Package ‘Greg’

July 21, 2025

Version 2.0.2

Title Regression Helper Functions

Maintainer Max Gordon <`max@gforge.se`>

Description Methods for manipulating regression models and for describing these in a style adapted for medical journals.
Contains functions for generating an HTML table with crude and adjusted estimates, plotting hazard ratio, plotting model
estimates and confidence intervals using forest plots, extending this to comparing multiple models in a single forest plots.
In addition to the descriptive methods, there are functions for the robust covariance matrix provided by the 'sandwich'
package, a function for adding non-
linearities to a model, and a wrapper around the 'Epi' package's Lexis() functions for
time-splitting a dataset when modeling non-proportional hazards in Cox regressions.

License GPL (>= 3)

URL `http://gforge.se`

BugReports `https://github.com/gforge/Greg/issues`

Biarch yes

Encoding UTF-8

Imports broom, Epi, dplyr, glue, graphics, grDevices, htmlTable (>= 2.0.0), Hmisc, knitr, methods, nlme, purrr, rlang, rms, sandwich, stats, stringr, tibble, tidyr, tidyselect, utils

Depends Gmisc (>= 1.0.3), forestplot, R (>= 4.1.0)

Suggests boot, testthat, cmprsk, survival, ggplot2, parallel, rmarkdown, rmeta, tidyverse

VignetteBuilder knitr

RoxygenNote 7.2.2

NeedsCompilation no

Author Max Gordon [aut, cre],
Reinhard Seifert [aut] (Author of original plotHR)

Repository CRAN

Date/Publication 2024-01-29 13:30:21 UTC

Contents

Greg-package 2
addNonlinearity 3
caDescribeOpts 5
confint.ols 6
confint_robust 7
forestplotCombineRegrObj 8
forestplotRegrObj 10
isFitCoxPH 14
plotHR 17
robcov_alt 21
tidy.rms 22
timeSplitter 24

Index **26**

`Greg-package` *Regression Helper Functions*

Description

This R-package provides functions that primarily aimed at helping you work with regression models. While much of the data presented by the standard regression output is useful and important - there is often a need for further simplification prior to publication. The methods implemented in this package are inspired by some of the top journals such as NEJM, BMJ, and other medical journals as this is my research field.

Output functions

The package has function that automatically prints the crude unadjusted estimates of a function next to the adjusted estimates, a common practice for medical publications.

The forestplot wrappers allows for easily displaying regression estimates, often convenient for models with a large number of variables. There is also functionality that can help you comparing different models, e.g. subsets of patients or compare different regression types.

Time-splitter

When working with Cox regressions the proportional hazards can sometimes be violated. As the `tt()` approach doesn't lend itself that well to big datasets I often rely on time-splitting the dataset and then using the start time as an interaction term. See the function `timeSplitter()` and the associated `vignette("timeSplitter")`.

Other regression functions

In addition to these funciton the package has some extentions to linear regression where it extends the functionality by allowing for robust covariance matrices. by integrating the **'sandwich'**-package for `rms::ols()`.

Important notice

This package has an extensive test-set for ensuring that everything behaves as expected. Despite this I strongly urge you to check that the values make sense. I commonly use the regression methods available in the **'rms'**-package and in the **'stats'**-package. In addition I use the `coxph()` in many of my analyses and should also be safe. Please send me a notice if you are using the package with some other regression models, especially if you have some tests verifying the functionality.

Author(s)

Max Gordon

`addNonlinearity` *Add a nonlinear function to the model*

Description

This function takes a model and adds a non-linear function if the likelihood-ratio supports this (via the `anova(..., test = "chisq")` test for **stats** while for **rms** you need to use the `rcs()` spline that is automatically evaluated for non-linearity).

Usage

```
addNonlinearity(
  model,
  variable,
  spline_fn,
  flex_param = 2:7,
  min_fn = AIC,
  sig_level = 0.05,
  verbal = FALSE,
  workers,
  ...
)

## S3 method for class 'negbin'
addNonlinearity(model, ...)
```

Arguments

`model`	The model that is to be evaluated and adapted for non-linearity
`variable`	The name of the parameter that is to be tested for non-linearity. *Note* that the variable should be included plain (i.e. as a linear variable) form in the model.
`spline_fn`	Either a string or a function that is to be used for testing alternative non-linearity models
`flex_param`	A `vector` with values that are to be tested as the default second parameter for the non-linearity function that you want to evaluate. This defaults to 2:7, for the `ns()` it tests the degrees of freedom ranging between 2 and 7.

`min_fn`	This is the function that we want to minimized if the variable supports the non-linearity assumption. E.g. `BIC()` or `AIC`, note that the `BIC()` will in the majority of cases support a lower complexity than the `AIC()`.
`sig_level`	The significance level for which the non-linearity is deemed as significant, defaults to 0.05.
`verbal`	Set this to `TRUE` if you want print statements with the anova test and the chosen knots.
`workers`	The function tries to run everything in parallel. Under some circumstances you may want to restrict the number of parallel threads to less than the default `detectCores() - 1`, e.g. you may run out of memory then you can provide this parameter. If you do not want to use parallel then simply set workers to `FALSE`. The cluster created using `makeCluster()` function.
`...`	Passed onto internal `prNlChooseDf()` function.

