Package 'parameters'

September 16, 2023

Type Package

Title Processing of Model Parameters

Version 0.21.2

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 list of supported models using the function 'insight::supported_models()'), 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 (>= 3.6)

Imports bayestestR (>= 0.13.0), datawizard (>= 0.7.0), insight (>= 0.19.4), graphics, methods, stats, utils

Suggests AER, afex, aod, BayesFactor, BayesFM, bbmle, betareg, BH, biglm, blme, boot, brglm2, brms, broom, cAIC4, car, carData, cgam, ClassDiscovery, clubSandwich, cluster, coda, cplm, dbscan, domir (>= 0.2.0), drc, DRR, effectsize (>= 0.8.2), EGAnet (>= 0.7), emmeans (>= 1.7.0), estimatr, factoextra, FactoMineR, fastICA, fixest, fpc, gam, gamlss, gee, geepack, ggeffects (>= 1.2.0), ggplot2, GLMMadaptive, glmmTMB, GPArotation, gt, haven, httr, Hmisc, ivprobit, ivreg, knitr, lavaan, lfe, lm.beta, lme4, lmerTest, lmtest, logspline, lqmm, M3C, marginaleffects (>= 0.10.0), MASS, Matrix, mclogit, mclust, MCMCglmm, mediation, merDeriv, metaBMA, metafor, mfx, mgcv, mice, mmrm, multcomp, MuMIn, NbClust, nFactors, nestedLogit, nlme, nnet, openxlsx, ordinal, panelr, pbkrtest, PCDimension, performance (>= 0.10.1), plm, PMCMRplus, poorman,

posterior, PROreg, pscl, psych, pvclust, quantreg, randomForest, rmarkdown, rms, rstanarm, sandwich, see (>= 0.7.5), sparsepca, survey, survival, testthat, tidyselect, TMB, truncreg, VGAM, withr, WRS2

VignetteBuilder knitr

Encoding UTF-8

Language en-US

RoxygenNote 7.2.3

Config/testthat/edition 3

Config/testthat/parallel true

Config/Needs/website rstudio/bslib, r-lib/pkgdown, easystats/easystatstemplate

Config/rcmdcheck/ignore-inconsequential-notes true

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),
Christina Maimone [rev],
Niels Ohlsen [rev] (@Niels_Bremen),
Douglas Ezra Morrison [ctb] (<https://orcid.org/0000-0002-7195-830X>, @demstats1),
Joseph Luchman [ctb] (<https://orcid.org/0000-0002-8886-9717>)

Repository CRAN

Date/Publication 2023-09-16 13:50:03 UTC

R topics documented:

`bootstrap_model` *Model bootstrapping*

Description

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

Usage

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

## Default S3 method:
bootstrap_model(
  model,
  iterations = 1000,
  type = "ordinary",
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1,
  verbose = FALSE,
  ...
)

## S3 method for class 'merMod'
bootstrap_model(
  model,
  iterations = 1000,
  type = "parametric",
  parallel = c("no", "multicore", "snow"),
  n_cpus = 1,
  cluster = NULL,
  verbose = FALSE,
  ...
)
```

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.
cluster	Optional cluster when parallel = "snow". See ?lme4::bootMer for details.

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_parameter() 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

```
## Not run:
model <- lm(mpg ~ wt + factor(cyl), data = mtcars)
b <- bootstrap_model(model)
print(head(b))

est <- emmeans::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

```
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"` (see `map_estimate()`), `"trimmed"` (which is just `mean(x, trim = threshold)`), `"mode"` or `"all"`.
`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 `"ETI"` (default, see `eti()`), `"HDI"` (see `hdi()`), `"BCI"` (see `bci()`), `"SPI"` (see `spi()`), 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. `bayestestR::rope()` or `bayestestR::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 `bayestestR::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_parameter()` 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

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

See Also

`bootstrap_model()`, `simulate_parameters()`, `simulate_model()`

Examples

