Package ‘insight’

June 29, 2023

Type Package

Title Easy Access to Model Information for Various Model Objects

Version 0.19.3

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 (>= 3.6)

Imports methods, stats, utils

Suggests AER, afex, aod, BayesFactor, bayesxtestR, bbmle, bdsmatrix, betareg, bife, biglm, blavaan, blme, boot, brms, broom, car, carData, censReg, cgam, clubSandwich, coxme, cplm, crch, curl, datawizard, effectsize, emmeans, epiR, estimatr, feisr, fixest, fungible, gam, gamlss, gamlss.data, gamm4, gbm, gee, geePac, geoR, GLMMAdaptive, glmmTMB, gml, grDevices, gt, htr, interp, ivreg, JM, knitr, lavaan, lavaSearch2, lfe, lme4, lmerTest, lmtree, logisf, logit, MASS, marginaleffects, Matrix, mclogit, mclust, MCMCglmm, merTools, metaBMA, metadat, metafor, metaplus, mgcv, mice, mlogit, mhurdle, mmm, multgee, nestedLm, nlme, nnet, nonnest2, ordinal, parameters, parsnip, pbkrtest, performance, plm, poorman, pscl, psych, quantreg, rmarkdown, rms, rpart, robustbase, robustlmm, rstanarm (>= 2.21.1), rstantools, rstudioapi, sandwich,
R topics documented:

speedglm, splines, statmod, survey, survival, testthat, trucreg, tweedie, TMB, VGAM

VignetteBuilder knitr
Encoding UTF-8
Language en-US
RoxygenNote 7.2.3
Config/testthat/edition 3
Config/testthat/parallel true
Config/Needs/website rstudio/bslib, r-lib/pkgdown, easystats/easystatstemplate
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, ctb] (<https://orcid.org/0000-0001-9560-6336>, @bmwiernik), Vincent Arel-Bundock [aut, ctb] (<https://orcid.org/0000-0003-2042-7063>), Etienne Bacher [aut, ctb] (<https://orcid.org/0000-0002-9271-5075>), Alex Hayes [rev] (<https://orcid.org/0000-0002-4985-5160>), Grant McDermott [ctb] (<https://orcid.org/0000-0001-7883-8573>), Rémi Thériault [ctb] (<https://orcid.org/0000-0003-4315-6788>, @rempsyc), Alex Reinhart [ctb] (<https://orcid.org/0000-0002-6658-514X>)

Repository CRAN
Date/Publication 2023-06-29 13:30:02 UTC

all_models_equal ................................................. 5
check_if_installed ............................................. 6
clean_names ...................................................... 7
clean_parameters .................................................. 8
color_if ............................................................ 9
compact_character .............................................. 11
compact_list ..................................................... 11
display ........................................................... 12
download_model .................................................. 13
R topics documented:

- ellipsis_info .................................................. 13
- export_table .................................................... 14
- find_algorithm .................................................. 17
- find_formula .................................................... 18
- find_interactions ............................................. 20
- find_offset ..................................................... 21
- find_parameters ............................................... 22
  - find_parameters.averaging .................................. 23
  - find_parameters.betamfx ..................................... 25
  - find_parameters.BGGM ......................................... 26
  - find_parameters.emmGrid ...................................... 29
  - find_parameters.gamls ................................. 30
  - find_parameters.glmmTMB ...................................... 31
  - find_parameters.zeroinfl .................................. 32
- find_predictors .................................................. 34
- find_random ...................................................... 36
- find_random_slopes ............................................ 37
- find_response ..................................................... 37
- find_smooth ..................................................... 38
- find_statistic ................................................... 39
- find_terms ......................................................... 40
- find_transformation ............................................ 41
- find_variables .................................................. 42
- find_weights ...................................................... 44
- fish .......................................................... 45
- format_bf ......................................................... 45
- format_capitalize ................................................ 46
- format_ci ......................................................... 46
- format_message .................................................. 48
- format_number ..................................................... 50
- format_p ........................................................ 51
- format_pd ........................................................ 52
- format_rope ....................................................... 53
- format_string ..................................................... 54
- format_table ....................................................... 55
- format_value ....................................................... 57
- get_auxiliary ...................................................... 59
- get_call ........................................................ 60
- get_data .......................................................... 61
- get_datagrid ........................................................ 63
- get_deviance ....................................................... 68
- get_df .......................................................... 69
- get_family ......................................................... 71
- get_intercept ...................................................... 71
- get_loglikelihood ............................................... 72
- get_modelmatrix .................................................. 73
- get_parameters .................................................... 74
  - get_parameters.betamfx ...................................... 75
get_parameters.betareg .................................................. 76
get_parameters.BGGM .................................................... 78
get_parameters.emmGrid ............................................... 81
get_parameters.gamm .................................................. 82
get_parameters.glmm .................................................. 83
get_parameters.htest ................................................. 85
get_parameters.zeroinfl ............................................... 86
get_predicted .......................................................... 87
get_predicted_ci ....................................................... 92
get_predictors .......................................................... 95
get_priors ............................................................... 96
get_random ............................................................. 97
get_residuals ............................................................. 98
get_response ........................................................... 99
gamma ................................................................. 100
gamma_statistic ....................................................... 101
get_transformation ................................................... 103
get_varcov .............................................................. 104
get_variance ............................................................ 107
get_weights ............................................................. 110
has_intercept ........................................................... 111
is_converged ............................................................ 111
is_empty_object ....................................................... 113
is_gam_model .......................................................... 114
is_mixed_model ........................................................ 114
is_model ................................................................. 115
is_model_supported ................................................... 116
is_multivariate ........................................................ 117
is_nested_models ...................................................... 118
is_nullmodel ........................................................... 119
link_function ............................................................ 119
link_inverse ............................................................. 120
model_info ............................................................... 121
model_name .............................................................. 124
null_model ............................................................... 125
n_grouplevels .......................................................... 125
n_obs ................................................................. 126
n_parameters ............................................................ 127
object_has_names ...................................................... 129
print_color .............................................................. 130
print_parameters ........................................................ 131
standardize_column_order .......................................... 133
standardize_names ................................................... 134
text_remove_backticks ................................................ 136
trim_ws ................................................................. 137

Index ................................................................. 139
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

all_models_equal(..., verbose = FALSE)
all_models_same_class(..., verbose = FALSE)

Arguments

... A list of objects.
verbose Toggle off warnings.

Value

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

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,
  quietly = FALSE,
  prompt = interactive(),
  ...
)

Arguments

- **package**: A character vector naming the package(s), 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**: A character vector, representing the minimum package version that is required for each package. Should be of same length as `package`. If `NULL`, no check for minimum version is done.
- **quietly**: Logical, if `TRUE`, invisibly returns a vector of logicals (TRUE for each installed package, FALSE otherwise), and does not stop or throw a warning. If `quietly = TRUE`, arguments `stop` and `prompt` are ignored. Use this argument to internally check for package dependencies without stopping or warnings.
- **prompt**: If TRUE, will prompt the user to install needed package(s). Ignored if `quietly = TRUE`.
- ...: Currently ignored

Value

If `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("insight")
try(check_if_installed("nonexistent_package"))
try(check_if_installed("insight", minimum_version = "99.8.7"))
try(check_if_installed(c("nonexistent", "also_not_here"), stop = FALSE))
```

```r
## 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 `log()` or `as.factor()` etc.

Usage

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

**S3 method for class 'character'**

```r
clean_names(x, include_names = FALSE, ...)
```

Arguments

- **x**: A fitted model, or a character vector.
- **...**: Currently not used.
- **include_names**: Logical, if TRUE, returns a named vector where names are the original values of `x`.

Value

The "cleaned" variable names as character vector, i.e. pattern like `s()` for splines or `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'.

---

**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 `log()` or `as.factor()` etc.

Usage

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

**S3 method for class 'character'**

```r
clean_names(x, include_names = FALSE, ...)
```

Arguments

- **x**: A fitted model, or a character vector.
- **...**: Currently not used.
- **include_names**: Logical, if TRUE, returns a named vector where names are the original values of `x`.

Value

The "cleaned" variable names as character vector, i.e. pattern like `s()` for splines or `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

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

---

color_if  

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

```r
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'.
color_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".
color_else, colour_else See color_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))

Value

The.

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 <- colour_if(iris, columns = "Species", predicate = `==` , value = "setosa")
cat(x$Species)

# own function, argument "value" not needed here
```r
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)
```

---

### compact_character

**Description**
Remove empty strings from character

**Usage**

```r
compact_character(x)
```

**Arguments**

- `x` A single character or a vector of characters.

**Value**
A character or a character vector with empty strings removed.

**Examples**

```r
compact_character(c("x", "y", NA))
compact_character(c("x", "NULL", "", "y"))
```

---

### compact_list

**Description**
Remove empty elements from lists

**Usage**

```r
compact_list(x, remove_na = FALSE)
```

**Arguments**

- `x` A list or vector.
- `remove_na` Logical to decide if NAs should be removed.
### Examples

```r
compact_list(list(NULL, 1, c(NA, NA)))
compact_list(c(1, NA, NA))
compact_list(c(1, NA, NA), remove_na = TRUE)
```

### 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, ...)
```

```r
## S3 method for class 'data.frame'
display(object, format = "markdown", ...)

## S3 method for class 'data.frame'
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

```r
display(iris[1:5, ])
```
**download_model**

*Download circus models*

**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**

```r
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.github.com/easystats/circus/master/data/`.

**Details**

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

**Value**

A model from the *circus*-repository, or `NULL` if model could not be downloaded (e.g., due to server problems).

**References**

[https://easystats.github.io/circus/](https://easystats.github.io/circus/)

---

**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:
ellipsis_info(..., only_models = TRUE, verbose = TRUE)
```
Arguments

- `objects, ...` Arbitrary number of objects. May also be a list of model objects.
- `only_models` Only keep supported models (default to `TRUE`).
- `verbose` Toggle warnings.

Value

The list with objects that were passed to the function, including additional information as attributes (e.g., if models have the 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, m5)
attributes(objects)$is_nested

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

Description

Data frame and Tables Pretty Formatting

Usage

```r
export_table(
  x,
  sep = " |",
  header = "-",
  cross = NULL,
  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,
table_width = NULL,
verbose = TRUE,

Arguments

x A data frame. May also be a list of data frames, to export multiple data frames into multiple tables.
sep Column separator.
header Header separator. Can be NULL.
cross Character that is used where separator and header lines cross.
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 Refers to the width of columns (with numeric values). Can be either NULL, a number or a named numeric vector. If NULL, the width for each column is adjusted to the minimum required width. If a number, columns with numeric values will have the minimum width specified in width. If a named numeric vector, value names are matched against column names, and for each match, the specified width is used (see 'Examples'). Only applies to text-format (see format).
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). If x is a list of data frames, caption may be a list of table captions, one for each table.
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. If x is a list of data frames, footer may be a list of table captions, one for
each 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.
table_width  Numeric, or "auto", indicating the width of the complete table. If table_width
= "auto" and the table is wider than the current width (i.e. line length) of the
console (or any other source for textual output, like markdown files), the table is
split into two parts. Else, if table_width is numeric and table rows are larger
than table_width, the table is split into two parts.
verbose  Toggle messages and warnings.
...  Currently not used.

