Package ‘parameters’

May 29, 2021

Type Package

Title Processing of Model Parameters

Version 0.14.0

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

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 (>= 3.4)

Imports bayestestR (>= 0.9.0), insight (>= 0.13.2), methods, stats, utils

Suggests AER, afex, aod, BayesFactor, BayesFM, bbmle, betareg, blme, boot, brglm2, brms, broom, cAIC4, car, cgam, clubSandwich, cluster, cplm, dplyr, DRR, effects (>= 0.4.4-1), emmeans (>= 1.4), EGAnet (>= 0.7), FactoMineR, fastICA, gam, gamlss, gee, geepack, gglplot2, GLMMadaptive, glmmTMB, GPArotation, gt, lavaan, lavaSearch2, lm.beta, lme4, lmerTest, logspline, lqmm, knitr, MASS, magrittr, Matrix, mclust, MCMCglmm, mediation, metaBMA, metafor, mice, mfx, mgcv, multcomp, multimode, MuMIn, M3C, NbClust, nFactors, nlme, panelr, performance (>= 0.7.2), plm, PMCMRplus, pbkrtest, projpred, pscl, psych, quantreg, randomForest, rmarkdown, rstanarm, sandwich, see, sjstats, spelling, survey, survival, testthat, TMB, tripack, truncreg, VGAM, WRS2

Encoding UTF-8
Language  en-US
RoxygenNote  7.1.1
VignetteBuilder  knitr
Config/testthat/edition  3
NeedsCompilation  no
Author  Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>, @strengejacke),
       Dominique Makowski [aut] (<https://orcid.org/0000-0001-5375-9967>, @Dom_Makowski),
       Mattan S. Ben-Shachar [aut] (<https://orcid.org/0000-0002-4287-4801>),
       Indrajeet Patil [aut] (<https://orcid.org/0000-0003-1995-6531>, @patilindrajeets),
       Søren Højsgaard [aut],
       Brenton M. Wiernik [aut] (<https://orcid.org/0000-0001-9560-6336>, @bmwiernik),
       Zen J. Lau [ctb],
       Vincent Arel-Bundock [ctb] (<https://orcid.org/0000-0003-1995-6531>, @vincentab),
       Jeffrey Girard [ctb] (<https://orcid.org/0000-0002-7359-3746>, @jeffreymgirard)
Repository  CRAN
Date/Publication  2021-05-29 16:30:03 UTC

\textbf{R topics documented:}

\begin{enumerate}
\item bootstrap_model .......................................................... 4
\item bootstrap_parameters ..................................................... 6
\item center ................................................................. 7
\item check_clusterstructure .................................................. 9
\item check_factorstructure .................................................. 10
\item check_heterogeneity .................................................. 11
\item check_kmo ............................................................. 15
\item check_multimodal ..................................................... 16
\item check_sphericity_bartlett ............................................. 17
\item ci.default ........................................................... 18
\item ci_betwithin ......................................................... 22
\item ci_kenward ........................................................... 23
\item ci_ml1 ................................................................. 24
\item ci_satterthwaite ...................................................... 26
\item ci_wald .............................................................. 27
\item cluster_analysis ..................................................... 29
\item cluster_discrimination ............................................... 31
\item compare_parameters .................................................. 31
\item convert_data_to_numeric ............................................. 34
\item convert_efa_to_cfa .................................................. 35
\item data_partition ....................................................... 36
\end{enumerate}
R topics documented:

- degrees_of_freedom ........................................ 37
- describe_distribution ........................................ 38
- display.parameters_model ................................... 40
- equivalence_test.lm ......................................... 45
- factor_analysis ............................................. 48
- fish ............................................................. 52
- format_order .................................................... 52
- format_parameters ............................................ 53
- format_p_adjust ............................................... 54
- get_scores ...................................................... 55
- model_parameters ............................................. 56
- model_parameters.aov ......................................... 58
- model_parameters.averaging .................................. 61
- model_parameters.befa ......................................... 63
- model_parameters.BFBayesFactor ............................. 65
- model_parameters.cgam ......................................... 66
- model_parameters.cpqlmm ...................................... 69
- model_parameters.data.frame ................................. 73
- model_parameters.default .................................... 77
- model_parameters.DirichletRegModel ......................... 81
- model_parameters.htest ....................................... 84
- model_parameters.kmeans ..................................... 86
- model_parameters.lavaan ...................................... 86
- model_parameters.Mclust ...................................... 88
- model_parameters.mira ......................................... 89
- model_parameters.PCA ......................................... 91
- model_parameters.PMCMR ..................................... 93
- model_parameters.rma ......................................... 94
- model_parameters.t1way ....................................... 96
- model_parameters.zcpglm ..................................... 97
- n_clusters ...................................................... 99
- n_factors ...................................................... 100
- parameters_type ............................................. 103
- pool_parameters .............................................. 104
- print.parameters_model ...................................... 106
- p_value ........................................................ 109
- p_value.BFBayesFactor ....................................... 110
- p_value.cpqlmm ............................................... 111
- p_value.DirichletRegModel ................................... 112
- p_value.poissonmfx .......................................... 113
- p_value.zcpglm ............................................... 114
- qol_cancer ..................................................... 115
- random_parameters .......................................... 116
- reduce_parameters ........................................... 117
- rescale_weights .............................................. 119
- reshape_loadings ............................................. 121
- select_parameters .......................................... 122
- simulate_model .............................................. 123
### bootstrap_model

Model bootstrapping

**Description**

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

**Usage**

```r
bootstrap_model(model, iterations = 1000, ...)
```

**Arguments**

- **model**: Statistical model.
- **iterations**: The number of draws to simulate/bootstrap.
- **...**: Arguments passed to or from other methods.
**type**
Character string specifying the type of bootstrap. For mixed models of class `merMod` or `glmmTMB`, may be "parametric" (default) or "semiparametric" (see `?lme4::bootMer` for details). For all other models, see argument `sim` in `?boot::boot` (defaults to "ordinary").

**parallel**
The type of parallel operation to be used (if any).

**n_cpus**
Number of processes to be used in parallel operation.

**verbose**
Toggle warnings and messages.

**Details**
By default, `boot::boot()` is used to generate bootstraps from the model data, which are then used to `update()` the model, i.e. refit the model with the bootstrapped samples. For `merMod` objects (`lme4`) or models from `glmmTMB`, the `lme4::bootMer()` function is used to obtain bootstrapped samples. `bootstrap_parameters()` summarizes the bootstrapped model estimates.

**Value**
A data frame of bootstrapped estimates.

**Using with emmeans**
The output can be passed directly to the various functions from the emmeans package, to obtain bootstrapped estimates, contrasts, simple slopes, etc. and their confidence intervals. These can then be passed to `model_parameters()` to obtain standard errors, p-values, etc (see example).

Note that that p-values returned here are estimated under the assumption of translation equivariance: that shape of the sampling distribution is unaffected by the null being true or not. If this assumption does not hold, p-values can be biased, and it is suggested to use proper permutation tests to obtain non-parametric p-values.

**See Also**
`bootstrap_parameters`, `simulate_model`, `simulate_parameters`

**Examples**
```r
## Not run:
if (require("boot", quietly = TRUE)) {
  model <- lm(mpg ~ wt + factor(cyl), data = mtcars)
  b <- bootstrap_model(model)
  print(head(b))

  if (require("emmeans")) {
    est <- emmeans(b, consec ~ cyl)
    print(model_parameters(est))
  }
}
## End(Not run)
```
**bootstrap_parameters**  Parameters bootstrapping

**Description**

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

**Usage**

```r
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 .95 (95%).
- `ci_method`  The type of index used for Credible Interval. Can be "HDI" (default, see `hdi`), "ETI" (see `eti`), "BCI" (see `bci`) 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.

Note that that p-values returned here are estimated under the assumption of translation equivariance: that shape of the sampling distribution is unaffected by the null being true or not. If this assumption does not hold, p-values can be biased, and it is suggested to use proper permutation tests to obtain non-parametric p-values.
Value

A data frame summarizing the bootstrapped parameters.

Using with emmeans

The output can be passed directly to the various functions from the emmeans package, to obtain bootstrapped estimates, contrasts, simple slopes, etc. and their confidence intervals. These can then be passed to model_parameters() to obtain standard errors, p-values, etc (see example).

Note that that p-values returned here are estimated under the assumption of translation equivariance: that shape of the sampling distribution is unaffected by the null being true or not. If this assumption does not hold, p-values can be biased, and it is suggested to use proper permutation tests to obtain non-parametric p-values.

References


See Also

bootstrap_model, simulate_parameters, simulate_model

Examples

```r
## Not run:
if (require("boot", quietly = TRUE)) {
  set.seed(2)
  model <- lm(Sepal.Length ~ Species * Petal.Width, data = iris)
  b <- bootstrap_parameters(model)
  print(b)

  if (require("emmeans")) {
    est <- emmeans(b, trt.vs.ctrl ~ Species)
    print(model_parameters(est))
  }
}
## End(Not run)
```

Description

Performs a grand-mean centering of data.
Usage

center(x, ...)  
  
## S3 method for class 'numeric'

## S3 method for class 'data.frame'

Arguments

x  
A data frame, a (numeric or character) vector or a factor.

...  
Currently not used.

weights  
Can be NULL (for no weighting), or:

  • For data frames: a numeric vector of weights, or a character of the name of a column in the data.frame that contains the weights.
  • For numeric vectors: a numeric vector of weights.

robust  
Logical, if TRUE, centering is done by subtracting the median from the variables. If FALSE, variables are centered by subtracting the mean.

verbose  
Toggle warnings and messages.

select  
Character vector of column names. If NULL (the default), all variables will be selected.

exclude  
Character vector of column names to be excluded from selection.

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

append  
Logical, if TRUE and x is a data frame, standardized variables will be added as additional columns; if FALSE, existing variables are overwritten.

suffix  
Character value, will be appended to variable (column) names of x, if x is a data frame and append = TRUE.

Value

The centered variables.
check_clusterstructure

See Also

If centering within-clusters (instead of grand-mean centering) is required, see `demean`.

Examples

```r
data(iris)
head(iris$Sepal.Width)
head(center(iris$Sepal.Width))
head(center(iris))
head(center(iris, force = TRUE))
```

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

```r
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

See Also

`check_kmo, check_sphericity_bartlett` and `check_factorstructure`.

Examples

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

---

```r
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_bartlett` and `check_clusterstructure`.

Examples

```r
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.

degroup() is more generic in terms of the centering-operation. While demean() always uses mean-centering, degroup() can also use the mode or median for centering.

Usage

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

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

degroup(
  x,
  select,
  group,
  center = "mean",
  suffix_demean = "_within",
  suffix_groupmean = "_between",
  add_attributes = TRUE,
  verbose = TRUE
)

detrend(
  x,
  select,
  group,
  center = "mean",
  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 warnings and messages.
- **center**: Method for centering. `demean()` always performs mean-centering, while `degroup()` can use center = "median" or center = "mode" for median- or mode-centering, and also "min" or "max".

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 regression coefficient of a group-meaned variable is the *between-subject-effect*. 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. The regression coefficient of a de-meaned variable represents the *within-subject-effect*.

**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, there are two recommendations. First is to include the raw (untransformed) binary predictor as fixed effect only and the de-meaned variable as random effect (random slope). The alternative would be to add the de-meaned version(s) of binary time-varying covariates as additional fixed effect as well (instead of adding it as random slope). Centering time-varying binary variables to obtain within-effects (level 1) isn’t necessary. They have a sensible interpretation when left in the typical 0/1 format (Hoffmann 2015, chapter 8-2.1), `demean()` will thus coerce categorical time-varying predictors to numeric to compute the de- and group-meaned versions for these variables, where the raw (untransformed) binary predictor and the de-meaned version should be added to the model.

**De-meaning of factors with more than 2 levels:** Factors with more than two levels are demeaned 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 demeaned 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**


**See Also**

If grand-mean centering (instead of centering within-clusters) is required, see center.
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, meritorious, 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 credit goes to its author.