```
## Not run:
set.seed(2)
model <- lm(Sepal.Length ~ Species * Petal.Width, data = iris)
b <- bootstrap_parameters(model)
print(b)

est <- emmeans::emmeans(b, trt.vs.ctrl ~ Species)
print(model_parameters(est))

## End(Not run)
```

ci.default *Confidence Intervals (CI)*

Description

`ci()` attempts to return confidence intervals of model parameters.

Usage

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

## S3 method for class 'glmmTMB'
ci(
  x,
  ci = 0.95,
  dof = NULL,
  method = "wald",
  component = "all",
  verbose = TRUE,
  ...
)

## S3 method for class 'merMod'
ci(x, ci = 0.95, dof = NULL, method = "wald", iterations = 500, ...)
```

Arguments

`x`	A statistical model.
`ci`	Confidence Interval (CI) level. Default to `0.95` (95%).
`dof`	Number of degrees of freedom to be used when calculating confidence intervals. If `NULL` (default), the degrees of freedom are retrieved by calling `degrees_of_freedom()` with approximation method defined in `method`. If not `NULL`, use this argument to override the default degrees of freedom used to compute confidence intervals.
`method`	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): `"residual"`, `"normal"`, `"likelihood"`, `"satterthwaite"`, `"kenward"`, `"wald"`, `"profile"`, `"boot"`, `"uniroot"`, `"ml1"`, `"betwithin"`, `"hdi"`, `"quantile"`, `"ci"`, `"eti"`, `"si"`, `"bci"`, or `"bcai"`. See section *Confidence intervals and approximation of degrees of freedom* in `model_parameters()` for further details.
`...`	Additional arguments
`component`	Model component for which parameters should be shown. See the documentation for your object's class in `model_parameters()` or `p_value()` for further details.

verbose	Toggle warnings and messages.
iterations	The number of bootstrap replicates. Only applies to models of class merMod when method=boot.

Value

A data frame containing the CI bounds.

Confidence intervals and approximation of degrees of freedom

There are different ways of approximating the degrees of freedom depending on different assumptions about the nature of the model and its sampling distribution. The ci_method argument modulates the method for computing degrees of freedom (df) that are used to calculate confidence intervals (CI) and the related p-values. Following options are allowed, depending on the model class:

Classical methods:

Classical inference is generally based on the **Wald method**. The Wald approach to inference computes a test statistic by dividing the parameter estimate by its standard error (Coefficient / SE), then comparing this statistic against a t- or normal distribution. This approach can be used to compute CIs and p-values.

"wald":

- Applies to *non-Bayesian models*. For *linear models*, CIs computed using the Wald method (SE and a *t-distribution with residual df*); p-values computed using the Wald method with a *t-distribution with residual df*. For other models, CIs computed using the Wald method (SE and a *normal distribution*); p-values computed using the Wald method with a *normal distribution*.

"normal"

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a normal distribution.

"residual"

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a *t-distribution with residual df* when possible. If the residual df for a model cannot be determined, a normal distribution is used instead.

Methods for mixed models:

Compared to fixed effects (or single-level) models, determining appropriate df for Wald-based inference in mixed models is more difficult. See the R GLMM FAQ for a discussion.

Several approximate methods for computing df are available, but you should also consider instead using profile likelihood ("profile") or bootstrap ("boot") CIs and p-values instead.

"satterthwaite"

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with Satterthwaite df*); p-values computed using the Wald method with a *t-distribution with Satterthwaite df*.

"kenward"

- Applies to *linear mixed models*. CIs computed using the Wald method (*Kenward-Roger SE* and a *t-distribution with Kenward-Roger df*); p-values computed using the Wald method with *Kenward-Roger SE and t-distribution with Kenward-Roger df*.

`"ml1"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with m-l-1 approximated df*); p-values computed using the Wald method with a *t-distribution with m-l-1 approximated df*. See `ci_ml1()`.

`"betwithin"`

- Applies to *linear mixed models* and *generalized linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with between-within df*); p-values computed using the Wald method with a *t-distribution with between-within df*. See `ci_betwithin()`.

