Package 'parameters'

September 26, 2019

Type Package

Title Processing of Model Parameters

Version 0.2.0

Maintainer Dominique Makowski <dom.makowski@gmail.com>

Description
Utilities for processing the parameters of various statistical models. Beyond computing p values, CIs, and other indices for a wide variety of models (see support list of insight; Lüdecke, Waggoner & Makowski (2019) <doi:10.21105/joss.01412>), this package implements features like standardization or bootstrapping of parameters and models, feature reduction (feature extraction and variable selection) as well as conversion between indices of effect size.

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

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

Depends R (>= 3.0)

Imports insight (>= 0.5.0), bayestestR (>= 0.3.0), methods, stats, tools, utils

Suggests AER, aod, BayesFM, BayesFactor, betareg, boot, brms, cAIC4, covr, dplyr, DRR, EGAnet (>= 0.7), FactoMineR, fastICA, gamlss, gee, geepack, GLMMadaptive, glmmTMB, knitr, lavaan, lme4, lmerTest, logspline, MASS, Matrix, MCMCglmm, mgcv, MuMIn, nFactors, nlme, panelr, pbkrtest, plm, projpred, pscl, psych, randomForest, rmarkdown, rstanarm, see, survey, survival, testthat, VGAM, Zelig

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Dominique Makowski [aut, cre] (<https://orcid.org/0000-0001-5375-9967>),
Daniel Lüdecke [aut] (<https://orcid.org/0000-0002-8895-3206>),
Mattan S. Ben-Shachar [aut] (<https://orcid.org/0000-0002-4287-4801>),
Ken Kelley [ctb],
David Stanley [ctb]

Repository CRAN

Date/Publication 2019-09-26 10:40:02 UTC

R topics documented:

`check_factorstructure` *Check suitability of data for Factor Analysis (FA)*

Description

This checks whether the data is appropriate for Factor Analysis (FA) by running the Bartlett's Test of Sphericity and the Kaiser, Meyer, Olkin (KMO) Measure of Sampling Adequacy (MSA).

Usage

```
check_factorstructure(x, silent = FALSE, ...)
```

Arguments

`x`	A dataframe.
`silent`	Hide results printing.
`...`	Arguments passed to or from other methods.

Value

A list of lists of indices related to sphericity and KMO.

See Also

check_kmo check_sphericity

Examples

```
library(parameters)

check_factorstructure(mtcars)
```

---

check_kmo *Kaiser, Meyer, Olkin (KMO) Measure of Sampling Adequacy (MSA) for Factor Analysis*

Description

Kaiser (1970) introduced a Measure of Sampling Adequacy (MSA), later modified by Kaiser and Rice (1974). The Kaiser-Meyer-Olkin (KMO) statistic, which can vary from 0 to 1, indicates the degree to which each variable in a set is predicted without error by the other variables.

Usage

```
check_kmo(x, silent = FALSE, ...)
```

Arguments

x	A dataframe.
silent	Hide results printing.
...	Arguments passed to or from other methods.

Details

A value of 0 indicates that the sum of partial correlations is large relative to the sum correlations, indicating factor analysis is likely to be inappropriate. A KMO value close to 1 indicates that the sum of partial correlations is not large relative to the sum of correlations and so factor analysis should yield distinct and reliable factors.

Kaiser (1975) suggested that KMO > .9 were marvelous, in the .80s, meritourious, in the .70s, middling, in the .60s, mediocre, in the 50s, miserable, and less than .5, unacceptable. Hair et al. (2006) suggest accepting a value > 0.5. Values between 0.5 and 0.7 are mediocre, and values between 0.7 and 0.8 are good.

This function is strongly inspired by the `KMO` function in the `psych` package (Revelle, 2016). All credits go to its author.

Value

A list of indices related to KMO.

References

- Revelle, W. (2016). How To: Use the psych package for Factor Analysis and data reduction.
- Kaiser, H. F. (1970). A second generation little jiffy. Psychometrika, 35(4), 401-415.
- Kaiser, H. F., \& Rice, J. (1974). Little jiffy, mark IV. Educational and psychological measurement, 34(1), 111-117.
- Kaiser, H. F. (1974). An index of factorial simplicity. Psychometrika, 39(1), 31-36.

Examples

```
library(parameters)

check_kmo(mtcars)
```

---

check_smoothness *Quantify the smoothness of a vector*

Description

Quantify the smoothness of a vector

Usage

```
check_smoothness(x, method = "cor", lag = 1)

smoothness(x, method = "cor", lag = 1)
```

Arguments

x	Numeric vector (similar to a time series).
method	Can be "diff" (the standard deviation of the standardized differences) or "cor" (default, lag-one autocorrelation).
lag	An integer indicating which lag to use. If less than 1, will be interpreted as expressed in percentage of the length of the vector.

Value

Value of smoothness.

References

https://stats.stackexchange.com/questions/24607/how-to-measure-smoothness-of-a-time-series-in-r

Examples

```
x <- (-10:10)^3 + rnorm(21, 0, 100)
plot(x)
smoothness(x, method = "cor")
smoothness(x, method = "diff")
```

`check_sphericity` *Bartlett's Test of Sphericity*

Description

Bartlett (1951) introduced the test of sphericity, which tests whether a matrix is significantly different from an identity matrix. This statistical test for the presence of correlations among variables, providing the statistical probability that the correlation matrix has significant correlations among at least some of variables. As for factor analysis to work, some relationships between variables are needed, thus, a significant Bartlett's test of sphericity is required, say $p < .001$.

Usage

```
check_sphericity(x, silent = FALSE, ...)
```

Arguments

`x`	A dataframe.
`silent`	Hide results printing.
`...`	Arguments passed to or from other methods.

Details

This function is strongly inspired by the `cortest.bartlett` function in the `psych` package (Revelle, 2016). All credits go to its author.

Value

A list of indices related to sphericity.

References

- Revelle, W. (2016). How To: Use the psych package for Factor Analysis and data reduction.
- Bartlett, M. S. (1951). The effect of standardization on a Chi-square approximation in factor analysis. Biometrika, 38(3/4), 337-344.

Examples

```
library(parameters)

check_sphericity(mtcars)
```

`ci.merMod` *Confidence Interval (CI)*

Description

Compute confidence intervals (CI) for frequentist models.

Usage

```
## S3 method for class 'merMod'
ci(x, ci = 0.95, method = c("wald", "kenward",
  "boot"), ...)

## S3 method for class 'glm'
ci(x, ci = 0.95, method = c("profile", "wald"), ...)

## S3 method for class 'glmmTMB'
ci(x, ci = 0.95, component = c("all", "conditional",
  "zi", "zero_inflated"), ...)

## S3 method for class 'zeroinfl'
ci(x, ci = 0.95, component = c("all", "conditional",
  "zi", "zero_inflated"), ...)

## S3 method for class 'hurdle'
ci(x, ci = 0.95, component = c("all", "conditional",
  "zi", "zero_inflated"), ...)

## S3 method for class 'MixMod'
ci(x, ci = 0.95, component = c("all", "conditional",
  "zi", "zero_inflated"), ...)
```

Arguments

`x`	A statistical model.
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`method`	For mixed models of class `merMod`, can be `"wald"` (default), `"kenward"` or `"boot"` (see `p_value_kenward` and `lme4::confint.merMod`). For generalized linear models, can be `"profile"` (default) or `"wald"`.
`...`	Arguments passed to or from other methods.
`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). May be abbreviated.

Value

A data frame containing the CI bounds.

Examples

```
library(parameters)
library(glmmTMB)

model <- glmmTMB(
  count ~ spp + mined + (1 | site),
  ziformula = ~mined,
  family = poisson(),
  data = Salamanders
)

ci(model)
ci(model, component = "zi")
```

---

ci_wald *Wald-test approximation for CIs and p-values*

Description

The Wald-test approximation treats t-values as Wald z. Since the t distribution converges to the z distribution as degrees of freedom increase, this is like assuming infinite degrees of freedom. While this is unambiguously anti-conservative, this approximation appears as reasonable for reasonable sample sizes (Barr et al., 2013). That is, if we take the p-value to measure the probability of a false positive, this approximation produces a higher false positive rate than the nominal 5% at p = 0.05.

Usage

```
ci_wald(model, ci = 0.95, dof = NULL, component = c("all",
  "conditional", "zi", "zero_inflated"))

p_value_wald(model, ...)

## S3 method for class 'merMod'
p_value_wald(model, dof = Inf, ...)
```

Arguments

model	A statistical model.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
dof	Degrees of Freedom. If not specified, for ci_wald(), defaults to model's residual degrees of freedom (i.e. n-k, where n is the number of observations and k is the number of parameters). For p_value_wald(), defaults to Inf.

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). May be abbreviated.

... — Currently not used.

Value

The p-values.

References

Barr, D. J. (2013). Random effects structure for testing interactions in linear mixed-effects models. Frontiers in psychology, 4, 328.

Examples

```
library(lme4)
model <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
p_value_wald(model)
ci_wald(model, ci = c(0.90, 0.95))
```

---

cmds *Classical Multidimensional Scaling (cMDS)*

Description

Also referred to as principal Coordinates Analysis (PCoA), Classical Multidimensional Scaling (cMDS) takes a set of dissimilarities (i.e., a distance matrix) and returns a set of points such that the distances between the points are approximately equal to the dissimilarities.

Usage

```
cmds(x, n = "all", distance = "euclidean", ...)
```

Arguments

x — A dataframe or a statistical model.

n — Number of components to extract. If n = "all" =, then n is set as the number of variables minus 1 (ncol(x)-1). If n = "auto" (default) or n = NULL, the number of components is selected through n_factors. In parameters_reduction, can also be "max", in which case it will select all the components that are maximally pseudo-loaded (i.e., correlated) by at least one variable.

distance	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Any unambiguous substring can be given.
...	Arguments passed to or from other methods.

References

- Nguyen, L. H., \& Holmes, S. (2019). Ten quick tips for effective dimensionality reduction. PLOS Computational Biology, 15(6).

