothing specification from the add_smoothing() step. The entry and exit are adapted to their opposite directions and join the unchanged curve continuously. "linear" uses continuous straight transitions. "step" applies the multiplier immediately at both boundaries and therefore permits deliberate jumps. Explicit shape-constrained

transition names accepted by `add_smoothing()` can also be supplied. When a constrained transition is inherited or selected, the edited curve is checked for the corresponding monotonicity and curvature.

`from_value` and `to_value` instead prescribe curve values at the interval boundaries. `control_positions` and `control_values` add points that the edited curve should follow inside the interval. Relative adjustments and explicit target values cannot be combined in one `edit_smoothing()` call because they represent different actuarial instructions. They may be used in separate consecutive edits, which are then evaluated in their stored order.

`slope_adjustment` changes the remaining increase or decrease after `from`, while keeping the curve before that point unchanged. If $R(x)$ is the current smoothing and a is `from`, the edited curve is $R(a) + s[R(x) - R(a)]$ for $x > a$, where s is `slope_adjustment`. A value of 1.10 therefore makes the change after the anchor 10 percent stronger; 0.90 makes it 10 percent weaker. The curve is continuous at the anchor.

Each call applies one edit type: a relative adjustment, a `slope_adjustment`, or explicit target/control-point values. Apply multiple changes in consecutive calls so that every actuarial intervention remains a separate, inspectable refinement step.

Actuarial interpretation:

The edited interval is an explicit tariff assumption layered on the statistically fitted smoothing curve. It should be supported by an actuarial rationale and reviewed against exposure, observed experience and the continuity of adjacent segments. The edit does not add information to sparse parts of the portfolio and should not be interpreted as a new model estimate.

Keep the `rating_refinement` object, call `refit()` to assess the current specification, edit that same refinement object, and call `refit()` again. The previously fitted GLM remains unchanged. This retains the order and content of manual adjustments as part of the reproducible refinement specification.

Value

A `rating_refinement` object with a separate smoothing-edit step appended to the ordered specification. The pricing GLM is not fitted again until `refit()` is called.

Author(s)

Martin Haringa

See Also

`prepare_refinement()`, `add_smoothing()`, `add_restriction()`, `add_shrinkage()`, `add_rebasing()`, `add_relativities()`, `refit()`

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
  )

  refinement <- prepare_refinement(model, data = portfolio) |>
    add_smoothing(
      model_variable = "age_band",
      source_variable = "driver_age",
      breaks = c(18, 30, 40, 50, 60),
      weights = "exposure"
    )

  # Fit and inspect the initial smoothing specification.
  initial_model <- refit(refinement)

  # Edit the retained specification and fit it again.
  explicit_refinement <- refinement |>
    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)
    )

  explicit_model <- refit(explicit_refinement)

  # Keep the current shape as the basis and raise the middle of this interval
  # by up to 5 percent. The inherited transition remains continuous.
  adjusted_refinement <- refinement |>
    edit_smoothing(
      model_variable = "age_band",
      from = 30,
      to = 50,
      adjustment = 1.05
    )

```

```

adjusted_model <- refit(adjusted_refinement)

# Keep the curve unchanged through age 40, then strengthen its remaining
# change by 10 percent while retaining continuity at age 40.
steeper_refinement <- refinement |>
  edit_smoothing(
    model_variable = "age_band",
    from = 40,
    slope_adjustment = 1.10
  )

# A one-sided adjustment applies from age 40 to the end of the range.
upper_tail_refinement <- refinement |>
  edit_smoothing(
    model_variable = "age_band",
    from = 40,
    adjustment = 1.05,
    transition = "linear"
  )

```

extract_model_data	<i>Recover the portfolio data used by a fitted model</i>
--------------------	--

Description

Recover the estimation data and pricing metadata stored with a fitted GLM or a model produced by the refinement workflow. The result provides a reproducible basis for rating grids, coefficient tables and portfolio-level model diagnostics.

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

Data represented by the result:

For an ordinary GLM, the function recovers the data stored with the model or its model frame and records the response, model terms, risk factors, weights and offsets. The recovered data represent the observations available to the fitted model. Rows omitted during fitting, for example because of missing model variables, may therefore not be present.

For a refined model, technical columns used to construct smoothing and restriction terms are removed from the returned data. The mappings required to interpret the refined coefficients are retained as attributes.

Actuarial use:

The extracted object is intended for downstream calculations that must remain consistent with the fitted pricing model, such as `rating_grid()` and `rating_table()`. It should not be interpreted as a replacement for the original raw portfolio extract: preprocessing, filtering and missing-value handling applied before or during model fitting remain part of the data provenance.

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 and smooths for refit objects;
- new_nm, old_nm: new and old column names for refit objects.

Author(s)

Martin Haringa

See Also

`rating_grid()`, `rating_table()`, `prepare_refinement()`

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	<i>Summarise observed portfolio experience by risk factor</i>
-----------------	---

Description

Aggregate observed claim, exposure and premium experience for one or more discrete risk factors. The result supports exploratory pricing analysis by showing how portfolio volume and unadjusted actuarial metrics vary across factor levels.

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 containing portfolio observations.
risk_factors	Non-empty character vector naming the discrete risk factors to analyse.
claim_amount	Optional character string naming the total claim-amount column.
claim_count	Optional character string naming the claim-count column.
exposure	Optional character string naming the exposure column.
premium	Optional character string naming the premium-amount column.
group_by	Optional character vector naming additional grouping variables, such as under-writing year or product segment.
df, x, severity, nclaims, by	Deprecated argument names. Use data, risk_factors, claim_amount, claim_count, and group_by instead.

Details

Calculated measures:

Depending on the supplied columns, the function calculates:

- frequency = claim_count / exposure;

- `average_severity = claim_amount / claim_count;`
- `risk_premium = claim_amount / exposure;`
- `loss_ratio = claim_amount / premium;`
- `average_premium = premium / exposure.`

Input amount columns are summed before ratios are calculated. A measure is omitted when its required inputs were not supplied. A zero or missing denominator produces `NA_real_` rather than an infinite value.

Actuarial interpretation:

These are observed, univariate or stratified portfolio measures. They are not adjusted for correlation between rating factors and should not be interpreted as conditional GLM effects. Differences between levels may reflect portfolio mix, small exposure, claim volatility or changes over time. Claim counts, exposure and stability should therefore be reviewed alongside the ratios.

`group_by` can be used to compare the same risk-factor pattern across periods or portfolio segments. `autoplot.factor_analysis()` provides the corresponding graphical review. Modelled effects can subsequently be inspected with `rating_table()`.

Column interface:

Column names are supplied as character strings. Deprecated `univariate()` remains available for compatibility with its former interface.

Value

A data frame with classes `"factor_analysis"`, `"univariate"` and `"data.frame"`. It contains the grouping columns, aggregated input columns and all actuarial measures supported by the supplied inputs. The original column names are retained for claim amount, claim count, exposure and premium.

Author(s)

Martin Haringa

See Also

`autoplot.factor_analysis()`, `rating_table()`, `add_portfolio_experience()`

Examples

```
area_experience <- factor_analysis(
  MTPL2,
  risk_factors = "area",
  claim_amount = "amount",
  claim_count = "nclaims",
  exposure = "exposure",
  premium = "premium"
)

area_experience
autoplot(area_experience, metrics = c("frequency", "risk_premium"))
```

fisher_classify	<i>Fisher's natural breaks classification</i>
-----------------	---

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

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

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	<i>Reduce portfolio periods by merging adjacent date ranges</i>
-------------------	---

Description

Combine overlapping or nearly adjacent coverage periods for the same policy, risk or portfolio segment. The result provides a consolidated time basis for exposure calculations, active-policy counts and period-based reporting.

Together with [rating_grid\(\)](#), this function belongs to the portfolio reduction workflow. Both functions reduce row-level portfolio data while retaining selected totals. `merge_date_ranges()` reduces temporally connected records; [rating_grid\(\)](#) reduces records with identical risk-factor values.

Usage

```
merge_date_ranges(
  data = NULL,
  ...,
  period_start = NULL,
  period_end = NULL,
  group_by = NULL,
  aggregate_cols = NULL,
  aggregate_fun = "sum",
  merge_gap_days = 1,
  df = NULL,
  begin = NULL,
  end = NULL,
  agg_cols = NULL,
  agg = NULL,
  min.gapwidth = NULL
)
```

Arguments

<code>data</code>	A <code>data.frame</code> or <code>data.table</code> containing the portfolio periods.
<code>period_start</code>	Character string. Name of the column with period start dates.
<code>period_end</code>	Character string. Name of the column with period end dates.
<code>group_by</code>	Character vector with columns that identify the portfolio entity or rating segment within which date ranges should be merged.
<code>aggregate_cols</code>	Character vector with numeric columns to aggregate over merged ranges, for example premium or exposure.
<code>aggregate_fun</code>	Function or function name used to combine <code>aggregate_cols</code> within a merged interval. The default, "sum", is generally appropriate for additive measures such as premium or exposure.

`merge_gap_days` Non-negative whole number. Ranges with fewer uncovered days than this value are treated as continuous. The default, 1, merges overlapping periods and periods that start on the day after the preceding period ends. Use 0 to merge overlapping periods only. A value above 1 also bridges short uncovered gaps and should represent an explicit administrative assumption.

`df, begin, end, . . ., agg_cols, agg, min_gapwidth`
 Deprecated argument names kept for backward compatibility. Use `data`, `period_start`, `period_end`, `group_by`, `aggregate_cols`, `aggregate_fun`, and `merge_gap_days`.

Details

Portfolio reduction:

`merge_date_ranges()` performs temporal portfolio reduction. It combines connected periods within the same `group_by` values and retains the selected additive amounts. Use `rating_grid()` for the complementary categorical reduction of records with identical risk-factor combinations.

Connected periods:

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. The grouping columns should identify records for which combining coverage periods is actuarially and operationally meaningful; periods belonging to different risks or contracts should not be pooled. Missing values are not permitted in `group_by`, because such records cannot be assigned reliably to the same policy, risk or segment.

Start and end dates are treated as inclusive. Consequently, two periods for which the second starts one day after the first ends have zero uncovered days. They are merged with the default `merge_gap_days = 1`. A value of 5 merges periods with at most four uncovered days between them.

Aggregation and implementation:

Aggregated amounts are combined over the source rows, not prorated over calendar days. Summing premium or exposure is appropriate when each source row contains a distinct additive amount. If overlapping rows already contain amounts for the same covered days, users should resolve that overlap before aggregation to avoid double counting.

Each output row represents one consolidated period within a `group_by` combination. This is a temporal reduction of the portfolio; records with the same risk-factor values are not combined when their periods remain separate.

Internally, interval construction and aggregation use `data.table::data.table()` on a local copy. The supplied object is not modified by reference. For a portfolio that should remain outside R memory, use `merge_date_ranges_db()` to perform the same interval reduction in DuckDB.

Value

A regular `data.frame` with classes "merged_date_ranges" and "reduce", and attributes:

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

Author(s)

Martin Haringa

See Also

[merge_date_ranges_db\(\)](#), [rating_grid\(\)](#), [rating_grid_db\(\)](#), [active_rows_by_date\(\)](#), [split_periods_to_months\(\)](#)

Examples

