Package 'parameters'

September 21, 2020

Type Package

Title Processing of Model Parameters

Version 0.8.6

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 bootstrapping or simulating of parameters and models, feature reduction (feature extraction and variable selection) as well as functions to describe data and variable characteristics (e.g. skewness, kurtosis, smoothness or distribution).

License GPL-3

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

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

Depends R (>= 3.2)

Imports bayestestR (>= 0.7.0), insight (>= 0.8.1), methods, stats, tools, utils

Suggests AER, aod, BayesFactor, BayesFM, bbmle, betareg, boot, brglm2, brms, cAIC4, car, clubSandwich, cluster, covr, cplm, dplyr, DRR, effectsize (>= 0.3.0), EGAnet (>= 0.7), FactoMineR, fastICA, gamlss, gee, geepack, ggplot2, GLMMadaptive, glmmTMB, GPArotation, lavaan, lavaSearch2, lfe, lme4, lmerTest, logspline, knitr, MASS, Matrix, mclust, MCMCglmm, metafor, mice, mfx, mgcv, multcomp, multimode, MuMIn, M3C, NbClust, nFactors, nlme, performance, plm, projpred, pscl, psych, quantreg, randomForest, rmarkdown, rstanarm, sandwich, see, sjstats, spelling, survey, survival, testthat, TMB, tripack, truncreg, VGAM

Encoding UTF-8

Language en-US

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>),
Dominique Makowski [aut] (<https://orcid.org/0000-0001-5375-9967>),
Mattan S. Ben-Shachar [aut] (<https://orcid.org/0000-0002-4287-4801>),
Indrajeet Patil [aut] (<https://orcid.org/0000-0003-1995-6531>),
Søren Højsgaard [aut],
Zen J. Lau [ctb]

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

Repository CRAN

Date/Publication 2020-09-21 16:30:03 UTC

R topics documented:

`bootstrap_model` *Model bootstrapping*

Description

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

Usage

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

Arguments

model	Statistical model.
iterations	The number of draws to simulate/bootstrap.
verbose	Show or hide possible warnings and messages.
...	Arguments passed to or from other methods.

Value

A data frame.

See Also

bootstrap_parameters, simulate_model, simulate_parameters

Examples

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

---

bootstrap_parameters *Parameters bootstrapping*

Description

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

Usage

```
bootstrap_parameters(
  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), "ETI" (see eti) or "SI" (see si).
test	The indices to compute. Character (vector) with one or more of these options: "p-value" (or "p"), "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

This function first calls bootstrap_model to generate bootstrapped coefficients. The resulting replicated for each coefficient are treated as "distribution", and is passed to describe_posterior() to calculate the related indices defined in the "test" argument.

Value

Bootstrapped parameters.

References

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

See Also

bootstrap_model, simulate_parameters, simulate_model

Examples

```
library(parameters)

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

`check_clusterstructure` *Check suitability of data for clustering*

Description

This checks whether the data is appropriate for clustering using the Hopkins' H statistic of given data. If the value of Hopkins statistic is close to 0 (below 0.5), then we can reject the null hypothesis and conclude that the dataset is significantly clusterable. A value for H lower than 0.25 indicates a clustering tendency at the 90% confidence level. The visual assessment of cluster tendency (VAT) approach (Bezdek and Hathaway, 2002) consists in investigating the heatmap of the ordered dissimilarity matrix. Following this, one can potentially detect the clustering tendency by counting the number of square shaped blocks along the diagonal.

Usage

```
check_clusterstructure(x, standardize = TRUE, distance = "euclidean", ...)
```

Arguments

`x`	A data frame.
`standardize`	Standardize the dataframe before clustering (default).
`distance`	Distance method used. Other methods than "euclidean" (default) are exploratory in the context of clustering tendency. See `dist()` for list of available methods.
`...`	Arguments passed to or from other methods.

Value

The H statistic (numeric)

References

- Lawson, R. G., & Jurs, P. C. (1990). New index for clustering tendency and its application to chemical problems. Journal of chemical information and computer sciences, 30(1), 36-41.
- Bezdek, J. C., & Hathaway, R. J. (2002, May). VAT: A tool for visual assessment of (cluster) tendency. In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN02 (3), 2225-2230. IEEE.

See Also

`check_kmo`, `check_sphericity` and `check_factorstructure`.

Examples

```
library(parameters)
check_clusterstructure(iris[, 1:4])
plot(check_clusterstructure(iris[, 1:4]))
```

`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, ...)
```

Arguments

x	A dataframe.
...	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` and `check_clusterstructure`.

Examples

```
library(parameters)
check_factorstructure(mtcars)
```

check_heterogeneity *Compute group-meaned and de-meaned variables*

Description

demean() computes group- and de-meaned versions of a variable that can be used in regression analysis to model the between- and within-subject effect. check_heterogeneity() checks if model predictors or variables may cause a heterogeneity bias, i.e. if variables have a within- and/or between-effect.

Usage

```
check_heterogeneity(x, select = NULL, group = NULL)

demean(
  x,
  select,
  group,
  suffix_demean = "_within",
  suffix_groupmean = "_between",
  add_attributes = TRUE,
  verbose = TRUE
)
```

Arguments

x	A data frame. For check_heterogeneity(), may also be a mixed model object.
select	Character vector (or formula) with names of variables to select that should be group- and de-meaned. For check_heterogeneity(), if x is a mixed model object, this argument be ignored.
group	Character vector (or formula) with the name of the variable that indicates the group- or cluster-ID. For check_heterogeneity(), if x is a model object, this argument be ignored.
suffix_demean, suffix_groupmean	String value, will be appended to the names of the group-meaned and de-meaned variables of x. By default, de-meaned variables will be suffixed with "_within" and grouped-meaned variables with "_between".
add_attributes	Logical, if TRUE, the returned variables gain attributes to indicate the within- and between-effects. This is only relevant when printing model_parameters() - in such cases, the within- and between-effects are printed in separated blocks.
verbose	Toggle off warnings.

Details

Heterogeneity Bias: Mixed models include different levels of sources of variability, i.e. error terms at each level. When macro-indicators (or level-2 predictors, or higher-level units, or more general: *group-level predictors that* **vary** *within and across groups*) are included as fixed effects (i.e. treated as covariate at level-1), the variance that is left unaccounted for this covariate will be absorbed into the error terms of level-1 and level-2 (*Bafumi and Gelman 2006; Gelman and Hill 2007, Chapter 12.6.*): "Such covariates contain two parts: one that is specific to the higher-level entity that does not vary between occasions, and one that represents the difference between occasions, within higher-level entities" (*Bell et al. 2015*). Hence, the error terms will be correlated with the covariate, which violates one of the assumptions of mixed models (iid, independent and identically distributed error terms). This bias is also called the *heterogeneity bias* (*Bell et al. 2015*). To resolve this problem, level-2 predictors used as (level-1) covariates should be separated into their "within" and "between" effects by "de-meaning" and "group-meaning": After demeaning time-varying predictors, "at the higher level, the mean term is no longer constrained by Level 1 effects, so it is free to account for all the higher-level variance associated with that variable" (*Bell et al. 2015*).

Panel data and correlating fixed and group effects: `demean()` is intended to create group- and de-meaned variables for panel regression models (fixed effects models), or for complex random-effect-within-between models (see *Bell et al. 2015, 2018*), where group-effects (random effects) and fixed effects correlate (see *Bafumi and Gelman 2006*). This can happen, for instance, when analyzing panel data, which can lead to *Heterogeneity Bias*. To control for correlating predictors and group effects, it is recommended to include the group-meaned and de-meaned version of *time-varying covariates* (and group-meaned version of *time-invariant covariates* that are on a higher level, e.g. level-2 predictors) in the model. By this, one can fit complex multilevel models for panel data, including time-varying predictors, time-invariant predictors and random effects.

Why mixed models are preferred over fixed effects models: A mixed models approach can model the causes of endogeneity explicitly by including the (separated) within- and between-effects of time-varying fixed effects and including time-constant fixed effects. Furthermore, mixed models also include random effects, thus a mixed models approach is superior to classic fixed-effects models, which lack information of variation in the group-effects or between-subject effects. Furthermore, fixed effects regression cannot include random slopes, which means that fixed effects regressions are neglecting "cross-cluster differences in the effects of lower-level controls (which) reduces the precision of estimated context effects, resulting in unnecessarily wide confidence intervals and low statistical power" (*Heisig et al. 2017*).

Terminology: The group-meaned variable is simply the mean of an independent variable within each group (or id-level or cluster) represented by `group`. It represents the cluster-mean of an independent variable. The de-meaned variable is then the centered version of the group-meaned variable. De-meaning is sometimes also called person-mean centering or centering within clusters.

De-meaning with continuous predictors: For continuous time-varying predictors, the recommendation is to include both their de-meaned and group-meaned versions as fixed effects, but not the raw (untransformed) time-varying predictors themselves. The de-meaned predictor should also be included as random effect (random slope). In regression models, the coefficient of the de-meaned predictors indicates the within-subject effect, while the coefficient of the group-meaned predictor indicates the between-subject effect.

De-meaning with binary predictors: For binary time-varying predictors, the recommendation is to include the raw (untransformed) binary predictor as fixed effect only and the *de-meaned* variable as random effect (random slope) (*Hoffmann 2015, chapter 8-2.I*). `demean()` will thus coerce categorical time-varying predictors to numeric to compute the de- and group-meaned versions for these variables.

De-meaning of factors with more than 2 levels: Factors with more than two levels are de-meaned in two ways: first, these are also converted to numeric and de-meaned; second, dummy variables are created (binary, with 0/1 coding for each level) and these binary dummy-variables are de-meaned in the same way (as described above). Packages like **panelr** internally convert factors to dummies before demeaning, so this behaviour can be mimicked here.

De-meaning interaction terms: There are multiple ways to deal with interaction terms of within- and between-effects. A classical approach is to simply use the product term of the de-meaned variables (i.e. introducing the de-meaned variables as interaction term in the model formula, e.g. `y ~ x_within * time_within`). This approach, however, might be subject to bias (see *Giesselmann & Schmidt-Catran 2020*).

Another option is to first calculate the product term and then apply the de-meaning to it. This approach produces an estimator "that reflects unit-level differences of interacted variables whose moderators vary within units", which is desirable if *no* within interaction of two time-dependent variables is required.

A third option, when the interaction should result in a genuine within estimator, is to "double de-mean" the interaction terms (*Giesselmann & Schmidt-Catran 2018*), however, this is currently not supported by `demean()`. If this is required, the `wmb()` function from the **panelr** package should be used.

To de-mean interaction terms for within-between models, simply specify the term as interaction for the `select`-argument, e.g. `select = "a*b"` (see 'Examples').

Analysing panel data with mixed models using lme4: A description of how to translate the formulas described in *Bell et al. 2018* into R using `lmer()` from **lme4** can be found in this vignette.

Value

A data frame with the group-/de-meaned variables, which get the suffix `"_between"` (for the group-meaned variable) and `"_within"` (for the de-meaned variable) by default.

References

- Bafumi J, Gelman A. 2006. Fitting Multilevel Models When Predictors and Group Effects Correlate. In. Philadelphia, PA: Annual meeting of the American Political Science Association.
- Bell A, Fairbrother M, Jones K. 2019. Fixed and Random Effects Models: Making an Informed Choice. Quality & Quantity (53); 1051-1074
- Bell A, Jones K. 2015. Explaining Fixed Effects: Random Effects Modeling of Time-Series Cross-Sectional and Panel Data. Political Science Research and Methods, 3(1), 133–153.

- Gelman A, Hill J. 2007. Data Analysis Using Regression and Multilevel/Hierarchical Models. Analytical Methods for Social Research. Cambridge, New York: Cambridge University Press
- Giesselmann M, Schmidt-Catran, AW. 2020. Interactions in fixed effects regression models. Sociological Methods & Research, 1–28. https://doi.org/10.1177/0049124120914934
- Heisig JP, Schaeffer M, Giesecke J. 2017. The Costs of Simplicity: Why Multilevel Models May Benefit from Accounting for Cross-Cluster Differences in the Effects of Controls. American Sociological Review 82 (4): 796–827.
- Hoffman L. 2015. Longitudinal analysis: modeling within-person fluctuation and change. New York: Routledge

Examples

