

Package ‘insurancerating’

October 9, 2024

Type Package

Title Analytic Insurance Rating Techniques

Version 0.7.5

Maintainer Martin Haringa <mharinga@gmail.com>

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

Description Functions to build, evaluate, and visualize insurance rating models. It simplifies the process of modeling premiums, and allows to analyze insurance risk factors effectively. The package employs a data-driven strategy for constructing insurance tariff classes, drawing on the work 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

RoxygenNote 7.3.2

Imports ciTools, classInt, colorspace, data.table, DHARMA, dplyr, evtree, fitdistrplus, ggplot2, insight, lubridate, mgcv, patchwork, scales, scam, stringr

Depends R (>= 3.3)

Suggests spelling, knitr, rmarkdown, testthat

Language en-US

NeedsCompilation no

Author Martin Haringa [aut, cre]

Repository CRAN

Date/Publication 2024-10-09 17:20:02 UTC

Contents

add_prediction	3
autoplot.bootstrap_rmse	3
autoplot.check_residuals	4
autoplot.constructtariffclasses	5
autoplot.fitgam	6
autoplot.restricted	8
autoplot.riskfactor	8
autoplot.smooth	10
autoplot.truncated_dist	10
autoplot.univariate	11
biggest_reference	13
bootstrap_rmse	14
check_overdispersion	16
check_residuals	17
construct_model_points	18
construct_tariff_classes	19
fisher	21
fit_gam	22
fit_truncated_dist	24
histbin	26
model_data	27
model_performance	27
MTPL	28
MTPL2	29
period_to_months	30
rating_factors	31
reduce	32
refit_glm	34
restrict_coef	34
rgammat	36
rlnormt	36
rmse	37
rows_per_date	38
smooth_coef	39
summary.reduce	42
univariate	42
update_glm	44

Index

45

add_prediction	<i>Add predictions to a data frame</i>
----------------	--

Description

Add model predictions and confidence bounds to a data frame.

Usage

```
add_prediction(data, ..., var = NULL, conf_int = FALSE, alpha = 0.1)
```

Arguments

data	a data frame of new data.
...	one or more objects of class <code>glm</code> .
var	the name of the output column(s), defaults to <code>NULL</code>
conf_int	determines whether confidence intervals will be shown. Defaults to <code>conf_int = FALSE</code> .
alpha	a real number between 0 and 1. Controls the confidence level of the interval estimates (defaults to 0.10, representing 90 percent confidence interval).

Value

data.frame

Examples

```
mod1 <- glm(nclaims ~ age_policyholder, data = MTPL,  
  offset = log(exposure), family = poisson())  
mtpl_pred <- add_prediction(MTPL, mod1)  
  
# Include confidence bounds  
mtpl_pred_ci <- add_prediction(MTPL, mod1, conf_int = TRUE)
```

autoplot.bootstrap_rmse

Automatically create a ggplot for objects obtained from bootstrap_rmse()

Description

Takes an object produced by `bootstrap_rmse()`, and plots the simulated RMSE

Usage

```
## S3 method for class 'bootstrap_rmse'
autoplot(object, fill = NULL, color = NULL, ...)
```

Arguments

object	bootstrap_rmse object produced by bootstrap_rmse()
fill	color to fill histogram (default is "steelblue")
color	color to plot line colors of histogram
...	other plotting parameters to affect the plot

Value

a ggplot object

Author(s)

Martin Haringa

autoplot.check_residuals

*Automatically create a ggplot for objects obtained from
check_residuals()*

Description

Takes an object produced by check_residuals(), and produces a uniform quantile-quantile plot.#'

Usage

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

Arguments

object	check_residuals object produced by check_residuals()
show_message	show output from test (defaults to TRUE)
...	other plotting parameters to affect the plot

Value

a ggplot object

Author(s)

Martin Haringa

```
autoplot.constructtariffclasses
```

Automatically create a ggplot for objects obtained from construct_tariff_classes()

Description

Takes an object produced by `construct_tariff_classes()`, and plots the fitted GAM. In addition the constructed tariff classes are shown.

Usage

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

Arguments

<code>object</code>	constructtariffclasses object produced by <code>construct_tariff_classes</code>
<code>conf_int</code>	determines whether 95% The default is <code>conf_int = FALSE</code>
<code>color_gam</code>	a color can be specified either by name (e.g.: "red") or by hexadecimal code (e.g. : "#FF1234") (default is "steelblue")
<code>show_observations</code>	add observed frequency/severity points for each level of the variable for which tariff classes are constructed
<code>color_splits</code>	change the color of the splits in the graph ("grey50" is default)
<code>size_points</code>	size for points (1 is default)
<code>color_points</code>	change the color of the points in the graph ("black" is default)
<code>rotate_labels</code>	rotate x-labels 45 degrees (this might be helpful for overlapping x-labels)
<code>remove_outliers</code>	do not show observations above this number in the plot. This might be helpful for outliers.
<code>...</code>	other plotting parameters to affect the plot

Value

a ggplot object

Author(s)

Martin Haringa

Examples

```
## Not run:
library(ggplot2)
library(dplyr)
x <- fit_gam(MTPL,
nclaims = nclaims, x = age_policyholder, exposure = exposure) |>
  construct_tariff_classes()
autoplot(x, show_observations = TRUE)

## End(Not run)
```

autoplot.fitgam

Automatically create a ggplot for objects obtained from fit_gam()

Description

Takes an object produced by `fit_gam()`, and plots the fitted GAM.

Usage

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

Arguments

object	fitgam object produced by fit_gam()
conf_int	determines whether 95 percent confidence intervals will be plotted. The default is conf_int = FALSE.
color_gam	a color can be specified either by name (e.g.: "red") or by hexadecimal code (e.g. : "#FF1234") (default is "steelblue")
show_observations	add observed frequency/severity points for each level of the variable for which tariff classes are constructed
x_stepsize	set step size for labels horizontal axis
size_points	size for points (1 is default)
color_points	change the color of the points in the graph ("black" is default)
rotate_labels	rotate x-labels 45 degrees (this might be helpful for overlapping x-labels)
remove_outliers	do not show observations above this number in the plot. This might be helpful for outliers.
...	other plotting parameters to affect the plot

Value

a ggplot object

Author(s)

Martin Haringa

Examples

```
## Not run:
library(ggplot2)
library(dplyr)
fit_gam(MTPL, nclaims = nclaims, x = age_policyholder,
        exposure = exposure) |>
  autoplot(show_observations = TRUE)

## End(Not run)
```

autoplot.restricted *Automatically create a ggplot for objects obtained from restrict_coef()*

Description

[Experimental] Takes an object produced by `restrict_coef()`, and produces a line plot with a comparison between the restricted coefficients and estimated coefficients obtained from the model.