```
portfolio <- data.frame(
  policy_id = rep(c("P001", "P002"), each = 3),
  coverage = rep(c("Fire", "Liability"), each = 3),
  period_start = as.Date(c(
    "2024-01-01", "2024-02-01", "2024-04-01",
    "2024-01-01", "2024-02-03", "2024-03-03"
  )),
  period_end = as.Date(c(
    "2024-01-31", "2024-02-29", "2024-04-30",
    "2024-01-31", "2024-03-02", "2024-03-31"
  )),
  earned_premium = c(100, 110, 120, 80, 90, 95)
)

# Reduce directly adjacent periods within each policy and coverage.
pt1 <- merge_date_ranges(
  portfolio,
  period_start = "period_start",
  period_end = "period_end",
  group_by = c("policy_id", "coverage")
)

summary(pt1, period = "months", policy_id, coverage)

# Bridge a short administrative gap and retain additive premium totals.
pt2 <- merge_date_ranges(
  portfolio,
  period_start = "period_start",
  period_end = "period_end",
  group_by = c("policy_id", "coverage"),
  aggregate_cols = "earned_premium",
  # Explicitly bridge administrative gaps of up to four uncovered days.
  merge_gap_days = 5
)

summary(pt2, period = "months", policy_id, coverage)
```

merge_date_ranges_db *Merge connected portfolio periods in DuckDB*

Description

Construct the temporal portfolio reduction performed by `merge_date_ranges()` as a lazy DuckDB query. Overlapping or adjacent periods are merged inside DuckDB, so only the consolidated periods need to be copied into R.

Usage

```
merge_date_ranges_db(
  data,
  period_start,
  period_end,
  group_by,
  aggregate_cols = NULL,
  aggregate_fun = c("sum", "mean", "min", "max"),
  merge_gap_days = 1
)
```

Arguments

<code>data</code>	A lazy DuckDB table created with <code>dplyr::tbl()</code> .
<code>period_start</code>	Character string naming the period-start column.
<code>period_end</code>	Character string naming the period-end column.
<code>group_by</code>	Character vector identifying the policy, risk or segment within which periods may be combined.
<code>aggregate_cols</code>	Optional character vector naming numeric columns to aggregate over each merged period.
<code>aggregate_fun</code>	Character string specifying the SQL aggregation for <code>aggregate_cols</code> : "sum", "mean", "min", or "max".
<code>merge_gap_days</code>	Non-negative whole number. The interpretation is the same as in <code>merge_date_ranges()</code> : 1 merges overlapping and directly adjacent periods, while 0 merges overlapping periods only.

Details

Merging connected intervals is a SQL gaps-and-islands calculation. The query uses ordered window functions to compare each start date with the latest preceding end date, assigns an interval identifier, and then aggregates each resulting interval.

Date arithmetic differs between database systems. This implementation is therefore deliberately restricted to DuckDB. It does not download the source table and does not create a permanent database table.

Value

A lazy DuckDB table. Use `dbplyr::sql_render()` to inspect the SQL or `dplyr::collect()` to import the consolidated periods into R.

Author(s)

Martin Haringa

See Also

`merge_date_ranges()`, `rating_grid_db()`, `rating_grid()`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
portfolio_db <- dplyr::tbl(con, "portfolio_periods")

periods_db <- merge_date_ranges_db(
  portfolio_db,
  period_start = "period_start",
  period_end = "period_end",
  group_by = c("policy_id", "coverage"),
  aggregate_cols = c("earned_exposure", "earned_premium")
)

periods <- dplyr::collect(periods_db)
DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

model_performance

Compare fitted GLMs using common performance measures

Description

Compare one or more fitted GLMs using AIC, BIC and response-scale RMSE. The resulting table provides a concise first comparison of alternative pricing-model specifications fitted to the same portfolio outcome.

Usage

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

Arguments

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

Details

The following measures are reported:

AIC Akaike information criterion, balancing likelihood fit and model complexity.

BIC Bayesian information criterion, applying a stronger sample-size-dependent complexity penalty.

RMSE Root mean squared error between observed and response-scale predicted values.

Lower values are preferred within each measure, but the measures answer different questions. AIC and BIC depend on the model likelihood, whereas RMSE measures error on the response scale. Comparisons are therefore most meaningful when models use the same response, estimation records, weights and offsets.

The table does not select a pricing model automatically. In actuarial model assessment, statistical fit should be considered together with portfolio calibration, residual behaviour, coefficient stability, exposure by level and the practical interpretability of the resulting tariff structure.

The implementation 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

See Also

[rmse\(\)](#), [bootstrap_performance\(\)](#), [check_overdispersion\(\)](#), [check_residuals\(\)](#)

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 Total claim amount for the portfolio record (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 Total claim amount for the portfolio record (in euros).

exposure Exposure, expressed in years.

premium Earned premium.

Author(s)

Martin Haringa

outlier_histogram	<i>Portfolio histogram with tail bins</i>
-------------------	---

Description

Visualise 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

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

<code>data</code>	A <code>data.frame</code> with claim-level observations.
<code>claim_amount</code>	Character string. Name of the claim amount column.
<code>risk_factor</code>	Optional character string. Name of the risk factor used to split the severity distribution. If <code>NULL</code> , the overall claim amount distribution is shown.
<code>top_n</code>	Positive whole number. Number of categories to keep after filtering and sorting.
<code>min_claims</code>	Positive whole number. Categories with fewer than this number of claim observations are removed.
<code>sort</code>	Character. Metric used to sort and select categories. One of "median", "mean" or "n_claims".
<code>threshold</code>	Optional numeric scalar. If supplied, claims above this threshold are highlighted and a dotted threshold line is shown.
<code>mean</code>	Logical. If <code>TRUE</code> , add a marker for the average claim amount.
<code>median</code>	Logical. If <code>TRUE</code> , add a marker for the median claim amount.
<code>distribution</code>	Character. Distribution layer. One of "none", "half_violin" or "violin". Default is "none".
<code>point_method</code>	Character. Point placement method. One of "quasirandom", "jitter" or "none".
<code>orientation</code>	Character. "horizontal" places claim amount on the x-axis and categories on the y-axis. "vertical" reverses this.
<code>log_scale</code>	Logical. If <code>TRUE</code> , use a log10 scale for claim amounts.
<code>boxplot</code>	Logical. If <code>TRUE</code> , add a small centred boxplot. Default is <code>FALSE</code> .
<code>boxplot_width</code>	Numeric scalar. Width of the optional boxplot. Smaller values keep the boxplot as a subtle summary layer behind the individual claim points.
<code>show_labels</code>	Logical. If <code>TRUE</code> , add direct labels for the mean, median and, when supplied, threshold. Requires the suggested package <code>ggrepel</code> .
<code>all_claims_label</code>	Character string used as the category label when <code>risk_factor = NULL</code> .
<code>mean_label</code>	Character string used for the direct mean marker label. Default is "Mean".
<code>median_label</code>	Character string used for the direct median marker label. Default is "Median".
<code>threshold_label</code>	Character string used for the optional threshold label.
<code>x_label</code>	Optional character string. X-axis label. If <code>NULL</code> , a default is chosen from <code>claim_amount</code> , <code>risk_factor</code> and <code>orientation</code> .
<code>y_label</code>	Optional character string. Y-axis label. If <code>NULL</code> , a default is chosen from <code>claim_amount</code> , <code>risk_factor</code> and <code>orientation</code> .

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
)
```

```
premium_change
```

Interpret the premium effect of a smoothing curve

Description

Translate an effective smoothing curve in a refinement specification into concrete modelled-premium comparisons. By default, each selected value is compared with twice that value. Supplying increment instead compares each value with a fixed increment above it.

Format an object returned by premium_change(). One refinement state is shown as a three-column table. Multiple states are shown side by side; when exactly two are selected, their difference is added in percentage points.

Usage

```

premium_change(
  x,
  variable = NULL,
  at = NULL,
  change = "double",
  increment = NULL,
  steps = "current",
  basis = c("curve", "segments"),
  ...
)

## S3 method for class 'premium_change'
as_gt(x, locale = "en-US", decimals = 1, title = NULL, subtitle = NULL, ...)

```

Arguments

<code>x</code>	For <code>premium_change()</code> , a <code>rating_refinement</code> object containing at least one smoothing step. For <code>as_gt()</code> , an object returned by <code>premium_change()</code> .
<code>variable</code>	Optional character string identifying the smoothed model variable or its continuous source variable. This may be omitted when the refinement contains exactly one smoothing lineage.
<code>at</code>	Optional numeric vector of starting values. Each starting and comparison value must lie inside the supported smoothing range of every selected refinement state. Doubling retains the existing requirement that starting values are positive. If <code>NULL</code> , approximately six representative values are selected automatically.
<code>change</code>	Character comparison mode. "double" (default) compares x with $2x$. When increment is supplied, omit change; fixed- increment mode is then selected automatically.
<code>increment</code>	Optional positive finite numeric increase in the units of the source variable. When supplied, compares x with $x + increment$. It cannot be combined with an explicitly supplied change instruction.
<code>steps</code>	Refinement states to evaluate. Use "current" for the latest state, "all" for every state from the selected smoothing onwards, or a numeric vector such as <code>c(1, 6)</code> for stored refinement positions.
<code>basis</code>	Character string determining the interpretation basis. "curve", the default, evaluates the continuous effective smoothing at the exact values. "segments" compares the effective relativities of the tariff intervals containing those values.
<code>...</code>	Additional arguments are not accepted.
<code>locale</code>	Character string passed to <code>gt</code> for numeric formatting.
<code>decimals</code>	Non-negative integer. Number of decimal places for changes.
<code>title</code>	Optional table title. The source-variable name is used by default.
<code>subtitle</code>	Optional table subtitle.

Details

`premium_change()` is an interpretation helper for smoothing created with `add_smoothing()` and subsequently modified with `edit_smoothing()`. It is not a smoothing method and does not change the refinement specification.

For a multiplicative relativity curve $R(x)$, doubling reports $R(2x)/R(x) - 1$. Fixed-increment mode reports $R(x + h)/R(x) - 1$, where h is increment. If total modelled premium can be written as $P(x, z) = C(z)R(x)$, all other multiplicative model effects $C(z)$ cancel in this ratio. No particular policy profile is therefore required for the interpretation.