Examples

```
cmds(iris[, 1:4])
```

---

cohens_f *ANOVA Effect Size (Omega Squared, Eta Squared, Epsilon Squared)*

Description

Set of functions to compute (partial) indices for ANOVAs, such as omega squared, the eta squared or the epsilon squared (Kelly, 1935). They are measures of effect size, or the degree of association for a population. They are an estimate of how much variance in the response variables are accounted for by the explanatory variables.

Usage

```
cohens_f(model)

epsilon_squared(model)

eta_squared(model, partial = TRUE, ci = NULL, ...)

## S3 method for class 'aov'
eta_squared(model, partial = TRUE, ci = NULL,
  iterations = 1000, ...)

omega_squared(model, partial = TRUE, ci = NULL, iterations = 1000)
```

Arguments

model	An ANOVA object.
partial	If TRUE, return partial indices (adjusted for sample size).
ci	Confidence Interval (CI) level computed via bootstrap.
...	Arguments passed to or from other methods.
iterations	The number of draws to simulate/bootstrap.

Details

Omega Squared: Omega squared is considered as a lesser biased alternative to eta-squared, especially when sample sizes are small (Albers \& Lakens, 2018). Field (2013) suggests the following interpretation heuristics:

- Omega Squared = 0 - 0.01: Very small
- Omega Squared = 0.01 - 0.06: Small
- Omega Squared = 0.06 - 0.14: Medium
- Omega Squared > 0.14: Large

Epsilon Squared: It is one of the least common measures of effect sizes: omega squared and eta squared are used more frequently. Although having a different name and a formula in appearance different, this index is equivalent to the adjusted R2 (Allen, 2017, p. 382).

Cohen's f: Cohen's f statistic is one appropriate effect size index to use for a oneway analysis of variance (ANOVA). Cohen's f can take on values between zero, when the population means are all equal, and an indefinitely large number as standard deviation of means increases relative to the average standard deviation within each group. Cohen has suggested that the values of 0.10, 0.25, and 0.40 represent small, medium, and large effect sizes, respectively.

Value

Data.frame containing the effect size values.

References

- Albers, C., \& Lakens, D. (2018). When power analyses based on pilot data are biased: Inaccurate effect size estimators and follow-up bias. Journal of experimental social psychology, 74, 187-195.
- Allen, R. (2017). Statistics and Experimental Design for Psychologists: A Model Comparison Approach. World Scientific Publishing Company.
- Field, A. (2013). Discovering statistics using IBM SPSS statistics. sage.
- Kelley, K. (2007). Methods for the behavioral, educational, and social sciences: An R package. Behavior Research Methods, 39(4), 979-984.
- Kelley, T. (1935) An unbiased correlation ratio measure. Proceedings of the National Academy of Sciences. 21(9). 554-559.

The computation of CIs is based on the implementation done by Stanley (2018) in the `ApaTables` package and Kelley (2007) in the `MBESS` package. All credits go to them.

Examples

```
library(parameters)

df <- iris
df$Sepal.Big <- ifelse(df$Sepal.Width >= 3, "Yes", "No")

model <- aov(Sepal.Length ~ Sepal.Big, data = df)
omega_squared(model)
```

```
eta_squared(model)
epsilon_squared(model)
cohens_f(model)

model <- anova(lm(Sepal.Length ~ Sepal.Big * Species, data = df))
omega_squared(model)
eta_squared(model)
epsilon_squared(model)

# Don't work for now
model <- aov(Sepal.Length ~ Sepal.Big + Error(Species), data = df)
omega_squared(model)
eta_squared(model)
epsilon_squared(model)
```

`convert_data_to_numeric` *Convert data to numeric*

Description

Convert data to numeric by converting characters to factors and factors to either numeric levels or dummy variables.

Usage

```
convert_data_to_numeric(x, dummy_factors = TRUE, ...)

data_to_numeric(x, dummy_factors = TRUE, ...)
```

Arguments

`x`	A data.frame or a vector.
`dummy_factors`	Transform factors to dummy factors (all factor levels as different columns filled with a binary 0-1 value).
`...`	Arguments passed to or from other methods.

Value

Data.frame of numeric variables.

Examples

```
convert_data_to_numeric(iris)
```

convert_efa_to_cfa *Conversion between EFA results and CFA structure*

Description

Enables a conversion between Exploratory Factor Analysis (EFA) and Confirmatory Factor Analysis (CFA) lavaan-ready structure.

Usage

```
convert_efa_to_cfa(model, ...)

## S3 method for class 'fa'
convert_efa_to_cfa(model, threshold = "max", names = NULL,
  ...)

efa_to_cfa(model, ...)
```

Arguments

model	An EFA model (e.g., a psych::fa object).
...	Arguments passed to or from other methods.
threshold	A value between 0 and 1 indicates which (absolute) values from the loadings should be removed. An integer higher than 1 indicates the n strongest loadings to retain. Can also be "max", in which case it will only display the maximum loading per variable (the most simple structure).
names	Vector containing dimension names.

Value

Converted index.

Examples

```
library(psych)
library(lavaan)
library(parameters)

efa <- psych::fa(attitude, nfactors = 3)

model1 <- efa_to_cfa(efa)
model2 <- efa_to_cfa(efa, threshold = 0.3)

anova(
  lavaan::cfa(model1, data = attitude),
  lavaan::cfa(model2, data = attitude)
)
```

`data_partition` *Partition data into a test and a training set*

Description

Creates a training and a test set based on a dataframe. Can also be stratified (i.e., evenly spread a given factor) using the `group` argument.

Usage

```
data_partition(x, training_proportion = 0.7, group = NULL)
```

Arguments

`x`	A data frame, or an object that can be coerced to a data frame.
`training_proportion`	The proportion (between 0 and 1) of the training set. The remaining part will be used for the test set.
`group`	A character vector indicating the name(s) of the column(s) used for stratified partitioning.

Value

A list of two data frames, named `test` and `training`.

Examples

```
df <- iris
df$Smell <- rep(c("Strong", "Light"), 75)

data_partition(df)
data_partition(df, group = "Species")
data_partition(df, group = c("Species", "Smell"))
```

describe_distribution *Describe a Distribution*

Description

This function describes a distribution by a set of indices (e.g., measures of centrality, dispersion, range, skewness, kurtosis).

Usage

```
describe_distribution(x, centrality = "mean", dispersion = TRUE,
  range = TRUE, ...)
```

Arguments

x	A numeric vector.
centrality	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: "median", "mean", "MAP" or "all".
dispersion	Logical, if TRUE, computes indices of dispersion related to the estimate(s) (SD and MAD for mean and median, respectively).
range	Return the range (min and max).
...	Additional arguments to be passed to or from methods.

Value

Converted index.

Examples

```
describe_distribution(rnorm(100))
```

DRR *Dimensionality Reduction via Regression (DRR)*

Description

Dimensionality Reduction via Regression (DRR) is a very recent technique extending PCA (Laparra et al., 2015). Starting from a rotated PCA, it predicts redundant information from the remaining components using non-linear regression. Some of the most notable advantages of performing PCR are avoidance of multicollinearity between predictors and overfitting mitigation. PCR tends to perform well when the first principal components are enough to explain most of the variation in the predictors. Requires the **DRR** package to be installed.

Usage

```
DRR(x, n = "all", ...)
```

Arguments

x	A dataframe or a statistical model.
n	Number of components to extract. If n = "all" =, then n is set as the number of variables minus 1 (ncol(x)-1). If n = "auto" (default) or n = NULL, the number of components is selected through n_factors. In parameters_reduction, can also be "max", in which case it will select all the components that are maximally pseudo-loaded (i.e., correlated) by at least one variable.
...	Arguments passed to or from other methods.

References

- Laparra, V., Malo, J., & Camps-Valls, G. (2015). Dimensionality reduction via regression in hyperspectral imagery. IEEE Journal of Selected Topics in Signal Processing, 9(6), 1026-1036.

Examples

```
DRR(iris[, 1:4])
```

---

d_to_r *Conversion between standardized difference d and correlation r*

Description

Enables a conversion between standardized difference (Cohen's d) and correlation r.

Usage

```
d_to_r(d, ...)

r_to_d(r, ...)

convert_d_to_r(d, ...)

convert_r_to_d(r, ...)
```

Arguments

d	A standardized difference value (Cohen's d).
...	Arguments passed to or from other methods.
r	A correlation coefficient r.

Value

Converted index.

References

- Borenstein, Michael, et al. "Converting among effect sizes." Introduction to meta-analysis (2009): 45-49.

Examples

```
d_to_r(d = 1.1547)
```

---

equivalence_test.lm *Equivalence test*

Description

Compute the equivalence test for frequentist models.

Usage

```
## S3 method for class 'lm'
equivalence_test(x, range = "default", ci = 0.95,
  verbose = TRUE, ...)
```

Arguments

x	A statistical model.
range	The range of practical equivalence of an effect. May be "default", to automatically define this range based on properties of the model's data.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
verbose	Toggle off warnings.
...	Arguments passed to or from other methods.

Value

A data frame.

See Also

For more details, see equivalence_test.

Examples

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

format_bf *Bayes Factor Formatting*

Description

Bayes Factor Formatting

Usage

```
format_bf(bf, stars = FALSE, stars_only = FALSE, name = "BF")
```

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.

Value

A formatted string.