Usage

```
## S3 method for class 'restricted'  
autoplot(object, ...)
```

Arguments

`object` object produced by `restrict_coef()`
`...` other plotting parameters to affect the plot

Value

Object of class `ggplot2`

Author(s)

Martin Haringa

Examples

```
freq <- glm(nclaims ~ bm + zip, weights = power, family = poisson(),  
          data = MTPL)  
zip_df <- data.frame(zip = c(0,1,2,3), zip_rst = c(0.8, 0.9, 1, 1.2))  
freq |>  
  restrict_coef(restrictions = zip_df) |>  
  autoplot()
```

autoplot.riskfactor *Automatically create a ggplot for objects obtained from rating_factors()*

Description

Takes an object produced by `rating_factors()`, and plots the available input.

Usage

```
## S3 method for class 'riskfactor'
autoplot(
  object,
  risk_factors = NULL,
  ncol = 1,
  labels = TRUE,
  dec.mark = ",",
  ylab = "rate",
  fill = NULL,
  color = NULL,
  linetype = FALSE,
  ...
)
```

Arguments

object	riskfactor object produced by rating_factors()
risk_factors	character vector to define which factors are included. Defaults to all risk factors.
ncol	number of columns in output (default is 1)
labels	show labels with the exposure (default is TRUE)
dec.mark	control the format of the decimal point, as well as the mark between intervals before the decimal point, choose either "," (default) or "."
ylab	modify label for the y-axis
fill	color to fill histogram
color	color to plot line colors of histogram (default is "skyblue")
linetype	use different linetypes (default is FALSE)
...	other plotting parameters to affect the plot

Value

a ggplot2 object

Author(s)

Martin Haringa

Examples

```
library(dplyr)
df <- MTPL2 |>
  mutate(across(c(area), as.factor)) |>
  mutate(across(c(area), ~biggest_reference(., exposure)))

mod1 <- glm(nclaims ~ area + premium, offset = log(exposure),
  family = poisson(), data = df)
mod2 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(),
```

```

data = df)

x <- rating_factors(mod1, mod2, model_data = df, exposure = exposure)
autoplot(x)

```

```

autoplot.smooth      Automatically create a ggplot for objects obtained from smooth_coef()

```

Description

[Experimental] Takes an object produced by `smooth_coef()`, and produces a plot with a comparison between the smoothed coefficients and estimated coefficients obtained from the model.

Usage

```

## S3 method for class 'smooth'
autoplot(object, ...)

```

Arguments

<code>object</code>	object produced by <code>smooth_coef()</code>
<code>...</code>	other plotting parameters to affect the plot

Value

Object of class `ggplot2`

Author(s)

Martin Haringa

```

autoplot.truncated_dist
      Automatically create a ggplot for objects obtained from
      fit_truncated_dist()

```

Description

Takes an object produced by `fit_truncated_dist()`, and plots the available input.

Usage

```
## S3 method for class 'truncated_dist'
autoplot(
  object,
  geom_ecdf = c("point", "step"),
  xlab = NULL,
  ylab = NULL,
  ylim = c(0, 1),
  xlim = NULL,
  print_title = TRUE,
  print_dig = 2,
  print_trunc = 2,
  ...
)
```

Arguments

object	object univariate object produced by <code>fit_truncated_dist()</code>
geom_ecdf	the geometric object to use display the data (point or step)
xlab	the title of the x axis
ylab	the title of the y axis
ylim	two numeric values, specifying the lower limit and the upper limit of the scale
xlim	two numeric values, specifying the left limit and the right limit of the scale
print_title	show title (default to TRUE)
print_dig	number of digits for parameters in title (default 2)
print_trunc	number of digits for truncation values to print
...	other plotting parameters to affect the plot

Value

a ggplot2 object

Author(s)

Martin Haringa

autoplot.univariate *Automatically create a ggplot for objects obtained from univariate()*

Description

Takes an object produced by `univariate()`, and plots the available input.

Usage

```
## S3 method for class 'univariate'
autoplot(
  object,
  show_plots = 1:9,
  ncol = 1,
  background = TRUE,
  labels = TRUE,
  sort = FALSE,
  sort_manual = NULL,
  dec.mark = ",",
  color = "dodgerblue",
  color_bg = "lightskyblue",
  label_width = 10,
  coord_flip = FALSE,
  show_total = FALSE,
  total_color = NULL,
  total_name = NULL,
  rotate_angle = NULL,
  custom_theme = NULL,
  remove_underscores = FALSE,
  ...
)
```

Arguments

object	univariate object produced by univariate()
show_plots	numeric vector of plots to be shown (default is c(1,2,3,4,5,6,7,8,9)), there are nine available plots: <ul style="list-style-type: none"> • 1. frequency (i.e. number of claims / exposure) • 2. average severity (i.e. severity / number of claims) • 3. risk premium (i.e. severity / exposure) • 4. loss ratio (i.e. severity / premium) • 5. average premium (i.e. premium / exposure) • 6. exposure • 7. severity • 8. nclaims • 9. premium
ncol	number of columns in output (default is 1)
background	show exposure as a background histogram (default is TRUE)
labels	show labels with the exposure (default is TRUE)
sort	sort (or order) risk factor into descending order by exposure (default is FALSE)
sort_manual	sort (or order) risk factor into own ordering; should be a character vector (default is NULL)
dec.mark	decimal mark; defaults to ","

color	change the color of the points and line ("dodgerblue" is default)
color_bg	change the color of the histogram ("#f8e6b1" is default)
label_width	width of labels on the x-axis (10 is default)
coord_flip	flip cartesian coordinates so that horizontal becomes vertical, and vertical, horizontal (default is FALSE)
show_total	show line for total if by is used in univariate (default is FALSE)
total_color	change the color for the total line ("black" is default)
total_name	add legend name for the total line (e.g. "total")
rotate_angle	numeric value for angle of labels on the x-axis (degrees)
custom_theme	list with customized theme options
remove_underscores	logical. Defaults to FALSE. Remove underscores from labels
...	other plotting parameters to affect the plot