The effective curve is reconstructed from the stored refinement history. Consequently, `steps = "current"` reflects all smoothing edits recorded up to the current state. Numeric step identifiers refer to positions in the complete refinement sequence. If another type of refinement occurs after a smoothing step, the previously effective smoothing is carried forward.

With the default `basis = "curve"`, evaluation uses the continuous effective smoothing line retained by the refinement system. It therefore describes the shape and steepness of the estimated or edited curve at exactly x and the corresponding comparison value; it does not use neighbouring tariff-segment relativities.

With `basis = "segments"`, both values are assigned to the effective tariff intervals created by the smoothing. Their current segment relativities are compared. This describes the premium effect of the implementable segmented tariff. The result can be zero when both values fall in the same segment and can change discretely when the comparison crosses a segment boundary.

Values are never extrapolated. When `at = NULL`, six representative starting values are selected from the common range for which both the starting and comparison values are supported in every selected refinement state.

Multiplying an entire curve by a common rebasing constant does not alter the result because that constant cancels in the relativity ratio.

Value

A tibble with class `premium_change` in long format, containing the variable, refinement state, starting and comparison values, evaluated relativities, and premium change as a decimal.

A `gt_tbl` object.

See Also

`add_smoothing()`, `edit_smoothing()`, `autoplot.rating_refinement()`, `as_gt()`

Examples

```
age <- rep(seq(20, 70, by = 5), each = 5)
portfolio <- data.frame(
  claims = rep(c(0, 1, 0, 2, 1), length(age) / 5),
  exposure = 1,
  age = age
)
portfolio$age_band <- cut(
  portfolio$age,
  breaks = c(15, 30, 45, 60, 75),
```

```

    include.lowest = TRUE
  )
  model <- glm(
    claims ~ age_band + offset(log(exposure)),
    family = poisson(),
    data = portfolio
  )
  refinement <- prepare_refinement(model, data = portfolio) |>
    add_smoothing(
      model_variable = "age_band",
      source_variable = "age",
      breaks = seq(15, 75, by = 5),
      smoothing = "poly",
      degree = 2,
      weights = "exposure"
    )
  premium_change(refinement, at = c(20, 25, 30))
  premium_change(refinement, at = c(20, 25, 30), increment = 5)
  premium_change(refinement, at = c(20, 25, 30), basis = "segments")

  edited <- refinement |>
    edit_smoothing(
      model_variable = "age_band",
      from = 20,
      to = 60,
      adjustment = 1.05,
      transition = "linear"
    )
  premium_change(edited, at = c(20, 25, 30), steps = c(1, 2))

```

prepare_refinement	<i>Prepare a model refinement workflow</i>
--------------------	--

Description

Create an editable refinement specification from a fitted pricing GLM. Smoothing, coefficient restrictions, shrinkage, rebasing and sublevel relativities can then be added in a defined order. These steps do not alter the fitted GLM until `refit()` is called.

Usage

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

Arguments

<code>model</code>	Object of class <code>glm</code> .
<code>data</code>	Optional <code>data.frame</code> containing exactly the observations retained in the fitted GLM and all required model variables. If model fitting omitted rows because of missing values, supply the retained model data rather than the original unfiltered data. If <code>NULL</code> , the data are retrieved from the model object.

Details

`prepare_refinement()` creates a persistent refinement specification. This object contains the original GLM, the corresponding model data and the ordered smoothing, restriction, shrinkage, rebasing and relativity steps. Retain this object during actuarial review so that assumptions can be inspected, revised and applied again in the same order.

Actuarial interpretation:

Preparing a refinement does not change coefficients, fitted values or the tariff structure. It separates the original statistical model from subsequent actuarial adjustments. Each adjustment remains an explicit step rather than being embedded directly in transformed data or overwritten model coefficients. This supports comparison between the unrestricted model and alternative refinement specifications.

`refit()` applies the stored specification and returns a fitted GLM for model diagnostics, prediction and tariff reporting. The returned GLM is a result, not an editable refinement specification. Functions such as `add_smoothing()`, `edit_smoothing()`, `add_restriction()`, `add_shrinkage()`, `add_rebasing()` and `add_relativities()` therefore accept a `rating_refinement` object and do not accept an ordinary or refitted GLM directly.

A practical iterative workflow therefore keeps both objects:

```
refinement <- prepare_refinement(model) |>
  add_smoothing(...)
```

```
fitted_model <- refit(refinement)
```

```
refinement <- refinement |>
  edit_smoothing(...)
```

```
fitted_model <- refit(refinement)
```

`prepare_refinement()` is normally required only once for such an iteration. Calling it on a model returned by `refit()` deliberately starts a new refinement workflow with the already refined model as its baseline; it does not recover the earlier smoothing or restriction steps for further editing.

Value

A `rating_refinement` object containing the original GLM, retained model data and ordered refinement specification. No GLM is fitted again until `refit()` is called.

Author(s)

Martin Haringa

See Also

`summary.rating_refinement()`, `add_smoothing()`, `edit_smoothing()`, `add_restriction()`, `add_shrinkage()`, `add_rebasing()`, `add_relativities()`, `refit()`

Examples

```

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

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

refinement <- prepare_refinement(model, data = portfolio) |>
  add_restriction(data.frame(
    risk_class = "B",
    risk_class_restricted = 1.15
  ))

summary(refinement)

fitted_model <- refit(refinement)

# Retain and revise the specification rather than editing fitted_model.
refinement <- refinement |>
  add_restriction(data.frame(
    risk_class = "B",
    risk_class_restricted = 1.10
  ))

updated_model <- refit(refinement)

```

rating_grid

Construct observed rating-grid points

Description

Collapse portfolio records with identical risk-factor combinations into observed rating-grid points. Exposure and other numeric measures can be aggregated alongside the combinations for prediction, tariff comparison and portfolio diagnostics.

Together with [merge_date_ranges\(\)](#), this function belongs to the portfolio reduction workflow. Both functions reduce row-level portfolio data while retaining selected totals. `rating_grid()` reduces across identical risk-factor combinations; [merge_date_ranges\(\)](#) reduces temporally connected records within the same policy, risk or portfolio segment.

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

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

Details

Portfolio reduction:

`rating_grid()` performs categorical portfolio reduction. It combines rows with the same observed `group_by` values and retains the corresponding totals. Use `merge_date_ranges()` for the complementary temporal reduction of connected coverage periods.

Observed combinations:

The grid represents the combinations present in the supplied portfolio or model data. It deliberately does not construct combinations that were not observed. This avoids creating artificial model points and is particularly relevant when risk factors are structurally related, such as product, coverage and distribution channel.

Each output row therefore represents one observed combination of the variables in `group_by`. Exposure and `aggregate_cols` are summed over the source records belonging to that combination. Such a row is a model point: one observed covariate combination together with its aggregated additive portfolio quantities. This is a categorical reduction of the portfolio; no date intervals are combined.

Estimating a GLM on aggregated data:

For a standard Poisson frequency GLM, aggregation before model fitting can preserve the coefficient estimates exactly. This applies when records are grouped by every predictor used in the model, claim counts are summed, earned exposure is summed, and the aggregated model uses `offset(log(exposure))`. Within such a group all records have the same linear predictor. Their contribution to the coefficient estimation therefore depends on total claims and total exposure, which are retained by the rating grid.

This equivalence is conditional, not a general property of every GLM. Aggregating over a model variable changes the model, and row-level weights, interactions, offsets or non-additive quantities must be retained correctly. Binomial, severity, quasi-likelihood and dispersion analyses require their own sufficient totals and weights. Row-level residuals and influence diagnostics are also no longer available after aggregation, even when the fitted Poisson coefficients are unchanged.

In practice, a rating grid is particularly useful before estimating a frequency model on a large portfolio. It can reduce repeated policy records to a much smaller table, lowering memory use and fitting time. Keep the unaggregated data when policy-level predictions, sampling, validation or diagnostics are required, and verify on a representative sample that the selected aggregation retains all inputs needed by the intended model.

Estimating a severity GLM on aggregated data:

Claim count and claim amount are additive portfolio measures and can be supplied through `aggregate_cols`. Frequency is then calculated as total claim count divided by total exposure. Average severity is total claim amount divided by total claim count.

A Gamma severity GLM fitted to grouped average severities can produce the same coefficient estimates as a model fitted to the underlying individual claims. This requires grouping by every predictor in the severity model, calculating `average_severity = claim_amount / claim_count`, using `claim_count` as the model weight, and applying the same family and link. Within each grid row all underlying claims then have the same linear predictor, while the weight retains the number of claims represented by the average.

This equivalence concerns the coefficient estimates, not the complete model output. Aggregation removes claim-level residuals and outlier information. Deviance, residual degrees of freedom, estimated dispersion, standard errors and significance tests can therefore differ from a claim-level fit. Claim-level data should remain available for severity-distribution checks, influential-claim analysis and model validation. The equivalence is also lost if a severity predictor varies within a grid row or if the totals and weights do not represent the underlying claims correctly.

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.

Aggregation, reshaping and refinement joins are performed internally with `data.table::data.table()` to support large pricing portfolios. A local copy is used, so the supplied object is not modified by reference. The output is a regular `data.frame`, irrespective of the class of the input data.

When the row-level portfolio does not fit comfortably in R memory, use `rating_grid_db()` to perform the grouped reduction in a database and collect only the resulting grid.