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
export_table(head(iris))
export_table(head(iris), cross = "*")
export_table(head(iris), sep = " ", header = "*", digits = 1)

# split longer tables
### 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
```r
find_algorithm(x, ...)  
```
find_formula

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

```r
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`).
find_formula

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, \texttt{conditional}, is returned. For more complex models, the returned list may have following elements:

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

Note

For models of class \texttt{lme} or \texttt{gls} the correlation-component is only returned, when it is explicitly defined as named argument (\texttt{form}), e.g. \texttt{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
format(f)
}

find_interactions  Find interaction terms from models

Description

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

Usage

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. Dupli-
            cated values are removed.

Value

A list of character vectors that represent the interaction terms. Depending on component, the re-
turned 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
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)
```

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
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.

Model components

Possible values for the `component` argument depend on the model class. Following are valid options:

- "all": returns all model components, applies to all models, but will only have an effect for models with more than just the conditional model component.
- "conditional": only returns the conditional component, i.e. "fixed effects" terms from the model. Will only have an effect for models with more than just the conditional model component.
• "smooth_terms": returns smooth terms, only applies to GAMs (or similar models that may contain smooth terms).
• "zero_inflated" (or "zi"): returns the zero-inflation component.
• "dispersion": returns the dispersion model component. This is common for models with zero-inflation or that can model the dispersion parameter.
• "instruments": for instrumental-variable or some fixed effects regression, returns the instruments.
• "location": returns location parameters such as conditional, zero_inflated, smooth_terms, or instruments (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters).
• "distributional" (or "auxiliary"): components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

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 '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"),
```
find_parameters.averaging

```r
find_parameters(
  x,
  component = c("all", "conditional", "survival"),
  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.
**find_parameters.betamfx**

Examples

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

---

**find_parameters.betamfx**

*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**

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

```r
## S3 method for class 'logitmfx'
find_parameters(
  x,
  component = c("all", "conditional", "marginal", "location"),
  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.
find_parameters.BGGM

Parameters:

- 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)
```

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

```r
## 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,
  ...
)

## S3 method for class 'bayesx'
find_parameters(
  x,
  component = c("all", "conditional", "smooth_terms"),
  flatten = FALSE,
  parameters = NULL,
  ...
)

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

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

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)
find_parameters.emmGrid

Find model parameters from estimated marginal means objects

Description

Returns the parameter names from a model.

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. Dupli-
cated 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 pa-
rameters 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

```r
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

```r
## 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.
- `...` Currently not used.
- `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 'nlmerMod'
find_parameters(
  x,
  effects = c("all", "fixed", "random"),
  component = c("all", "conditional", "nonlinear"),
  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
find_parameters.zeroinfl

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.

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)
- nonlinear, the parameters from the nonlinear formula.

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

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

```r
## 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 \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:  
            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 - depend-  
            ing 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 \texttt{TRUE}, the values are returned as character vector, not as list. Dupli-  
            cated values are removed.

...        Currently not used.

Value

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

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

Examples

data(mtcars)  
m <- \texttt{lm(mpg \sim wt + cyl + vs, data = mtcars)}  
find_parameters(m)
find_predictors — Find names of model predictors

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

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.

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

Model components

Possible values for the component argument depend on the model class. Following are valid options:

• "all": returns all model components, applies to all models, but will only have an effect for models with more than just the conditional model component.
• "conditional": only returns the conditional component, i.e. "fixed effects" terms from the model. Will only have an effect for models with more than just the conditional model component.
• "smooth_terms": returns smooth terms, only applies to GAMs (or similar models that may contain smooth terms).
• "zero_inflated" (or "zi"): returns the zero-inflation component.
• "dispersion": returns the dispersion model component. This is common for models with zero-inflation or that can model the dispersion parameter.
• "instruments": for instrumental-variable or some fixed effects regression, returns the instruments.
• "location": returns location parameters such as conditional, zero_inflated, smooth_terms, or instruments (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters).
• "distributional" (or "auxiliary"): components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

Examples

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

Find names of random effects

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.

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

**Examples**

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

---

**find_response**  
*Find name of the response variable*

**Description**  
Returns the name(s) of the response variable(s) from a model object.

**Usage**  
`find_response(x, combine = TRUE, ...)`
find_smooth

Arguments

x A fitted model.

combine Logical, if TRUE and the response is a matrix-column, the name of the response matches the notation in formula, and would for instance also contain patterns like "cbind(...)". Else, the original variable names from the matrix-column are returned. See 'Examples'.

... Currently not used.

Value

The name(s) of the response variable(s) from x as character vector, or NULL if response variable could not be found.

Examples

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

  find_response(m, combine = TRUE)
  find_response(m, combine = FALSE)
}

find_smooth

Find smooth terms from a model object

Description

Return the names of smooth terms from a model object.

Usage

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.
find_statistic

Examples

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

find_statistic(x, ...)

Arguments

x
  An object.

...  Currently not used.

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

```r
find_terms(x, ...)
```

```r
## Default S3 method:
find_terms(x, flatten = FALSE, as_term_labels = 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.
- `as_term_labels`  
  Logical, if TRUE, extracts model formula and tries to access the "term.labels" attribute. This should better mimic the `terms()` behaviour even for those models that do not have such a method, but may be insufficient, e.g. for mixed models.
- `verbose`  
  Toggle warnings.

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).

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

  find_terms(m)
}

# sometimes, it is necessary to retrieve terms from "term.labels" attribute
m <- lm(mpg ~ hp * (am + cyl), data = mtcars)
find_terms(m, as_term_labels = TRUE)

find_transformation(x)

Arguments

x  A regression model.

Value

A string, with the name of the function of the applied transformation. Returns "identity" for no transformation, and e.g. "log(x+3)" when a specific values was added to the response variables before log-transforming. For unknown transformations, returns NULL.

Examples

# identity, no transformation
model <- lm(Sepal.Length ~ Species, data = iris)
find_transformation(model)

# log-transformation
model <- lm(log(Sepal.Length) ~ Species, data = iris)
find_transformation(model)
# log+2
model <- lm(log(Sepal.Length + 2) ~ Species, data = iris)
find_transformation(model)

---

### 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 = "all",
  component = "all",
  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.
- **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
- conditional, the names of the predictor variables from the *conditional* model (as opposed to the zero-inflated part of a model)
- cluster, the names of cluster or grouping variables
- dispersion, the name of the dispersion terms
- instruments, the names of instrumental variables
- 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)

**Model components**

Possible values for the component argument depend on the model class. Following are valid options:

- "all": returns all model components, applies to all models, but will only have an effect for models with more than just the conditional model component.
- "conditional": only returns the conditional component, i.e. "fixed effects" terms from the model. Will only have an effect for models with more than just the conditional model component.
- "smooth_terms": returns smooth terms, only applies to GAMs (or similar models that may contain smooth terms).
- "zero_inflated" (or "zi"): returns the zero-inflation component.
- "dispersion": returns the dispersion model component. This is common for models with zero-inflation or that can model the dispersion parameter.
- "instruments": for instrumental-variable or some fixed effects regression, returns the instruments.
- "location": returns location parameters such as conditional, zero_inflated, smooth_terms, or instruments (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters).
- "distributional" (or "auxiliary"): components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

**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(cbppl)
  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)
  }
```
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

```r
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)
```

---

**format_capitalize**  
*Capitalizes the first letter in a string*

**Description**

This function converts the first letter in a string into upper case.

**Usage**

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

**Arguments**

- `x`: A character vector or a factor. The latter is coerced to character. All other objects are returned unchanged.
- `verbose`: Toggle warnings.

**Value**

`x`, with first letter capitalized.

**Examples**

```r
format_capitalize("hello")
format_capitalize(c("hello", "world"))
unique(format_capitalize(iris$Species))
```

---

**format_ci**  
*Confidence/Credible Interval (CI) Formatting*

**Description**

Confidence/Credible Interval (CI) Formatting
format_ci

Usage

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

```r
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 spaces.

`format_alert()` is a wrapper that combines formatting a string with a call to `message()`, `warning()` or `stop()`. By default, `format_alert()` creates a `message()`. `format_warning()` and `format_error()` change the default type of exception to `warning()` and `stop()`, respectively.

**Usage**

```r
format_message(
  string,
  ..., 
  line_length = 0.9 * getOption("width", 80),
  indent = " "
)

format_alert(
  string,
  ..., 
  line_length = 0.9 * getOption("width", 80),
  indent = " ",
  type = "message",
  call. = FALSE
)

format_warning(...)

format_error(...)
```
**Arguments**

- **string**: A string.
- **...**: Further strings that will be concatenated as indented new lines.
- **line_length**: Numeric, the maximum length of a line. The default is 90% of the width of the console window.
- **indent**: Character vector. If further lines are specified in . . . , a user-defined string can be specified to indent subsequent lines. Defaults to " " (two white spaces), hence for each start of the line after the first line, two white space characters are inserted.
- **type**: Type of exception alert to raise. Can be "message" for message(), "warning" for warning(), or "error" for stop().
- **call**: Logical. Indicating if the call should be included in the the error message. This is usually confusing for users when the function producing the warning or error is deep within another function, so the default is FALSE.

**Details**

There is an experimental formatting feature implemented in this function. You can use following tags:

- {.b text} for bold formatting
- {.i text} to use italic font style
- {.url www.url.com} formats the string as URL (i.e., enclosing URL in < and >, blue color and italic font style)
- {.pkg packagename} formats the text in blue color.

This feature has some limitations: it’s hard to detect the exact length for each line when the string has multiple lines (after line breaks) and the string contains formatting tags. Thus, it can happen that lines are wrapped at an earlier length than expected. Furthermore, if you have multiple words in a format tag ( {.b one two three} ), a line break might occur inside this tag, and the formatting no longer works (messing up the message-string).

**Value**

For format_message(), a formatted string. For format_alert() and related functions, the requested exception, with the exception formatted using format_message().

**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)",
```

```r
line_length = 30
message(msg)
msg <- format_message("Much too long string for just one line, I guess!",
  "First new line",
  "Second new line",
  "(not indented)",
  line_length = 30,
  indent = ""
)
message(msg)

# Caution, experimental! See 'Details'
msg <- format_message(
  "This is {.i italic}, visit {.url easystats.github.io/easystats}",
  line_length = 30
)
message(msg)

# message
format_alert("This is a message.")
format_alert("This is a warning.", type = "message")

# error
try(format_error("This is an error."))

# warning
format_warning("This is a warning.")
```