Examples

```
format_bf(1.20)
format_bf(c(1.20, 1557, 3.5, 12), stars = TRUE)
format_bf(c(1.20, 1557, 3.5, 12), name = NULL)
```

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)
```

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.

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)
```

---

format_number *Convert number to words*

Description

Convert number to words. The code has been adapted from here https://github.com/ateucher/useful_code/blob/master/R/numb

Usage

```
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.

Examples

```
format_number(2)
format_number(45)
format_number(324.68765)
```

format_order *Order (first, second, ...) formatting*

Description

Format order.

Usage

```
format_order(order, textual = TRUE, ...)
```

Arguments

order	value or vector of orders.
textual	Return number as words. If FALSE, will run format_value.
...	Arguments to be passed to format_value if textual is FALSE.

Value

A formatted string.

Examples

```
format_order(2)
format_order(8)
format_order(25, textual = FALSE)
```

format_p *p-values formatting*

Description

Format p-values.

Usage

```
format_p(p, stars = FALSE, stars_only = FALSE, name = "p",
  missing = "")
```

Arguments

p	value or vector of p-values.
stars	Add significance stars (e.g., p < .001***).
stars_only	Return only significance stars.
name	Name prefixing the text. Can be NULL.
missing	Value by which NA values are replaced. By default, an empty string (i.e. "") is returned for NA.

Value

A formatted string.

Examples

```
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)
```

---

format_parameters *Parameters Names Formatting*

Description

Parameters Names Formatting

Usage

```
format_parameters(model)
```

Arguments

model	A statistical model.

Value

The formatted parameter names.

Examples

```
library(parameters)

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

model <- lm(Sepal.Length ~ Species / Petal.Length, data = iris)
format_parameters(model)

model <- lm(Sepal.Length ~ Species / Petal.Length * Sepal.Width, data = iris)
format_parameters(model)

model <- lm(Sepal.Length ~ Species / (Petal.Length * Sepal.Width), data = iris)
format_parameters(model)

model <- lm(Sepal.Length ~ Species + poly(Sepal.Width, 2), data = iris)
format_parameters(model)

model <- lm(Sepal.Length ~ Species + poly(Sepal.Width, 2, raw = TRUE), data = iris)
format_parameters(model)
```

format_pd *Probability of direction (pd) Formatting*

Description

Probability of direction (pd) Formatting

Usage

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

Arguments

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

Value

A formatted string.

Examples

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

format_rope *Percentage in ROPE Formatting*

Description

Percentage in ROPE Formatting

Usage

```
format_rope(rope_percentage, name = "in ROPE")
```

Arguments

rope_percentage	Value or vector of percentages in ROPE.
name	Name prefixing the text. Can be NULL.

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_standardize` *Transform a standardized vector into character*

Description

Transform a standardized vector into character, e.g., `c("-1 SD","Mean","+1 SD")`.

Usage

```
format_standardize(x, reference = x, robust = FALSE, digits = NULL,
  ...)
```

Arguments

`x`	A standardized numeric vector.
`reference`	The reference vector from which to compute the mean and SD.
`robust`	Logical, if `TRUE`, centering is done by substracting the median from the variables and divide it by the median absolute deviation (MAD). If `FALSE`, variables are standardized by substracting the mean and divide it by the standard deviation (SD).
`digits`	Number of significant digits.
`...`	Arguments passed to or from other methods.

Examples

```
format_standardize(c(-1, 0, 1))
format_standardize(c(-1, 0, 1, 2), reference = rnorm(1000))
format_standardize(c(-1, 0, 1, 2), reference = rnorm(1000), robust = TRUE)
```

ICA *Independent Component Analysis (ICA)*

Description

Performs an Independent Component Analysis using the FastICA algorithm. Contrary to PCA, that attempts to find uncorrelated sources (through least squares minimization), ICA attempts to find independent sources, i.e., the source space that maximizes the "non-gaussianity" of all sources. Contrary to PCA, ICA does not rank each source, which makes it a poor tool for dimensionality reduction. Requires the **fastICA** package to be installed.

Usage

```
ICA(x, n = "all", ...)
```

Arguments

x	A dataframe or a statistical model.
n	Number of components to extract. If n = "all" =, then n is set as the number of variables minus 1 (ncol(x)-1). If n = "auto" (default) or n = NULL, the number of components is selected through n_factors. In parameters_reduction, can also be "max", in which case it will select all the components that are maximally pseudo-loaded (i.e., correlated) by at least one variable.
...	Arguments passed to or from other methods.

Examples

```
ICA(iris[, 1:4])
```

model_bootstrap *Model bootstrapping*

Description

Bootstrap a statistical model n times to return a data.frame of estimates.

Usage

```
model_bootstrap(model, iterations = 1000, verbose = FALSE, ...)
```

Arguments

model	Statistical model.
iterations	The number of draws to simulate/bootstrap.
verbose	Hide possible refit messages.
...	Arguments passed to or from other methods.

Value

A data.frame.

See Also

parameters_bootstrap, model_simulate, parameters_simulate

Examples

```
model <- lm(mpg ~ wt + cyl, data = mtcars)
head(model_bootstrap(model))
```

---

model_parameters *Model Parameters*

Description

Compute and extract model parameters. See the documentation for your object's class:

- Correlations and t-tests
- ANOVAs
- Regression models (lm, glm, **survey**, ...)
- Additive models (gam, gamm, ...)
- Zero-inflated models (hurdle, zeroinfl, zerocount)
- Mixed models (**lme4**, **nlme**, **glmmTMB**, ...)
- Bayesian tests (**BayesFactor**)
- Bayesian models (**rstanarm**, **brms**, **MCMCglmm**)
- PCA and FA (**psych**)
- CFA and SEM (**lavaan**)

Usage

```
model_parameters(model, ...)

parameters(model, ...)
```

Arguments

model	Statistical Model.
...	Arguments passed to or from other methods.

Value

A data frame of indices related to the model's parameters.

See Also

standardize_names() to rename columns into a consistent, standardized naming scheme.

`model_parameters.aov` *ANOVAs Parameters*

Description

Parameters of ANOVAs.

Usage

```
## S3 method for class 'aov'
model_parameters(model, omega_squared = NULL,
  eta_squared = NULL, epsilon_squared = NULL, ...)
```

Arguments

`model`	Object of class aov, anova or `aovlist`.
`omega_squared`	Compute omega squared as index of effect size. Can be "partial" (adjusted for effect size) or "raw".
`eta_squared`	Compute eta squared as index of effect size. Can be "partial" (adjusted for effect size) or "raw".
`epsilon_squared`	Compute epsilon squared as index of effect size.
`...`	Arguments passed to or from other methods.

Value

A data.frame of indices related to the model's parameters.

Examples

```
df <- iris
df$Sepal.Big <- ifelse(df$Sepal.Width >= 3, "Yes", "No")

model <- aov(Sepal.Length ~ Sepal.Big, data = df)
model_parameters(model, omega_squared = "partial", eta_squared = "partial", epsilon_squared = TRUE)

model <- anova(lm(Sepal.Length ~ Sepal.Big, data = df))
model_parameters(model)
model_parameters(model, omega_squared = "partial", eta_squared = "partial", epsilon_squared = TRUE)

model <- aov(Sepal.Length ~ Sepal.Big + Error(Species), data = df)
model_parameters(model)

library(lme4)

model <- anova(lmer(Sepal.Length ~ Sepal.Big + (1 | Species), data = df))
model_parameters(model)
```

`model_parameters.befa` *Format PCA/FA from the psych package*

Description

Format PCA/FA objects from the psych package (Revelle, 2016).

Usage

```
## S3 method for class 'befa'
model_parameters(model, sort = FALSE,
  centrality = "median", dispersion = FALSE, ci = 0.89,
  ci_method = "hdi", test = NULL, ...)
```

Arguments

`model`	Bayesian EFA created by the `BayesFM::befa`.
`sort`	Sort the loadings.
`centrality`	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: `"median"`, `"mean"`, `"MAP"` or `"all"`.
`dispersion`	Logical, if `TRUE`, computes indices of dispersion related to the estimate(s) (`SD` and `MAD` for `mean` and `median`, respectively).
`ci`	Value or vector of probability of the CI (between 0 and 1) to be estimated. Default to `.89` (89%) for Bayesian models and `.95` (95%) for frequentist models.
`ci_method`	The type of index used for Credible Interval. Can be `"HDI"` (default, see `hdi`) or `"ETI"` (see `eti`).
`test`	The indices of effect existence to compute. Character (vector) or list with one or more of these options: `"p_direction"` (or `"pd"`), `"rope"`, `"p_map"`, `"equivalence_test"` (or `"equitest"`), `"bayesfactor"` (or `"bf"`) or `"all"` to compute all tests. For each "test", the corresponding **bayestestR** function is called (e.g. `rope` or `p_direction`) and its results included in the summary output.
`...`	Arguments passed to or from other methods.

Value

A data.frame of loadings.

Examples

```
library(parameters)

library(BayesFM)
efa <- BayesFM::befa(mtcars, iter = 1000)
results <- model_parameters(efa, sort = TRUE)

results
```

```
attributes(results)$loadings_long
efa_to_cfa(results)
```

`model_parameters.BFBayesFactor` *BayesFactor objects Parameters*

Description

Parameters of BayesFactor objects.

Usage

```
## S3 method for class 'BFBayesFactor'
model_parameters(model, centrality = "median",
  dispersion = FALSE, ci = 0.89, ci_method = "hdi", test = c("pd",
  "rope"), rope_range = "default", rope_ci = 0.89, priors = TRUE,
  ...)
```

Arguments

`model`	Object of class `BFBayesFactor`.
`centrality`	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: `"median"`, `"mean"`, `"MAP"` or `"all"`.
`dispersion`	Logical, if `TRUE`, computes indices of dispersion related to the estimate(s) (`SD` and `MAD` for `mean` and `median`, respectively).
`ci`	Value or vector of probability of the CI (between 0 and 1) to be estimated. Default to `.89` (89%) for Bayesian models and `.95` (95%) for frequentist models.
`ci_method`	The type of index used for Credible Interval. Can be `"HDI"` (default, see `hdi`) or `"ETI"` (see `eti`).
`test`	The indices of effect existence to compute. Character (vector) or list with one or more of these options: `"p_direction"` (or `"pd"`), `"rope"`, `"p_map"`, `"equivalence_test"` (or `"equitest"`), `"bayesfactor"` (or `"bf"`) or `"all"` to compute all tests. For each "test", the corresponding **bayestestR** function is called (e.g. `rope` or `p_direction`) and its results included in the summary output.
`rope_range`	ROPE's lower and higher bounds. Should be a list of two values (e.g., `c(-0.1,0.1)`) or `"default"`. If `"default"`, the bounds are set to `x +-0.1*SD(response)`.
`rope_ci`	The Credible Interval (CI) probability, corresponding to the proportion of HDI, to use for the percentage in ROPE.
`priors`	Add the prior used for each parameter.
`...`	Additional arguments to be passed to or from methods.

Value

A data.frame of indices related to the model's parameters.

Examples