Value

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

Author(s)

Martin Haringa

See Also

`rating_grid_db()`, `merge_date_ranges()`, `merge_date_ranges_db()`, `extract_model_data()`, `rating_table()`

Examples

```
portfolio <- data.frame(
  policy_id = 1:10,
  sector = rep(c("Industry", "Retail"), each = 5),
  region = rep(c("North", "South"), 5),
  underwriting_year = rep(c(2024, 2025), each = 5),
  earned_exposure = c(1, 0.8, 1, 0.5, 1, 1, 0.7, 1, 0.9, 1),
  claim_count = c(0, 1, 2, 0, 1, 0, 1, 0, 2, 1),
  claim_amount = c(0, 2500, 18000, 0, 6000, 0, 4500, 0, 22000, 9000)
)

# Aggregate policy records into observed combinations of sector and region.
# The resulting exposure is the total earned exposure in each combination.
rating_grid(
  portfolio,
  group_by = c("sector", "region"),
  exposure = "earned_exposure"
)

# Split earned exposure by underwriting year. This is useful when reviewing
# whether the portfolio mix within each rating combination changes over time.
rating_grid(
  portfolio,
  group_by = c("sector", "region"),
  exposure = "earned_exposure",
  exposure_by = "underwriting_year"
)

# Claim count and claim amount remain additive totals in the rating grid.
# Frequency and average severity can subsequently be derived from them.
claims_grid <- rating_grid(
  portfolio,
  group_by = c("sector", "region"),
  exposure = "earned_exposure",
  aggregate_cols = c("claim_count", "claim_amount")
)
```

```

)

claims_grid$frequency <-
  claims_grid$claim_count / claims_grid$earned_exposure
claims_grid$average_severity <- ifelse(
  claims_grid$claim_count > 0,
  claims_grid$claim_amount / claims_grid$claim_count,
  NA_real_
)
claims_grid

# Fit a severity model to grouped average claim amounts. Grid rows without
# claims are excluded because average severity is undefined for those rows.
severity_model_grid <- glm(
  average_severity ~ sector + region,
  weights = claim_count,
  family = Gamma(link = "log"),
  data = subset(claims_grid, claim_count > 0)
)
coef(severity_model_grid)

# For a fitted GLM, extract_model_data() retains the model variables and
# exposure information required to construct the observed rating grid.
mtpl_portfolio <- MTPL
mtpl_portfolio$zip <- factor(mtpl_portfolio$zip)

frequency_model <- glm(
  nclaims ~ bm + zip + offset(log(exposure)),
  family = poisson(link = "log"),
  data = mtpl_portfolio
)

frequency_model |>
  extract_model_data() |>
  rating_grid()

# For this Poisson frequency model, fitting on the corresponding aggregated
# grid gives the same coefficient estimates as fitting on the policy rows.
frequency_grid <- rating_grid(
  mtpl_portfolio,
  group_by = c("bm", "zip"),
  exposure = "exposure",
  aggregate_cols = "nclaims"
)

frequency_model_grid <- glm(
  nclaims ~ bm + zip + offset(log(exposure)),
  family = poisson(link = "log"),
  data = frequency_grid
)

isTRUE(all.equal(
  unname(coef(frequency_model)),

```

```

  unname(coef(frequency_model_grid)),
  tolerance = 1e-8
))

```

rating_grid_db

Reduce a database portfolio to observed rating-grid points

Description

Construct the same type of observed rating-factor combinations as `rating_grid()`, while leaving the calculation in the database. The function returns a lazy query and does not copy the source portfolio into R. Each output row is a model point: one observed covariate combination together with aggregated exposure and other additive quantities requested by the user.

This is useful when the row-level portfolio is too large for available R memory. The database performs the grouping and aggregation; the reduced result can subsequently be imported with `dplyr::collect()`.

Usage

```

rating_grid_db(
  x,
  group_by,
  exposure = NULL,
  aggregate_cols = NULL,
  drop_na = FALSE,
  exposure_by = NULL
)

```

Arguments

x	A lazy database table created with <code>dplyr::tbl()</code> .
group_by	Character vector containing the columns that define an observed rating-grid point.
exposure	Optional character string naming an exposure column to sum. If NULL, the output contains a count column with the number of source records in each combination.
aggregate_cols	Optional character vector naming additional columns to sum within each combination.
drop_na	Logical. If TRUE, records with missing group_by values are excluded. If FALSE, missing values remain an observed group.
exposure_by	Reserved for consistency with <code>rating_grid()</code> . Splitting an exposure into dynamically named wide columns is not performed in a lazy database query. Include this variable in group_by, collect the reduced long table, and reshape it in R instead.

Details

`rating_grid_db()` performs only operations that translate naturally to SQL: grouping, row counting and sums. Column names are supplied as strings. No SQL is executed until the lazy query is printed, collected or otherwise used by the database backend.

Database tables have no inherent row order. Apply `dplyr::arrange()` after the final database operation when a specific presentation order is required.

The database table should already contain the rating variables required for the intended model or portfolio analysis. Unlike `rating_grid()`, this function does not inspect a fitted R model or refinement metadata because those objects are held in R rather than in the database.

Value

A lazy database table. Use `dbplyr::sql_render()` to inspect the SQL or `dplyr::collect()` to import the reduced result into R.

Author(s)

Martin Haringa

See Also

`rating_grid()`, `merge_date_ranges_db()`, `merge_date_ranges()`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
portfolio_db <- dplyr::tbl(con, "portfolio")

grid_db <- rating_grid_db(
  portfolio_db,
  group_by = c("sector", "region"),
  exposure = "earned_exposure",
  aggregate_cols = "earned_premium"
)

dbplyr::sql_render(grid_db)
grid <- dplyr::collect(grid_db)
DBI::dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

rating_table	<i>Present fitted pricing-model effects as a rating table</i>
--------------	---

Description

Extract coefficients from one or more fitted GLMs and organise them by risk factor and level. Reference levels are made explicit, coefficients can be expressed as multiplicative relativities, and portfolio exposure can be attached to support actuarial review.

Usage

```
rating_table(
  ...,
  model_data = NULL,
  exposure = TRUE,
  exposure_output = NULL,
  estimate_name = NULL,
  exponentiate = TRUE,
  significance = FALSE,
  reference_first = TRUE,
  level_order = c("estimate_descending", "estimate_ascending", "model", "alphabetical"),
  level_order_by_risk_factor = NULL,
  numeric_level_order = c("ascending", "as_specified"),
  risk_factor_order = c("model", "alphabetical"),
  order_model = NULL,
  round_exposure = 0,
  exposure_name = NULL,
  signif_stars = NULL
)
```

Arguments

...	One or more fitted glm objects, including models returned by <code>refit()</code> . Object expressions are used to construct the dynamic estimate column names.
model_data	Optional data frame used to fit the models. If NULL, the function tries to use <code>model\$data</code> for each supplied model.
exposure	Logical or character string. If TRUE, 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 character string naming the exposure column in the output. If NULL, the original exposure column name is used.
estimate_name	Optional character vector with the exact output column name for each model estimate. Supply one value for one model, an unnamed vector in model order, or a named vector whose names identify the supplied model objects. If NULL, columns retain the default <code>est_<model></code> names.

exponentiate	Logical. If TRUE, coefficients are exponentiated and shown as relativities. If FALSE, coefficients are shown on the model scale.
significance	Logical. If TRUE, add a separate signif_* column for each model containing significance indicators based on coefficient p-values. The corresponding est_* columns remain numeric.
reference_first	Logical. If TRUE, place the reference level first when the global ordering of a nominal risk factor is "model" or "alphabetical". Numeric levels, ordered factors, estimate-based ordering and explicit per-factor overrides retain their selected order. For an ordinary GLM, the reference is obtained from the fitted factor contrasts. After add_rebasing() , the selected rebasing level is used.
level_order	Character string controlling the default order of nominal factor levels. "estimate_descending" (default) places the highest fitted effect first, "estimate_ascending" places the lowest first, "model" retains the fitted model order and "alphabetical" sorts labels. Numeric levels and explicitly ordered factors use their substantive order instead.
level_order_by_risk_factor	Optional named character vector providing an ordering override for individual risk factors. Names identify risk factors and values must be "model", "alphabetical", "estimate_ascending" or "estimate_descending". For example, <code>c(urbanisation = "model", sector = "estimate_descending")</code> preserves an ordinal urbanisation scale while ordering sector relativities from high to low. Numeric ordering still takes precedence.
numeric_level_order	Character string controlling levels that are all recognisable as numbers or numeric intervals. "ascending" orders them by numeric value, or by the lower and then upper interval boundary, regardless of level_order. This correctly orders labels such as (100, 200] and (1000, 2000]. "as_specified" leaves these levels to level_order. Numeric ordering takes precedence over reference_first, so the reference level is not moved away from its numerical position.
risk_factor_order	Character string controlling risk-factor order. "model" retains the order in the fitted model; "alphabetical" sorts risk-factor names. The intercept, when present, remains first.
order_model	Optional character string naming the supplied model whose level order, reference levels and estimates are used for sorting. This is mainly relevant when several models are compared. If NULL, the first supplied model is used. Both "frequency" and "est_frequency" are accepted for a model object named frequency. If that model does not contain a particular risk factor, the first supplied model containing the factor provides its order and reference level.
round_exposure	Non-negative number of digits used to round exposure.
exposure_name	Deprecated. Use exposure_output instead.
signif_stars	Deprecated. Use significance instead.

Details

Coefficients and relativities:

The table contains one row per model term level. For factor variables, the reference level is added explicitly with relativity 1 when `exponentiate = TRUE`, or coefficient 0 when `exponentiate = FALSE`. Numeric model terms are retained on the scale supplied by the fitted model structure.

By default, estimate columns are named from the supplied model expressions, for example `est_frequency` for an object named `frequency`. `estimate_name` can replace these with exact user-supplied names. With several models, use an unnamed vector in model order or a named vector such as `c(frequency = "freq_relativity", severity = "sev_relativity")`. Effects are joined by risk factor and level.

Actuarial interpretation:

With a log-link GLM, exponentiated coefficients represent conditional multiplicative effects relative to the model reference level. They should be interpreted together with the model specification and should not be confused with the unadjusted observed measures returned by `factor_analysis()`.

Exposure by level provides context for the amount of portfolio information supporting each fitted effect. Significance indicators describe evidence conditional on the fitted model; they do not measure practical materiality, temporal stability or suitability for direct tariff implementation.

Comparing multiple models is useful for assessing changes between unrestricted and refined specifications, or between alternative model formulations. Comparable response definitions and coefficient scales remain the responsibility of the analyst.

Row order and reference levels:

By default, risk factors follow the model formula. Numeric levels and intervals are shown from low to high, explicitly ordered factors retain their factor-level sequence, and remaining nominal factors are shown from highest to lowest fitted effect. This separates structural order from an ordering used to compare tariff differentiation.

`reference_first` applies only when a nominal factor uses model or alphabetical order. It does not move the reference level ahead of a numeric, ordinal or estimate-based sequence. The reference remains recorded in the rating-table metadata, including a reference selected with `add_rebasing()`.

Alternative level ordering is useful for specific review tasks. Alphabetical order supports lookup and export, while model order can retain a deliberately specified factor sequence. Use `level_order_by_risk_factor` when nominal and ordinal factors require different treatment in the same table. With several models, `order_model` defines which fitted specification provides estimate-based ordering. `as_gt()` and `autoplot.rating_table()` retain the row order established here.

Only a factor stored with `ordered = TRUE` is identified automatically as an ordinal scale. A regular factor may also have deliberately arranged levels, but that intention cannot be distinguished reliably from an arbitrary model order. Use `level_order_by_risk_factor = c(variable = "model")` to preserve that sequence explicitly.

Numeric labels and intervals receive separate treatment because alphabetical ordering can give an incorrect tariff sequence. With the default `numeric_level_order = "ascending"`, a risk factor is sorted numerically only when every displayed level is either a complete number or a valid interval with two numeric boundaries. Mixed labels such as "Industry 1" remain categorical. Set `numeric_level_order = "as_specified"` when the fitted model order or another `level_order` should be retained deliberately.