---

**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)
format_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,
  whitespace = TRUE,
  name = "p",
  missing = "",
  decimal_separator = NULL,
  digits = 3,
  ...
)
```

Arguments

- **p**: value or vector of p-values.
- **stars**: Add significance stars (e.g., p < .001***).
- **stars_only**: Return only significance stars.
- **whitespace**: Logical, if TRUE (default), preserves whitespaces. Else, all whitespace characters are removed from the returned string.
- **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.
- **decimal_separator**: Character, if not NULL, will be used as decimal separator.
**format_p**

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)

model <- lm(mpg ~ wt + cyl, data = mtcars)
p <- coef(summary(model))[, 4]
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.
format_rope

Description
Percentage in ROPE formatting

Usage
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)
format_string

String Values Formatting

Description

String Values Formatting

Usage

format_string(x, ...)

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

Arguments

x String value.
...
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)
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

```r
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,
  ic_digits = 1,
  zap_small = FALSE,
  preserve_attributes = FALSE,
  exact = TRUE,
  use_symbols = getOption("insight_use_symbols", 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` If TRUE, add significance stars (e.g., p < .001***). Can also be a character vector, naming the columns that should include stars for significant values. This is especially useful for Bayesian models, where we might have multiple columns with significant values, e.g. BF for the Bayes factor or pd for the probability of direction. In such cases, use `stars = c("pd", "BF")` to add stars to both columns, or `stars = "BF"` to only add stars to the Bayes factor and exclude the pd column. Currently, following columns are recognized: "BF", "pd" and "p".
- `digits`, `ci_digits`, `p_digits`, `rope_digits`, `ic_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.

**exact**
Formatting for Bayes factor columns, in case the provided data frame contains
such a column (i.e. columns named "BF" or "log_BF"). For exact = TRUE, very
large or very small values are then either reported with a scientific format (e.g.,
4.24e5), else as truncated values (as "> 1000" and "< 1/10000").

**use_symbols**
Logical, if TRUE, column names that refer to particular effect sizes (like Phi,
Omega or Epsilon) include the related unicode-character instead of the written
name. This only works on Windows for R >= 4.2, and on OS X or Linux for
R >= 4.0. It is possible to define a global option for this setting, see 'Note'.

**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!

**Note**

options(insight_use_symbols = TRUE) override the use_symbols argument and always dis-
plays symbols, if possible.

**See Also**

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

**Examples**

```r
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", warn.conflicts = FALSE) &&
require("parameters", , warn.conflicts = FALSE)) {
  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**

format_value() converts numeric values into formatted string values, where formatting can be something like rounding digits, scientific notation etc. format_percent() is a short-cut for format_value(as_percent = TRUE).

**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,
  lead_zero = TRUE,
  style_positive = "none",
  style_negative = "hyphen",
  ...
)
```

```r
## S3 method for class 'numeric'
format_value(
  x,
  digits = 2,
  protect_integers = FALSE,
  missing = "",
  width = NULL,
  as_percent = FALSE,
  zap_small = FALSE,
  lead_zero = TRUE,
  style_positive = "none",
  style_negative = "hyphen",
  ...
)
```
format_value

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 = "scientific:4"` to have scientific notation with 4 decimal places, or `digits = "signif:5"` 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.
- `lead_zero`: Logical, if TRUE (default), includes leading zeros, else leading zeros are dropped.
- `style_positive`: A string that determines the style of positive numbers. May be "none" (default), "plus" to add a plus-sign or "space" to precede the string by a Unicode "figure space", i.e., a space equally as wide as a number or +.
- `style_negative`: A string that determines the style of negative numbers. May be "hyphen" (default), "minus" for a proper Unicode minus symbol or "parens" to wrap the number in parentheses.

Value

A formatted string.

Examples

```r
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 = "scientific:2")
format_value(c(0.045, .12, .34), lead_zero = FALSE)
```
# 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(head(iris))