```
library(BayesFactor)
model <- ttestBF(x = rnorm(100, 1, 1))
model_parameters(model)
```

`model_parameters.gam` *Parameters of Generalized Additive (Mixed) Models*

Description

Extract and compute indices and measures to describe parameters of generalized additive models (GAM(M)s).

Usage

```
## S3 method for class 'gam'
model_parameters(model, ci = 0.95, bootstrap = FALSE,
  iterations = 1000, ...)
```

Arguments

`model`	A gam/gamm model.
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`bootstrap`	Should estimates be based on bootstrapped model? If `TRUE`, then arguments of Bayesian regressions apply (see also `parameters_bootstrap()`).
`iterations`	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
`...`	Arguments passed to or from other methods (e.g., to `standardize()`).

Value

A data frame of indices related to the model's parameters.

See Also

`standardize_names()` to rename columns into a consistent, standardized naming scheme.

Examples

```
library(parameters)
library(mgcv)

dat <- gamSim(1, n = 400, dist = "normal", scale = 2)
model <- gam(y ~ s(x0) + s(x1) + s(x2) + s(x3), data = dat)
model_parameters(model)
```

`model_parameters.glmmTMB`
Mixed Model Parameters

Description

Parameters of mixed models.

Usage

```
## S3 method for class 'glmmTMB'
model_parameters(model, ci = 0.95,
  standardize = FALSE, standardize_robust = FALSE, bootstrap = FALSE,
  iterations = 1000, component = c("all", "conditional", "zi",
  "zero_inflated"), ...)

## S3 method for class 'lme'
model_parameters(model, ci = 0.95, standardize = FALSE,
  standardize_robust = FALSE, bootstrap = FALSE, iterations = 1000,
  ...)

## S3 method for class 'merMod'
model_parameters(model, ci = 0.95,
  standardize = FALSE, standardize_robust = FALSE, bootstrap = FALSE,
  p_method = "wald", ci_method = "wald", iterations = 1000, ...)
```

Arguments

`model`	A mixed model.
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`standardize`	Add standardized parameters. Can be `FALSE` or a character indicating the standardization method (see `parameters_standardize()`), such as `"refit"`, `"2sd"`, `"smart"` or `"classic"`. The two former are based on model refitting using a standardized version of data. It is the most accurate, although computationally heavy (as it must re-fit a second model). The "smart" and "classic" are post-hoc methods, fast, but inaccurate (especially if the model includes interactions).
`standardize_robust`	Robust standardization. See `parameters_standardize`.

bootstrap	Should estimates be based on bootstrapped model? If TRUE, then arguments of Bayesian regressions apply (see also parameters_bootstrap()).
iterations	The number of draws to simulate/bootstrap.
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). May be abbreviated.
...	Arguments passed to or from other methods.
p_method	Method for computing p values. See p_value().
ci_method	Method for computing confidence intervals (CI). See ci().

Value

A data frame of indices related to the model's parameters.

See Also

standardize_names() to rename columns into a consistent, standardized naming scheme.

Examples

```
library(parameters)
library(lme4)
library(glmmTMB)

model <- lmer(mpg ~ wt + (1 | gear), data = mtcars)
model_parameters(model, standardize = "refit")

model <- glmmTMB(
  count ~ spp + mined + (1 | site),
  ziformula = ~mined,
  family = poisson(),
  data = Salamanders
)
model_parameters(model)

model <- lme4::lmer(mpg ~ wt + (1 | gear), data = mtcars)
model_parameters(model, standardize = "smart", bootstrap = TRUE, iterations = 50)
```

---

model_parameters.htest

Correlations and t-test Parameters

Description

Parameters of h-tests (correlations, t-tests).

Usage

```
## S3 method for class 'htest'
model_parameters(model, bootstrap = FALSE, ...)
```

Arguments

model	Object of class htest.
bootstrap	Should estimates be bootstrapped?
...	Arguments passed to or from other methods (e.g., to standardize).

Value

A data.frame of indices related to the model's parameters.

Examples

```
model <- cor.test(mtcars$mpg, mtcars$cyl, method = "pearson")
model_parameters(model)

model <- t.test(iris$Sepal.Width, iris$Sepal.Length)
model_parameters(model)

model <- t.test(mtcars$mpg ~ mtcars$vs)
model_parameters(model)

model <- t.test(iris$Sepal.Width, mu = 1)
model_parameters(model)
```

---

model_parameters.lavaan

Format CFA/SEM from the lavaan package

Description

Format CFA/SEM objects from the lavaan package (Rosseel, 2012).

Usage

```
## S3 method for class 'lavaan'
model_parameters(model, ci = 0.95,
  standardize = FALSE, type = c("regression", "correlation",
  "loading"), ...)
```

Arguments

model	CFA or SEM created by the lavaan::cfa or lavaan::sem functions.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
standardize	Add standardized parameters. Can be FALSE or a character indicating the standardization method (see parameters_standardize()), such as "refit", "2sd", "smart" or "classic". The two former are based on model refitting using a standardized version of data. It is the most accurate, although computationally heavy (as it must re-fit a second model). The "smart" and "classic" are post-hoc methods, fast, but inaccurate (especially if the model includes interactions).
type	What type of links to return. Can be "all" or some of c("regression","correlation","loading","v
...	Arguments passed to or from other methods.

Value

A data.frame of indices related to the model's parameters.

References

- Rosseel (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36.

Examples

```
library(parameters)

# lavaan -------------------------------------
library(lavaan)

# Confirmatory Factor Analysis (CFA) ---------

structure <- " visual  =~ x1 + x2 + x3
               textual =~ x4 + x5 + x6
               speed   =~ x7 + x8 + x9 "
model <- lavaan::cfa(structure, data = HolzingerSwineford1939)
model_parameters(model)
model_parameters(model, standardize = TRUE)

# Structural Equation Model (SEM) ------------

structure <- "
  # latent variable definitions
    ind60 =~ x1 + x2 + x3
    dem60 =~ y1 + a*y2 + b*y3 + c*y4
    dem65 =~ y5 + a*y6 + b*y7 + c*y8
  # regressions
    dem60 ~ ind60
    dem65 ~ ind60 + dem60
  # residual correlations
    y1 ~~ y5
    y2 ~~ y4 + y6
```

```
    y3 ~~ y7
    y4 ~~ y8
    y6 ~~ y8
"
model <- lavaan::sem(structure, data = PoliticalDemocracy)
model_parameters(model)
model_parameters(model, standardize = TRUE)

# blavaan ------------------------------------
# library(blavaan)

# model <- blavaan::bsem(structure, data=PoliticalDemocracy)
# model_parameters(model)
# model_parameters(model, standardize = TRUE)
```

`model_parameters.lm` *Parameters of (General) Linear Models*

Description

Extract and compute indices and measures to describe parameters of (general) linear models (GLMs).

Usage

```
## S3 method for class 'lm'
model_parameters(model, ci = 0.95, standardize = FALSE,
  standardize_robust = FALSE, bootstrap = FALSE, iterations = 1000,
  ...)

## S3 method for class 'polr'
model_parameters(model, ci = 0.95, bootstrap = FALSE,
  iterations = 1000, ...)
```

Arguments

`model`	Model object.
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`standardize`	Add standardized parameters. Can be `FALSE` or a character indicating the standardization method (see `parameters_standardize()`), such as `"refit"`, `"2sd"`, `"smart"` or `"classic"`. The two former are based on model refitting using a standardized version of data. It is the most accurate, although computationally heavy (as it must re-fit a second model). The "smart" and "classic" are post-hoc methods, fast, but inaccurate (especially if the model includes interactions).
`standardize_robust`	Robust standardization. See `parameters_standardize`.

bootstrap	Should estimates be based on bootstrapped model? If TRUE, then arguments of Bayesian regressions apply (see also parameters_bootstrap()).
iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
...	Arguments passed to or from other methods (e.g., to standardize()).

Value

A data frame of indices related to the model's parameters.

Note

Standardization (argument standardize) is not supported by all model objects.

See Also

standardize_names() to rename columns into a consistent, standardized naming scheme.

Examples

```
library(parameters)
model <- lm(mpg ~ wt + cyl, data = mtcars)

model_parameters(model, standardize = "refit")
model_parameters(model, bootstrap = TRUE)

model <- glm(vs ~ wt + cyl, data = mtcars, family = "binomial")
model_parameters(model)
```

---

model_parameters.PCA *Structural Models (PCA, EFA, ...)*

Description

Format structural models from the **psych** or **FactoMineR** packages.

Usage

```
## S3 method for class 'PCA'
model_parameters(model, sort = FALSE, threshold = NULL,
  labels = NULL, ...)