Significance indicators:

When `significance = TRUE`, every model receives its own `signif_*` column. For example, models named `frequency` and `severity` produce `est_frequency`, `signif_frequency`, `est_severity`

and `signif_severity`. Keeping estimates and indicators separate preserves the numeric type of the fitted effects for subsequent calculations, filtering and export.

`as_gt()` combines each estimate with its corresponding significance indicator for presentation and adds the significance thresholds as a source note below the table. Reference levels generally have no separate coefficient test and therefore have no significance indicator.

`rating_table()` accepts fitted models only. A `rating_refinement` specification must first be fitted with `refit()`.

Value

A data frame with classes "rating_table", legacy "riskfactor" and "data.frame". It can be inspected and manipulated directly with ordinary data-frame operations. For backward compatibility, `x$df` returns the same table without the package-specific class and metadata. The table contains:

risk_factor Model term or risk-factor name.

level Factor level or term representation.

Estimate column Coefficient or exponentiated relativity for each supplied model. Its default `est_*` name is derived from the model expression and can be replaced with `estimate_name`.

`signif_*` Optional significance indicator for each model.

Exposure column Optional aggregated exposure, retaining the requested output name.

Author(s)

Martin Haringa

See Also

`as_gt()` for grouped tabular presentation, `autoplot.rating_table()` for graphical comparison, `factor_analysis()` for observed portfolio experience, and `refit()` for fitting a refinement specification.

Examples

```
df <- MTPL
df$zip <- as.factor(df$zip)

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

fitted_effects <- rating_table(
  freq,
  model_data = df,
  exposure = "exposure"
)

fitted_effects
head(fitted_effects)
```

```

# Give the estimate column an explicit name
rating_table(
  freq,
  model_data = df,
  exposure = "exposure",
  estimate_name = "frequency_relativity"
)

# For several models, names can be supplied in model order or by model name
freq_alternative <- update(freq, . ~ . - bm)
rating_table(
  freq,
  freq_alternative,
  model_data = df,
  exposure = "exposure",
  estimate_name = c(
    freq = "current_relativity",
    freq_alternative = "alternative_relativity"
  )
)

# The historical accessor remains available for existing code
identical(fitted_effects$df, as.data.frame(fitted_effects))
if (requireNamespace("gt", quietly = TRUE)) {
  as_gt(fitted_effects)
}

# Keep coefficients on the model scale instead of exponentiating
rating_table(
  freq,
  model_data = df,
  exposure = "exposure",
  exponentiate = FALSE
)

# Significance is supplementary to exposure and stability assessment
rating_table(
  freq,
  model_data = df,
  exposure = "exposure",
  significance = TRUE
)

# Compare two fitted models side by side
freq_simple <- glm(
  nclaims ~ bm + offset(log(exposure)),
  family = poisson(),
  data = df
)

rating_table(
  freq_simple,

```

```

    freq,
    model_data = df,
    exposure = FALSE
  )

# Order all levels by fitted relativity
rating_table(
  freq,
  model_data = df,
  exposure = "exposure",
  level_order = "estimate_descending"
)

```

redistribute_excess_loss

Redistribute large losses for severity or risk-premium modelling

Description

Large claims can have a disproportionate influence on observed severity and on estimated risk-factor effects. `redistribute_excess_loss()` decomposes each selected claim amount into a retained component up to a specified threshold and an excess component above that threshold. The excess component is subsequently allocated using portfolio-wide, risk-factor-level or partially pooled experience. The allocation preserves the total excess loss, subject to numerical tolerance.

The allocated excess can be incorporated in the pricing analysis in two ways:

- `output = "redistributed_claim"` adds the allocated excess loss to the retained claim amount. The resulting variable contains both components and can be used as the response in a single severity model.
- `output = "excess_loading"` keeps retained claim severity and excess loss as separate quantities. The function returns an excess loading per unit of `redistribution_weight`. This loading can be added to the risk premium based on predicted frequency and retained severity.

Both output forms use the same threshold, credibility and allocation calculations. They therefore differ only in how the allocated excess loss is represented in subsequent modelling; the total amount allocated is the same.

The default is `output = "redistributed_claim"`.

Usage

```

redistribute_excess_loss(
  data,
  claim_amount,
  threshold,
  claim_count = NULL,
  redistribution_weight = NULL,
  receives_redistribution = NULL,

```

```

redistribute_excess = NULL,
risk_factor = NULL,
redistribution_method = c("portfolio", "risk_factor", "partial"),
credibility = NULL,
credibility_basis = c("claims", "excess_records"),
credibility_threshold = 50,
credibility_scale = 1,
calculation_details = TRUE,
output = c("redistributed_claim", "excess_loading")
)

```

Arguments

<code>data</code>	A data.frame containing portfolio-level or claim-level observations.
<code>claim_amount</code>	Character string naming a finite, non-negative numeric column with observed claim amounts or aggregate claim loss per row.
<code>threshold</code>	Positive numeric scalar defining the boundary between retained and excess loss. For selected rows, the amount above this value is allocated.
<code>claim_count</code>	Optional character string naming a non-negative, whole-number claim-count column. Claim count identifies claim-bearing rows and is the denominator of the adjusted average claim amount. It is also the default redistribution weight. If NULL, each row with <code>claim_amount > 0</code> is treated as one claim.
<code>redistribution_weight</code>	Optional character string naming a finite, non-negative numeric column. The column determines the relative allocation shares and the unit of the resulting loading. If NULL, claim count is used. Claim count or expected claim count expresses the allocation per claim; earned exposure expresses it per exposure unit. Rows with zero weight receive no allocation. At least one eligible row must have positive weight.
<code>receives_redistribution</code>	Optional character string. Logical column indicating which rows may receive allocated excess loss. Rows with FALSE receive zero, while their observed excess remains in the total allocation unless excluded by <code>redistribute_excess</code> . If NULL, all otherwise eligible rows are included. Eligibility additionally requires positive claim count for redistributed claims and positive redistribution weight for excess loadings.
<code>redistribute_excess</code>	Optional character string. Logical column indicating which rows contribute their excess component to the allocation. Rows with FALSE retain their full observed amount and contribute no excess to the allocation pool. If NULL, every row above threshold contributes.
<code>risk_factor</code>	Optional character string naming the risk-factor column for <code>redistribution_method = "risk_factor"</code> or <code>redistribution_method = "partial"</code> .
<code>redistribution_method</code>	Character string specifying the experience level used in the allocation: "portfolio", "risk_factor" or "partial".

credibility	Optional numeric scalar in $[0, 1]$. For partial redistribution, the supplied value is applied to every risk-factor level. If NULL, credibility is calculated from credibility_basis.
credibility_basis	Character string specifying the experience measure used in automatic credibility: "claims" or "excess_records".
credibility_threshold	Positive numeric scalar representing the amount of credibility-basis experience at which automatic credibility equals 0.5, before applying credibility_scale.
credibility_scale	Non-negative numeric scalar multiplying automatic credibility before truncation to $[0, 1]$.
calculation_details	Logical. If TRUE, append the risk-factor loading, credibility, portfolio loading, blended loading, scaling factor and final loading used in the row-level calculation. If FALSE, these columns are omitted from data but remain available through <code>summary.excess_redistribution()</code> .
output	Character string specifying the representation of allocated excess. "redistributed_claim" adds it to retained claim amounts; "excess_loading" returns it separately per unit of redistribution_weight.

Details

For each row selected through `redistribute_excess`, the observed claim amount is decomposed as:

$$claim_amount = capped_claim_amount + excess_claim_amount$$

If a row is not selected through `redistribute_excess`, its full observed amount is retained. Any amount above the threshold remains available as a diagnostic quantity but is excluded from the amount to be allocated.

The excess amount is allocated over eligible rows in proportion to `redistribution_weight`. If no weight column is supplied, claim count is used. For redistributed-claim output, only rows with a positive claim count are eligible. For excess-loading output, eligibility is determined by a positive redistribution weight, so policy rows without observed claims may receive a loading when exposure is used.

For output = "redistributed_claim", the redistributed claim amount is:

$$adjusted_claim_amount = capped_claim_amount + redistributed_excess$$

This redistributed claim amount is returned as `<claim_amount>_adjusted`. The corresponding average per claim is suitable as the response in a claim-count-weighted severity GLM. Because this response includes allocated excess loss, the same excess component should not subsequently be added to the estimated risk premium.

For output = "excess_loading", capped claim severity remains separate and:

$$excess_loading_i = \frac{allocated_excess_loss_i}{redistribution_weight_i}$$

The loading can be added to the risk premium derived from predicted frequency and retained severity. When earned exposure is used as `redistribution_weight`, the loading is expressed per unit of earned exposure. The total allocated excess loss is preserved, subject to numerical tolerance.

Interpretation of the output forms:

Redistributed-claim output combines retained and allocated loss in one model response. It requires one severity model and assigns the complete historical loss burden to that response. Its interpretation is most direct when the modelled risk-factor levels contain sufficient claim experience and the estimated effects are stable across observation periods.

The allocated component is not an observed loss for the receiving row. For example, an observed claim of 10,000 may receive an allocation of 20,000, resulting in a redistributed amount of 30,000. If the row belongs to a risk-factor level with few claims, the fitted model may attribute a material part of this allocated portfolio experience to that individual level. This can increase the sampling variability of its estimated effect.

Excess-loading output estimates retained severity from observed loss up to the threshold and represents allocated excess as a separate risk-premium component:

$$retained\ risk\ premium = predicted\ frequency \cdot predicted\ retained\ severity$$

$$total\ risk\ premium = retained\ risk\ premium + excess\ loading$$

The two output forms therefore imply different model interpretations rather than different total loss amounts. The selection should reflect claim volume, the stability of risk-factor effects and the intended construction of the technical risk premium.

Sparse risk-factor levels:

Before fitting a redistributed-claim severity model, claim volume should be assessed by risk-factor level. Levels with limited information may be combined using an economically or actuarially meaningful hierarchy. Coefficient stability across periods and agreement between observed and predicted severity provide additional diagnostics. Excess-loading output is an alternative when separate level estimates remain weakly supported.

For example, a model-preparation rule may map sectors with fewer than 20 claims to "Other". The value 20 is illustrative and should not be treated as a general minimum. An appropriate threshold depends on portfolio size, heterogeneity and validation results. Grouping is therefore outside the scope of this function.