Value
A list of indices related to KMO.

References

Examples
```r
library(parameters)
check_kmo(mtcars)
```

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

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

References


Examples

```r
## Not run:
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)
}
## End(Not run)
```

check_sphericity_bartlett

Bartlett’s Test of Sphericity

Description

Bartlett’s (1951) test of sphericity tests whether a matrix (of correlations) is significantly different from an identity matrix. The test provides probability that the correlation matrix has significant correlations among at least some of the variables in a dataset, a prerequisite for factor analysis to work. In other words, before starting with factor analysis, one needs to check whether Bartlett’s test of sphericity is significant.

Usage

```r
check_sphericity_bartlett(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 (Revell, 2016). All credit goes to its author.

Value

A list of indices related to sphericity.

References


Examples

```r
library(parameters)
check_sphericity_bartlett(mtcars)
```

---

### ci.default

**Confidence Intervals (CI)**

**Description**

Compute confidence intervals (CI) for frequentist models.

**Usage**

```r
## Default S3 method:
ci(x, ci = 0.95, dof = Inf, method = NULL, ...)

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

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

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

## S3 method for class 'glmmTMB'
```
ci.default

x,
ci = 0.95,
component = c("all", "conditional", "zi", "zero_inflated", "dispersion"),
method = c("wald", "ml1", "betwithin", "robust", "profile", "uniroot"),
verbose = TRUE,
...
)

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

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

## 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 'MixMod'
ci(  
x,  
ci = 0.95,  
component = c("all", "conditional", "zi", "zero_inflated"),  
verbose = TRUE,  
...
)

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

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

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

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

## S3 method for class 'HLfit'
.ci(  
x,  
   ci = 0.95,  
   method = c("wald", "ml1", "betwithin", "profile", "boot"),  
   iterations = 100,  
   ...  
)

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

### Arguments

- **x** 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.
- **method** For mixed models, can be "wald" (default), "ml1" or "betwithin". For linear mixed model, can also be "satterthwaite", "kenward" or "boot" (see
lme4::confint.merMod). For (generalized) linear models, can be "robust" to compute confidence intervals based on robust covariance matrix estimation, and for generalized linear models and models from packages \texttt{lme4} or \texttt{glmmTMB}, may also be "profile", "uniroot" or "wald" (default).

... Arguments passed down to \texttt{standard_error_robust()} when confidence intervals or p-values based on robust standard errors should be computed.

\textbf{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. \texttt{component} may be one of "conditional", "zi", "zero-inflated", "dispersion" or "all" (default). May be abbreviated.

\textbf{verbose} Toggle warnings and messages.

\textbf{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.

\textbf{iterations} The number of draws to simulate/bootstrap.

\textbf{Value}

A data frame containing the CI bounds.

\textbf{Note}

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

\textbf{Examples}

```r
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

```r
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.
ci_kenward

References


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

```r
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

```r
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


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

```r
if (require("lme4", quietly = TRUE)) {
  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 statistics (see Li and Redden 2015). The \(ml-1\) 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 \(ml-1\) 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 "ml-1" Heuristic:** Note that the "ml-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 (\(dof_kendward()\)) is recommended. However, the "ml-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

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

```r
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**

```r
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 statistics. Unlike simpler approximation heuristics like the "m-l-1" rule (dof_m1()), 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


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

Examples

```r
if (require("lme4", quietly = TRUE)) {
  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

```r
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.

cli 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",
"dispersion" or "all" (default). May be abbreviated.

robust Logical, if TRUE, robust standard errors are calculated (if possible), and con-
fidence intervals and p-values are based on these robust standard errors. Ad-
ditional arguments like vcov_estimation or vcov_type are passed down to
other methods, see standard_error_robust() for details and this vignette for
working examples.

... Arguments passed down to standard_error_robust() when confidence inter-
vals 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 Package from which methods are to be called to determine the number of clusters. Can be "all" or a vector containing "NbClust", "mclust", "cluster" and "M3C".

verbose Toggle warnings and messages.

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

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

```r
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

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

# goodness of group classification
cluster_discrimination(iris[, 1:4], cluster_groups = groups)
## End(Not run)
```

compare_parameters

*Compare model parameters of multiple models*

Description

Compute and extract model parameters of multiple regression models. See `model_parameters` for further details.
Usage

compare_parameters(
  ..., 
  ci = 0.95,
  effects = "fixed",
  component = "conditional",
  standardize = NULL,
  exponentiate = FALSE,
  df_method = "wald",
  p_adjust = NULL,
  style = NULL,
  column_names = NULL,
  groups = NULL,
  verbose = TRUE
)

compare_models(
  ..., 
  ci = 0.95,
  effects = "fixed",
  component = "conditional",
  standardize = NULL,
  exponentiate = FALSE,
  df_method = "wald",
  p_adjust = NULL,
  style = NULL,
  column_names = NULL,
  groups = NULL,
  verbose = TRUE
)

Arguments

... One or more regression model objects, or objects returned by `model_parameters()`. Regression models may be of different model types.

ci Confidence Interval (CI) level. Default to 0.95 (95%).
effects Should parameters for fixed effects ("fixed"), random effects ("random"), or both ("all") be returned? Only applies to mixed models. May be abbreviated.
component Model component for which parameters should be shown. See documentation for related model class in `model_parameters`.
standardize The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic", "pseudo" or NULL (default) for no standardization. See 'Details' in `standardize_parameters`. **Important:** Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use `standardize="basic"` to mimic behaviour of SPSS or packages such as `lm.beta`, or standardize the data.
**compare_parameters**

with `effectsize::standardize(force=TRUE)` before fitting the model. Robust estimation (i.e. robust=TRUE) of standardized parameters only works when `standardize="refit"`.

**exponentiate** Logical, indicating whether or not to exponentiate the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. **Note:** Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = "nongaussian" will only exponentiate coefficients from non-Gaussian families.

**df_method** Method for computing degrees of freedom for p values, standard errors and confidence intervals (CI). See documentation for related model class in model_parameters.

**p_adjust** Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from emmeans).

**style** String, indicating which style of output is requested. Following templates are possible:

- "ci": Estimate and confidence intervals, no asterisks for p-values.
- "se": Estimate and standard errors, no asterisks for p-values.
- "ci_p": Estimate, confidence intervals and asterisks for p-values.
- "se_p": Estimate, standard errors and asterisks for p-values.
- "ci_p2": Estimate, confidence intervals and numeric p-values, in two columns.
- "se_p2": Estimate, standard errors and numeric p-values, in two columns.

**column_names** Character vector with strings that should be used as column headers. Must be of same length as number of models in ....

**groups** Named list, can be used to group parameters in the printed output. List elements may either be character vectors that match the name of those parameters that belong to one group, or list elements can be row numbers of those parameter rows that should belong to one group. The names of the list elements will be used as group names, which will be inserted as "header row". A possible use case might be to emphasize focal predictors and control variables, see "Examples". Parameters will be re-ordered according to the order used in groups, while all non-matching parameters will be added to the end.

**verbose** Toggle warnings and messages.

**Value**

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

**Note**

This function is in an early stage and does not yet cope with more complex models, and probably does not yet properly render all model components.
Examples

data(iris)
lm1 <- lm(Sepal.Length ~ Species, data = iris)
lm2 <- lm(Sepal.Length ~ Species + Petal.Length, data = iris)
lm3 <- lm(Sepal.Length ~ Species * Petal.Length, data = iris)
compare_parameters(lm1, lm2, lm3)

data(mtcars)
m1 <- lm(mpg ~ wt, data = mtcars)
m2 <- glm(vs ~ wt + cyl, data = mtcars, family = "binomial")
compare_parameters(m1, m2)

## Not run:
# exponentiate coefficients, but not for lm
compare_parameters(m1, m2, exponentiate = "nongaussian")

# change column names
compare_parameters(m1, m2, column_names = c("linear model", "logistic reg."))

## 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

```r
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, seed = 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.
seed A random number generator seed. Enter an integer (e.g., 123) so that the random sampling will be the same each time you run the function.

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.

Note

In many cases, degrees_of_freedom returns the same as df.residuals, or n-k (number of observations minus number of parameters). However, degrees_of_freedom refers to the model’s parameters degrees of freedom of the distribution for the related test statistic. Thus, for models with z-statistic, results from degrees_of_freedom and df.residuals differ. Furthermore, for other approximation methods like "kenward" or "satterthwaite", each model parameter can have a different degree of freedom.

Examples

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

model <- glm(vs ~ mpg * cyl, data = mtcars, family = "binomial")
dof(model)
## Not run:
if (require("lme4", quietly = TRUE)) {
  model <- lmer(Sepal.Length ~ Petal.Length + (1 | Species), data = iris)
  dof(model)
}
if (require("rstanarm", quietly = TRUE)) {
  model <- stan_glm(
    Sepal.Length ~ Petal.Length * Species,
    data = iris,
    chains = 2,
    refresh = 0
  )
  dof(model)
}
## End(Not run)
```

---

**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**

```r
describe_distribution(x, ...)  
```

```r
## S3 method for class 'numeric'
describe_distribution(
  x,
  centrality = "mean",
  dispersion = TRUE,
  iqr = TRUE,
  range = TRUE,
  quartiles = FALSE,
  ci = NULL,
  iterations = 100,
  threshold = 0.1,
  ...
)
```

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

```r
## S3 method for class 'data.frame'
describe_distribution(
  x,
  centrality = "mean",
  dispersion = TRUE,
  iqr = TRUE,
  range = TRUE,
  quartiles = FALSE,
  include_factors = FALSE,
  ci = NULL,
  iterations = 100,
  threshold = 0.1,
  ...
)
```

**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).
- **quartiles** Return the first and third quartiles (25th and 75th percentiles).
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). If centrality = "all", the bootstrapped confidence interval refers to the first centrality index (which is typically the median).

iterations  The number of bootstrap replicates for computing confidence intervals. Only applies when ci is not NULL.

threshold  For centrality = "trimmed" (i.e. trimmed mean), indicates the fraction (0 to 0.5) of observations to be trimmed from each end of the vector before the mean is computed.

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, quartiles = TRUE)

display.parameters_model
Print tables in different output formats

Description
Prints tables (i.e. data frame) in different output formats. print_md() is a alias for display(format = "markdown").