```
data(iris)
iris$ID <- sample(1:4, nrow(iris), replace = TRUE) # fake-ID
iris$binary <- as.factor(rbinom(150, 1, .35)) # binary variable

x <- demean(iris, select = c("Sepal.Length", "Petal.Length"), group = "ID")
head(x)

x <- demean(iris, select = c("Sepal.Length", "binary", "Species"), group = "ID")
head(x)

check_heterogeneity(iris, select = c("Sepal.Length", "Petal.Length"), group = "ID")

# demean interaction term x*y
dat <- data.frame(
  a = c(1, 2, 3, 4, 1, 2, 3, 4),
  x = c(4, 3, 3, 4, 1, 2, 1, 2),
  y = c(1, 2, 1, 2, 4, 3, 2, 1),
  ID = c(1, 2, 3, 1, 2, 3, 1, 2)
)
demean(dat, select = c("a", "x*y"), group = "ID")

# or in formula-notation
demean(dat, select = ~a + x * y, group = ~ID)
```

---

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, ...)
```

Arguments

x	A dataframe.
...	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_multimodal` *Check if a distribution is unimodal or multimodal*

Description

For univariate distributions (one-dimensional vectors), this functions performs a Ameijeiras-Alonso et al. (2018) excess mass test. For multivariate distributions (dataframes), it uses mixture modelling. However, it seems that it always returns a significant result (suggesting that the distribution is multimodal). A better method might be needed here.

Usage

```
check_multimodal(x, ...)
```

Arguments

x	A numeric vector or a data frame.
...	Arguments passed to or from other methods.

References

- Ameijeiras-Alonso, J., Crujeiras, R. M., \& Rodríguez-Casal, A. (2019). Mode testing, critical bandwidth and excess mass. Test, 28(3), 900-919.

Examples

```
if (require("multimode")) {
  # Univariate
  x <- rnorm(1000)
  check_multimodal(x)
}

if (require("multimode") && require("mclust")) {
  x <- c(rnorm(1000), rnorm(1000, 2))
  check_multimodal(x)

  # Multivariate
  m <- data.frame(
    x = rnorm(200),
    y = rbeta(200, 2, 1)
  )
  plot(m$x, m$y)
  check_multimodal(m)

  m <- data.frame(
    x = c(rnorm(100), rnorm(100, 4)),
    y = c(rbeta(100, 2, 1), rbeta(100, 1, 4))
  )
  plot(m$x, m$y)
  check_multimodal(m)
}
```

---

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, ...)
```

Arguments

x	A dataframe.
...	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 Intervals (CI)*

Description

Compute confidence intervals (CI) for frequentist models.

Usage

```
## S3 method for class 'merMod'
ci(
  x,
  ci = 0.95,
  method = c("wald", "ml1", "betwithin", "satterthwaite", "kenward", "boot"),
  ...
)

## Default S3 method:
ci(x, ci = 0.95, method = NULL, ...)

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

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

## S3 method for class 'mixor'
ci(x, ci = 0.95, effects = c("all", "fixed", "random"), ...)

## S3 method for class 'DirichletRegModel'
ci(x, ci = 0.95, component = c("all", "conditional", "precision"), ...)

## S3 method for class 'glmmTMB'
ci(
  x,
  ci = 0.95,
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  method = c("wald", "ml1", "betwithin", "robust"),
  ...
)

## S3 method for class 'zeroinfl'
ci(
  x,
  ci = 0.95,
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  method = c("wald", "ml1", "betwithin", "robust"),
  ...
)

## S3 method for class 'hurdle'
ci(
  x,
  ci = 0.95,
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  method = c("wald", "ml1", "betwithin", "robust"),
```

```
  ...
)

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

## S3 method for class 'poissonmfx'
ci(
  x,
  ci = 0.95,
  component = c("all", "conditional", "marginal"),
  method = NULL,
  ...
)

## S3 method for class 'betamfx'
ci(
  x,
  ci = 0.95,
  component = c("all", "conditional", "precision", "marginal"),
  method = NULL,
  ...
)

## S3 method for class 'betareg'
ci(x, ci = 0.95, component = c("all", "conditional", "precision"), ...)

## S3 method for class 'clm2'
ci(x, ci = 0.95, component = c("all", "conditional", "scale"), ...)

## S3 method for class 'lme'
ci(x, ci = 0.95, method = c("wald", "betwithin", "ml1", "satterthwaite"), ...)
```

Arguments

x	A statistical model.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
method	For mixed models, can be "wald" (default), "ml1" or "betwithin". For linear mixed model, can also be "satterthwaite", "kenward" or "boot" and lme4::confint.merMod). For (generalized) linear models, can be "robust" to compute confidence intervals based on robust standard errors, and for generalized linear models, may also be "profile" (default) or "wald".

...	Arguments passed down to `standard_error_robust()` when confidence intervals or p-values based on robust standard errors should be computed.
effects	Should standard errors for fixed effects or random effects be returned? Only applies to mixed models. May be abbreviated. When standard errors for random effects are requested, for each grouping factor a list of standard errors (per group level) for random intercepts and slopes is returned.
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.

Note

`ci_robust()` resp. `ci(method = "robust")` rely on the **sandwich** or **clubSandwich** package (the latter if `vcov_estimation = "CR"` for cluster-robust standard errors) and will thus only work for those models supported by those packages.

Examples

```
library(parameters)
if (require("glmmTMB")) {
  model <- glmmTMB(
    count ~ spp + mined + (1 | site),
    ziformula = ~mined,
    family = poisson(),
    data = Salamanders
  )

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

---

ci_betwithin *Between-within approximation for SEs, CIs and p-values*

Description

Approximation of degrees of freedom based on a "between-within" heuristic.

Usage

```
ci_betwithin(model, ci = 0.95)

dof_betwithin(model)

p_value_betwithin(model, dof = NULL)

se_betwithin(model)
```

Arguments

model	A mixed model.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
dof	Degrees of Freedom.

Details

Small Sample Cluster corrected Degrees of Freedom: Inferential statistics (like p-values, confidence intervals and standard errors) may be biased in mixed models when the number of clusters is small (even if the sample size of level-1 units is high). In such cases it is recommended to approximate a more accurate number of degrees of freedom for such inferential statistics (see *Li and Redden 2015*). The *Between-within* denominator degrees of freedom approximation is recommended in particular for (generalized) linear mixed models with repeated measurements (longitudinal design). `dof_betwithin`) implements a heuristic based on the between-within approach. **Note** that this implementation does not return exactly the same results as shown in *Li and Redden 2015*, but similar.

Degrees of Freedom for Longitudinal Designs (Repeated Measures): In particular for repeated measure designs (longitudinal data analysis), the *between-within* heuristic is likely to be more accurate than simply using the residual or infinite degrees of freedom, because `dof_betwithin()` returns different degrees of freedom for within-cluster and between-cluster effects.

Value

A data frame.

References

- Elff, M.; Heisig, J.P.; Schaeffer, M.; Shikano, S. (2019). Multilevel Analysis with Few Clusters: Improving Likelihood-based Methods to Provide Unbiased Estimates and Accurate Inference, British Journal of Political Science.
- Li, P., Redden, D. T. (2015). Comparing denominator degrees of freedom approximations for the generalized linear mixed model in analyzing binary outcome in small sample cluster-randomized trials. BMC Medical Research Methodology, 15(1), 38. doi: 10.1186/s12874015-0026x

See Also

dof_betwithin() and se_betwithin() are small helper-functions to calculate approximated degrees of freedom and standard errors of model parameters, based on the "between-within" heuristic.

Examples

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

---

ci_kenward *Kenward-Roger approximation for SEs, CIs and p-values*

Description

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

Usage

```
ci_kenward(model, ci = 0.95)

dof_kenward(model)

p_value_kenward(model, dof = NULL)

se_kenward(model)
```

Arguments

model	A statistical model.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
dof	Degrees of Freedom.

Details

Inferential statistics (like p-values, confidence intervals and standard errors) may be biased in mixed models when the number of clusters is small (even if the sample size of level-1 units is high). In such cases it is recommended to approximate a more accurate number of degrees of freedom for such inferential statistics. Unlike simpler approximation heuristics like the "m-l-1" rule (dof_ml1), the Kenward-Roger approximation is also applicable in more complex multilevel designs, e.g. with cross-classified clusters. However, the "m-l-1" heuristic also applies to generalized mixed models, while approaches like Kenward-Roger or Satterthwaite are limited to linear mixed models only.

Value

A data frame.

References

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

See Also

`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.

`dof_satterthwaite()` and `dof_ml1()` approximate degrees of freedom based on Satterthwaite's method or the "m-l-1" rule.

Examples

```
if (require("lme4")) {
  model <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
  p_value_kenward(model)
}
```

---

ci_ml1 *"m-l-1" approximation for SEs, CIs and p-values*

Description

Approximation of degrees of freedom based on a "m-l-1" heuristic as suggested by Elff et al. (2019).

Usage

```
ci_ml1(model, ci = 0.95)

dof_ml1(model)

p_value_ml1(model, dof = NULL)

se_ml1(model)
```

Arguments

`model`	A mixed model.
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`dof`	Degrees of Freedom.

Details

Small Sample Cluster corrected Degrees of Freedom: Inferential statistics (like p-values, confidence intervals and standard errors) may be biased in mixed models when the number of clusters is small (even if the sample size of level-1 units is high). In such cases it is recommended to approximate a more accurate number of degrees of freedom for such inferential statitics (see *Li and Redden 2015*). The *m-l-1* heuristic is such an approach that uses a t-distribution with fewer degrees of freedom (`dof_ml1`) to calculate p-values (`p_value_ml1`), standard errors (`se_ml1`) and confidence intervals (`ci(method = "ml1")`).

Degrees of Freedom for Longitudinal Designs (Repeated Measures): In particular for repeated measure designs (longitudinal data analysis), the *m-l-1* heuristic is likely to be more accurate than simply using the residual or infinite degrees of freedom, because `dof_ml1()` returns different degrees of freedom for within-cluster and between-cluster effects.

