Package ‘insight’
August 17, 2021

Type Package
Title Easy Access to Model Information for Various Model Objects
Version 0.14.3
Maintainer Daniel Lüdecke <d.luedecke@uke.de>
Description A tool to provide an easy, intuitive and consistent access to information contained in various R models, like model formulas, model terms, information about random effects, data that was used to fit the model or data from response variables. 'insight' mainly revolves around two types of functions: Functions that find (the names of) information, starting with 'find_'; and functions that get the underlying data, starting with 'get_'. The package has a consistent syntax and works with many different model objects, where otherwise functions to access these information are missing.

License GPL-3
URL https://easystats.github.io/insight/
BugReports https://github.com/easystats/insight/issues
Depends R (>= 3.4)
Imports methods, stats, utils
Suggests AER, afex, aod, BayesFactor, bayestestR, bbmle, bdsmatrix, betareg, biglm, blavaan, blme, boot, brms, censReg, cgam, clubSandwich, coxme, eplm, crch, dplyr, effectsize, emmeans, epiR, estimatr, feisr, fixest, fungible, gam, gamlss, gamm4, gbm, gee, geepack, GLMMadaptive, glmmTMB, gmln, gt, htr, ivreg, JM, knitr, lavaan, lfe, lme4, lmeTest, logistf, MASS, Matrix, mclust, MCMCgamm, merTools, metaBMA, mgcv, mice, mlogit, multgee, nlme, nnet, nonnest2, ordinal, panelr, parameters, parsnip, performance, plm, pscl, psych, quantreg, rmarkdown, rms, robustbase, robustlmm, rstanarm (>= 2.21.1), rstans, rstanapi, sandwich, speedglm, spelling, splines, statmod, survey, survival, testthat, tripack, truncreg, VGAM

VignetteBuilder knitr
Encoding UTF-8
Language: en-US
RoxygenNote: 7.1.1.9001
Config/testthat/edition: 3
NeedsCompilation: no
Author: Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>, @strengejacke),
  Dominique Makowski [aut, ctb] (<https://orcid.org/0000-0001-5375-9967>, @Dom_Makowski),
  Indrajeet Patil [aut, ctb] (<https://orcid.org/0000-0003-1995-6531>, @patilindrajeets),
  Philip Waggoner [aut, ctb] (<https://orcid.org/0000-0002-7825-7573>),
  Mattan S. Ben-Shachar [aut, ctb] (<https://orcid.org/0000-0002-4287-4801>),
  Brenton M. Wiernik [aut] (<https://orcid.org/0000-0001-9560-6336>, @bmwiernik),
  Alex Hayes [rev] (<https://orcid.org/0000-0002-4985-5160>)
Repository: CRAN
Date/Publication: 2021-08-17 08:00:05 UTC

R topics documented:

all_models_equal .............................................. 4
check_if_installed ........................................... 5
clean_names .................................................. 6
clean_parameters .............................................. 7
color_if ......................................................... 8
data_match ..................................................... 10
data_relocate ................................................ 10
data_restoretype ............................................. 11
data_to_long .................................................. 12
display ......................................................... 14
download_model .............................................. 15
ellipsis_info .................................................. 16
export_table ................................................... 17
find_algorithm ............................................... 19
find_formula .................................................. 20
find_interactions ............................................ 22
find_offset .................................................... 23
find_parameters .............................................. 24
find_parameters.averaging .................................... 25
find_parameters.betamfx ..................................... 26
find_parameters.BGGM ......................................... 28
find_parameters.emmGrid ..................................... 30
find_parameters.gamlss ...................................... 31
find_parameters.glmmTMB .................................... 32
find_parameters.zeroinfl ................................... 34
R topics documented:

find_predictors .................................................... 35
find_random .......................................................... 36
find_random_slopes .................................................. 37
find_response ......................................................... 38
find_smooth ........................................................... 39
find_statistic .......................................................... 39
find_terms .............................................................. 40
find_variables .......................................................... 41
find_weights ............................................................ 43
fish ................................................................. 43
format_bf ............................................................. 44
format_ci .............................................................. 45
format_message ....................................................... 46
format_number .......................................................... 47
format_p ............................................................... 48
format_pd .............................................................. 49
format_rope ............................................................. 49
format_string ........................................................... 50
format_table ............................................................ 51
format_value ............................................................ 52
get_auxiliary ............................................................ 55
get_call .............................................................. 56
get_data .............................................................. 57
get_deviance ............................................................ 59
get_df ............................................................... 60
get_family ............................................................. 61
get_intercept ........................................................... 61
get_loglikelihood ...................................................... 62
get_modelmatrix ......................................................... 63
get_parameters .......................................................... 64
get_parameters.betamfx ............................................... 65
get_parameters.betareg ............................................... 66
get_parameters.BGGM .................................................. 67
get_parameters.emmGrid ............................................... 71
get_parameters.gamm ................................................... 72
get_parameters.glmm .................................................... 73
get_parameters.htest .................................................... 74
get_parameters.zeroinfl ............................................... 75
get_predicted ........................................................... 76
get_predicted_ci .......................................................... 79
get_predictors ........................................................... 81
get_priors ............................................................. 82
get_random ............................................................. 83
get_residuals ............................................................. 84
get_response ............................................................ 85
get_sigma .............................................................. 86
get_statistic ............................................................ 87
get_varcov ............................................................. 89
all_models_equal  
Checks if all objects are models of same class

Description
Small helper that checks if all objects are supported (regression) model objects and of same class.

Usage

\[
\text{all_models_equal}(\ldots, \text{verbose} = \text{FALSE})
\]

\[
\text{all_models_same_class}(\ldots, \text{verbose} = \text{FALSE})
\]

Arguments

\[
\ldots \quad \text{A list of objects.}
\]

verbose  
Toggle off warnings.

Value
A logical, TRUE if x are all supported model objects of same class.
check_if_installed

Examples

if (require("lme4")) {
  data(mtcars)
  data(sleepstudy)

  m1 <- lm(mpg ~ wt + cyl + vs, data = mtcars)
  m2 <- lm(mpg ~ wt + cyl, data = mtcars)
  m3 <- lmer(Reaction ~ Days + (1 | Subject), data = sleepstudy)
  m4 <- glm(formula = vs ~ wt, family = binomial(), data = mtcars)

  all_models_same_class(m1, m2)
  all_models_same_class(m1, m2, m3)
  all_models_same_class(m1, m4, m2, m3, verbose = TRUE)
  all_models_same_class(m1, m4, mtcars, m2, m3, verbose = TRUE)
}

check_if_installed  Checking if needed package is installed

Description

Checking if needed package is installed

Usage

check_if_installed(
  package,
  reason = "for this function to work",
  stop = TRUE,
  minimum_version = NULL,
  ...
)

Arguments

package A string (vector) naming the package, whose installation needs to be checked in any of the libraries.
reason A phrase describing why the package is needed. The default is a generic description.
stop Logical that decides whether the function should stop if the needed package is not installed.
minimum_version String, representing the minimum package version that is required. If NULL, no check for minimum version is done. Note that minimum_version only works when package is of length 1.
... Currently ignored
clean_names

Value

If \texttt{stop = TRUE}, and package is not yet installed, the function stops and throws an error. Else, a named logical vector is returned, indicating which of the packages are installed, and which not.

Examples

```r
## Not run:
check_if_installed("inexistent_package")
check_if_installed("insight")
check_if_installed("insight", minimum_version = "99.8.7")

x <- check_if_installed(c("inexistent", "also_not_here"), stop = FALSE)
x

## End(Not run)
```

---

clean_names

Get clean names of model terms

Description

This function "cleans" names of model terms (or a character vector with such names) by removing patterns like \texttt{log()} or \texttt{as.factor()} etc.

Usage

```r
clean_names(x, ...)
```

### S3 method for class 'character'
```r
clean_names(x, include_names = FALSE, ...)
```

Arguments

- \texttt{x} A fitted model, or a character vector.
- \texttt{...} Currently not used.
- \texttt{include_names} Logical, if \texttt{TRUE}, returns a named vector where names are the original values of \texttt{x}.

Value

The "cleaned" variable names as character vector, i.e. pattern like \texttt{s()} for splines or \texttt{log()} are removed from the model terms.
Note

Typically, this method is intended to work on character vectors, in order to remove patterns that obscure the variable names. For convenience reasons it is also possible to call `clean_names()` also on a model object. If `x` is a regression model, this function is (almost) equal to calling `find_variables()`. The main difference is that `clean_names()` always returns a character vector, while `find_variables()` returns a list of character vectors, unless `flatten = TRUE`. See 'Examples'.

Examples

```r
# example from stats::glm
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- as.numeric(gl(3, 1, 9))
treatment <- gl(3, 3)
m <- glm(counts ~ log(outcome) + as.factor(treatment), family = poisson())
clean_names(m)

# difference "clean_names()" and "find_variables()"
if (require("lme4")) {
m <- glmer(
  cbind(incidence, size - incidence) ~ period + (1 | herd),
  data = cbpp,
  family = binomial
)

  clean_names(m)
  find_variables(m)
  find_variables(m, flatten = TRUE)
}
```

---

**clean_parameters**  Get clean names of model parameters

Description

This function "cleans" names of model parameters by removing patterns like "r_" or "b[]" (mostly applicable to Stan models) and adding columns with information to which group or component parameters belong (i.e. fixed or random, count or zero-inflated...)

The main purpose of this function is to easily filter and select model parameters, in particular of - but not limited to - posterior samples from Stan models, depending on certain characteristics. This might be useful when only selective results should be reported or results from all parameters should be filtered to return only certain results (see `print_parameters()`).

Usage

```r
clean_parameters(x, ...)
```
Arguments

x  A fitted model.
...  Currently not used.

Details

The Effects column indicate if a parameter is a fixed or random effect. The Component can either be conditional or zero_inflated. For models with random effects, the Group column indicates the grouping factor of the random effects. For multivariate response models from brms or rstanarm, an additional Response column is included, to indicate which parameters belong to which response formula. Furthermore, Cleaned_Parameter column is returned that contains "human readable" parameter names (which are mostly identical to Parameter, except for for models from brms or rstanarm, or for specific terms like smooth- or spline-terms).

Value

A data frame with "cleaned" parameter names and information on effects, component and group where parameters belong to. To be consistent across different models, the returned data frame always has at least four columns Parameter, Effects, Component and Cleaned_Parameter. See 'Details'.

Examples

## Not run:
library(brms)
model <- download_model("brms_zi_2")
clean_parameters(model)

## End(Not run)

---

color_if  

Color-formatting for data columns based on condition

Description

Convenient function that formats columns in data frames with color codes, where the color is chosen based on certain conditions. Columns are then printed in color in the console.

Usage

color_if(
  x,
  columns,
  predicate = `>`,
  value = 0,
  color_if = "green",
  color_else = "red",
)

digits = 2
)
colour_if(
  x,
columns,
predicate = `>`,
value = 0,
colour_if = "green",
colour_else = "red",
digits = 2
)

Arguments

x  A data frame
columns  Character vector with column names of x that should be formatted.
predicate  A function that takes columns and value as input and which should return TRUE
or FALSE, based on if the condition (in comparison with value) is met.
value  The comparator. May be used in conjunction with predicate to quickly set
up a function which compares elements in columns to value. May be ignored
when predicate is a function that internally computes other comparisons. See
'Examples'.
colour_if, colour_if  Character vector, indicating the color code used to format values in x that meet
the condition of predicate and value. May be one of "red", "yellow", 
"green", "blue", "violet", "cyan" or "grey". Formatting is also possible
with "bold" or "italic".
colour_else  See colour_if, but only for conditions that are not met.
digits  Digits for rounded values.

Details

The predicate-function simply works like this: which(predicate(x[,columns],value))

Examples

# all values in Sepal.Length larger than 5 in green, all remaining in red
x <- colour_if(iris[1:10, ], columns = "Sepal.Length", predicate = `>`, value = 5)
x
cat(x$Sepal.Length)

# all levels "setosa" in Species in green, all remaining in red
x <- color_if(iris, columns = "Species", predicate = `==`, value = "setosa")
cat(x$Species)

# own function, argument "value" not needed here
p <- function(x, y) {
  x >= 4.9 & x <= 5.1
}
# all values in Sepal.Length between 4.9 and 5.1 in green, all remaining in red
x <- color_if(iris[1:10, ], columns = "Sepal.Length", predicate = p)
cat(x$Sepal.Length)

---

**data_match**

*Find rows of a data frame that are matching a specific subset*

**Description**

Find row indices of a data frame that are matching a specific configuration.

**Usage**

data_match(x, to)

**Arguments**

- **x**: A data frame.
- **to**: The data frame of which to meet the characteristics.

**Examples**

```r
matching_rows <- data_match(mtcars, data.frame(vs = 0, am = 1))
mtcars[matching_rows, ]

matching_rows <- data_match(mtcars, data.frame(vs = 0, am = c(0, 1)))
mtcars[matching_rows, ]
```

---

**data_relocate**

*Relocate (reorder) columns of a data frame*

**Description**

Relocate (reorder) columns of a data frame

**Usage**

data_relocate(data, cols, before = NULL, after = NULL, safe = TRUE)
**Arguments**

- **data**: A data frame to pivot.
- **cols**: A character vector indicating the names of the columns to move.
- **before, after**: Destination of columns. Supplying neither will move columns to the left-hand side; specifying both is an error.
- **safe**: If TRUE, will disregard non-existing columns.

**Value**

A data frame with reordered columns.

**Examples**

```r
# Reorder columns
data_relocate(iris, cols = "Species", before = "Sepal.Length")
data_relocate(iris, cols = "Species", before = "Sepal.Width")
data_relocate(iris, cols = "Sepal.Width", after = "Species")
data_relocate(iris, cols = c("Species", "Petal.Length"), after = "Sepal.Width")
```

**data_restoretype**

*Restore the type of columns according to a reference data frame*

**Description**

Restore the type of columns according to a reference data frame

**Usage**

```r
data_restoretype(data, reference = NULL)
```

**Arguments**

- **data**: A data frame to pivot.
- **reference**: A reference data frame from which to find the correct column types.

**Examples**

```r
data <- data.frame(
    Sepal.Length = c("1", "3", "2"),
    Species = c("setosa", "versicolor", "setosa"),
    New = c("1", "3", "4")
)
fixed <- data_restoretype(data, reference = iris)
summary(fixed)
```
Description

This function "lengthens" data, increasing the number of rows and decreasing the number of columns. This is a dependency-free base-R equivalent of `tidyr::pivot_longer()`.

Usage

data_to_long(
data,
cols = "all",
colnames_to = "Name",
values_to = "Value",
rows_to = NULL,
..., names_to = colnames_to
)

data_to_wide(
data,
values_from = "Value",
colnames_from = "Name",
rows_from = NULL,
sep = "_",
..., names_from = colnames_from
)

reshape_longer(
data,
cols = "all",
colnames_to = "Name",
values_to = "Value",
rows_to = NULL,
..., names_to = colnames_to
)

reshape_wider(
data,
values_from = "Value",
colnames_from = "Name",
rows_from = NULL,
sep = "_",
...,
data_to_long

names_from = colnames_from
)

Arguments

- **data**: A data frame to pivot.
- **cols**: A vector of column names or indices to pivot into longer format.
- **colnames_to**: The name of the new column that will contain the column names.
- **values_to**: The name of the new column that will contain the values of the pivoted variables.
- **rows_to**: The name of the column that will contain the row-number from the original data. If NULL, will be removed.
- **...**: Additional arguments passed on to methods.
- **names_to, names_from**: Same as colnames_to, is there for compatibility with tidyr::pivot_longer().
- **values_from**: The name of the column that contains the values of the put in the columns.
- **colnames_from**: The name of the column that contains the levels to be used as future columns.
- **rows_from**: The name of the column that identifies the rows. If NULL, will use all the unique rows.
- **sep**: The indicating a separating character in the variable names in the wide format.

Value

data.frame

Examples

```
wide_data <- data.frame(replicate(5, rnorm(10)))

# From wide to long
# ------------------
# Default behaviour (equivalent to tidyr::pivot_longer(wide_data, cols = 1:5))
data_to_long(wide_data)

# Customizing the names
data_to_long(wide_data,
  cols = c(1, 2),
  colnames_to = "Column",
  values_to = "Numbers",
  rows_to = "Row"
)

# From long to wide
# ---------------
long_data <- data_to_long(wide_data, rows_to = "Row_ID") # Save row number
data_to_wide(long_data,
  colnames_from = "Name",
  values_from = "Value",
  rows_from = "Row_ID"
)
```
# Full example
# ------------------
if (require("psych")) {
  data <- psych::bfi # Wide format with one row per participant's personality test

  # Pivot long format
  long <- data_to_long(data,
                       cols = "d", # Select all columns that contain a digit
                       colnames_to = "Item",
                       values_to = "Score",
                       rows_to = "Participant"
  )

  # Separate facet and question number
  long$Facet <- gsub("\\d", "", long$Item)
  long$Item <- gsub("[A-Z]\", "", long$Item)
  long$Item <- paste0("I", long$Item)

  wide <- data_to_wide(long,
                       colnames_from = "Item",
                       values_from = "Score"
  )
  head(wide)
}

---

**display**  
*Generic export of data frames into formatted tables*

**Description**

`display()` is a generic function to export data frames into various table formats (like plain text, markdown, ...). `print_md()` usually is a convenient wrapper for `display(format = "markdown")`. Similar, `print_html()` is a shortcut for `display(format = "html")`. See the documentation for the specific objects’ classes.