Likelihood-based methods:

Likelihood-based inference is based on comparing the likelihood for the maximum-likelihood estimate to the the likelihood for models with one or more parameter values changed (e.g., set to zero or a range of alternative values). Likelihood ratios for the maximum-likelihood and alternative models are compared to a χ-squared distribution to compute CIs and p-values.

`"profile"`

- Applies to *non-Bayesian models* of class `glm`, `polr`, `merMod` or `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using linear interpolation to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

`"uniroot"`

- Applies to *non-Bayesian models* of class `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using root finding to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

Methods for bootstrapped or Bayesian models:

Bootstrap-based inference is based on **resampling** and refitting the model to the resampled datasets. The distribution of parameter estimates across resampled datasets is used to approximate the parameter's sampling distribution. Depending on the type of model, several different methods for bootstrapping and constructing CIs and p-values from the bootstrap distribution are available.

For Bayesian models, inference is based on drawing samples from the model posterior distribution.

`"quantile"` (or `"eti"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *equal tailed intervals* using the quantiles of the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::eti()`.

`"hdi"`

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *highest density intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::hdi()`.

`"bci"` (or `"bcai"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *bias corrected and accelerated intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::bci()`.

`"si"`

- Applies to *Bayesian models* with proper priors. CIs computed as *support intervals* comparing the posterior samples against the prior samples; p-values are based on the *probability of direction*. See `bayestestR::si()`.

`"boot"`

- Applies to *non-Bayesian models* of class `merMod`. CIs computed using *parametric bootstrapping* (simulating data from the fitted model); p-values computed using the Wald method with a *normal-distribution)* (note: this might change in a future update!).

For all iteration-based methods other than `"boot"` (`"hdi"`, `"quantile"`, `"ci"`, `"eti"`, `"si"`, `"bci"`, `"bcai"`), p-values are based on the probability of direction (`bayestestR::p_direction()`), which is converted into a p-value using `bayestestR::pd_to_p()`.

Examples

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

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

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

Description

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

Usage

```
ci_betwithin(model, ci = 0.95, ...)

dof_betwithin(model)

p_value_betwithin(model, dof = NULL, ...)
```

Arguments

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

Details

Small Sample Cluster corrected Degrees of Freedom:

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

Degrees of Freedom for Longitudinal Designs (Repeated Measures):

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

Value

A data frame.

References

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

See Also

`dof_betwithin()` is a small helper-function to calculate approximated degrees of freedom of model parameters, based on the "between-within" heuristic.

Examples

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

---

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

Description

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

Usage

```
ci_kenward(model, ci = 0.95)

dof_kenward(model)

p_value_kenward(model, dof = NULL)

se_kenward(model)
```

Arguments

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

Details

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

Value

A data frame.

References

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

See Also

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

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

Examples

```
if (require("lme4", 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, ...)
```

Arguments

model	A mixed model.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
...	Additional arguments
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 *m-l-1* heuristic is such an approach that uses a t-distribution with fewer degrees of freedom (dof_ml1()) to calculate p-values (p_value_ml1()) and confidence intervals (ci(method = "ml1")).

Degrees of Freedom for Longitudinal Designs (Repeated Measures):

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

Limitations of the "m-l-1" Heuristic:

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

Value

A data frame.

References

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

See Also

dof_ml1() is a small helper-function to calculate approximated degrees of freedom of model parameters, based on the "m-l-1" heuristic.

Examples

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

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

Description

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

Usage

```
ci_satterthwaite(model, ci = 0.95, ...)

dof_satterthwaite(model)

p_value_satterthwaite(model, dof = NULL, ...)

se_satterthwaite(model)
```

Arguments

`model`	A statistical model.
`ci`	Confidence Interval (CI) level. Default to `0.95` (95%).
`...`	Additional arguments
`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 Satterthwaite approximation is also applicable in more complex multilevel designs. However, the "m-l-1" heuristic also applies to generalized mixed models, while approaches like Kenward-Roger or Satterthwaite are limited to linear mixed models only.