Limitations of the "m-l-1" Heuristic: Note that the "m-l-1" heuristic is not applicable (or at least less accurate) for complex multilevel designs, e.g. with cross-classified clusters. In such cases, more accurate approaches like the Kenward-Roger approximation (`dof_kenward()`) is recommended. However, the "m-l-1" heuristic also applies to generalized mixed models, while approaches like Kenward-Roger or Satterthwaite are limited to linear mixed models only.

Value

A data frame.

References

- Elff, M.; Heisig, J.P.; Schaeffer, M.; Shikano, S. (2019). Multilevel Analysis with Few Clusters: Improving Likelihood-based Methods to Provide Unbiased Estimates and Accurate Inference, British Journal of Political Science.
- Li, P., Redden, D. T. (2015). Comparing denominator degrees of freedom approximations for the generalized linear mixed model in analyzing binary outcome in small sample cluster-randomized trials. BMC Medical Research Methodology, 15(1), 38. doi: 10.1186/s12874015-0026x

See Also

`dof_ml1()` and `se_ml1()` are small helper-functions to calculate approximated degrees of freedom and standard errors of model parameters, based on the "m-l-1" heuristic.

Examples

```
if (require("lme4")) {
  model <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
  p_value_ml1(model)
}
```

`ci_satterthwaite` *Satterthwaite approximation for SEs, CIs and p-values*

Description

An approximate F-test based on the Satterthwaite (1946) approach.

Usage

```
ci_satterthwaite(model, ci = 0.95)

dof_satterthwaite(model)

p_value_satterthwaite(model, dof = NULL)

se_satterthwaite(model)
```

Arguments

`model`	A statistical model.
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`dof`	Degrees of Freedom.

Details

Inferential statistics (like p-values, confidence intervals and standard errors) may be biased in mixed models when the number of clusters is small (even if the sample size of level-1 units is high). In such cases it is recommended to approximate a more accurate number of degrees of freedom for such inferential statitics. Unlike simpler approximation heuristics like the "m-l-1" rule (`dof_ml1`), the Satterthwaite approximation is also applicable in more complex multilevel designs. However, the "m-l-1" heuristic also applies to generalized mixed models, while approaches like Kenward-Roger or Satterthwaite are limited to linear mixed models only.

Value

A data frame.

References

Satterthwaite FE (1946) An approximate distribution of estimates of variance components. Biometrics Bulletin 2 (6):110–4.

See Also

dof_satterthwaite() and se_satterthwaite() are small helper-functions to calculate approximated degrees of freedom and standard errors for model parameters, based on the Satterthwaite (1946) approach.

dof_kenward() and dof_ml1() approximate degrees of freedom based on Kenward-Roger's method or the "m-l-1" rule.

Examples

```
if (require("lme4")) {
  model <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
  p_value_satterthwaite(model)
}
```

---

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,
  effects = c("fixed", "random", "all"),
 component = c("all", "conditional", "zi", "zero_inflated", "dispersion", "precision",
    "scale", "smooth_terms", "full", "marginal"),
  robust = FALSE,
  ...
)

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.
effects	Should standard errors for fixed effects or random effects be returned? Only applies to mixed models. May be abbreviated. When standard errors for random effects are requested, for each grouping factor a list of standard errors (per group level) for random intercepts and slopes is returned.
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.
robust	Logical, if TRUE, robust standard errors are calculated (if possible), and confidence intervals and p-values are based on these robust standard errors. Additional arguments like vcov_estimation or vcov_type are passed down to other methods, see standard_error_robust() for details.
...	Arguments passed down to standard_error_robust() when confidence intervals or p-values based on robust standard errors should be computed.

Value

A data frame.

References

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

Examples

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

`cluster_analysis` *Compute cluster analysis and return group indices*

Description

Compute hierarchical or kmeans cluster analysis and return the group assignment for each observation as vector.

Usage

```
cluster_analysis(
  x,
  n_clusters = NULL,
  method = c("hclust", "kmeans"),
 distance = c("euclidean", "maximum", "manhattan", "canberra", "binary", "minkowski"),
 agglomeration = c("ward", "ward.D", "ward.D2", "single", "complete", "average",
    "mcquitty", "median", "centroid"),
  iterations = 20,
  algorithm = c("Hartigan-Wong", "Lloyd", "MacQueen"),
  force = TRUE,
  package = c("NbClust", "mclust"),
  verbose = TRUE
)
```

Arguments

`x`	A data frame.
`n_clusters`	Number of clusters used for the cluster solution. By default, the number of clusters to extract is determined by calling `n_clusters`.
`method`	Method for computing the cluster analysis. By default (`"hclust"`), a hierarchical cluster analysis, will be computed. Use `"kmeans"` to compute a kmeans cluster analysis. You can specify the initial letters only.
`distance`	Distance measure to be used when `method = "hclust"` (for hierarchical clustering). Must be one of `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"` or `"minkowski"`. See `dist`. If is `method = "kmeans"` this argument will be ignored.
`agglomeration`	Agglomeration method to be used when `method = "hclust"` (for hierarchical clustering). This should be one of `"ward"`, `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"median"` or `"centroid"`. Default is `"ward"` (see `hclust`). If `method = "kmeans"` this argument will be ignored.
`iterations`	Maximum number of iterations allowed. Only applies, if `method = "kmeans"`. See `kmeans` for details on this argument.
`algorithm`	Algorithm used for calculating kmeans cluster. Only applies, if `method = "kmeans"`. May be one of `"Hartigan-Wong"` (default), `"Lloyd"` (used by SPSS), or `"MacQueen"`. See `kmeans` for details on this argument.

force	Logical, if TRUE, ordered factors (ordinal variables) are converted to numeric values, while character vectors and factors are converted to dummy-variables (numeric 0/1) and are included in the cluster analysis. If FALSE, factors and character vectors are removed before computing the cluster analysis.
package	These are the packages from which methods are used to determine the number of clusters. Can be "all" or a vector containing "NbClust", "mclust", "cluster" and "M3C".
verbose	Toggle off warnings.

Details

The print() and plot() methods show the (standardized) mean value for each variable within each cluster. Thus, a higher absolute value indicates that a certain variable characteristic is more pronounced within that specific cluster (as compared to other cluster groups with lower absolute mean values).

Value

The group classification for each observation as vector. The returned vector includes missing values, so it has the same length as nrow(x).

Note

There is also a plot()-method implemented in the **see**-package.

References

Maechler M, Rousseeuw P, Struyf A, Hubert M, Hornik K (2014) cluster: Cluster Analysis Basics and Extensions. R package.

See Also

n_clusters to determine the number of clusters to extract, cluster_discrimination to determine the accuracy of cluster group classification and check_clusterstructure to check suitability of data for clustering.

Examples

```
# Hierarchical clustering of mtcars-dataset
groups <- cluster_analysis(iris[, 1:4], 3)
groups

# K-means clustering of mtcars-dataset, auto-detection of cluster-groups
## Not run:
groups <- cluster_analysis(iris[, 1:4], method = "k")
groups

## End(Not run)
```

cluster_discrimination

Compute a linear discriminant analysis on classified cluster groups

Description

Computes linear discriminant analysis on classified cluster groups, and determines the goodness of classification for each cluster group.

Usage

```
cluster_discrimination(x, cluster_groups = NULL)
```

Arguments

x	A data frame
cluster_groups	Group classification of the cluster analysis, which can be retrieved from the cluster_analysis function.

See Also

n_clusters to determine the number of clusters to extract, cluster_analysis to compute a cluster analysis and check_clusterstructure to check suitability of data for clustering.

Examples

```
## Not run:
# retrieve group classification from hierarchical cluster analysis
groups <- cluster_analysis(iris[, 1:4])

# goodness of group classificatoin
cluster_discrimination(iris[, 1:4], cluster_groups = groups)

## End(Not run)
```

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

A data frame of numeric variables.

Examples

```
head(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(parameters)
if (require("psych") && require("lavaan")) {
  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)

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

`degrees_of_freedom` *Degrees of Freedom (DoF)*

Description

Estimate or extract degrees of freedom of models parameters.

Usage

```
degrees_of_freedom(model, ...)

## Default S3 method:
degrees_of_freedom(model, method = "analytical", ...)

dof(model, ...)
```

Arguments

`model`	A statistical model.
`...`	Currently not used.
`method`	Can be `"analytical"` (default, DoFs are estimated based on the model type), `"fit"`, in which case they are directly taken from the model if available (for Bayesian models, the goal (looking for help to make it happen) would be to refit the model as a frequentist one before extracting the DoFs), `"ml1"` (see `dof_ml1`), `"betwithin"` (see `dof_betwithin`), `"satterthwaite"` (see `dof_satterthwaite`), `"kenward"` (see `dof_kenward`) or `"any"`, which tries to extract DoF by any of those methods, whichever succeeds.

Details

Methods for calculating degrees of freedom:

- `"analytical"` for models of class `lmerMod`, Kenward-Roger approximated degrees of freedoms are calculated, for other models, `n-k` (number of observations minus number of parameters).
- `"fit"` tries to extract residual degrees of freedom, and returns `Inf` if residual degrees of freedom could not be extracted.
- `"any"` first tries to extract residual degrees of freedom, and if these are not available, extracts analytical degrees of freedom.

- "nokr" same as "analytical", but does not Kenward-Roger approximation for models of class lmerMod. Instead, always uses n-k to calculate df for any model.
- "wald" returns Inf.
- "kenward" calls dof_kenward.
- "satterthwaite" calls dof_satterthwaite.
- "ml1" calls dof_ml1.
- "betwithin" calls dof_betwithin.

For models with z-statistic, the returned degrees of freedom for model parameters is Inf (unless method = "ml1" or method = "betwithin"), because there is only one distribution for the related test statistic.

Examples

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

model <- glm(vs ~ mpg * cyl, data = mtcars, family = "binomial")
dof(model)


if (require("lme4")) {
  model <- lmer(Sepal.Length ~ Petal.Length + (1 | Species), data = iris)
  dof(model)
}

if (require("rstanarm")) {
  model <- stan_glm(
    Sepal.Length ~ Petal.Length * Species,
    data = iris,
    chains = 2,
    refresh = 0
  )
  dof(model)
}
```

---

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, ...)