Value

a ggplot2 object

Author(s)

Marc Haine, Martin Haringa

Examples

```
library(ggplot2)
x <- univariate(MTPL2, x = area, severity = amount, nclaims = nclaims,
exposure = exposure)
autoplot(x)
autoplot(x, show_plots = c(6,1), background = FALSE, sort = TRUE)

# Group by `zip`
xzip <- univariate(MTPL, x = bm, severity = amount, nclaims = nclaims,
exposure = exposure, by = zip)
autoplot(xzip, show_plots = 1:2)
```

biggest_reference *Set reference group to the group with largest exposure*

Description

This function specifies the first level of a factor to the level with the largest exposure. Levels of factors are sorted using an alphabetic ordering. If the factor is used in a regression context, then the first level will be the reference. For insurance applications it is common to specify the reference level to the level with the largest exposure.

Usage

```
biggest_reference(x, weight)
```

Arguments

x an unordered factor
weight a vector containing weights (e.g. exposure). Should be numeric.

Value

a factor of the same length as x

Author(s)

Martin Haringa

References

Kaas, Rob & Goovaerts, Marc & Dhaene, Jan & Denuit, Michel. (2008). Modern Actuarial Risk Theory: Using R. doi:10.1007/978-3-540-70998-5.

Examples

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

bootstrap_rmse	<i>Bootstrapped RMSE</i>
----------------	--------------------------

Description

Generate n bootstrap replicates to compute n root mean squared errors.

Usage

```
bootstrap_rmse(  
  model,  
  data,  
  n = 50,  
  frac = 1,  
  show_progress = TRUE,  
  rmse_model = NULL  
)
```

Arguments

model	a model object
data	data used to fit model object
n	number of bootstrap replicates (defaults to 50)
frac	fraction used in training set if cross-validation is applied (defaults to 1)
show_progress	show progress bar (defaults to TRUE)
rmse_model	numeric RMSE to show as vertical dashed line in autoplot() (defaults to NULL)

Details

To test the predictive ability of the fitted model it might be helpful to determine the variation in the computed RMSE. The variation is calculated by computing the root mean squared errors from n generated bootstrap replicates. More precisely, for each iteration a sample with replacement is taken from the data set and the model is refitted using this sample. Then, the root mean squared error is calculated.

Value

A list with components

rmse_bs	numerical vector with n root mean squared errors
rmse_mod	root mean squared error for fitted (i.e. original) model

Author(s)

Martin Haringa

Examples

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

# Use all records in MTPL
x <- bootstrap_rmse(mod1, MTPL, n = 80, show_progress = FALSE)
print(x)
autoplot(x)

# Use 80% of records to test whether predictive ability depends on which 80%
# is used. This might for example be useful in case portfolio contains large
# claim sizes
x_frac <- bootstrap_rmse(mod1, MTPL, n = 50, frac = .8,
  show_progress = FALSE)
autoplot(x_frac) # Variation is quite small for Poisson GLM

## End(Not run)
```

check_overdispersion *Check overdispersion of Poisson GLM*

Description

Check Poisson GLM for overdispersion.

Usage

```
check_overdispersion(object)
```

Arguments

object fitted model of class glm and family Poisson

Details

A dispersion ratio larger than one indicates overdispersion, this occurs when the observed variance is higher than the variance of the theoretical model. If the dispersion ratio is close to one, a Poisson model fits well to the data. A p-value < .05 indicates overdispersion. Overdispersion > 2 probably means there is a larger problem with the data: check (again) for outliers, obvious lack of fit. Adopted from `performance::check_overdispersion()`.

Value

A list with dispersion ratio, chi-squared statistic, and p-value.

Author(s)

Martin Haringa

References

- Bolker B et al. (2017): [GLMM FAQ](#).

Examples

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

check_residuals	<i>Check model residuals</i>
-----------------	------------------------------

Description

Detect overall deviations from the expected distribution.

Usage

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

Arguments

object	a model object
n_simulations	number of simulations (defaults to 30)

Details

Misspecifications in GLMs cannot reliably be diagnosed with standard residual plots, and GLMs are thus often not as thoroughly checked as LMs. One reason why GLMs residuals are harder to interpret is that the expected distribution of the data changes with the fitted values. As a result, standard residual plots, when interpreted in the same way as for linear models, seem to show all kind of problems, such as non-normality, heteroscedasticity, even if the model is correctly specified. `check_residuals()` aims at solving these problems by creating readily interpretable residuals for GLMs that are standardized to values between 0 and 1, and that can be interpreted as intuitively as residuals for the linear model. This is achieved by a simulation-based approach, similar to the Bayesian p-value or the parametric bootstrap, that transforms the residuals to a standardized scale. This explanation is adopted from `DHARMA::simulateResiduals()`.

It might happen that in the fitted model for a data point all simulations have the same value (e.g. zero), this returns the error message `Error in approxfun: need at least two non-NA values to interpolate*`. If that is the case, it could help to increase the number of simulations.

Value

Invisibly returns the p-value of the test statistics. A p-value < 0.05 indicates a significant deviation from expected distribution.

Author(s)

Martin Haringa

References

Dunn, K. P., and Smyth, G. K. (1996). Randomized quantile residuals. *Journal of Computational and Graphical Statistics* 5, 1-10.

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

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)
check_residuals(m1, n_simulations = 50) |> autoplot()

## End(Not run)
```

construct_model_points

Construct model points from Generalized Linear Model

Description

[Experimental] `construct_model_points()` is used to construct model points from generalized linear models, and must be preceded by `model_data()`. `construct_model_points()` can also be used in combination with a `data.frame`.