Examples

```
library(Greg)
data("melanoma", package = "boot", envir = environment())

library(dplyr)
melanoma <- mutate(melanoma,
                   status = factor(status,
                                   levels = 1:3,
                                   labels = c("Died from melanoma",
                                              "Alive",
                                              "Died from other causes")),
                   ulcer = factor(ulcer,
                                  levels = 0:1,
                                  labels = c("Absent", "Present")),
                   time = time/365.25, # All variables should be in the same time unit
                   sex = factor(sex,
                                levels = 0:1,
                                labels = c("Female", "Male")))

library(survival)
model <- coxph(Surv(time, status == "Died from melanoma") ~ sex + age,
  data = melanoma
)

nl_model <- addNonlinearity(model, "age",
  spline_fn = "pspline",
  verbal = TRUE,
  workers = FALSE
)
# Note that there is no support for nonlinearity in this case
```

`caDescribeOpts` *A function for gathering all the description options*

Description

Since there are so many different description options for the `printCrudeAndAdjustedModel()` function they have been gathered into a list. This function is simply a helper in order to generate a valid list.

Usage

```
caDescribeOpts(
  show_tot_perc = FALSE,
  numb_first = TRUE,
  continuous_fn = describeMean,
  prop_fn = describeFactors,
  factor_fn = describeFactors,
  digits = 1,
  colnames = c("Total", "Event")
)
```

Arguments

`show_tot_perc`	Show percentages for the total column
`numb_first`	Whether to show the number before the percentages
`continuous_fn`	Stat function used for the descriptive statistics, defaults to `describeMean()`
`prop_fn`	Stat function used for the descriptive statistics, defaults to `describeFactors()` since there has to be a reference in the current setup.
`factor_fn`	Stat function used for the descriptive statistics, defaults to `describeFactors()`
`digits`	Number of digits to use in the descriptive columns. Defaults to the general digits if not specified.
`colnames`	The names of the two descriptive columns. By default Total and Event.

Value

`list` Returns a list with all the options

`confint.ols` *A* `confint` *function for the* `ols`

Description

This function checks that there is a `df.residual` before running the `qt()`. If not found it then defaults to the `qnorm()` function. Otherwise it is a copy of the `confint()` function.

Usage

```
## S3 method for class 'ols'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	a fitted `ols`-model object.
`parm`	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
`level`	the confidence level required.
`...`	additional argument(s) for methods.

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 in

Examples

```
# Generate some data
n <- 500
x1 <- runif(n) * 2
x2 <- runif(n)
y <- x1^3 + x2 + rnorm(n)

library(rms)
library(sandwich)
dd <- datadist(x1, x2, y)
org.op <- options(datadist = "dd")

# Main function
f <- ols(y ~ rcs(x1, 3) + x2)

# Check the bread
bread(f)
# Check the HC-matrix
vcovHC(f, type = "HC4m")
# Adjust the model so that it uses the HC4m variance
```

```
f_rob <- robcov_alt(f, type = "HC4m")
# Get the new HC4m-matrix
# - this function just returns the f_rob$var matrix
vcov(f_rob)
# Now check the confidence interval for the function
confint(f_rob)

options(org.op)
```

`confint_robust` *The confint function adapted for vcovHC*

Description

The confint.lm uses the t-distribution as the default confidence interval estimator. When there is reason to believe that the normal distribution is violated an alternative approach using the `vcovHC()` may be more suitable.

Usage

```
confint_robust(
  object,
  parm,
  level = 0.95,
  HC_type = "HC3",
  t_distribution = FALSE,
  ...
)
```

Arguments

`object`	The regression model object, either an ols or lm object
`parm`	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
`level`	the confidence level required.
`HC_type`	See options for `vcovHC()`
`t_distribution`	A boolean for if the t-distribution should be used or not. Defaults to FALSE. According to Cribari-Nieto and Lima's study from 2009 this should not be the case.
`...`	Additional parameters that are passed on to `vcovHC()`

Value

`matrix` A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 in

References

F. Cribari-Neto and M. da G. A. Lima, "Heteroskedasticity-consistent interval estimators", Journal of Statistical Computation and Simulation, vol. 79, no. 6, pp. 787-803, 2009 (doi:10.1080/00949650801935327)

Examples

```
n <- 50
x <- runif(n)
y <- x + rnorm(n)

fit <- lm(y~x)
library("sandwich")
confint_robust(fit, HC_type = "HC4m")
```

`forestplotCombineRegrObj` *Compares different scores in different regression objects.*

Description

Creates a composite from different regression objects into one forestplot where you can choose the variables of interest to get an overview and easier comparison.

Usage