## S3 method for class 'numeric'
describe_distribution(
  x,
  centrality = "mean",
  dispersion = TRUE,
  iqr = TRUE,
  range = TRUE,
  ci = NULL,
  iterations = 100,
  ...
)

## S3 method for class 'factor'
describe_distribution(x, dispersion = TRUE, range = TRUE, ...)

## S3 method for class 'data.frame'
describe_distribution(
  x,
  centrality = "mean",
  dispersion = TRUE,
  iqr = TRUE,
  range = TRUE,
  include_factors = FALSE,
  ci = NULL,
  iterations = 100,
  ...
)
```

Arguments

x	A numeric vector.
...	Additional arguments to be passed to or from methods.
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).
iqr	Logical, if TRUE, the interquartile range is calculated (based on IQR, using type = 6).
range	Return the range (min and max).
ci	Confidence Interval (CI) level. Default is NULL, i.e. no confidence intervals are computed. If not NULL, confidence intervals are based on bootstrap replicates (see iterations).
iterations	The number of bootstrap replicates for computing confidence intervals. Only applies when ci is not NULL.

include_factors
: Logical, if TRUE, factors are included in the output, however, only columns for range (first and last factor levels) as well as n and missing will contain information.

Value

A data frame with columns that describe the properties of the variables.

Note

There is also a plot()-method implemented in the **see**-package.

Examples

```
describe_distribution(rnorm(100))

data(iris)
describe_distribution(iris)
describe_distribution(iris, include_factors = TRUE)
```

---

equivalence_test.lm *Equivalence test*

Description

Compute the (conditional) equivalence test for frequentist models.

Usage

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

## S3 method for class 'merMod'
equivalence_test(
  x,
  range = "default",
  ci = 0.95,
  rule = "bayes",
  effects = c("fixed", "random"),
```

```
  p_values = FALSE,
  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%).
rule	Character, indicating the rules when testing for practical equivalence. Can be "bayes", "classic" or "cet". See 'Details'.
p_values	Logical, if TRUE, adjusted p-values for equivalence testing are calculated.
verbose	Toggle off warnings.
...	Arguments passed to or from other methods.
effects	Should parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.

Details

In classical null hypothesis significance testing (NHST) within a frequentist framework, it is not possible to accept the null hypothesis, H0 - unlike in Bayesian statistics, where such probability statements are possible. "[...] one can only reject the null hypothesis if the test statistics falls into the critical region(s), or fail to reject this hypothesis. In the latter case, all we can say is that no significant effect was observed, but one cannot conclude that the null hypothesis is true." (*Pernet 2017*). One way to address this issues without Bayesian methods is *Equivalence Testing*, as implemented in `equivalence_test()`. While you either can reject the null hypothesis or claim an inconclusive result in NHST, the equivalence test adds a third category, *"accept"*. Roughly speaking, the idea behind equivalence testing in a frequentist framework is to check whether an estimate and its uncertainty (i.e. confidence interval) falls within a region of "practical equivalence". Depending on the rule for this test (see below), statistical significance does not necessarily indicate whether the null hypothesis can be rejected or not, i.e. the classical interpretation of the p-value may differ from the results returned from the equivalence test.

Calculation of equivalence testing:

"bayes" - Bayesian rule (Kruschke 2018) This rule follows the "HDI+ROPE decision rule" *(Kruschke, 2014, 2018)* used for the `Bayesian counterpart`. This means, if the confidence intervals are completely outside the ROPE, the "null hypothesis" for this parameter is "rejected". If the ROPE completely covers the CI, the null hypothesis is accepted. Else, it's undecided whether to accept or reject the null hypothesis. Desirable results are low proportions inside the ROPE (the closer to zero the better).

"classic" - The TOST rule (Lakens 2017) This rule follows the "TOST rule", i.e. a two one-sided test procedure (*Lakens 2017*). Following this rule, practical equivalence of an effect (i.e. H0) is *rejected*, when the coefficient is statistically significant *and* the narrow confidence intervals (i.e. `1-2*alpha`) *include* or *exceed* the ROPE boundaries. Practical equivalence is assumed (i.e. H0 accepted) when the narrow confidence intervals are completely inside the

ROPE, no matter if the effect is statistically significant or not. Else, the decision whether to accept or reject H0 is undecided.

"cet" - Conditional Equivalence Testing (Campbell/Gustafson 2018) The Conditional Equivalence Testing as described by *Campbell and Gustafson 2018*. According to this rule, practical equivalence is rejected when the coefficient is statistically significant. When the effect is *not* significant and the narrow confidence intervals are completely inside the ROPE, we accept H0, else it is undecided.

Levels of Confidence Intervals used for Equivalence Testing: For `rule = "classic"`, "narrow" confidence intervals are used for equivalence testing. "Narrow" means, the the intervals is not 1 - alpha, but 1 - 2 * alpha. Thus, if `ci = .95`, alpha is assumed to be 0.05 and internally a ci-level of 0.90 is used. `rule = "cet"` uses both regular and narrow confidence intervals, while `rule = "bayes"` only uses the regular intervals.

Second Generation p-Value (SGPV): Second generation p-values (SGPV) were proposed as a statistic that represents "the proportion of data-supported hypotheses that are also null hypotheses" *(Blume et al. 2018)*. This statistic is actually computed in the same way as the percentage inside the ROPE as returned by `equivalence_test()` (see *Lakens and Delacre 2020* for details on computation of the SGPV). Thus, the `"inside ROPE"` column reflects the SGPV.

Adjustment for multiple testing: The calculation of p-values is somewhat "experimental". For parameters, where H0...

- ... is rejected, the p-value equals a NHST as if the upper / lower boundary of the ROPE (see `range`) would be the point-null to test against.
- ... is accepted, the p-value is set to 1.
- ... is undecided, the p-value equals a NHST against the point-null, however, the "uncertainty" (i.e. ROPE range) is added to the confidence intervals (so the upper confidence interval limit equals the regular upper confidence interval limit + half the ROPE range).

All p-values are then adjusted for multiple testing (using `p.adjust` with `method = "fdr"`).

ROPE range: Some attention is required for finding suitable values for the ROPE limits (argument `range`). See 'Details' in `rope_range` for further information.

Value

A data frame.

Note

There is also a `plot()`-method implemented in the **see**-package.

References

- Blume, J. D., D'Agostino McGowan, L., Dupont, W. D., & Greevy, R. A. (2018). Second-generation p-values: Improved rigor, reproducibility, & transparency in statistical analyses. PLOS ONE, 13(3), e0188299. https://doi.org/10.1371/journal.pone.0188299
- Campbell, H., & Gustafson, P. (2018). Conditional equivalence testing: An alternative remedy for publication bias. PLOS ONE, 13(4), e0195145. doi: 10.1371/journal.pone.0195145

- Kruschke, J. K. (2014). Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan. Academic Press
- Kruschke, J. K. (2018). Rejecting or accepting parameter values in Bayesian estimation. Advances in Methods and Practices in Psychological Science, 1(2), 270-280. doi: 10.1177/2515245918771304
- Lakens, D. (2017). Equivalence Tests: A Practical Primer for t Tests, Correlations, and Meta-Analyses. Social Psychological and Personality Science, 8(4), 355–362. doi: 10.1177/1948550617697177
- Lakens, D., & Delacre, M. (2020). Equivalence Testing and the Second Generation P-Value. Meta-Psychology, 4. https://doi.org/10.15626/MP.2018.933
- Pernet, C. (2017). Null hypothesis significance testing: A guide to commonly misunderstood concepts and recommendations for good practice. F1000Research, 4, 621. doi: 10.12688/f1000research.6963.5

See Also

For more details, see `equivalence_test()`. Further readings can be found in the references.

Examples

```
data(qol_cancer)
model <- lm(QoL ~ time + age + education, data = qol_cancer)

# default rule
equivalence_test(model)

# conditional equivalence test
equivalence_test(model, rule = "cet")

# plot method
if (require("see")) {
  result <- equivalence_test(model)
  plot(result)
}
```

---

factor_analysis *Factor Analysis (FA)*

Description

This function performs a Factor Analysis (FA).

Usage

```
factor_analysis(
  x,
  n = "auto",
  rotation = "none",
  sort = FALSE,
  threshold = NULL,
```

```
  standardize = TRUE,
  cor = NULL,
  ...
)
```

Arguments

x	A data frame 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 reduce_parameters, 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 / FA 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 (centered and scaled) to have unit variance before the analysis takes place (in general, such scaling is advisable).
cor	An optional correlation matrix that can be used (note that the data must still be passed as the first argument). If NULL, will compute it by running cor() on the passed data.
...	Arguments passed to or from other methods.

Details

Complexity: Complexity 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 (*Hofman, 1978; Pettersson and Turkheimer, 2010*) .

FA or PCA?: There is a simplified rule of thumb that may help do decide whether to run a principal component analysis or a factor analysis:

- Run *factor analysis* if you assume or wish to test a theoretical model of latent factors causing observed variables.
- Run *principal component analysis* If you want to simply reduce your correlated observed variables to a smaller set of important independent composite variables.

(Source: CrossValidated)

Value

A data frame of loadings.

Note

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

References

- Hofmann, R. (1978). Complexity and simplicity as objective indices descriptive of factor solutions. Multivariate Behavioral Research, 13:2, 247-250, doi: 10.1207/s15327906mbr1302_9
- Pettersson, E., & Turkheimer, E. (2010). Item selection, evaluation, and simple structure in personality data. Journal of research in personality, 44(4), 407-420, doi: 10.1016/j.jrp.2010.03.002

Examples

```
library(parameters)
if (require("psych")) {
  factor_analysis(mtcars[, 1:7], n = "all", threshold = 0.2)
 factor_analysis(mtcars[, 1:7], n = 2, rotation = "oblimin", threshold = "max", sort = TRUE)
  factor_analysis(mtcars[, 1:7], n = 2, threshold = 2, sort = TRUE)

  efa <- factor_analysis(mtcars[, 1:5], n = 2)
  summary(efa)
  predict(efa)

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

}
```

`fish` *Sample data set*

Description

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

`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_parameters *Parameter names formatting*

Description

Parameter 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, data = iris)
format_parameters(model)

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

model <- lm(Sepal.Length ~ Petal.Length + (Species / 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)
```

`get_scores` *Get Scores from Principal Component Analysis (PCA)*

Description

`get_scores()` takes `n_items` amount of items that load the most (either by loading cutoff or number) on a component, and then computes their average.

Usage

```
get_scores(x, n_items = NULL)
```

Arguments

`x`	An object returned by `principal_components`.
`n_items`	Number of required (i.e. non-missing) items to build the sum score. If `NULL`, the value is chosen to match half of the number of columns in a data frame.

Details

`get_scores()` takes the results from `principal_components` and extracts the variables for each component found by the PCA. Then, for each of these "subscales", row means are calculated (which equals adding up the single items and dividing by the number of items). This results in a sum score for each component from the PCA, which is on the same scale as the original, single items that were used to compute the PCA.

Value

A data frame with subscales, which are average sum scores for all items from each component.

Examples

```
library(parameters)
pca <- principal_components(mtcars[, 1:7], n = 2, rotation = "varimax")

# PCA extracted two components
pca

# assignment of items to each component
closest_component(pca)

# now we want to have sum scores for each component
get_scores(pca)
```

```
# compare to manually computed sum score for 2nd component, which
# consists of items "hp" and "qsec"
(mtcars$hp + mtcars$qsec) / 2
```

`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`)
- Multinomial, ordinal and cumulative link models (`bracl`, `multinom`, `mlm`, ...)
- Mixed models (**lme4**, **nlme**, **glmmTMB**, ...)
- Bayesian tests (**BayesFactor**)
- Bayesian models (**rstanarm**, **brms**, **MCMCglmm**, ...)
- PCA and FA (**psych**)
- CFA and SEM (**lavaan**, **blavaan**)
- Cluster models (k-means, ...)
- Meta-Analysis via linear (mixed) models (`rma`)
- Hypothesis Testing (`glht`)
- Multiply imputed repeated analyses (`mira`)

Usage

```
model_parameters(model, ...)

parameters(model, ...)
```

Arguments

`model`	Statistical Model.
`...`	Arguments passed to or from other methods. Non-documented arguments are `digits`, `p_digits` and `ci_digits` to set the number of digits for the output. See 'Examples' in `model_parameters.default`.