Value

A data frame.

References

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

See Also

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

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

Examples

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

---

cluster_analysis *Cluster Analysis*

Description

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

Usage

```
cluster_analysis(
  x,
  n = NULL,
  method = "kmeans",
  include_factors = FALSE,
  standardize = TRUE,
  verbose = TRUE,
  distance_method = "euclidean",
  hclust_method = "complete",
  kmeans_method = "Hartigan-Wong",
  dbscan_eps = 15,
  iterations = 100,
  ...
)
```

Arguments

x	A data frame (with at least two variables), or a matrix (with at least two columns).
n	Number of clusters used for supervised cluster methods. If NULL, the number of clusters to extract is determined by calling n_clusters(). Note that this argument does not apply for unsupervised clustering methods like dbscan, hdbscan, mixture, pvclust, or pamk.
method	Method for computing the cluster analysis. Can be "kmeans" (default; k-means using kmeans()), "hkmeans" (hierarchical k-means using factoextra::hkmeans()), pam (K-Medoids using cluster::pam()), pamk (K-Medoids that finds out the number of clusters), "hclust" (hierarchical clustering using hclust() or pvclust::pvclust()), dbscan (DBSCAN using dbscan::dbscan()), hdbscan (Hierarchical DBSCAN using dbscan::hdbscan()), or mixture (Mixture modeling using mclust::Mclust(), which requires the user to run library(mclust) before).
include_factors	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.
standardize	Standardize the dataframe before clustering (default).
verbose	Toggle warnings and messages.
distance_method	Distance measure to be used for methods based on distances (e.g., when method = "hclust" for hierarchical clustering. For other methods, such as "kmeans", this argument will be ignored). Must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". See dist() and pvclust::pvclust() for more information.
hclust_method	Agglomeration method to be used when method = "hclust" or method = "hkmeans" (for hierarchical clustering). This should be one of "ward", "ward.D2", "single", "complete", "average", "mcquitty", "median" or "centroid". Default is "complete" (see hclust()).
kmeans_method	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.
dbscan_eps	The eps argument for DBSCAN method. See n_clusters_dbscan().
iterations	The number of replications.
...	Arguments passed to or from other methods.

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).

Clusters classification can be obtained via print(x, newdata = NULL, ...).

Value

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

Note

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

References

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

See Also

- `n_clusters()` to determine the number of clusters to extract.
- `cluster_discrimination()` to determine the accuracy of cluster group classification via linear discriminant analysis (LDA).
- `performance::check_clusterstructure()` to check suitability of data for clustering.
- https://www.datanovia.com/en/lessons/

Examples

```
set.seed(33)
# K-Means ============================================
rez <- cluster_analysis(iris[1:4], n = 3, method = "kmeans")
rez # Show results
predict(rez) # Get clusters
summary(rez) # Extract the centers values (can use 'plot()' on that)
if (requireNamespace("MASS", quietly = TRUE)) {
  cluster_discrimination(rez) # Perform LDA
}

# Hierarchical k-means (more robust k-means)
if (require("factoextra", quietly = TRUE)) {
  rez <- cluster_analysis(iris[1:4], n = 3, method = "hkmeans")
  rez # Show results
  predict(rez) # Get clusters
}

# Hierarchical Clustering (hclust) ===========================
rez <- cluster_analysis(iris[1:4], n = 3, method = "hclust")
rez # Show results
predict(rez) # Get clusters

# K-Medoids (pam) ============================================
if (require("cluster", quietly = TRUE)) {
  rez <- cluster_analysis(iris[1:4], n = 3, method = "pam")
  rez # Show results
  predict(rez) # Get clusters
```

```
}

# PAM with automated number of clusters
if (require("fpc", quietly = TRUE)) {
  rez <- cluster_analysis(iris[1:4], method = "pamk")
  rez # Show results
  predict(rez) # Get clusters
}

# DBSCAN ==================================================
if (require("dbscan", quietly = TRUE)) {
  # Note that you can assimilate more outliers (cluster 0) to neighbouring
  # clusters by setting borderPoints = TRUE.
  rez <- cluster_analysis(iris[1:4], method = "dbscan", dbscan_eps = 1.45)
  rez # Show results
  predict(rez) # Get clusters
}

# Mixture =================================================
if (require("mclust", quietly = TRUE)) {
  library(mclust) # Needs the package to be loaded
  rez <- cluster_analysis(iris[1:4], method = "mixture")
  rez # Show results
  predict(rez) # Get clusters
}
```