Usage

## S3 method for class 'parameters_model'
display(
  object,
  format = "markdown",
  pretty_names = TRUE,
  split_components = TRUE,
)
select = NULL,
caption = NULL,
subtitle = NULL,
footer = NULL,
align = NULL,
digits = 2,
CI_digits = 2,
PF_digits = 3,
footer_digits = 3,
ci_brackets = c("",""),
show_sigma = FALSE,
show_formula = FALSE,
zap_small = FALSE,
verbose = TRUE,
...
)

## S3 method for class 'parameters_stan'
display(
  object,
  split_components = TRUE,
  select = NULL,
  format = "markdown",
  ...
)

## S3 method for class 'parameters_sem'
display(
  object,
  format = "markdown",
  digits = 2,
  CI_digits = 2,
  PF_digits = 3,
  ci_brackets = c("",""),
  ...
)

## S3 method for class 'parameters_efa_summary'
display(object, format = "markdown", digits = 3, ...)

## S3 method for class 'parameters_efa'
display(
  object,
  format = "markdown",
  digits = 2,
  sort = FALSE,
  threshold = NULL,
  labels = NULL,
display.parameters_model

## S3 method for class 'equivalence_test_lm'
display(object, format = "markdown", digits = 2, ...)

## S3 method for class 'parameters_model'
format(
  x,
  pretty_names = TRUE,
  split_components = TRUE,
  select = NULL,
  digits = 2,
  ci_digits = 2,
  p_digits = 3,
  ci_width = NULL,
  ci_brackets = NULL,
  zap_small = FALSE,
  format = NULL,
  groups = NULL,
  ...
)

## S3 method for class 'parameters_model'
print_html(
  x,
  pretty_names = TRUE,
  split_components = TRUE,
  select = NULL,
  caption = NULL,
  subtitle = NULL,
  footer = NULL,
  align = NULL,
  digits = 2,
  ci_digits = 2,
  p_digits = 3,
  footer_digits = 3,
  ci_brackets = c("", ""),
  show_sigma = FALSE,
  show_formula = FALSE,
  zap_small = FALSE,
  groups = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'parameters_model'
print_md(
  x,
  pretty_names = TRUE,
  split_components = TRUE,
  select = NULL,
  caption = NULL,
  subtitle = NULL,
  footer = NULL,
  align = NULL,
  digits = 2,
  ci_digits = 2,
  p_digits = 3,
Arguments

object    An object returned by `model_parameters()`, `simulate_parameters()`, `equivalence_test()` or `principal_components()`.
format    String, indicating the output format. Can be "markdown" or "html".
pretty_names Return "pretty" (i.e. more human readable) parameter 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.
caption   Table caption as string. If NULL, no table caption is printed.
subtitle  Table title (same as caption) and subtitle, as strings. If NULL, no title or subtitle is printed, unless it is stored as attributes (table_title, or its alias table_caption, and table_subtitle).
footer    Table footer, as string. For markdown-formatted tables, table footers, due to the limitation in markdown rendering, are actually just a new text line under the table.
align     Only applies to HTML tables. May be one of "left", "right" or "center".
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.
footer_digits  Number of decimal places for values in the footer summary.
ci_brackets Logical, if TRUE (default), CI-values are encompassed in square brackets (else in parentheses).
show_sigma Logical, if TRUE, adds information about the residual standard deviation.
show_formula Logical, if TRUE, adds the model formula to the output.
zap_small Logical, if TRUE, small values are rounded after digits decimal places. If FALSE, values with more decimal places than digits are printed in scientific notation.
verbose  Toggle messages and warnings.
... Arguments passed to or from other methods.
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.
x  An object returned by model_parameters()．
ci_width  Minimum width of the returned string for confidence intervals. If not NULL and width is larger than the string's length, leading whitespaces are added to the string. If width="auto", width will be set to the length of the longest string.
groups Named list, can be used to group parameters in the printed output. List elements may either be character vectors that match the name of those parameters that belong to one group, or list elements can be row numbers of those parameter rows that should belong to one group. The names of the list elements will be used as group names, which will be inserted as "header row". A possible use case might be to emphasize focal predictors and control variables, see 'Examples'. Parameters will be re-ordered according to the order used in groups, while all non-matching parameters will be added to the end.

Details
display() is useful when the table-output from functions, which is usually printed as formatted text-table to console, should be formatted for pretty table-rendering in markdown documents, or if knitted from rmarkdown to PDF or Word files. See vignette for examples.

Value
If format = "markdown", the return value will be a character vector in markdown-table format. If format = "html", an object of class gt_tbl.
Examples

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

---

### equivalence_test.lm

**Equivalence test**

#### Description

Compute the (conditional) equivalence test for frequentist models.

#### Usage

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

```r
## S3 method for class 'merMod'
equivalence_test(
  x,
  range = "default",
  ci = 0.95,
  rule = "classic",
  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.
In classical null hypothesis significance testing (NHST) within a frequentist framework, it is not possible to accept the null hypothesis, \( H_0 \) - 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. \( H_0 \)) 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. \( H_0 \) 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 \( H_0 \) 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 \( H_0 \), 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**


**See Also**

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

```r
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", quietly = TRUE)) {
  result <- equivalence_test(model)
  plot(result)
}
```

factor_analysis

Principal Component Analysis (PCA) and Factor Analysis (FA)

Description

The functions `principal_components()` and `factor_analysis()` can be used to perform a principal component analysis (PCA) or a factor analysis (FA). They return the loadings as a data frame, and various methods and functions are available to access / display other information (see the Details section).

Usage

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

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

closest_component(pca_results)
rotated_data(pca_results)

## S3 method for class 'parameters_efa'
predict(object, newdata = NULL, names = NULL, keep_na = TRUE, ...)

## S3 method for class 'parameters_efa'
print(x, digits = 2, sort = FALSE, threshold = NULL, labels = NULL, ...)

## S3 method for class 'parameters_efa'
sort(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 num-
ber of components is selected through n_factors resp. n_components. 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",
or "cluster" (and more). 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 (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.
pca_results
The output of the principal_components() function.
object
An object of class parameters_pca or parameters_efa
newdata
An optional data frame in which to look for variables with which to predict. If
omitted, the fitted values are used.

names
Optional character vector to name columns of the returned data frame.
keep_na
Logical, if TRUE, predictions also return observations with missing values from
the original data, hence the number of rows of predicted data and original data
is equal.
digits, labels  Arguments for print().

Details

Methods and Utilities:

- **n_components** and **n_factors** automatically estimate the optimal number of dimensions to retain.
- **check_factorstructure** checks the suitability of the data for factor analysis using the sphericity and the sphericity KMO measure.
- **check_itemscale** computes various measures of internal consistencies applied to the (sub)scales (i.e., components) extracted from the PCA.
- Running **summary** returns information related to each component/factor, such as the explained variance and the Eivenvalues.
- Running **get_scores** computes scores for each subscale.
- Running **closest_component** will return a numeric vector with the assigned component index for each column from the original data frame.
- Running **rotated_data** will return the rotated data, including missing values, so it matches the original data frame.
- Running **plot()** visually displays the loadings (that requires the see package to work).

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)

Computing Item Scores:  Use **get_scores** to compute scores for the "subscales" represented by the extracted principal components. **get_scores()** takes the results from **principal_components()** and extracts the variables for each component found by the PCA. Then, for each of these "sub-scales", raw 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. One can also use
\texttt{predict()} to back-predict scores for each component, to which one can provide \texttt{newdata} or a
vector of names for the components.

**Explained Variance and Eigenvales:** Use \texttt{summary()} to get the Eigenvalues and the ex-
plained variance for each extracted component. The eigenvectors and eigenvalues represent the
"core" of a PCA: The eigenvectors (the principal components) determine the directions of the new
feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues ex-
plain the variance of the data along the new feature axes.

**Value**

A data frame of loadings.

**References**

- Hofmann, R. (1978). Complexity and simplicity as objective indices descriptive of factor solu-
- Pettersson, E., & Turkheimer, E. (2010). Item selection, evaluation, and simple structure in
Pearson Education.

**Examples**

library(parameters)

# Principal Component Analysis (PCA) -------------------
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")
pca # Print loadings
summary(pca) # Print information about the factors
predict(pca, names = c("Component1", "Component2")) # Back-predict scores

# which variables from the original data belong to which extracted component?
closest_component(pca)
# rotated_data(pca) # TODO: doesn't work
# Automated number of components
principal_components(mtcars[,1:4], n = "auto")

# Factor Analysis (FA) ------------------------
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)

Description
This functions formats the names of model parameters (coefficients) to make them more human-readable.

Usage
format_parameters(model, ...)

## Default S3 method:
format_parameters(model, brackets = c("[", "]"), ...)

Arguments
model A statistical model.
... Currently not used.
brackets A character vector of length two, indicating the opening and closing brackets.

Value
A (names) character vector with formatted parameter names. The value names refer to the original names of the coefficients.

Interpretation of Interaction Terms
Note that the interpretation of interaction terms depends on many characteristics of the model. The number of parameters, and overall performance of the model, can differ or not between a * b a : b, and a / b, suggesting that sometimes interaction terms give different parameterizations of the same model, but other times it gives completely different models (depending on a or b being factors of covariates, included as main effects or not, etc.). Their interpretation depends of the full context of the model, which should not be inferred from the parameters table alone - rather, we recommend to use packages that calculate estimated marginal means or marginal effects, such as modelbased, emmeans or ggeffects. To raise awareness for this issue, you may use print(..., show_formula=TRUE) to add the model-specification to the output of the print() method for model_parameters().
Examples

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

---

**format_p_adjust**

*Format the name of the p-value adjustment methods*

**Description**

Format the name of the p-value adjustment methods.

**Usage**

```r
format_p_adjust(method)
```

**Arguments**

- `method` Name of the method.

**Value**

A string with the full surname(s) of the author(s), including year of publication, for the adjustment-method.

**Examples**

```r
library(parameters)

format_p_adjust("holm")
format_p_adjust("bonferroni")
```
**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**

```r
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**

```r
if (require("psych")) {
  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
}
```
Description

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

- Correlations, t-tests, ... (htest, pairwise.htest)
- ANOVAs (aov, anova, afex, ...)
- Regression models (lm, glm, survey, ...)
- Additive models (gam, gamm, ...)
- Zero-inflated models (hurdle, zeroinfl, zerocount)
- Multinomial, ordinal and cumulative link models (bracl, multinom, m1m, ...)
- Other special models (model.avg, betareg, glmx, ...)
- Mixed models (lme4, nlme, glmmTMB, afex, ...)
- Bayesian tests (BayesFactor)
- Bayesian models (rstanarm, brms, MCMCglmm, blavaan, ...)
- PCA and FA (psych)
- CFA and SEM (lavaan)
- Cluster models (k-means, ...)
- Meta-Analysis via linear (mixed) models (rma, metaplus, metaBMA, ...)
- Hypothesis testing (glht, PMCMRplus)
- Robust statistical tests (WRS2)
- 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, ci_digits and footer_digits to set the number of digits for the output. group can also be passed to the print() method. See details in print.parameters_model and 'Examples' in model_parameters.default.
Details

**Standardization of model coefficients:** 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. The default, `standardize = "refit"`, never standardizes categorical predictors (i.e., factors), which may be a different behaviour compared to other R packages or other software packages (like SPSS). To mimic behaviour of SPSS or packages such as `lm.beta`, use `standardize = "basic"`.

**Methods of standardization:** For full details, please refer to `standardize_parameters()`.

- **refit** This method is based on a complete model re-fit with a standardized version of the data. Hence, this method is equal to standardizing the variables before fitting the model. It is the "purest" and the most accurate (Neter et al., 1989), but it is also the most computationally costly and long (especially for heavy models such as Bayesian models). The `robust` argument (default to `FALSE`) enables a robust standardization of data, i.e., based on the median and MAD instead of the mean and SD.

- **posthoc** Post-hoc standardization of the parameters, aiming at emulating the results obtained by "refit" without refitting the model. The coefficients are divided by the standard deviation (or MAD if `robust=TRUE`) of the outcome (which becomes their expression 'unit'). Then, the coefficients related to numeric variables are additionally multiplied by the standard deviation (or MAD) of the related terms, so that they correspond to changes of 1 SD of the predictor. This does not apply to binary variables or factors, so the coefficients are still related to changes in levels. This method is not accurate and tend to give aberrant results when interactions are specified.

- **smart** (Standardization of Model’s parameters with Adjustment, Reconnaissance and Transformation - experimental): Similar to method="posthoc" in that it does not involve model refitting. The difference is that the SD (or MAD) of the response is computed on the relevant section of the data. For instance, if a factor with 3 levels A (the intercept), B and C is entered as a predictor, the effect corresponding to B vs. A will be scaled by the variance of the response at the intercept only. As a results, the coefficients for effects of factors are similar to a Glass’ delta.

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

- **pseudo (for 2-level (G)LMMs only)** In this (post-hoc) method, the response and the predictor are standardized based on the level of prediction (levels are detected with `check_heterogeneity`): Predictors are standardized based on their SD at level of prediction (see also `demean`). The outcome (in linear LMMs) is standardized based on a fitted random-intercept-model, where `sqrt(random-intercept-variance)` is used for level 2 predictors, and `sqrt(residual-variance)` is used for level 1 predictors (Hoffman 2015, page 342). A warning is given when a within-group variable is found to have access between-group variance.

**Value**

A data frame of indices related to the model’s parameters.
Labeling the Degrees of Freedom

Throughout the parameters package, we decided to label the residual degrees of freedom \textit{df\_error}. The reason for this is that these degrees of freedom not always refer to the residuals. For certain models, they refer to the estimate error - in a linear model these are the same, but in - for instance - any mixed effects model, this isn't strictly true. Hence, we think that \textit{df\_error} is the most generic label for these degrees of freedom.

