Package 'insight'

April 20, 2020

Type Package

Title Easy Access to Model Information for Various Model Objects

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.

Version 0.8.3

Maintainer Daniel Lüdecke <d.luedecke@uke.de>

License GPL-3

URL https://easystats.github.io/insight/

BugReports https://github.com/easystats/insight/issues

Depends R (>= 3.2)

Imports methods, stats, utils

Suggests AER, afex, aod, BayesFactor, bayestestR, betareg, biglm, blavaan, blme, brms, censReg, cgam, coxme, cplm, crch, estimatr, feisr, fixest, gam, gamm4, gamlss, gbm, gee, geepack, GLMMadaptive, glmmTMB, gmnl, HRQoL, httr, lavaan, lfe, logistf, MASS, Matrix, MCMCglmm, mlogit, multgee, lme4, mgcv, nnet, nlme, ordinal, panelr, plm, pscl, quantreg, rms, robustbase, robustlmm, rstanarm, rstudioapi, speedglm, splines, statmod, survey, survival, tripack, truncreg, testthat, VGAM, knitr, rmarkdown, spelling

Encoding UTF-8

RoxygenNote 7.1.0

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>),
Dominique Makowski [aut, ctb] (<https://orcid.org/0000-0001-5375-9967>),
Indrajeet Patil [aut, ctb] (<https://orcid.org/0000-0003-1995-6531>),
Philip Waggoner [aut, ctb] (<https://orcid.org/0000-0002-7825-7573>)

Repository CRAN

Date/Publication 2020-04-20 08:50:02 UTC

R topics documented:

`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

```
library(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)
```

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

```
clean_names(x)
```

Arguments

x	A fitted model, or a character vector.

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

```
# example from ?stats::glm
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- c(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()"
library(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

```
clean_parameters(x, ...)
```

Arguments

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

Details

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

Value

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

Examples

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

## End(Not run)
```

color_if *Color-formatting for data columns based on condition*

Description

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

Usage

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

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

Arguments

x	A data frame
columns	Character vector with column names of x that should be formatted.
predicate	A function that takes columns and value as input and which should return TRUE or FALSE, based on if the condition (in comparison with value) is met.
value	The comparator. May be used in conjunction with predicate to quickly set up a function which compares elements in colums 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 <- color_if(iris[1:10, ], columns = "Sepal.Length", predicate = `>`, value = 5)
x
cat(x$Sepal.Length)

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

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

---

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

```
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.

Value

A model from the *circus*-repository.

References

https://easystats.github.io/circus/

---

find_algorithm *Find sampling algorithm and optimizers*

Description

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

Usage

```
find_algorithm(x, ...)
```

Arguments

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

Value

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

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

For frequentist mixed models:

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

For Bayesian models:

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

Examples

```
library(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, ...).

Usage

```
find_formula(x, ...)
```

Arguments

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

Value

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

- `conditional`, the "fixed effects" part from the model. One exception are `DirichletRegModel` models from **DirichletReg**, which has two or three components, depending on `model`.
- `random`, the "random effects" part from the model (or the `id` for gee-models and similar)
- `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 formula
- `instruments`, for fixed-effects regressions like `ivreg`, `felm` or `plm`, the instrumental variables
- `cluster`, for fixed-effects regressions like `felm`, the cluster specification
- `correlation`, for models with correlation-component like `gls`, the formula that describes the correlation structure

- slopes, for fixed-effects individual-slope models like feis, the formula for the slope parameters
- precision, for DirichletRegModel models from **DirichletReg**, when parametrization (i.e. model) is "alternative".

Note

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

Examples

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

---

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. Duplicated values are removed.

Value

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

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

Examples

```
data(mtcars)

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

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

`find_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()`.

Usage

```
find_parameters(x, ...)