```
forestplotCombineRegrObj(
  regr.obj,
  variablesOfInterest.regexp = NULL,
  estimate.txt = NULL,
  add_first_as_ref = FALSE,
  ref_txt = "ref.",
  digits = 1,
  post_process_data = function(x) x,
  is.summary = NULL,
  xlab = NULL,
  zero = NULL,
  xlog = NULL,
  exp = xlog,
  ...
)
```

Arguments

`regr.obj`	A list with all the fits that have variables that are to be identified through the regular expression

`variablesOfInterest.regexp`	A regular expression identifying the variables that are of interest of comparing. For instance it can be "(score|index|measure)" that finds scores in different models that should be compared.
`estimate.txt`	The text of the estimate, usually HR for hazard ratio, OR for odds ratio
`add_first_as_ref`	If you want that the first variable should be reference for that group of variables. The ref is a variable with the estimate 1 or 0 depending if exp() and the confidence interval 0.
`ref_txt`	Text instead of estimate number
`digits`	Number of digits to use for the estimate output
`post_process_data`	A function that takes the data frame just prior to calling ‘forestplot‘ and allows you to manipulate it. Primarily used for changing the ‘column_label‘ that has the names shown in the final plot.
`is.summary`	A vector indicating by `TRUE`/`FALSE` if the value is a summary value which means that it will have a different font-style
`xlab`	x-axis label
`zero`	Indicates what is zero effect. For survival/logistic fits the zero is 1 while in most other cases it's 0.
`xlog`	If TRUE, x-axis tick marks are to follow a logarithmic scale, e.g. for logistic regression (OR), survival estimates (HR), Poisson regression etc. *Note:* This is an intentional break with the original `forestplot` function as I've found that exponentiated ticks/clips/zero effect are more difficult to for non-statisticians and there are sometimes issues with rounding the tick marks properly.
`exp`	Report in exponential form. Default true since the function was built for use with survival models.
`...`	Passed to `forestplot()`

See Also

Other forestplot wrappers: `forestplotRegrObj()`

Examples

```
org.par <- par("ask" = TRUE)

# simulated data to test
library(tidyverse)
set.seed(10)
cov <- tibble(ftime = rexp(200),
              fstatus = sample(0:1, 200, replace = TRUE),
              x1 = runif(200),
              x2 = runif(200),
              x3 = runif(200)) |>
  # Add some column labels
  Gmisc::set_column_labels(x1 = "First variable",
```

```
                              x2 = "Second variable")

library(rms)
ddist <- datadist(cov)
options(datadist = "ddist")

fit1 <- cph(Surv(ftime, fstatus) ~ x1 + x2, data = cov)
fit2 <- cph(Surv(ftime, fstatus) ~ x1 + x3, data = cov)

list(`First model` = fit1,
     `Second model` = fit2) |>
  forestplotCombineRegrObj(variablesOfInterest.regexp = "(x2|x3)") |>
  fp_set_style(lines = "steelblue",
               box = "darkblue")

# How to add expressions to the plot label
list(fit1, fit2) |>
  forestplotCombineRegrObj(variablesOfInterest.regexp = "(x2|x3)",
                           reference.names = c("First model", "Second model"),
                           post_process_data = \(data) {
                        data$column_label[4] <- c(rlang::expr(expression(Fever >= 38.5)))
                             return(data)
                           })

par(org.par)
```

`forestplotRegrObj` *Forest plot for multiple models*

Description

Plot different model fits with similar variables in order to compare the model's estimates and confidence intervals. Each model is represented by a separate line on top of eachother and are therefore ideal for comparing different models. This extra appealing when you have lots of variables included in the models.

Usage

```
forestplotRegrObj(
  regr.obj,
  postprocess_estimates.fn = function(x) x,
  rowname = "Variable",
  ci.txt = "CI",
  ci.glue = "{lower} to {higher}",
  digits = 1,
  get_box_size = fpBoxSize,
  ...
)
```

```
## Default S3 method:
forestplotRegrObj(
  regr.obj,
  postprocess_estimates.fn = function(x) x,
  rowname = "Variable",
  ci.txt = "CI",
  ci.glue = "{lower} to {higher}",
  digits = 1,
  get_box_size = fpBoxSize,
  ...
)

## S3 method for class 'coxph'
forestplotRegrObj(
  regr.obj,
  postprocess_estimates.fn = function(x) x,
  rowname = "Variable",
  ci.txt = "CI",
  ci.glue = "{lower} to {higher}",
  digits = 1,
  get_box_size = fpBoxSize,
  xlab = "Hazard Ratio",
  estimate.txt = "HR",
  xlog = TRUE,
  zero = 1,
  exp = TRUE,
  ...
)