Details

Standardization is based on `standardize_parameters()`. In case of `standardize = "refit"`, the data used to fit the model will be standardized and the model is completely refitted. In such cases, standard errors and confidence intervals refer to the standardized coefficient.

Value

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

Note

The `print()` method has several arguments to tweak the output. There is also a `plot()`-method implemented in the **see**-package.

See Also

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

`model_parameters.aov` *Parameters from ANOVAs*

Description

Parameters from ANOVAs.

Usage

```
## S3 method for class 'aov'
model_parameters(
  model,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  df_error = NULL,
  type = 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), `"raw"` or `"adjusted"` (the latter option only for anova-tables from mixed models).
`epsilon_squared`	Compute epsilon squared as index of effect size. Can be `"partial"` (adjusted for effect size) or `"raw"`.
`df_error`	Denominator degrees of freedom (or degrees of freedom of the error estimate, i.e., the residuals). This is used to compute effect sizes for anova tables from mixed models. See 'Examples'.

type	Numeric, type of sums of squares. May be 1, 2 or 3. If 2 or 3, anova-tables using car::Anova() will be returned.
...	Arguments passed to or from other methods.

Value

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

Note

For anova-tables from mixed models (i.e. anova(lmer())), only partial or adjusted effect sizes can be computed.

Examples

```
if (requireNamespace("effectsize", quietly = TRUE)) {
  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 = "partial"
  )

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

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

  if (require("lme4")) {
    mm <- lmer(Sepal.Length ~ Sepal.Big + Petal.Width + (1 | Species),
               data = df)
    model <- anova(mm)

    # simple parameters table
    model_parameters(model)

    # parameters table including effect sizes
    model_parameters(
      model,
      eta_squared = "partial",
      df_error = dof_satterthwaite(mm)
```

```
    )
  }
}
```

`model_parameters.befa` *Parameters from PCA/FA*

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`), `"ETI"` (see `eti`) or `"SI"` (see `si`).
`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)

if (require("BayesFM")) {
  efa <- BayesFM::befa(mtcars, iter = 1000)
  results <- model_parameters(efa, sort = TRUE)
  results
  efa_to_cfa(results)
}
```

---

model_parameters.BFBayesFactor

Parameters from BayesFactor objects

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), "ETI" (see eti) or "SI" (see si).
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.

Details

The meaning of the extracted parameters: For ttestBF: Difference is the raw difference between the means. For correlationBF: rho is the linear correlation estimate (equivalent to Pearson's *r*). lmBF / generalTestBF / For regressionBF / anovaBF: in addition to parameters of the fixed and random effects, there are: mu is the (mean-centered) intercept; sig2 is the model's sigma; g / g_* are the *g* parameters; See the *Bayes Factors for ANOVAs* paper.

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 from 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,
  standardize = NULL,
  exponentiate = FALSE,
  robust = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'rqss'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("conditional", "smooth_terms", "all"),
  standardize = NULL,
  exponentiate = FALSE,
  ...
)

## S3 method for class 'cgam'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("conditional", "smooth_terms", "all"),
  standardize = NULL,
  exponentiate = FALSE,
  ...
)
```

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 bootstrap_parameters()).
iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.

standardize	The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Note that robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".
exponentiate	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
robust	Logical, if TRUE, robust standard errors are calculated (if possible), and confidence intervals and p-values are based on these robust standard errors. Additional arguments like vcov_estimation or vcov_type are passed down to other methods, see standard_error_robust() for details.
p_adjust	Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like ci_method are passed down to describe_posterior.
component	Model component for which parameters should be shown. May be one of "conditional", "precision" (**betareg**), "scale" (**ordinal**), "extra" (**glmx**), "marginal" (**mfx**) or "all".

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)
if (require("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.glht *Parameters from Hypothesis Testing*

Description

Parameters from Hypothesis Testing.

Usage

```
## S3 method for class 'glht'
model_parameters(model, ci = 0.95, exponentiate = FALSE, ...)
```

Arguments

model	Object of class glht (**multcomp**).
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
exponentiate	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like ci_method are passed down to describe_posterior.

Value

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

Examples

```
if (require("multcomp")) {
  # multiple linear model, swiss data
  lmod <- lm(Fertility ~ ., data = swiss)
  mod <- glht(
    model = lmod,
    linfct = c(
      "Agriculture = 0",
      "Examination = 0",
      "Education = 0",
      "Catholic = 0",
      "Infant.Mortality = 0"
    )
  )
  model_parameters(mod)
}
```

---

model_parameters.htest

Parameters from Correlations and t-tests

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.

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.kmeans

Parameters from Cluster Models (k-means, ...)

Description

Format cluster models obtained for example by kmeans.

Usage

```
## S3 method for class 'kmeans'
model_parameters(model, ...)
```

Arguments

model	Cluster model.
...	Arguments passed to or from other methods.

Examples

```
library(parameters)

model <- kmeans(iris[1:4], centers = 3)
model_parameters(model)
```

`model_parameters.lavaan` *Parameters from CFA/SEM models*

Description

Format CFA/SEM objects from the (b)lavaan package (Rosseel, 2012; Merkle and Rosseel 2018).

Usage

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

Arguments

`model`	CFA or SEM created by the `lavaan::cfa` or `lavaan::sem` functions (or from **blavaan**).
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`standardize`	Return standardized parameters (standardized coefficients). Can be `TRUE` (or `"all"` or `"std.all"`) for standardized estimates based on both the variances of observed and latent variables; `"latent"` (or `"std.lv"`) for standardized estimates based on the variances of the latent variables only; or `"no_exogenous"` (or `"std.nox"`) for standardized estimates based on both the variances of observed and latent variables, but not the variances of exogenous covariates. See `lavaan::standardizedsolution` for details.
`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.

Note

There is also a `plot()`-method implemented in the **see**-package.

References

- Rosseel Y (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36.
- Merkle EC , Rosseel Y (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. http://www.jstatsoft.org/v85/i04/

Examples

```
library(parameters)

# lavaan -------------------------------------
if (require("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)
}
```

---

model_parameters.logitor

Parameters from (General) Linear Models

Description

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

Usage

```
## S3 method for class 'logitor'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = TRUE,
  robust = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'poissonmfx'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "marginal"),
  standardize = NULL,
  exponentiate = FALSE,
  robust = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'betamfx'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "precision", "marginal"),
  standardize = NULL,
  exponentiate = FALSE,
  robust = FALSE,
  p_adjust = NULL,
  ...
)

## Default S3 method:
model_parameters(
```

```
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  robust = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'glm'
model_parameters(
  model,
  ci = 0.95,
  df_method = "profile",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  robust = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'betareg'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("conditional", "precision", "all"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'clm2'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "scale"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
```

```
  ...
)

## S3 method for class 'glmx'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "extra"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  ...
)
```

Arguments

model	Model object.
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 bootstrap_parameters()).
iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
standardize	The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Note that robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".
exponentiate	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
robust	Logical, if TRUE, robust standard errors are calculated (if possible), and confidence intervals and p-values are based on these robust standard errors. Additional arguments like vcov_estimation or vcov_type are passed down to other methods, see standard_error_robust() for details.
p_adjust	Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like ci_method are passed down to describe_posterior.
component	Model component for which parameters should be shown. May be one of "conditional", "precision" (**betareg**), "scale" (**ordinal**), "extra" (**glmx**), "marginal" (**mfx**) or "all".
df_method	Method for computing degrees of freedom for confidence intervals (CI). Only applies to models of class glm or polr. May be "profile" or "wald".

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

model_parameters(model)

# bootstrapped parameters
model_parameters(model, bootstrap = TRUE)

# standardized parameters
model_parameters(model, standardize = "refit")

# different p-value style in output
model_parameters(model, p_digits = 5)
model_parameters(model, digits = 3, ci_digits = 4, p_digits = "scientific")

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

# show odds ratio / exponentiated coefficients
model_parameters(model, exponentiate = TRUE)
```

---

model_parameters.Mclust

Parameters from Mixture Models

Description

Format mixture models obtained for example by `mclust::Mclust`.

Usage

```
## S3 method for class 'Mclust'
model_parameters(model, ...)
```

Arguments

`model`	Mixture model.
`...`	Arguments passed to or from other methods.

Examples

```
library(parameters)
if (require("mclust")) {
  model <- mclust::Mclust(iris[1:4], verbose = FALSE)
  model_parameters(model)
}
```

---

model_parameters.merMod
Parameters from Mixed Models

Description

Parameters from (linear) mixed models.

Usage

```
## S3 method for class 'merMod'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  df_method = "wald",
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  robust = FALSE,
  details = FALSE,
  p_adjust = NULL,
  wb_component = TRUE,
  ...
)

## S3 method for class 'glmmTMB'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  standardize = NULL,
  exponentiate = FALSE,
  df_method = NULL,
  details = FALSE,
  wb_component = TRUE,
  ...
)
```

```
## S3 method for class 'mixor'
model_parameters(
  model,
  ci = 0.95,
  effects = c("all", "fixed", "random"),
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  details = FALSE,
  ...
)

## S3 method for class 'clmm'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  details = FALSE,
  df_method = NULL,
  ...
)
```

Arguments

model	A mixed 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 bootstrap_parameters()).
df_method	Method for computing degrees of freedom for p values, standard errors and confidence intervals (CI). May be "wald" (default, see degrees_of_freedom), "ml1" (see dof_ml1), "betwithin" (see dof_betwithin), "satterthwaite" (see dof_satterthwaite) or "kenward" (see dof_kenward). Note that when df_method is not "wald", robust standard errors etc. cannot be computed.
iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
standardize	The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Note that robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".
exponentiate	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors

	are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
robust	Logical, if TRUE, robust standard errors are calculated (if possible), and confidence intervals and p-values are based on these robust standard errors. Additional arguments like vcov_estimation or vcov_type are passed down to other methods, see standard_error_robust() for details.
details	Logical, if TRUE, a summary of the random effects is included. See random_parameters for details.
p_adjust	Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details.
wb_component	Logical, if TRUE and models contains within- and between-effects (see demean), the Component column will indicate which variables belong to the within-effects, between-effects, and cross-level interactions. By default, the Component column indicates, which parameters belong to the conditional or zero-inflated component of the model.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like ci_method are passed down to describe_posterior.
component	Model component for which parameters should be shown. May be one of "conditional", "precision" (**betareg**), "scale" (**ordinal**), "extra" (**glmx**), "marginal" (**mfx**) or "all".
effects	Should parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.

Value

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

Note

There is also a plot()-method implemented in the **see**-package.