**Usage**

```r
display(object, ...)
print_md(x, ...)
print_html(x, ...)
```
download_model

print_md(x, ...)

## S3 method for class 'data.frame'
print_html(x, ...)

Arguments

object, x A data frame.
...
Arguments passed to other methods.
format String, indicating the output format. Can be "markdown" or "html".

Value

Depending on format, either an object of class gt_tbl or a character vector of class knitr_kable.

Examples

display(iris[1:5, ])

download_model

Download circus models

download_model

Description

Downloads pre-compiled models from the circus-repository. The circus-repository contains a variety of fitted models to help the systematic testing of other packages.

Usage

download_model(name, url = NULL)

Arguments

name Model name.
url String with the URL from where to download the model data. Optional, and should only be used in case the repository-URL is changing. By default, models are downloaded from https://raw.githubusercontent.com/easystats/circus/master/data/.

Details

The code that generated the model is available at the https://easystats.github.io/circus/reference/index.html.

Value

A model from the circus-repository.
ellipsis_info

Gather information about objects in ellipsis (dot dot dot)

Description

Provides information regarding the models entered in an ellipsis. It detects whether all are models, regressions, nested regressions etc., assigning different classes to the list of objects.

Usage

```r
ellipsis_info(objects, ...)
```

## Default S3 method:

```r
ellipsis_info(..., only_models = TRUE)
```

Arguments

- `objects, ...` Arbitrary number of objects.
- `only_models` Only keep supported models (default to TRUE).

Value

The list with objects that were passed to the function, including additional information as attributes (e.g. if models have same response or are nested).

Examples

```r
m1 <- lm(Sepal.Length ~ Petal.Width + Species, data = iris)
m2 <- lm(Sepal.Length ~ Species, data = iris)
m3 <- lm(Sepal.Length ~ Petal.Width, data = iris)
m4 <- lm(Sepal.Length ~ 1, data = iris)
m5 <- lm(Petal.Width ~ 1, data = iris)
objects <- ellipsis_info(m1, m2, m3, m4)
class(objects)

objects <- ellipsis_info(m1, m2, m4)
attributes(objects)$is_nested

objects <- ellipsis_info(m1, m2, m5)
attributes(objects)$same_response
```
export_table

Data frame and Tables Pretty Formatting

Description

Data frame and Tables Pretty Formatting

Usage

```
export_table(
  x,
  sep = " | ",
  header = "-",
  empty_line = NULL,
  digits = 2,
  protect_integers = TRUE,
  missing = "",
  width = NULL,
  format = NULL,
  title = NULL,
  caption = title,
  subtitle = NULL,
  footer = NULL,
  align = NULL,
  group_by = NULL,
  zap_small = FALSE,
  verbose = TRUE
)
```

Arguments

- **x**: A data frame.
- **sep**: Column separator.
- **header**: Header separator. Can be `NULL`.
- **empty_line**: Separator used for empty lines. If `NULL`, line remains empty (i.e. filled with whitespaces).
- **digits**: Number of digits for rounding or significant figures. May also be "signif" to return significant figures or "scientific" to return scientific notation. Control the number of digits by adding the value as suffix, e.g. `digits = "scientific4"` to have scientific notation with 4 decimal places, or `digits = "signif5"` for 5 significant figures (see also `signif()`).
- **protect_integers**: Should integers be kept as integers (i.e., without decimals)?
- **missing**: Value by which NA values are replaced. By default, an empty string (i.e. "") is returned for NA.
width Minimum width of the returned string. If not NULL and width is larger than the string's length, leading whitespaces are added to the string.

format Name of output-format, as string. If NULL (or "text"), returned output is used for basic printing. Can be one of NULL (the default) resp. "text" for plain text, "markdown" (or "md") for markdown and "html" for HTML output.

title, caption, subtitle Table title (same as caption) and subtitle, as strings. If NULL, no title or subtitle is printed, unless it is stored as attributes (table_title, or its alias table_caption, and table_subtitle).

footer Table footer, as string. For markdown-formatted tables, table footers, due to the limitation in markdown rendering, are actually just a new text line under the table.

align Column alignment. For markdown-formatted tables, the default align = NULL will right-align numeric columns, while all other columns will be left-aligned. If format = "html", the default is left-align first column and center all remaining. May be a string to indicate alignment rules for the complete table, like "left", "right", "center" or "firstleft" (to left-align first column, center remaining); or maybe a string with abbreviated alignment characters, where the length of the string must equal the number of columns, for instance, align = "lccrl" would left-align the first column, center the second and third, right-align column four and left-align the fifth column. For HTML-tables, may be one of "center", "left" or "right".

group_by Name of column in x that indicates grouping for tables. Only applies when format = "html". group_by is passed down to gt::gt(groupname_col = group_by).

zap_small Logical, if TRUE, small values are rounded after digits decimal places. If FALSE, values with more decimal places than digits are printed in scientific notation.

verbose Toggle messages and warnings.

Value
A data frame in character format.

Note
The values for caption, subtitle and footer can also be provided as attributes of x, e.g. if caption = NULL and x has attribute table_caption, the value for this attribute will be used as table caption. table_subtitle is the attribute for subtitle, and table_footer for footer.

See Also
Vignettes Formatting, printing and exporting tables and Formatting model parameters.

Examples
```
cat(export_table(iris))
cat(export_table(iris, sep = " ", header = "*", digits = 1))
## Not run:
```
find_algorithm

# colored footers
data(iris)
x <- as.data.frame(iris[1:5, ])
attr(x, "table_footer") <- c("This is a yellow footer line.", "yellow")
cat(export_table(x))

attr(x, "table_footer") <- list(  
c("\nA yellow line", "yellow"),  
c("\nAnd a red line", "red"),  
c("\nAnd a blue line", "blue")  
)
cat(export_table(x))

attr(x, "table_footer") <- list(  
c("Without the ", "yellow"),  
c("new-line character ", "red"),  
c("we can have multiple colors per line.", "blue")  
)
cat(export_table(x))

## End(Not run)

find_algorithm Find sampling algorithm and optimizers

Description

Returns information on the sampling or estimation algorithm as well as optimization functions, or for Bayesian model information on chains, iterations and warmup-samples.

Usage

find_algorithm(x, ...)

Arguments

x A fitted model.
...

Currently not used.

Value

A list with elements depending on the model.
For frequentist models:

- algorithm, for instance "OLS" or "ML"
- optimizer, name of optimizing function, only applies to specific models (like gam)

For frequentist mixed models:

- algorithm, for instance "REML" or "ML"
• optimizer, name of optimizing function

For Bayesian models:

• algorithm, the algorithm
• chains, number of chains
• iterations, number of iterations per chain
• warmup, number of warmups per chain

Examples

if (require("lme4")) {
  data(sleepstudy)
  m <- lmer(Reaction ~ Days + (1 | Subject), data = sleepstudy)
  find_algorithm(m)
}
## Not run:
library(rstanarm)
m <- stan_lmer(Reaction ~ Days + (1 | Subject), data = sleepstudy)
find_algorithm(m)
## End(Not run)

find_formula

Find model formula

Description

Returns the formula(s) for the different parts of a model (like fixed or random effects, zero-inflated component, ...). formula_ok() checks if a model formula has valid syntax regarding writing TRUE instead of T inside poly() and that no data names are used (i.e. no data$variable, but rather variable).

Usage

find_formula(x, verbose = TRUE, ...)

formula_ok(x, verbose = TRUE, ...)

Arguments

x A fitted model.
verbose Toggle warnings.
... Currently not used.
Value

A list of formulas that describe the model. For simple models, only one list-element, conditional, is returned. For more complex models, the returned list may have following elements:

- conditional, the "fixed effects" part from the model. One exception are DirichletRegModel models from DirichletReg, which has two or three components, depending on model.
  - \item random, the "random effects" part from the model (or the \id\ for gee-models and similar)
  - \item zero_inflated, the "fixed effects" part from the zero-inflation component of the model
  - \item zero_inflated_random, the "random effects" part from the zero-inflation component of the model
  - \item dispersion, the dispersion formula
  - \item instruments, for fixed-effects regressions like \ivreg::ivreg(), \lfe::felm() or \plm::plm(), the instrumental variables
  - \item cluster, for fixed-effects regressions like \lfe::felm(), the cluster specification
  - \item correlation, for models with correlation-component like \nlme::gls(), the formula that describes the correlation structure
  - \item slopes, for fixed-effects individual-slope models like \feisr::feis(), the formula for the slope parameters
  - \item precision, for DirichletRegModel models from \pkg{DirichletReg}, when parametrization (i.e. \model) is "alternative".

Note

For models of class lme or gls the correlation-component is only returned, when it is explicitly defined as named argument (form), e.g. corAR1(form = ~1 | Mare)

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_formula(m)

if (require("lme4")) {
m <- lmer(Sepal.Length ~ Sepal.Width + (1 | Species), data = iris)
f <- find_formula(m)
f}
find_interactions

Description

Returns all lowest to highest order interaction terms from a model.

Usage

```r
find_interactions(
  x,
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion",
                 "instruments"),
  flatten = FALSE
)
```

Arguments

- `x` A fitted model.
- `component` Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the `conditional` component is also called `count` or `mean` component, depending on the model.
- `flatten` Logical, if `TRUE`, the values are returned as character vector, not as list. Duplicated values are removed.

Value

A list of character vectors that represent the interaction terms. Depending on component, the returned list has following elements (or `NULL`, if model has no interaction term):

- `conditional`, interaction terms that belong to the "fixed effects" terms from the model
- `zero_inflated`, interaction terms that belong to the "fixed effects" terms from the zero-inflation component of the model
- `instruments`, for fixed-effects regressions like `ivreg`, `felm` or `plm`, interaction terms that belong to the instrumental variables
find_offset

Examples

```r
data(mtcars)

m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_interactions(m)

m <- lm(mpg ~ wt * cyl + vs * hp * gear + carb, data = mtcars)
find_interactions(m)
```

---

find_offset  

Find possible offset terms in a model

Description

Returns a character vector with the name(s) of offset terms.

Usage

```r
find_offset(x)
```

Arguments

- `x`  
  A fitted model.

Value

A character vector with the name(s) of offset terms.

Examples

```r
# Generate some zero-inflated data
set.seed(123)
N <- 100 # Samples
x <- runif(N, 0, 10) # Predictor
off <- rgamma(N, 3, 2) # Offset variable
yhat <- -1 + x * 0.5 + log(off) # Prediction on log scale
dat <- data.frame(y = NA, x, logOff = log(off))
dat$y <- rpois(N, exp(yhat)) # Poisson process
dat$y <- ifelse(rbinom(N, 1, 0.3), 0, dat$y) # Zero-inflation process

if (require("pscl")) {
  m1 <- zeroinfl(y ~ offset(logOff) + x | 1, data = dat, dist = "poisson")
  find_offset(m1)

  m2 <- zeroinfl(y ~ x | 1, data = dat, offset = logOff, dist = "poisson")
  find_offset(m2)
}
```
find_parameters Find names of model parameters

Description

Returns the names of model parameters, like they typically appear in the `summary()` output. For Bayesian models, the parameter names equal the column names of the posterior samples after coercion from `as.data.frame()`. See the documentation for your object’s class:

- Bayesian models (`rstanarm, brms, MCMCglmm, ...`)
- Generalized additive models (`mgcv, VGAM, ...`)
- Marginal effects models (`mfx`)
- Estimated marginal means (`emmeans`)
- Mixed models (`lme4, glmmTMB, GLMMadaptive, ...`)
- Zero-inflated and hurdle models (`pscl, ...`)
- Models with special components (`betareg, MuMIn, ...`)

Usage

```r
find_parameters(x, ...)
```

## Default S3 method:
```r
default find_parameters(x, flatten = FALSE, verbose = TRUE, ...)
```

Arguments

- `x` A fitted model.
- `...` Currently not used.
- `flatten` Logical, if `TRUE`, the values are returned as character vector, not as list. Duplicated values are removed.
- `verbose` Toggle messages and warnings.

Value

A list of parameter names. For simple models, only one list-element, `conditional`, is returned.

Examples

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_parameters(m)
```
find_parameters.averaging

Find model parameters from models with special components

Description

Returns the names of model parameters, like they typically appear in the `summary()` output.

Usage

```r
## S3 method for class 'averaging'
find_parameters(x, component = c("conditional", "full"), flatten = FALSE, ...)

## S3 method for class 'betareg'
find_parameters(
x,
  component = c("all", "conditional", "precision", "location", "distributional", "auxiliary"),
  flatten = FALSE,
  ...
)

## S3 method for class 'DirichletRegModel'
find_parameters(
x,
  component = c("all", "conditional", "precision", "location", "distributional", "auxiliary"),
  flatten = FALSE,
  ...
)

## S3 method for class 'mjoint'
find_parameters(
x,
  component = c("all", "conditional", "survival"),
  flatten = FALSE,
  ...
)

## S3 method for class 'glmx'
find_parameters(
x,
  component = c("all", "conditional", "extra"),
  flatten = FALSE,
  ...
)
```
Arguments

x
A fitted model.

component
Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model, the dispersion term, the instrumental variables or marginal effects be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variables (so called fixed-effects regressions), or models with marginal effects from `mfx`. May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as conditional, zero_inflated, smooth_terms, or instruments are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

flatten
Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.

... Currently not used.

Value

A list of parameter names. The returned list may have following elements:

- conditional, the "fixed effects" part from the model.
- full, parameters from the full model.

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_parameters(m)

Find names of model parameters from marginal effects models

Description

Returns the names of model parameters, like they typically appear in the `summary()` output.

Usage

```
## S3 method for class 'betamfx'
find_parameters(
x,
component = c("all", "conditional", "precision", "marginal", "location"),
```
Arguments

- **x**: A fitted model.
- **component**: Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model, the dispersion term, the instrumental variables or marginal effects be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variables (so called fixed-effects regressions), or models with marginal effects from `mfx`. May be abbreviated. Note that the *conditional* component is also called *count* or *mean* component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as conditional, zero_inflated, smooth_terms, or instruments are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.
- **flatten**: Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.
- **...**: Currently not used.

Value

A list of parameter names. The returned list may have following elements:

- **conditional**, the "fixed effects" part from the model.
- **marginal**, the marginal effects.
- **precision**, the precision parameter.

Examples

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_parameters(m)
```
find_parameters.BGGM  Find names of model parameters from Bayesian models

Description

Returns the names of model parameters, like they typically appear in the summary() output. For Bayesian models, the parameter names equal the column names of the posterior samples after coercion from as.data.frame().

Usage

## S3 method for class 'BGGM'
find_parameters(
  x,
  component = c("correlation", "conditional", "intercept", "all"),
  flatten = FALSE,
  ...
)

## S3 method for class 'BFBayesFactor'
find_parameters(
  x,
  effects = c("all", "fixed", "random"),
  component = c("all", "extra"),
  flatten = FALSE,
  ...
)

## S3 method for class 'MCMCglmm'
find_parameters(
  x,
  effects = c("all", "fixed", "random"),
  flatten = FALSE, ...
)

## S3 method for class 'bamlss'
find_parameters(
  x,
  flatten = FALSE,
  component = c("all", "conditional", "location", "distributional", "auxiliary"),
  parameters = NULL,
  ...
)

## S3 method for class 'brmsfit'
find_parameters(
  x,
  effects = "all",
  component = "all",
  flatten = FALSE,
  parameters = NULL,
find_parameters.BGGM

## Arguments

**x**  A fitted model.

**component**  Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model, the dispersion term, the instrumental variables or marginal effects be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variables (so called fixed-effects regressions), or models with marginal effects from `mfx`. May be abbreviated. Note that the **conditional** component is also called **count** or **mean** component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as conditional, zero_inflated, smooth_terms, or instruments are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.
find_parameters.emmGrid

flatten Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.

... Currently not used.
effects Should parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
parameters Regular expression pattern that describes the parameters that should be returned.

Value

A list of parameter names. For simple models, only one list-element, conditional, is returned. For more complex models, the returned list may have following elements:

- conditional, the "fixed effects" part from the model
- random, the "random effects" part from the model
- zero_inflated, the "fixed effects" part from the zero-inflation component of the model
- zero_inflated_random, the "random effects" part from the zero-inflation component of the model
- smooth_terms, the smooth parameters

Furthermore, some models, especially from brms, can also return auxiliary parameters. These may be one of the following:

- sigma, the residual standard deviation (auxiliary parameter)
- dispersion, the dispersion parameters (auxiliary parameter)
- beta, the beta parameter (auxiliary parameter)
- simplex, simplex parameters of monotonic effects (brms only)
- mix, mixture parameters (brms only)
- shiftprop, shifted proportion parameters (brms only)

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_parameters(m)

find_parameters.emmGrid

Find model parameters from estimated marginal means objects

Description

Returns the parameter names from a model.
find_parameters.gamlss

Usage

## S3 method for class 'emmGrid'
find_parameters(x, flatten = FALSE, merge_parameters = FALSE, ...)

Arguments

x
A fitted model.

flatten
Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.

merge_parameters
Logical, if TRUE and x has multiple columns for parameter names (like emmGrid objects may have), these are merged into a single parameter column, with parameters names and values as values.

...
Currently not used.

Value

A list of parameter names. For simple models, only one list-element, conditional, is returned.

Examples

data(mtcars)
model <- lm(mpg ~ wt * factor(cyl), data = mtcars)
if (require("emmeans", quietly = TRUE)) {
  emm <- emmeans(model, c("wt", "cyl"))
  find_parameters(emm)
}

find_parameters.gamlss

Find names of model parameters from generalized additive models

Description

Returns the names of model parameters, like they typically appear in the summary() output.