## S3 method for class 'lrm'
forestplotRegrObj(
  regr.obj,
  postprocess_estimates.fn = function(x) x,
  rowname = "Variable",
  ci.txt = "CI",
  ci.glue = "{lower} to {higher}",
  digits = 1,
  get_box_size = fpBoxSize,
  xlab = "Odds ratio",
  estimate.txt = "HR",
  xlog = TRUE,
  zero = 1,
  exp = TRUE,
  ...
)

## S3 method for class 'lm'
forestplotRegrObj(
```

```
  regr.obj,
  postprocess_estimates.fn = function(x) x,
  rowname = "Variable",
  ci.txt = "CI",
  ci.glue = "{lower} to {higher}",
  digits = 1,
  get_box_size = fpBoxSize,
  xlab = "Effect",
  estimate.txt = "Coef",
  xlog = FALSE,
  zero = 0,
  exp = FALSE,
  ...
)

## S3 method for class 'glm'
forestplotRegrObj(
  regr.obj,
  postprocess_estimates.fn = function(x) x,
  rowname = "Variable",
  ci.txt = "CI",
  ci.glue = "{lower} to {higher}",
  digits = 1,
  get_box_size = fpBoxSize,
  xlab = NULL,
  xlog = NULL,
  zero = NULL,
  estimate.txt = NULL,
  exp = NULL,
  ...
)

## S3 method for class 'list'
forestplotRegrObj(
  regr.obj,
  postprocess_estimates.fn = function(x) x,
  rowname = "Variable",
  ci.txt = "CI",
  ci.glue = "{lower} to {higher}",
  digits = 1,
  get_box_size = fpBoxSize,
  xlab = NULL,
  xlog = NULL,
  zero = NULL,
  estimate.txt = NULL,
  exp = NULL,
  ...
)
```

```
fpBoxSize(p_values, variable_count, boxsize, significant = 0.05)
```

Arguments

`regr.obj`	A regression model object. It should be of coxph, crr or glm class. Warning: The glm is not fully tested.
`postprocess_estimates.fn`	A function that takes the regression outputs and returns the same data with modifications. The input columns are: * ‘Rowname‘ * ‘Coef‘ * ‘Lower‘ * ‘Upper‘ * ‘Sort‘
`rowname`	The name of the variables
`ci.txt`	The text above the confidence interval, defaults to ‘"CI"‘
`ci.glue`	The string used for [glue::glue()] the ‘lower‘ and ‘higher‘ confidence intervals together.
`digits`	The number of digits to round presented values to
`get_box_size`	A function for extracting the box sizes
`...`	Passed to `forestplot()`
`xlab`	x-axis label
`estimate.txt`	The text above the estimate, e.g. Est, HR
`xlog`	If TRUE, x-axis tick marks are to follow a logarithmic scale, e.g. for logistic regression (OR), survival estimates (HR), Poisson regression etc. *Note:* This is an intentional break with the original `forestplot` function as I've found that exponentiated ticks/clips/zero effect are more difficult to for non-statisticians and there are sometimes issues with rounding the tick marks properly.
`zero`	Indicates what is zero effect. For survival/logistic fits the zero is 1 while in most other cases it's 0.
`exp`	Report in exponential form. Default true since the function was built for use with survival models.
`p_values`	The p-values that will work as the foundation for the box size
`variable_count`	The number of variables
`boxsize`	The default box size
`significant`	Level of significance .05

See Also

Other forestplot wrappers: `forestplotCombineRegrObj()`

Examples

```
org.par <- par("ask" = TRUE)

library(tidyverse)
# simulated data to test
set.seed(102)
```

```
cov <- tibble(ftime = rexp(200)) |>
  mutate(x1 = runif(n()),
         x2 = runif(n()),
         x3 = runif(n()),
         fstatus1 = if_else(x1 * 1 +
                              x2 * 0.2 +
                              x3 * 0.5 +
                              runif(n()) * 0.5 > 1,
                            1, 0),
         fstatus2 = if_else(x1 * 0.2 +
                              x2 * 0.5 +
                              x3 * 0.1 +
                              runif(n()) * 2 > 1,
                            1, 0)) |>
  # Add some column labels
  Gmisc::set_column_labels(x1 = "First variable",
                           x2 = "Second variable")

library(rms)
dd <- datadist(cov)
options(datadist = "dd")

fit1 <- cph(Surv(ftime, fstatus1 == 1) ~ x1 + x2 + x3, data = cov)

fit1 |>
  forestplotRegrObj() |>
  fp_set_zebra_style("#f0f0f0")

fit2 <- update(fit1, Surv(ftime, fstatus2 == 1) ~ .)
list("Frist model" = fit1, "Second model"  = fit2) |>
  forestplotRegrObj(legend_args = fpLegend(title = "Type of regression"),
                    postprocess_estimates.fn = function(x) {
                      x |>
                        filter(str_detect(column_term, "(x2|x3)"))
                    }) |>
  fp_set_style(box = rep(c("darkblue", "darkred"), each = 3))


par(org.par)
```

`isFitCoxPH` *Functions for checking regression type*

Description

The *isFitCoxPH* A simple check if object inherits either "coxph" or "crr" class indicating that it is a survival function.

Usage