Usage

```
construct_model_points(
  x,
  exposure = NULL,
  exposure_by = NULL,
  agg_cols = NULL,
  drop_na = FALSE
)
```

Arguments

<code>x</code>	Object of class <code>model_data</code> or of class <code>data.frame</code>
<code>exposure</code>	column with exposure
<code>exposure_by</code>	split column exposure by (e.g. year)
<code>agg_cols</code>	list of columns to aggregate (sum) by, e.g. number of claims
<code>drop_na</code>	drop na values (default to FALSE)

Value

`data.frame`

Author(s)

Martin Haringa

Examples

```
## Not run:
# With data.frame
library(dplyr)
mtcars |>
  select(cyl, vs) |>
  construct_model_points()

mtcars |>
  select(cyl, vs, disp) |>
  construct_model_points(exposure = disp)

mtcars |>
  select(cyl, vs, disp, gear) |>
  construct_model_points(exposure = disp, exposure_by = gear)

mtcars |>
  select(cyl, vs, disp, gear, mpg) |>
  construct_model_points(exposure = disp, exposure_by = gear,
    agg_cols = list(mpg))

# With glm
library(datasets)
data1 <- warpbreaks |>
  mutate(jaar = c(rep(2000, 10), rep(2010, 44))) |>
  mutate(exposure = 1) |>
  mutate(nclaims = 2)

pmodel <- glm(breaks ~ wool + tension, data1, offset = log(exposure),
  family = poisson(link = "log"))

model_data(pmodel) |>
  construct_model_points()

model_data(pmodel) |>
  construct_model_points(agg_cols = list(nclaims))

model_data(pmodel) |>
  construct_model_points(exposure = exposure, exposure_by = jaar) |>
  add_prediction(pmodel)

## End(Not run)
```

Description

Constructs insurance tariff classes to `fitgam` objects produced by `fit_gam`. The goal is to bin the continuous risk factors such that categorical risk factors result which capture the effect of the covariate on the response in an accurate way, while being easy to use in a generalized linear model (GLM).

Usage

```
construct_tariff_classes(
  object,
  alpha = 0,
  niterations = 10000,
  ntrees = 200,
  seed = 1
)
```

Arguments

<code>object</code>	fitgam object produced by <code>fit_gam</code>
<code>alpha</code>	complexity parameter. The complexity parameter (<code>alpha</code>) is used to control the number of tariff classes. Higher values for <code>alpha</code> render less tariff classes. (<code>alpha = 0</code> is default).
<code>niterations</code>	in case the run does not converge, it terminates after a specified number of iterations defined by <code>niterations</code> .
<code>ntrees</code>	the number of trees in the population.
<code>seed</code>	an numeric seed to initialize the random number generator (for reproducibility).

Details

Evolutionary trees are used as a technique to bin the `fitgam` object produced by `fit_gam` into risk homogeneous categories. This method is based on the work by Henckaerts et al. (2018). See Grubinger et al. (2014) for more details on the various parameters that control aspects of the `evtree` fit.

Value

A list of class `constructtariffclasses` with components

<code>prediction</code>	data frame with predicted values
<code>x</code>	name of continuous risk factor for which tariff classes are constructed
<code>model</code>	either 'frequency', 'severity' or 'burning'
<code>data</code>	data frame with predicted values and observed values
<code>x_obs</code>	observations for continuous risk factor
<code>splits</code>	vector with boundaries of the constructed tariff classes
<code>tariff_classes</code>	values in vector <code>x</code> coded according to which constructed tariff class they fall

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/s10182-011-0152-7.

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

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

Wood, S.N. (2011). Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society (B)* 73(1):3-36. doi:10.1111/j.1467-9868.2010.00749.x.

Examples

```
## Not run:
library(dplyr)
fit_gam(MTPL, nclaims = nclaims,
x = age_policyholder, exposure = exposure) |>
  construct_tariff_classes()

## End(Not run)
```

fisher

Fisher's natural breaks classification

Description

The function provides an interface to finding class intervals for continuous numerical variables, for example for choosing colours for plotting maps.

Usage

```
fisher(vec, n = 7, diglab = 2)
```

Arguments

vec	a continuous numerical variable
n	number of classes required (n = 7 is default)
diglab	number of digits (n = 2 is default)

Details

The "fisher" style uses the algorithm proposed by W. D. Fisher (1958) and discussed by Slocum et al. (2005) as the Fisher-Jenks algorithm. This function is adopted from the classInt package.

Value

Vector with clustering

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.

fit_gam

Generalized additive model

Description

Fits a generalized additive model (GAM) to continuous risk factors in one of the following three types of models: the number of reported claims (claim frequency), the severity of reported claims (claim severity) or the burning cost (i.e. risk premium or pure premium).

Usage

```
fit_gam(  
  data,  
  nclaims,  
  x,  
  exposure,  
  amount = NULL,  
  pure_premium = NULL,  
  model = "frequency",  
  round_x = NULL  
)
```

Arguments

data	data.frame of an insurance portfolio
nclaims	column in data with number of claims
x	column in data with continuous risk factor
exposure	column in data with exposure
amount	column in data with claim amount
pure_premium	column in data with pure premium
model	choose either 'frequency', 'severity' or 'burning' (model = 'frequency' is default). See details section.
round_x	round elements in column x to multiple of round_x. This gives a speed enhancement for data containing many levels for x.

Details

The 'frequency' specification uses a Poisson GAM for fitting the number of claims. The logarithm of the exposure is included as an offset, such that the expected number of claims is proportional to the exposure.

The 'severity' specification uses a lognormal GAM for fitting the average cost of a claim. The average cost of a claim is defined as the ratio of the claim amount and the number of claims. The number of claims is included as a weight.

The 'burning' specification uses a lognormal GAM for fitting the pure premium of a claim. The pure premium is obtained by multiplying the estimated frequency and the estimated severity of claims. The word burning cost is used here as equivalent of risk premium and pure premium. Note that the functionality for fitting a GAM for pure premium is still experimental (in the early stages of development).