Reproducing the redistribution:

The optional calculation columns allow the allocation to be reproduced. For partial redistribution, the loading before total-preservation scaling is:

$$blended_loading = Z_g \cdot risk_factor_loading_g + (1 - Z_g) \cdot portfolio_loading$$

where Z_g denotes the credibility assigned to risk-factor level g . Blending may change the total amount implied by the unscaled loadings. A common scaling factor is therefore applied:

$$preservation_factor = \frac{total\ excess\ to\ redistribute}{\sum_i blended_loading_i \cdot redistribution_weight_i}$$

The final amount received by row i is:

$$redistributed_excess_i = blended_loading_i \cdot preservation_factor \cdot redistribution_weight_i$$

For example, let the sector loading be 30 per unit of weight, the portfolio loading 20 and sector credibility 0.40. The blended loading equals $0.40 \times 30 + 0.60 \times 20 = 24$. With a scaling factor of 1.10, the final loading equals 26.4. A receiving row with redistribution weight 2 is then allocated $26.4 \times 2 = 52.8$. In redistributed-claim output, 52.8 is added to the retained claim amount; in excess-loading output, 26.4 is retained as the loading per unit of weight.

With portfolio redistribution, credibility is zero and the blend equals the portfolio loading. With risk-factor redistribution, credibility is one and the blend equals the risk-factor loading.

Eligibility and redistribution weights:

`receives_redistribution` identifies the rows to which excess loss may be allocated. Rows with value FALSE receive zero. Their observed excess loss is nevertheless included in the total amount to be allocated unless excluded through `redistribute_excess`. For redistributed-claim output, a receiving row must also have a positive claim count. For excess-loading output, a receiving row must have a positive redistribution weight.

`redistribute_excess` controls which large losses contribute their excess part to the redistribution. If it is NULL, every row with `claim_amount > threshold` contributes. For a row marked FALSE, the claim is not capped: its full observed amount is retained and its excess does not enter the allocation pool. This permits specific loss types, such as events treated outside the regular large-loss procedure, to remain unchanged.

`redistribution_weight` controls both the relative shares among receiving rows and the unit of the resulting loading. Claim count produces an amount per claim, expected claim count an amount per expected claim, earned exposure an amount per exposure unit, and insured amount an amount per unit insured. If it is NULL, claim count is used. For an excess loading that is added to an annual risk premium, earned exposure is usually the corresponding unit.

Rows with zero redistribution weight remain in the output but receive zero. For example, `claim_count * insured_amount` assigns a larger share to claim observations with a higher insured amount. The excess threshold and an insured-amount criterion for receiving allocations are separate model specifications.

Redistribution methods:

The `redistribution_method` argument determines the level at which excess experience is estimated:

- "portfolio" derives one loading from the complete allocation portfolio. Risk-factor-level excess experience is not used.
- "risk_factor" derives a separate loading for each risk-factor level. Each level is based only on its own excess experience and allocation weight.
- "partial" blends the risk-factor loading with the portfolio loading using credibility.

For partial redistribution, credibility is either supplied directly through `credibility` or calculated as:

$$Z_g = \frac{n_g}{n_g + \text{credibility_threshold}}$$

With `credibility_basis = "claims"`, `n_g` is the number of claims in the risk-factor level. With `credibility_basis = "excess_records"`, it is the number of records containing a positive excess amount. The resulting value is multiplied by `credibility_scale` and bounded between zero and one.

Credibility and redistribution weight have distinct roles. Credibility determines the contribution of risk-factor-level experience to a partial loading. Redistribution weight determines the row-level allocation shares and the unit of the final loading. Thus, with `credibility_basis = "claims"` and earned exposure as `redistribution_weight`, claim volume determines credibility while the resulting loading is expressed per unit of exposure.

Aggregated portfolio rows:

When a row contains multiple claims, `claim_amount` is treated as the total loss for that row and threshold is applied to that row total. The function cannot identify which individual claims exceeded the threshold from an aggregated row. Claim-level input is required when the threshold is intended to apply separately to each claim.

Value

The input `data.frame` with additional columns and class `"excess_redistribution"`. The object uses standard `data.frame` printing. `summary.excess_redistribution()` aggregates contributed, allocated and shifted loss. Both output forms add:

`<claim_amount>_capped` Observed claim amount capped at threshold when its excess is allocated. Rows excluded through `redistribute_excess` retain their full observed amount.

`<claim_amount>_excess` Observed amount above threshold, whether or not that amount is selected for allocation.

`<claim_amount>_is_excess` Logical indicator that the observed claim amount exceeds threshold.

With `output = "redistributed_claim"`, the result additionally contains:

`<claim_amount>_redistributed_excess` Row-level allocated excess loss. Rows without claims receive zero.

`<claim_amount>_adjusted` Retained claim amount plus row-level allocated excess loss.

`<claim_amount>_adjusted_average` Redistributed claim amount divided by claim count. Rows without claims contain zero.

With `output = "excess_loading"`, the result additionally contains:

`allocated_excess_loss` Absolute excess-loss amount allocated to the row.

`excess_loading` Allocated excess loss per unit of `redistribution_weight`.

With `calculation_details = TRUE`, the result also contains `<risk_factor>_excess_loading`, `<risk_factor>_credibility`, `portfolio_excess_loading`, `blended_excess_loading`, `redistribution_scaling_factor` and `final_redistribution_loading`. For receiving rows, `final_redistribution_loading`

equals `blended_excess_loading` multiplied by `redistribution_scaling_factor`. The selected output and effective redistribution-weight label are stored in the "output" and "redistribution_weight_label" attributes.

Author(s)

Martin Haringa

See Also

[summary.excess_redistribution\(\)](#)

Examples

```
portfolio <- data.frame(
  policy_id = 1:10,
  sector = c(rep("Industry", 5), rep("Retail", 4), "Office"),
  claim_count = c(0, 1, 1, 1, 1, 0, 1, 1, 1, 1),
  claim_amount = c(
    0, 25000, 120000, 50000, 175000,
    0, 40000, 90000, 150000, 300000
  ),
  policy_years = rep(1, 10)
)

# Output form 1: include allocated excess in the severity response.
adjusted <- redistribute_excess_loss(
  portfolio,
  claim_amount = "claim_amount",
  threshold = 100000,
  claim_count = "claim_count",
  risk_factor = "sector",
  redistribution_method = "partial",
  output = "redistributed_claim"
)
summary(adjusted)

# Inspect the row-level calculation. The allocated amount in the final column
# equals final_redistribution_loading times redistribution weight.
adjusted[c(
  "sector_excess_loading", "sector_credibility",
  "portfolio_excess_loading", "blended_excess_loading",
  "redistribution_scaling_factor", "final_redistribution_loading",
  "claim_amount_redistributed_excess"
)]

# Omit row-level calculation columns while retaining them in summary().
compact_adjusted <- redistribute_excess_loss(
  portfolio,
  claim_amount = "claim_amount",
  threshold = 100000,
  claim_count = "claim_count",
```

```

    risk_factor = "sector",
    redistribution_method = "partial",
    calculation_details = FALSE
  )
summary(compact_adjusted)

# Combine levels with limited claim experience before model estimation.
# Three claims is used for this small example; it is not a general minimum.
adjusted$sector_claim_count <- ave(
  adjusted$claim_count, adjusted$sector, FUN = sum
)
adjusted$sector_model <- ifelse(
  adjusted$sector_claim_count >= 3,
  adjusted$sector,
  "Other"
)

# Fit a severity model to redistributed average claim amount. For aggregated
# rows, claim count represents the number of observations underlying each
# average. With one row per claim, this additional weight is unnecessary.
severity_data <- adjusted[adjusted$claim_count > 0, ]
stats::glm(
  claim_amount_adjusted_average ~ sector_model,
  weights = claim_count,
  family = stats::Gamma(link = "log"),
  data = severity_data
)

# Output form 2: estimate retained severity and excess loading separately.
# Using policy years expresses excess_loading per policy year.
loading_result <- redistribute_excess_loss(
  portfolio,
  claim_amount = "claim_amount",
  threshold = 100000,
  claim_count = "claim_count",
  redistribution_weight = "policy_years",
  risk_factor = "sector",
  redistribution_method = "partial",
  output = "excess_loading"
)

frequency_model <- stats::glm(
  claim_count ~ sector + offset(log(policy_years)),
  family = stats::poisson(link = "log"),
  data = loading_result
)
retained_severity_model <- stats::glm(
  claim_amount_capped ~ sector,
  weights = claim_count,
  family = stats::Gamma(link = "log"),
  data = loading_result[loading_result$claim_count > 0, ]
)

```

```

loading_result$predicted_claim_frequency <- stats::predict(
  frequency_model,
  newdata = loading_result,
  type = "response"
) / loading_result$policy_years
loading_result$predicted_retained_severity <- stats::predict(
  retained_severity_model,
  newdata = loading_result,
  type = "response"
)
loading_result$predicted_retained_risk_premium <-
  loading_result$predicted_claim_frequency *
  loading_result$predicted_retained_severity
loading_result$predicted_total_risk_premium <-
  loading_result$predicted_retained_risk_premium +
  loading_result$excess_loading

# Portfolio redistribution estimates one loading across all sectors. Sector-
# specific excess experience does not enter the allocation loading.
portfolio_adjusted <- redistribute_excess_loss(
  portfolio,
  claim_amount = "claim_amount",
  threshold = 100000,
  claim_count = "claim_count",
  redistribution_method = "portfolio"
)
summary(portfolio_adjusted, by = "sector")

# Risk-factor redistribution estimates a separate loading for each sector.
# The estimate for a sector uses only that sector's excess loss and weight.
sector_adjusted <- redistribute_excess_loss(
  portfolio,
  claim_amount = "claim_amount",
  threshold = 100000,
  claim_count = "claim_count",
  risk_factor = "sector",
  redistribution_method = "risk_factor"
)

# Allocate in proportion to claim count times insured amount, restricted to
# policies with an insured amount of at least 100,000.
weighted_portfolio <- transform(
  portfolio,
  insured_amount = rep(c(50000, 250000), each = 5)
)
weighted_portfolio$receives_redistribution <-
  weighted_portfolio$insured_amount >= 100000
weighted_portfolio$redistribution_weight <-
  weighted_portfolio$claim_count * weighted_portfolio$insured_amount

weighted_adjusted <- redistribute_excess_loss(
  weighted_portfolio,
  claim_amount = "claim_amount",

```

```

    threshold = 100000,
    claim_count = "claim_count",
    redistribution_weight = "redistribution_weight",
    receives_redistribution = "receives_redistribution"
  )

  # Exclude catastrophe events and unsettled claims from the allocation pool.
  # Their full observed claim amounts remain retained in the model data.
  portfolio$is_catastrophe <- c(
    FALSE, FALSE, FALSE, FALSE, TRUE,
    FALSE, FALSE, FALSE, FALSE, FALSE
  )
  portfolio$claim_status <- c(
    "settled", "settled", "settled", "settled", "settled",
    "settled", "settled", "open", "settled", "settled"
  )
  portfolio$redistribute_excess <-
    !portfolio$is_catastrophe &
    portfolio$claim_status == "settled"

  selected_adjusted <- redistribute_excess_loss(
    portfolio,
    claim_amount = "claim_amount",
    threshold = 100000,
    claim_count = "claim_count",
    redistribute_excess = "redistribute_excess"
  )

```

refit

Fit a prepared refinement specification

Description

Apply the ordered steps stored in a `rating_refinement` object and fit the resulting pricing GLM. This evaluates the current refinement specification; it may be called repeatedly while smoothing, restrictions, shrinkage, rebasing or sublevel relativities are being reviewed.