## S3 method for class 'principal'
model_parameters(model, sort = FALSE,
  threshold = NULL, labels = NULL, ...)
```

Arguments

model	PCA or FA created by the **psych** or **FactoMineR** packages (e.g. through psych::principal or psych::fa).
sort	Sort the loadings.
threshold	A value between 0 and 1 indicates which (absolute) values from the loadings should be removed. An integer higher than 1 indicates the n strongest loadings to retain. Can also be "max", in which case it will only display the maximum loading per variable (the most simple structure).
labels	A character vector containing labels to be added to the loadings data. Usually, the question related to the item.
...	Arguments passed to or from other methods.

Details

For the structural models obtained with **psych**, the following indices are present:

- **Complexity** (Hoffman's, 1978; Pettersson and Turkheimer, 2010) represents the number of latent components needed to account for the observed variables. Whereas a perfect simple structure solution has a complexity of 1 in that each item would only load on one factor, a solution with evenly distributed items has a complexity greater than 1.
- **Uniqueness** represents the variance that is 'unique' to the variable and not shared with other variables. It is equal to `1 communality` (variance that is shared with other variables). A uniqueness of `0.20` suggests that 20% or that variable's variance is not shared with other variables in the overall factor model. The greater 'uniqueness' the lower the relevance of the variable in the factor model.

Value

A data.frame of loadings.

References

- Pettersson, E., \& Turkheimer, E. (2010). Item selection, evaluation, and simple structure in personality data. Journal of research in personality, 44(4), 407-420.
- Revelle, W. (2016). How To: Use the psych package for Factor Analysis and data reduction.

Examples

```
library(parameters)
library(psych)

# Principal Component Analysis (PCA) ---------
pca <- psych::principal(attitude)
model_parameters(pca)

pca <- psych::principal(attitude, nfactors = 3, rotate = "none")
model_parameters(pca, sort = TRUE, threshold = 0.2)
```

```
principal_components(attitude, n = 3, sort = TRUE, threshold = 0.2)

# Exploratory Factor Analysis (EFA) ---------
efa <- psych::fa(attitude, nfactors = 3)
model_parameters(efa, threshold = "max", sort = TRUE, labels = as.character(1:ncol(attitude)))



# FactoMineR ---------
library(FactoMineR)

model <- FactoMineR::PCA(iris[, 1:4], ncp = 2)
model_parameters(model)
attributes(model_parameters(model))$scores

model <- FactoMineR::FAMD(iris, ncp = 2)
model_parameters(model)
```

`model_parameters.stanreg` *Bayesian Models Parameters*

Description

Parameters of Bayesian models.

Usage

```
## S3 method for class 'stanreg'
model_parameters(model, centrality = "median",
  dispersion = FALSE, ci = 0.89, ci_method = "hdi", test = c("pd",
  "rope"), rope_range = "default", rope_ci = 1, bf_prior = NULL,
  diagnostic = c("ESS", "Rhat"), priors = TRUE, standardize = FALSE,
  standardize_robust = FALSE, iterations = 1000, ...)
```

Arguments

`model`	Bayesian model.
`centrality`	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: `"median"`, `"mean"`, `"MAP"` or `"all"`.
`dispersion`	Logical, if `TRUE`, computes indices of dispersion related to the estimate(s) (`SD` and `MAD` for `mean` and `median`, respectively).
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`ci_method`	The type of index used for Credible Interval. Can be `"HDI"` (default, see `hdi`) or `"ETI"` (see `eti`).

test	The indices of effect existence to compute. Character (vector) or list with one or more of these options: "p_direction" (or "pd"), "rope", "p_map", "equivalence_test" (or "equitest"), "bayesfactor" (or "bf") or "all" to compute all tests. For each "test", the corresponding **bayestestR** function is called (e.g. rope or p_direction) and its results included in the summary output.
rope_range	ROPE's lower and higher bounds. Should be a list of two values (e.g., c(-0.1,0.1)) or "default". If "default", the bounds are set to x +-0.1*SD(response).
rope_ci	The Credible Interval (CI) probability, corresponding to the proportion of HDI, to use for the percentage in ROPE.
bf_prior	Distribution representing a prior for the computation of Bayes factors. Used if the input is a posterior, otherwise (in the case of models) ignored.
diagnostic	Diagnostic metrics to compute. Character (vector) or list with one or more of these options: "ESS", "Rhat", "MCSE" or "all".
priors	Add the prior used for each parameter.
standardize	Add standardized parameters. Can be FALSE or a character indicating the standardization method (see parameters_standardize()), such as "refit", "2sd", "smart" or "classic". The two former are based on model refitting using a standardized version of data. It is the most accurate, although computationally heavy (as it must re-fit a second model). The "smart" and "classic" are post-hoc methods, fast, but inaccurate (especially if the model includes interactions).
standardize_robust	Robust standardization. See parameters_standardize.
iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
...	Arguments passed to or from other methods (e.g., to standardize()).

Value

A data.frame of indices related to the model's parameters.

See Also

standardize_names() to rename columns into a consistent, standardized naming scheme.

Examples

```
library(parameters)
library(rstanarm)

model <- rstanarm::stan_glm(Sepal.Length ~ Petal.Length * Species,
  data = iris, iter = 500, refresh = 0
)

model_parameters(model, standardize = "smart")
```

model_parameters.zeroinfl

Model Parameters for Zero-Inflated Models

Description

Parameters of zero-inflated models.

Usage

```
## S3 method for class 'zeroinfl'
model_parameters(model, ci = 0.95,
  standardize = "refit", standardize_robust = FALSE,
  bootstrap = FALSE, iterations = 1000, component = c("all",
  "conditional", "zi", "zero_inflated"), ...)
```

Arguments

model	A model with zero-inflation component.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
standardize	Add standardized parameters. Can be FALSE or a character indicating the standardization method (see parameters_standardize()), such as "refit", "2sd", "smart" or "classic". The two former are based on model refitting using a standardized version of data. It is the most accurate, although computationally heavy (as it must re-fit a second model). The "smart" and "classic" are post-hoc methods, fast, but inaccurate (especially if the model includes interactions).
standardize_robust	Robust standardization. See parameters_standardize.
bootstrap	Should estimates be based on bootstrapped model? If TRUE, then arguments of Bayesian regressions apply (see also parameters_bootstrap()).
iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
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). May be abbreviated.
...	Arguments passed to or from other methods (e.g., to standardize()).

Value

A data frame of indices related to the model's parameters.

See Also

standardize_names() to rename columns into a consistent, standardized naming scheme.

Examples

```
library(parameters)
library(pscl)

data("bioChemists")
model <- zeroinfl(art ~ fem + mar + kid5 + ment | kid5 + phd, data = bioChemists)
model_parameters(model)
```

`model_simulate` *Simulated draws from model coefficients*

Description

Simulate draws from a statistical model to return a data.frame of estimates.

Usage

```
model_simulate(model, iterations = 1000, ...)

## S3 method for class 'glmmTMB'
model_simulate(model, iterations = 1000,
  component = c("all", "conditional", "zi", "zero_inflated"), ...)
```

Arguments

`model`	Statistical model (no Bayesian models).
`iterations`	The number of draws to simulate/bootstrap.
`...`	Arguments passed to or from other methods.
`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). May be abbreviated.

Details

Technical Details: `model_simulate()` is a computationally faster alternative to `model_bootstrap()`. Simulated draws for coefficients are based on a multivariate normal distribution (`MASS::mvrnorm()`) with mean `mu = coef(model)` and variance `Sigma = vcov(model)`.

Models with Zero-Inflation Component: For models from packages **glmmTMB**, **pscl**, **GLMMadaptive** and **countreg**, the `component` argument can be used to specify which parameters should be simulated. For all other models, parameters from the conditional component (fixed effects) are simulated. This may include smooth terms, but not random effects.

Value

A data frame.

See Also

parameters_simulate(), model_bootstrap(), parameters_bootstrap()

Examples

```
library(parameters)
library(glmmTMB)

model <- lm(Sepal.Length ~ Species * Petal.Width + Petal.Length, data = iris)
head(model_simulate(model))

model <- glmmTMB(
  count ~ spp + mined + (1 | site),
  ziformula = ~mined,
  family = poisson(),
  data = Salamanders
)
head(model_simulate(model))
head(model_simulate(model, component = "zero_inflated"))
```

---

normalize *Normalization*

Description

Performs a normalization of data. This scales all numeric variables in the range 0 - 1.

Usage

```
normalize(x, ...)

## S3 method for class 'numeric'
normalize(x, include_bounds = TRUE, verbose = TRUE,
  ...)

## S3 method for class 'grouped_df'
normalize(x, select = NULL, exclude = NULL,
  include_bounds = TRUE, ...)

## S3 method for class 'data.frame'
normalize(x, select = NULL, exclude = NULL,
  include_bounds = TRUE, ...)
```

Arguments

x	Object.
...	Arguments passed to or from other methods.

include_bounds — Logical, if TRUE, return value may include 0 and 1. If FALSE, the return value is compressed, using the formula (x * (n -1) + 0.5) / n (*Smithson and Verkuilen 2006*), to avoid zeros and ones in the normalized variables. This can be useful in case of beta-regression, where the response variable is not allowed to include zeros and ones.

verbose — Toggle warnings on or off.

select — For a data frame, character vector of column names to be standardized. If NULL (the default), all variables will be standardized.

exclude — For a data frame, character vector of column names to be excluded from standardization.

Value

A normalized object.

References

Smithson M, Verkuilen J (2006). A Better Lemon Squeezer? Maximum-Likelihood Regression with Beta-Distributed Dependent Variables. Psychological Methods, 11(1), 54–71.

---

n_factors *Number of Components/Factors to Retain*

Description

This function runs many existing procedures for determining how many factors to retain for your factor analysis (FA) or dimension reduction (PCA). It returns the number of factors based on the maximum consensus. In case of ties, it will select the solution with the less factors.

Usage