Interpretation of Interaction Terms

Note that the \textit{interpretation} of interaction terms depends on many characteristics of the model. The number of parameters, and overall performance of the model, can differ \textit{or not} between \texttt{a * b a : b}, and \texttt{a / b}, suggesting that sometimes interaction terms give different parameterizations of the same model, but other times it gives completely different models (depending on \texttt{a} or \texttt{b} being factors of covariates, included as main effects or not, etc.). Their interpretation depends of the full context of the model, which should not be inferred from the parameters table alone - rather, we recommend to use packages that calculate estimated marginal means or marginal effects, such as \texttt{modelbased, emmeans} or \texttt{ggeffects}. To raise awareness for this issue, you may use \texttt{print(..., show_formula=TRUE)} to add the model-specification to the output of the \texttt{print()} method for \texttt{model_parameters()}. 

Note

The \texttt{print()} method has several arguments to tweak the output. There is also a \texttt{plot}-method implemented in the \texttt{see}-package, and a dedicated method for use inside rmarkdown files, \texttt{print_md()}. 

References


See Also

\texttt{standardize_names()} to rename columns into a consistent, standardized naming scheme.

---

### model\_parameters.aov

**Parameters from ANOVAs**

**Description**

Parameters from ANOVAs
Usage

```r
## S3 method for class 'aov'
model_parameters(
  model,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  df_error = NULL,
  type = NULL,
  ci = NULL,
  test = NULL,
  power = FALSE,
  parameters = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

- `model` Object of class `aov`, `anova`, `aovlist`, `Gam`, `manova`, `Anova.mlm`, `afex_aov` or `maov`.
- `omega_squared` Compute omega squared as index of effect size. Can be "partial" (the default, adjusted for effect size) or "raw".
- `eta_squared` Compute eta squared as index of effect size. Can be "partial" (the default, 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" (the default, 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'. (Ignored for `afex_aov`.)
- `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. (Ignored for `afex_aov`.)
- `ci` Confidence Interval (CI) level for effect sizes `omega_squared`, `eta_squared` etc. The default, `NULL`, will compute no confidence intervals. `ci` should be a scalar between 0 and 1.
- `test` String, indicating the type of test for `Anova.mlm` to be returned. If "multivariate" (or `NULL`), returns the summary of the multivariate test (that is also given by the `print`-method). If `test = "univariate"`, returns the summary of the univariate test.
- `power` Logical, if `TRUE`, adds a column with power for each parameter.
- `parameters` Character vector of length 1 with a regular expression pattern that describes the parameters that should be returned from the data frame, or a named list of regular expressions. All non-matching parameters will be removed from the
output. If parameters is a character vector, every parameter in the “Parameters” column that matches the regular expression in parameters will be selected from the returned data frame. Furthermore, if parameters has more than one element, these will be merged into a regular expression pattern like this: “(one|two|three)”. If parameters is a named list of regular expression patterns, the names of the list-element should equal the column name where selection should be applied. This is useful for model objects where model_parameters() returns multiple columns with parameter components, like in model_parameters.lavaan.

verbose
Toggle warnings and messages.

... 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. Note that type 3 ANOVAs with interactions involved only give sensible and informative results when covariates are mean-centred and factors are coded with orthogonal contrasts (such as those produced by contr.sum, contr.poly, or contr.helmert, but not by the default contr.treatment).

Examples

```r
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_parameters(
    model,
    omega_squared = "partial",
    eta_squared = "partial",
    ci = .9
  )

  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)

## Not run:
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",
    ci = .9,
    df_error = dof_satterthwaite(mm)[2:3]
  )
}

## End(Not run)

model_parameters.averaging

Parameters from special models

Description
Parameters from special regression models not listed under one of the previous categories yet.

Usage

## S3 method for class 'averaging'
model_parameters(
  model,
  ci = 0.95,
  component = c("conditional", "full"),
  exponentiate = FALSE,
  p_adjust = NULL,
  verbose = TRUE,
  ...
)

## 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,
  verbose = TRUE,
  ...
)

## 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,
  verbose = TRUE,
  ...
)

Arguments

model Model object.

.ci Confidence Interval (CI) level. Default to 0.95 (95%).

.component Model component for which parameters should be shown. May be one of
"conditional", "precision" (betareg), "scale" (ordinal), "extra" (glmx),
"marginal" (mfx), "conditional" or "full" (for MuMIn::model.avg()) or
"all".

.exponentiate Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. Note: Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = "nongaussian" will only exponentiate coefficients from non-Gaussian families.

.p_adjust Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from emmeans).
model_parameters.befa

Parameters from PCA/FA

Description

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

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

verbose

Toggle warnings and messages.

Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like type or parallel are passed down to bootstrap_model(), and arguments like ci_method are passed down to describe_posterior.

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", "pseudo" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Important: Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use standardize="basic" to mimic behaviour of SPSS or packages such as lm.beta, or standardize the data with effectsize::standardize(force=TRUE) before fitting the model. Robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".

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", quietly = TRUE)) {
  data("stemcell")
  model <- bracl(
    research ~ as.numeric(religion) + gender,
    weights = frequency,
    data = stemcell,
    type = "ML"
  )
  model_parameters(model)
}
Usage

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

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 0.95 (95%).
- **ci_method**: The type of index used for Credible Interval. Can be "HDI" (default, see `hdi`), "ETI" (see `eti`), "BCI" (see `bci`) 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.
- **verbose**: Toggle off warnings.
- **...**: Arguments passed to or from other methods.

Value

A data frame of loadings.

Examples

```r
library(parameters)

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

Description
Parameters from 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,
  verbose = 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. De-
  fault to .95 (95%).

ci_method
  The type of index used for Credible Interval. Can be "HDI" (default, see hdi),
  "ETI" (see eti), "BCI" (see bci) 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.
verbose  Toggle off warnings.
...  Additional arguments to be passed to or from methods.

Details

The meaning of the extracted parameters:

- For \texttt{ttestBF}: Difference is the raw difference between the means.
- For \texttt{correlationBF}: rho is the linear correlation estimate (equivalent to Pearson’s r).
- For \texttt{lmBF / generalTestBF / regressionBF / anovaBF}: in addition to parameters of the fixed and random effects, there are: \textit{mu} is the (mean-centered) intercept; \textit{sig2} is the model’s sigma; \textit{g / g_*} are the \textit{g} parameters; See the \textit{Bayes Factors for ANOVAs} paper (doi: 10.1016/j.jmp.2012.08.001).

Value

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

Examples

```r
if (require("BayesFactor")) {
  model <- ttestBF(x = rnorm(100, 1, 1))
  model_parameters(model)
}
```

---

\textbf{model\_parameters\_cgam}  \textit{Parameters from Generalized Additive (Mixed) Models}

\textbf{Description}

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

\textbf{Usage}

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

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

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

### 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", "pseudo" or NULL (default) for no standardization. See 'De-
tails' in `standardize_parameters`. **Important:** Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use `standardize="basic"` to mimic behaviour of SPSS or packages such as `lm.beta`, or standardize the data with `effectsize::standardize(force=TRUE)` before fitting the model. 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 logistic regression, or more generally speaking, for models with log or logit links. **Note:** Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For `compare_parameters()`, `exponentiate = "nongaussian"` will only exponentiate coefficients from non-Gaussian families.

**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 and this vignette for working examples.

**p_adjust** Character vector, if not `NULL`, indicates the method to adjust p-values. See `p.adjust` for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for `emmGrid` objects (from `emmeans`).

**parameters** Character vector of length 1 with a regular expression pattern that describes the parameters that should be returned from the data frame, or a named list of regular expressions. All non-matching parameters will be removed from the output. If `parameters` is a character vector, every parameter in the "Parameters" column that matches the regular expression in `parameters` will be selected from the returned data frame. Furthermore, if `parameters` has more than one element, these will be merged into a regular expression pattern like this: "(one|two|three)". If `parameters` is a named list of regular expression patterns, the names of the list-element should equal the column name where selection should be applied. This is useful for model objects where `model_parameters()` returns multiple columns with parameter components, like in `model_parameters.lavaan`.

**verbose** Toggle warnings and messages.

**...** Arguments passed to or from other methods. For instance, when `bootstrap = TRUE`, arguments like `type` or `parallel` are passed down to `bootstrap_model()`, and arguments like `ci_method` are passed down to `describe_posterior`.

**Details**

The reporting of degrees of freedom for the spline terms slightly differs from the output of `summary(model)`, for example in the case of `mgcv::gam()`. The estimated degrees of freedom, column `edf` in the
summary-output, is named df in the returned data frame, while the column df_error in the returned data frame refers to the residual degrees of freedom that are returned by df.residual(). Hence, the values in the the column df_error differ from the column Ref.df from the summary, which is intentional, as these reference degrees of freedom “is not very interpretable” (web).

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)
}

---

Parameters from Mixed Models

Description

Parameters from (linear) mixed models.

Usage

## S3 method for class 'cpglmml'

model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  df_method = NULL,
  p_adjust = NULL,
  verbose = TRUE,
  ...
)
## S3 method for class 'glmmTMB'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  effects = "all",
  component = "all",
  group_level = FALSE,
  standardize = NULL,
  exponentiate = FALSE,
  df_method = NULL,
  p_adjust = NULL,
  wb_component = TRUE,
  summary = FALSE,
  parameters = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'merMod'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  df_method = "wald",
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  robust = FALSE,
  p_adjust = NULL,
  wb_component = TRUE,
  summary = FALSE,
  parameters = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'mixor'
model_parameters(
  model,
  ci = 0.95,
  effects = "all",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
Arguments

model A mixed model.

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 draws to simulate/bootstrap.

standardize The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic", "pseudo" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Important: Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use standardize="basic" to mimic behaviour of SPSS or packages such as lm.beta, or standardize the data with effectsize::standardize(force=TRUE) before fitting the model. Robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".

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

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.

exponentiate Logical, indicating whether or not to exponentiate the the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. Note: Delta-method standard
errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = "nongaussian" will only exponentiate coefficients from non-Gaussian families.

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). The options df_method = "boot", df_method = "profile" and df_method = "uniroot" only affect confidence intervals; in this case, bootstrapped resp. profiled confidence intervals are computed. "uniroot" only applies to models of class glmmTMB. For models of class lmerMod, when df_method = "wald", residual degrees of freedom are returned. Note that when df_method is not "wald", robust standard errors etc. cannot be computed.

p_adjust Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from emmeans).

verbose Toggle warnings and messages.

... 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", "dispersion" or "all" (default). May be abbreviated.

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.

summary Logical, if TRUE, prints summary information about the model (model formula, number of observations, residual standard deviation and more).

parameters Character vector of length 1 with a regular expression pattern that describes the parameters that should be returned from the data frame, or a named list of regular expressions. All non-matching parameters will be removed from the output. If parameters is a character vector, every parameter in the "Parameters" column that matches the regular expression in parameters will be selected from the returned data frame. Furthermore, if parameters has more than one element, these will be merged into a regular expression pattern like this: "(one|two|three)". If parameters is a named list of regular expression patterns, the names of the list-element should equal the column name where selection should be applied. This is useful for model objects where model_parameters() returns multiple columns with parameter components, like in model_parameters.lavaan.

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 and this vignette for working examples.

**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**

```r
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, effects = "all")
}

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

```r
## S3 method for class 'data.frame'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.89,
  ci_method = "hdi",
  test = c("pd", "rope"),
  rope_range = "default",
  rope_ci = 1,
  parameters = NULL,
  verbose = TRUE,
  ...
)
```

```r
## 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 = FALSE,
  effects = "fixed",
  component = "all",
  exponentiate = FALSE,
  standardize = NULL,
  group_level = FALSE,
  parameters = NULL,
  verbose = TRUE,
  ...
)
```

```r
## 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",
  ...
)
```
model_parameters.data.frame

rope_ci = 1,
bf_prior = NULL,
diagnostic = c("ESS", "Rhat"),
priors = TRUE,
effects = "fixed",
exponentiate = FALSE,
standardize = NULL,
group_level = FALSE,
parameters = NULL,
verbose = TRUE,
... 
)

Arguments

model
  Bayesian model (including SEM from blavaan. 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), "BCI" (see bci) 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.

parameters
  Character vector of length 1 with a regular expression pattern that describes the parameters that should be returned from the data frame, or a named list of regular expressions. All non-matching parameters will be removed from the output. If parameters is a character vector, every parameter in the "Parameters" column that matches the regular expression in parameters will be selected from the returned data frame. Furthermore, if parameters has more than one element, these will be merged into a regular expression pattern like this: "(one|two|three)". If parameters is a named list of regular expression patterns, the names of the list-element should equal the column name where selection should be applied. This is useful for model objects where model_parameters() returns multiple columns with parameter components, like in model_parameters.lavaan.

verbose
  Toggle messages and warnings.