Usage

## S3 method for class 'gamlss'
find_parameters(x, flatten = FALSE, ...)

## S3 method for class 'gam'
find_parameters(
  x,
  component = c("all", "conditional", "smooth_terms", "location"),
  flatten = FALSE,
  ...
)
Arguments

- **x**: A fitted model.
- **flatten**: Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.
- **component**: Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model, the dispersion term, the instrumental variables or marginal effects be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variables (so called fixed-effects regressions), or models with marginal effects from `mfx`. May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as conditional, zero_inflated, smooth_terms, or instruments are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

Value

- A list of parameter names. The returned list may have following elements:
  - **conditional**, the "fixed effects" part from the model.
  - **smooth_terms**, the smooth parameters.

Examples

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_parameters(m)
```

Description

Returns the names of model parameters, like they typically appear in the `summary()` output.

Usage

```r
## S3 method for class 'glmmTMB'
find_parameters(
x, 
effects = c("all", "fixed", "random"),
)```
component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
flatten = FALSE,
...)

## S3 method for class 'merMod'
find_parameters(x, effects = c("all", "fixed", "random"), flatten = FALSE, ...)

Arguments

x A fitted model.
effects Should parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
component Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model or the dispersion term? Applies to models with zero-inflated and/or dispersion formula. Note that the conditional component is also called count or mean component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as conditional or zero_inflated are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma or dispersion (and other auxiliary parameters) are returned.
flatten Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.
... Currently not used.

Value

A list of parameter names. The returned list may have following elements:

- conditional, the "fixed effects" part from the model.
- random, the "random effects" part from the model.
- zero_inflated, the "fixed effects" part from the zero-inflation component of the model.
- zero_inflated_random, the "random effects" part from the zero-inflation component of the model.
- dispersion, the dispersion parameters (auxiliary parameter)

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_parameters(m)
find_parameters.zeroinfl

Find names of model parameters from zero-inflated models

Description

Returns the names of model parameters, like they typically appear in the summary() output.

Usage

## S3 method for class 'zeroinfl'
find_parameters(
  x,
  component = c("all", "conditional", "zi", "zero_inflated"),
  flatten = FALSE,
  ...
)

## S3 method for class 'mhurdle'
find_parameters(
  x,
  component = c("all", "conditional", "zi", "zero_inflated", "infrequent_purchase", "ip", "auxiliary"),
  flatten = FALSE,
  ...
)

Arguments

x            A fitted model.

component    Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model, the dispersion term, the instrumental variables or marginal effects be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variables (so called fixed-effects regressions), or models with marginal effects from mfx. May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as conditional, zero_inflated, smooth_terms, or instruments are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

flatten      Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.

...          Currently not used.
find_predictors

Value
A list of parameter names. The returned list may have following elements:

- conditional, the "fixed effects" part from the model.
- zero_inflated, the "fixed effects" part from the zero-inflation component of the model.

Examples

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_predictors(m)
```

Description
Returns the names of the predictor variables for the different parts of a model (like fixed or random effects, zero-inflated component,...). Unlike `find_parameters()`, the names from `find_predictors()` match the original variable names from the data that was used to fit the model.

Usage

```r
find_predictors(x, ...)
## Default S3 method:
find_predictors(
x,
effects = c("fixed", "random", "all"),
component = c("all", "conditional", "zi", "zero_inflated", "dispersion",
"instruments", "correlation", "smooth_terms"),
flatten = FALSE,
verbose = TRUE,
...)
```

Arguments

- `x` A fitted model.
- `...` Currently not used.
- `effects` Should variables for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
- `component` Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model.
find_random

flatten Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.
verbose Toggle warnings.

Value

A list of character vectors that represent the name(s) of the predictor variables. Depending on the combination of the arguments effects and component, the returned list has following elements:

- conditional, the "fixed effects" terms from the model
- random, the "random effects" terms from the model
- zero_inflated, the "fixed effects" terms from the zero-inflation component of the model
- zero_inflated_random, the "random effects" terms from the zero-inflation component of the model
- dispersion, the dispersion terms
- instruments, for fixed-effects regressions like ivreg, felm or plm, the instrumental variables
- correlation, for models with correlation-component like gls, the variables used to describe the correlation structure

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_predictors(m)

Description

Return the name of the grouping factors from mixed effects models.

Usage

find_random(x, split_nested = FALSE, flatten = FALSE)

Arguments

x A fitted mixed model.
split_nested Logical, if TRUE, terms from nested random effects will be returned as separated elements, not as single string with colon. See ’Examples’.
flatten Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.
find_random_slopes

Value
A list of character vectors that represent the name(s) of the random effects (grouping factors). Depending on the model, the returned list has following elements:

- random, the "random effects" terms from the conditional part of model
- zero_inflated_random, the "random effects" terms from the zero-inflation component of the model

Examples
if (require("lme4")) {
  data(sleepstudy)
  sleepstudy$mygrp <- sample(1:5, size = 180, replace = TRUE)
  sleepstudy$mysubgrp <- NA
  for (i in 1:5) {
    filter_group <- sleepstudy$mygrp == i
    sleepstudy$mysubgrp[filter_group] <- sample(1:30, size = sum(filter_group), replace = TRUE)
  }
  m <- lmer(Reaction ~ Days + (1 | mygrp / mysubgrp) + (1 | Subject), data = sleepstudy)
  find_random(m)
  find_random(m, split_nested = TRUE)
}

find_random_slopes  Find names of random slopes

Description
Return the name of the random slopes from mixed effects models.

Usage
find_random_slopes(x)

Arguments
x A fitted mixed model.

Value
A list of character vectors with the name(s) of the random slopes, or NULL if model has no random slopes. Depending on the model, the returned list has following elements:

- random, the random slopes from the conditional part of model
- zero_inflated_random, the random slopes from the zero-inflation component of the model
find_response

Examples

```r
if (require("lme4")) {
  data(sleepstudy)
  m <- lmer(Reaction ~ Days + (1 + Days | Subject), data = sleepstudy)
  find_random_slopes(m)
}
```
find_smooth  
*Find smooth terms from a model object*

**Description**

Return the names of smooth terms from a model object.

**Usage**

```r
find_smooth(x, flatten = FALSE)
```

**Arguments**

- `x` A (gam) model.
- `flatten` Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.

**Value**

A character vector with the name(s) of the smooth terms.

**Examples**

```r
if (require("mgcv")) {
  data(iris)
  model <- gam(Petal.Length ~ Petal.Width + s(Sepal.Length), data = iris)
  find_smooth(model)
}
```

find_statistic  
*Find statistic for model*

**Description**

Returns the statistic for a regression model (\(t\)-statistic, \(z\)-statistic, etc.).

Small helper that checks if a model is a regression model object and return the statistic used.

**Usage**

```r
find_statistic(x, ...)
```

**Arguments**

- `x` An object.
- `...` Currently not used.
find_terms

**Value**

A character describing the type of statistic. If there is no statistic available with a distribution, NULL will be returned.

**Examples**

```
# regression model object
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
find_statistic(m)
```

---

**find_terms**

*Find all model terms*

**Description**

Returns a list with the names of all terms, including response value and random effects, "as is". This means, on-the-fly transformations or arithmetic expressions like `log()`, `I()`, `as.factor()` etc. are preserved.

**Usage**

```
find_terms(x, flatten = FALSE, verbose = TRUE, ...)
```

**Arguments**

- **x** A fitted model.
- **flatten** Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.
- **verbose** Toggle warnings.
- **...** Currently not used.

**Value**

A list with (depending on the model) following elements (character vectors):

- **response**, the name of the response variable
- **conditional**, the names of the predictor variables from the conditional model (as opposed to the zero-inflated part of a model)
- **random**, the names of the random effects (grouping factors)
- **zero_inflated**, the names of the predictor variables from the zero-inflated part of the model
- **zero_inflated_random**, the names of the random effects (grouping factors)
- **dispersion**, the name of the dispersion terms
- **instruments**, the names of instrumental variables

Returns NULL if no terms could be found (for instance, due to problems in accessing the formula).
find_variables

Note

The difference to find_variables() is that find_terms() may return a variable multiple times in case of multiple transformations (see examples below), while find_variables() returns each variable name only once.

Examples

```r
if (require("lme4")) {
  data(sleepstudy)
  m <- lmer(
    log(Reaction) ~ Days + I(Days^2) + (1 + Days + exp(Days) | Subject),
    data = sleepstudy
  )

  find_terms(m)
}
```

find_variables

Find names of all variables

Description

Returns a list with the names of all variables, including response value and random effects.

Usage

```r
find_variables(
  x,
  effects = c("all", "fixed", "random"),
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion",
                 "instruments", "smooth_terms"),
  flatten = FALSE,
  verbose = TRUE
)
```

Arguments

- `x` A fitted model.
- `effects` Should variables for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
- `component` Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model.
find_variables

- **flatten**: Logical, if TRUE, the values are returned as character vector, not as list. Duplicated values are removed.
- **verbose**: Toggle warnings.

**Value**

A list with (depending on the model) following elements (character vectors):

- **response**: the name of the response variable
- **\texttt{\textbackslash conditional\texttt{\textbackslash}}**, the names of the predictor variables from the *conditional* model (as opposed to the zero-inflated part of a model)
- **\texttt{\textbackslash random\texttt{\textbackslash}}**, the names of the random effects (grouping factors)
- **\texttt{\textbackslash zero\_inflated\texttt{\textbackslash}}**, the names of the predictor variables from the *zero-inflated* part of the model
- **\texttt{\textbackslash zero\_inflated\_random\texttt{\textbackslash}}**, the names of the random effects (grouping factors)
- **\texttt{\textbackslash dispersion\texttt{\textbackslash}}**, the name of the dispersion terms
- **\texttt{\textbackslash instruments\texttt{\textbackslash}}**, the names of instrumental variables

**Note**

The difference to **find_terms()** is that **find_variables()** returns each variable name only once, while **find_terms()** may return a variable multiple times in case of transformations or when arithmetic expressions were used in the formula.

**Examples**

```r
if (require("lme4")) {
  data(cbpp)
  data(sleepstudy)
  # some data preparation...
  cbpp$trials <- cbpp$size - cbpp$incidence
  sleepstudy$mygrp <- sample(1:5, size = 180, replace = TRUE)
  sleepstudy$mysubgrp <- NA
  for (i in 1:5) {
    filter_group <- sleepstudy$mygrp == i
    sleepstudy$mysubgrp[filter_group] <-
      sample(1:30, size = sum(filter_group), replace = TRUE)
  }
  m1 <- glmer(
    cbind(incidence, size - incidence) ~ period + (1 | herd),
    data = cbpp,
    family = binomial
  )
}
```
find_weights

Find names of model weights

Description

Returns the name of the variable that describes the weights of a model.

Usage

find_weights(x, ...)

Arguments

x

A fitted model.

...

Currently not used.

Value

The name of the weighting variable as character vector, or NULL if no weights were specified.

Examples

data(mtcars)
mtcars$weight <- rnorm(nrow(mtcars), 1, .3)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars, weights = weight)
find_weights(m)

fish

Sample data set

Description

A sample data set, used in tests and some examples.
format_bf

Bayes Factor formatting

Description

Bayes Factor formatting

Usage

format_bf(
  bf,
  stars = FALSE,
  stars_only = FALSE,
  name = "BF",
  protect_ratio = FALSE,
  na_reference = NA,
  exact = FALSE
)

Arguments

bf                  Bayes Factor.
stars               Add significance stars (e.g., p < .001***).
stars_only          Return only significance stars.
name                Name prefixing the text. Can be NULL.
protect_ratio       Should values smaller than 1 be represented as ratios?
na_reference        How to format missing values (NA).
exact               Should very large or very small values be reported with a scientific format (e.g., 4.24e5), or as truncated values (as "> 1000" and "< 1/1000")?

Value

A formatted string.

Examples

format_bf(bfs <- c(0.000045, 0.033, NA, 1557, 3.54))
format_bf(bfs, exact = TRUE, name = NULL)
format_bf(bfs, stars = TRUE)
format_bf(bfs, protect_ratio = TRUE)
format_bf(bfs, protect_ratio = TRUE, exact = TRUE)
format_bf(bfs, na_reference = 1)
**Description**

Confidence/Credible Interval (CI) Formatting

**Usage**

```r
format_ci(
  CI_low,
  CI_high,
  ci = 0.95,
  digits = 2,
  brackets = TRUE,
  width = NULL,
  width_low = width,
  width_high = width,
  missing = "",
  zap_small = FALSE
)
```

**Arguments**

- `CI_low`: Lower CI bound.
- `CI_high`: Upper CI bound.
- `ci`: CI level in percentage.
- `digits`: Number of digits for rounding or significant figures. May also be "signif" to return significant figures or "scientific" to return scientific notation. Control the number of digits by adding the value as suffix, e.g. `digits = "scientific4"` to have scientific notation with 4 decimal places, or `digits = "signif5"` for 5 significant figures (see also `signif()`).
- `brackets`: Either a logical, and if TRUE (default), values are encompassed in square brackets. If FALSE or NULL, no brackets are used. Else, a character vector of length two, indicating the opening and closing brackets.
- `width`: Minimum width of the returned string. If not NULL and width is larger than the string’s length, leading whitespaces are added to the string. If `width="auto"`, width will be set to the length of the longest string.
- `width_low, width_high`: Like width, but only applies to the lower or higher confidence interval value. This can be used when the values for the lower and upper CI are of very different length.
- `missing`: Value by which NA values are replaced. By default, an empty string (i.e. "") is returned for NA.
zap_small Logical, if TRUE, small values are rounded after digits decimal places. If FALSE, values with more decimal places than digits are printed in scientific notation.

Value

A formatted string.

Examples

format_ci(1.20, 3.57, ci = 0.90)
format_ci(1.20, 3.57, ci = NULL)
format_ci(1.20, 3.57, ci = NULL, brackets = FALSE)
format_ci(c(1.205645, 23.4), c(3.57, -1.35), ci = 0.90)
format_ci(c(1.20, NA, NA), c(3.57, -1.35, NA), ci = 0.90)

# automatic alignment of width, useful for printing multiple CIs in columns
x <- format_ci(c(1.205, 23.4, 100.43), c(3.57, -13.35, 9.4))
cat(x, sep = "\n")

x <- format_ci(c(1.205, 23.4, 100.43), c(3.57, -13.35, 9.4), width = "auto")
cat(x, sep = "\n")

format_message

Format messages and warnings

Description

Inserts line breaks into a longer message or warning string. Line length is adjusted to maximum length of the console, if the width can be accessed. By default, new lines are indented by two whitespace.

Usage

format_message(string, ..., line_length = options()$width)

Arguments

string A string.
...
line_length Numeric, the maximum length of a line.

Value

A formatted string.
Examples

```r
msg <- format_message("Much too long string for just one line, I guess!",
  line_length = 15
)
message(msg)

msg <- format_message("Much too long string for just one line, I guess!",
  "First new line",
  "Second new line",
  "(both indented)",
  line_length = 30
)
message(msg)
```

---

**format_number**  
*Convert number to words*

---

**Description**

Convert number to words

**Usage**

```r
format_number(x, textual = TRUE, ...)
```

**Arguments**

- `x`  
  Number.

- `textual`  
  Return words. If FALSE, will run `format_value()`.

- `...`  
  Arguments to be passed to `format_value()` if textual is FALSE.

**Value**

A formatted string.

**Note**

The code has been adapted from here https://github.com/ateucher/useful_code/blob/master/R/numbers2words.r

**Examples**

```r
format_number(2)
fraction_number(45)
format_number(324.68765)
```
**format_p**  
*p-values formatting*

---

**Description**

Format p-values.

**Usage**

```r
format_p(
  p,  
  stars = FALSE,  
  stars_only = FALSE,  
  name = "p",  
  missing = "",  
  digits = 3,  
  ...
)
```

**Arguments**

- `p` value or vector of p-values.
- `stars` Add significance stars (e.g., `p < .001***`).
- `stars_only` Return only significance stars.
- `name` Name prefixing the text. Can be `NULL`.
- `missing` Value by which `NA` values are replaced. By default, an empty string (i.e. "") is returned for `NA`.
- `digits` Number of significant digits. May also be "scientific" to return exact p-values in scientific notation, or "apa" to use an APA 7th edition-style for p-values (equivalent to `digits = 3`). If "scientific", control the number of digits by adding the value as a suffix, e.g., `digits = "scientific4"` to have scientific notation with 4 decimal places.
- `...` Arguments from other methods.

---

**Value**

A formatted string.

**Examples**

```r
format_p(c(.02, .065, 0, .23))
format_p(c(.02, .065, 0, .23), name = NULL)
format_p(c(.02, .065, 0, .23), stars_only = TRUE)
```

```r
model <- lm(mpg ~ wt + cyl, data = mtcars)
p <- coef(summary(model))[, 4]
```
**format_pd**

```r
format_p(p, digits = "apa")
format_p(p, digits = "scientific")
format_p(p, digits = "scientific2")
```

---

**format_pd**  
_Probability of direction (pd) formatting_

**Description**

Probability of direction (pd) formatting

**Usage**

```r
format_pd(pd, stars = FALSE, stars_only = FALSE, name = "pd")
```

**Arguments**

- **pd**: Probability of direction (pd).
- **stars**: Add significance stars (e.g., p < .001***).
- **stars_only**: Return only significance stars.
- **name**: Name prefixing the text. Can be **NULL**.

**Value**

A formatted string.

**Examples**