```
n_factors(x, type = "FA", rotation = "varimax",
  algorithm = "default", package = c("nFactors", "psych"),
  safe = TRUE, ...)
```

Arguments

x — A dataframe.

type — Can be "FA" or "PCA", depending on what you want to do.

rotation — Only used for VSS (Very Simple Structure criterion, see VSS. The rotation to apply. Can be "none", "varimax", "quartimax", "bentlerT", "equamax", "varimin", "geominT" and "bifactor" for orthogonal rotations, and "promax", "oblimin", "simplimax", "bentlerQ", "geominQ", "biquartimin" and "cluster" for oblique transformations.

algorithm	Factoring method used by VSS. Can be "pa" for Principal Axis Factor Analysis, "minres" for minimum residual (OLS) factoring, "mle" for Maximum Likelihood FA and "pc" for Principal Components. "default" will select "minres" if type = "FA" and "pc" if type = "PCA".
package	These are the packages from which methods are used. Can be "all" or a vector containing "nFactors", "psych" and "EGAnet". However, "EGAnet" can be very slow for bigger datasets. Thus, by default, c("nFactors","psych") are selected.
safe	If TRUE, will run all the procedures in try blocks, and will only return those that work and silently skip the ones that may fail.
...	Arguments passed to or from other methods.

Value

A data.frame.

References

- Bartlett, M. S. (1950). Tests of significance in factor analysis. British Journal of statistical psychology, 3(2), 77-85.
- Bentler, P. M., & Yuan, K. H. (1996). Test of linear trend in eigenvalues of a covariance matrix with application to data analysis. British Journal of Mathematical and Statistical Psychology, 49(2), 299-312.
- Cattell, R. B. (1966). The scree test for the number of factors. Multivariate behavioral research, 1(2), 245-276.
- Zoski, K. W., & Jurs, S. (1996). An objective counterpart to the visual scree test for factor analysis: The standard error scree. Educational and Psychological Measurement, 56(3), 443-451.
- Zoski, K., & Jurs, S. (1993). Using multiple regression to determine the number of factors to retain in factor analysis. Multiple Linear Regression Viewpoints, 20(1), 5-9.
- Nasser, F., Benson, J., & Wisenbaker, J. (2002). The performance of regression-based variations of the visual scree for determining the number of common factors. Educational and psychological measurement, 62(3), 397-419.
- Golino, H., Shi, D., Garrido, L. E., Christensen, A. P., Nieto, M. D., Sadana, R., & Thiyagarajan, J. A. (2018). Investigating the performance of Exploratory Graph Analysis and traditional techniques to identify the number of latent factors: A simulation and tutorial.
- Golino, H. F., & Epskamp, S. (2017). Exploratory graph analysis: A new approach for estimating the number of dimensions in psychological research. PloS one, 12(6), e0174035.
- Revelle, W., & Rocklin, T. (1979). Very simple structure: An alternative procedure for estimating the optimal number of interpretable factors. Multivariate Behavioral Research, 14(4), 403-414.
- Velicer, W. F. (1976). Determining the number of components from the matrix of partial correlations. Psychometrika, 41(3), 321-327.

Examples

```
library(parameters)

n_factors(mtcars, type = "PCA")

result <- n_factors(mtcars[, 1:5], type = "FA")
as.numeric(result)
summary(result)

n_factors(mtcars, type = "PCA", package = "all")
n_factors(mtcars, type = "FA", algorithm = "mle", package = "all")
```

---

n_parameters *Count how many parameters in a model*

Description

Returns the number of parameters of a model.

Usage

```
n_parameters(x, ...)
```

Arguments

x	A statistical model.
...	Arguments passed to or from other methods.

Value

The number of parameters in the model.

---

odds_to_d *Conversion between (log)odds and standardized difference*

Description

Enables a conversion between (log)odds and standardized difference using Cohen's (1988) formula $\frac{\log(odds) \times \sqrt{3}}{\pi}$.

Usage

```
odds_to_d(odds, log = FALSE)

convert_odds_to_d(odds, log = FALSE)

d_to_odds(d, log = FALSE)

convert_d_to_odds(d, log = FALSE)
```

Arguments

odds	Odds values in vector or dataframe.
log	Are these Log odds (such as in logistic models)?
d	standardized difference values in vector or dataframe.

Value

Converted index.

References

- Sánchez-Meca, J., Marín-Martínez, F., & Chacón-Moscoso, S. (2003). Effect-size indices for dichotomized outcomes in meta-analysis. Psychological methods, 8(4), 448.
- Borenstein, Michael, et al. "Converting among effect sizes." Introduction to meta-analysis (2009): 45-49.

Examples

```
odds_to_d(0.2)
odds_to_d(-1.45, log = TRUE)
```

---

odds_to_probs *Conversion between (log)odds and probabilities*

Description

Enables a conversion between (log)odds and probabilities.

Usage

```
odds_to_probs(x, log = FALSE, ...)

## S3 method for class 'data.frame'
odds_to_probs(x, log = FALSE, select = NULL,
  exclude = NULL, ...)
```

```
probs_to_odds(x, log = FALSE, ...)

convert_odds_to_probs(x, log = FALSE, ...)

convert_probs_to_odds(x, log = FALSE, ...)
```

Arguments

x	Odds or probs values in vector or dataframe.
log	Are these Log odds (such as in logistic models)?
...	Arguments passed to or from other methods.
select	Character or list of of column names to be transformed.
exclude	Character or list of column names to be excluded from transformation.

Value

Converted index.

Examples

```
odds_to_probs(-1.45)
odds_to_probs(3.22)
```

---

parameters_bootstrap *Parameters bootstrapping*

Description

Compute bootstrapped parameters and their related indices such as Confidence Intervals (CI) and p-values.

Usage

```
parameters_bootstrap(model, iterations = 1000, centrality = "median",
  ci = 0.95, ci_method = "quantile", test = "p-value", ...)
```

Arguments

model	Statistical model.
iterations	The number of draws to simulate/bootstrap.
centrality	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: "median", "mean", "MAP" or "all".
ci	Value or vector of probability of the CI (between 0 and 1) to be estimated. Default to .89 (89%) for Bayesian models and .95 (95%) for frequentist models.
ci_method	The type of index used for Credible Interval. Can be "HDI" (default, see hdi) or "ETI" (see eti).

test	The indices of effect existence to compute. Character (vector) or list with one or more of these options: "p_direction" (or "pd"), "rope", "p_map", "equivalence_test" (or "equitest"), "bayesfactor" (or "bf") or "all" to compute all tests. For each "test", the corresponding **bayestestR** function is called (e.g. rope or p_direction) and its results included in the summary output.
...	Arguments passed to or from other methods.

Value

Bootstrapped parameters.

References

Davison, A. C., & Hinkley, D. V. (1997). Bootstrap methods and their application (Vol. 1). Cambridge university press.

See Also

model_bootstrap, parameters_simulate, model_simulate

Examples

```
library(parameters)

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

---

parameters_reduction *Dimensionality reduction (DR) / Features Reduction*

Description

This function performs a reduction in the parameters space (the number of variables). It starts by creating a new set of variables, based on a given method (the default method is "PCA", but other are available via the method argument, such as "cMDS", "DRR" or "ICA"). Then, it names this new dimensions using the original variables that correlates the most with it. For instance, a variable named 'V1_0.97/V4_-0.88' means that the V1 and the V4 variables correlate maximally (with respective coefficients of .97 and -.88) with this dimension. Although this function can be useful in exploratory data analysis, it's best to perform the dimension reduction step in a separate and dedicated stage, as this is a very important process in the data analysis workflow.

Usage

```
parameters_reduction(x, method = "PCA", n = "max", ...)
```

Arguments

x	A dataframe or a statistical model.
method	The features reduction method. Can be one of 'PCA', 'cMDS', 'DRR', 'ICA' (see the Details section).
n	Number of components to extract. If n = "all" =, then n is set as the number of variables minus 1 (ncol(x)-1). If n = "auto" (default) or n = NULL, the number of components is selected through n_factors. In parameters_reduction, can also be "max", in which case it will select all the components that are maximally pseudo-loaded (i.e., correlated) by at least one variable.
...	Arguments passed to or from other methods.

Details

The different methods available are described below:

Supervised Methods:

- **PCA**: See principal_components.
- **cMDS / PCoA**: See cmds. Classical Multidimensional Scaling (cMDS) takes a set of dissimilarities (i.e., a distance matrix) and returns a set of points such that the distances between the points are approximately equal to the dissimilarities.
- **DRR**: See DRR. Dimensionality Reduction via Regression (DRR) is a very recent technique extending PCA (Laparra et al., 2015). Starting from a rotated PCA, it predicts redundant information from the remaining components using non-linear regression. Some of the most notable advantages of performing PCR are avoidance of multicollinearity between predictors and overfitting mitigation. PCR tends to perform well when the first principal components are enough to explain most of the variation in the predictors. Requires the **DRR** package to be installed.
- **ICA**: See ICA. Performs an Independent Component Analysis using the FastICA algorithm. Contrary to PCA, that attempts to find uncorrelated sources (through least squares minimization), ICA attempts to find independent sources, i.e., the source space that maximizes the "non-gaussianity" of all sources. Contrary to PCA, ICA does not rank each source, which makes it a poor tool for dimensionality reduction. Requires the **fastICA** package to be installed.

References

- Nguyen, L. H., \& Holmes, S. (2019). Ten quick tips for effective dimensionality reduction. PLOS Computational Biology, 15(6).
- Laparra, V., Malo, J., & Camps-Valls, G. (2015). Dimensionality reduction via regression in hyperspectral imagery. IEEE Journal of Selected Topics in Signal Processing, 9(6), 1026-1036.

Examples

```
parameters_reduction(iris, method = "PCA", n = "max")
```

parameters_selection *Parameters Selection*

Description

This function performs an automated selection of the 'best' parameters, updating and returning the "best" model. For frequentist simple GLMs, it performs an AIC-based stepwise selection. For Bayesian models, it uses the `projpred` package.

Usage

```
parameters_selection(model, ...)

## S3 method for class 'lm'
parameters_selection(model, direction = "both",
  steps = 1000, k = 2, ...)

## S3 method for class 'merMod'
parameters_selection(model, direction = "backward",
  steps = 1000, ...)

## S3 method for class 'stanreg'
parameters_selection(model, method = NULL,
  cross_validation = FALSE, ...)
```

Arguments

`model`	A statistical model.
`...`	Arguments passed to or from other methods.
`direction`	the mode of stepwise search, can be one of `"both"`, `"backward"`, or `"forward"`, with a default of `"both"`. If the `scope` argument is missing the default for `direction` is `"backward"`. Values can be abbreviated.
`steps`	the maximum number of steps to be considered. The default is 1000 (essentially as many as required). It is typically used to stop the process early.
`k`	the multiple of the number of degrees of freedom used for the penalty. Only `k = 2` gives the genuine AIC: `k = log(n)` is sometimes referred to as BIC or SBC.
`method`	The method used in the variable selection. Can be NULL (default), "forward" or "L1". See `projpred::varsel`.
`cross_validation`	Select with cross-validation.