`cluster_centers` *Find the cluster centers in your data*

Description

For each cluster, computes the mean (or other indices) of the variables. Can be used to retrieve the centers of clusters. Also returns the within Sum of Squares.

Usage

```
cluster_centers(data, clusters, fun = mean, ...)
```

Arguments

`data`	A data.frame.
`clusters`	A vector with clusters assignments (must be same length as rows in data).
`fun`	What function to use, `mean` by default.
`...`	Other arguments to be passed to or from other functions.

Value

A dataframe containing the cluster centers. Attributes include performance statistics and distance between each observation and its respective cluster centre.

Examples

```
k <- kmeans(iris[1:4], 3)
cluster_centers(iris[1:4], clusters = k$cluster)
cluster_centers(iris[1:4], clusters = k$cluster, fun = median)
```

`cluster_discrimination` *Compute a linear discriminant analysis on classified cluster groups*

Description

Computes linear discriminant analysis (LDA) on classified cluster groups, and determines the goodness of classification for each cluster group. See `MASS::lda()` for details.

Usage

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

Arguments

`x`	A data frame
`cluster_groups`	Group classification of the cluster analysis, which can be retrieved from the `cluster_analysis()` function.
`...`	Other arguments to be passed to or from.

See Also

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

Examples

```
# Retrieve group classification from hierarchical cluster analysis
clustering <- cluster_analysis(iris[, 1:4], n = 3)

# Goodness of group classification
cluster_discrimination(clustering)
```

`cluster_meta` *Metaclustering*

Description

One of the core "issue" of statistical clustering is that, in many cases, different methods will give different results. The **metaclustering** approach proposed by *easystats* (that finds echoes in *consensus clustering*; see Monti et al., 2003) consists of treating the unique clustering solutions as a ensemble, from which we can derive a probability matrix. This matrix contains, for each pair of observations, the probability of being in the same cluster. For instance, if the 6th and the 9th row of a dataframe has been assigned to a similar cluster by 5 our of 10 clustering methods, then its probability of being grouped together is 0.5.

Usage

```
cluster_meta(list_of_clusters, rownames = NULL, ...)
```

Arguments

`list_of_clusters`	A list of vectors with the clustering assignments from various methods.
`rownames`	An optional vector of row.names for the matrix.
`...`	Currently not used.

Details

Metaclustering is based on the hypothesis that, as each clustering algorithm embodies a different prism by which it sees the data, running an infinite amount of algorithms would result in the emergence of the "true" clusters. As the number of algorithms and parameters is finite, the probabilistic perspective is a useful proxy. This method is interesting where there is no obvious reasons to prefer one over another clustering method, as well as to investigate how robust some clusters are under different algorithms.

This metaclustering probability matrix can be transformed into a dissimilarity matrix (such as the one produced by `dist()`) and submitted for instance to hierarchical clustering (`hclust()`). See the example below.

Value

A matrix containing all the pairwise (between each observation) probabilities of being clustered together by the methods.

Examples

```
## Not run:
data <- iris[1:4]

rez1 <- cluster_analysis(data, n = 2, method = "kmeans")
```

```
rez2 <- cluster_analysis(data, n = 3, method = "kmeans")
rez3 <- cluster_analysis(data, n = 6, method = "kmeans")

list_of_clusters <- list(rez1, rez2, rez3)

m <- cluster_meta(list_of_clusters)

# Visualize matrix without reordering
heatmap(m, Rowv = NA, Colv = NA, scale = "none") # Without reordering
# Reordered heatmap
heatmap(m, scale = "none")

# Extract 3 clusters
predict(m, n = 3)

# Convert to dissimilarity
d <- as.dist(abs(m - 1))
model <- hclust(d)
plot(model, hang = -1)

## End(Not run)
```

`cluster_performance` *Performance of clustering models*

Description

Compute performance indices for clustering solutions.