```
isFitCoxPH(fit)

isFitLogit(fit)
```

Arguments

`fit`	Regression object

Value

`boolean` Returns `TRUE` if the object is of that type otherwise it returns `FALSE`.

Examples

```
# simulated data to use
set.seed(10)
ds <- data.frame(
  ftime = rexp(200),
  fstatus = sample(0:1, 200, replace = TRUE),
  x1 = runif(200),
  x2 = runif(200),
  x3 = runif(200)
)

library(survival)
library(rms)

dd <- datadist(ds)
options(datadist = "dd")

s <- Surv(ds$ftime, ds$fstatus == 1)
fit <- cph(s ~ x1 + x2 + x3, data = ds)

if (isFitCoxPH(fit)) {
  print("Correct, the cph is of cox PH hazard type")
}

fit <- coxph(s ~ x1 + x2 + x3, data = ds)
if (isFitCoxPH(fit)) {
  print("Correct, the coxph is of cox PH hazard type")
}

library(cmprsk)
set.seed(10)
ftime <- rexp(200)
fstatus <- sample(0:2, 200, replace = TRUE)
cov <- matrix(runif(600), nrow = 200)
dimnames(cov)[[2]] <- c("x1", "x2", "x3")
fit <- crr(ftime, fstatus, cov)

if (isFitCoxPH(fit)) {
```

```
  print(paste(
    "Correct, the competing risk regression is",
    "considered a type of cox regression",
    "since it has a Hazard Ratio"
  ))
}
# ** Borrowed code from the lrm example **

# Fit a logistic model containing predictors age, blood.pressure, sex
# and cholesterol, with age fitted with a smooth 5-knot restricted cubic
# spline function and a different shape of the age relationship for males
# and females.

n <- 1000 # define sample size
set.seed(17) # so can reproduce the results
age <- rnorm(n, 50, 10)
blood.pressure <- rnorm(n, 120, 15)
cholesterol <- rnorm(n, 200, 25)
sex <- factor(sample(c("female", "male"), n, TRUE))
label(age) <- "Age" # label is in Hmisc
label(cholesterol) <- "Total Cholesterol"
label(blood.pressure) <- "Systolic Blood Pressure"
label(sex) <- "Sex"
units(cholesterol) <- "mg/dl" # uses units.default in Hmisc
units(blood.pressure) <- "mmHg"

# To use prop. odds model, avoid using a huge number of intercepts by
# grouping cholesterol into 40-tiles

# Specify population model for log odds that Y = 1
L <- .4 * (sex == "male") + .045 * (age - 50) +
  (log(cholesterol - 10) - 5.2) * (-2 * (sex == "female") + 2 * (sex == "male"))
# Simulate binary y to have Prob(y = 1) = 1/[1+exp(-L)]
y <- ifelse(runif(n) < plogis(L), 1, 0)
cholesterol[1:3] <- NA # 3 missings, at random

ddist <- datadist(age, blood.pressure, cholesterol, sex)
options(datadist = "ddist")

fit_lrm <- lrm(y ~ blood.pressure + sex * (age + rcs(cholesterol, 4)),
  x = TRUE, y = TRUE
)

if (isFitLogit(fit_lrm) == TRUE) {
  print("Correct, the lrm is a logistic regression")
}

fit_lm <- lm(blood.pressure ~ sex)
if (isFitLogit(fit_lm) == FALSE) {
  print("Correct, the lm is not a logistic regression")
}

fit_glm_logit <- glm(y ~ blood.pressure + sex * (age + rcs(cholesterol, 4)),
```

```
  family = binomial()
)

if (isFitLogit(fit_glm_logit) == TRUE) {
  print("Correct, the glm with a family of binomial is a logistic regression")
}

fit_glm <- glm(blood.pressure ~ sex)
if (isFitLogit(fit_glm) == FALSE) {
  print("Correct, the glm without logit as a family is not a logistic regression")
}
```

`plotHR` *Plot a spline in a Cox regression model*

Description

This function is a more specialized version of the `termplot()` function. It creates a plot with the spline against hazard ratio. The plot can additianally have indicator of variable density and have multiple lines.

Usage

```
plotHR(
  models,
  term = 1,
  se = TRUE,
 cntrst = ifelse(inherits(models, "rms") || inherits(models[[1]], "rms"), TRUE, FALSE),
  polygon_ci = TRUE,
  rug = "density",
  xlab = "",
  ylab = "Hazard Ratio",
  main = NULL,
  xlim = NULL,
  ylim = NULL,
  col.term = "#08519C",
  col.se = "#DEEBF7",
  col.dens = grey(0.9),
  lwd.term = 3,
  lty.term = 1,
  lwd.se = lwd.term,
  lty.se = lty.term,
  x.ticks = NULL,
  y.ticks = NULL,
  ylog = TRUE,
  cex = 1,
  y_axis_side = 2,
  plot.bty = "n",
```