... 
  Currently not used.
### 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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bf_prior</td>
<td>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.</td>
</tr>
<tr>
<td>diagnostic</td>
<td>Diagnostic metrics to compute. Character (vector) or list with one or more of these options: &quot;ESS&quot;, &quot;Rhat&quot;, &quot;MCSE&quot; or &quot;all&quot;.</td>
</tr>
<tr>
<td>priors</td>
<td>Add the prior used for each parameter.</td>
</tr>
<tr>
<td>effects</td>
<td>Should results for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.</td>
</tr>
<tr>
<td>component</td>
<td>Model component for which parameters should be shown. May be one of &quot;conditional&quot;, &quot;precision&quot; (betareg), &quot;scale&quot; (ordinal), &quot;extra&quot; (glimx), &quot;marginal&quot; (mfx), &quot;conditional&quot; or &quot;full&quot; (for MuMin::model.avg()) or &quot;all&quot;.</td>
</tr>
<tr>
<td>exponentiate</td>
<td>Logical, indicating whether or not to exponentiate the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. Note: Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = &quot;nongaussian&quot; will only exponentiate coefficients from non-Gaussian families.</td>
</tr>
<tr>
<td>standardize</td>
<td>The method used for standardizing the parameters. Can be &quot;refit&quot;, &quot;posthoc&quot;, &quot;smart&quot;, &quot;basic&quot;, &quot;pseudo&quot; or NULL (default) for no standardization. See 'Details' in standardize_parameters. Important: Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use standardize=&quot;basic&quot; to mimic behaviour of SPSS or packages such as lm.beta, or standardize the data with effectsize::standardize(force=TRUE) before fitting the model. Robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize=&quot;refit&quot;.</td>
</tr>
<tr>
<td>group_level</td>
<td>Logical, for multilevel models (i.e. models with random effects) and when effects = &quot;all&quot; or effects = &quot;random&quot;, include the parameters for each group level from random effects. If group_level = FALSE (the default), only information on SD and COR are shown.</td>
</tr>
</tbody>
</table>
See Also

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

Examples

```r
## Not run:
library(parameters)
if (require("rstanarm")) {
  model <- stan_glm(
    Sepal.Length ~ Petal.Length * Species,
    data = iris, iter = 500, refresh = 0
  )
  model_parameters(model)
}
## End(Not run)
```

---

**model_parameters.default**

*Parameters from (General) Linear Models*

**Description**

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

**Usage**

```r
## Default S3 method:
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  robust = FALSE,
  p_adjust = NULL,
  summary = FALSE,
  parameters = NULL,
  verbose = TRUE,
  ...
)

## 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,
summary = FALSE,
verbose = TRUE,
...)

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

## 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,
  verbose = TRUE,
  ...
)

## S3 method for class 'betamfx'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "precision", "marginal"),
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", "pseudo" or NULL (default) for no standardization. See 'Details' in standardize_parameters. **Important**: Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use standardize="basic" to mimic behaviour of SPSS or packages such as lm.beta, or standardize the data with effectsize::standardize(force=TRUE) before fitting the model. 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 logistic regression, or more generally speaking, for models with log or logit links. **Note**: Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = "nongaussian" will only exponentiate coefficients from non-Gaussian families.
- **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 and this vignette for working examples.
- **p_adjust**: Character vector, if not NULL, indicates the method to adjust p-values. See p_adjust for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from emmeans).
- **summary**: Logical, if TRUE, prints summary information about the model (model formula, number of observations, residual standard deviation and more).
parameters  Character vector of length 1 with a regular expression pattern that describes the parameters that should be returned from the data frame, or a named list of regular expressions. All non-matching parameters will be removed from the output. If parameters is a character vector, every parameter in the "Parameters" column that matches the regular expression in parameters will be selected from the returned data frame. Furthermore, if parameters has more than one element, these will be merged into a regular expression pattern like this: "(one|two|three)". If parameters is a named list of regular expression patterns, the names of the list-element should equal the column name where selection should be applied. This is useful for model objects where model_parameters() returns multiple columns with parameter components, like in model_parameters.lavaan.

verbose  Toggle warnings and messages.

...  Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like type or parallel are passed down to bootstrap_model(), and arguments like ci_method are passed down to describe_posterior.

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".

component  Model component for which parameters should be shown. May be one of "conditional", "precision" (betareg), "scale" (ordinal), "extra" (glmx), "marginal" (mfx), "conditional" or "full" (for MuMIn::model.avg()) 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)
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 / exponeniated coefficients
model_parameters(model, exponentiate = TRUE)

---

model_parameters.DirichletRegModel

Parameters from multinomial or cumulative link models

**Description**

Parameters from multinomial or cumulative link models

**Usage**

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

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

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

component

Model component for which parameters should be shown. May be one of "conditional", "precision" (betareg), "scale" (ordinal), "extra" (glmx), "marginal" (mfx), "conditional" or "full" (for MuMIn::model.avg()) or "all".

standardize

The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic", "pseudo" or NULL (default) for no standardization. See 'Details' in standardize_parameters. **Important:** Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use standardize="basic" to mimic behaviour of SPSS or packages such as lm.beta, or standardize the data with effectsize::standardize(force=TRUE) before fitting the model. Robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".

exponentiate

Logical, indicating whether or not to exponentiate the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. **Note:** Delta-method standard errors are also computed (by multiplying the standard errors by the transformed
coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = "nongaussian" will only exponentiate coefficients from non-Gaussian families.

verbose Toggle warnings and messages.

... Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like type or parallel are passed down to bootstrap_model(), and arguments like ci_method are passed down to describe_posterior.

p_adjust Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from emmeans).

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", quietly = TRUE)) {
  data("stemcell")
  model <- bracl(    research ~ as.numeric(religion) + gender,
    weights = frequency,
    data = stemcell,
    type = "ML"
  )
  model_parameters(model)
}
### model_parameters.htest

Parameters from hypothesis tests

#### Description

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

#### Usage

```r
## S3 method for class 'htest'
model_parameters(
  model,
  cramers_v = NULL,
  phi = NULL,
  standardized_d = NULL,
  hedges_g = NULL,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  cohens_g = NULL,
  rank_biserial = NULL,
  rank_epsilon_squared = NULL,
  kendalls_w = NULL,
  ci = 0.95,
  bootstrap = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'pairwise.htest'
model_parameters(model, verbose = TRUE, ...)
```

#### Arguments

- **model**: Object of class `htest` or `pairwise.htest`.
- **cramers_v, phi**: Compute Cramer’s V or phi as index of effect size. Can be "raw" or "adjusted" (effect size will be bias-corrected). Only applies to objects from `chisq.test()`.
- **standardized_d**: If TRUE, compute standardized d as index of effect size. Only applies to objects from `t.test()`. Calculation of d is based on the t-value (see `t_to_d`) for details.
- **hedges_g**: If TRUE, compute Hedge’s g as index of effect size. Only applies to objects from `t.test()`.
- **omega_squared, eta_squared, epsilon_squared**: Logical, if TRUE, returns the non-partial effect size Omega, Eta or Epsilon squared. Only applies to objects from `oneway.test()`.
model_parameters.htest

cohens_g If TRUE, compute Cohen's g as index of effect size. Only applies to objects from mcnemar.test().
rank_biserial If TRUE, compute the rank-biserial correlation as effect size measure. Only applies to objects from wilcox.test().
rank_epsilon_squared If TRUE, compute the rank epsilon squared as effect size measure. Only applies to objects from kruskal.test().
kendalls_w If TRUE, compute the Kendall's coefficient of concordance as effect size measure. Only applies to objects from friedman.test().
parent Any arguments passed to or from other methods.

Value

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

Examples

```r
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 <- pairwise.t.test(airquality$Ozone, airquality$Month)
model_parameters(model)

smokers <- c(83, 90, 129, 70)
patients <- c(86, 93, 136, 82)
model <- pairwise.prop.test(smokers, patients)
model_parameters(model)
```
model_parameters.kmeans

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

Description
Format cluster models obtained for example by \texttt{kmeans}.

Usage

\begin{verbatim}
## S3 method for class 'kmeans'
model_parameters(model, verbose = TRUE, ...)
\end{verbatim}

Arguments

- \texttt{model}: Cluster model.
- \texttt{verbose}: Toggle warnings and messages.
- \texttt{...}: Arguments passed to or from other methods.

Examples

\begin{verbatim}
library(parameters)
model <- kmeans(iris[1:4], centers = 3)
model_parameters(model)
\end{verbatim}

model_parameters.lavaan

Parameters from CFA/SEM models

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

Usage

\begin{verbatim}
## S3 method for class 'lavaan'
model_parameters(
  model,
  ci = 0.95,
  standardize = FALSE,
  component = c("regression", "correlation", "loading", "defined"),
  parameters = NULL,
  verbose = TRUE,
  ...)
\end{verbatim}
Arguments

- **model**: CFA or SEM created by the `lavaan::cfa` or `lavaan::sem` functions.
- **ci**: Confidence Interval (CI) level. Default to 0.95 (95%).
- **standardize**: 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.
- **component**: What type of links to return. Can be "all" or some of c("regression","correlation","loading","variance","mean").
- **parameters**: Character vector of length 1 with a regular expression pattern that describes the parameters that should be returned from the data frame, or a named list of regular expressions. All non-matching parameters will be removed from the output. If parameters is a character vector, every parameter in the "Parameters" column that matches the regular expression in parameters will be selected from the returned data frame. Furthermore, if parameters has more than one element, these will be merged into a regular expression pattern like this: "(one|two|three)". If parameters is a named list of regular expression patterns, the names of the list-element should equal the column name where selection should be applied. This is useful for model objects where `model_parameters()` returns multiple columns with parameter components, like in `model_parameters.lavaan`.
- **verbose**: Toggle warnings and messages.
- **...**: 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


Examples

```r
library(parameters)

# lavaan -------------------------------------
if (require("lavaan", quietly = TRUE)) {
```
# 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)

# filter parameters
model_parameters(
  model,
  parameters = list(
    To = "^(?!visual)",
    From = "^(?!((x7|x8))"
  )
)

# 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.Mclust

Parameters from Mixture Models

Description

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

## S3 method for class 'Mclust'
model_parameters(model, verbose = TRUE, ...)

Arguments

- **model**: Mixture model.
- **verbose**: Toggle warnings and messages.
- **...**: Arguments passed to or from other methods.

Examples

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

Description

Parameters from multiply imputed repeated analyses

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

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 logistic regression, or more generally speaking, for models with log or logit links. **Note**: Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as
Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For `compare_parameters()`, `exponentiate = "nongaussian"` will only exponentiate coefficients from non-Gaussian families.

**p_adjust**
Character vector, if not NULL, indicates the method to adjust p-values. See `p.adjust` for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for `emmean` objects (from `emmeans`).

**verbose**
Toggle warnings and messages.

... Arguments passed to or from other methods.

**Details**

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

**Examples**

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

Format structural models from the psych or FactoMineR packages.

Usage

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

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

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

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.
- **verbose**: Toggle warnings and messages.
- **...**: 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


Examples

```r
library(parameters)
if (require("psych", quietly = TRUE)) {
  # 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)))
}
```r
# Omega -------
omega <- psych::omega(mtcars, nfactors = 3)
params <- model_parameters(omega)
summary(params)
}

# FactoMineR -------
if (require("FactoMineR", quietly = TRUE)) {
  model <- FactoMineR::PCA(iris[, 1:4], ncp = 2)
  model_parameters(model)
  attributes(model_parameters(model))$scores
  model <- FactoMineR::FAMD(iris, ncp = 2)
  model_parameters(model)
}
```

---

**model_parameters.PMCMR**

*Parameters from Hypothesis Testing*

**Description**

Parameters from Hypothesis Testing.