```r
format_pd(0.12)
format_pd(c(0.12, 1, 0.9999, 0.98, 0.995, 0.96), name = NULL)
format_pd(c(0.12, 1, 0.9999, 0.98, 0.995, 0.96), stars = TRUE)
```

---

**format_rope**  
_Percentage in ROPE formatting_

**Description**

Percentage in ROPE formatting

**Usage**

```r
format_rope(rope_percentage, name = "in ROPE", digits = 2)
```
Arguments

rope_percentage  Value or vector of percentages in ROPE.
name            Name prefixing the text. Can be NULL.
digits          Number of significant digits. May also be "scientific" to return exact p-values in scientific notation, or "apa" to use an APA 7th edition-style for p-values (equivalent to digits = 3). If "scientific", control the number of digits by adding the value as a suffix, e.g. digits = "scientific4" to have scientific notation with 4 decimal places.

Value

A formatted string.

Examples

format_rope(c(0.02, 0.12, 0.357, 0))
format_rope(c(0.02, 0.12, 0.357, 0), name = NULL)

Description

String Values Formatting

Usage

format_string(x, ...)

## S3 method for class 'character'
format_string(x, length = NULL, abbreviate = "...", ...)

Arguments

x                String value.
...              Arguments passed to or from other methods.
length           Numeric, maximum length of the returned string. If not NULL, will shorten the string to a maximum length, however, it will not truncate inside words. I.e. if the string length happens to be inside a word, this word is removed from the returned string, so the returned string has a maximum length of length, but might be shorter.
abbreviate       String that will be used as suffix, if x was shortened.

Value

A formatted string.
Examples

s <- "This can be considered as very long string!"
# string is shorter than max.length, so returned as is
format_string(s, 60)

# string is shortened to as many words that result in
# a string of maximum 20 chars
format_string(s, 20)

---

format_table

Parameter table formatting

Description

This function takes a data frame with model parameters as input and formats certain columns into a more readable layout (like collapsing separate columns for lower and upper confidence interval values). Furthermore, column names are formatted as well. Note that format_table() converts all columns into character vectors!

Usage

format_table(
  x,
  pretty_names = TRUE,
  stars = FALSE,
  digits = 2,
  ci_width = "auto",
  ci_brackets = TRUE,
  ci_digits = 2,
  p_digits = 3,
  rope_digits = 2,
  zap_small = FALSE,
  preserve_attributes = FALSE,
  verbose = TRUE,
  ...
)

parameters_table(
  x,
  pretty_names = TRUE,
  stars = FALSE,
  digits = 2,
  ci_width = "auto",
  ci_brackets = TRUE,
  ci_digits = 2,
  p_digits = 3,
  rope_digits = 2,
format_table

zap_small = FALSE,
preserve_attributes = FALSE,
verbose = TRUE,
...
)

Arguments

x A data frame of model's parameters, as returned by various functions of the easystats-packages. May also be a result from broom::tidy().

pretty_names Return "pretty" (i.e. more human readable) parameter names.

stars Add significance stars (e.g., p < .001***).

digits, ci_digits, p_digits, rope_digits Number of digits for rounding or significant figures. May also be "signif" to return significant figures or "scientific" to return scientific notation. Control the number of digits by adding the value as suffix, e.g. digits = "scientific4" to have scientific notation with 4 decimal places, or digits = "signif5" for 5 significant figures (see also signif()).

ci_width Minimum width of the returned string for confidence intervals. If not NULL and width is larger than the string's length, leading whitespaces are added to the string. If width="auto", width will be set to the length of the longest string.

ci_brackets Logical, if TRUE (default), CI-values are encompassed in square brackets (else in parentheses).

zap_small Logical, if TRUE, small values are rounded after digits decimal places. If FALSE, values with more decimal places than digits are printed in scientific notation.

preserve_attributes Logical, if TRUE, preserves all attributes from the input data frame.

verbose Toggle messages and warnings.

... Arguments passed to or from other methods.

Value

A data frame. Note that format_table() converts all columns into character vectors!

See Also

Vignettes Formatting, printing and exporting tables and Formatting model parameters.

Examples

format_table(head(iris), digits = 1)

if (require("parameters")) {
  x <- model_parameters(lm(Sepal.Length ~ Species * Sepal.Width, data = iris))
as.data.frame(format_table(x))
as.data.frame(format_table(x, p_digits = "scientific"))
if (require("rstanarm") && require("parameters")) {
  model <- stan_glm(Sepal.Length ~ Species, data = iris, refresh = 0, seed = 123)
  x <- model_parameters(model, ci = c(0.69, 0.89, 0.95))
  as.data.frame(format_table(x))
}

---

### format_value

**Numeric Values Formatting**

**Description**

Numeric Values Formatting

**Usage**

format_value(x, ...)

```r
## S3 method for class 'data.frame'
format_value(
  x,
  digits = 2,
  protect_integers = FALSE,
  missing = "",
  width = NULL,
  as_percent = FALSE,
  zap_small = FALSE,
  ...
)

## S3 method for class 'numeric'
format_value(
  x,
  digits = 2,
  protect_integers = FALSE,
  missing = "",
  width = NULL,
  as_percent = FALSE,
  zap_small = FALSE,
  ...
)
```

**Arguments**

- `x`  
  Numeric value.
... Arguments passed to or from other methods.

digits Number of digits for rounding or significant figures. May also be "signif" to return significant figures or "scientific" to return scientific notation. Control the number of digits by adding the value as suffix, e.g. digits = "scientific4" to have scientific notation with 4 decimal places, or digits = "signif5" for 5 significant figures (see also signif()).

protect_integers Should integers be kept as integers (i.e., without decimals)?

missing Value by which NA values are replaced. By default, an empty string (i.e. "") is returned for NA.

width Minimum width of the returned string. If not NULL and width is larger than the string's length, leading whitespaces are added to the string.

as_percent Logical, if TRUE, value is formatted as percentage value.

zap_small Logical, if TRUE, small values are rounded after digits decimal places. If FALSE, values with more decimal places than digits are printed in scientific notation.

Value
A formatted string.

Examples

format_value(1.20)
format_value(1.2)
format_value(1.2012313)
format_value(c(0.0045, 234, -23))
format_value(c(0.0045, .12, .34))
format_value(c(0.0045, .12, .34), as_percent = TRUE)
format_value(c(0.0045, .12, .34), digits = "scientific")
format_value(c(0.0045, .12, .34), digits = "scientific2")

# default
format_value(c(0.0045, .123, .345))
# significant figures
format_value(c(0.0045, .123, .345), digits = "signif")

format_value(as.factor(c("A", "B", "A")))
format_value(iris$Species)

format_value(3)
format_value(3, protect_integers = TRUE)

format_value(iris)
get_auxiliary

Get auxiliary parameters from models

Description

Returns the requested auxiliary parameters from models, like dispersion, sigma, or beta...

Usage

get_auxiliary(
  x,
  type = "sigma",
  summary = TRUE,
  centrality = "mean",
  verbose = TRUE,
  ...
)

Arguments

x
  A model.

type
  The name of the auxiliary parameter that should be retrieved. "sigma" is available for most models, "dispersion" for models of class glm, glmerMod or glmmTMB as well as brmsfit. "beta" and other parameters are currently only returned for brmsfit models. See 'Details'.

summary
  Logical, indicates whether the full posterior samples (summary = FALSE)) or the summarized centrality indices of the posterior samples (summary = TRUE)) should be returned as estimates.

centrality
  Only for models with posterior samples, and when summary = TRUE. In this case, centrality = "mean" would calculate means of posterior samples for each parameter, while centrality = "median" would use the more robust median value as measure of central tendency.

verbose
  Toggle warnings.

...
  Currently not used.

Details

Currently, only sigma and the dispersion parameter are returned, and only for a limited set of models.

Sigma Parameter: See get_sigma().

Dispersion Parameter: There are many different definitions of "dispersion", depending on the context. get_auxiliary() returns the dispersion parameters that usually can be considered as variance-to-mean ratio for generalized (linear) mixed models. Exceptions are models of class glmmTMB, where the dispersion equals $\sigma^2$. In detail, the computation of the dispersion parameter
for generalized linear models is the ratio of the sum of the squared working-residuals and the
residual degrees of freedom. For mixed models of class glmer, the dispersion parameter is also
called \( \phi \) and is the ratio of the sum of the squared Pearson-residuals and the residual degrees
of freedom. For models of class glmmTMB, dispersion is \( \sigma^2 \).

**brms models:** For models of class brmsfit, there are different options for the type argument.
See a list of supported auxiliary parameters here: `find_parameters.BGGM()`.

### Value

The requested auxiliary parameter, or **NULL** if this information could not be accessed.

### Examples

```r
# from ?glm
clotting <- data.frame(
  u = c(5, 10, 15, 20, 30, 40, 60, 80, 100),
  lot1 = c(118, 58, 42, 35, 27, 25, 21, 19, 18),
  lot2 = c(69, 35, 26, 21, 18, 16, 13, 12, 12)
)
model <- glm(lot1 ~ log(u), data = clotting, family = Gamma())
get_auxiliary(model, type = "dispersion") # same as summary(model)$dispersion
```

---

### get_call

*Get the model’s function call*

### Description

Returns the model’s function call when available.

### Usage

`get_call(x)`

### Arguments

- `x` : A fitted mixed model.

### Value

A function call.
get_data

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_call(m)

if (require("lme4")) {
m <- lmer(Sepal.Length ~ Sepal.Width + (1 | Species), data = iris)
get_call(m)
}

dput(get_data)

get_data

Get the data that was used to fit the model

Description

This functions tries to get the data that was used to fit the model and returns it as data frame.

Usage

give_data(x, ...)

## S3 method for class 'gee'
give_data(x, effects = c("all", "fixed", "random"), verbose = TRUE, ...)

## S3 method for class 'rqss'
give_data(
x,
  component = c("all", "conditional", "smooth_terms"),
  verbose = TRUE,
  ...
)

## S3 method for class 'hurdle'
give_data(
x,
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  verbose = TRUE,
  ...
)

## S3 method for class 'zcpplm'
give_data(
x,
  component = c("all", "conditional", "zi", "zero_inflated"),
  verbose = TRUE,
  ...
)
get_data

## S3 method for class 'glmmTMB'
get_data(
  x,
  effects = c("all", "fixed", "random"),
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  verbose = TRUE,
  ...
)

## S3 method for class 'merMod'
get_data(x, effects = c("all", "fixed", "random"), verbose = TRUE, ...)

## S3 method for class 'glmmadmb'
get_data(x, effects = c("all", "fixed", "random"), verbose = TRUE, ...)

## S3 method for class 'rlmerMod'
get_data(x, effects = c("all", "fixed", "random"), ...)

## S3 method for class 'clmm'
get_data(x, effects = c("all", "fixed", "random"), ...)

## S3 method for class 'mixed'
get_data(x, effects = c("all", "fixed", "random"), ...)

## S3 method for class 'afex_aov'
get_data(x, shape = c("long", "wide"), ...)

## S3 method for class 'lme'
get_data(x, effects = c("all", "fixed", "random"), ...)

## S3 method for class 'MixMod'
get_data(
  x,
  effects = c("all", "fixed", "random"),
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  verbose = TRUE,
  ...
)

## S3 method for class 'brmsfit'
get_data(x, effects = "all", component = "all", verbose = TRUE, ...)

## S3 method for class 'stanreg'
get_data(x, effects = c("all", "fixed", "random"), verbose = TRUE, ...)

## S3 method for class 'MCMCglmm'
get_data(x, effects = c("all", "fixed", "random"), ...)

Arguments

x A fitted model.

... Currently not used.

effects Should model data for fixed effects, random effects or both be returned? Only applies to mixed models.

verbose Toggle messages and warnings.

component Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model.

shape Return long or wide data? Only applicable in repeated measures designs.

Value

The data that was used to fit the model.

Note

Unlike model.frame(), which may contain transformed variables (e.g. if poly() or scale() was used inside the formula to specify the model), get_data() aims at returning the "original", untransformed data (if possible). Consequently, column names are changed accordingly, i.e. "log(x)" will become "x" etc. for all data columns with transformed values.

Examples

if (require("lme4")) {
  data(cbpp, package = "lme4")
  cbpp$trials <- cbpp$size - cbpp$incidence
  m <- glm(cbind(incidence, trials) ~ period, data = cbpp, family = binomial)
  head(get_data(m))
}

generate_data

Description

Returns model deviance (see stats::deviance()).

Usage

generate_data(x, ...) # Only applicable in repeated measures designs.

get_deviance

Model Deviance

Usage

generate_deviance(x, ...) # Only applicable in repeated measures designs.
get_df

Arguments

x A model.
...
verbose Toggle warnings and messages.

Details

For GLMMs of class glmerMod, glmmTMB or MixMod, the absolute unconditional deviance is returned (see "Details" in ?lme4::merMod-class"), i.e. minus twice the log-likelihood. To get the relative conditional deviance (relative to a saturated model, conditioned on the conditional modes of random effects), use deviance(). The value returned ‘get_deviance()’ usually equals the deviance-value from the ‘summary()’.

Value

The model deviance.

Examples

data(mtcars)
x <- lm(mpg ~ cyl, data = mtcars)
get_deviance(x)

------

get_df Extract degrees of freedom

Description

Estimate or extract residual or model-based degrees of freedom from regression models.

Usage

get_df(x, ...)

## Default S3 method:
get_df(x, type = "residual", verbose = TRUE, ...)

Arguments

x A statistical model.
...

verbose Toggle warnings.

Details

For GLMMs of class glmerMod, glmmTMB or MixMod, the absolute unconditional deviance is returned (see "Details" in ?lme4::merMod-class"), i.e. minus twice the log-likelihood. To get the relative conditional deviance (relative to a saturated model, conditioned on the conditional modes of random effects), use deviance(). The value returned ‘get_deviance()’ usually equals the deviance-value from the ‘summary()’.

Value

The model deviance.

Examples

data(mtcars)
x <- lm(mpg ~ cyl, data = mtcars)
get_deviance(x)

------

get_df Extract degrees of freedom

Description

Estimate or extract residual or model-based degrees of freedom from regression models.

Usage

get_df(x, ...)

## Default S3 method:
get_df(x, type = "residual", verbose = TRUE, ...)

Arguments

x A statistical model.
...

verbose Toggle warnings.

Details

For GLMMs of class glmerMod, glmmTMB or MixMod, the absolute unconditional deviance is returned (see "Details" in ?lme4::merMod-class"), i.e. minus twice the log-likelihood. To get the relative conditional deviance (relative to a saturated model, conditioned on the conditional modes of random effects), use deviance(). The value returned ‘get_deviance()’ usually equals the deviance-value from the ‘summary()’.

Value

The model deviance.

Examples

data(mtcars)
x <- lm(mpg ~ cyl, data = mtcars)
get_deviance(x)

------

get_df Extract degrees of freedom

Description

Estimate or extract residual or model-based degrees of freedom from regression models.

Usage

get_df(x, ...)

## Default S3 method:
get_df(x, type = "residual", verbose = TRUE, ...)

Arguments

x A statistical model.
...

verbose Toggle warnings.

Details

For GLMMs of class glmerMod, glmmTMB or MixMod, the absolute unconditional deviance is returned (see "Details" in ?lme4::merMod-class"), i.e. minus twice the log-likelihood. To get the relative conditional deviance (relative to a saturated model, conditioned on the conditional modes of random effects), use deviance(). The value returned ‘get_deviance()’ usually equals the deviance-value from the ‘summary()’.

Value

The model deviance.

Examples

data(mtcars)
x <- lm(mpg ~ cyl, data = mtcars)
get_deviance(x)

------

get_df Extract degrees of freedom

Description

Estimate or extract residual or model-based degrees of freedom from regression models.

Usage

get_df(x, ...)

## Default S3 method:
get_df(x, type = "residual", verbose = TRUE, ...)

Arguments

x A statistical model.
...

verbose Toggle warnings.

Details

For GLMMs of class glmerMod, glmmTMB or MixMod, the absolute unconditional deviance is returned (see "Details" in ?lme4::merMod-class"), i.e. minus twice the log-likelihood. To get the relative conditional deviance (relative to a saturated model, conditioned on the conditional modes of random effects), use deviance(). The value returned ‘get_deviance()’ usually equals the deviance-value from the ‘summary()’.

Value

The model deviance.

Examples

data(mtcars)
x <- lm(mpg ~ cyl, data = mtcars)
get_deviance(x)

------

get_df Extract degrees of freedom

Description

Estimate or extract residual or model-based degrees of freedom from regression models.

Usage

get_df(x, ...)

## Default S3 method:
get_df(x, type = "residual", verbose = TRUE, ...)

Arguments

x A statistical model.
...

verbose Toggle warnings.
### Examples

```r
model <- lm(Sepal.Length ~ Petal.Length * Species, data = iris)
get_df(model) # same as df.residual(model)
get_df(model, type = "model") # same as attr(logLik(model), "df")
```

---

### Description

A robust and resilient alternative to `stats::family`. To avoid issues with models like `gamm4`.

### Usage

```r
get_family(x, ...)
```

### Arguments

- **x**: A statistical model.
- **...**: Further arguments passed to methods.

### Examples

```r
data(mtcars)
x <- glm(vs ~ wt, data = mtcars, family = "binomial")
get_family(x)

if (require("mgcv")) {
  x <- mgcv::gamm(
    vs ~ am + s(wt),
    random = list(cyl = ~1),
    data = mtcars,
    family = "binomial"
  )
  get_family(x)
}
```

---

### Description

Returns the value at the intercept (i.e., the intercept parameter), and `NA` if there isn’t one.

### Usage