---

**get_auxiliary**  
*Get auxiliary parameters from models*

**Description**

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

**Usage**

```r
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.
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.
get_data

Value

A function call.

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)
}

Description

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

Usage

get_data(x, ...)

## Default S3 method:
get_data(x, source = "environment", verbose = TRUE, ...)

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

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

## S3 method for class 'rma'
get_data(
x,
source = "environment",
verbose = TRUE,
include_interval = FALSE,
transf = NULL,
transf_args = NULL,
ci = 0.95,
...
)

Arguments

x
A fitted model.

... Currently not used.

source
String, indicating from where data should be recovered. If source = "environment" (default), data is recovered from the environment (e.g., if the data is in the workspace). This option is usually the fastest way of getting data and ensures that the original variables used for model fitting are returned. Note that always the current data is recovered from the environment. Hence, if the data was modified after model fitting (e.g., variables were recoded or rows filtered), the returned data may no longer equal the model data. If source = "frame" (or "mf"), the data is taken from the model frame. Any transformed variables are back-transformed, if possible. This option returns the data even if it is not available in the environment, however, in certain edge cases back-transforming to the original data may fail. If source = "environment" fails to recover the data, it tries to extract the data from the model frame; if source = "frame" and data cannot be extracted from the model frame, data will be recovered from the environment. Both ways only returns observations that have no missing data in the variables used for model fitting.

verbose
Toggle messages and warnings.

effects
Should model data for fixed effects ("fixed"), random effects ("random") or both ("all") be returned? Only applies to mixed or gee models.

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.

include_interval
For meta-analysis models, should normal-approximation confidence intervals be added for each response effect size?

transf
For meta-analysis models, if intervals are included, a function applied to each response effect size and its interval.

transf_args
For meta-analysis models, an optional list of arguments passed to the transf function.

ci
For meta-analysis models, the Confidence Interval (CI) level if include_interval = TRUE. Default to 0.95 (95%).
get_datagrid

Value
The data that was used to fit the model.

Model components
Possible values for the component argument depend on the model class. Following are valid options:

- "all": returns all model components, applies to all models, but will only have an effect for models with more than just the conditional model component.
- "conditional": only returns the conditional component, i.e. "fixed effects" terms from the model. Will only have an effect for models with more than just the conditional model component.
- "smooth_terms": returns smooth terms, only applies to GAMs (or similar models that may contain smooth terms).
- "zero_inflated" (or "zi"): returns the zero-inflation component.
- "dispersion": returns the dispersion model component. This is common for models with zero-inflation or that can model the dispersion parameter.
- "instruments": for instrumental-variable or some fixed effects regression, returns the instruments.
- "location": returns location parameters such as conditional, zero_inflated, smooth_terms, or instruments (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters).
- "distributional" (or "auxiliary"): components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

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))
}

get_datagrid
Create a reference grid

Description
Create a reference matrix, useful for visualisation, with evenly spread and combined values. Usually used to make generate predictions using get_predicted(). See this vignette for a tutorial on how to create a visualisation matrix using this function.
get_datagrid(x, ...)  

## S3 method for class 'data.frame'
get_datagrid(
  x,
  at = "all",
  factors = "reference",
  numerics = "mean",
  preserve_range = FALSE,
  reference = x,
  length = 10,
  range = "range",
  ...
)

## S3 method for class 'numeric'
get_datagrid(x, length = 10, range = "range", ...)

## S3 method for class 'factor'
get_datagrid(x, ...)

## Default S3 method:
get_datagrid(
  x,
  at = "all",
  factors = "reference",
  numerics = "mean",
  preserve_range = TRUE,
  reference = x,
  include_smooth = TRUE,
  include_random = FALSE,
  include_response = FALSE,
  data = NULL,
  verbose = TRUE,
  ...
)

Arguments

**x**: An object from which to construct the reference grid.

**...**: Arguments passed to or from other methods (for instance, length or range to control the spread of numeric variables.).

**at**: Indicates the focal predictors (variables) for the reference grid and at which values focal predictors should be represented. If not specified otherwise, representative values for numeric variables or predictors are evenly distributed from the minimum to the maximum, with a total number of length values covering that range (see 'Examples'). Possible options for at are:
• "all", which will include all variables or predictors.
• a character vector of one or more variable or predictor names, like c("Species", "Sepal.Width"), which will create a grid of all combinations of unique values. For factors, will use all levels, for numeric variables, will use a range of length length (evenly spread from minimum to maximum) and for character vectors, will use all unique values.
• a list of named elements, indicating focal predictors and their representative values, e.g. at = list(Sepal.Length = c(2, 4), Species = "setosa").
• a string with assignments, e.g. at = "Sepal.Length = 2" or at = c("Sepal.Length = 2", "Species = 'setosa'") - note the usage of single and double quotes to assign strings within strings.

There is a special handling of assignments with brackets, i.e. values defined inside [ and ]. For numeric variables, the value(s) inside the brackets should either be
• two values, indicating minimum and maximum (e.g. at = "Sepal.Length = [0, 5]"), for which a range of length length (evenly spread from given minimum to maximum) is created.
• more than two numeric values at = "Sepal.Length = [2,3,4,5]", in which case these values are used as representative values.
• a "token" that creates pre-defined representative values:
  – for mean and +/- 1 SD around the mean: "x = [sd]"
  – for median and +/- 1 MAD around the median: "x = [mad]"
  – for Tukey's five number summary (minimum, lower-hinge, median, upper-hinge, maximum): "x = [fivenum]"
  – for terciles, including minimum and maximum: "x = [terciles]"
  – for terciles, excluding minimum and maximum: "x = [terciles2]"
  – for quartiles, including minimum and maximum: "x = [quartiles]"
  – for quartiles, excluding minimum and maximum: "x = [quartiles2]"
  – for minimum and maximum value: "x = [minmax]"
  – for 0 and the maximum value: "x = [zeromax]"

For factor variables, the value(s) inside the brackets should indicate one or more factor levels, like at = "Species = [setosa, versicolor]". Note: the length argument will be ignored when using brackets-tokens.

The remaining variables not specified in at will be fixed (see also arguments factors and numerics).

**factors**
Type of summary for factors. Can be "reference" (set at the reference level), "mode" (set at the most common level) or "all" to keep all levels.

**numerics**
Type of summary for numeric values. Can be "all" (will duplicate the grid for all unique values), any function ("mean", "median", ...) or a value (e.g., numerics = 0).

**preserve_range**
In the case of combinations between numeric variables and factors, setting preserve_range = TRUE will drop the observations where the value of the numeric variable is originally not present in the range of its factor level. This leads to an unbalanced grid. Also, if you want the minimum and the maximum to closely match the actual ranges, you should increase the length argument.
get_datagrid

**reference**
The reference vector from which to compute the mean and SD. Used when standardizing or unstandardizing the grid using `effectsize::standardize`.

**length**
Length of numeric target variables selected in "at". This argument controls the number of (equally spread) values that will be taken to represent the continuous variables. A longer length will increase precision, but can also substantially increase the size of the datagrid (especially in case of interactions). If NA, will return all the unique values. In case of multiple continuous target variables, length can also be a vector of different values (see examples).

**range**
Option to control the representative values given in `at`, if no specific values were provided. Use in combination with the `length` argument to control the number of values within the specified range. `range` can be one of the following:

- "range" (default), will use the minimum and maximum of the original data vector as end-points (min and max).
- if an interval type is specified, such as "iqr", "ci", "hdi" or "eti", it will spread the values within that range (the default CI width is 95% but this can be changed by adding for instance `ci = 0.90`). See `IQR()` and `bayestestR::ci()`.
- if "sd" or "mad", it will spread by this dispersion index around the mean or the median, respectively. If the `length` argument is an even number (e.g., 4), it will have one more step on the positive side (i.e., -1, 0, +1, +2). The result is a named vector. See 'Examples.'
- "grid" will create a reference grid that is useful when plotting predictions, by choosing representative values for numeric variables based on their position in the reference grid. If a numeric variable is the first predictor in `at`, values from minimum to maximum of the same length as indicated in `length` are generated. For numeric predictors not specified at first in `at`, mean and -1/+1 SD around the mean are returned. For factors, all levels are returned.

**include_smooth**
If `x` is a model object, decide whether smooth terms should be included in the data grid or not.

**include_random**
If `x` is a mixed model object, decide whether random effect terms should be included in the data grid or not. If `include_random` is FALSE, but `x` is a mixed model with random effects, these will still be included in the returned grid, but set to their "population level" value (e.g., NA for `glmmTMB` or 0 for `merMod`). This ensures that common `predict()` methods work properly, as these usually need data with all variables in the model included.

**include_response**
If `x` is a model object, decide whether the response variable should be included in the data grid or not.

**data**
Optional, the data frame that was used to fit the model. Usually, the data is retrieved via `get_data()`.

**verbose**
Toggle warnings.

**Value**
Reference grid data frame.
get_datagrid

See Also

get_predicted()

Examples

# Datagrids of variables and dataframes
if (require("bayestestR", quietly = TRUE) & require("datawizard", quietly = TRUE)) {
  # Single variable is of interest; all others are "fixed"
  # Factors
  get_datagrid(iris, at = "Species") # Returns all the levels
  get_datagrid(iris, at = "Species = c('setosa', 'versicolor')") # Specify an expression

  # Numeric variables
  get_datagrid(iris, at = "Sepal.Length") # default spread length = 10
  get_datagrid(iris, at = "Sepal.Length", length = 3) # change length
  get_datagrid(iris[2:150, ], at = "Sepal.Length",
              factors = "mode", numerics = "median") # change non-targets fixing
  get_datagrid(iris, at = "Sepal.Length", range = "ci", ci = 0.90) # change min/max of target
  get_datagrid(iris, at = "Sepal.Length = [0, 1]") # Manually change min/max
  get_datagrid(iris, at = "Sepal.Length = [sd]") # -1 SD, mean and +1 SD
  get_datagrid(iris, at = "Sepal.Length", range = "sd", length = 3)
  get_datagrid(iris, at = "Sepal.Length = [quartiles]") # quartiles

  # Numeric and categorical variables, generating a grid for plots
  get_datagrid(iris, at = c("Sepal.Length", "Species"), range = "grid")
  get_datagrid(iris, at = c("Sepal.Length", "Petal.Length"), length = c(3, 2))
  get_datagrid(iris, at = c(1, 3), length = 3)
  get_datagrid(iris, at = c("Sepal.Length", "Species"), preserve_range = TRUE)
  get_datagrid(iris, at = c("Sepal.Length", "Species"), numerics = 0)
  get_datagrid(iris, at = c("Sepal.Length = 3", "Species"))
  get_datagrid(iris, at = c("Sepal.Length = c(3, 1)", "Species = 'setosa'"))

  # With list-style at-argument

data <- get_datagrid(iris, at = "Sepal.Length", range = "sd", length = 3)
data$Sepal.Length # It is a named vector (extract names with names(out$Sepal.Length))
datawizard::standardize(data, select = "Sepal.Length")
data <- get_datagrid(iris, at = "Sepal.Length = c(-2, 0, 2)") # Manually specify values
data datawizard::unstandardize(data, select = "Sepal.Length")

# Multiple variables are of interest, creating a combination
get_datagrid(iris, at = c("Sepal.Length", "Species"), length = 3)
get_datagrid(iris, at = c("Sepal.Length", "Petal.Length"), length = c(3, 2))
get_datagrid(iris, at = c("Sepal.Length", "Species"), preserve_range = TRUE)
get_datagrid(iris, at = c("Sepal.Length", "Species"), numerics = 0)
get_datagrid(iris, at = c("Sepal.Length = 3", "Species"))
get_datagrid(iris, at = c("Sepal.Length = c(3, 1)", "Species = 'setosa'"))
```r
get_datagrid(iris, at = list(Sepal.Length = c(1, 3), Species = "setosa"))
}

# With models

# Fit a linear regression
model <- lm(Sepal.Length ~ Sepal.Width * Petal.Length, data = iris)
# Get datagrid of predictors
data <- get_datagrid(model, length = c(20, 3), range = c("range", "sd"))
# same as: get_datagrid(model, range = "grid", length = 20)
# Add predictions
data$Sepal.Length <- get_predicted(model, data = data)
# Visualize relationships (each color is at -1 SD, Mean, and +1 SD of Petal.Length)
plot(data$Sepal.Width, data$Sepal.Length,
     col = data$Petal.Length,
     main = "Relationship at -1 SD, Mean, and +1 SD of Petal.Length")
```

---

**get_deviance**

<table>
<thead>
<tr>
<th>Model Deviance</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```r
get_deviance(x, ...)
```

## Default S3 method:

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

**Arguments**

- `x` A model.
- `...` Not used.
- `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

```r
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

```r
get_df(x, ...)
```

## Default S3 method:

```r
get_df(x, type = "residual", verbose = TRUE, ...)
```

Arguments

- `x` A statistical model.
- `...` Currently not used.
- `type` Can be "residual", "wald", "normal", or "model". "analytical" is an alias for "residual".
  - "residual" (aka "analytical") returns the residual degrees of freedom, which usually is what `stats::df.residual()` returns. If a model object has no method to extract residual degrees of freedom, these are calculated as \( n - p \), i.e. the number of observations minus the number of estimated parameters. If residual degrees of freedom cannot be extracted by either approach, returns Inf.
  - "wald" returns residual (aka analytical) degrees of freedom for models with t-statistic, 1 for models with Chi-squared statistic, and Inf for all other models. Also returns Inf if residual degrees of freedom cannot be extracted.
  - "normal" always returns Inf.
  - "model" returns model-based degrees of freedom, i.e. the number of (estimated) parameters.
  - For mixed models, can also be "ml1" (approximation of degrees of freedom based on a "m-l-1" heuristic as suggested by Elff et al. 2019) or "betwithin", and for models of class merMod, type can also be "satterthwaite" or "kenward-roger". See 'Details'.

Usually, when degrees of freedom are required to calculate p-values or confidence intervals, type = "wald" is likely to be the best choice in most cases.

- `verbose` Toggle warnings.
Details

Degrees of freedom for mixed models
Inferential statistics (like p-values, confidence intervals and standard errors) may be biased in mixed models when the number of clusters is small (even if the sample size of level-1 units is high). In such cases it is recommended to approximate a more accurate number of degrees of freedom for such inferential statistics (see Li and Redden 2015).

m-l-1 degrees of freedom
The m-l-1 heuristic is an approach that uses a t-distribution with fewer degrees of freedom. In particular for repeated measure designs (longitudinal data analysis), the m-l-1 heuristic is likely to be more accurate than simply using the residual or infinite degrees of freedom, because get_df(type = "ml1") returns different degrees of freedom for within-cluster and between-cluster effects. Note that the "m-l-1" heuristic is not applicable (or at least less accurate) for complex multilevel designs, e.g. with cross-classified clusters. In such cases, more accurate approaches like the Kenward-Roger approximation is recommended. However, the "m-l-1" heuristic also applies to generalized mixed models, while approaches like Kenward-Roger or Satterthwaite are limited to linear mixed models only.

Between-within degrees of freedom
The Between-within denominator degrees of freedom approximation is, similar to the "m-l-1" heuristic, recommended in particular for (generalized) linear mixed models with repeated measurements (longitudinal design). get_df(type = "betwithin") implements a heuristic based on the between-within approach, i.e. this type returns different degrees of freedom for within-cluster and between-cluster effects. Note that this implementation does not return exactly the same results as shown in Li and Redden 2015, but similar.

Satterthwaite and Kenward-Rogers degrees of freedom
Unlike simpler approximation heuristics like the "m-l-1" rule (type = "ml1"), the Satterthwaite or Kenward-Rogers approximation is also applicable in more complex multilevel designs. However, the "m-l-1" or "between-within" heuristics also apply to generalized mixed models, while approaches like Kenward-Roger or Satterthwaite are limited to linear mixed models only.

References


Examples

model <- lm(Sepal.Length ~ Petal.Length * Species, data = iris)
### get_family

A robust alternative to `stats::family`.

#### 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)
}
```

### get_intercept

Get the value at the intercept.

#### 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.
...

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,
check_response = FALSE,
verbose = TRUE,
...
)
get_modelmatrix

Arguments

x  A model.

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. estimator="REML" will give same results as logLik(..., REML=TRUE).