See Also

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

Examples

```
library(parameters)
if (require("lme4")) {
  data(mtcars)
  model <- lmer(mpg ~ wt + (1 | gear), data = mtcars)
  model_parameters(model)
}

if (require("glmmTMB")) {
  data(Salamanders)
  model <- glmmTMB(
    count ~ spp + mined + (1 | site),
```

```
    ziformula = ~mined,
    family = poisson(),
    data = Salamanders
  )
  model_parameters(model, details = TRUE)

  # plot-method
  if (require("see")) {
    result <- model_parameters(model)
    plot(result)
  }
}

if (require("lme4")) {
  model <- lmer(mpg ~ wt + (1 | gear), data = mtcars)
  model_parameters(model, bootstrap = TRUE, iterations = 50)
}
```

`model_parameters.mira` *Parameters from multiply imputed repeated analyses*

Description

Format models of class `mira`, obtained from `mice::width.mids()`.

Usage

```
## S3 method for class 'mira'
model_parameters(model, ci = 0.95, exponentiate = FALSE, p_adjust = NULL, ...)
```

Arguments

`model`	An object of class `mira`.
`ci`	Confidence Interval (CI) level. Default to 0.95 (95%).
`exponentiate`	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
`p_adjust`	Character vector, if not `NULL`, indicates the method to adjust p-values. See `p.adjust` for details.
`...`	Arguments passed to or from other methods.

Details

`model_parameters()` for objects of class `mira` works similar to `mice::pool()`, i.e. it generates the pooled summary of multiple imputed repeated regression analyses.

Examples

```
library(parameters)
if (require("mice")) {
  data(nhanes2)
  imp <- mice(nhanes2)
  fit <- with(data = imp, exp = lm(bmi ~ age + hyp + chl))
  model_parameters(fit)
}

# model_parameters() also works for models that have no "tidy"-method in mice
if (require("mice") && require("gee")) {
  data(warpbreaks)
  set.seed(1234)
  warpbreaks$tension[sample(1:nrow(warpbreaks), size = 10)] <- NA
  imp <- mice(warpbreaks)
  fit <- with(data = imp, expr = gee(breaks ~ tension, id = wool))

  # does not work:
  # summary(pool(fit))

  model_parameters(fit)
}

# and it works with pooled results
if (require("mice")) {
  data("nhanes2")
  imp <- mice(nhanes2)
  fit <- with(data = imp, exp = lm(bmi ~ age + hyp + chl))
  pooled <- pool(fit)

  model_parameters(pooled)
}
```

---

model_parameters.mlm *Parameters from multinomial or cumulative link models*

Description

Parameters from multinomial or cumulative link models

Usage

```
## S3 method for class 'mlm'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
```

```
  exponentiate = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'multinom'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'bracl'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  ...
)

## S3 method for class 'DirichletRegModel'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "precision"),
  standardize = NULL,
  exponentiate = FALSE,
  ...
)
```

Arguments

model	A model with multinomial or categorical response value.
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 bootstrap_parameters()).

iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
standardize	The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Note that robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".
exponentiate	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
p_adjust	Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like ci_method are passed down to describe_posterior.
component	Model component for which parameters should be shown. May be one of "conditional", "precision" (**betareg**), "scale" (**ordinal**), "extra" (**glmx**), "marginal" (**mfx**) or "all".

Details

Multinomial or cumulative link models, i.e. models where the response value (dependent variable) is categorical and has more than two levels, usually return coefficients for each response level. Hence, the output from model_parameters() will split the coefficient tables by the different levels of the model's response.

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)
if (require("brglm2")) {
  data("stemcell")
  model <- bracl(
    research ~ as.numeric(religion) + gender,
    weights = frequency,
    data = stemcell,
    type = "ML"
  )
  model_parameters(model)
}
```

`model_parameters.PCA` *Parameters from 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, ...)

## S3 method for class 'omega'
model_parameters(model, ...)
```

Arguments

`model`	PCA or FA created by the **psych** or **FactoMineR** packages (e.g. through `psych::principal`, `psych::fa` or `psych::omega`).
`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.

- **MSA** represents the Kaiser-Meyer-Olkin Measure of Sampling Adequacy (*Kaiser and Rice, 1974*) for each item. It indicates whether there is enough data for each factor give reliable results for the PCA. The value should be > 0.6, and desirable values are > 0.8 (*Tabachnick and Fidell, 2013*).

Value

A data frame of loadings.

References

- Kaiser, H.F. and Rice. J. (1974). Little jiffy, mark iv. Educational and Psychological Measurement, 34(1):111–117
- 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.
- Tabachnick, B. G., and Fidell, L. S. (2013). Using multivariate statistics (6th ed.). Boston: Pearson Education.

Examples

```
library(parameters)
if (require("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)))


  # Omega ---------
  omega <- psych::omega(mtcars, nfactors = 3)
  params <- model_parameters(omega)
  params
  summary(params)
}

# FactoMineR ---------
## Not run:
if( require("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)
}

## End(Not run)
```

`model_parameters.rma` *Parameters from Meta-Analysis*

Description

Extract and compute indices and measures to describe parameters of meta-analysis models.

Usage

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

Arguments

`model`	Model object.
`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 `bootstrap_parameters()`).
`iterations`	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
`standardize`	The method used for standardizing the parameters. Can be `"refit"`, `"posthoc"`, `"smart"`, `"basic"` or `NULL` (default) for no standardization. See 'Details' in `standardize_parameters`. Note that robust estimation (i.e. `robust=TRUE`) of standardized parameters only works when `standardize="refit"`.
`exponentiate`	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
`...`	Arguments passed to or from other methods. For instance, when `bootstrap = TRUE`, arguments like `ci_method` are passed down to `describe_posterior`.

Value

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

Examples

```
library(parameters)
mydat <<- data.frame(
  effectsize = c(-0.393, 0.675, 0.282, -1.398),
  stderr = c(0.317, 0.317, 0.13, 0.36)
)
if (require("metafor")) {
  model <- rma(yi = effectsize, sei = stderr, method = "REML", data = mydat)
  model_parameters(model)
}

## Not run:
# with subgroups
if (require("metafor")) {
  data(dat.bcg)
  dat <- escalc(
    measure = "RR",
    ai = tpos,
    bi = tneg,
    ci = cpos,
    di = cneg,
    data = dat.bcg
  )
  dat$alloc <- ifelse(dat$alloc == "random", "random", "other")
  model <- rma(yi, vi, mods = ~ alloc, data = dat, digits = 3, slab = author)
  model_parameters(model)
}

## End(Not run)
```

---

model_parameters.stanreg

Parameters from Bayesian Models

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,
  effects = "fixed",
  exponentiate = FALSE,
  standardize = NULL,
  group_level = FALSE,
  ...
)

## S3 method for class 'brmsfit'
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,
  effects = "fixed",
  component = "all",
  exponentiate = FALSE,
  standardize = NULL,
  group_level = FALSE,
  ...
)
```

Arguments

model	Bayesian model. May also be a data frame with posterior samples.
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	Credible Interval (CI) level. Default to 0.89 (89%). See ci for further details.
ci_method	The type of index used for Credible Interval. Can be "HDI" (default, see hdi), "ETI" (see eti) or "SI" (see si).
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 / SI. 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.
effects	Should results for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
exponentiate	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
standardize	The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Note that robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".
group_level	Logical, for multilevel models (i.e. models with random effects) and when effects = "all" or effects = "random", include the parameters for each group level from random effects. If group_level = FALSE (the default), only information on SD and COR are shown.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like ci_method are passed down to describe_posterior.
component	Model component for which parameters should be shown. May be one of "conditional", "precision" (**betareg**), "scale" (**ordinal**), "extra" (**glmx**), "marginal" (**mfx**) or "all".

Details

Currently supported models are brmsfit, stanreg, stanmvreg, MCMCglmm, mcmc and bcplm.

Value

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

Note

When standardize = "refit", columns diagnostic, bf_prior and priors refer to the *original* model. If model is a data frame, arguments diagnostic, bf_prior and priors are ignored.

There is also a plot()-method implemented in the **see**-package.

See Also

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

Examples

```
library(parameters)
if (require("rstanarm")) {
  model <- stan_glm(
    Sepal.Length ~ Petal.Length * Species,
    data = iris, iter = 500, refresh = 0
  )
  model_parameters(model)
}
```

---

model_parameters.zeroinfl

Parameters from Zero-Inflated Models

Description

Parameters from zero-inflated models.

Usage

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

Arguments

model	A model with zero-inflation component.
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 bootstrap_parameters()).

iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.
component	Model component for which parameters should be shown. May be one of "conditional", "precision" (**betareg**), "scale" (**ordinal**), "extra" (**glmx**), "marginal" (**mfx**) or "all".
standardize	The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Note that robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".
exponentiate	Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for, say, logistic regressions, or more generally speaking: for models with log or logit link. **Note:** standard errors are also transformed (by multiplying the standard errors with the exponentiated coefficients), to mimic behaviour of other software packages, such as Stata.
robust	Logical, if TRUE, robust standard errors are calculated (if possible), and confidence intervals and p-values are based on these robust standard errors. Additional arguments like vcov_estimation or vcov_type are passed down to other methods, see standard_error_robust() for details.
p_adjust	Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like ci_method are passed down to describe_posterior.

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

---

n_clusters *Number of clusters to extract*

Description

This function runs many existing procedures for determining how many clusters are present in your data. It returns the number of clusters based on the maximum consensus. In case of ties, it will select the solution with the less clusters.

Usage

```
n_clusters(
  x,
  standardize = TRUE,
  force = FALSE,
  package = c("NbClust", "mclust", "cluster", "M3C"),
  fast = TRUE,
  ...
)
```

Arguments

x	A data frame.
standardize	Standardize the dataframe before clustering (default).
force	Logical, if TRUE, factors are converted to numerical values in order to be included in the data for determining the number of clusters. By default, factors are removed, because most methods that determine the number of clusters need numeric input only.
package	These are the packages from which methods are used to determine the number of clusters. Can be "all" or a vector containing "NbClust", "mclust", "cluster" and "M3C".
fast	If FALSE, will compute 4 more indices (sets index = "allong" in NbClust). This has been deactivated by default as it is computationally heavy.
...	Arguments passed to or from other methods.

Note

There is also a plot()-method implemented in the **see**-package.

Examples

```
library(parameters)

n_clusters(iris[, 1:4], package = c("NbClust", "mclust", "cluster"))
```

---

n_factors *Number of components/factors to retain in PCA/FA*

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 between methods. In case of ties, it will keep the simplest models and select the solution with the less factors.

Usage

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

n_components(
  x,
  type = "PCA",
  rotation = "varimax",
  algorithm = "default",
  package = c("nFactors", "psych"),
  cor = NULL,
  safe = TRUE,
  ...
)
```

Arguments

x	A data frame.
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.
cor	An optional correlation matrix that can be used (note that the data must still be passed as the first argument). If NULL, will compute it by running cor() on the passed data.
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.

Note

There is also a `plot()`-method implemented in the **see**-package. `n_components()` is a convenient short for `n_factors(type = "PCA")`.

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.
- Finch, W. H. (2019). Using Fit Statistic Differences to Determine the Optimal Number of Factors to Retain in an Exploratory Factor Analysis. Educational and Psychological Measurement.
- 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.data.frame(result)
summary(result)

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

`n_parameters` *Count number of parameters in a model*

Description

Returns the number of parameters of a model.