**Usage**

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

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

**Arguments**

- `model` Object of class `glht` (**multcomp**) or of class `PMCMR`, `trendPMCMR` or `osrt` (**PMCMRplus**).
- `...` Arguments passed to or from other methods. For instance, when `bootstrap = TRUE`, arguments like `type` or `parallel` are passed down to `bootstrap_model()`, and arguments like `ci_method` are passed down to `describe_posterior`.
- `ci` Confidence Interval (CI) level. Default to 0.95 (95%).
- `exponentiate` Logical, indicating whether or not to exponentiate the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. **Note:** Delta-method standard
errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For `compare_parameters()`, `exponentiate = "nongaussian"` will only exponentiate coefficients from non-Gaussian families.

verbose Toggle warnings and messages.

Value

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

Examples

```r
if (require("multcomp", quietly = TRUE)) {
  # 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)
}
if (require("PMCMRplus", quietly = TRUE)) {
  model <- kwAllPairsConoverTest(count ~ spray, data = InsectSprays)
  model_parameters(model)
}
```

model_parameters.rma  Parameters from Meta-Analysis

Description

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

Usage

```r
## S3 method for class 'rma'
model_parameters(
  model,
  ci = 0.95,
)
model_parameters.rma

bootstrap = FALSE,
iterations = 1000,
standardize = NULL,
exponentiate = FALSE,
include_studies = TRUE,
verbose = TRUE,
...)

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", "pseudo" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Important: Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use standardize="basic" to mimic behaviour of SPSS or packages such as lm.beta, or standardize the data with effectsize::standardize(force=TRUE) before fitting the model. Robust estimation (i.e. robust=TRUE) of standardized parameters only works when standardize="refit".

exponentiate Logical, indicating whether or not to exponentiate the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. Note: Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = "nongaussian" will only exponentiate coefficients from non-Gaussian families.

include_studies Logical, if TRUE (default), includes parameters for all studies. Else, only parameters for overall-effects are shown.

verbose Toggle warnings and messages.

... Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like type or parallel are passed down to bootstrap_model(), and 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", quietly = TRUE)) {
  model <- rma(yi = effectsize, sei = stderr, method = "REML", data = mydat)
  model_parameters(model)
}
## Not run:
# with subgroups
if (require("metafor", quietly = TRUE)) {
  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)
}
if (require("metaBMA", quietly = TRUE)) {
  data(towels)
  m <- meta_random(logOR, SE, study, data = towels)
  model_parameters(m)
}
## End(Not run)

model_parameters.t1way

Parameters from robust statistical objects in WRS2

Description

Parameters from robust statistical objects in WRS2

Usage

## S3 method for class 't1way'
model_parameters(model, verbose = TRUE, ...)

---

model_parameters.t1way

Parameters from robust statistical objects in WRS2

Description

Parameters from robust statistical objects in WRS2

Usage

## S3 method for class 't1way'
model_parameters(model, verbose = TRUE, ...)
model_parameters.zcpglm

Arguments

- **model**: Object from `WRS2` package.
- **verbose**: Toggle warnings and messages.
- **...**: Arguments passed to or from other methods.

Value

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

Examples

```r
if (require("WRS2")) {
  model <- t1way(libido ~ dose, data = viagra)
  model_parameters(model)
}
```

---

**model_parameters.zcpglm**

Parameters from Zero-Inflated Models

Description

Parameters from zero-inflated models (from packages like `pscl.cplm` or `countreg`).

Usage

```r
## S3 method for class 'zcpglm'
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,
  parameters = NULL,
  verbose = TRUE,
  ...
)
```
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), "conditional" or "full" (for MuMIn::model.avg()) or "all".

.standardize  The method used for standardizing the parameters. Can be "refit", "posthoc", "smart", "basic", "pseudo" or NULL (default) for no standardization. See 'Details' in standardize_parameters. Important: Categorical predictors (i.e. factors) are never standardized by default, which may be a different behaviour compared to other R packages or other software packages (like SPSS). If standardizing categorical predictors is desired, either use standardize="basic" to mimic behaviour of SPSS or packages such as lm.beta, or standardize the data with effectsize::standardize(force=TRUE) before fitting the model. 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 logistic regression, or more generally speaking, for models with log or logit links. Note: Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For compare_parameters(), exponentiate = "nongaussian" will only exponentiate coefficients from non-Gaussian families.

.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 and this vignette for working examples.

.p_adjust  Character vector, if not NULL, indicates the method to adjust p-values. See p.adjust for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from emmeans).

.parameters  Character vector of length 1 with a regular expression pattern that describes the parameters that should be returned from the data frame, or a named list of regular expressions. All non-matching parameters will be removed from the output. If parameters is a character vector, every parameter in the "Parameters" column that matches the regular expression in parameters will be selected from the returned data frame. Furthermore, if parameters has more than
one element, these will be merged into a regular expression pattern like this: 
“(one|two|three)”. If parameters is a named list of regular expression patterns, the names of the list-element should equal the column name where selection should be applied. This is useful for model objects where \texttt{model_parameters()} returns multiple columns with parameter components, like in \texttt{model_parameters.lavaan}.

\begin{itemize}
  \item \texttt{verbose} \quad Toggle warnings and messages.
  \item \texttt{...} \quad Arguments passed to or from other methods. For instance, when \texttt{bootstrap = TRUE}, arguments like \texttt{type} or \texttt{parallel} are passed down to \texttt{bootstrap_model()}, and arguments like \texttt{ci_method} are passed down to \texttt{describe_posterior}.
\end{itemize}

\textbf{Value}

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

\textbf{See Also}

\texttt{standardize_names()} to rename columns into a consistent, standardized naming scheme.

\textbf{Examples}

\begin{verbatim}
library(parameters)
if (require("pscl")) {
  data("bioChemists")
  model <- zeroinfl(art ~ fem + mar + kid5 + ment | kid5 + phd, data = bioChemists)
  model_parameters(model)
}
\end{verbatim}

\begin{table}[H]
\centering
\begin{tabular}{ll}
\hline
\textbf{n_clusters} & \textit{Number of clusters to extract} \\
\hline
\end{tabular}
\end{table}

\textbf{Description}

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

\textbf{Usage}

\begin{verbatim}
n_clusters(
  x, 
  standardize = TRUE, 
  force = FALSE, 
  package = c("NbClust", "mclust", "cluster", "M3C"), 
  fast = TRUE, 
  ... 
)
\end{verbatim}
n_factors

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  Package from which methods are to be called 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)

if (require("mclust", quietly = TRUE) && require("NbClust", quietly = TRUE) && require("cluster", quietly = TRUE)) {
  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/extract from 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 model and select the solution with the fewer factors.

Usage

n_factors(
  x,
  type = "FA",
  rotation = "varimax",
  algorithm = "default",
  package = c("nFactors", "psych"),
)
Arguments

\textbf{x} \hspace{1cm} \text{A data frame.}

\textbf{type} \hspace{1cm} \text{Can be "FA" or "PCA", depending on what you want to do.}

\textbf{rotation} \hspace{1cm} \text{Only used for VSS (Very Simple Structure criterion, see \texttt{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.}

\textbf{algorithm} \hspace{1cm} \text{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".}

\textbf{package} \hspace{1cm} \text{Package from which respective 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, \texttt{c("nFactors","psych")} are selected.}

\textbf{cor} \hspace{1cm} \text{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 \texttt{cor()} on the passed data.}

\textbf{safe} \hspace{1cm} \text{If TRUE, the function will run all the procedures in try blocks, and will only return those that work and silently skip the ones that may fail.}

\ldots \hspace{1cm} \text{Arguments passed to or from other methods.}

Details

\texttt{n_components} is actually an alias for \texttt{n_factors}, with different defaults for the function arguments.

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


Examples

```r
library(parameters)
if (require("nFactors", quietly = TRUE) & require("EGAnet", quietly = TRUE)) {
  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")

parameters_type <- function(model, ...) {
  parameters_type(model)
}

parameters_type <- function(model, ...) {
  A data frame.
}

## 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)

---

**pool_parameters**

**Pool Model Parameters**

**Description**

This function "pools" (i.e. combines) model parameters in a similar fashion as `mice::pool()`. However, this function pools parameters from `parameters_model` objects, as returned by `model_parameters`.

**Usage**

```r
pool_parameters(
  x,
  exponentiate = FALSE,
  effects = "fixed",
  component = "conditional",
  verbose = TRUE,
  ...
)
```

**Arguments**

- `x` A list of `parameters_model` objects, as returned by `model_parameters`, or a list of model-objects that is supported by `model_parameters()`.
- `exponentiate` Logical, indicating whether or not to exponentiate the coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. **Note:** Delta-method standard errors are also computed (by multiplying the standard errors by the transformed coefficients). This is to mimic behaviour of other software packages, such as Stata, but these standard errors poorly estimate uncertainty for the transformed coefficient. The transformed confidence interval more clearly captures this uncertainty. For `compare_parameters()`, `exponentiate = "nongaussian"` will only exponentiate coefficients from non-Gaussian families.
- `effects` Should parameters for fixed effects ("fixed"), random effects ("random"), or both ("all") be returned? Only applies to mixed models. May be abbreviated.
component Model component for which parameters should be shown. May be one of "conditional", "precision" (betareg), "scale" (ordinal), "extra" (glmx), "marginal" (mfx), "conditional" or "full" (for MuMIn::model.avg()) or "all".

verbose Toggle warnings and messages.

... Currently not used.

Details

Averaging of parameters follows Rubin’s rules (Rubin, 1987, p. 76). The pooled degrees of freedom is based on the Barnard-Rubin adjustment for small samples (Barnard and Rubin, 1999).

Value

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

Note

Models with multiple components, (for instance, models with zero-inflation, where predictors appear in the count and zero-inflated part) may fail in case of identical names for coefficients in the different model components, since the coefficient table is grouped by coefficient names for pooling. In such cases, coefficients of count and zero-inflated model parts would be combined. Therefore, the component argument defaults to "conditional" to avoid this.

References


Examples

```r
# example for multiple imputed datasets
if (require("mice")) {
  data("nhanes2")
  imp <- mice(nhanes2, printFlag = FALSE)
  models <- lapply(1:5, function(i) {
    lm(bmi ~ age + hyp + chl, data = complete(imp, action = i))
  })
  pool_parameters(models)

  # should be identical to:
m <- with(data = imp, exp = lm(bmi ~ age + hyp + chl))
  summary(pool(m))
}
```
print.parameters_model

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,
  caption = NULL,
  digits = 2,
  ci_digits = 2,
  p_digits = 3,
  footer_digits = 3,
  show_sigma = FALSE,
  show_formula = FALSE,
  zap_small = FALSE,
  groups = NULL,
  ...
)

## S3 method for class 'parameters_model'
summary(object, ...)

Arguments

x, object  An object returned by model_parameters().
pretty_names  Return "pretty" (i.e. more human readable) parameter 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.
caption  Table caption as string. If NULL, no table caption is printed.
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.

footer_digits Number of decimal places for values in the footer summary.

show_sigma Logical, if TRUE, adds information about the residual standard deviation.

show_formula Logical, if TRUE, adds the model formula to the output.