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.

check_response  Logical, if TRUE, checks if the response variable is transformed (like log() or sqrt()), and if so, returns a corrected log-likelihood. To get back to the original scale, the likelihood of the model is multiplied by the Jacobian/derivative of the transformation.

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()).
get_parameters

Examples

data(mtcars)

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

get_parameters

Get model parameters

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

Model components

Possible values for the component argument depend on the model class. Following are valid options:

- "all": returns all model components, applies to all models, but will only have an effect for models with more than just the conditional model component.
- "conditional": only returns the conditional component, i.e. "fixed effects" terms from the model. Will only have an effect for models with more than just the conditional model component.
- "smooth_terms": returns smooth terms, only applies to GAMs (or similar models that may contain smooth terms).
- "zero_inflated" (or "zi"): returns the zero-inflation component.
- "dispersion": returns the dispersion model component. This is common for models with zero-inflation or that can model the dispersion parameter.
- "instruments": for instrumental-variable or some fixed effects regression, returns the instruments.
- "location": returns location parameters such as conditional, zero_inflated, smooth_terms, or instruments (everything that are fixed or random effects - depending on the effects argument - but no auxiliary parameters).
- "distributional" (or "auxiliary"): components like sigma, dispersion, beta or precision (and other auxiliary parameters) are returned.

Examples

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

get_parameters.betamfx

Get model parameters from marginal effects models

Description

Returns the coefficients from a model.
Usage

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

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

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

data(mtcars)

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

generate_parameters(m)

Description

Returns the coefficients from a model.
get_parameters.betareg

### 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 **distributional**.

---

### Usage

#### S3 method for class 'betareg'

```r
get_parameters(
  x,
  component = c("all", "conditional", "precision", "location", "distributional", "auxiliary"),
  ...
)
```

#### S3 method for class 'DirichletRegModel'

```r
get_parameters(
  x,
  component = c("all", "conditional", "precision", "location", "distributional", "auxiliary"),
  ...
)
```

#### S3 method for class 'averaging'

```r
get_parameters(x, component = c("conditional", "full"), ...)
```

#### S3 method for class 'glmx'

```r
get_parameters(
  x,
  component = c("all", "conditional", "extra", "location", "distributional", "auxiliary"),
  ...
)
```

#### S3 method for class 'clm2'

```r
get_parameters(x, component = c("all", "conditional", "scale"), ...)
```

#### S3 method for class 'mvord'

```r
get_parameters(
  x,
  component = c("all", "conditional", "thresholds", "correlation"),
  ...
)
```

#### S3 method for class 'mjoint'

```r
get_parameters(x, component = c("all", "conditional", "survival"), ...)
```
get_parameters.BGGM

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

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

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

## S3 method for class 'BFBayesFactor'
get_parameters(
  x,
  effects = c("all", "fixed", "random"),
  component = c("all", "extra"),
  iterations = 4000,
  ...)
get_parameters.BGGM

    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",
  ...
)

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

## 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 \texttt{conditional}, \texttt{zero_inflated}, \texttt{smooth_terms}, or \texttt{instruments} are returned (everything that are fixed or random effects - depending on the \texttt{effects} argument - but no auxiliary parameters). For \texttt{component = "distributional"} (or "auxiliary"), components like \texttt{sigma}, \texttt{dispersion}, \texttt{beta} or \texttt{precision} (and other auxiliary parameters) are returned.

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

centrality Only for models with posterior samples, and when \texttt{summary = TRUE}. In this case, \texttt{centrality = "mean"} would calculate means of posterior samples for each parameter, while \texttt{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

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, ...)
get_parameters.gamm

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 parameters 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)

  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

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

### S3 method for class 'gam'

```r
get_parameters(
  x,
  component = c("all", "conditional", "smooth_terms", "location"),
  ...
)
```

### S3 method for class 'rqss'

```r
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.

#### Examples

```r
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.
get_parameters.glmm

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 'nlmerMod'
get_parameters(
  x,
  effects = c("fixed", "random"),
  component = c("all", "conditional", "nonlinear"),
  ...
)

## 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)

generate tests

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

Description

Returns the parameters from a hypothesis test.

Usage

## S3 method for class 'htest'
generate tests

Arguments

x A fitted model.

Value

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

Examples

generate tests
get_parameters.zeroinfl

Get model parameters from zero-inflated and hurdle models

Description

Returns the coefficients from a model.

Usage

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

## S3 method for class 'zcpglm'
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.
**get_predicted**

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

---

**get_predicted**  
*Model predictions (robust) and their confidence intervals*

**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.

`get_predicted_ci()` returns the confidence (or prediction) interval (CI) associated with predictions made by a model. This function can be called separately on a vector of predicted values. `get_predicted()` usually returns confidence intervals (included as attribute, and accessible via the `as.data.frame()` method) by default.

**Usage**

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

## Default S3 method:
```r
default <- function(x, data = NULL, predict = "expectation", ci = NULL, ci_type = "confidence", ci_method = NULL, dispersion_method = "sd", vcov = NULL, vcov_args = NULL, verbose = TRUE, ...)
```

## S3 method for class 'lm'
```r
default <- function(x, data = NULL, predict = "expectation", ...)  
```
get_predicted

```r
get_predicted(ci = NULL,
    iterations = NULL,
    verbose = TRUE,
    ...)

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

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

## S3 method for class 'lmerMod'
get_predicted(
    x,
    data = NULL,
    predict = "expectation",
    ci = NULL,
    ci_method = NULL,
    include_random = "default",
    iterations = NULL,
    verbose = TRUE,
    ...)

## S3 method for class 'principal'
get_predicted(x, data = NULL, ...)
```
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. Visualization matrices can be generated using `get_datagrid()`.

- **predict**
  string or NULL
  - "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).
  - "prediction" 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).
  - "classification" only differs from "prediction" for binomial models where it additionally transforms the predictions into the original response's type (for instance, to a factor).
  - Other strings are passed directly to the `type` argument of the `predict()` method supplied by the modelling package.
  - When `predict = NULL`, alternative arguments such as type will be captured by the ... ellipsis and passed directly to the `predict()` method supplied by the modelling package. Note that this might result in conflicts with multiple matching type arguments - thus, the recommendation is to use the predict argument for those values.
  - Notes: You can see the 4 options for predictions as on a gradient from "close to the model" to "close to the response data": "link", "expectation", "prediction", "classification". The predict argument modulates two things: the scale of the output and the type of certainty interval. Read more about in the Details section below.

- **ci**
  The interval level. Default is NULL, to be fast even for larger models. Set the interval level to an explicit value, e.g. 0.95, for 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 regressions). 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 `lm`), as for other models (e.g., `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]).
ci_method  The method for computing p values and confidence intervals. Possible values depend on model type.
- NULL uses the default method, which varies based on the model type.
- Most frequentist models: "wald" (default), "residual" or "normal".
- Bayesian models: "quantile" (default), "hdi", "eti", and "spi".
- Mixed effects lme4 models: "wald" (default), "residual", "normal", "satterthwaite", and "kenward-roger".

See get_df() for details.

dispersion_method
Bootstrap dispersion and Bayesian posterior summary: "sd" or "mad".

vcov  Variance-covariance matrix used to compute uncertainty estimates (e.g., for robust standard errors). This argument accepts a covariance matrix, a function which returns a covariance matrix, or a string which identifies the function to be used to compute the covariance matrix.
- A covariance matrix
- A function which returns a covariance matrix (e.g., stats::vcov())
- A string which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC
  - Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See ?sandwich::vcovBS
  - Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".

vcov_args  List of arguments to be passed to the function identified by the vcov argument.
This function is typically supplied by the sandwich or clubSandwich packages.
Please refer to their documentation (e.g., ?sandwich::vcovHAC) to see the list of available arguments. If no estimation type (argument type) is given, the default type for "HC" (or "vcovHC") equals the default from the sandwich package; for type "CR" (or "vcovCR"), the default is set to "CR3".

verbose  Toggle warnings.

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 by running as.data.frame(..., keep_iterations = TRUE) on the output.

include_random  If "default", include all random effects in the prediction, unless random effect variables are not in the data. If TRUE, include all random effects in the prediction (in this case, it will be checked if actually all random effect variables are in data). 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 data is provided, make sure to include the random effect variables in data 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 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).

• **Generalized 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:** When users set the predict argument to "expectation", the predictions are returned on the response scale, which is arguably the most convenient way to understand and visualize relationships of interest. When users set the predict argument to "link", predictions are returned on the link scale, and no transformation is applied. For instance, for a logistic regression 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 users select predict="classification" in binomial models, the get_predicted() function will first calculate predictions as if the user had selected predict="expectation". Then, it will round the responses in order to return the most likely outcome.

**Heteroscedasticity consistent standard errors:** The arguments vcov and vcov_args can be used to calculate robust standard errors for confidence intervals of predictions. These arguments, when provided in get_predicted(), are passed down to get_predicted_ci(), thus, see the related documentation there for more details.

**Bayesian and Bootstrapped models and iterations:** For predictions based on multiple iterations, for instance in the case of Bayesian models and bootstrapped predictions, the function used to compute the centrality (point-estimate predictions) can be modified via the centrality_function argument. For instance, get_predicted(model, centrality_function = stats::median). The default is mean. Individual draws can be accessed by running `iter <- as.data.frame(get_predicted(model))`, and their iterations can be reshaped into a long format by bayestestR::reshape_iterations(iter).

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_datagrid()

Examples

data(mtcars)
x <- lm(mpg ~ cyl + hp, data = mtcars)
predictions <- get_predicted(x, ci = 0.95)
predictions

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

# Get CI
as.data.frame(predictions)

if (require("boot")) {
  # Bootstrapped
  as.data.frame(generate(x, iterations = 4))
  # Same as as.data.frame(...) keeping iterations = FALSE
  summary(as.data.frame(generate(x, iterations = 4))"
}

# 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 <- generate(x, predict = "expectation", ci = 0.95)
head(as.data.frame(pred))

# Prediction: response scale + PI
pred <- generate(x, predict = "prediction", ci = 0.95)
head(as.data.frame(pred))

# Link: link scale + CI
pred <- generate(x, predict = "link", ci = 0.95)
head(as.data.frame(pred))

# Classification: classification "type" + PI
pred <- generate(x, predict = "classification", ci = 0.95)
head(as.data.frame(pred))
Description

Confidence intervals around predicted values

Usage

get_predicted_ci(x, ...)

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

predictions
  A vector of predicted values (as obtained by stats::fitted(), stats::predict() or
  get_predicted()).