```
  axes = TRUE,
  alpha = 0.05,
  ...
)

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

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

Arguments

`models`	A single model or a list() with several models
`term`	The term of interest. Can be either the name or the number of the covariate in the model.
`se`	Boolean if you want the confidence intervals or not
`cntrst`	By contrasting values you can have the median as a reference point making it easier to compare hazard ratios.
`polygon_ci`	If you want a polygon as indicator for your confidence interval. This can also be in the form of a vector if you have several models. Sometimes you only want one model to have a polygon and the rest to be dotted lines. This gives the reader an indication of which model is important.
`rug`	The rug is the density of the population along the spline variable. Often this is displayed as a jitter with bars that are thicker & more common when there are more observations in that area or a smooth density plot that looks like a mountain. Use "density" for the mountain view and "ticks" for the jitter format.
`xlab`	The label of the x-axis
`ylab`	The label of the y-axis
`main`	The main title of the plot
`xlim`	A vector with 2 elements containing the upper & the lower bound of the x-axis
`ylim`	A vector with 2 elements containing the upper & the lower bound of the y-axis
`col.term`	The color of the estimate line. If multiple lines you can have different colors by giving a vector.
`col.se`	The color of the confidence interval. If multiple lines you can have different colors by giving a vector.
`col.dens`	The color of the density plot. Ignored if you're using jitter
`lwd.term`	The width of the estimated line. If you have more than one model then provide the function with a vector if you want to have different lines for different width for each model.
`lty.term`	The typeof the estimated line, see lty. If you have more than one model then provide the function with a vector if you want to have different line types for for each model.

`lwd.se`	The line width of your confidence interval. This is ignored if you're using polygons for all the confidence intervals.
`lty.se`	The line type of your confidence interval. This is ignored if you're using polygons for all the confidence intervals.
`x.ticks`	The ticks for the x-axis if you desire other than the default.
`y.ticks`	The ticks for the y-axis if you desire other than the default.
`ylog`	Show a logarithmic y-axis. Not having a logarithmic axis might seem easier to understand but it's actually not really a good idea. The distance between HR 0.5 and 2.0 should be the same. This will only show on a logarithmic scale and therefore it is strongly recommended to use the logarithmic scale.
`cex`	Increase if you want larger font size in the graph.
`y_axis_side`	The side that the y axis is to be plotted, see axis() for details
`plot.bty`	Type of box that you want. See the bty description in graphical parameters (par). If bty is one of "o" (the default), "l", "7", "c", "u", or "]" the resulting box resembles the corresponding upper case letter. A value of "n" suppresses the box.
`axes`	A boolean that is used to identify if axes are to be plotted
`alpha`	The alpha level for the confidence intervals
`...`	Any additional values that are to be sent to the plot() function
`x`	Sent the 'plotHR' object to plot
`y`	Ignored in plot

Value

The function does not return anything

Multiple models in one plot

The function allows for plotting multiple splines in one graph. Sometimes you might want to show more than one spline for the same variable. This allows you to create that comparison.

Examples of a situation where I've used multiple splines in one plot is when I want to look at a variables behavior in different time periods. This is another way of looking at the proportional hazards assumption. The Schoenfeld residuals can be a little tricky to look at when you have the splines.

Another example of when I've used this is when I've wanted to plot adjusted and unadjusted splines. This can very nicely demonstrate which of the variable span is mostly confounded. For instance - younger persons may exhibit a higher risk for a procedure but when you put in your covariates you find that the increased hazard changes back to the basic

Author(s)