Value

A list with components

prediction	data frame with predicted values
x	name of continuous risk factor
model	either 'frequency', 'severity' or 'burning'
data	data frame with predicted values and observed values
x_obs	observations for continuous risk factor

Author(s)

Martin Haringa

References

- Antonio, K. and Valdez, E. A. (2012). Statistical concepts of a priori and a posteriori risk classification in insurance. *Advances in Statistical Analysis*, 96(2):187–224. doi:10.1007/s10182-011-0152-7.
- Grubinger, T., Zeileis, A., and Pfeiffer, K.-P. (2014). evtree: Evolutionary learning of globally optimal classification and regression trees in R. *Journal of Statistical Software*, 61(1):1–29. doi:10.18637/jss.v061.i01.
- Henckaerts, R., Antonio, K., Clijsters, M. and Verbelen, R. (2018). A data driven binning strategy for the construction of insurance tariff classes. *Scandinavian Actuarial Journal*, 2018:8, 681-705. doi:10.1080/03461238.2018.1429300.
- Wood, S.N. (2011). Fast stable restricted maximum likelihood and marginal likelihood estimation of semiparametric generalized linear models. *Journal of the Royal Statistical Society (B)* 73(1):3-36. doi:10.1111/j.1467-9868.2010.00749.x.

Examples

```
fit_gam(MTPL, nclaims = nclaims, x = age_policyholder,
        exposure = exposure)
```

```
fit_truncated_dist    Fit a distribution to truncated severity (loss) data
```

Description

[Experimental] Estimate the original distribution from truncated data. Truncated data arise frequently in insurance studies. It is common that only claims above a certain threshold are known.

Usage

```
fit_truncated_dist(
  y,
  dist = c("gamma", "lognormal"),
  left = NULL,
  right = NULL,
  start = NULL,
  print_initial = TRUE
)
```

Arguments

y	vector with observations of losses
dist	distribution for severity ("gamma" or "lognormal"). Defaults to "gamma".
left	numeric. Observations below this threshold are not present in the sample.
right	numeric. Observations above this threshold are not present in the sample. Defaults to Inf.
start	list of starting parameters for the algorithm.
print_initial	print attempts for initial parameters.

Value

fitdist returns an object of class "fitdist"

Author(s)

Martin Haringa

Examples

```
## Not run:
# Original observations for severity
set.seed(1)
e <- rgamma(1000, scale = 148099.5, shape = 0.4887023)

# Truncated data (only claims above 30.000 euros)
threshold <- 30000
f <- e[e > threshold]

library(dplyr)
library(ggplot2)
data.frame(value = c(e, f),
  variable = rep(c("Original data", "Only claims above 30.000 euros"),
    c(length(e), length(f)))) %>%
  filter(value < 5e5) %>%
  mutate(value = value / 1000) %>%
  ggplot(aes(x = value)) +
  geom_histogram(colour = "white") +
  facet_wrap(~variable, ncol = 1) +
  labs(y = "Number of observations",
    x = "Severity (x 1000 EUR)")

# scale = 156259.7 and shape = 0.4588. Close to parameters of original
# distribution!
x <- fit_truncated_dist(f, left = threshold, dist = "gamma")

# Print cdf
autoplot(x)

# CDF with modifications
autoplot(x, print_dig = 5, xlab = "loss", ylab = "cdf", ylim = c(.9, 1))

est_scale <- x$estimate[1]
est_shape <- x$estimate[2]

# Generate data from truncated distribution (between 30k en 20 mln)
rg <- rgammat(10, scale = est_scale, shape = est_shape, lower = 3e4,
  upper = 20e6)

# Calculate quantiles
quantile(rg, probs = c(.5, .9, .99, .995))

## End(Not run)
```

histbin*Create a histogram with outlier bins*

Description

Visualize the distribution of a single continuous variable by dividing the x axis into bins and counting the number of observations in each bin. Data points that are considered outliers can be binned together. This might be helpful to display numerical data over a very wide range of values in a compact way.

Usage

```
histbin(  
  data,  
  x,  
  left = NULL,  
  right = NULL,  
  line = FALSE,  
  bins = 30,  
  fill = NULL,  
  color = NULL,  
  fill_outliers = "#a7d1a7"  
)
```

Arguments

<code>data</code>	<code>data.frame</code>
<code>x</code>	variable name in <code>data.frame data</code> that should be mapped
<code>left</code>	numeric indicating the floor of the range
<code>right</code>	numeric indicating the ceiling of the range
<code>line</code>	show density line (default is <code>FALSE</code>)
<code>bins</code>	numeric to indicate number of bins
<code>fill</code>	color used to fill bars
<code>color</code>	color for bar lines
<code>fill_outliers</code>	color used to fill outlier bars

Details

Wrapper function around `ggplot2::geom_histogram()`. The method is based on suggestions from <https://edwinth.github.io/blog/outlier-bin/>.

Value

a `ggplot2` object

Author(s)

Martin Haringa

Examples

```
histbin(MTPL2, premium)
histbin(MTPL2, premium, left = 30, right = 120, bins = 30)
```

model_data	<i>Get model data</i>
------------	-----------------------

Description

[Experimental] `model_data()` is used to get data from `glm`, and must be preceded by `update_glm()` or `glm()`.

Usage

```
model_data(x)
```

Arguments

`x` Object of class `refitsmooth`, `refitrestricted` or `glm`

Value

`data.frame`

Author(s)

Martin Haringa

model_performance	<i>Performance of fitted GLMs</i>
-------------------	-----------------------------------

Description

Compute indices of model performance for (one or more) GLMs.

Usage

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

Arguments

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

Details

The following indices are computed:

AIC Akaike's Information Criterion

BIC Bayesian Information Criterion

RMSE Root mean squared error

Adopted from `performance::model_performance()`.

Value

data frame

Author(s)

Martin Haringa

Examples

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

MTPL

Characteristics of 30,000 policyholders in a Motor Third Party Liability (MTPL) portfolio.

Description

A dataset containing the age, number of claims, exposure, claim amount, power, bm, and region of 30,000 policyholders.

Usage

MTPL

Format

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

age_policyholder age of policyholder, in years.

nclaims number of claims.

exposure exposure, for example, if a vehicle is insured as of July 1 for a certain year, then during that year, this would represent an exposure of 0.5 to the insurance company.

amount claim amount in Euros.

power engine power of vehicle (in kilowatts).

bm level occupied in the 23-level (0-22) bonus-malus scale (the higher the level occupied, the worse the claim history).

zip region indicator (0-3).

Author(s)

Martin Haringa

Source

The data is derived from the portfolio of a large Dutch motor insurance company.

MTPL2

Characteristics of 3,000 policyholders in a Motor Third Party Liability (MTPL) portfolio.