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. Visualization matrices can be
  generated using get_datagrid().

se Numeric vector of standard error of predicted values. If NULL, standard errors
  are calculated based on the variance-covariance matrix.

ci The interval level. Default is NULL, to be fast even for larger models. Set the
  interval level to an explicit value, e.g. 0.95, for 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 param-
  eters (and gives the range of the link; for instance of the regression line in a linear
  regressions). 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 in-
  tervals 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 lm), as for other models (e.g., 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]).

**ci_method**  
The method for computing p values and confidence intervals. Possible values depend on model type.
- NULL uses the default method, which varies based on the model type.
- Most frequentist models: "wald" (default), "residual" or "normal".
- Bayesian models: "quantile" (default), "hdi", "eti", and "spi".
- Mixed effects **lme4** models: "wald" (default), "residual", "normal", "satterthwaite", and "kenward-roger".

See `get_df()` for details.

**dispersion_method**  
Bootstrap dispersion and Bayesian posterior summary: "sd" or "mad".

**vcov**  
Variance-covariance matrix used to compute uncertainty estimates (e.g., for robust standard errors). This argument accepts a covariance matrix, a function which returns a covariance matrix, or a string which identifies the function to be used to compute the covariance matrix.
- A covariance matrix
- A function which returns a covariance matrix (e.g., `stats::vcov()`)
- A string which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See `?sandwich::vcovHC`  
  - Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See `?sandwich::vcovBS`  
  - Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".

**vcov_args**  
List of arguments to be passed to the function identified by the vcov argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., `?sandwich::vcovHAC`) to see the list of available arguments. If no estimation type (argument type) is given, the default type for "HC" (or "vcovHC") equals the default from the **sandwich** package; for type "CR" (or "vcovCR"), the default is set to "CR3".

**verbose**  
Toggle warnings.

**Examples**

```r
# Confidence Intervals for Model Predictions
# ------------------------------------------

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)

head(ci_vals)

if (require("boot")) {
  predictions <- get_predicted(x, iterations = 500)
  get_predicted_ci(x, predictions)
}

if (require("datawizard") && require("bayestestR")) {
  ci_vals <- get_predicted_ci(x, predictions, ci = c(0.80, 0.95))
  head(ci_vals)
  datawizard::reshape_ci(ci_vals)
}

# Logistic model
# --------------

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

head(ci_vals)

head(ci_vals)

get_predictors

Get the data from model predictors

Description

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

Usage

get_predictors(x, verbose = TRUE)
get_priors

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

get_priors(x, ...)

## S3 method for class 'brmsfit'
get_priors(x, verbose = TRUE, ...)

Arguments

x A Bayesian model.
...
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

## Not run:
library(rstanarm)
model <- stan_glmer(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

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

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")
get_response

Get the values from the response variable

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,
  as_proportion = TRUE,
  source = "environment",
  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.

as_proportion Logical, if TRUE and the response value is a proportion (e.g. y1 / y2), then the returned response value will be a vector with the result of this proportion. Else, always a data frame is returned.

source String, indicating from where data should be recovered. If source = "environment" (default), data is recovered from the environment (e.g. if the data is in the workspace). This option is usually the fastest way of getting data and ensures that the original variables used for model fitting are returned. Note that always the current data is recovered from the environment. Hence, if the data was modified after model fitting (e.g., variables were recoded or rows filtered), the returned data may no longer equal the model data. If source = "frame" (or "mf"), the data is taken from the model frame. Any transformed variables are back-transformed, if possible. This option returns the data even if it is not available in the environment, however, in certain edge cases back-transforming to the original data may fail. If source = "environment" fails to recover the data, it tries to extract the data from the model frame; if source = "frame" and data cannot be extracted from the model frame, data will be recovered from the environment. Both ways only returns observations that have no missing data in the variables used for model fitting.

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
data <<- cbpp

  m <- glm(cbind(incidence, trials) ~ period, data = dat, 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**

```r
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**


**Examples**

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

---

**Description**

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

**Usage**

```r
get_statistic(x, ...)
```

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

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

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

```r
## S3 method for class 'betamfx'
get_statistic(
```
```r
```
get_statistic

\[x,\]
\[
  \text{component} = \text{c("all", "conditional", "precision", "marginal")},\
  \]
\[
  \ldots\
  \]

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

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

### S3 method for class 'emmGrid'
get_statistic(x, ci = 0.95, adjust = "none", merge_parameters = FALSE, \ldots)

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

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

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

**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.
get_transformation

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

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

get_transformation Return function of transformed response variables

Description
This functions checks whether any transformation, such as log- or exp-transforming, was applied to the response variable (dependent variable) in a regression formula, and returns the related function that was used for transformation.

Usage
```r
get_transformation(x)
```

Arguments
- `x` A regression model.

Value
A list of two functions: `$transformation`, the function that was used to transform the response variable; `$inverse`, the inverse-function of `$transformation` (can be used for "back-transformation"). If no transformation was applied, both list-elements `$transformation` and `$inverse` just return function(x) x. If transformation is unknown, NULL is returned.

Examples
```r
# identity, no transformation
model <- lm(Sepal.Length ~ Species, data = iris)
get_transformation(model)

# log-transformation
model <- lm(log(Sepal.Length) ~ Species, data = iris)
get_transformation(model)

# log-function
get_transformation(model)$transformation(.3) log(.3)

# inverse function is exp()
get_transformation(model)$inverse(.3) exp(.3)
```
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, vcov = NULL, vcov_args = NULL, ...)

## S3 method for class 'nestedLogit'
get_varcov(x, component = "all",
            verbose = TRUE,
            vcov = NULL,
            vcov_args = NULL,
            ...
)

## S3 method for class 'betareg'
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"), verbose = TRUE, ...)

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

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

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

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

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

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

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

Arguments

x A model.

... Currently not used.

verbose Toggle warnings.

vcov Variance-covariance matrix used to compute uncertainty estimates (e.g., for robust standard errors). This argument accepts a covariance matrix, a function which returns a covariance matrix, or a string which identifies the function to be used to compute the covariance matrix.

• A covariance matrix
• A function which returns a covariance matrix (e.g., stats::vcov())
• A string which indicates the kind of uncertainty estimates to return.
  – Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC
  – Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See ?sandwich::vcovBS
  – Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".

vcov_args List of arguments to be passed to the function identified by the vcov argument. This function is typically supplied by the sandwich or clubSandwich packages. Please refer to their documentation (e.g., ?sandwich::vcovHAC) to see the list of available arguments. If no estimation type (argument type) is given, the default type for "HC" (or "vcovHC") equals the default from the sandwich package; for type "CR" (or "vcoCR"), the default is set to "CR3".

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.

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.

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

Value

The variance-covariance matrix, as matrix-object.

Note

get_varcov() tries to return the nearest positive definite matrix in case of negative eigenvalues of the variance-covariance matrix. This ensures that it is still possible, for instance, to calculate standard errors of model parameters. A message is shown when the matrix is negative definite and a corrected matrix is returned.

Examples

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

# vcov of zero-inflation component from hurdle-model
if (require("pscl")) {
data("bioChemists", package = "pscl")
mod <- hurdle(art ~ phd + fem | ment, data = bioChemists, dist = "negbin")
get_variance

get_varcov(mod, component = "zero_inflated")
}

# robust vcov of, count component from hurdle-model
if (require("pscl") & & require("sandwich")) {
  data("bioChemists", package = "pscl")
  mod <- hurdle(art ~ phd + fem | ment, data = bioChemists, dist = "negbin")
  get_varcov(
    mod,
    component = "conditional",
    vcov = "BS",
    vcov_args = list(R = 50)
  )
}

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

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_variance

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 com-
ponents, 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
$\beta \ast X$ (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 ran-
dom 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, $\sigma^2$, 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 $\sigma^2$ indicates the within-subject variance.
**Random slope variance:** The random slope variance \((\tau_{11})\) is obtained from \texttt{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 \texttt{VarCorr()} . This measure is only available for mixed models with random intercepts and slopes.

### Value

A list with following elements:

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

### Note

This function supports models of class \texttt{merMod} (including models from \texttt{blme} , \texttt{clmm} , \texttt{cpglmm} , \texttt{glmmadmb} , \texttt{glmmTMB} , \texttt{MixMod} , \texttt{lme} , \texttt{mixed} , \texttt{rlmerMod} , \texttt{stanreg} , \texttt{brmsfit} or \texttt{wbm} ). Support for objects of class \texttt{MixMod} (\texttt{GLMMadaptive} ), \texttt{lme} (\texttt{nlme} ) or \texttt{brmsfit} (\texttt{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_weights

Get the values from model 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. 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

data(mtcars)
set.seed(123)
mtcars$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)
has_intercept

Checks if model has an intercept

Description

Checks if model has an intercept.

Usage

has_intercept(x, verbose = TRUE)

Arguments

x

A model object.

verbose

Toggle warnings.

Value

TRUE if x has an intercept, FALSE otherwise.

Examples

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_converged

Convergence test for mixed effects models

Description

is_converged() provides an alternative convergence test for merMod-objects.

Usage

is_converged(x, tolerance = 0.001, ...)

Arguments

x A `merMod` or `glmmTMB`-object.
tolerance Indicates up to which value the convergence result is accepted. The smaller tolerance is, the stricter the test will be.

Details

**Convergence and log-likelihood:** Convergence problems typically arise when the model hasn’t converged to a solution where the log-likelihood has a true maximum. This may result in unreliable and overly complex (or non-estimable) estimates and standard errors.

**Inspect model convergence:** `lme4` performs a convergence-check (see `?lme4::convergence`), however, as as discussed [here](#) and suggested by one of the lme4-authors in [this comment](#), this check can be too strict. `is_converged()` thus provides an alternative convergence test for `merMod`-objects.

**Resolving convergence issues:** Convergence issues are not easy to diagnose. The help page on `?lme4::convergence` provides most of the current advice about how to resolve convergence issues. Another clue might be large parameter values, e.g. estimates (on the scale of the linear predictor) larger than 10 in (non-identity link) generalized linear model might indicate complete separation, which can be addressed by regularization, e.g. penalized regression or Bayesian regression with appropriate priors on the fixed effects.

**Convergence versus Singularity:** Note the different meaning between singularity and convergence: singularity indicates an issue with the “true” best estimate, i.e. whether the maximum likelihood estimation for the variance-covariance matrix of the random effects is positive definite or only semi-definite. Convergence is a question of whether we can assume that the numerical optimization has worked correctly or not.

Value

TRUE if convergence is fine and FALSE if convergence is suspicious. Additionally, the convergence value is returned as attribute.