Usage

```
n_parameters(x, ...)

## Default S3 method:
n_parameters(x, ...)

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

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

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

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

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

```
    "sigma", "smooth_terms"),
  ...
)
```

Arguments

x	A statistical model.
...	Arguments passed to or from other methods.
effects	Should number of parameters for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
component	Should total number of parameters, number parameters for the conditional model, the zero-inflated part of the model, the dispersion term or the instrumental variables be returned? Applies to models with zero-inflated and/or dispersion formula, or to models with instrumental variable (so called fixed-effects regressions). May be abbreviated.

Value

The number of parameters in the model.

Examples

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

---

parameters_table *Parameter table formatting*

Description

Parameter table formatting

Usage

```
parameters_table(
  x,
  pretty_names = TRUE,
  stars = FALSE,
  digits = 2,
  ci_digits = 2,
  p_digits = 3,
  ...
)
```

Arguments

x	A data frame of model's parameters.
pretty_names	Pretty parameters' names.
stars	Add significance stars (e.g., p < .001***).
digits	Number of decimal places for numeric values (except confidence intervals and p-values).
ci_digits	Number of decimal places for confidence intervals.
p_digits	Number of decimal places for p-values. May also be "scientific" for scientific notation of p-values.
...	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))
as.data.frame(parameters_table(x, p_digits = "scientific"))

if (require("rstanarm")) {
  model <- stan_glm(Sepal.Length ~ Species, data = iris, refresh = 0, seed = 123)
  x <- model_parameters(model, ci = c(0.69, 0.89, 0.95))
  as.data.frame(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 data frame.

Usage

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

closest_component(x)
```

Arguments

x	A data frame 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 reduce_parameters, 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 / FA 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 (centered 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: Complexity 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 (*Hofman, 1978; Pettersson and Turkheimer, 2010*) .

Uniqueness: 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.

MSA: MSA represents the Kaiser-Meyer-Olkin Measure of Sampling Adequacy (*Kaiser and Rice, 1974*) for each item. It indicates whether there is enough data for each factor give reliable results for the PCA. The value should be > 0.6, and desirable values are > 0.8 (*Tabachnick and Fidell, 2013*).

PCA or FA?: There is a simplified rule of thumb that may help do decide whether to run a factor analysis or a principal component analysis:

- Run factor analysis if you assume or wish to test a theoretical model of latent factors causing observed variables.
- Run principal component analysis If you want to simply reduce your correlated observed variables to a smaller set of important independent composite variables.

(Source: CrossValidated)

Value

A data frame of loadings.

Note

There is a `summary()`-method that prints the Eigenvalues and (explained) variance for each extracted component. `closest_component()` will return a numeric vector with the assigned component index for each column from the original data frame. There is also a `plot()`-method implemented in the **see**-package.

References

- Kaiser, H.F. and Rice. J. (1974). Little jiffy, mark iv. Educational and Psychological Measurement, 34(1):111–117
- Hofmann, R. (1978). Complexity and simplicity as objective indices descriptive of factor solutions. Multivariate Behavioral Research, 13:2, 247-250, doi: 10.1207/s15327906mbr1302_9
- Pettersson, E., & Turkheimer, E. (2010). Item selection, evaluation, and simple structure in personality data. Journal of research in personality, 44(4), 407-420, doi: 10.1016/j.jrp.2010.03.002
- Tabachnick, B. G., and Fidell, L. S. (2013). Using multivariate statistics (6th ed.). Boston: Pearson Education.

See Also

`check_itemscale` to compute various measures of internal consistencies applied to the (sub)scales (i.e. components) extracted from the PCA.

Examples

```
library(parameters)
if (require("psych")) {
  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, rotation = "varimax")
  summary(pca)
  predict(pca)

  # which variables from the original data belong to which extracted component?
  closest_component(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,
  select = NULL,
  digits = 2,
  ci_digits = 2,
  p_digits = 3,
  ...
)
```

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.
`select`	Character vector (or numeric index) of column names that should be printed. If `NULL` (default), all columns are printed. The shortcut `select = "minimal"` prints coefficient, confidence intervals and p-values, while `select = "short"` prints coefficient, standard errors and p-values.

digits	Number of decimal places for numeric values (except confidence intervals and p-values).
ci_digits	Number of decimal places for confidence intervals.
p_digits	Number of decimal places for p-values. May also be "scientific" for scientific notation of p-values.
...	Arguments passed to or from other methods.

Value

NULL

Examples

```
library(parameters)
if (require("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)

  print(mp, select = c("Parameter", "Coefficient", "SE"))

  print(mp, select = "minimal")
}
```

---

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**): By default, p-values are based on Wald-test approximations (see p_value_wald). For certain situations, the "m-l-1" rule might be a better approximation. That is, for method = "ml1", p_value_ml1 is called. For lmerMod objects, if method = "kenward", p-values are based on Kenward-Roger approximations, i.e. p_value_kenward is called, and method = "satterthwaite" calls p_value_satterthwaite.
- Bayesian models (**rstanarm**, **brms**): For Bayesian models, the p-values corresponds to the *probability of direction* (p_direction), which is converted to a p-value using bayestestR::convert_pd_to_p().

Usage

```
p_value(model, ...)

## Default S3 method:
p_value(model, method = NULL, ...)

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

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

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

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

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

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

## S3 method for class 'emmGrid'
p_value(model, ci = 0.95, adjust = "none", ...)

## S3 method for class 'poissonmfx'
p_value(model, component = c("all", "conditional", "marginal"), ...)

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

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

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

## S3 method for class 'clm2'
```

```
p_value(model, component = c("all", "conditional", "scale"), ...)

## S3 method for class 'gee'
p_value(model, method = NULL, ...)
```

Arguments

model	A statistical model.
...	Arguments passed down to standard_error_robust() when confidence intervals or p-values based on robust standard errors should be computed.
method	For mixed models, can be "wald" (default), "ml1", "betwithin", "satterthwaite" or "kenward". For models that are supported by the **sandwich** or **clubSandwich** packages, may also be method = "robust" to compute p-values based ob robust standard errors.
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.
effects	Should standard errors for fixed effects or random effects be returned? Only applies to mixed models. May be abbreviated. When standard errors for random effects are requested, for each grouping factor a list of standard errors (per group level) for random intercepts and slopes is returned.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
adjust	Character value naming the method used to adjust p-values or confidence intervals. See ?emmeans::summary.emmGrid for details.

Value

The p-values.

Note

p_value_robust() resp. p_value(method = "robust") rely on the **sandwich** or **clubSandwich** package (the latter if vcov_estimation = "CR" for cluster-robust standard errors) and will thus only work for those models supported by those packages.

Examples

```
if (require("lme4")) {
  data(iris)
  model <- lmer(Petal.Length ~ Sepal.Length + (1 | Species), data = iris)
  p_value(model)
}
```

qol_cancer *Sample data set*

Description

A sample data set with longitudinal data, used in the vignette describing the `demean()` function.

random_parameters *Summary information from random effects*

Description

This function extracts the different variance components of a mixed model and returns the result as a data frame.

Usage

```
random_parameters(model)
```

Arguments

`model`	A mixed effects model (including `stanreg` models).

Details

The variance components are obtained from `get_variance` and are denoted as following:

Within-group (or residual) variance: The residual variance, σ^2_ϵ, is the sum of the distribution-specific variance and the variance due to additive dispersion. It indicates the *within-group variance*.

Between-group random intercept variance: The random intercept variance, or *between-group* variance for the intercept (τ_{00}), is obtained from `VarCorr()`. It indicates how much groups or subjects differ from each other.

Between-group random slope variance: The random slope variance, or *between-group* variance for the slopes (τ_{11}) is obtained from `VarCorr()`. This measure is only available for mixed models with random slopes. It indicates how much groups or subjects differ from each other according to their 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.

Note: For the within-group and between-group variance, variance and standard deviations (which are simply the square root of the variance) are shown.

Value

A data frame with random effects statistics for the variance components, including number of levels per random effect group, as well as complete observations in the model.

Examples

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

---

reduce_parameters *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. `reduce_data()` is an alias for `reduce_parameters.data.frame()`.

Usage

```
reduce_parameters(x, method = "PCA", n = "max", distance = "euclidean", ...)

reduce_data(x, method = "PCA", n = "max", distance = "euclidean", ...)
```

Arguments

`x`	A data frame 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 `reduce_parameters`, 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. Only applies when `method = "cMDS"`. 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.

Details

The different methods available are described below:

Supervised Methods:

- **PCA**: See `principal_components`.
- **cMDS / 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.
- **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**: 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.

See also package vignette.

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

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

out <- reduce_data(iris, method = "PCA", n = "max")
head(out)
```

`rescale_weights` *Rescale design weights for multilevel analysis*

Description

Most functions to fit multilevel and mixed effects models only allow to specify frequency weights, but not design (i.e. sampling or probability) weights, which should be used when analyzing complex samples and survey data. `rescale_weights()` implements an algorithm proposed by *Asparouhov (2006)* and *Carle (2009)* to rescale design weights in survey data to account for the grouping structure of multilevel models, which then can be used for multilevel modelling.

Usage

```
rescale_weights(data, group, probability_weights, nest = FALSE)
```

Arguments

`data`	A data frame.
`group`	Variable names (as character vector, or as formula), indicating the grouping structure (strata) of the survey data (level-2-cluster variable). It is also possible to create weights for multiple group variables; in such cases, each created weighting variable will be suffixed by the name of the group variable.
`probability_weights`	Variable indicating the probability (design or sampling) weights of the survey data (level-1-weight).
`nest`	Logical, if `TRUE` and `group` indicates at least two group variables, then groups are "nested", i.e. groups are now a combination from each group level of the variables in `group`.

Details

Rescaling is based on two methods: For `pweights_a`, the sample weights `probability_weights` are adjusted by a factor that represents the proportion of group size divided by the sum of sampling weights within each group. The adjustment factor for `pweights_b` is the sum of sample weights within each group divided by the sum of squared sample weights within each group (see *Carle (2009)*, Appendix B).

Regarding the choice between scaling methods A and B, Carle suggests that "analysts who wish to discuss point estimates should report results based on weighting method A. For analysts more interested in residual between-group variance, method B may generally provide the least biased estimates". In general, it is recommended to fit a non-weighted model and weighted models with both scaling methods and when comparing the models, see whether the "inferential decisions converge", to gain confidence in the results.

Though the bias of scaled weights decreases with increasing group size, method A is preferred when insufficient or low group size is a concern.

The group ID and probably PSU may be used as random effects (e.g. nested design, or group and PSU as varying intercepts), depending on the survey design that should be mimicked.

Value

data, including the new weighting variables: pweights_a and pweights_b, which represent the rescaled design weights to use in multilevel models (use these variables for the weights argument).

References

- Carle A.C. (2009). Fitting multilevel models in complex survey data with design weights: Recommendations. BMC Medical Research Methodology 9(49): 1-13
- Asparouhov T. (2006). General Multi-Level Modeling with Sampling Weights. Communications in Statistics - Theory and Methods 35: 439-460

Examples

```
if (require("sjstats")) {
  data(nhanes_sample, package = "sjstats")
  head(rescale_weights(nhanes_sample, "SDMVSTRA", "WTINT2YR"))

  # also works with multiple group-variables...
  head(rescale_weights(nhanes_sample, c("SDMVSTRA", "SDMVPSU"), "WTINT2YR"))

  # or nested structures.
  x <- rescale_weights(
    data = nhanes_sample,
    group = c("SDMVSTRA", "SDMVPSU"),
    probability_weights = "WTINT2YR",
    nest = TRUE
  )
  head(x)
}

if (require("lme4") && require("sjstats")) {
  data(nhanes_sample, package = "sjstats")
  nhanes_sample <- rescale_weights(nhanes_sample, "SDMVSTRA", "WTINT2YR")
  glmer(
    total ~ factor(RIAGENDR) * (log(age) + factor(RIDRETH1)) + (1 | SDMVPSU),
    family = poisson(),
    data = nhanes_sample,
    weights = pweights_a
  )
}
```

---

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 data frame 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)
```

---

select_parameters *Automated selection of model parameters*

Description

This function performs an automated selection of the 'best' parameters, updating and returning the "best" model.