```r
get_intercept(x, ...)
```
get_loglikelihood

Arguments

x A model.

... Not used.

Value

The value of the intercept.

Examples

get_intercept(lm(Sepal.Length ~ Petal.Width, data = iris))
get_intercept(lm(Sepal.Length ~ 0 + Petal.Width, data = iris))

if (require("lme4")) {
  get_intercept(lme4::lmer(Sepal.Length ~ Sepal.Width + (1 | Species), data = iris))
}
if (require("gamm4")) {
  get_intercept(gamm4::gamm4(Sepal.Length ~ s(Petal.Width), data = iris))
}

get_loglikelihood Log-Likelihood

Description

A robust function to compute the log-likelihood of a model, as well as individual log-likelihoods (for each observation) whenever possible. Can be used as a replacement for stats::logLik() out of the box, as the returned object is of the same class (and it gives the same results by default).

Usage

get_loglikelihood(x, ...)

loglikelihood(x, ...)

## S3 method for class 'lm'
get_loglikelihood(x, estimator = "ML", REML = FALSE, verbose = TRUE, ...)

Arguments

x A model.

... Passed down to logLik(), if possible.

estimator Corresponds to the different estimators for the standard deviation of the errors. If estimator="ML" (default), the scaling is done by n (the biased ML estimator), which is then equivalent to using stats::logLik(). If estimator="OLS", it returns the unbiased OLS estimator.
REML Only for linear models. This argument is present for compatibility with stats::logLik(). Setting it to TRUE will overwrite the estimator argument and is thus equivalent to setting estimator = "REML". It will give the same results as stats::logLik(..., REML = TRUE). Note that individual log-likelihoods are not available under REML.

verbose Toggle warnings and messages.

Value
An object of class "logLik", also containing the log-likelihoods for each observation as a per_observation attribute (attributes(get_loglikelihood(x))$per_observation) when possible. The code was partly inspired from the nonnest2 package.

Examples
x <- lm(Sepal.Length ~ Petal.Width + Species, data = iris)

get_loglikelihood(x, estimator = "ML") # Equivalent to stats::logLik(x)
get_loglikelihood(x, estimator = "REML") # Equivalent to stats::logLik(x, REML = TRUE)
get_loglikelihood(x, estimator = "OLS")

get_modelmatrix

Model Matrix

Description
Creates a design matrix from the description. Any character variables are coerced to factors.

Usage
get_modelmatrix(x, ...)

Arguments
x An object.
... Passed down to other methods (mainly model.matrix()).

Examples
data(mtcars)

model <- lm(am ~ vs, data = mtcars)
get_modelmatrix(model)
Description

Returns the coefficients (or posterior samples for Bayesian models) from a model. See the documentation for your object’s class:

- Bayesian models (rstanarm, brms, MCMCglmm, ...)
- Estimated marginal means (emmeans)
- Generalized additive models (mgcv, VGAM, ...)
- Marginal effects models (mfx)
- Mixed models (lme4, glmmTMB, GLMMadaptive, ...)
- Zero-inflated and hurdle models (pscl, ...)
- Models with special components (betareg, MuMIn, ...)
- Hypothesis tests (htest)

Usage

get_parameters(x, ...)

## Default S3 method:
get_parameters(x, verbose = TRUE, ...)

Arguments

x A fitted model.

... Currently not used.

verbose Toggle messages and warnings.

Details

In most cases when models either return different "effects" (fixed, random) or "components" (conditional, zero-inflated, ...), the arguments effects and component can be used.

get_parameters() is comparable to coef(), however, the coefficients are returned as data frame (with columns for names and point estimates of coefficients). For Bayesian models, the posterior samples of parameters are returned.

Value

- for non-Bayesian models, a data frame with two columns: the parameter names and the related point estimates.
- for Anova (aov()) with error term, a list of parameters for the conditional and the random effects parameters
get_parameters.betamfx

Examples

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_parameters(m)
```
get_parameters.betareg

Get model parameters from models with special components

Description

Returns the coefficients from a model.

Usage

## S3 method for class 'betareg'
get_parameters(
x, 
component = c("all", "conditional", "precision", "location", "distributional", "auxiliary"),
...
)

## S3 method for class 'DirichletRegModel'
get_parameters(
x, 
component = c("all", "conditional", "precision", "location", "distributional", "auxiliary"),
...
)

## S3 method for class 'averaging'
get_parameters(x, component = c("conditional", "full"), ...)

## S3 method for class 'glmx'
get_parameters(
x, 
component = c("all", "conditional", "extra", "location", "distributional", "auxiliary"),
...
)

## S3 method for class 'clm2'
get_parameters(x, component = c("all", "conditional", "scale"), ...)

## S3 method for class 'mvord'
get_parameters(
x, 
component = c("all", "conditional", "thresholds", "correlation"),
...
)
get_parameters.BGGM

## S3 method for class 'mjoint'
get_parameters(x, component = c("all", "conditional", "survival"), ...)

**Arguments**

- `x`: A fitted model.
- `component`: Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model.
- `...`: Currently not used.

**Value**

A data frame with three columns: the parameter names, the related point estimates and the component.

**Examples**

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_parameters(m)
```

---

**get_parameters.BGGM** Get model parameters from Bayesian models

**Description**

Returns the coefficients (or posterior samples for Bayesian models) from a model.

**Usage**

```r
## S3 method for class 'BGGM'
get_parameters(
  x,
  component = c("correlation", "conditional", "intercept", "all"),
  summary = FALSE,
  centrality = "mean",
  ...
)
```

```r
## S3 method for class 'MCMCglmm'
get_parameters(
  x,
  effects = c("fixed", "random", "all"),
)
get_parameters.BGGM

summary = FALSE,
    centrality = "mean",
    ...
)

## S3 method for class 'BFBayesFactor'
get_parameters(
    x,
    effects = c("all", "fixed", "random"),
    component = c("all", "extra"),
    iterations = 4000,
    progress = FALSE,
    verbose = TRUE,
    summary = FALSE,
    centrality = "mean",
    ...
)

## S3 method for class 'stanmvreg'
get_parameters(
    x,
    effects = c("fixed", "random", "all"),
    parameters = NULL,
    summary = FALSE,
    centrality = "mean",
    ...
)

## S3 method for class 'brmsfit'
get_parameters(
    x,
    effects = "fixed",
    component = "all",
    parameters = NULL,
    summary = FALSE,
    centrality = "mean",
    ...
)

## S3 method for class 'stanreg'
get_parameters(
    x,
    effects = c("fixed", "random", "all"),
    component = c("location", "all", "conditional", "smooth_terms", "sigma",
                  "distributional", "auxiliary"),
    parameters = NULL,
    summary = FALSE,
    centrality = "mean",
    ...
get_parameters.BGGM

## S3 method for class 'bayesx'
get_parameters(
  x,
  component = c("conditional", "smooth_terms", "all"),
  summary = FALSE,
  centrality = "mean",
  ...
)

## S3 method for class 'bamlss'
get_parameters(
  x,
  component = c("all", "conditional", "smooth_terms", "location", "distributional", "auxiliary"),
  parameters = NULL,
  summary = FALSE,
  centrality = "mean",
  ...
)

## S3 method for class 'sim.merMod'
get_parameters(
  x,
  effects = c("fixed", "random", "all"),
  parameters = NULL,
  summary = FALSE,
  centrality = "mean",
  ...
)

## S3 method for class 'sim'
get_parameters(x, parameters = NULL, summary = FALSE, centrality = "mean", ...)

Arguments

- **x**: A fitted model.
- **component**: Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model, the dispersion term, the instrumental variables or marginal effects be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variables (so called fixed-effects regressions), or models with marginal effects from \texttt{mfx}. May be abbreviated. Note that the \textit{conditional} component is also called \textit{count} or \textit{mean} component, depending on the model. There are three convenient shortcuts: \texttt{component = "all"} returns all possible parameters. If \texttt{component = "location"}, location parameters such as conditional, zero_inflated, smooth_terms, or...
instruments are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

**summary** Logical, indicates whether the full posterior samples (summary = FALSE) or the summarized centrality indices of the posterior samples (summary = TRUE) should be returned as estimates.

**centrality** Only for models with posterior samples, and when summary = TRUE. In this case, centrality = "mean" would calculate means of posterior samples for each parameter, while centrality = "median" would use the more robust median value as measure of central tendency.

... Currently not used.

**effects** Should parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.

**iterations** Number of posterior draws.

**progress** Display progress.

**verbose** Toggle messages and warnings.

**parameters** Regular expression pattern that describes the parameters that should be returned.

**Details**

In most cases when models either return different "effects" (fixed, random) or "components" (conditional, zero-inflated, ...), the arguments effects and component can be used.

**Value**

The posterior samples from the requested parameters as data frame. If summary = TRUE, returns a data frame with two columns: the parameter names and the related point estimates (based on centrality).

**BFBayesFactor Models**

Note that for BFBayesFactor models (from the BayesFactor package), posteriors are only extracted from the first numerator model (i.e., model[1]). If you want to apply some function foo() to another model stored in the BFBayesFactor object, index it directly, e.g. foo(model[2]), foo(1/model[5]), etc. See also bayestestR::weighted_posteriors().

**Examples**

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_parameters(m)
```
get_parameters.emmGrid

Get model parameters from estimated marginal means objects

Description

Returns the coefficients from a model.

Usage

## S3 method for class 'emmGrid'
get_parameters(x, summary = FALSE, merge_parameters = FALSE, ...)

## S3 method for class 'emm_list'
get_parameters(x, summary = FALSE, ...)

Arguments

x
A fitted model.

summary
Logical, indicates whether the full posterior samples (summary = FALSE) or
the summarized centrality indices of the posterior samples (summary = TRUE)
should be returned as estimates.

merge_parameters
Logical, if TRUE and x has multiple columns for parameter names (like emmGrid
objects may have), these are merged into a single parameter column, with pa-
rameters names and values as values.

...
Currently not used.

Value

A data frame with two columns: the parameter names and the related point estimates.

Note

Note that emmGrid or emm_list objects returned by functions from emmeans have a different
structure compared to usual regression models. Hence, the Parameter column does not always
contain names of variables, but may rather contain values, e.g. for contrasts. See an example for
pairwise comparisons below.

Examples

data(mtcars)
model <- lm(mpg ~ wt * factor(cyl), data = mtcars)
if (require("emmeans", quietly = TRUE)) {
  emm <- emmeans(model, "cyl")
  get_parameters(emm)
get_parameters.gamm

```r
  emm <- emmeans(model, pairwise ~ cyl)
  get_parameters(emm)
}
```

---

get_parameters.gamm  Get model parameters from generalized additive models

### Description

Returns the coefficients from a model.

### Usage

```r
## S3 method for class 'gamm'
get_parameters(  
  x,  
  component = c("all", "conditional", "smooth_terms", "location"),  
  ...  
)
## S3 method for class 'gam'
get_parameters(  
  x,  
  component = c("all", "conditional", "smooth_terms", "location"),  
  ...  
)
## S3 method for class 'rqss'
get_parameters(x, component = c("all", "conditional", "smooth_terms"), ...)
```

### Arguments

- **x** A fitted model.
- **component** Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the *conditional* component is also called *count* or *mean* component, depending on the model.
- **...** Currently not used.

### Value

For models with smooth terms or zero-inflation component, a data frame with three columns: the parameter names, the related point estimates and the component.
get_parameters.glmm

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_parameters(m)

get_parameters.glmm Get model parameters from mixed models

Description

Returns the coefficients from a model.

Usage

## S3 method for class 'glmm'
get_parameters(x, effects = c("all", "fixed", "random"), ...)

## S3 method for class 'coxme'
get_parameters(x, effects = c("fixed", "random"), ...)

## S3 method for class 'merMod'
get_parameters(x, effects = c("fixed", "random"), ...)

## S3 method for class 'glmmTMB'
get_parameters(
x,
effects = c("fixed", "random"),
component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
...)

## S3 method for class 'glimML'
get_parameters(x, effects = c("fixed", "random", "all"), ...)

Arguments

x A fitted model.
effects Should parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
... Currently not used.
component Which type of parameters to return, such as parameters for the conditional model, the zero-inflated part of the model or the dispersion term? Applies to models with zero-inflated and/or dispersion formula. Note that the conditional component is also called count or mean component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as
conditional or zero_inflated are returned (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma or dispersion (and other auxiliary parameters) are returned.

Details

In most cases when models either return different "effects" (fixed, random) or "components" (conditional, zero-inflated, ...), the arguments effects and component can be used.

Value

If effects = "fixed", a data frame with two columns: the parameter names and the related point estimates. If effects = "random", a list of data frames with the random effects (as returned by ranef()), unless the random effects have the same simplified structure as fixed effects (e.g. for models from MCMCglmm).

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_parameters(m)

get_parameters.htest  Get model parameters from htest-objects

Description

Returns the parameters from a hypothesis test.

Usage

## S3 method for class 'htest'
get_parameters(x, ...)

Arguments

x 
A fitted model.

... 
Currently not used.

Value

A data frame with two columns: the parameter names and the related point estimates.

Examples

get_parameters(t.test(1:10, y = c(7:20)))
get_parameters.zeroinfl

*Get model parameters from zero-inflated and hurdle models*

Description

Returns the coefficients from a model.

Usage

```r
## S3 method for class 'zeroinfl'
get_parameters(
  x,
  component = c("all", "conditional", "zi", "zero_inflated"),
  ...
)

## S3 method for class 'zcp glm'
get_parameters(
  x,
  component = c("all", "conditional", "zi", "zero_inflated"),
  ...
)

## S3 method for class 'mhurdle'
get_parameters(
  x,
  component = c("all", "conditional", "zi", "zero_inflated", "infrequent_purchase",
                "ip", "auxiliary"),
  ...
)
```

Arguments

- `x` A fitted model.
- `component` Should all predictor variables, predictor variables for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated. Note that the `conditional` component is also called `count` or `mean` component, depending on the model.
- `...` Currently not used.

Value

For models with smooth terms or zero-inflation component, a data frame with three columns: the parameter names, the related point estimates and the component.
Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_parameters(m)

get_predicted  Model Predictions (robust)

Description

The `get_predicted()` function is a robust, flexible and user-friendly alternative to base R `predict()` function. Additional features and advantages include availability of uncertainty intervals (CI), bootstrapping, a more intuitive API and the support of more models than base R’s `predict` function. However, although the interface are simplified, it is still very important to read the documentation of the arguments. This is because making "predictions" (a lose term for a variety of things) is a non-trivial process, with lots of caveats and complications. Read the Details section for more information.

Usage

```r
get_predicted(x, ...)

## S3 method for class 'lm'
get_predicted(
x,
data = NULL,
predict = "expectation",
iterations = NULL,
verbose = TRUE,
...)

## S3 method for class 'stanreg'
get_predicted(
x,
data = NULL,
predict = "expectation",
iterations = NULL,
include_random = TRUE,
include_smooth = TRUE,
verbose = TRUE,
...)
```
Arguments

x  A statistical model (can also be a data.frame, in which case the second argument has to be a model).

... Other argument to be passed for instance to get_predicted_ci().

data  An optional data frame in which to look for variables with which to predict. If omitted, the data used to fit the model is used.

predict  Can be "link", "expectation" (default), "prediction", or "response". You can see these 4 options for predictions as on a gradient from "close to the model" to "close to the response data". More specifically, the predict argument modulates two things; the scale of the output as well as the type of certainty interval (see the details and examples). More specifically, "link" returns predictions on the model’s link-scale (for logistic models, that means the log-odds scale) with a confidence interval (CI). "expectation" (default) also returns confidence intervals, but this time the output is on the response scale (for logistic models, that means probabilities). "predict" also gives an output on the response scale, but this time associated with a prediction interval (PI), which is larger than a confidence interval (though it mostly make sense for linear models). Finally, "response" only differs from the previous option for binomial models where it additionally transforms the predictions into the original response’s type (for instance, to a factor). Read more about in the Details section below.

iterations  For Bayesian models, this corresponds to the number of posterior draws. If NULL, will return all the draws (one for each iteration of the model). For frequentist models, if not NULL, will generate bootstrapped draws, from which bootstrapped CIs will be computed. Iterations can be accessed via as.data.frame.

verbose  Toggle warnings.

include_random  If TRUE (default), include all random effects in the prediction. If FALSE, don’t take them into account. Can also be a formula to specify which random effects to condition on when predicting (passed to the re.form argument). If include_random = TRUE and newdata is provided, make sure to include the random effect variables in newdata as well.

include_smooth  For General Additive Models (GAMs). If FALSE, will fix the value of the smooth to its average, so that the predictions are not depending on it. (default), mean(), or bayestestR::map_estimate().

Details

In insight::get_predicted(), the predict argument jointly modulates two separate concepts, the scale and the uncertainty interval.

Confidence Interval (CI) vs. Prediction Interval (PI):

- **Linear models** - lm(): For linear models, Prediction intervals (predict = "prediction") show the range that likely contains the value of a new observation (in what range it is likely to fall), whereas confidence intervals (predict = "expectation" or predict = "link") reflect the uncertainty around the estimated parameters (and gives the range of uncertainty of the regression line). In general, Prediction Intervals (PIs) account for both the uncertainty in the model’s parameters, plus the random variation of the individual values. Thus, prediction...
get_predicted

intervals are always wider than confidence intervals. Moreover, prediction intervals will not necessarily become narrower as the sample size increases (as they do not reflect only the quality of the fit, but also the variability within the data).

- **General Linear models** - `glm()`: For binomial models, prediction intervals are somewhat useless (for instance, for a binomial (bernoulli) model for which the dependent variable is a vector of 1s and 0s, the prediction interval is... [0, 1]).

**Link scale vs. Response scale**: Having the output is on the scale of the response variable is arguably the most convenient to understand and visualize the relationships. If on the link-scale, no transformation is applied and the values are on the scale of the model. For instance, for a logistic model, the response scale corresponds to the predicted probabilities, whereas the link-scale makes predictions of log-odds (probabilities on the logit scale). Note that, when `predict = "response"`, the probabilities are rounded (so that the prediction corresponds to the most likely outcome).

### Value

The fitted values (i.e. predictions for the response). For Bayesian or bootstrapped models (when `iterations != NULL`), iterations (as columns and observations are rows) can be accessed via `as.data.frame`.

### See Also

- `get_predicted_ci()`

### Examples