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

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

## S3 method for class 'zeroinfl'
find_parameters(
  x,
```

```
  component = c("all", "conditional", "zi", "zero_inflated"),
  flatten = FALSE,
  ...
)

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

## S3 method for class 'zcpglm'
find_parameters(
  x,
  component = c("all", "conditional", "zi", "zero_inflated"),
  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 'brmsfit'
find_parameters(
  x,
  effects = c("all", "fixed", "random"),
 component = c("all", "conditional", "zi", "zero_inflated", "dispersion", "simplex",
    "sigma", "smooth_terms"),
  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("all", "conditional", "smooth_terms"),
  flatten = FALSE,
  parameters = NULL,
  ...
)

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

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

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

## S3 method for class 'bayesx'
get_statistic(x, ...)
```

Arguments

x	A fitted model.
...	Currently not used.
component	Should all parameters, 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. 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.
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.

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. Not all model classes that support these arguments are listed here in the 'Usage' section.

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
- `dispersion`, the dispersion parameters
- `simplex`, simplex parameters of monotonic effects (**brms** only)
- `smooth_terms`, the smooth parameters

Examples

```
data(mtcars)
m <- lm(mpg ~ 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,
  effects = c("fixed", "random", "all"),
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion",
    "instruments", "correlation", "smooth_terms"),
  flatten = FALSE
)
```

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.

Value

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

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

Examples

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

---

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

```
library(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

```
library(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)
```

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'.

Value

The name(s) of the response variable(s) from x as character vector.

Examples

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

find_terms *Find all model terms*

Description

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

Usage

```
find_terms(x, 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.

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

Note

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

Examples

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

find_terms(m)
```

---

find_variables *Find names of all variables*

Description

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

Usage

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

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.

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

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

```
library(lme4)
data(cbpp)
data(sleepstudy)
# some data preparation...
cbpp$trials <- cbpp$size - cbpp$incidence
sleepstudy$mygrp <- sample(1:5, size = 180, replace = TRUE)
sleepstudy$mysubgrp <- NA
for (i in 1:5) {
  filter_group <- sleepstudy$mygrp == i
  sleepstudy$mysubgrp[filter_group] <-
    sample(1:30, size = sum(filter_group), replace = TRUE)
}

m1 <- glmer(
  cbind(incidence, size - incidence) ~ period + (1 | herd),
  data = cbpp,
  family = binomial
)
find_variables(m1)

m2 <- lmer(
  Reaction ~ Days + (1 | mygrp / mysubgrp) + (1 | Subject),
  data = sleepstudy
)
find_variables(m2)
find_variables(m2, flatten = 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_ci *Confidence/Credible Interval (CI) Formatting*

Description

Confidence/Credible Interval (CI) Formatting

Usage

```
format_ci(
  CI_low,
  CI_high,
  ci = 0.95,
  digits = 2,
  brackets = TRUE,
  width = NULL,
  width_low = width,
  width_high = width
)
```

Arguments

CI_low	Lower CI bound.
CI_high	Upper CI bound.
ci	CI level in percentage.
digits	Number of significant digits.
brackets	Logical, if TRUE (default), values are encompassed in square 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.

Value

A formatted string.

Examples

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

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

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

---

format_table *Dataframe and Tables Pretty Formatting*

Description

Dataframe and Tables Pretty Formatting

Usage

```
format_table(
  x,
  sep = " | ",
  header = "-",
```

```
  digits = 2,
  protect_integers = TRUE,
  missing = "",
  width = NULL
)
```

Arguments

x	A data frame.
sep	Column separator.
header	Header separator. Can be NULL.
digits	Number of significant digits.
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.

Value

A data frame in character format.

Examples

```
cat(format_table(iris))
cat(format_table(iris, sep = " ", header = "*", digits = 1))
```

---

format_value *Numeric Values Formatting*

Description

Numeric Values Formatting

Usage

```
format_value(
  x,
  digits = 2,
  protect_integers = FALSE,
  missing = "",
  width = NULL,
  as_percent = FALSE,
  ...
)
```

Arguments

x	Numeric value.
digits	Number of significant digits.
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.
...	Arguments passed to or from other methods.

Value

A formatted string.

Examples

```
format_value(1.20)
format_value(1.2)
format_value(1.2012313)
format_value(c(0.0045, 234, -23))
format_value(c(0.0045, .12, .34))
format_value(c(0.0045, .12, .34), as_percent = TRUE)

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

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

format_value(iris)
```

---

get_data *Get the data that was used to fit the model*

Description

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

Usage

```
get_data(x, ...)