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

groups Named list, can be used to group parameters in the printed output. List elements may either be character vectors that match the name of those parameters that belong to one group, or list elements can be row numbers of those parameter rows that should belong to one group. The names of the list elements will be used as group names, which will be inserted as "header row". A possible use case might be to emphasize focal predictors and control variables, see 'Examples'. Parameters will be re-ordered according to the order used in groups, while all non-matching parameters will be added to the end.

... Arguments passed to or from other methods.

Details

summary() is a convenient shortcut for print(object, select = "minimal", show_sigma = TRUE, show_formula = TRUE).

Value

Invisibly returns the original input object.

Interpretation of Interaction Terms

Note that the interpretation of interaction terms depends on many characteristics of the model. The number of parameters, and overall performance of the model, can differ or not between a * b a : b, and a / b, suggesting that sometimes interaction terms give different parameterizations of the same model, but other times it gives completely different models (depending on a or b being factors of covariates, included as main effects or not, etc.). Their interpretation depends of the full context of the model, which should not be inferred from the parameters table alone - rather, we recommend to use packages that calculate estimated marginal means or marginal effects, such as modelbased, emmeans or ggeffects. To raise awareness for this issue, you may use print(..., show_formula=TRUE) to add the model-specification to the output of the print() method for model_parameters().

Labeling the Degrees of Freedom

Throughout the parameters package, we decided to label the residual degrees of freedom df_error. The reason for this is that these degrees of freedom not always refer to the residuals. For certain
models, they refer to the estimate error - in a linear model these are the same, but in - for instance - any mixed effects model, this isn’t strictly true. Hence, we think that df_error is the most generic label for these degrees of freedom.

See Also

There is a dedicated method to use inside rmarkdown files, print_md().

Examples

```r
library(parameters)
if (require("glmmTMB", quietly = TRUE)) {
  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")
}

# group parameters -------
data(iris)
model <- lm(
  Sepal.Width ~ Sepal.Length + Species + Petal.Length,
  data = iris
)
# don't select "Intercept" parameter
mp <- model_parameters(model, parameters = "^(?!\(Intercept)\)")
groups <- list(
  "Focal Predictors" = c("Speciesversicolor", "Speciesvirginica"),
  "Controls" = c("Sepal.Length", "Petal.Length")
)
print(mp, groups = groups)

# or use row indices
print(mp, groups = list("Focal Predictors" = c(1, 4),
  "Controls" = c(2, 3)))

# only show coefficients, CI and p,
# put non-matched parameters to the end
```
p_value

data(mtcars)
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)
model <- lm(mpg ~ hp + gear * vs + cyl + drat, data = mtcars)

# don't select "Intercept" parameter
mp <- model_parameters(model, parameters = "^\?!\!(Intercept)"
print(mp, groups = list("Engine" = c("cyl6", "cyl8", "vs", "hp"),
    "Interactions" = c("gear4:vs", "gear5:vs")))

Description

This function attempts to return, or compute, p-values of a model’s parameters. See the documentation for your object’s class:

- Mixed models (lme4, nlme, glmmTMB, ...)
- Bayesian models (rstanarm, brms, MCMCglmm, ...)
- Zero-inflated models (hurdle, zeroInfl, zerocount, ...)
- Marginal effects models (mfx)
- Models with special components (DirichletRegModel, clm2, cgam, ...)

Usage

p_value(model, ...)

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

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

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. Only available for models where method = "robust" is supported.
- **method** If "robust", and if model is supported by the sandwich or clubSandwich packages, computes p-values based on robust covariance matrix estimation.
- **verbose** Toggle warnings and messages.
- **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

A data frame with at least two columns: the parameter names and the p-values. Depending on the model, may also include columns for model components etc.

Note

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

Examples

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

---

\texttt{p_value.BFBayesFactor} \textit{p-values for Bayesian Models}

Description

This function attempts to return, or compute, p-values of Bayesian models.

Usage

\texttt{## S3 method for class 'BFBayesFactor'}
\texttt{p_value(model, \ldots)}

Arguments

\texttt{model} \quad \texttt{A statistical model.}

\texttt{\ldots} \quad \texttt{Arguments passed down to \texttt{standard_error.robust()} when confidence intervals or p-values based on robust standard errors should be computed. Only available for models where \texttt{method = "robust"} is supported.}

Details

For Bayesian models, the p-values corresponds to the \textit{probability of direction (p\_direction)}, which is converted to a p-value using \texttt{bayestestR::convert\_pd\_to\_p()}.

Value

The p-values.

Examples

\texttt{data(iris)}
\texttt{model <- lm(Petal.Length ~ Sepal.Length + Species, data = iris)}
\texttt{p_value(model)}
p_value.cpglmm  p-values for Mixed Models

Description
This function attempts to return, or compute, p-values of mixed models.

Usage

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

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

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

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

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

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

Arguments

model  A statistical model.
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 on robust standard errors.

...  Arguments passed down to standard_error_robust() when confidence intervals or p-values based on robust standard errors should be computed. Only available for models where method = "robust" is supported.
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", "dispersion" or "all" (default). May be abbreviated.

verbose: Toggle warnings and messages.

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.

Details

By default, p-values are based on Wald-test approximations (see \code{p_value_wald}). For certain situations, the "ml1-1" rule might be a better approximation. That is, for \code{method = "ml1"}, \code{p_value_ml1} is called. For \code{lmerMod} objects, if \code{method = "kenward"}, p-values are based on Kenward-Roger approximations, i.e. \code{p_value_kenward} is called, and \code{method = "satterthwaite"} calls \code{p_value_satterthwaite}.

Value

A data frame with at least two columns: the parameter names and the p-values. Depending on the model, may also include columns for model components etc.

Note

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

Examples

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

p_value.DirichletRegModel

\emph{p-values for Models with Special Components}

Description

This function attempts to return, or compute, p-values of models with special model components.
**Usage**

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

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

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

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

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

**Arguments**

- `model`: A statistical model.
- `component`: Should all parameters, parameters for the conditional model, precision- or scale-component or smooth_terms be returned? component may be one of "conditional", "precision", "scale", "smooth_terms", "full" or "all" (default).
- `...`: Arguments passed down to `standard_error_robust()` when confidence intervals or p-values based on robust standard errors should be computed. Only available for models where `method = "robust"` is supported.

**Value**

The p-values.

---

**Description**

This function attempts to return, or compute, p-values of marginal effects models from package `mfx`.

**Usage**

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

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

## Arguments

- **model**: A statistical model.
- **component**: Should all parameters, parameters for the conditional model, precision-component or marginal effects be returned? `component` may be one of "conditional", "precision", "marginal" or "all" (default).
- **...**: Currently not used.

## Value

A data frame with at least two columns: the parameter names and the p-values. Depending on the model, may also include columns for model components etc.

## Examples

```r
if (require("mfx", quietly = TRUE)) {
  set.seed(12345)
  n <- 1000
  x <- rnorm(n)
  y <- rnegbin(n, mu = exp(1 + 0.5 * x), theta = 0.5)
  d <- data.frame(y, x)
  model <- poissonmfx(y ~ x, data = d)

  p_value(model)
  p_value(model, component = "marginal")
}
```

---

## Description

This function attempts to return, or compute, p-values of hurdle and zero-inflated models.

## Usage

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

# S3 method for class 'zeroinfl'
p_value(
```

---

## $p$-values for Models with Zero-Inflation

This function attempts to return, or compute, p-values of hurdle and zero-inflated models.
Arguments

model A statistical model.
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", "dispersion" or "all" (default). May be abbreviated.

... Arguments passed down to standard_error_robust() when confidence intervals or p-values based on robust standard errors should be computed. Only available for models where method = "robust" is supported.

method If "robust", and if model is supported by the sandwich or clubSandwich packages, computes p-values based on robust covariance matrix estimation.

verbose Toggle warnings and messages.

Value

A data frame with at least two columns: the parameter names and the p-values. Depending on the model, may also include columns for model components etc.

Examples

if (require("pscl", quietly = TRUE)) {
  data("bioChemists")
  model <- zeroinfl(art ~ fem + mar + kid5 | kid5 + phd, data = bioChemists)
  p_value(model)
  p_value(model, component = "zi")
}

qol_cancer

Sample data set

Description

A sample data set with longitudinal data, used in the vignette describing the demean() function. Health-related quality of life from cancer-patients was measured at three time points (pre-surgery, 6 and 12 months after surgery).
random_parameters

Format
A data frame with 564 rows and 7 variables:

<table>
<thead>
<tr>
<th>ID</th>
<th>Patient ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoL</td>
<td>Quality of Life Score</td>
</tr>
<tr>
<td>time</td>
<td>Timepoint of measurement</td>
</tr>
<tr>
<td>age</td>
<td>Age in years</td>
</tr>
<tr>
<td>phq4</td>
<td>Patients’ Health Questionnaire, 4-item version</td>
</tr>
<tr>
<td>hospital</td>
<td>Hospital ID, where patient was treated</td>
</tr>
<tr>
<td>education</td>
<td>Patients’ educational level</td>
</tr>
</tbody>
</table>

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, component = "conditional")

Arguments

| model | A mixed effects model (including stanreg models). |
| component | Should all parameters, parameters for the conditional model, or for the zero-inflated part of the model be returned? Applies to models with zero-inflated component. component may be one of "conditional" (default), "zi" or "zero-inflated". May be abbreviated. |

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

**Within-group (or residual) variance**: The residual variance, $\sigma^2$, 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 ($\tau_{00}$), 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 ($\tau_{11}$) 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 ($\rho_{01}$) 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
```r
if (require("lme4")) {
  data(sleepstudy)
  model <- lmer(Reaction ~ Days + (1 + Days | Subject), data = sleepstudy)
  random_parameters(model)
}
```

Description
This function performs a reduction in the parameter space (the number of variables). It starts by creating a new set of variables, based on the 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 feature reduction method. Can be one of 'PCA', 'cMDS', 'DRR', 'ICA' (see the Details section).
reduce_parameters

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 resp. n_components. 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 DRR are avoidance of multicollinearity between predictors and overfitting mitigation. DRR tends to perform well when the first principal component is 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, which 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


Examples

data(iris)
model <- lm(Sepal.Width ~ Species * Sepal.Length + Petal.Width, data = iris)
model
reduce_parameters(model)
```r
out <- reduce_data(iris, method = "PCA", n = "max")
head(out)
```

---

**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**

```r
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**


**Examples**

```r
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-effects 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

```r
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)
}

## Not run:
# rstanarm -------------------------------------------
if (require("rstanarm") && require("projpred")) {
  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)
}

## End(Not run)
```

Description

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

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

```r
# 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.
- `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", "dispersion" or "all" (default). May be abbreviated.
- `verbose`: Toggle 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::mvtnorm()`) with mean `mu = coef(model)` and variance `Sigma = vcov(model)`.

**Models with Zero-Inflation Component:** For models from packages `glmmTMB`, `pscl`, `GLMMadapтивe` 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

```r
library(parameters)
model <- lm(Sepal.Length ~ Species * Petal.Width + Petal.Length, data = iris)
head(simulate_model(model))
```
if (require("glmmTMB", quietly = TRUE)) {
  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.glmmTMB**

*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**

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

```
simulate_parameters(model, ...)
```

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

```r
```
simulate_parameters.glmmTMB

Arguments

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

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, GLM-
Madaptive 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)
## Not run:
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")
}
## End(Not run)

skewness

Compute Skewness and (Excess) Kurtosis

Description

Compute Skewness and (Excess) 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, ...)