Usage

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

Arguments

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

Details

`refit()` applies the stored steps in their recorded order, constructs the required tariff variables and offsets, updates the model formula and calls `stats::glm()` with the original model family. Additional fitting arguments can be supplied through `...`

Actuarial interpretation:

The refitted model represents the combined effect of the original GLM structure and the explicit actuarial assumptions stored in the refinement. Its coefficients and predictions should be assessed against exposure, observed experience, model diagnostics and the unrestricted model. A refit does not establish that a manual restriction or curve edit is statistically estimated; it applies that assumption as specified.

Intercept-only recalibration:

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 their existing fitted relativities. Only the intercept is then estimated. Consequently, relative differences between those fixed effects remain unchanged while the overall expected premium level is recalibrated to the supplied model data.

In practical actuarial work, `intercept_only = TRUE` is generally suitable for a controlled actuarial or commercial refinement of an accepted tariff structure. Examples include a small manual restriction, a limited curve adjustment or a final calibration in which the relativities of unaffected risk factors should remain unchanged.

Use `intercept_only = FALSE` when the refinement forms part of substantive model development. The remaining free model terms are then estimated again, allowing the GLM to account for dependence between risk factors and find a new joint statistical optimum conditional on the fixed refinement steps. Coefficients of risk factors that were not directly refined may therefore also change.

Model result and further refinement:

Printing the returned model first shows the original and refitted formulas, the model family, whether an intercept-only refit was used, and a concise description of every restriction, smoothing, shrinkage or relativity step. This is followed by the regular `glm` output with the model call, coefficients, degrees of freedom, deviance and AIC. The object continues to inherit from `glm`, so standard methods such as `stats::predict.glm()` and `summary.glm()` remain available.

The returned GLM is a fitted result, not an editable refinement specification. Retain the original `rating_refinement` object when further changes may be required. Passing the refitted GLM to `prepare_refinement()` starts a new workflow from that model and does not reconstruct the earlier sequence of refinement steps.

Value

A fitted object inheriting from `glm`. Compatibility classes `refitrestricted`, `refitsmooth`, or both are added when relevant. The object stores refinement metadata used by `rating_table()` and `rating_grid()` to identify fixed relativities, smoothed variables and derived tariff factors.

Author(s)

Martin Haringa

See Also

[prepare_refinement\(\)](#), [add_smoothing\(\)](#), [edit_smoothing\(\)](#), [add_restriction\(\)](#), [add_shrinkage\(\)](#), [add_rebasing\(\)](#), [add_relativities\(\)](#), [rating_table\(\)](#), [rating_grid\(\)](#), [audit_refinement\(\)](#)

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
)

refinement <- prepare_refinement(model) |>
  add_restriction(zip_df)

refined_model <- refit(refinement, intercept_only = TRUE)
```

relativity_specification

Define sublevel relativity specifications

Description

Use [split_level\(\)](#) to describe how one existing GLM factor level is divided into more detailed portfolio levels with specified multiplicative relativities. Use [relativities\(\)](#) to combine one or more of these definitions into the specification supplied to [add_relativities\(\)](#).

Usage

```
split_level(level, new_levels, relativities)

relativities(...)
```

Arguments

<code>level</code>	Character string. Existing level of the risk factor to split.
<code>new_levels</code>	Character vector. Levels of the more detailed portfolio variable within <code>level</code> .
<code>relativities</code>	Numeric vector. Multiplicative relativities corresponding to <code>new_levels</code> . Must have the same length as <code>new_levels</code> .
<code>...</code>	One or more objects created by split_level() .

Details

level identifies the existing parent level in model_variable. new_levels identifies the corresponding levels of split_variable. relativities gives their relative tariff effects before any optional exposure normalisation by `add_relativities()`.

Each call to `split_level()` represents one parent level. Several parent levels can be refined in one step by passing their definitions to `relativities()`. Parent levels must be unique within the combined specification. Levels of the original model variable that are not included remain unsplit.

These helpers assemble and validate explicit tariff assumptions. They do not estimate, normalise or apply the supplied relativities and do not alter a fitted GLM. Exposure normalisation, when requested, is performed by `add_relativities()`.

Value

`split_level()` returns a named list of length one. Its name is level; its value is a data frame with columns new_level and relativity. `relativities()` returns the combined named list expected by the relativities argument of `add_relativities()`.

Author(s)

Martin Haringa

See Also

`add_relativities()`

Examples

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

relativities(
  construction_split,
  split_level(
    "commercial",
    new_levels = c("shop", "office"),
    relativities = c(1.10, 0.90)
  )
)
```

`rgammat`*Simulate severities from a truncated gamma distribution*

Description

Generate random claim severities from a gamma distribution conditional on the result falling inside the interval $(lower, upper)$.

Usage

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

Arguments

<code>n</code>	Integer. Number of observations to generate.
<code>shape</code>	Numeric. Shape parameter of the gamma distribution.
<code>scale</code>	Numeric. Scale parameter of the gamma distribution.
<code>lower</code>	Numeric. Lower truncation bound.
<code>upper</code>	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.

The resulting sample follows the specified conditional distribution; values outside the truncation interval are not generated.

In severity analysis, this can be used for simulation and model checking when the available claims are observed only between a lower reporting threshold and an upper modelling limit. Truncation should not be confused with censoring or capping: the function assumes that values outside the interval are absent rather than recorded at a boundary.

Value

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

Author(s)

Martin Haringa

See Also

[fit_truncated_severity\(\)](#), [rlnormt\(\)](#)

risk_factor_gam	<i>Estimate a smooth effect for a continuous risk factor</i>
-----------------	--

Description

Estimate the relationship between a continuous risk factor and claim frequency, average severity or risk premium with a generalized additive model (GAM). The fitted curve is intended for exploratory risk-factor analysis before selecting a functional form, applying refinement or deriving categorical tariff segments.

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 portfolio observations.
risk_factor	Character string. Numeric continuous risk-factor column in data.
claim_count	Character string. Claim-count column. Required for model = "frequency" and model = "severity".
exposure	Character string. Exposure column used as an offset or aggregation weight.
claim_amount	Optional character string. Total claim-amount column. Required for model = "severity".
pure_premium	Optional character string. Row-level risk-premium column. Required for model = "pure_premium" and aggregated using exposure weights.
model	Character string. Response context: "frequency", "severity" or "pure_premium". The deprecated value "burning" maps to "pure_premium".
round_risk_factor	Optional positive numeric value. The continuous risk factor is rounded to multiples of this value before aggregation and model fitting. This can reduce computation and local volatility when the variable has many distinct values, but it also removes detail.

`x`, `nclaims`, `amount`, `round_x`

Deprecated argument names. Use `risk_factor`, `claim_count`, `claim_amount`, and `round_risk_factor` instead.

Details

Statistical specification:

- "frequency" fits a Poisson GAM to aggregated claim counts with $\log(\text{exposure})$ as offset.
- "severity" fits a Gamma GAM with log link to average claim amount. The response is total claim amount divided by claim count and claim count is used as model weight.
- "pure_premium" fits a Gamma GAM with log link to exposure-weighted risk premium.

Observations are first aggregated by the risk-factor value after optional rounding. Predictions and pointwise confidence intervals are then evaluated over the observed range.

Actuarial interpretation:

The fitted curve describes the marginal pattern in the selected portfolio data. It can reveal non-linearity, broad turning points and areas with sparse support, but it is not by itself a final tariff structure. Correlation with other risk factors, exposure concentration, claim volume, tail observations and stability across periods should be considered before using the pattern in a multivariate GLM.

`autoplot.riskfactor_gam()` can be used to inspect the curve and observed experience. `derive_tariff_segments()` can subsequently translate the smooth pattern into candidate intervals. Alternatively, `add_smoothing()` supports smoothing within the structured refinement workflow.

Column interface and compatibility:

Column names are supplied as character strings. Deprecated `fit_gam()` and `riskfactor_gam()` interfaces remain available for compatibility.

Value

A list of class "risk_factor_gam" with compatibility classes "riskfactor_gam" and "fitgam". It contains:

prediction Prediction grid with fitted values and pointwise confidence limits.

x Name of the continuous risk factor.

model Response context: "frequency", "severity" or "pure_premium".

data Aggregated observed experience and fitted values at observed risk-factor values.

x_obs Risk-factor values in the original portfolio row order, after optional rounding.

round_risk_factor Rounding increment used for the risk factor, or NULL when no rounding was applied.

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.

See Also

`autoplot.riskfactor_gam()`, `derive_tariff_segments()`, `add_smoothing()`

Examples

```
age_frequency <- risk_factor_gam(
  MTPL,
  risk_factor = "age_policyholder",
  claim_count = "nclaims",
  exposure = "exposure",
  model = "frequency"
)

autoplot(age_frequency, show_observations = TRUE)
```

rlnormt

Simulate severities from a truncated lognormal distribution

Description

Generate random claim severities from a lognormal distribution conditional on the result falling inside the interval (*lower*, *upper*).

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.

The resulting sample follows the specified conditional distribution; values outside the truncation interval are not generated.

In severity analysis, this can be used for simulation and model checking when the available claims are observed only between a lower reporting threshold and an upper modelling limit. Truncation should not be confused with censoring or capping: the function assumes that values outside the interval are absent rather than recorded at a boundary.

Value

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

Author(s)

Martin Haringa

See Also

[fit_truncated_severity\(\)](#), [rgammat\(\)](#)

rmse

Calculate response-scale prediction error

Description

Calculate the root mean squared error (RMSE) between observed outcomes and response-scale predictions from a fitted model. RMSE summarises the typical absolute prediction error in the same unit as the model response.

Usage

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

Arguments

<code>x</code>	A fitted model object, for example a "glm".
<code>data</code>	Optional data frame on which the observed response and predictions are evaluated. If NULL, the data stored with the fitted model are used.

Details

RMSE is defined as

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}.$$

In pricing work, RMSE can be used to compare alternative specifications for the same response, portfolio and exposure treatment. Lower values indicate smaller response-scale errors. Because errors are squared, individual large deviations receive relatively high weight. This can be relevant for severity models, but it also makes RMSE sensitive to large claims.