Description

A dataset containing the area, number of claims, exposure, claim amount, exposure, and premium of 3,000 policyholders

Usage

MTPL2

Format

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

customer_id customer id

area region where customer lives (0-3)

nclaims number of claims

amount claim amount (severity)

exposure exposure

premium earned premium

Author(s)

Martin Haringa

Source

The data is derived from the portfolio of a large Dutch motor insurance company.

period_to_months	<i>Split period to months</i>
------------------	-------------------------------

Description

The function splits rows with a time period longer than one month to multiple rows with a time period of exactly one month each. Values in numeric columns (e.g. exposure or premium) are divided over the months proportionately.

Usage

```
period_to_months(df, begin, end, ...)
```

Arguments

df	data.frame
begin	column in df with begin dates
end	column in df with end dates
...	numeric columns in df to split

Details

In insurance portfolios it is common that rows relate to periods longer than one month. This is for example problematic in case exposures per month are desired.

Since insurance premiums are constant over the months, and do not depend on the number of days per month, the function assumes that each month has the same number of days (i.e. 30).

Value

data.frame with same columns as in df, and one extra column called `id`

Author(s)

Martin Haringa

Examples

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

rating_factors	<i>Include reference group in regression output</i>
----------------	---

Description

Extract coefficients in terms of the original levels of the coefficients rather than the coded variables.

Usage

```
rating_factors(  
  ...,  
  model_data = NULL,  
  exposure = NULL,  
  exponentiate = TRUE,  
  signif_stars = FALSE,  
  round_exposure = 0  
)
```

Arguments

...	glm object(s) produced by glm()
model_data	data.frame used to create glm object(s), this should only be specified in case the exposure is desired in the output, default value is NULL
exposure	column in model_data with exposure, default value is NULL
exponentiate	logical indicating whether or not to exponentiate the coefficient estimates. Defaults to TRUE.
signif_stars	show significance stars for p-values (defaults to TRUE)
round_exposure	number of digits for exposure (defaults to 0)

Details

A fitted linear model has coefficients for the contrasts of the factor terms, usually one less in number than the number of levels. This function re-expresses the coefficients in the original coding. This function is adopted from `dummy.coef()`. Our adoption prints a data.frame as output. Use `rating_factors_()` for standard evaluation.

Value

data.frame

Author(s)

Martin Haringa

Examples

```
df <- MTPL2 |>
dplyr::mutate(dplyr::across(c(area), as.factor)) |>
dplyr::mutate(dplyr::across(c(area), ~biggest_reference(., exposure)))

mod1 <- glm(nclaims ~ area + premium, offset = log(exposure),
family = poisson(), data = df)
mod2 <- glm(nclaims ~ area, offset = log(exposure), family = poisson(),
data = df)

rating_factors(mod1, mod2, model_data = df, exposure = exposure)
```

reduce

Reduce portfolio by merging redundant date ranges

Description

Transform all the date ranges together as a set to produce a new set of date ranges. Ranges separated by a gap of at least `min.gapwidth` days are not merged.

Usage

```
reduce(df, begin, end, ..., agg_cols = NULL, agg = "sum", min.gapwidth = 5)
```

Arguments

<code>df</code>	data.frame
<code>begin</code>	name of column df with begin dates
<code>end</code>	name of column in df with end dates
<code>...</code>	names of columns in df used to group date ranges by
<code>agg_cols</code>	list with columns in df to aggregate by (defaults to NULL)
<code>agg</code>	aggregation type (defaults to "sum")
<code>min.gapwidth</code>	ranges separated by a gap of at least <code>min.gapwidth</code> days are not merged. Defaults to 5.

Details

This function is adopted from `IRanges::reduce()`.

Value

An object of class "reduce". The function summary is used to obtain and print a summary of the results. An object of class "reduce" is a list usually containing at least the following elements:

df	data frame with reduced time periods
begin	name of column in df with begin dates
end	name of column in df with end dates
cols	names of columns in df used to group date ranges by

Author(s)

Martin Haringa

Examples

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

# Merge periods
pt1 <- reduce(portfolio, begin = begin_dat, end = end_dat, policy_nr,
productgroup, product, min.gapwidth = 5)

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

# Merge periods and sum premium per period
pt2 <- reduce(portfolio, begin = begin_dat, end = end_dat, policy_nr,
productgroup, product, agg_cols = list(premium), min.gapwidth = 5)

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

refit_glm	<i>Refitting Generalized Linear Models</i>
-----------	--

Description

[Experimental] refit_glm() is used to refit generalized linear models, and must be preceded by restrict_coef().

Usage

```
refit_glm(x)
```

Arguments

x	Object of class restricted or of class smooth
---	---

Value

Object of class GLM

Author(s)

Martin Haringa

restrict_coef	<i>Restrict coefficients in the model</i>
---------------	---

Description

[Experimental] Add restrictions, like a bonus-malus structure, on the risk factors used in the model. restrict_coef() must always be followed by update_glm().

Usage

```
restrict_coef(model, restrictions)
```

Arguments

model	object of class glm/restricted
restrictions	data.frame with two columns containing restricted data. The first column, with the name of the risk factor as column name, must contain the levels of the risk factor. The second column must contain the restricted coefficients.

Details

Although restrictions could be applied either to the frequency or the severity model, it is more appropriate to impose the restrictions on the premium model. This can be achieved by calculating the pure premium for each record (i.e. expected number of claims times the expected claim amount), then fitting an "unrestricted" Gamma GLM to the pure premium, and then imposing the restrictions in a final "restricted" Gamma GLM.

Value

Object of class `restricted`.

Author(s)

Martin Haringa

See Also

[update_glm\(\)](#) for refitting the restricted model, and [autoplot.restricted\(\)](#).

Other `update_glm`: [smooth_coef\(\)](#)

Examples

```
## Not run:
# Add restrictions to risk factors for region (zip) -----

# Fit frequency and severity model
library(dplyr)
freq <- glm(nclaims ~ bm + zip, offset = log(exposure), family = poisson(),
            data = MTPL)
sev <- glm(amount ~ bm + zip, weights = nclaims,
            family = Gamma(link = "log"),
            data = MTPL |> filter(amount > 0))

# Add predictions for freq and sev to data, and calculate premium
premium_df <- MTPL |>
  add_prediction(freq, sev) |>
  mutate(premium = pred_nclaims_freq * pred_amount_sev)

# Restrictions on risk factors for region (zip)
zip_df <- data.frame(zip = c(0,1,2,3), zip_rst = c(0.8, 0.9, 1, 1.2))

# Fit unrestricted model
burn <- glm(premium ~ bm + zip, weights = exposure,
            family = Gamma(link = "log"), data = premium_df)

# Fit restricted model
burn_rst <- burn |>
  restrict_coef(restrictions = zip_df) |>
  update_glm()

# Show rating factors
```

```
rating_factors(burn_rst)

## End(Not run)
```

rgammat	<i>Generate data from truncated gamma distribution</i>
---------	--

Description

Random generation for the truncated Gamma distribution with parameters shape and scale.