```r
data(mtcars)
x <- lm(mpg ~ cyl + hp, data = mtcars)
predictions <- get_predicted(x)
predictions

# Options and methods ---------------
get_predicted(x, predict = "prediction")

# Get CI
as.data.frame(predictions)

# Bootstrapped
as.data.frame(get_predicted(x, iterations = 4))
summary(get_predicted(x, iterations = 4)) # Same as as.data.frame(..., keep_iterations = F)

# Different prediction types ------------
data(iris)
data <- droplevels(iris[1:100, ])

# Fit a logistic model
x <- glm(Species ~ Sepal.Length, data = data, family = "binomial")

# Expectation (default): response scale + CI
pred <- get_predicted(x, predict = "expectation")
```
get_predicted_ci

head(as.data.frame(pred))

# Prediction: response scale + PI
pred <- get_predicted(x, predict = "prediction")
head(as.data.frame(pred))

# Link: link scale + CI
pred <- get_predicted(x, predict = "link")
head(as.data.frame(pred))

# Response: response "type" + PI
pred <- get_predicted(x, predict = "response")
head(as.data.frame(pred))

get_predicted_ci  Confidence and Prediction Interval for Model Predictions

Description

Returns the Confidence (or Prediction) Interval (CI) associated with predictions made by a model.

Usage

get_predicted_ci(x, predictions = NULL, ...)

## Default S3 method:
get_predicted_ci(
  x,
  predictions = NULL,
  data = NULL,
  ci = 0.95,
  ci_type = "confidence",
  vcov_estimation = NULL,
  vcov_type = NULL,
  vcov_args = NULL,
  dispersion_method = "sd",
  ci_method = "quantile",
...
)

Arguments

x  A statistical model (can also be a data.frame, in which case the second argument has to be a model).
predictions  A vector of predicted values (as obtained by stats::fitted(), stats::predict() or get_predicted()).
...  Not used for now.
get_predicted_ci

_data_ An optional data frame in which to look for variables with which to predict. If omitted, the data used to fit the model is used.

_ci_ The interval level (default 0.95, i.e., 95% CI).

_ci_type_ Can be "prediction" or "confidence". Prediction intervals show the range that likely contains the value of a new observation (in what range it would fall), whereas confidence intervals reflect the uncertainty around the estimated parameters (and gives the range of the link; for instance of the regression line in a linear regression). Prediction intervals account for both the uncertainty in the model’s parameters, plus the random variation of the individual values. Thus, prediction intervals are always wider than confidence intervals. Moreover, prediction intervals will not necessarily become narrower as the sample size increases (as they do not reflect only the quality of the fit). This applies mostly for "simple" linear models (like \texttt{lm}), as for other models (e.g., \texttt{glm}), prediction intervals are somewhat useless (for instance, for a binomial model for which the dependent variable is a vector of 1s and 0s, the prediction interval is... [0, 1]).

_vcov_estimation_ String, indicating the suffix of the vcov\(^*\)()-function from the \texttt{sandwich} or \texttt{clubSandwich} package, e.g. \texttt{vcov_estimation = "CL"} (which calls \texttt{sandwich::vcovCL()} to compute clustered covariance matrix estimators), or \texttt{vcov_estimation = "HC"} (which calls \texttt{sandwich::vcovHC()} to compute heteroskedasticity-consistent covariance matrix estimators).

_vcov_type_ Character vector, specifying the estimation type for the robust covariance matrix estimation (see \texttt{sandwich::vcovHC()} or \texttt{clubSandwich::vcovCR()} for details).

_vcov_args_ List of named vectors, used as additional arguments that are passed down to the \texttt{sandwich}-function specified in \texttt{vcov_estimation}.

_dispersion_method, ci_method_ These arguments are only used in the context of bootstrapped and Bayesian models. Possible values are \texttt{dispersion_method = c("sd","mad")} and \texttt{ci_method = c("quantile","hdi","eti")}. For the latter, the \texttt{bayestestR} package is required.

**Value**

The Confidence (or Prediction) Interval (CI).

**Examples**

```r
data(mtcars)

# Linear model
# --------
x <- lm(mpg ~ cyl + hp, data = mtcars)
predictions <- predict(x)

ci_vals <- get_predicted_ci(x, predictions, ci_type = "prediction")
head(ci_vals)

ci_vals <- get_predicted_ci(x, predictions, ci_type = "confidence")
head(ci_vals)
```
ci_vals <- get_predicted_ci(x, predictions, ci = c(0.8, 0.9, 0.95))
head(ci_vals)

# Bootstrapped
# ------------
predictions <- get_predicted(x, iterations = 500)
get_predicted_ci(x, predictions)

if (require("bayestestR")) {
  ci_vals <- get_predicted_ci(x, predictions, ci = c(0.80, 0.95))
  head(ci_vals)
  bayestestR::reshape_ci(ci_vals)
  ci_vals <- get_predicted_ci(x, predictions, dispersion_method = "MAD",
                              ci_method = "HDI")
  head(ci_vals)
}

# Logistic model
# --------------
x <- glm(vs ~ wt, data = mtcars, family = "binomial")
predictions <- predict(x, type = "link")

# Logistic model
# --------------
x <- glm(vs ~ wt, data = mtcars, family = "binomial")
predictions <- predict(x, type = "link")

get_predictors

---

**get_predictors**

Get the data from model predictors

**Description**

Returns the data from all predictor variables (fixed effects).

**Usage**

`get_predictors(x, verbose = TRUE)`

**Arguments**

- `x` A fitted model.
- `verbose` Toggle messages and warnings.

**Value**

The data from all predictor variables, as data frame.
get_priors

Examples

```r
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
head(get_predictors(m))
```

get_priors

*Get summary of priors used for a model*

Description

Provides a summary of the prior distributions used for the parameters in a given model.

Usage

```r
get_priors(x, ...)
```

## S3 method for class 'brmsfit'

```r
get_priors(x, verbose = TRUE, ...)
```

Arguments

- `x`  A Bayesian model.
- `...`  Currently not used.
- `verbose`  Toggle warnings and messages.

Value

A data frame with a summary of the prior distributions used for the parameters in a given model.

Examples

```r
## Not run:
library(rstanarm)
model <- stan_glm(Sepal.Width ~ Species * Petal.Length, data = iris)
get_priors(model)
## End(Not run)
```
**get_random**

*Get the data from random effects*

**Description**

Returns the data from all random effects terms.

**Usage**

```r
get_random(x)
```

**Arguments**

- `x`: A fitted mixed model.

**Value**

The data from all random effects terms, as data frame. Or NULL if model has no random effects.

**Examples**

```r
if (require("lme4")) {
  data(sleepstudy)
  # prepare some data...
  sleepstudy$mygrp <- sample(1:5, size = 180, replace = TRUE)
  sleepstudy$mysubgrp <- NA
  for (i in 1:5) {
    filter_group <- sleepstudy$mygrp == i
    sleepstudy$mysubgrp[filter_group] <-
      sample(1:30, size = sum(filter_group), replace = TRUE)
  }
  m <- lmer(
    Reaction ~ Days + (1 | mygrp / mysubgrp) + (1 | Subject),
    data = sleepstudy
  )
  head(get_random(m))
}
```
get_residuals

Extract model residuals

Description

Returns the residuals from regression models.

Usage

get_residuals(x, ...)

## Default S3 method:
get_residuals(x, weighted = FALSE, verbose = TRUE, ...)

Arguments

x
A model.

... Passed down to residuals(), if possible.

weighted Logical, if TRUE, returns weighted residuals.

verbose Toggle warnings and messages.

Value

The residuals, or NULL if this information could not be accessed.

Note

This function returns the default type of residuals, i.e. for the response from linear models, the deviance residuals for models of class glm etc. To access different types, pass down the type argument (see 'Examples').

This function is a robust alternative to residuals(), as it works for some special model objects that otherwise do not respond properly to calling residuals().

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_residuals(m)

m <- glm(vs ~ wt + cyl + mpg, data = mtcars, family = binomial())
get_residuals(m) # type = "deviance" by default
get_residuals(m, type = "response")
Description

Returns the values the response variable(s) from a model object. If the model is a multivariate response model, a data frame with values from all response variables is returned.

Usage

get_response(x, select = NULL, verbose = TRUE)

Arguments

x
A fitted model.

select
Optional name(s) of response variables for which to extract values. Can be used in case of regression models with multiple response variables.

verbose
Toggle warnings.

Value

The values of the response variable, as vector, or a data frame if x has more than one defined response variable.

Examples

```r
if (require("lme4")) {
  data(cbpp)
  cbpp$trials <- cbpp$size - cbpp$incidence

  m <- glm(cbind(incidence, trials) ~ period, data = cbpp, family = binomial)
  head(get_response(m))
  get_response(m, select = "incidence")
}

data(mtcars)
  m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
  get_response(m)
```
get_sigma

Get residual standard deviation from models

Description

Returns sigma, which corresponds the estimated standard deviation of the residuals. This function extends the sigma() base R generic for models that don’t have implemented it. It also computes the confidence interval (CI), which is stored as an attribute.

Sigma is a key-component of regression models, and part of the so-called auxiliary parameters that are estimated. Indeed, linear models for instance assume that the residuals comes from a normal distribution with mean 0 and standard deviation sigma. See the details section below for more information about its interpretation and calculation.

Usage

get_sigma(x, ci = NULL, verbose = TRUE)

Arguments

x A model.

ci Scalar, the CI level. The default (NULL) returns no CI.

verbose Toggle messages and warnings.

Details

Interpretation of Sigma: The residual standard deviation, σ, indicates that the predicted outcome will be within +/- σ units of the linear predictor for approximately 68% of the data points (Gelman, Hill & Vehtari 2020, p.84). In other words, the residual standard deviation indicates the accuracy for a model to predict scores, thus it can be thought of as “a measure of the average distance each observation falls from its prediction from the model” (Gelman, Hill & Vehtari 2020, p.168). σ can be considered as a measure of the unexplained variation in the data, or of the precision of inferences about regression coefficients.

Calculation of Sigma: By default, get_sigma() tries to extract sigma by calling stats::sigma(). If the model-object has no sigma() method, the next step is calculating sigma as square-root of the model-deviance divided by the residual degrees of freedom. Finally, if even this approach fails, and x is a mixed model, the residual standard deviation is accessed using the square-root from get_variance_residual().

Value

The residual standard deviation (sigma), or NULL if this information could not be accessed.

References

get_statistic

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_sigma(m)

get_statistic

Get statistic associated with estimates

Description

Returns the statistic (t, z, ...) for model estimates. In most cases, this is the related column from coef(summary()).

Usage

get_statistic(x, ...)

## Default S3 method:
get_statistic(x, column_index = 3, verbose = TRUE, ...)

## S3 method for class 'glmmTMB'
get_statistic(
x,
column = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
...
)

## S3 method for class 'clm2'
get_statistic(x, component = c("all", "conditional", "scale"), ...)

## S3 method for class 'betamfx'
get_statistic(
x,
component = c("all", "conditional", "precision", "marginal"),
...
)

## S3 method for class 'logitmfx'
get_statistic(x, component = c("all", "conditional", "marginal"), ...)

## S3 method for class 'mjoint'
get_statistic(x, component = c("all", "conditional", "survival"), ...)

## S3 method for class 'emmmGrid'
get_statistic(x, ci = 0.95, adjust = "none", merge_parameters = FALSE, ...)

## S3 method for class 'gee'
get_statistic(x, robust = FALSE, ...)

## S3 method for class 'betareg'
get_statistic(x, component = c("all", "conditional", "precision"), ...)

## S3 method for class 'DirichletRegModel'
get_statistic(x, component = c("all", "conditional", "precision"), ...)

Arguments

x A model.

... Currently not used.

column_index For model objects that have no defined get_statistic() method yet, the default method is called. This method tries to extract the statistic column from coef(summary()), where the index of the column that is being pulled is column_index. Defaults to 3, which is the default statistic column for most models’ summary-output.

verbose Toggle messages and warnings.

component Should all parameters, parameters for the conditional model, or for the zero-inflated part of the model be returned? Applies to models with zero-inflated component. component may be one of "conditional", "zi", "zero-inflated" or "all" (default). For models with smooth terms, component = "smooth_terms" is also possible. May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model.

ci Confidence Interval (CI) level. Default to 0.95 (95%). Currently only applies to objects of class emmGrid.

adjust Character value naming the method used to adjust p-values or confidence intervals. See ?emmeans::summary.emmGrid for details.

merge_parameters Logical, if TRUE and x has multiple columns for parameter names (like emmGrid objects may have), these are merged into a single parameter column, with parameters names and values as values.

robust Logical, if TRUE, test statistic based on robust standard errors is returned.

Value

A data frame with the model’s parameter names and the related test statistic.

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_statistic(m)
get_varcov

Get variance-covariance matrix from models

Description

Returns the variance-covariance, as retrieved by stats::vcov(), but works for more model objects that probably don’t provide a vcov()-method.

Usage

get_varcov(x, ...)

## Default S3 method:
get_varcov(x, verbose = TRUE, ...)

## S3 method for class 'betareg'
get_varcov(
  x,
  component = c("conditional", "precision", "all"),
  verbose = TRUE,
  ...
)

## S3 method for class 'DirichletRegModel'
get_varcov(
  x,
  component = c("conditional", "precision", "all"),
  verbose = TRUE,
  ...
)

## S3 method for class 'clm2'
get_varcov(x, component = c("all", "conditional", "scale"), ...)

## S3 method for class 'truncreg'
get_varcov(x, component = c("conditional", "all"), ...)

## S3 method for class 'gamlss'
get_varcov(x, component = c("conditional", "all"), ...)

## S3 method for class 'hurdle'
get_varcov(x, component = c("conditional", "zero_inflated", "zi", "all"), ...)

## S3 method for class 'zcpglm'
get_varcov(x, component = c("conditional", "zero_inflated", "zi", "all"), ...)

## S3 method for class 'MixMod'
get_varcov(x, component = c("conditional", "zero_inflated", "zi", "all"), ...)

## S3 method for class 'glmmTMB'
get_varcov(
  x,
  component = c("conditional", "zero_inflated", "zi", "dispersion", "all"),
  ...
)

## S3 method for class 'brmsfit'
get_varcov(x, component = "conditional", ...)

## S3 method for class 'betamfx'
get_varcov(x, component = c("conditional", "precision", "all"), ...)

## S3 method for class 'aov'
get_varcov(x, complete = FALSE, ...)

## S3 method for class 'mixor'
get_varcov(x, effects = c("all", "fixed", "random"), ...)

Arguments

x A model.

... Currently not used.

verbose Toggle warnings.

component Should the complete variance-covariance matrix of the model be returned, or only for specific model components only (like count or zero-inflated model parts)? Applies to models with zero-inflated component, or models with precision (e.g. betareg) component. component may be one of "conditional", "zi", "zero-inflated", "dispersion", "precision", or "all". May be abbreviated. Note that the conditional component is also called count or mean component, depending on the model.

cOMPLETE Logical, if TRUE, for aov, returns the full variance-covariance matrix.

effects Should the complete variance-covariance matrix of the model be returned, or only for specific model parameters only? Currently only applies to models of class mixor.

Value

The variance-covariance matrix, as matrix-object.

Note

get_varcov() tries to return the nearest positive definite matrix in case of a negative variance-covariance matrix.
get_variance

Examples

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
get_varcov(m)
```

get_variance

Get variance components from random effects models

Description

This function extracts the different variance components of a mixed model and returns the result as list. Functions like `get_variance_residual(x)` or `get_variance_fixed(x)` are shortcuts for `get_variance(x, component = "residual")` etc.

Usage

```r
get_variance(
  x,
  component = c("all", "fixed", "random", "residual", "distribution", "dispersion",
                  "intercept", "slope", "rho01", "rho00"),
  verbose = TRUE,
  ...
)
get_variance_residual(x, verbose = TRUE, ...)
get_variance_fixed(x, verbose = TRUE, ...)
get_variance_random(x, verbose = TRUE, tolerance = 1e-05, ...)
get_variance_distribution(x, verbose = TRUE, ...)
get_variance_dispersion(x, verbose = TRUE, ...)
get_variance_intercept(x, verbose = TRUE, ...)
get_variance_slope(x, verbose = TRUE, ...)
get_correlation_slope_intercept(x, verbose = TRUE, ...)
get_correlation_slopes(x, verbose = TRUE, ...)
```

Arguments

- **x**: A mixed effects model.
component Character value, indicating the variance component that should be returned. By default, all variance components are returned. The distribution-specific ("distribution") and residual ("residual") variance are the most computational intensive components, and hence may take a few seconds to calculate.

verbose Toggle off warnings.
... Currently not used.
tolerance Tolerance for singularity check of random effects, to decide whether to compute random effect variances or not. Indicates up to which value the convergence result is accepted. The larger tolerance is, the stricter the test will be. See `performance::check_singularity()`.