RMSE values should not be compared across responses with different units or scales. A value calculated on the estimation data is an in-sample diagnostic, not an estimate of future predictive performance. Use resampling or separate validation data when out-of-sample performance is required, and interpret RMSE together with calibration, residual and distributional diagnostics.

Value

A numeric value: the root mean squared error.

Author(s)

Martin Haringa

See Also

`model_performance()`, `bootstrap_performance()`, `check_residuals()`

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.

Choosing a reference level does not change fitted values or the overall model fit. It changes the coefficient parameterisation and therefore the level against which the remaining factor relativities are expressed.

Usage

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

Arguments

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

Details

`method = "largest_weight"` is useful when the reference category should represent a substantial and relatively stable part of the portfolio. The supplied weight is commonly earned exposure, but another actuarially meaningful volume measure may be used.

`method = "manual"` is appropriate when the reference category is determined by tariff interpretation, governance or an established pricing convention. The selected category must already be an observed factor level.

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](https://doi.org/10.1007/9783540709985)

See Also

[add_rebasing\(\)](#) for rescaling current tariff relativities within a prepared refinement workflow after a model has been fitted.

Examples

```

portfolio <- data.frame(
  region = factor(c("North", "North", "South", "West")),
  exposure = c(120, 80, 60, 40)
)

set_reference_level(portfolio$region, portfolio$exposure)
set_reference_level(
  portfolio$region,
  method = "manual",
  reference_level = "South"
)

# Apply the largest-weight reference rule to every factor in a data frame
library(dplyr)
df <- chickwts |>
  mutate(across(where(is.character), as.factor)) |>
  mutate(across(where(is.factor), ~set_reference_level(., weight)))

```

split_periods_to_months

Split portfolio periods into calendar months

Description

Split policy periods that cross calendar-month boundaries into separate monthly records. Numeric amounts such as earned exposure and earned premium can be allocated over those records while preserving the amount of each original portfolio row.

Usage

```

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

```

Arguments

data	A data.frame or data.table containing policy or exposure periods.
period_start	Character string. Name of the column with policy period start dates.

<code>period_end</code>	Character string. Name of the column with policy period end dates.
<code>prorate_cols</code>	Character vector with names of numeric columns to prorate over the monthly rows, for example <code>exposure</code> or <code>premium</code> .
<code>df</code> , <code>begin</code> , <code>end</code> , <code>cols</code>	Deprecated argument names kept for backward compatibility. Use <code>data</code> , <code>period_start</code> , <code>period_end</code> , and <code>prorate_cols</code> .

Details

Pricing, reserving and monitoring analyses often require exposure and premium by calendar month, whereas policy administration data generally contains periods with arbitrary start and end dates. The function converts those periods into a monthly representation before aggregation, modelling or reporting.

This is a temporal expansion rather than a portfolio reduction. See [merge_date_ranges\(\)](#) for consolidating connected periods and [active_rows_by_date\(\)](#) for matching dated events to active periods.

Prorated columns are distributed according to the part of the policy period represented by each monthly row. Full months receive weight 1 and partial months use a 30-day convention. The monthly weights are normalised within each source row. Consequently, monthly exposure and premium sum to their original values, including for periods that contain partial months.

Column names are supplied as character strings, for example `period_start = "begin_date"`. The deprecated [period_to_months\(\)](#) interface used unquoted column names and is retained only for backward compatibility.

Expansion and proration are performed internally with `data.table::data.table()` on a local copy. The supplied object is not modified by reference, and the returned object is always a regular `data.frame`.

Value

A regular `data.frame` with one row for each calendar month covered by an original portfolio record. The original columns are retained and an `id` column identifies the source row. Columns supplied through `prorate_cols` contain their allocated monthly amounts.

Author(s)

Martin Haringa

See Also

[active_rows_by_date\(\)](#), [merge_date_ranges\(\)](#), [rating_grid\(\)](#)

Examples

```
portfolio <- data.frame(
  policy_id = c("P001", "P002", "P003"),
  sector = c("Industry", "Retail", "Services"),
  coverage_start = as.Date(c("2025-01-15", "2025-02-01", "2025-03-20")),
  coverage_end = as.Date(c("2025-03-31", "2025-02-28", "2025-05-10")),
```

```
    earned_exposure = c(0.21, 0.08, 0.14),
    earned_premium = c(420, 160, 280)
  )

  # Allocate each policy period and its amounts over calendar months.
  monthly_portfolio <- split_periods_to_months(
    portfolio,
    period_start = "coverage_start",
    period_end = "coverage_end",
    prorate_cols = c("earned_exposure", "earned_premium")
  )
  monthly_portfolio

  # The allocated monthly amounts reconcile to the source portfolio.
  aggregate(
    cbind(earned_exposure, earned_premium) ~ id,
    data = monthly_portfolio,
    FUN = sum
  )

  # Deprecated interface with unquoted column names
  ## Not run:
  period_to_months(
    portfolio,
    coverage_start,
    coverage_end,
    earned_exposure,
    earned_premium
  )

  ## End(Not run)
```

Index

* datasets

- MTPL, 83
- MTPL2, 83

- active_rows_by_date, 3
- active_rows_by_date(), 79, 126
- add_portfolio_experience, 5
- add_portfolio_experience(), 43, 46–49, 73
- add_prediction, 7
- add_prediction(), 59
- add_rebasing, 10
- add_rebasing(), 14, 18, 21, 26, 46, 68, 92, 101, 102, 116, 124
- add_relativities, 12
- add_relativities(), 10, 11, 16, 18, 20, 21, 26, 46, 68, 92, 116, 117
- add_restriction, 15
- add_restriction(), 10, 11, 13, 14, 20, 21, 26, 46, 68, 92, 116
- add_shrinkage, 20
- add_shrinkage(), 10, 11, 14, 18, 26, 46, 68, 92, 116
- add_smoothing, 22
- add_smoothing(), 14, 18, 21, 45, 46, 64–68, 90, 92, 116, 120, 121
- add_tariff_segments, 27
- add_tariff_segments(), 51, 64, 65
- as_gt, 32
- as_gt(), 48, 55, 90, 102, 103
- as_gt.premium_change (premium_change), 88
- as_gt.rating_table(), 49
- assess_excess_threshold, 29
- assess_excess_threshold(), 32
- audit_refinement, 36
- audit_refinement(), 11, 17, 21, 59, 116
- autoplot.bootstrap_performance, 38
- autoplot.bootstrap_performance(), 57
- autoplot.check_residuals, 39
- autoplot.factor_analysis, 40
- autoplot.factor_analysis(), 73
- autoplot.rating_refinement, 44
- autoplot.rating_refinement(), 90
- autoplot.rating_table, 46
- autoplot.rating_table(), 5, 6, 102, 103
- autoplot.riskfactor_gam
 - (autoplot.tariff_segments), 49
- autoplot.riskfactor_gam(), 120, 121
- autoplot.tariff_effect
 - (autoplot.tariff_segments), 49
- autoplot.tariff_segments, 49
- autoplot.tariff_segments(), 64, 65
- autoplot.truncated_dist
 - (autoplot.truncated_severity), 52
- autoplot.truncated_severity, 52

- bootstrap_coefficients, 54
- bootstrap_coefficients(), 32
- bootstrap_performance, 56
- bootstrap_performance(), 38, 39, 55, 82, 123

- calibrate_model, 58
- check_overdispersion, 59
- check_overdispersion(), 40, 82
- check_residuals, 61
- check_residuals(), 39, 40, 82, 123
- classInt::classIntervals(), 74

- data.table::data.table(), 78, 95, 126
- data.table::foverlaps(), 4
- dbplyr::sql_render(), 81, 99
- derive_tariff_segments, 62
- derive_tariff_segments(), 27, 28, 49–51, 120, 121
- DHARMA::simulateResiduals(), 61
- dplyr::arrange(), 99
- dplyr::collect(), 81, 98, 99

- dplyr::tbl(), 80, 98
- edit_smoothing, 66
- edit_smoothing(), 26, 45, 46, 90, 92, 116
- evtree::evtree(), 63
- extract_model_data, 70
- extract_model_data(), 94–96
- factor_analysis, 72
- factor_analysis(), 5, 6, 40, 41, 43, 49, 102, 103
- fisher_classify, 74
- fit_gam(), 120
- fit_truncated_severity, 75
- fit_truncated_severity(), 53, 118, 122
- fitdistrplus::fitdist(), 76
- ggplot2::ggplot, 38, 40, 85
- ggplot2::theme(), 48
- merge_date_ranges, 77
- merge_date_ranges(), 4, 80, 81, 93, 94, 96, 99, 126
- merge_date_ranges_db, 80
- merge_date_ranges_db(), 78, 79, 96, 99
- mgcv::gam(), 25
- model_performance, 81
- model_performance(), 55, 57, 123
- MTPL, 83
- MTPL2, 83
- outlier_histogram, 84
- period_to_months(), 126
- plot_severity_distribution, 86
- premium_change, 88
- prepare_refinement, 91
- prepare_refinement(), 10–12, 14, 16, 18, 20, 21, 23, 26, 37, 46, 65, 66, 68, 71, 114–116
- rating_grid, 93
- rating_grid(), 4, 36, 37, 71, 77–79, 81, 98, 99, 115, 116, 126
- rating_grid_db, 98
- rating_grid_db(), 79, 81, 96
- rating_table, 100
- rating_table(), 5, 6, 13, 14, 17, 18, 32, 34, 36, 46–49, 59, 71, 73, 96, 115, 116
- redistribute_excess_loss, 105
- redistribute_excess_loss(), 29
- refit, 114
- refit(), 11–14, 16–18, 21, 23–26, 36, 37, 44–46, 58, 59, 67, 68, 91, 92, 100, 103
- relativities
 - (relativity_specification), 116
- relativities(), 13, 14
- relativity_specification, 116
- rgammat, 118
- rgammat(), 53, 122
- risk_factor_gam, 119
- risk_factor_gam(), 26, 28, 49–51, 62, 63, 65
- riskfactor_gam(), 120
- rlnormt, 121
- rlnormt(), 53, 118
- rmse, 122
- rmse(), 39, 57, 82
- scam::scam(), 25
- set_reference_level, 123
- set_reference_level(), 11
- smooth_coef(), 25
- split_level (relativity_specification), 116
- split_level(), 13, 14
- split_periods_to_months, 125
- split_periods_to_months(), 4, 79
- stats::glm(), 114, 115
- stats::predict.glm(), 115
- summary.bootstrap_coefficients(), 54, 55
- summary.excess_redistribution(), 107, 110, 111
- summary.glm(), 115
- summary.rating_refinement(), 37, 92
- summary.refinement_audit(), 37
- univariate(), 41, 73