Examples

```r
if (require("lme4")) {
  data(cbpp)
  set.seed(1)
  cbpp$x <- rnorm(nrow(cbpp))
  cbpp$x2 <- runif(nrow(cbpp))

  model <- glmer(
    cbind(incidence, size - incidence) ~ period + x + x2 + (1 + x | herd),
    data = cbpp,
    family = binomial()
  )

  is_converged(model)
}
```
is_empty_object

## Not run:
if (require("glmmTMB")) {
  model <- glmmTMB(Sepal.Length ~ poly(Petal.Width, 4) * poly(Petal.Length, 4) +
                   (1 + poly(Petal.Width, 4) | Species), data = iris)

  is_converged(model)
}

## End(Not run)

---

### is_empty_object

Check if object is empty

#### Description

Check if object is empty

#### Usage

`is_empty_object(x)`

#### Arguments

- `x`: A list, a vector, or a dataframe.

#### Value

A logical indicating whether the entered object is empty.

#### Examples

```r
is_empty_object(c(1, 2, 3, NA))
is_empty_object(list(NULL, c(NA, NA)))
is_empty_object(list(NULL, NA))
```
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
is_gam_model(x)

Arguments
x A model object.

Value
A logical, TRUE if x is a generalized additive model and has smooth-terms

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
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)
}

is_mixed_model

Checks if a model is a mixed effects model

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

Usage
is_mixed_model(x)
is_model

Arguments

x 
A model object.

Value

A logical, TRUE if x is a mixed model.

Examples

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

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

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

is_model_supported(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 a regression model object is 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

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)

# to see all supported models
supported_models()
```
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
is_multivariate(x)

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 <- suppressWarnings(stan_mvmer(
  formula = list(
    logBili ~ year + (1 | id),
    albumin ~ sex + year + (year | id)
  ),
  data = pbcLong,
  chains = 1, cores = 1, seed = 12345, iter = 1000,
  show_messages = FALSE, refresh = 0
))

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.

Details

The term "nested" here means that all the fixed predictors of a model are contained within the fixed predictors of a larger model (sometimes referred to as the encompassing model). Currently, is_nested_models() ignores random effects parameters.

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**

```r
is_nullmodel(x)
```

**Arguments**

- `x` A model object.

**Value**

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

**Examples**

```r
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

link_function(x, ...)

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

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

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

## S3 method for class 'DirichletRegModel'
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

# 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)
Usage

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
- is_probit: model has probit link
- is_linear: family is gaussian
- is_tweedie: family is tweedie
- is_ordinal: family is ordinal or cumulative link
- is_cumulative: family is ordinal or cumulative link
- is_multinomial: family is multinomial or categorical link
- is_categorical: family is categorical link
- is_censored: model is a censored model (has a censored response, including survival models)
- is_truncated: model is a truncated model (has a truncated response)
- is_survival: model is a survival model
- is_zero_inflated: model has zero-inflation component
- is_hurdle: model has zero-inflation component and is a hurdle-model (truncated family distribution)
is_dispersion: model has dispersion component (not only dispersion parameter)
is_mixed: model is a mixed effects model (with random effects)
is_multivariate: model is a multivariate response model (currently only works for brmsfit objects)
is_trial: model response contains additional information about the trials
is_bayesian: model is a Bayesian model
is_gam: model is a generalized additive model
is_anova: model is an Anova object
is_ttest: model is an an object of class htest, returned by t.test()
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_variancetest: model is an an object of class htest, returned by bartlett.test(), shapiro.test() or car::leveneTest().
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_propgtest: 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: name of the distributional family of the model. For some exceptions (like some htest objects), can also be the name of the test.
n_obs: number of observations
n_grouplevels: for mixed models, returns names and numbers of random effect groups

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

Examples

```r
ldose <- rep(0:5, 2)
umdead <- c(1, 4, 9, 13, 18, 20, 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)
# Not run:
library(glmmTMB)
data("Salamanders")
```
```
m <- glmmTMB(
  count ~ spp + cover + mined + (1 | site),
  ziformula = ~ spp + mined,
  dispformula = ~ DOY,
  data = Salamanders,
  family = nbinom2
)

## End(Not run)

model_info(m)
```

<table>
<thead>
<tr>
<th>model_name</th>
<th>Name the model</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```
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**

```
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, \ldots)

Arguments

model A (mixed effects) model.
verbose Toggle off warnings.
\ldots 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_grouplevels

Count number of random effect levels in a mixed model

Description

Returns the number of group levels of random effects from mixed models.

Usage

n_grouplevels(x, \ldots)
Arguments

x  A mixed model.

... Additional arguments that can be passed to the function. Currently, you can use data to provide the model data, if available, to avoid retrieving model data multiple times.

Value

The number of group levels in the model.

Examples

```r
if (require("lme4")) {
  data(sleepstudy)
  set.seed(12345)
  sleepstudy$grp <- sample(1:5, size = 180, replace = TRUE)
  sleepstudy$subgrp <- NA
  for (i in 1:5) {
    filter_group <- sleepstudy$grp == i
    sleepstudy$subgrp[filter_group] <- sample(1:30, size = sum(filter_group), replace = TRUE)
  }
  model <- lmer(
    Reaction ~ Days + (1 | grp / subgrp) + (1 | Subject),
    data = sleepstudy
  )
  n_grouplevels(model)
}
```

Usage

```r
n_obs(x, ...)
```

## S3 method for class 'glm'
```r
n_obs(x, disaggregate = FALSE, ...)
```

## S3 method for class 'svyolr'
```r
n_obs(x, weighted = FALSE, ...)
```

## S3 method for class 'afex_aov'
```r
n_obs(x, shape = c("long", "wide"), ...)
```
n_parameters

## S3 method for class 'stanmvreg'
n_obs(x, select = NULL, ...)

Arguments

x
A fitted model.

... Currently not used.

disaggregate For binomial models with aggregated data, n_obs() returns the number of data rows by default. If disaggregate = TRUE, the total number of trials is returned instead (determined by summing the results of weights() for aggregated data, which will be either the weights input for proportion success response or the row sums of the response matrix if matrix response, see ’Examples’).

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)

if (require("lme4")) {
  data(cbpp, package = "lme4")
m <- glm(
  cbind(incidence, size - incidence) ~ period,
data = cbpp,
family = binomial(link = "logit")
)
n_obs(m)
n_obs(m, disaggregate = TRUE)
}

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 formula, 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)
```

---

**object_has_names**  
*Check names and rownames*

**Description**

object_has_names() checks if specified names are present in the given object. object_has_rownames() checks if rownames are present in a dataframe.

**Usage**

```r
object_has_names(x, names)
object_has_rownames(x)
```

**Arguments**

- **x**  A named object (an atomic vector, a list, a dataframe, etc.).
- **names**  A single character or a vector of characters.

**Value**

A logical or a vector of logicals.
Examples

# check if specified names are present in the given object
object_has_names(mtcars, "am")
object_has_names(anscombe, c("x1", "z1", "y1"))
object_has_names(list("x" = 1, "y" = 2), c("x", "a"))

# check if a dataframe has rownames
object_has_rownames(mtcars)

print_color

Coloured console output

Description

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

Usage

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 "white", "black", "red", "yellow", "green", "blue", "violet", "cyan" or "grey". Bright variants of colors are available by adding the prefix "b" (or "br_" or "bright_"), e.g. "bred" (or "br_red" resp. "bright_red"). Background colors can be set by adding the prefix "bg_" (e.g. "bg_red"). 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 dabedi dabedei", "blue")
```

print_parameters

Prepare summary statistics of model parameters for printing

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.
- **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()`.
print_parameters

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 the 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)
```

### standardize_column_order

**Standardize column order**

#### Description

Standardizes order of columns for dataframes and other objects from *easystats* and *broom* ecosystem packages.

#### Usage

```r
standardize_column_order(data, ...)
```

- **S3 method for class 'parameters_model'**

```r
standardize_column_order(data, style = c("easystats", "broom"), ...)
```

#### 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.
Value

A data frame, with standardized column order.

Examples

```r
# easystats conventions
df1 <- cbind.data.frame(
  CI_low = -2.873,
  t     = 5.494,
  CI_high = -1.088,
  p     = 0.00001,
  Parameter = -1.980,
  CI     = 0.95,
  df     = 29.234,
  Method = "Student's t-test"
)

standardize_column_order(df1, style = "easystats")

# broom conventions
df2 <- cbind.data.frame(
  conf.low = -2.873,
  statistic = 5.494,
  conf.high = -1.088,
  p.value  = 0.00001,
  estimate  = -1.980,
  conf.level = 0.95,
  df        = 29.234,
  method    = "Student's t-test"
)

standardize_column_order(df2, style = "broom")
```

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

```r
standardize_names(data, ...)

# 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")
}
```
**text_remove_backticks**  
*Remove backticks from a string*

**Description**

This function removes backticks from a string.

**Usage**

```r
text_remove_backticks(x, ...)  
```

## S3 method for class 'data.frame'

```r
text_remove_backticks(x, column = "Parameter", verbose = FALSE, ...)
```

**Arguments**

- `x`  
  A character vector, a data frame or a matrix. If a matrix, backticks are removed from the column and row names, not from values of a character vector.

- `...`  
  Currently not used.

- `column`  
  If `x` is a data frame, specify the column of character vectors, where backticks should be removed. If NULL, all character vectors are processed.

- `verbose`  
  Toggle warnings.

**Value**

- `x`, where all backticks are removed.

**Note**

If `x` is a character vector or data frame, backticks are removed from the elements of that character vector (or character vectors from the data frame.) If `x` is a matrix, the behaviour slightly differs: in this case, backticks are removed from the column and row names. The reason for this behaviour is that this function mainly serves formatting coefficient names. For `vcov()` (a matrix), row and column names equal the coefficient names and therefore are manipulated then.

**Examples**

```r
# example model
data(iris)
iris$'a m' <- iris$Species
iris$'Sepal Width' <- iris$Sepal.Width
model <- lm('Sepal Width' ~ Petal.Length + 'a m', data = iris)

# remove backticks from string
names(coef(model))
text_remove_backticks(names(coef(model)))
```
# remove backticks from character variable in a data frame
# column defaults to "Parameter".
d <- data.frame(
  Parameter = names(coef(model)),
  Estimate = unname(coef(model))
)
d
text_remove_backticks(d)

trim_ws

Small helper functions

Description

Collection of small helper functions. 
trim_ws() is an efficient function to trim leading and trailing whitespaces from character vectors or strings. n_unique() returns the number of unique values in a vector. has_single_value() is equivalent to n_unique() == 1 but is faster. safe_deparse() is comparable to deparse1(), i.e. it can safely deparse very long expressions into a single string. safe_deparse_symbol() only deparses a substituted expression when possible, which can be much faster than deparse(substitute()) for those cases where substitute() returns no valid object name.