Value

The model refitted with optimal number of parameters.

Examples

```
model <- lm(mpg ~ ., data = mtcars)
parameters_selection(model)

model <- lm(mpg ~ cyl * disp * hp * wt, data = mtcars)
parameters_selection(model)

# lme4 ----------------------------------------
library(lme4)
model <- lmer(Sepal.Width ~ Sepal.Length * Petal.Width * Petal.Length + (1 | Species), data = iris)
parameters_selection(model)

# rstanarm ------------------------------------------
library(rstanarm)
model <- stan_glm(mpg ~ ., data = mtcars, iter = 500, refresh = 0)
parameters_selection(model, cross_validation = TRUE)

model <- stan_glm(mpg ~ cyl * disp * hp, data = mtcars)
parameters_selection(model, cross_validation = FALSE)
```

---

parameters_simulate *Parameters simulation*

Description

Compute simulated draws of parameters and their related indices such as Confidence Intervals (CI) and p-values. Simulating parameter draws can be seen as a (computationally faster) alternative to bootstrapping.

Usage

```
parameters_simulate(model, iterations = 1000, centrality = "median",
  ci = 0.95, ci_method = "quantile", test = "p-value", ...)
```

Arguments

model	Statistical model (no Bayesian models).
iterations	The number of draws to simulate/bootstrap.
centrality	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: "median", "mean", "MAP" or "all".
ci	Value or vector of probability of the CI (between 0 and 1) to be estimated. Default to .89 (89%) for Bayesian models and .95 (95%) for frequentist models.
ci_method	The type of index used for Credible Interval. Can be "HDI" (default, see hdi) or "ETI" (see eti).

test	The indices of effect existence to compute. Character (vector) or list with one or more of these options: "p_direction" (or "pd"), "rope", "p_map", "equivalence_test" (or "equitest"), "bayesfactor" (or "bf") or "all" to compute all tests. For each "test", the corresponding **bayestestR** function is called (e.g. rope or p_direction) and its results included in the summary output.
...	Arguments passed to or from other methods.

Details

Technical Details: model_simulate() is a computationally faster alternative to model_bootstrap(). Simulated draws for coefficients are based on a multivariate normal distribution (MASS::mvrnorm()) with mean mu = coef(model) and variance Sigma = vcov(model).

Models with Zero-Inflation Component: For models from packages **glmmTMB**, **pscl**, **GLMMadaptive** and **countreg**, the component argument can be used to specify which parameters should be simulated. For all other models, parameters from the conditional component (fixed effects) are simulated. This may include smooth terms, but not random effects.

Value

A data frame with simulated parameters.

References

Gelman A, Hill J. Data analysis using regression and multilevel/hierarchical models. Cambridge; New York: Cambridge University Press 2007: 140-143

See Also

model_bootstrap, parameters_bootstrap, model_simulate

Examples

```
library(parameters)
library(glmmTMB)

model <- lm(Sepal.Length ~ Species * Petal.Width + Petal.Length, data = iris)
parameters_simulate(model)

model <- glmmTMB(
  count ~ spp + mined + (1 | site),
  ziformula = ~mined,
  family = poisson(),
  data = Salamanders
)
parameters_simulate(model, centrality = "mean")
parameters_simulate(model, ci = c(.8, .95), component = "zero_inflated")
```

`parameters_standardize` *Parameters standardization*

Description

Compute standardized model parameters (coefficients).

Usage

```
parameters_standardize(model, robust = FALSE, method = "refit",
  verbose = TRUE, ...)
```

Arguments

`model`	A statistical model.
`robust`	Logical, if `TRUE`, centering is done by substracting the median from the variables and divide it by the median absolute deviation (MAD). If `FALSE`, variables are standardized by substracting the mean and divide it by the standard deviation (SD).
`method`	The method used for standardizing the parameters. Can be `"refit"` (default), `"2sd"`, `"smart"` or `"classic"`.
`verbose`	Toggle warnings on or off.
`...`	Arguments passed to or from other methods.

Details

Methods:

- **refit**: This method is based on a complete model re-fit with a standardized version of data. Hence, this method is equal to standardizing the variables before fitting the model. It is the "purest" and the most accurate (Neter et al., 1989), but it is also the most computationally costly and long (especially for Bayesian models). This method is particularly recommended for complex models that include interactions or transformations (e.g., polynomial or spline terms). The `robust` (default to `FALSE`) argument enables a robust standardization of data, i.e., based on the `median` and `MAD` instead of the `mean` and `SD`.
- **2sd**: Same as `method = "refit"`, however, standardization is done by dividing by two times the `SD` or `MAD` (depending on `robust`). This method is useful to obtain coefficients of continuous parameters comparable to coefficients related to binary predictors (see Gelman, 2008).
- **smart** (Standardization of Model's parameters with Adjustment, Reconnaissance and Transformation): Post-hoc standardization of the parameters, aiming at emulating the results obtained by "refit". The coefficients are divided by the standard deviation (or MAD if `robust`) of the outcome (which becomes their expression 'unit'). Then, the coefficients related to numeric variables are additionally multiplied by the standard deviation (or MAD if `robust`) of the related term, so that they correspond to changes of 1 SD of the predictor (e.g., "A change in 1 SD of `x` is related to a change of 0.24 of the SD of `y`). This does not apply to binary variables or factors, so the coefficients are still related to changes in levels.

- **classic**: This method is similar to `method = "smart"`, but treats all variables as continuous: it also scales the coefficient by the standard deviation of model's matrix' parameter of factors levels (transformed to integers) or binary predictors. Although being inappropriate for these cases, this method is the one implemented by default in other software packages, such as `sjstats::std_beta()` or `lm.beta::lm.beta()`.

Value

Standardized parameters.

References

- Neter, J., Wasserman, W., & Kutner, M. H. (1989). Applied linear regression models.
- Gelman, A. (2008). Scaling regression inputs by dividing by two standard deviations. Statistics in medicine, 27(15), 2865-2873.

Examples

```
library(parameters)
data(iris)

model <- lm(Sepal.Length ~ Species * Petal.Width, data = iris)
parameters_standardize(model, method = "refit")
parameters_standardize(model, method = "refit", robust = TRUE)
parameters_standardize(model, method = "2sd")
parameters_standardize(model, method = "2sd", robust = TRUE)
parameters_standardize(model, method = "smart")
parameters_standardize(model, method = "smart", robust = TRUE)

iris$binary <- ifelse(iris$Sepal.Width > 3, 1, 0)
model <- glm(binary ~ Species * Sepal.Length, data = iris, family = "binomial")
parameters_standardize(model, method = "refit")
parameters_standardize(model, method = "refit", robust = TRUE)
parameters_standardize(model, method = "smart")
parameters_standardize(model, method = "smart", robust = TRUE)

library(rstanarm)
model <- stan_glm(Sepal.Length ~ Species * Petal.Width, data = iris, iter = 500, refresh = 0)
parameters_standardize(model, method = "smart", centrality = "all")
parameters_standardize(model, method = "smart", robust = TRUE, centrality = "all")
```

---

parameters_table *Parameters Table Formatting*

Description

Parameters Table Formatting

Usage

```
parameters_table(x, pretty_names = TRUE, stars = FALSE, ...)
```

Arguments

x	A dataframe of model's parameters.
pretty_names	Pretty parameters' names.
stars	Add significance stars (e.g., p < .001***).
...	Arguments passed to or from other methods.

Value

A data.frame.

Examples

```
library(parameters)

x <- model_parameters(lm(Sepal.Length ~ Species * Sepal.Width, data = iris))
as.data.frame(parameters_table(x))

library(rstanarm)
x <- model_parameters(stan_glm(Sepal.Length ~ Species, data = iris), ci = c(0.69, 0.89, 0.95))
as.data.frame(parameters_table(x))
```

---

parameters_type *Type of Model Parameters*

Description

Type of Model Parameters

Usage

```
parameters_type(model, ...)
```

Arguments

model	A statistical model.
...	Arguments passed to or from other methods.

Value

A data.frame.

Examples

```
library(parameters)

model <- lm(Sepal.Length ~ Petal.Length + Species, data = iris)
parameters_type(model)

model <- lm(Sepal.Length ~ Species + poly(Sepal.Width, 2), data = iris)
parameters_type(model)

model <- lm(Sepal.Length ~ Species + poly(Sepal.Width, 2, raw = TRUE), data = iris)
parameters_type(model)

# Interactions
model <- lm(Sepal.Length ~ Sepal.Width * Species, data = iris)
parameters_type(model)

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

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

model <- lm(Sepal.Length ~ Species / Sepal.Width, data = iris)
parameters_type(model)


# Complex interactions
data <- iris
data$fac2 <- ifelse(data$Sepal.Width > mean(data$Sepal.Width), "A", "B")
model <- lm(Sepal.Length ~ Species / fac2 / Petal.Length, data = data)
parameters_type(model)

model <- lm(Sepal.Length ~ Species / fac2 * Petal.Length, data = data)
parameters_type(model)
```

---

principal_components *Principal Component Analysis (PCA)*

Description

This function performs a principal component analysis (PCA) and returns the loadings as a dataframe.

Usage

```
principal_components(x, n = "auto", rotation = "none", sort = FALSE,
  threshold = NULL, standardize = TRUE, ...)
```

Arguments

x	A dataframe or a statistical model.
n	Number of components to extract. If n = "all" =, then n is set as the number of variables minus 1 (ncol(x)-1). If n = "auto" (default) or n = NULL, the number of components is selected through n_factors. In parameters_reduction, can also be "max", in which case it will select all the components that are maximally pseudo-loaded (i.e., correlated) by at least one variable.
rotation	If not "none", the PCA will be computed using the **psych** package. Possible options include "varimax", "quartimax", "promax", "oblimin", "simplimax", and "cluster". See fa for details.
sort	Sort the loadings.
threshold	A value between 0 and 1 indicates which (absolute) values from the loadings should be removed. An integer higher than 1 indicates the n strongest loadings to retain. Can also be "max", in which case it will only display the maximum loading per variable (the most simple structure).
standardize	A logical value indicating whether the variables should be standardized (centred and scaled) to have unit variance before the analysis takes place (in general, such scaling is advisable).
...	Arguments passed to or from other methods.