Usage

```
select_parameters(model, ...)

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

## S3 method for class 'merMod'
```

```
select_parameters(model, direction = "backward", steps = 1000, ...)

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

Arguments

model	A statistical model (of class lm, glm, merMod, stanreg or brmsfit).
...	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.

Details

Classical lm and glm: For frequentist GLMs, select_parameters() performs an AIC-based stepwise selection.

Mixed models: For mixed models of class merMod, stepwise selection is based on stepcAIC(). This step function only searches the "best" model based on the random effects structure, i.e. select_parameters() adds or excludes random effects until the cAIC can't be improved further.

Bayesian models: For Bayesian models, it uses the projpred package.

Value

The model refitted with optimal number of parameters.

Examples

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

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

# lme4 ----------------------------------------
if (require("lme4")) {
  model <- lmer(
    Sepal.Width ~ Sepal.Length * Petal.Width * Petal.Length + (1 | Species),
    data = iris
```

```
  )
  select_parameters(model)
}

# rstanarm ---------------------------------------
if (require("rstanarm")) {
  model <- stan_glm(
    mpg ~ ., data = mtcars,
    iter = 500, refresh = 0, verbose = FALSE
  )
  select_parameters(model, cross_validation = TRUE)

  model <- stan_glm(
    mpg ~ cyl * disp * hp, data = mtcars,
    iter = 500, refresh = 0, verbose = FALSE
  )
  select_parameters(model, cross_validation = FALSE)
}
```

`simulate_model` *Simulated draws from model coefficients*

Description

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

Usage

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

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

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.

verbose Show or hide possible warnings and messages.

Details

Technical Details: simulate_model() is a computationally faster alternative to bootstrap_model(). 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

simulate_parameters(), bootstrap_model(), bootstrap_parameters()

Examples

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


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

---

simulate_parameters *Simulate Model Parameters*

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

```
simulate_parameters(model, ...)

## Default S3 method:
simulate_parameters(
  model,
  iterations = 1000,
  centrality = "median",
  ci = 0.95,
  ci_method = "quantile",
  test = "p-value",
  ...
)

## S3 method for class 'glmmTMB'
simulate_parameters(
  model,
  iterations = 1000,
  centrality = "median",
  ci = 0.95,
  ci_method = "quantile",
  test = "p-value",
  ...
)
```

Arguments

model	Statistical model (no Bayesian models).
...	Arguments passed to or from other methods.
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), "ETI" (see eti) or "SI" (see si).
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.

Details

Technical Details: simulate_parameters() is a computationally faster alternative to bootstrap_parameters(). 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.

Note

There is also a `plot()`-method implemented in the **see**-package.

References

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

See Also

`bootstrap_model`, `bootstrap_parameters`, `simulate_model`

Examples

```
library(parameters)

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


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

---

skewness *Compute Skewness and Kurtosis*

Description

Compute Skewness and Kurtosis

Usage

```
skewness(x, na.rm = TRUE, type = "2", iterations = NULL, ...)

kurtosis(x, na.rm = TRUE, type = "2", iterations = NULL, ...)

## S3 method for class 'parameters_kurtosis'
print(x, digits = 3, test = FALSE, ...)

## S3 method for class 'parameters_skewness'
print(x, digits = 3, test = FALSE, ...)
```

Arguments

x	A numeric vector or data.frame.
na.rm	Remove missing values.
type	Type of algorithm for computing skewness. May be one of 1 (or "1", "I" or "classic"), 2 (or "2", "II" or "SPSS" or "SAS") or 3 (or "3", "III" or "Minitab"). See 'Details'.
iterations	The number of bootstrap replicates for computing standard errors. If NULL (default), parametric standard errors are computed. See 'Details'.
...	Arguments passed to or from other methods.
digits	Number of decimal places.
test	Logical, if TRUE, tests if skewness or kurtosis is significantly different from zero.

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*)

Types of Skewness: `skewness()` supports three different methods for estimating skewness, as discussed in *Joanes and Gill (1988)*:

- Type "1" is the "classical" method, which is `g1 = (sum((x -mean(x))^3) / n) / (sum((x -mean(x))^2) / n)^1.5`
- Type "2" first calculates the type-1 skewness, than adjusts the result: `G1 = g1 * sqrt(n * (n -1)) / (n -2)`. This is what SAS and SPSS usually return
- Type "3" first calculates the type-1 skewness, than adjusts the result: `b1 = g1 * ((1 -1 / n))^1.5`. This is what Minitab usually returns.

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*).

Types of Kurtosis: `kurtosis()` supports three different methods for estimating kurtosis, as discussed in *Joanes and Gill (1988)*:

- Type "1" is the "classical" method, which is `g2 = n * sum((x -mean(x))^4) / (sum((x -mean(x))^2)^2) -3`.
- Type "2" first calculates the type-1 kurtosis, than adjusts the result: `G2 = ((n + 1) * g2 + 6) * (n -1)/((n -2) * (n -3))`. This is what SAS and SPSS usually return
- Type "3" first calculates the type-1 kurtosis, than adjusts the result: `b2 = (g2 + 3) * (1 -1 / n)^2 -3`. This is what Minitab usually returns.

Standard Errors: It is recommended to compute empirical (bootstrapped) standard errors (via the `iterations` argument) than relying on analytic standard errors (*Wright & Herrington, 2011*).

Value

Values of skewness or kurtosis.

References

- D. N. Joanes and C. A. Gill (1998). Comparing measures of sample skewness and kurtosis. The Statistician, 47, 183–189.
- Wright, D. B., & Herrington, J. A. (2011). Problematic standard errors and confidence intervals for skewness and kurtosis. Behavior research methods, 43(1), 8-17.

Examples

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

`smoothness` *Quantify the smoothness of a vector*

Description

Quantify the smoothness of a vector

Usage

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

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.
iterations	The number of bootstrap replicates for computing standard errors. If NULL (default), parametric standard errors are computed. See 'Details'.
...	Arguments passed to or from other methods.

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")
```

---

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 *Standard Errors*

Description

`standard_error()` attempts to return standard errors of model parameters, while `standard_error_robust()` attempts to return robust standard errors.

Usage

```
standard_error(model, ...)

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

## Default S3 method:
standard_error(model, method = NULL, ...)

## S3 method for class 'merMod'
standard_error(model, effects = c("fixed", "random"), method = NULL, ...)

## S3 method for class 'glmmTMB'
```

```
standard_error(
  model,
  effects = c("fixed", "random"),
  component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
  ...
)

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

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

## S3 method for class 'coxph'
standard_error(model, method = NULL, ...)

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

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

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

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

## S3 method for class 'poissonmfx'
standard_error(model, component = c("all", "conditional", "marginal"), ...)

## S3 method for class 'betamfx'
standard_error(
  model,
  component = c("all", "conditional", "precision", "marginal"),
  ...
)
```

```
## S3 method for class 'averaging'
standard_error(model, component = c("conditional", "full"), ...)
```

Arguments

model	A model.
...	Arguments passed to or from other methods. For standard_error(), if method = "robust", arguments vcov_estimation, vcov_type and vcov_args can be passed down to standard_error_robust().
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 off warnings.
method	If "robust", robust standard errors are computed by calling standard_error_robust(). standard_error_robust(), in turn, calls one of the vcov*()-functions from the **sandwich** or **clubSandwich** package for robust covariance matrix estimators. For certain mixed models, method may also be one of "wald", "ml1", "betwithin", "satterthwaite" or "kenward".
effects	Should standard errors for fixed effects or random effects be returned? Only applies to mixed models. May be abbreviated. When standard errors for random effects are requested, for each grouping factor a list of standard errors (per group level) for random intercepts and slopes is returned.
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.

Note

For Bayesian models (from **rstanarm** or **brms**), the standard error is the SD of the posterior samples.

Examples

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

`standard_error_robust` *Robust estimation*

Description

`standard_error_robust()`, `ci_robust()` and `p_value_robust()` attempt to return indices based on robust estimation of the variance-covariance matrix, using the packages **sandwich** and **clubSandwich**.

Usage

```
standard_error_robust(
  model,
  vcov_estimation = "HC",
  vcov_type = NULL,
  vcov_args = NULL,
  ...
)

p_value_robust(
  model,
  vcov_estimation = "HC",
  vcov_type = NULL,
  vcov_args = NULL,
  ...
)

ci_robust(
  model,
  ci = 0.95,
  vcov_estimation = "HC",
  vcov_type = NULL,
  vcov_args = NULL,
  ...
)
```

Arguments

`model`	A model.
`vcov_estimation`	String, indicating the suffix of the `vcov*()`-function from the **sandwich** or **clubSandwich** package, e.g. `vcov_estimation = "CL"` (which calls `vcovCL` to compute clustered covariance matrix estimators), or `vcov_estimation = "HC"` (which calls `vcovHC()` to compute heteroskedasticity-consistent covariance matrix estimators).
`vcov_type`	Character vector, specifying the estimation type for the robust covariance matrix estimation (see `vcovHC()` or `clubSandwich::vcovCR()` for details).

vcov_args	List of named vectors, used as additional arguments that are passed down to the **sandwich**-function specified in vcov_estimation.
...	Arguments passed to or from other methods. For standard_error(), if method = "robust", arguments vcov_estimation, vcov_type and vcov_args can be passed down to standard_error_robust().
ci	Confidence Interval (CI) level. Default to 0.95 (95%).

Value

A data frame.

Note

These functions rely on the **sandwich** or **clubSandwich** package (the latter if vcov_estimation = "CR" for cluster-robust standard errors) and will thus only work for those models supported by those packages.

Examples

```
if (require("sandwich")) {
  # robust standard errors, calling sandwich::vcovHC(type="HC3") by default
  model <- lm(Petal.Length ~ Sepal.Length * Species, data = iris)
  standard_error_robust(model)
}

if (require("clubSandwich")) {
  # cluster-robust standard errors, using clubSandwich
  iris$cluster <- factor(rep(LETTERS[1:8], length.out = nrow(iris)))
  standard_error_robust(
    model,
    vcov_type = "CR2",
    vcov_args = list(cluster = iris$cluster)
  )
}
```

Index