Details
This function returns different variance components from mixed models, which are needed, for instance, to calculate r-squared measures or the intraclass-correlation coefficient (ICC).

**Fixed effects variance:** The fixed effects variance, $\sigma_f^2$, is the variance of the matrix-multiplication $\beta \times X$ (parameter vector by model matrix).

**Random effects variance:** The random effect variance, $\sigma_i^2$, represents the mean random effect variance of the model. Since this variance reflect the "average" random effects variance for mixed models, it is also appropriate for models with more complex random effects structures, like random slopes or nested random effects. Details can be found in Johnson 2014, in particular equation 10. For simple random-intercept models, the random effects variance equals the random-intercept variance.

**Distribution-specific variance:** The distribution-specific variance, $\sigma_d^2$, depends on the model family. For Gaussian models, it is $\sigma^2$ (i.e. `sigma(model)^2`). For models with binary outcome, it is $\pi^2/3$ for logit-link, 1 for probit-link, and $\pi^2/6$ for cloglog-links. Models from Gamma-families use $\mu^2$ (as obtained from `family$variance()`). For all other models, the distribution-specific variance is based on lognormal approximation, $\log(1 + \text{var}(x)/\mu^2)$ (see Nakagawa et al. 2017). The expected variance of a zero-inflated model is computed according to Zuur et al. 2012, p277.

**Variance for the additive overdispersion term:** The variance for the additive overdispersion term, $\sigma_e^2$, represents "the excess variation relative to what is expected from a certain distribution" (Nakagawa et al. 2017). In (most? many?) cases, this will be 0.

**Residual variance:** The residual variance, $\sigma_e^2$, is simply $\sigma_d^2 + \sigma_e^2$.

**Random intercept variance:** The random intercept variance, or between-subject variance ($\tau_{00}$), is obtained from `VarCorr()`. It indicates how much groups or subjects differ from each other, while the residual variance $\sigma_e^2$ indicates the within-subject variance.

**Random slope variance:** The random slope variance ($\tau_{11}$) is obtained from `VarCorr()`. This measure is only available for mixed models with random slopes.

**Random slope-intercept correlation:** The random slope-intercept correlation ($\rho_{01}$) is obtained from `VarCorr()`. This measure is only available for mixed models with random intercepts and slopes.
get_variance

Value

A list with following elements:

• `var.fixed`, variance attributable to the fixed effects
• `var.random`, (mean) variance of random effects
• `var.residual`, residual variance (sum of dispersion and distribution)
• `var.distribution`, distribution-specific variance
• `var.dispersion`, variance due to additive dispersion
• `var.intercept`, the random-intercept-variance, or between-subject-variance ($\tau_{00}$)
• `var.slope`, the random-slope-variance ($\tau_{11}$)
• `cor.slope_intercept`, the random-slope-intercept-correlation ($\rho_{01}$)
• `cor.slopes`, the correlation between random slopes ($\rho_{00}$)

Note

This function supports models of class `merMod` (including models from `blme`, `clmm`, `cpglmm`, `glmmadmb`, `glmmTMB`, `MixMod`, `lme`, `mixed`, `rlmerMod`, `stanreg`, `brmsfit` or `wbm`). Support for objects of class `MixMod` (`GLMMadaptive`), `lme` (`nlme`) or `brmsfit` (`brms`) is experimental and may not work for all models.

References


Examples

```r
## Not run:
library(lme4)
data(sleepstudy)
m <- lmer(Reaction ~ Days + (1 + Days | Subject), data = sleepstudy)
get_variance(m)
get_variance_fixed(m)
get_variance_residual(m)
## End(Not run)
```
get_weights

Description

Returns weighting variable of a model.

Usage

get_weights(x, ...)

## Default S3 method:
get_weights(x, na_rm = FALSE, null_as_ones = FALSE, ...)

Arguments

x
  A fitted model.

...  
  Currently not used.

na_rm  
  Logical, if TRUE, removes possible missing values.

null_as_ones  
  Logical, if TRUE, will return a vector of 1 if no weights were specified in the
  model (as if the weights were all set to 1).

Value

The weighting variable, or NULL if no weights were specified or if weights were 1. If the weighting
variable should also be returned (instead of NULL), when all weights are set to 1 (i.e. no weighting),
set null_as_ones = TRUE.

Examples

```r
  data(mtcars)
  mt$cars$weight <- rnorm(nrow(mtcars), 1, .3)

  # LMs
  m <- lm(mpg ~ wt + cyl + vs, data = mtcars, weights = weight)
  get_weights(m)

  get_weights(lm(mpg ~ wt, data = mtcars), null_as_ones = TRUE)

  # GLMs
  m <- glm(vs ~ disp + mpg, data = mtcars, weights = weight, family = quasibinomial)
  get_weights(m)
  m <- glm(cbind(cyl, gear) ~ mpg, data = mtcars, weights = weight, family = binomial)
  get_weights(m)
```
**Description**

Checks if model has an intercept.

**Usage**

```r
has_intercept(x, verbose = TRUE)
```

**Arguments**

- `x` A model object.
- `verbose` Toggle warnings.

**Value**

TRUE if `x` has an intercept, FALSE otherwise.

**Examples**

```r
model <- lm(mpg ~ 0 + gear, data = mtcars)
has_intercept(model)

model <- lm(mpg ~ gear, data = mtcars)
has_intercept(model)

if (require("lme4")) {
  model <- lmer(Reaction ~ 0 + Days + (Days | Subject), data = sleepstudy)
  has_intercept(model)

  model <- lmer(Reaction ~ Days + (Days | Subject), data = sleepstudy)
  has_intercept(model)
}
```

---

**is_gam_model**

*Checks if a model is a generalized additive model*

**Description**

Small helper that checks if a model is a generalized additive model.

**Usage**

```r
is_gam_model(x)
```
is_mixed_model

Arguments

x  A model object.

Value

A logical, TRUE if x is a mixed model.

Note

This function only returns TRUE when the model inherits from a typical GAM model class and when smooth terms are present in the model formula. If model has no smooth terms or is not from a typical gam class, FALSE is returned.

Examples

```r
if (require("mgcv")) {
  data(iris)
  model1 <- lm(Petal.Length ~ Petal.Width + Sepal.Length, data = iris)
  model2 <- gam(Petal.Length ~ Petal.Width + s(Sepal.Length), data = iris)
  is_gam_model(model1)
  is_gam_model(model2)
}
```

Description

Small helper that checks if a model is a mixed effects model, i.e. if it the model has random effects.

Usage

```r
is_mixed_model(x)
```
is_model

Examples

data(mtcars)
model <- lm(mpg ~ wt + cyl + vs, data = mtcars)
is_model(model)

if (require("lme4")) {
  data(sleepstudy)
  model <- lmer(Reaction ~ Days + (1 | Subject), data = sleepstudy)
  is_model(model)
}

is_model

Checks if an object is a regression model or statistical test object

Description

Small helper that checks if a model is a regression model or a statistical object. is_regression_model() is stricter and only returns TRUE for regression models, but not for, e.g., ttest objects.

Usage

is_model(x)

is_regression_model(x)

Arguments

x An object.

Details

This function returns TRUE if x is a model object.

Value

A logical, TRUE if x is a (supported) model object.

Examples

data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)

is_model(m)
is_model(mtcars)

test <- t.test(1:10, y = c(7:20))
is_model(test)
is_regression_model(test)
is_model_supported  Checks if an object is a regression model object supported in *insight* package.

**Description**
Small helper that checks if a model is a supported (regression) model object. `supported_models()` prints a list of currently supported model classes.

**Usage**

```r
is_model_supported(x)

supported_models()
```

**Arguments**

- `x`  
  An object.

**Details**
This function returns `TRUE` if `x` is a model object that works with the package’s functions. A list of supported models can also be found here: [https://github.com/easystats/insight](https://github.com/easystats/insight).

**Value**
A logical, `TRUE` if `x` is a (supported) model object.

**Examples**

```r
data(mtcars)
m <- lm(mpg ~ wt + cyl + vs, data = mtcars)

is_model_supported(m)
is_model_supported(mtcars)
```

---

is_multivariate  Checks if an object stems from a multivariate response model

**Description**
Small helper that checks if a model is a multivariate response model, i.e. a model with multiple outcomes.

**Usage**

```r
is_multivariate(x)
```
is_nested_models

Arguments
x
   A model object, or an object returned by a function from this package.

Value
A logical, TRUE if either x is a model object and is a multivariate response model, or TRUE if a return value from a function of insight is from a multivariate response model.

Examples
## Not run:
library(rstanarm)
data("pbcLong")
model <- stan_mvmer(
   formula = list(
     logBili ~ year + (1 | id),
     albumin ~ sex + year + (year | id)
   ),
   data = pbcLong,
   chains = 1, cores = 1, seed = 12345, iter = 1000
)

f <- find_formula(model)
is_multivariate(model)
is_multivariate(f)

## End(Not run)

is_nested_models Checks whether a list of models are nested models

Description
Checks whether a list of models are nested models, strictly following the order they were passed to the function.

Usage
is_nested_models(...) 

Arguments
... Multiple regression model objects.

Value
TRUE if models are nested, FALSE otherwise. If models are nested, also returns two attributes that indicate whether nesting of models is in decreasing or increasing order.
Examples

m1 <- lm(Sepal.Length ~ Petal.Width + Species, data = iris)
m2 <- lm(Sepal.Length ~ Species, data = iris)
m3 <- lm(Sepal.Length ~ Petal.Width, data = iris)
m4 <- lm(Sepal.Length ~ 1, data = iris)

is_nested_models(m1, m2, m4)
is_nested_models(m4, m2, m1)
is_nested_models(m1, m2, m3)

---

**is_nullmodel**

*Checks if model is a null-model (intercept-only)*

Description

Checks if model is a null-model (intercept-only), i.e. if the conditional part of the model has no predictors.

Usage

`is_nullmodel(x)`

Arguments

`x`  
A model object.

Value

`TRUE` if `x` is a null-model, `FALSE` otherwise.

Examples

model <- lm(mpg ~ 1, data = mtcars)
is_nullmodel(model)

model <- lm(mpg ~ gear, data = mtcars)
is_nullmodel(model)

if (require("lme4")) {
  model <- lmer(Reaction ~ 1 + (Days | Subject), data = sleepstudy)
  is_nullmodel(model)

  model <- lmer(Reaction ~ Days + (Days | Subject), data = sleepstudy)
  is_nullmodel(model)
}
**link_function**

Get link-function from model object

---

**Description**

Returns the link-function from a model object.

**Usage**

```r
link_function(x, ...)
```

### S3 method for class 'betamfx'

```r
link_function(x, what = c("mean", "precision"), ...)
```

### S3 method for class 'gamlss'

```r
link_function(x, what = c("mu", "sigma", "nu", "tau"), ...)
```

### S3 method for class 'betareg'

```r
link_function(x, what = c("mean", "precision"), ...)
```

### S3 method for class 'DirichletRegModel'

```r
link_function(x, what = c("mean", "precision"), ...)
```

**Arguments**

- **x**
  - A fitted model.
- **...**
  - Currently not used.
- **what**
  - For gamlss models, indicates for which distribution parameter the link (inverse) function should be returned; for betareg or DirichletRegModel, can be "mean" or "precision".

**Value**

A function, describing the link-function from a model-object. For multivariate-response models, a list of functions is returned.

**Examples**

```r
# example from ?stats::glm
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
m <- glm(counts ~ outcome + treatment, family = poisson())
link_function(m)(.3)
# same as
log(.3)
```
link_inverse

Get link-inverse function from model object

Description

Returns the link-inverse function from a model object.

Usage

link_inverse(x, ...)

## S3 method for class 'betareg'
link_inverse(x, what = c("mean", "precision"), ...)

## S3 method for class 'DirichletRegModel'
link_inverse(x, what = c("mean", "precision"), ...)

## S3 method for class 'betamfx'
link_inverse(x, what = c("mean", "precision"), ...)

## S3 method for class 'gamlss'
link_inverse(x, what = c("mu", "sigma", "nu", "tau"), ...)

Arguments

x A fitted model.
...
Currently not used.
what For gamlss models, indicates for which distribution parameter the link (inverse) function should be returned; for betareg or DirichletRegModel, can be "mean" or "precision".

Value

A function, describing the inverse-link function from a model-object. For multivariate-response models, a list of functions is returned.

Examples

# example from ?stats::glm
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
m <- glm(counts ~ outcome + treatment, family = poisson())
link_inverse(m)(.3)
# same as
exp(.3)
Description

Retrieve information from model objects.

Usage

```r
model_info(x, ...)  
## Default S3 method:  
model_info(x, verbose = TRUE, ...)
```

Arguments

- `x` A fitted model.
- `...` Currently not used.
- `verbose` Toggle off warnings.

Details

`model_info()` returns a list with information about the model for many different model objects. Following information is returned, where all values starting with `is_` are logicals.

- `is_binomial`: family is binomial (but not negative binomial)
  `is_bernoulli`: special case of binomial models: family is Bernoulli
  `is_poisson`: family is poisson
  `is_negbin`: family is negative binomial
  `is_count`: model is a count model (i.e. family is either poisson or negative binomial)
  `is_beta`: family is beta
  `is_betabinomial`: family is beta-binomial
  `is_dirichlet`: family is dirichlet
  `is_exponential`: family is exponential (e.g. Gamma or Weibull)
  `is_logit`: model has logit link
\item `is_probit`: model has probit link
\item `is_linear`: family is gaussian
\item `is_tweedie`: family is tweedie
\item `is_ordinal`: family is ordinal or cumulative link
\item `is_cumulative`: family is ordinal or cumulative link
\item `is_multinomial`: family is multinomial or categorical link
\item `is_categorical`: family is categorical link
\item `is_censored`: model is a censored model (has a censored response, including survival models)
\item `is_truncated`: model is a truncated model (has a truncated response)
\item `is_survival`: model is a survival model
\item `is_zero_inflated`: model has zero-inflation component
\item `is_hurdle`: model has zero-inflation component and is a hurdle-model (truncated family distribution)
\item `is_dispersion`: model has dispersion component
\item `is_mixed`: model is a mixed effects model (with random effects)
\item `is_multivariate`: model is a multivariate response model (currently only works for *brmsfit* objects)
\item `is_trial`: model response contains additional information about the trials
\item `is_bayesian`: model is a Bayesian model
\item `is_gam`: model is a generalized additive model
\item `is_anova`: model is an Anova object
\item `is_ttest`: model is an an object of class `htest`, returned by `t.test()`
\item `is_correlation`: model is an an object of class
`htest`, returned by `cor.test()`

- **is_ranktest**: model is an an object of class `htest`, returned by `cor.test()` (if Spearman’s rank correlation), `wilcox.test()` or `kruskal.test()`.

- **is_levenetest**: model is an an object of class `anova`, returned by `car::leveneTest()`.

- **is_onewaytest**: model is an an object of class `htest`, returned by `oneway.test()`

- **is_proptest**: model is an an object of class `htest`, returned by `prop.test()`

- **is_binomtest**: model is an an object of class `htest`, returned by `binom.test()`

- **is_chi2test**: model is an an object of class `htest`, returned by `chisq.test()`

- **is_xtab**: model is an an object of class `htest` or `BFBayesFactor`, and test-statistic stems from a contingency table (i.e. `chisq.test()` or `BayesFactor::contingencyTableBF()`).

- **link_function**: the link-function

- **family**: the family-object

- **n_obs**: number of observations

- **model_terms**: a list with all model terms, including terms such as random effects or from zero-inflated model parts.

### Value

A list with information about the model, like family, link-function etc. (see ‘Details’).

### Examples

```r
ldose <- rep(0:5, 2)
numdead <- c(1, 4, 9, 13, 18, 12, 0, 2, 6, 10, 12, 16)
sex <- factor(rep(c("M", "F"), c(6, 6)))
SF <- cbind(numdead, numalive = 20 - numdead)
dat <- data.frame(ldose, sex, SF, stringsAsFactors = FALSE)
m <- glm(SF ~ sex * ldose, family = binomial)
model_info(m)
```

---

```r
# Not run:
library(glmmTMB)
```
```r
data("Salamanders")
m <- glmmTMB(
  count ~ spp + cover + mined + (1 | site),
  ziformula = ~ spp + mined,
  dispformula = ~DOY,
  data = Salamanders,
  family = nbinom2
)
```

```r
## End(Not run)
model_info(m)
```

---

### model_name

#### Name the model

**Description**

Returns the "name" (class attribute) of a model, possibly including further information.

**Usage**

```r
model_name(x, ...)  
```

**Arguments**

- `x` A model.
- `...` Currently not used.
- `include_formula` Should the name include the model’s formula.
- `include_call` If TRUE, will return the function call as a name.

**Value**

A character string of a name (which usually equals the model’s class attribute).

**Examples**

```r
m <- lm(Sepal.Length ~ Petal.Width, data = iris)
model_name(m)
model_name(m, include_formula = TRUE)
model_name(m, include_call = TRUE)

if (require("lme4")) {
  model_name(lmer(Sepal.Length ~ Sepal.Width + (1 | Species), data = iris))
} 
```
null_model

Compute intercept-only model for regression models

Description

This function computes the null-model (i.e. \(y \sim 1\)) of a model. For mixed models, the null-model takes random effects into account.