Reinhard Seifert, Max Gordon

Examples

```
org_par <- par(xaxs = "i", ask = TRUE)
library(survival)
library(rms)
library(dplyr)
library(Gmisc)

# Get data for example
n <- 1000
set.seed(731)

ds <- tibble(age = round(50 + 12 * rnorm(n), 1),
             smoking = sample(c("Yes", "No"), n, rep = TRUE, prob = c(.2, .75)),
             sex = sample(c("Male", "Female"), n, rep = TRUE, prob = c(.6, .4))) |>
  # Build outcome
  mutate(h = .02 * exp(.02 * (age - 50) + .1 *
                         ((age - 50) / 10)^3 + .8 *
                         (sex == "Female") + 2 *
                         (smoking == "Yes")),
         cens = 15 * runif(n),
         dt = -log(runif(n)) / h,
         e = if_else(dt <= cens, 1, 0),
         dt = pmin(dt, cens),
         # Add missing data to smoking
         smoking = case_when(runif(n) < 0.05 ~ NA_character_,
                             TRUE ~ smoking)) |>
  set_column_labels(age = "Age",
                    dt = "Follow-up time") |>
  set_column_units(dt = "Year")


library(splines)
fit.coxph <- coxph(Surv(dt, e) ~ bs(age, 3) + sex + smoking, data = ds)

plotHR(fit.coxph, term = "age", plot.bty = "o", xlim = c(30, 70), xlab = "Age")

dd <- datadist(ds)
options(datadist = "dd")
fit.cph <- cph(Surv(dt, e) ~ rcs(age, 4) + sex + smoking, data = ds, x = TRUE, y = TRUE)

plotHR(fit.cph,
       term = 1,
       plot.bty = "L",
       xlim = c(30, 70),
       ylim = 2^c(-3, 3),
       xlab = "Age"
)

plotHR(fit.cph,
       term = "age",
       plot.bty = "l",
       xlim = c(30, 70),
```

```
        ylog = FALSE,
        rug = "ticks",
        xlab = "Age"
)

unadjusted_fit <- cph(Surv(dt, e) ~ rcs(age, 4), data = ds, x = TRUE, y = TRUE)
plotHR(list(fit.cph, unadjusted_fit),
        term = "age",
        xlab = "Age",
        polygon_ci = c(TRUE, FALSE),
        col.term = c("#08519C", "#77777799"),
        col.se = c("#DEEBF7BB", grey(0.6)),
        lty.term = c(1, 2),
        plot.bty = "l", xlim = c(30, 70)
)
par(org_par)
```

`robcov_alt` *Robust covariance matrix based upon the 'sandwich'-package*

Description

This is an alternative to the 'rms'-package robust covariance matrix that uses the **'sandwich'** package `vcovHC()` function instead of the **'rms'**-built-in estimator. The advantage being that many more estimation types are available.

Usage

```
robcov_alt(fit, type = "HC3", ...)
```

Arguments

`fit`	The ols fit that
`type`	a character string specifying the estimation type. See `vcovHC()` for options.
`...`	You should specify type= followed by some of the alternative available for the `vcovHC()` function.

Value

model The fitted model with adjusted variance and df.residual set to NULL

Examples

```
# Generate some data
n <- 500
x1 <- runif(n) * 2
x2 <- runif(n)
y <- x1^3 + x2 + rnorm(n)
```

```
library(rms)
library(sandwich)
dd <- datadist(x1, x2, y)
org.op <- options(datadist = "dd")

# Main function
f <- ols(y ~ rcs(x1, 3) + x2)

# Check the bread
bread(f)
# Check the HC-matrix
vcovHC(f, type = "HC4m")
# Adjust the model so that it uses the HC4m variance
f_rob <- robcov_alt(f, type = "HC4m")
# Get the new HC4m-matrix
# - this function just returns the f_rob$var matrix
vcov(f_rob)
# Now check the confidence interval for the function
confint(f_rob)

options(org.op)
```

`tidy.rms` *Tidy a(n) rms model object*

Description

Tidy summarizes information about the components of a model. A model component might be a single term in a regressions. Exactly what tidy considers to be a model component varies across models but is usually self-evident. If a model has several distinct types of components, you will need to specify which components to return.

Usage

```
## S3 method for class 'rms'
tidy(
  x,
  conf.int = FALSE,
  conf.level = 0.95,
  exponentiate = FALSE,
  ...,
 .add_print_p_and_stat_values = getOption("Greg.tidy_add_p_and_stat_values", default =
    FALSE)
)
```

Arguments

`x`	An rms model, e.g. [‘rms::cph()‘], [‘rms::lrm()‘]

`conf.int`	Logical indicating whether or not to include a confidence interval in the tidied output. Defaults to `FALSE`.
`conf.level`	The confidence level to use for the confidence interval if `conf.int = TRUE`. Must be strictly greater than 0 and less than 1. Defaults to 0.95, which corresponds to a 95 percent confidence interval.
`exponentiate`	Logical indicating whether or not to exponentiate the the coefficient estimates. This is typical for logistic and multinomial regressions, but a bad idea if there is no log or logit link. Defaults to `FALSE`.
`...`	Additional arguments. Not used. Needed to match generic signature only. **Cautionary note:** Misspelled arguments will be absorbed in `...`, where they will be ignored. If the misspelled argument has a default value, the default value will be used. For example, if you pass `conf.lvel = 0.9`, all computation will proceed using `conf.level = 0.95`. Two exceptions here are: • `tidy()` methods will warn when supplied an `exponentiate` argument if it will be ignored. • `augment()` methods will warn when supplied a `newdata` argument if it will be ignored.
`.add_print_p_and_stat_values`	For estimating print values there is a workaround that relies on capturing output from the ‘print(x)‘ and is not considered safe.

Details

This is a quick fix for addressing the lack of ‘rms‘-compatibility with the ‘broom‘ package, see [broom issue 30](https://github.com/tidymodels/broom/issues/30).