Details

- **Complexity** (Hoffman's, 1978; Pettersson and Turkheimer, 2010) represents the number of latent components needed to account for the observed variables. Whereas a perfect simple structure solution has a complexity of 1 in that each item would only load on one factor, a solution with evenly distributed items has a complexity greater than 1.

Value

A data.frame of loadings.

Note

There is a summary()-method that prints the Eigenvalues and (explained) variance for each extracted component.

References

- Pettersson, E., & Turkheimer, E. (2010). Item selection, evaluation, and simple structure in personality data. Journal of research in personality, 44(4), 407-420.

Examples

```
library(parameters)

principal_components(mtcars[, 1:7], n = "all", threshold = 0.2)
principal_components(mtcars[, 1:7], n = 2, rotation = "oblimin", threshold = "max", sort = TRUE)
principal_components(mtcars[, 1:7], n = 2, threshold = 2, sort = TRUE)
```

```
pca <- principal_components(mtcars[, 1:5], n = 2)
summary(pca)
predict(pca)

# Automated number of components
principal_components(mtcars[, 1:4], n = "auto")
```

---

print *Print model parameters*

Description

A `print()`-method for objects from `model_parameters()`.

Usage

```
## S3 method for class 'parameters_model'
print(x, pretty_names = TRUE,
  split_components = TRUE, ...)
```

Arguments

`x`	An object returned by `model_parameters()`.
`pretty_names`	Pretty parameters' names.
`split_components`	Logical, if `TRUE` (default), For models with multiple components (zero-inflation, smooth terms, ...), each component is printed in a separate table. If `FALSE`, model parameters are printed in a single table and a `Component` column is added to the output.
`...`	Arguments passed to or from other methods.

Value

`NULL`

Examples

```
library(parameters)
library(glmmTMB)

model <- glmmTMB(
  count ~ spp + mined + (1 | site),
  ziformula = ~mined,
  family = poisson(),
  data = Salamanders
```

```
)
mp <- model_parameters(model)

print(mp, pretty_names = FALSE)

print(mp, split_components = FALSE)
```

---

p_value *p-values*

Description

This function attempts to return, or compute, p-values of a model's parameters. The nature of the p-values is different depending on the model:

- Mixed models (lme4): TO BE IMPROVED.

Usage

```
p_value(model, ...)

## S3 method for class 'lmerMod'
p_value(model, method = "wald", ...)

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

## S3 method for class 'MixMod'
p_value(model, component = c("all", "conditional", "zi",
  "zero_inflated"), ...)
```

Arguments

model	A statistical model.
...	Arguments passed to or from other methods.
method	For mixed models, can be "wald" (default) or "kenward".
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). May be abbreviated.

Value

The p-values.

Examples

```
model <- lme4::lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
p_value(model)
```

`p_value_kenward` *p-values using Kenward-Roger approximation*

Description

An approximate F-test based on the Kenward-Roger (1997) approach.

Usage

```
p_value_kenward(model, dof = NULL)

dof_kenward(model)

se_kenward(model)
```

Arguments

`model`	A statistical model.
`dof`	Degrees of Freedom.

Details

`dof_kenward()` and `se_kenward()` are small helper-functions to calculate approximated degrees of freedom and standard errors for model parameters, based on the Kenward-Roger (1997) approach.

Value

The p-values.

References

Kenward, M. G., & Roger, J. H. (1997). Small sample inference for fixed effects from restricted maximum likelihood. Biometrics, 983-997.

Examples

```
library(lme4)
model <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
p_value_kenward(model)
```

reshape_loadings *Reshape loadings between wide/long formats*

Description

Reshape loadings between wide/long formats.

Usage

```
reshape_loadings(x, ...)

## S3 method for class 'parameters_efa'
reshape_loadings(x, threshold = NULL, ...)

## S3 method for class 'data.frame'
reshape_loadings(x, threshold = NULL,
  loadings_columns = NULL, ...)
```

Arguments

x	A dataframe or a statistical model.
...	Arguments passed to or from other methods.
threshold	A value between 0 and 1 indicates which (absolute) values from the loadings should be removed. An integer higher than 1 indicates the n strongest loadings to retain. Can also be "max", in which case it will only display the maximum loading per variable (the most simple structure).
loadings_columns	Vector indicating the columns corresponding to loadings.

Examples

```
library(parameters)

library(psych)

pca <- model_parameters(psych::fa(attitude, nfactors = 3))
loadings <- reshape_loadings(pca)

loadings
reshape_loadings(loadings)
```

skewness *Compute Skewness and Kurtosis*

Description

Compute Skewness and Kurtosis

Usage

```
skewness(x, na.rm = TRUE, ...)

kurtosis(x, ...)
```

Arguments

x	A numeric vector or data.frame.
na.rm	Remove missing values.
...	Arguments passed to or from other methods.

Details

Skewness

Symmetric distributions have a skewness around zero, while a negative skewness values indicates a "left-skewed" distribution, and a positive skewness values indicates a "right-skewed" distribution. Examples for the relationship of skewness and distributions are:

- Normal distribution (and other symmetric distribution) has a skewness of 0
- Half-normal distribution has a skewness just below 1
- Exponential distribution has a skewness of 2
- Lognormal distribution can have a skewness of any positive value, depending on its parameters

(*https://en.wikipedia.org/wiki/Skewness*)

Kurtosis

The kurtosis is a measure of "tailedness" of a distribution. A distribution with a kurtosis values of about zero is called "mesokurtic". A kurtosis value larger than zero indicates a "leptokurtic" distribution with *fatter* tails. A kurtosis value below zero indicates a "platykurtic" distribution with *thinner* tails (*https://en.wikipedia.org/wiki/Kurtosis*).

Value

Values of skewness or kurtosis.

Examples

```
skewness(rnorm(1000))
kurtosis(rnorm(1000))
```

`standardize` *Standardization (Z-scoring)*

Description

Performs a standardization of data (Z-scoring), i.e., centred and scaled, so that the data is expressed in terms of standard deviation (i.e., mean = 0, SD = 1) or Median Absolute Deviance (median = 0, MAD = 1). When applied to a statistical model, this function extracts the dataset, standardizes it, and refits the model with this standardized version of the dataset. The `normalize` function can also be used to scale all numeric variables within the 0 - 1 range.

Usage

```
standardize(x, ...)

## S3 method for class 'numeric'
standardize(x, robust = FALSE, method = "default",
  verbose = TRUE, ...)

## S3 method for class 'factor'
standardize(x, force = FALSE, ...)

## S3 method for class 'data.frame'
standardize(x, robust = FALSE, method = "default",
  select = NULL, exclude = NULL, verbose = TRUE, force = FALSE,
  ...)

## S3 method for class 'lm'
standardize(x, robust = FALSE, method = "default",
  include_response = TRUE, verbose = TRUE, ...)
```

Arguments

`x`	A dataframe, a vector or a statistical model.
`...`	Arguments passed to or from other methods.
`robust`	Logical, if `TRUE`, centering is done by substracting the median from the variables and divide it by the median absolute deviation (MAD). If `FALSE`, variables are standardized by substracting the mean and divide it by the standard deviation (SD).
`method`	The method of standardization. For data.frames, can be `"default"` (variables are centred by mean or median, and divided by SD or MAD, depending on `robust`) or `"2sd"`, in which case they are divided by two times the deviation (SD or MAD, again depending on `robust`).

verbose — Toggle warnings on or off.

force — Logical, if TRUE, forces standardization of factors as well. Factors are converted to numerical values, with the lowest level being the value 1 (unless the factor has numeric levels, which are converted to the corresponding numeric value).

select — For a data frame, character vector of column names to be standardized. If NULL (the default), all variables will be standardized.

exclude — For a data frame, character vector of column names to be excluded from standardization.

include_response — For a model, if TRUE (default), the response value will also be standardized. If FALSE, only the predictors will be standardized. Note that for certain models (logistic regression, count models, ...), the response value will never be standardized, to make re-fitting the model work.

Value

The standardized object (either a standardize dataframe or a statistical model fitted on standardized data).

See Also

normalize parameters_standardize

Examples

```
# Dataframes
summary(standardize(iris))

# Models
model <- lm(Sepal.Length ~ Species * Petal.Width, data = iris)
coef(standardize(model))
```

---

standardize_names *Standardize column names*

Description

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

Usage

```
standardize_names(data, ...)

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

Arguments

data	A data frame. Currently, only objects from model_parameters() are accepted.
...	Currently not used.
style	Standardization can either be based on the naming conventions from the easystats project, or on **broom**'s naming scheme.

Details

This method is in particular useful for package developers or users who use 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.

Value

A data frame, with standardized column names.

Examples

```
library(parameters)
model <- lm(mpg ~ wt + cyl, data = mtcars)
mp <- model_parameters(model)

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

---

standard_error *Extract standard errors*

Description

This function attempts to return standard errors of model parameters.

Usage

```
standard_error(model, ...)

se(model, ...)

## S3 method for class 'factor'
standard_error(model, force = FALSE, verbose = TRUE,
```

```
  ...)

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

## S3 method for class 'MixMod'
standard_error(model, component = c("all",
  "conditional", "zi", "zero_inflated"), ...)
```

Arguments

model	A model.
...	Arguments passed to or from other methods.
force	Logical, if TRUE, factors are converted to numerical values to calculate the standard error, with the lowest level being the value 1 (unless the factor has numeric levels, which are converted to the corresponding numeric value). By default, NA is returned for factors or character vectors.
verbose	Toggle warnings on or off.
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). May be abbreviated.

Value

A data frame.

Examples

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

Index