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

```
## S3 method for class 'rqss'
get_data(x, component = c("all", "conditional", "smooth_terms"), ...)

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

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

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

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

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

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

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

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

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

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

```
## S3 method for class 'brmsfit'
get_data(
  x,
  effects = c("all", "fixed", "random"),
  component = c("all", "conditional", "zi", "zero_inflated"),
  ...
)

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

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

Arguments

x	A fitted model.
...	Currently not used.
effects	Should model data for fixed effects, random effects or both be returned? Only applies to mixed models.
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.

Value

The data that was used to fit the model.

Note

Unlike `model.frame()`, which may contain transformed variables (e.g. if `poly()` or `scale()` was used inside the formula to specify the model), `get_data()` aims at returning the "original", untransformed data.

Examples

```
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_parameters *Get model parameters*

Description

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

Usage

```
get_parameters(x, ...)

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

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

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

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

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

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

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

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

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

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

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

```
)

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

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

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

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

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

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

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

## 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 'aovlist'
get_parameters(x, effects = c("fixed", "random", "all"), ...)

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

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

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

## S3 method for class 'brmsfit'
get_parameters(
  x,
  effects = c("fixed", "random", "all"),
 component = c("all", "conditional", "zi", "zero_inflated", "dispersion", "simplex",
    "sigma", "smooth_terms"),
  parameters = NULL,
  ...
)

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

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

Arguments

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

component	Should all parameters, 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. Note that the *conditional* component is also called *count* or *mean* component, depending on the model.
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.
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.

`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 and 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**)
- for Bayesian models, the posterior samples from the requested parameters as data frame
- for Anova (`aov()`) with error term, a list of parameters for the conditional and the random effects parameters
- for models with smooth terms or zero-inflation component, a data frame with three columns: the parameter names, the related point estimates and the component

Examples

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

get_predictors *Get the data from model predictors*

Description

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

Usage

```
get_predictors(x)
```

Arguments

x	A fitted model.

Value

The data from all predictor variables, as data frame.

Examples

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

get_priors *Get summary of priors used for a model*

Description

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

Usage

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

Arguments

x	A Bayesian model.
...	Currently not used.

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_glm(Sepal.Width ~ Species * Petal.Length, data = iris)
get_priors(model)

## End(Not run)
```

`get_random` *Get the data from random effects*

Description

Returns the data from all random effects terms.

Usage

```
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

```
library(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_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)
```

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.

Value

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

Examples

```
library(lme4)
data(cbpp)
data(mtcars)
cbpp$trials <- cbpp$size - cbpp$incidence

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

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

get_statistic *Get statistic associated with estimates*

Description

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

Usage

```
get_statistic(x, ...)

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

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

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

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

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

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

Arguments

x	A model.
...	Currently not used.
column_index	For model objects that have no defined get_statistic() method yet, the default method is called. This method tries to extract the statistic column from coef(summary()), where the index of the column that is being pulled is column_index. Defaults to 3, which is the default statistic column for most models' summary-output.
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.
robust	Logical, if TRUE, test statistic based on robust standard errors is returned.

Value

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

Examples

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

---

get_varcov *Get variance-covariance matrix from models*

Description

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

Usage

```
get_varcov(x, ...)

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

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

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

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

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

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

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

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

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

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

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

Arguments

x	A model.
...	Currently not used.
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", "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.

Value

The variance-covariance matrix, as matrix-object.

Note

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

Examples

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

---

get_variance *Get variance components from random effects models*

Description

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

Usage

```
get_variance(
  x,
 component = c("all", "fixed", "random", "residual", "distribution", "dispersion",
    "intercept", "slope", "rho01"),
  verbose = TRUE,
  ...
)

get_variance_residual(x, ...)

get_variance_fixed(x, ...)

get_variance_random(x, ...)

get_variance_distribution(x, ...)

get_variance_dispersion(x, ...)

get_variance_intercept(x, ...)

get_variance_slope(x, ...)

get_correlation_slope_intercept(x, ...)
```

Arguments

x	A mixed effects model.
component	Character value, indicating the variance component that should be returned. By default, all variance components are returned. The distribution-specific ("distribution") and residual ("residual") variance are the most computational intensive components, and hence may take a few seconds to calculate.
verbose	Toggle off warnings.
...	Currently not used.