Usage

trim_ws(x, ...)

## S3 method for class 'data.frame'
trim_ws(x, character_only = TRUE, ...)

n_unique(x, ...)

## Default S3 method:
n_unique(x, na.rm = TRUE, ...)

safe_deparse(x, ...)

safe_deparse_symbol(x)

has_single_value(x, na.rm = FALSE)

Arguments

x A (character) vector, or for some functions may also be a data frame.
...

Currently not used.
character_only Logical, if TRUE and x is a data frame or list, only processes character vectors.
na.rm    Logical, if missing values should be removed from the input.
Value

- \texttt{n_unique()}: For a vector, \texttt{n_unique} always returns an integer value, even if the input is \texttt{NULL} (the return value will be 0 then). For data frames or lists, \texttt{n_unique()} returns a named numeric vector, with the number of unique values for each element.
- \texttt{has_single_value()}: \texttt{TRUE} if \texttt{x} has only one unique value, \texttt{FALSE} otherwise.
- \texttt{trim_ws()}: A character vector, where trailing and leading white spaces are removed.
- \texttt{safe_deparse()}: A character string of the unevaluated expression or symbol.
- \texttt{safe_deparse_symbol()}: A character string of the unevaluated expression or symbol, if \texttt{x} was a symbol. If \texttt{x} is no symbol (i.e. if \texttt{is.name(x)} would return \texttt{FALSE}), \texttt{NULL} is returned.

Examples

\begin{verbatim}
trim_ws(" no space! ")
n_unique(iris$Species)
has_single_value(c(1, 1, 2))

# safe_deparse_symbol() compared to deparse(substitute())
safe_deparse_symbol(as.name("test"))
deparse(substitute(as.name("test")))
\end{verbatim}
Index

* data
  fish, 45

all_models_equal, 5
all_models_same_class
  (all_models_equal), 5

Bayesian models, 22, 74
bayestestR::ci(), 66
bayestestR::describe_posterior(), 131, 132
bayestestR::hdi(), 131
bayestestR::weighted_posteriors(), 81

check_if_installed, 6
clean_names, 7
clean_parameters, 8
clean_parameters(), 131, 132
color_if, 9
color_text (print_color), 130
color_theme (print_color), 130
colour_if (color_if), 9
colour_text (print_color), 130
compact_character, 11
compact_list, 11
display, 12
download_model, 13
effectsize::effectsize(), 133, 135
ellipsis_info, 13
Estimated marginal means, 22, 74
export_table, 14
export_table(), 131, 132
find_algorithm, 17
find_formula, 18
find_interactions, 20
find_offset, 21
find_parameters, 22
find_parameters(), 34
find_parameters.averaging, 23
find_parameters.bamlss
  (find_parameters.BGGM), 26
find_parameters.bayesx
  (find_parameters.BGGM), 26
find_parameters.betamfx, 25
find_parameters.betareg
  (find_parameters.averaging), 23
find_parameters.BFBayesFactor
  (find_parameters.BGGM), 26
find_parameters.BGGM, 26
find_parameters.BGGM(), 60
find_parameters.brmsfit
  (find_parameters.BGGM), 26
find_parameters.DirichletRegModel
  (find_parameters.averaging), 23
find_parameters.emmGrid, 29
find_parameters.gam
  (find_parameters.gamlss), 30
find_parameters.gamlss, 30
find_parameters.glmmTMB, 31
find_parameters.glmx
  (find_parameters.averaging), 23
find_parameters.logitmfx
  (find_parameters.betamfx), 25
find_parameters.MCMCglmm
  (find_parameters.BGGM), 26
find_parameters.merMod
  (find_parameters.glmmTMB), 31
find_parameters.mhurdle
  (find_parameters.zeroinfl), 32
find_parameters.mjoint
  (find_parameters.averaging), 23
find_parameters.nlmerMod
  (find_parameters.glmmTMB), 31
find_parameters.sim.merMod
  (find_parameters.BGGM), 26
find_parameters.stanreg
  (find_parameters.BGGM), 26
find_parameters.zeroinfl, 32
find_predictors, 34
find_random, 36
find_random_slopes, 37
find_response, 37
find_smooth, 38
find_statistic, 39
find_terms, 40
find_terms(), 43
find_transformation, 41
find_variables, 42
find_variables(), 40
find_weights, 44
fish, 45
format_alert (format_message), 48
format_bf, 45
format_capitalize, 46
format_ci, 46
format_error (format_message), 48
format_message, 48
format_number, 50
format_p, 51
format_pd, 52
format_percent (format_value), 57
format_rope, 53
format_table, 55
format_value, 57
format_value(), 50
format_warning (format_message), 48
formula_ok (find_formula), 18

Generalized additive models, 22, 74
get_auxiliary, 59
get_call, 60
get_correlation_slope_intercept (get_variance), 107
get_correlation_slopes (get_variance), 107
get_data, 61
get_datagrid, 63
get_datagrid(), 89, 92, 93
get_deviance, 68
get_df, 69
get_dfs, 90, 94
get_family, 71
get_intercept, 71
get_loglikelihood, 72
get_modelmatrix, 73
get_parameters, 74
get_parameters.averaging
  (get_parameters.betareg), 76
get_parameters.bamlss
  (get_parameters.BGGM), 78
get_parameters.bayesx
  (get_parameters.BGGM), 78
get_parameters.betamfx, 75
get_parameters.betareg, 76
get_parameters.BFBayesFactor
  (get_parameters.BGGM), 78
get_parameters.BGGM, 78
get_parameters.brmsfit
  (get_parameters.BGGM), 78
get_parameters.clm2
  (get_parameters.betareg), 76
get_parameters.coxme
  (get_parameters.glmm), 83
get_parameters.DirichletRegModel
  (get_parameters.betareg), 76
get_parameters.emm_list
  (get_parameters.emmGrid), 81
get_parameters.emmGrid, 81
get_parameters.gam
  (get_parameters.gamm), 82
get_parameters.gamm, 82
get_parameters.glimML
  (get_parameters.glmm), 83
get_parameters.glmmTMB
  (get_parameters.glmm), 83
get_parameters.slmx
  (get_parameters.betareg), 76
get_parameters.htest, 85
get_parameters.logitmfx
  (get_parameters.betamfx), 75
get_parameters.MCMCglmm
  (get_parameters.BGGM), 78
get_parameters.merMod
  (get_parameters.glmm), 83
get_parameters.mhurdle
  (get_parameters.zeroinfl), 86
get_parameters.mjoint
  (get_parameters.betareg), 76
get_parameters.mvord
  (get_parameters.betareg), 76
get_parameters.nlmerMod
  (get_parameters.glmm), 83
INDEX

get_parameters.rqss
  (get_parameters.gamm), 82
get_parameters.sim
  (get_parameters.BGGM), 78
get_parameters.stanmvreg
  (get_parameters.BGGM), 78
get_parameters.stanreg
  (get_parameters.BGGM), 78
get_parameters.zcpglm
  (get_parameters.zeroinfl), 86
get_parameters.zeroinfl
  (get_parameters.zeroinfl), 86
get_predicted
  (get_predicted), 63, 67, 93
get_predicted_ci
  (get_predicted), 92
get_predictors
  (get_predictors), 95
get_priors
  (get_priors), 96
get_random
  (get_random), 97
get_residuals
  (get_residuals), 98
get_response
  (get_response), 99
get_sigma
  (get_sigma), 100
get_sigma()
  (get_sigma), 100
get_statistic
  (get_statistic), 101
get_transformation
  (get_transformation), 103
get_varcov
  (get_varcov), 104
get_variance
  (get_variance), 107
get_variance_dispersion
  (get_variance), 107
get_variance_distribution
  (get_variance), 107
get_variance_fixed
  (get_variance), 107
get_variance_intercept
  (get_variance), 107
get_variance_random
  (get_variance), 107
get_variance_residual
  (get_variance), 107
get_variance_slope
  (get_variance), 107
get_weights
  (get_weights), 110
has_intercept
  (has_intercept), 111
has_single_value
  (has_single_value), 111
Hypothesis tests
  (Hypothesis tests), 74
IQR()
  (IQR()), 66
is_converged
  (is_converged), 111
is_empty_object
  (is_empty_object), 113
is_gam_model
  (is_gam_model), 114
is_mixed_model
  (is_mixed_model), 114
is_model
  (is_model), 115
is_model_supported
  (is_model_supported), 116
is_multivariate
  (is_multivariate), 117
is_nested_models
  (is_nested_models), 118
is_nullmodel
  (is_nullmodel), 119
is_regression_model
  (is_regression_model), 115
link_function
  (link_function), 119
link_inverse
  (link_inverse), 120
loglikelihood
  (loglikelihood), 72
Marginal effects models
  (Marginal effects models), 22, 74
Mixed models
  (Mixed models), 22, 74
model_info
  (model_info), 121
model_name
  (model_name), 124
Models with special components
  (Models with special components), 22, 74
n_grouplevels
  (n_grouplevels), 125
n_obs
  (n_obs), 126
n_parameters
  (n_parameters), 127
n_unique
  (n_unique), 137
null_model
  (null_model), 125
object_has_names
  (object_has_names), 129
object_has_rownames
  (object_has_rownames), 129
parameters::model_parameters()
  (parameters::model_parameters()), 131–135
performance::check_singularity()
  (performance::check_singularity()), 108
predict()
  (predict()), 87
print_color
  (print_color), 130
print_colour
  (print_colour), 130
print_html
  (print_html), 12
print_md
  (print_md), 12
print_parameters
  (print_parameters), 131
print_parameters()
  (print_parameters()), 8
safe_deparse
  (safe_deparse), 137
safe_deparse_symbol
  (safe_deparse_symbol), 137
signif()
  (signif()), 15, 47, 56, 58
standardize_column_order
  (standardize_column_order), 133
standardize_names
  (standardize_names), 134
stats::df.residual()
  (stats::df.residual()), 69
supported_models
  (supported_models), 116
text_remove_backticks
  (text_remove_backticks), 136
trim_ws
  (trim_ws), 137
Zero-inflated and hurdle models
  (Zero-inflated and hurdle models), 22, 74