Usage

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

Arguments

n	number of observations
scale	scale parameter
shape	shape parameter
lower	numeric. Observations below this threshold are not present in the sample.
upper	numeric. Observations above this threshold are not present in the sample.

Value

The length of the result is determined by n.

Author(s)

Martin Haringa

rlnormt	<i>Generate data from truncated lognormal distribution</i>
---------	--

Description

Random generation for the truncated log normal distribution whose logarithm has mean equal to meanlog and standard deviation equal to sdlog.

Usage

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

Arguments

n	number of observations
meanlog	mean of the distribution on the log scale
sdlog	standard deviation of the distribution on the log scale
lower	numeric. Observations below this threshold are not present in the sample.
upper	numeric. Observations above this threshold are not present in the sample.

Value

The length of the result is determined by n.

Author(s)

Martin Haringa

rmse	<i>Root Mean Squared Error</i>
------	--------------------------------

Description

Compute root mean squared error.

Usage

```
rmse(object, data)
```

Arguments

object	fitted model
data	data.frame (defaults to NULL)

Details

The RMSE is the square root of the average of squared differences between prediction and actual observation and indicates the absolute fit of the model to the data. It can be interpreted as the standard deviation of the unexplained variance, and is in the same units as the response variable. Lower values indicate better model fit.

Value

numeric value

Author(s)

Martin Haringa

Examples

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

rows_per_date

Find active rows per date

Description

Fast overlap joins. Usually, `df` is a very large `data.table` (e.g. insurance portfolio) with small interval ranges, and `dates` is much smaller with (e.g.) claim dates.

Usage

```
rows_per_date(
  df,
  dates,
  df_begin,
  df_end,
  dates_date,
  ...,
  nomatch = NULL,
  mult = "all"
)
```

Arguments

<code>df</code>	data.frame with portfolio (df should include time period)
<code>dates</code>	data.frame with dates to join
<code>df_begin</code>	column name with begin dates of time period in <code>df</code>
<code>df_end</code>	column name with end dates of time period in <code>df</code>
<code>dates_date</code>	column name with dates in <code>dates</code>
<code>...</code>	additional column names in <code>dates</code> to join by
<code>nomatch</code>	When a row (with interval say, <code>[a,b]</code>) in <code>x</code> has no match in <code>y</code> , <code>nomatch=NA</code> means <code>NA</code> is returned for <code>y</code> 's non-by. <code>y</code> columns for that row of <code>x</code> . <code>nomatch=NULL</code> (default) means no rows will be returned for that row of <code>x</code> .
<code>mult</code>	When multiple rows in <code>y</code> match to the row in <code>x</code> , <code>mult</code> controls which values are returned - "all" (default), "first" or "last".

Value

returned class is equal to class of `df`

Author(s)

Martin Haringa

Examples

```

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

## Find active rows on different dates
dates0 <- data.frame(active_date = seq(ymd("2014-01-01"), ymd("2014-05-01"),
  by = "months"))
rows_per_date(portfolio, dates0, df_begin = begin1, df_end = end,
  dates_date = active_date)

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

### Only rows are returned that can be matched
rows_per_date(portfolio, claim_dates, df_begin = begin1,
  df_end = end, dates_date = claim_date, car_type)

### When row cannot be matched, NA is returned for that row
rows_per_date(portfolio, claim_dates, df_begin = begin1,
  df_end = end, dates_date = claim_date, car_type, nomatch = NA)

```

smooth_coef

Smooth coefficients in the model

Description

[Experimental] Apply smoothing on the risk factors used in the model. `smooth_coef()` must always be followed by `update_glm()`.

Usage

```

smooth_coef(
  model,
  x_cut,
  x_org,
  degree = NULL,
  breaks = NULL,

```

```

    smoothing = "spline",
    k = NULL,
    weights = NULL
)

```

Arguments

model	object of class glm/smooth
x_cut	column name with breaks/cut
x_org	column name where x_cut is based on
degree	order of polynomial
breaks	numerical vector with new clusters for x
smoothing	choose smoothing specification (all the shape constrained smooth terms (SCOP-splines) are constructed using the B-splines basis proposed by Eilers and Marx (1996) with a discrete penalty on the basis coefficients: <ul style="list-style-type: none"> • 'spline' (default) • 'mpi': monotone increasing SCOP-splines • 'mpd': monotone decreasing SCOP-splines • 'cx': convex SCOP-splines • 'cv': concave SCOP-splines • 'micx': increasing and convex SCOP-splines • 'micv': increasing and concave SCOP-splines • 'mdcx': decreasing and convex SCOP-splines • 'mdcv': decreasing and concave SCOP-splines • 'gam': spline based smooth (thin plate regression spline)
k	number of basis functions be computed
weights	weights used for smoothing, must be equal to the exposure (defaults to NULL)

Details

Although smoothing could be applied either to the frequency or the severity model, it is more appropriate to impose the smoothing on the premium model. This can be achieved by calculating the pure premium for each record (i.e. expected number of claims times the expected claim amount), then fitting an "unrestricted" Gamma GLM to the pure premium, and then imposing the restrictions in a final "restricted" Gamma GLM.

Value

Object of class smooth

Author(s)

Martin Haringa

See Also

[update_glm\(\)](#) for refitting the smoothed model, and [autoplot.smooth\(\)](#).