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, σ_f^2, is the variance of the matrix-multiplication $\beta * X$ (parameter vector by model matrix).

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

Distribution-specific variance: The distribution-specific variance, σ_d^2, depends on the model family. For Gaussian models, it is σ^2 (i.e. `sigma(model)^2`). For models with binary outcome, it is $\pi^2/3$ for logit-link and `1` for probit-link. For all other models, the distribution-specific variance is based on lognormal approximation, $log(1 + var(x)/\mu^2)$ (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, σ_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, σ_ϵ^2, is simply $\sigma_d^2 + \sigma_e^2$.

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

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

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

Value

A list with following elements:

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

Note

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

References

- Johnson, P. C. D. (2014). Extension of Nakagawa & Schielzeth's R2 GLMM to random slopes models. Methods in Ecology and Evolution, 5(9), 944–946. doi: 10.1111/2041210X.12225

- Nakagawa, S., Johnson, P. C. D., & Schielzeth, H. (2017). The coefficient of determination R2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. Journal of The Royal Society Interface, 14(134), 20170213. doi: 10.1098/rsif.2017.0213
- Zuur, A. F., Savel'ev, A. A., & Ieno, E. N. (2012). Zero inflated models and generalized linear mixed models with R. Newburgh, United Kingdom: Highland Statistics.

Examples

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

get_variance(m)
get_variance_fixed(m)
get_variance_residual(m)

## End(Not run)
```

---

get_weights *Get the values from model weights*

Description

Returns weighting variable of a model.

Usage

```
get_weights(x, ...)
```

Arguments

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

Value

The weighting variable, 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)
get_weights(m)
```

has_intercept *Checks if model has an intercept*

Description

Checks if model has an intercept.

Usage

```
has_intercept(x)
```

Arguments

x	A model object.

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)

library(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_model *Checks if an object is a regression model object*

Description

Small helper that checks if a model is a regression model object.

Usage

```
is_model(x)
```

Arguments

x	An object.

Details

This function returns TRUE if x is a model object.

Value

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

Examples

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

is_model(m)
is_model(mtcars)
```

---

is_model_supported *Checks if an object is a regression model object supported in* **insight** *package.*

Description

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

Usage

```
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

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

is_model_supported(m)
is_model_supported(mtcars)
```

`is_multivariate` *Checks if an object stems from a multivariate response model*

Description

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

Usage

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

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

## End(Not run)
```

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

Description

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

Usage

```
is_nullmodel(x)
```

Arguments

x	A model object.

Value

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

Examples

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

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

library(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 '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 '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)
```

---

model_info *Access information from model objects*

Description

Retrieve information from model objects.

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_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_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_anova`: model is an Anova object
- `link_function`: the link-function
- `family`: the family-object
- `n_obs`: number of observations
- `model_terms`: a list with all model terms, including terms such as random effects or from zero-inflated model parts.

Value

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

Examples

```
ldose <- rep(0:5, 2)
numdead <- 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)
```

---

null_model *Compute intercept-only model for mixed models*

Description

This function compute the null-model (i.e. `(y ~ 1)`) for the fixed effects part) of a random-intercept model.

Usage

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

Arguments

model	A mixed effects model.
verbose	Toggle off warnings.

Value

The null-model of x

Examples

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

---

n_obs *Get number of observations from a model*

Description

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

Usage

```
n_obs(x, ...)

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

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

Arguments

x	A fitted model.
...	Currently not used.
weighted	For survey designs, returns the weighted sample size.
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)
```

---

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)
```

Arguments

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

Details

This function prints text directly to the console using cat(), so no string is returned.

Value

Nothing.

Examples

```
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 hdi or equivalence_test, as input and splits this information into several parts, depending on the model. See details below.

Usage

```
print_parameters(
  x,
  ...,
  split_by = c("Effects", "Component", "Group", "Response")
)
```

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.

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 hdi, where we have a) a Parameters column and b) columns with the HDI values.

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 statistic of interest (in our example: the HDI) is 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

```
## 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 informaton 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)
```

Index