Value

A tibble::tibble() with columns: - ‘term‘ The name of the regression term. - ‘factor‘ The factor if the term is a character/factor term. - ‘column_term‘ The full name as in the original input data - ‘estimate‘ The estimated value of the regression term. - ‘conf.high‘ Upper bound on the confidence interval for the estimate.c - ‘conf.low‘ Lower bound on the confidence interval for the estimate. - ‘p.value‘ The two-sided p-value associated with the observed statistic. - ‘statistic‘ The value of a statistic to use in a hypothesis that the regression term is non-zero. - ‘std.error‘ The standard error of the regression term.

Examples

```
library(rms)
library(broom)
library(tidyverse)

set.seed(10)
cov <- tibble(x1 = runif(200)) |>
  mutate(x_bool_fact = if_else(x1 > 0.5,
                               "Yes",
                               sample(c("Yes", "No"), size = n(), replace = TRUE)),
       x_multi_fact = sample(c("Strange", "Factor", "Names"), size = n(), replace = TRUE),
```

```
          ftime = rexp(n()),
          fstatus = sample(0:1, size = n(), replace = TRUE),
          x_good_predictor = fstatus * runif(n()))

ddist <- datadist(cov)
options(datadist = "ddist")

cph_fit <- cph(Surv(ftime, fstatus) ~ x1 + x_bool_fact +
                 x_multi_fact + x_good_predictor, data = cov)
tidy(cph_fit)
```

`timeSplitter` *A function for splitting a time according to time periods*

Description

If we have a violation of the cox proprtional hazards assumption we need to split an individual's followup time into several. See `vignette("timeSplitter", package = "Greg")` for a detailed description.

Usage

```
timeSplitter(
  data,
  by,
  time_var,
  event_var,
  event_start_status,
  time_related_vars,
  time_offset
)
```

Arguments

`data`	The dataset that you want to split according to the `time_var` option.
`by`	The time period that you want to split the dataset by. The size of the variable must be in proportion to the the `time_var`. The by variable can also be a vector for each time split, useful if the effect has large varyations over time.
`time_var`	The name of the main time variable in the dataset. This variable must be a numeric variable.
`event_var`	The event variable
`event_start_status`	The start status of the event status, e.g. "Alive"
`time_related_vars`	A dataset often contains other variabels that you want to update during the split, most commonly these are age or calendar time.
`time_offset`	If you want to skip the initial years you can offset the entire dataset by setting this variable. See detailed description below.

Details

Important note: The time variables must have the same time unit. I.e. function can not dedu if all variables are in years or if one happens to be in days.

Value

`data.frame` with the split data. The starting time for each period is named `Start_time` and the ending time is called `Stop_time`. Note that the resulting event_var will now contain the time-splitted eventvar.

The time_offset - details

Both time_var and other variables will be adjusted by the time_offset, e.g. if we the time scale is in years and we want to skip the first 4 years we set the `time_offset = 4`. In the outputted dataset the smallest `time_var` will be 0. *Note:* 0 will not be included as we generally want to look at those that survived the start date, e.g. if a patient dies on the 4-year mark we would not include him/her in our study.

Examples

```
test_data <- data.frame(
  id = 1:4,
  time = c(4, 3.5, 1, 5),
  event = c("alive", "censored", "dead", "dead"),
  age = c(62.2, 55.3, 73.7, 46.3),
  date = as.Date(
    c("2003-01-01",
      "2010-04-01",
      "2013-09-20",
      "2002-02-23")),
  stringsAsFactors = TRUE
)
timeSplitter(test_data, .5,
             time_var = "time",
             time_related_vars = c("age", "date"),
             event_var = "event")
```

Index

∗ **forestplot wrappers**
    `forestplotCombineRegrObj`, 8
    `forestplotRegrObj`, 10

`addNonlinearity`, 3
`AIC`, *4*
`anova`, *3*

`BIC`, *4*

`caDescribeOpts`, 5
`confint`, *6*
`confint.ols`, 6
`confint_robust`, 7
`coxph`, *3*

`describeFactors`, *5*
`describeMean`, *5*
`detectCores`, *4*

`forestplot`, *9*, *13*
`forestplotCombineRegrObj`, 8, *13*
`forestplotRegrObj`, *9*, 10
`fpBoxSize` *(*`forestplotRegrObj`*)*, 10

`Greg-package`, 2

`isFitCoxPH`, 14
`isFitLogit` *(*`isFitCoxPH`*)*, 14

`makeCluster`, *4*

`ns`, *3*

`ols`, *2*, *6*

`plot.plotHR` *(*`plotHR`*)*, 17
`plotHR`, 17
`print.plotHR` *(*`plotHR`*)*, 17
`printCrudeAndAdjustedModel`, *5*
`prNlChooseDf`, *4*

`rcs`, *3*
`robcov_alt`, 21

`termplot`, *17*
`tidy.rms`, 22
`timeSplitter`, *2*, 24

`vcovHC`, *7*, *21*