Other update_glm: [restrict_coef\(\)](#)

Examples

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

# Fit GAM for claim frequency
age_policyholder_frequency <- fit_gam(data = MTPL,
                                     nclaims = nclaims,
                                     x = age_policyholder,
                                     exposure = exposure)

# Determine clusters
clusters_freq <- construct_tariff_classes(age_policyholder_frequency)

# Add clusters to MTPL portfolio
dat <- MTPL |>
  mutate(age_policyholder_freq_cat = clusters_freq$tariff_classes) |>
  mutate(across(where(is.character), as.factor)) |>
  mutate(across(where(is.factor), ~biggest_reference(., exposure)))

# Fit frequency and severity model
freq <- glm(nclaims ~ bm + age_policyholder_freq_cat, offset = log(exposure),
           family = poisson(), data = dat)
sev <- glm(amount ~ bm + zip, weights = nclaims,
           family = Gamma(link = "log"), data = dat |> filter(amount > 0))

# Add predictions for freq and sev to data, and calculate premium
premium_df <- dat |>
  add_prediction(freq, sev) |>
  mutate(premium = pred_nclaims_freq * pred_amount_sev)

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

# Impose smoothing and create figure
burn_unrestricted |>
  smooth_coef(x_cut = "age_policyholder_freq_cat",
             x_org = "age_policyholder",
             breaks = seq(18, 95, 5)) |>
  autoplot()

# Impose smoothing and refit model
burn_restricted <- burn_unrestricted |>
  smooth_coef(x_cut = "age_policyholder_freq_cat",
```

```

        x_org = "age_policyholder",
        breaks = seq(18, 95, 5)) |>
update_glm()

# Show new rating factors
rating_factors(burn_restricted)

## End(Not run)

```

summary.reduce	<i>Automatically create a summary for objects obtained from reduce()</i>
----------------	--

Description

Takes an object produced by `reduce()`, and counts new and lost customers.

Usage

```

## S3 method for class 'reduce'
summary(object, ..., period = "days", name = "count")

```

Arguments

object	reduce object produced by <code>reduce()</code>
...	names of columns to aggregate counts by
period	a character string indicating the period to aggregate on. Four options are available: "quarters", "months", "weeks", and "days" (the default option)
name	The name of the new column in the output. If omitted, it will default to count.

Value

data.frame

univariate	<i>Univariate analysis for discrete risk factors</i>
------------	--

Description

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

- frequency (i.e. number of claims / exposure)
- average severity (i.e. severity / number of claims)
- risk premium (i.e. severity / exposure)
- loss ratio (i.e. severity / premium)
- average premium (i.e. premium / exposure)

If input arguments are not specified, the summary statistics related to these arguments are ignored.

Usage

```
univariate(
  df,
  x,
  severity = NULL,
  nclaims = NULL,
  exposure = NULL,
  premium = NULL,
  by = NULL
)
```

Arguments

df	data.frame with insurance portfolio
x	column in df with risk factor, or use <code>vec_ext()</code> for use with an external vector (see examples)
severity	column in df with severity (default is NULL)
nclaims	column in df with number of claims (default is NULL)
exposure	column in df with exposure (default is NULL)
premium	column in df with premium (default is NULL)
by	list of column(s) in df to group by

Value

A data.frame

Author(s)

Martin Haringa

Examples

```
# Summarize by `area`
univariate(MTPL2, x = area, severity = amount, nclaims = nclaims,
           exposure = exposure, premium = premium)

# Summarize by `area`, with column name in external vector
xt <- "area"
univariate(MTPL2, x = vec_ext(xt), severity = amount, nclaims = nclaims,
           exposure = exposure, premium = premium)

# Summarize by `zip` and `bm`
univariate(MTPL, x = zip, severity = amount, nclaims = nclaims,
           exposure = exposure, by = bm)

# Summarize by `zip`, `bm` and `power`
univariate(MTPL, x = zip, severity = amount, nclaims = nclaims,
           exposure = exposure, by = list(bm, power))
```

`update_glm`*Refitting Generalized Linear Models*

Description

[Experimental] `update_glm()` is used to refit generalized linear models, and must be preceded by `restrict_coef()`.

Usage

```
update_glm(x, intercept_only = FALSE)
```

Arguments

`x` Object of class `restricted` or of class `smooth`

`intercept_only` Logical. Default is `FALSE`. If `TRUE`, only the intercept is updated, ensuring that the changes have no impact on the other variables.

Value

Object of class `GLM`

Author(s)

Martin Haringa

Index

- * **autoplot.restricted**
 - restrict_coef, [34](#)
- * **autoplot.smooth**
 - smooth_coef, [39](#)
- * **datasets**
 - MTPL, [28](#)
 - MTPL2, [29](#)
- * **update_glm**
 - restrict_coef, [34](#)
 - smooth_coef, [39](#)

[add_prediction](#), [3](#)
[autoplot.bootstrap_rmse](#), [3](#)
[autoplot.check_residuals](#), [4](#)
[autoplot.constructtariffclasses](#), [5](#)
[autoplot.fitgam](#), [6](#)
[autoplot.restricted](#), [8](#)
[autoplot.restricted\(\)](#), [35](#)
[autoplot.riskfactor](#), [8](#)
[autoplot.smooth](#), [10](#)
[autoplot.smooth\(\)](#), [41](#)
[autoplot.truncated_dist](#), [10](#)
[autoplot.univariate](#), [11](#)

[biggest_reference](#), [13](#)
[bootstrap_rmse](#), [14](#)

[check_overdispersion](#), [16](#)
[check_residuals](#), [17](#)
[construct_model_points](#), [18](#)
[construct_tariff_classes](#), [19](#)

[DHARMA::simulateResiduals\(\)](#), [17](#)

[fisher](#), [21](#)
[fit_gam](#), [22](#)
[fit_truncated_dist](#), [24](#)

[histbin](#), [26](#)

[model_data](#), [27](#)
[model_performance](#), [27](#)
MTPL, [28](#)
MTPL2, [29](#)
[period_to_months](#), [30](#)
[rating_factors](#), [31](#)
[reduce](#), [32](#)
[refit_glm](#), [34](#)
[restrict_coef](#), [34](#), [41](#)
[rgammat](#), [36](#)
[rlnormt](#), [36](#)
[rmse](#), [37](#)
[rows_per_date](#), [38](#)

[smooth_coef](#), [35](#), [39](#)
[summary.reduce](#), [42](#)

[univariate](#), [42](#)
[update_glm](#), [44](#)
[update_glm\(\)](#), [35](#), [41](#)