## S3 method for class 'parameters_skewness'
summary(object, test = FALSE, ...)

## S3 method for class 'parameters_kurtosis'
summary(object, test = FALSE, ...)

Arguments

x A numeric vector or data.frame.

na.rm Remove missing values.
skewness

**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.

**object**
An object returned by skewness() or kurtosis().

**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


**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 $g_1 = \frac{\sum((x - \text{mean}(x))^3)}{n} / \left(\frac{\sum((x - \text{mean}(x))^2)}{n}\right)^{1.5}$
- Type "2" first calculates the type-1 skewness, than adjusts the result: $G_1 = g_1 \times \sqrt{n \times (n - 1)} / (n - 2)$. This is what SAS and SPSS usually return
- Type "3" first calculates the type-1 skewness, than adjusts the result: $b_1 = g_1 \times ((1 - 1 / n)^{1.5}$. 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 $g_2 = n \times \sum((x - \text{mean}(x))^4) / \left(\sum((x - \text{mean}(x))^2)^2\right)^{-3}$.
- Type "2" first calculates the type-1 kurtosis, than adjusts the result: $G_2 = ((n + 1) \times g_2 + 6) * (n - 1) / ((n - 2) * (n - 3))$. This is what SAS and SPSS usually return
- Type "3" first calculates the type-1 kurtosis, than adjusts the result: $b_2 = (g_2 + 3) \times (1 - 1 / n)^{2 - 3}$. 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).
smoothness

Value

Values of skewness or kurtosis.

References


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

Examples

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

**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, ...)`

```r
## Default S3 method:
standard_error(model, method = NULL, verbose = TRUE, ...)
```

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

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

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

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

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

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

```r
## 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 'MixMod'
standard_error(
  model,
  effects = c("fixed", "random"),
  component = c("all", "conditional", "zi", "zero_inflated"),
  verbose = TRUE,
  ...
)

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

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

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

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

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().

- **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".

- **verbose**
  Toggle warnings and messages.
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", "dispersion" or "all" (default). May be abbreviated.

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.

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.

Value
A data frame with at least two columns: the parameter names and the standard errors. Depending on the model, may also include columns for model components etc.

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

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

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.

component Should all parameters or parameters for specific model components be returned?

... 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.
See Also

Working examples can be found in this vignette.

Examples

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

## Not run:
if (require("clubSandwich", quietly = TRUE)) {
  # 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)
  )
}

## End(Not run)
```
Index

* data
  fish, 52
  qol_cancer, 115

Additive models, 56
anova, 59
anovaBF, 66
ANOVAs, 56
aov, 59

Bartlett’s Test of Sphericity, 10
Bayesian counterpart, 46
Bayesian models, 56, 109
Bayesian regressions, 63, 67, 71, 79, 82, 95, 98
Bayesian tests, 56
bootstrap_model, 4, 6, 7, 126
bootstrap_model(), 124
bootstrap_parameters, 5, 6, 126
bootstrap_parameters(), 63, 67, 71, 79, 82, 95, 98, 124

center, 7, 14
CFA and SEM, 56
check_clusterstructure, 9, 10, 30, 31
check_factorstructure, 10, 10, 50
check_heterogeneity, 11, 57
check_itemscale, 50
check_kmo, 10, 15
check_multimodal, 16
check_sphericity_bartlett, 10, 17
ci, 75
ci.betafemx (ci.default), 18
ci.betafemx(ci.default), 18
ci.clm2 (ci.default), 18
ci.clm2 (ci.default), 18
ci.DirichletRegModel (ci.default), 18
ci.glm (ci.default), 18
ci.glmmTMB (ci.default), 18
ci.HLfit (ci.default), 18

ci.hurdle (ci.default), 18
ci.lme (ci.default), 18
ci.merMod (ci.default), 18
ci.MixMod (ci.default), 18
ci.mixor (ci.default), 18
ci.poissonfemx (ci.default), 18
ci.polr (ci.default), 18
ci.svyglm (ci.default), 18
ci.zeroinfl (ci.default), 18
ci_betwithin, 22
ci_kenward, 23
ci_ml1, 24
ci_robust (standard_error_robust), 132
ci_satterthwaite, 26
ci_wald, 27
closest_component, 50
closest_component (factor_analysis), 48
Cluster models, 56
cluster_analysis, 29, 31
cluster_discrimination, 30, 31
compare_models (compare_parameters), 31
compare_parameters, 31
convert_data_to_numeric, 34
convert_efam_to_cfa, 35
correlationBF, 66
Correlations, t-tests, ...., 56

data_partition, 36
data_to_numeric
  (convert_data_to_numeric), 34
degrees_of_freedom, 37, 72
degroup (check_heterogeneity), 11
demean, 9, 57, 72
demean (check_heterogeneity), 11
describe_distribution, 38
describe_posterior, 63, 68, 80, 83, 93, 95, 99
describe_posterior(), 6
detrend (check_heterogeneity), 11
display.equivalence_test_lm
(display.parameters_model), 40

display.parameters_efa
(display.parameters_model), 40

display.parameters_efa_summary
(display.parameters_model), 40

display.parameters_model, 40

display.parameters_sem
(display.parameters_model), 40

display.parameters_stan
(display.parameters_model), 40

dist, 29

dist(), 9

dof (degrees_of_freedom), 37

dof_betwithin, 37, 38, 72

dof_betwithin (ci_betwithin), 22

dof_kenward, 37, 72

dof_kenward (ci_kenward), 23

dof_ml1, 37, 72

dof_ml1 (ci_ml1), 24

dof_ml1(), 24, 27

dof_satterthwaite, 37, 72

dof_satterthwaite (ci_satterthwaite), 26

dof_satterthwaite(), 24

efa_to_cfa (convert_efa_to_cfa), 35

equivalence_test(), 43, 47

equivalence_test.lm, 45

equivalence_test.merMod
  (equivalence_test.lm), 45

eti, 6, 64, 65, 75, 126

fa, 49

factor_analysis, 48

fish, 52

model_parameters_model
  (display.parameters_model), 40

format.order, 52

format_p_adjust, 54

format_parameters, 53

format_value, 52

format_value(), 52

generalTestBF, 66

get_scores, 50, 55

get_variance, 116

glht, 93

hclust, 29

hdi, 6, 64, 65, 75, 126

Hypothesis testing, 56

IQR, 39

Kaiser, Meyer, Olkin (KMO) Measure of Sampling Adequacy (MSA), 10

kmeans, 29, 86

kurtosis (skewness), 127

lmBF, 66

manova, 59

Marginal effects models, 109

Meta-Analysis via linear (mixed) models, 56

Mixed models, 56, 109

model_parameters, 31–33, 56, 104

model_parameters(), 43, 44, 106

model_parameters.aov, 58

model_parameters.averaging, 61

model_parameters.befa, 63

model_parameters.betamfx
  (model_parameters.default), 77

model_parameters.betareg
  (model_parameters.averaging), 61

model_parameters.BFBayesFactor, 65

model_parameters.bracl
  (model_parameters.DirichletRegModel), 81

model_parameters.brmsfit
  (model_parameters.data.frame), 73

model_parameters.cgam, 66

model_parameters.clm2
  (model_parameters.DirichletRegModel), 81

model_parameters.clm
  (model_parameters.cpglmm), 69

model_parameters.cpglmm, 69

model_parameters.data.frame, 73

model_parameters.default, 56, 77

model_parameters.DirichletRegModel, 81

model_parameters.gam
  (model_parameters.cpglmm), 66

model_parameters.glht
  (model_parameters.PMCMR), 93

model_parameters.glm
  (model_parameters.default), 77
INDEX

model_parameters.glmmTMB
  (model_parameters.cpglmm), 69
model_parameters.glmx
  (model_parameters.averaging), 61
model_parameters.htest, 84
model_parameters.kmeans, 86
model_parameters.lavaan, 60, 68, 72, 75, 80, 86, 87, 99
model_parameters.logitor
  (model_parameters.default), 77
model_parameters.Mclust, 88
model_parameters.merMod
  (model_parameters.cpglmm), 69
model_parameters.mira, 89
model_parameters.mixor
  (model_parameters.cpglmm), 69
model_parameters.mlm
  (model_parameters.DirichletRegModel), 81
model_parameters.omega
  (model_parameters.PCA), 91
model_parameters.pairwise.htest
  (model_parameters.averaging), 84
model_parameters.PCA, 91
model_parameters.PMCMR, 93
model_parameters.poissonmfx
  (model_parameters.default), 77
model_parameters.principal
  (model_parameters.PCA), 91
model_parameters.rma, 94
model_parameters.rqss
  (model_parameters.cpglmm), 66
model_parameters.stanreg
  (model_parameters.data.frame), 73
model_parameters.t1way, 96
model_parameters.zcpglm, 97
Models with special components, 109
Multinomial, ordinal and cumulative
  link models, 56
Multiply imputed repeated analyses, 56
n_clusters, 29–31, 99
n_components, 49, 50, 118
n_components (n_factors), 100
n_factors, 49, 50, 100, 118
Other special models, 56
p.adjust, 33, 47, 62, 68, 72, 79, 83, 90, 98
p_direction, 6, 64, 65, 75, 110, 126
p_value, 109
p_value.averaging
  (p_value.DirichletRegModel), 112
p_value.beta (p_value.poissonmfx), 113
p_value.betaor (p_value.poissonmfx), 113
p_value.betareg
  (p_value.DirichletRegModel), 112
p_value.BFBayesFactor, 110
p_value.cgam
  (p_value.DirichletRegModel), 112
p_value.clm2
  (p_value.DirichletRegModel), 112
p_value.cpglmm, 111
p_value.DirichletRegModel, 112
p_value.glmmTMB (p_value.cpglmm), 111
p_value.lmerMod (p_value.cpglmm), 111
p_value.merMod (p_value.cpglmm), 111
p_value.MixMod (p_value.cpglmm), 111
p_value.mixor (p_value.cpglmm), 111
p_value.poissonmfx, 113
p_value.zcpglm, 114
p_value.zeroinfl (p_value.zcpglm), 114
p_value.kenward (ci_betwithin), 22
p_value.kenward (ci_kenward), 23
p_value.ml1, 112
p_value.ml1 (ci.ml1), 24
p_value.robust (standard_error.robust), 132
p_value.satterthwaite, 112
p_value.satterthwaite
  (ci_satterthwaite), 26
p_value.wald, 112
p_value.wald (ci_wald), 27
parameters (model_parameters), 56
parameters_type, 103
PCA and FA, 56
pool_parameters, 104
predict.parameters_efa
  (factor_analysis), 48
principal_components, 55, 118
principal_components(factor_analysis), 48
principal_components(), 43
print(), 53, 58, 107
print.parameters_efa(factor_analysis), 48
print.parameters_kurtosis(skewness), 127
print.parameters_model, 56, 106
print.parameters_skewness(skewness), 127
print_html.parameters_model (display.parameters_model), 40
print_md(), 58, 108
print_md.parameters_model (display.parameters_model), 40
qol_cancer, 115
random_parameters, 116
reduce_data(reduce_parameters), 117
reduce_parameters, 49, 117, 118
Regression models, 56
regressionBF, 66
rescale_weights, 119
reshape_loadings, 121
Robust statistical tests, 56
rope, 6, 64, 65, 75, 126
rope_range, 47
rotated_data, 50
rotated_data(factor_analysis), 48
se_betwithin(ci_betwithin), 22
se_kenward(ci_kenward), 23
se_ml1(ci_ml1), 24
se_satterthwaite(ci_satterthwaite), 26
select_parameters, 122
si, 6, 64, 65, 75, 126
simulate_model, 5, 7, 123, 126
simulate_parameters, 5, 7
simulate_parameters (simulate_parameters.glmmTMB), 125
simulate_parameters(), 43, 124
simulate_parameters.glmmTMB, 125
skewness, 127
smoothness, 129
sort.parameters_efa(factor_analysis), 48
sphericity, 50
standard_error, 130
standard_error_robust, 132
standard_error_robust(), 28, 68, 73, 79, 98, 131
standardize_names(), 58, 63, 69, 73, 77, 80, 83, 99
standardize_parameters, 32, 63, 68, 71, 76, 79, 82, 95, 98
standardize_parameters(), 57
stepAIC(), 122
summary.parameters_kurtosis(skewness), 127
summary.parameters_model (print.parameters_model), 106
summary.parameters_skewness(skewness), 127
t_to_d, 84
ttestBF, 66
vcovCL, 133
vcovHC(), 133
VSS, 101
Zero-inflated models, 56, 109