Usage

null_model(model, verbose = TRUE, ...)

Arguments

model A (mixed effects) model.
verbose Toggle off warnings.
... Arguments passed to or from other methods.

Value

The null-model of \(x\)

Examples

if (require("lme4")) {
  data(sleepstudy)
  m <- lmer(Reaction ~ Days + (1 + Days | Subject), data = sleepstudy)
  summary(m)
  summary(null_model(m))
}

n_obs

Get number of observations from a model

Description

This method returns the number of observation that were used to fit the model, as numeric value.
Usage

n_obs(x, ...)

## S3 method for class 'svyolr'

n_obs(x, weighted = FALSE, ...)

## S3 method for class 'afex_aov'

n_obs(x, shape = c("long", "wide"), ...)

## S3 method for class 'stanmvreg'

n_obs(x, select = NULL, ...)

Arguments

x
A fitted model.

... Currently not used.

weighted For survey designs, returns the weighted sample size.

shape Return long or wide data? Only applicable in repeated measures designs.

select Optional name(s) of response variables for which to extract values. Can be used in case of regression models with multiple response variables.

Value

The number of observations used to fit the model, or NULL if this information is not available.

Examples

data(mtcars)

m <- lm(mpg ~ wt + cyl + vs, data = mtcars)
n_obs(m)

n_parameters
Count number of parameters in a model

Description

Returns the number of parameters (coefficients) of a model.

Usage

n_parameters(x, ...)

## Default S3 method:

n_parameters(x, remove_nonestimable = FALSE, ...)

## S3 method for class 'merMod'
n_parameters(
  x,
  effects = c("fixed", "random"),
  remove_nonestimable = FALSE,
  ...
)

## S3 method for class 'glmmTMB'
n_parameters(
  x,
  effects = c("fixed", "random"),
  component = c("all", "conditional", "zi", "zero_inflated"),
  remove_nonestimable = FALSE,
  ...
)

## S3 method for class 'zeroinfl'
n_parameters(
  x,
  component = c("all", "conditional", "zi", "zero_inflated"),
  remove_nonestimable = FALSE,
  ...
)

## S3 method for class 'gam'
n_parameters(
  x,
  component = c("all", "conditional", "smooth_terms"),
  remove_nonestimable = FALSE,
  ...
)

## S3 method for class 'brmsfit'
n_parameters(x, effects = "all", component = "all", ...)

Arguments

x A statistical model.
...
Arguments passed to or from other methods.
remove_nonestimable Logical, if TRUE, removes (i.e. does not count) non-estimable parameters (which may occur for models with rank-deficient model matrix).
effects Should number of parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
component Should total number of parameters, number parameters for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion for-
mula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated.

Value

The number of parameters in the model.

Note

This function returns the number of parameters for the fixed effects by default, as returned by `find_parameters(x, effects = "fixed")`. It does not include all estimated model parameters, i.e. auxiliary parameters like sigma or dispersion are not counted. To get the number of all estimated parameters, use `get_df(x, type = "model")`.

Examples

```r
data(iris)
model <- lm(Sepal.Length ~ Sepal.Width * Species, data = iris)
n_parameters(model)
```

**print_color**  
*Coloured console output*

Description

Convenient function that allows coloured output in the console. Mainly implemented to reduce package dependencies.

Usage

```r
print_color(text, color)
print_colour(text, colour)
color_text(text, color)
colour_text(text, colour)
color_theme()
```

Arguments

- **text**: The text to print.
- **color, colour**: Character vector, indicating the colour for printing. May be one of "red", "yellow", "green", "blue", "violet", "cyan" or "grey". Formatting is also possible with "bold" or "italic".
Details

This function prints text directly to the console using `cat()`, so no string is returned. `color_text()`, however, returns only the formatted string, without using `cat()`. `color_theme()` either returns "dark" when RStudio is used with dark color scheme, "light" when it's used with light theme, and `NULL` if the theme could not be detected.

Value

Nothing.

Examples

```r
print_color("I'm blue dabedí dabedéí", "blue")
```

Description

This function takes a data frame, typically a data frame with information on summaries of model parameters like `bayestestR::describe_posterior()`, `bayestestR::hdi()` or `parameters::model_parameters()`, as input and splits this information into several parts, depending on the model. See details below.

Usage

```r
print_parameters(
  x,
  ...,
  split_by = c("Effects", "Component", "Group", "Response"),
  format = "text",
  parameter_column = "Parameter",
  keep_parameter_column = TRUE,
  remove_empty_column = FALSE,
  titles = NULL,
  subtitles = NULL
)
```

Arguments

- **x**: A fitted model, or a data frame returned by `clean_parameters()`.
- **...**: One or more objects (data frames), which contain information about the model parameters and related statistics (like confidence intervals, HDI, ROPE, ...).
- **split_by**: `split_by` should be a character vector with one or more of the following elements: "Effects", "Component", "Response" and "Group". These are the column names returned by `clean_parameters()`, which is used to extract the information from which the group or component model parameters belong. If `NULL`, the merged data frame is returned. Else, the data frame is split into a list, split by the values from those columns defined in `split_by`. 
print_parameters

format
Name of output-format, as string. If NULL (or "text"), assumed use for output is basic printing. If "markdown", markdown-format is assumed. This only affects the style of title- and table-caption attributes, which are used in export_table().

parameter_column
String, name of the column that contains the parameter names. Usually, for data frames returned by functions the easystats-packages, this will be "Parameter".

keep_parameter_column
Logical, if TRUE, the data frames in the returned list have both a "Cleaned_Parameter" and "Parameter" column. If FALSE, the (unformatted) "Parameter" is removed, and the column with cleaned parameter names ("Cleaned_Parameter") is renamed into "Parameter".

remove_empty_column
Logical, if TRUE, columns with completely empty character values will be removed.

titles, subtitles
By default, the names of the model components (like fixed or random effects, count or zero-inflated model part) are added as attributes "table_title" and "table_subtitle" to each list element returned by print_parameters(). These attributes are then extracted and used as table (sub) titles in export_table(). Use titles and subtitles to override the default attribute values for "table_title" and "table_subtitle". titles and subtitles may be any length from 1 to same length as returned list elements. If titles and subtitles are shorter than existing elements, only the first default attributes are overwritten.

Details

This function prepares data frames that contain information about model parameters for clear printing.

First, x is required, which should either be a model object or a prepared data frame as returned by clean_parameters(). If x is a model, clean_parameters() is called on that model object to get information with which model components the parameters are associated.

Then, ... take one or more data frames that also contain information about parameters from the same model, but also have additional information provided by other methods. For instance, a data frame in ... might be the result of, for instance, bayestestR::describe_posterior(), or parameters::model_parameters(), where we have a) a Parameter column and b) columns with other parameter values (like CI, HDI, test statistic, etc.).

Now we have a data frame with model parameters and information about the association to the different model components, a data frame with model parameters, and some summary statistics. print_parameters() then merges these data frames, so the parameters or statistics of interest are also associated with the different model components. The data frame is split into a list, so for a clear printing. Users can loop over this list and print each component for a better overview. Further, parameter names are "cleaned", if necessary, also for a cleaner print. See also 'Examples'.
**Value**

A data frame or a list of data frames (if `split_by` is not `NULL`). If a list is returned, the element names reflect the model components where the extracted information in the data frames belong to, e.g. `random.zero_inflated.Intercept: persons`. This is the data frame that contains the parameters for the random effects from group-level "persons" from the zero-inflated model component.

**Examples**

```r
## Not run:
library(bayestestR)
model <- download_model("brms_zi_2")
x <- hdi(model, effects = "all", component = "all")

# hdi() returns a data frame; here we use only the 
# information on parameter names and HDI values
tmp <- as.data.frame(x)[, 1:4]
tmp

# Based on the "split_by" argument, we get a list of data frames that 
# is split into several parts that reflect the model components.
print_parameters(model, tmp)

# This is the standard print()-method for "bayestestR::hdi"-objects. 
# For printing methods, it is easy to print complex summary statistics 
# in a clean way to the console by splitting the information into 
# different model components.
x

## End(Not run)
```

---

**reshape_ci**

Reshape CI between wide/long formats

**Description**

Reshape CI between wide/long formats.

**Usage**

`reshape_ci(x)`

**Arguments**

- `x`: A data frame containing columns named `CI_low` and `CI_high`. 
Examples

```r
x <- data.frame(
  Parameter = c("Term 1", "Term 2", "Term 1", "Term 2"),
  CI = c(.8, .8, .9, .9),
  CI_low = c(.2, .3, .1, .15),
  CI_high = c(.5, .6, .8, .85),
  stringsAsFactors = FALSE
)
reshape_ci(x)
reshape_ci(reshape_ci(x))
```

---

**standardize_names**

**Standardize column names**

**Description**

Standardize column names from data frames, in particular objects returned from `parameters::model_parameters()`, so column names are consistent and the same for any model object.

**Usage**

`standardize_names(data, ...)`

```r
## S3 method for class 'parameters_model'
standardize_names(
  data,
  style = c("easystats", "broom"),
  ignore_estimate = FALSE,
  ...
)
```

**Arguments**

- `data` A data frame. In particular, objects from `easystats` package functions like `parameters::model_parameters()` or `effectsize::effectsize()` are accepted, but also data frames returned by `broom::tidy()` are valid objects.
- `...` Currently not used.
- `style` Standardization can either be based on the naming conventions from the `easystats-project`, or on `broom`'s naming scheme.
- `ignore_estimate` Logical, if TRUE, column names like "mean" or "median" will not be converted to "Coefficient" resp. "estimate".
Details

This method is in particular useful for package developers or users who use, e.g., `parameters::model_parameters()` in their own code or functions to retrieve model parameters for further processing. As `model_parameters()` returns a data frame with varying column names (depending on the input), accessing the required information is probably not quite straightforward. In such cases, `standardize_names()` can be used to get consistent, i.e. always the same column names, no matter what kind of model was used in `model_parameters()`.

For style = "broom", column names are renamed to match `broom`'s naming scheme, i.e. Parameter is renamed to term, Coefficient becomes estimate and so on.

For style = "easystats", when data is an object from `broom::tidy()`, column names are converted from "broom"-style into "easystats"-style.

Value

A data frame, with standardized column names.

Examples

```r
if (require("parameters")) {
  model <- lm(mpg ~ wt + cyl, data = mtcars)
  mp <- model_parameters(model)

  as.data.frame(mp)
  standardize_names(mp)
  standardize_names(mp, style = "broom")
}
```

---

**to_numeric**

Convert to Numeric (if possible)

Description

Tries to convert vector to numeric if possible (if no warnings or errors). Otherwise, leaves it as is.

Usage

to_numeric(x)

Arguments

x

A vector to be converted.

Value

Numeric vector (if possible)
Examples

```r
to_numeric(c("1", "2"))

to_numeric(c("1", "2", "A"))
```
* data
  fish, 43

all_models_equal, 4
all_models_same_class
  (all_models_equal), 4

Bayesian models, 24, 64
bayestestR::describe_posterior(), 111, 112
bayestestR::hdi(), 111
bayestestR::weighted_posteriors(), 70

check_if_installed, 5
clean_names, 6
clean_parameters, 7
clean_parameters(), 111, 112
color_if, 8
color_text (print_color), 110
color_theme (print_color), 110
colour_if (color_if), 8
colour_text (print_color), 110

data_match, 10
data_relocate, 10
data_restoretype, 11
data_to_long, 12
data_to_wide (data_to_long), 12
display, 14
download_model, 15

effectsize::effectsize(), 114
ellipsis_info, 16
Estimated marginal means, 24, 64
export_table, 17
export_table(), 112

find_algorithm, 19
find_formula, 20
find_interactions, 22
find_offset, 23

find_parameters, 24
find_parameters(), 35
find_parameters.averaging, 25
find_parameters.bamlss
  (find_parameters.BGGM), 28
find_parameters.bayesx
  (find_parameters.BGGM), 28
find_parameters.betamfx, 26
find_parameters.betareg
  (find_parameters.averaging), 25
find_parameters.BFBayesFactor
  (find_parameters.BGGM), 28
find_parameters.BGGM, 28
find_parameters.BGGM(), 56
find_parameters.brmsfit
  (find_parameters.BGGM), 28
find_parameters.DirichletRegModel
  (find_parameters.averaging), 25
find_parameters.emmGrid, 30
find_parameters.gam
  (find_parameters.gamlss), 31
find_parameters.gamlss, 31
find_parameters.glmmTMB, 32
find_parameters.glmx
  (find_parameters.averaging), 25
find_parameters.logitmfx
  (find_parameters.betamfx), 26
find_parameters.MCMCglmm
  (find_parameters.BGGM), 28
find_parameters.merMod
  (find_parameters.glmmTMB), 32
find_parameters.mhurdle
  (find_parameters.zeroinfl), 34
find_parameters.mjjoint
  (find_parameters.averaging), 25
find_parameters.sim.merMod
  (find_parameters.BGGM), 28
find_parameters.stanreg
  (find_parameters.BGGM), 28
find_parameters.zeroinfl, 34
find_predictors, 35
find_random, 36
find_random_slopes, 37
find_response, 38
find_smooth, 39
find_statistic, 39
find_terms, 40
find_terms(), 42
find_variables, 41
find_variables(), 41
find_weights, 43
fish, 43
format_bf, 44
format_ci, 45
format_message, 46
format_number, 47
format_p, 48
format_pd, 49
format_rope, 49
format_string, 50
format_table, 51
format_value, 53
format_value(), 47
formula_ok(find_formula), 20

Generalized additive models, 24, 64
get_auxiliary, 55
get_call, 56
get_correlation_slope_intercept
   (get_variance), 91
get_correlation_slopes(get_variance), 91
get_data, 57
get_deviance, 59
get_df, 60
get_family, 61
get_intercept, 61
get_loglikelihood, 62
get_modelmatrix, 63
get_parameters, 64
get_parameters.averaging
   (get_parameters.betareg), 66
get_parameters.bamss
   (get_parameters.BGGM), 67
get_parameters.bayesx
   (get_parameters.BGGM), 67
get_parameters.betamfx, 65
get_parameters.betareg, 66
get_parameters.BFBayesFactor
   (get_parameters.BGGM), 67
get_parameters.BGGM, 67
get_parameters.brmsfit
   (get_parameters.BGGM), 67
get_parameters.clm2
   (get_parameters.betareg), 66
get_parameters.coxme
   (get_parameters.glmm), 73
get_parameters.DirichletRegModel
   (get_parameters.betareg), 66
get_parameters.emm_list
   (get_parameters.emmGrid), 71
get_parameters.emmGrid, 71
get_parameters.gam
   (get_parameters.gamm), 72
get_parameters.gamm, 72
get_parameters.glimML
   (get_parameters.glmm), 73
get_parameters.glmm, 73
get_parameters.glmmTMB
   (get_parameters.glmm), 73
get_parameters.glmx
   (get_parameters.betareg), 66
get_parameters.htest, 74
get_parameters.logitmlx
   (get_parameters.betamfx), 65
get_parameters.MCMCglmm
   (get_parameters.BGGM), 67
get_parameters.merMod
   (get_parameters.glmm), 73
get_parameters.mhurdle
   (get_parameters.zeroinfl), 75
get_parameters.mjoint
   (get_parameters.betareg), 66
get_parameters.mvord
   (get_parameters.betareg), 66
get_parameters.rqss
   (get_parameters.gamm), 72
get_parameters.sim
   (get_parameters.BGGM), 67
get_parameters.stanmvreg
   (get_parameters.BGGM), 67
get_parameters.stanreg
   (get_parameters.BGGM), 67
get_parameters.zcpglm
   (get_parameters.zeroinfl), 75
get_parameters.zeroinfl, 75
INDEX

get_predicted, 76
get_predicted(), 79
get_predicted_ci, 79
get_predicted_ci(), 77, 78
get_predictors, 81
get_priors, 82
get_random, 83
get_residuals, 84
get_response, 85
get_sigma, 86
get_statistic, 87
get_varcov, 89
get_variance, 91
get_variance_distribution (get_variance), 91
get_variance_distribution (get_variance), 91
get_variance_fixed (get_variance), 91
get_variance_intercept (get_variance), 91
get_variance_random (get_variance), 91
get_variance_residual (get_variance), 91
get_variance_slope (get_variance), 91
get_weights, 94

has_intercept, 95
Hypothesis tests, 64

is_gam_model, 95
is_mixed_model, 96
is_model, 97
is_model_supported, 98
is_multivariate, 98
is_nested_models, 99
is_nullmodel, 100
is_regression_model (is_model), 97

link_function, 101
link_inverse, 102
loglikelihood (get_loglikelihood), 62

Marginal effects models, 24, 64
Mixed models, 24, 64
model_info, 103
model_name, 106
Models with special components, 24, 64
n_obs, 107
n_parameters, 108
null_model, 107
parameters::model_parameters(), 111,
112, 114, 115
parameters_table (format_table), 51
performance::check_singularity(), 92
predict(), 76
print_color, 110
print_colours (print_color), 110
print_html (display), 14
print_md (display), 14
print_parameters, 111
print_parameters(), 7
reshape_ci, 113
reshape_longer (data_to_long), 12
reshape_wider (data_to_long), 12
sandwich::vcovCL(), 80
sandwich::vcovHC(), 80
signif(), 17, 45, 52, 54
standardize_names, 114
supported_models (is_model_supported), 98
to_numeric, 115

Zero-inflated and hurdle models, 24, 64