Usage

```
cluster_performance(model, ...)

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

## S3 method for class 'hclust'
cluster_performance(model, data, clusters, ...)

## S3 method for class 'dbscan'
cluster_performance(model, data, ...)

## S3 method for class 'parameters_clusters'
cluster_performance(model, ...)
```

Arguments

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

data	A data.frame.
clusters	A vector with clusters assignments (must be same length as rows in data).

Examples

```
# kmeans
model <- kmeans(iris[1:4], 3)
cluster_performance(model)
# hclust
data <- iris[1:4]
model <- hclust(dist(data))
clusters <- cutree(model, 3)

rez <- cluster_performance(model, data, clusters)
rez

# DBSCAN
model <- dbscan::dbscan(iris[1:4], eps = 1.45, minPts = 10)

rez <- cluster_performance(model, iris[1:4])
rez

# Retrieve performance from parameters
params <- model_parameters(kmeans(iris[1:4], 3))
cluster_performance(params)
```

---

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,
  ci_method = "wald",
  p_adjust = NULL,
  select = NULL,
  column_names = NULL,
  pretty_names = TRUE,
```

```
  keep = NULL,
  drop = NULL,
  verbose = TRUE
)

compare_models(
  ...,
  ci = 0.95,
  effects = "fixed",
  component = "conditional",
  standardize = NULL,
  exponentiate = FALSE,
  ci_method = "wald",
  p_adjust = NULL,
  select = NULL,
  column_names = NULL,
  pretty_names = TRUE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE
)
```

Arguments

`...`	One or more regression model objects, or objects returned by `model_parameters()`. Regression models may be of different model types. Model objects may be passed comma separated, or as a list. If model objects are passed with names or the list has named elements, these names will be used as column names.
`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. If the calculation of random effects parameters takes too long, you may use `effects = "fixed"`.
`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 `NULL` (default; no standardization), `"refit"` (for re-fitting the model on standardized data) or one of `"basic"`, `"posthoc"`, `"smart"`, `"pseudo"`. See 'Details' in `standardize_parameters()`. **Importantly**: • The `"refit"` method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use `standardize="basic"` or standardize the data with `datawizard::standardize(force=TRUE)` *before* fitting the model. • For mixed models, when using methods other than `"refit"`, only the fixed effects will be standardized.

- Robust estimation (i.e., vcov set to a value other than NULL) 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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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.

ci_method — Method for computing degrees of freedom for p-values 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 stats::p.adjust() for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from **emmeans**).

select — Determines which columns and and which layout columns are printed. There are three options for this argument:

1. Selecting columns by name or index
   select can be a character vector (or numeric index) of column names that should be printed. There are two pre-defined options for selecting columns: select = "minimal" prints coefficients, confidence intervals and p-values, while select = "short" prints coefficients, standard errors and p-values.
2. A string expression with layout pattern
   select is a string with "tokens" enclosed in braces. These tokens will be replaced by their associated columns, where the selected columns will be collapsed into one column. However, it is possible to create multiple columns as well. Following tokens are replaced by the related coefficients or statistics: {estimate}, {se}, {ci} (or {ci_low} and {ci_high}), {p} and {stars}. The token {ci} will be replaced by {ci_low}, {ci_high}. Furthermore, a | separates values into new cells/columns. If format = "html", a
 inserts a line break inside a cell. See 'Examples'.
3. A string indicating a pre-defined layout
   select can be one of the following string values, to create one of the following pre-defined column layouts:
   - "ci": Estimates and confidence intervals, no asterisks for p-values. This is equivalent to select = "{estimate} ({ci})".
   - "se": Estimates and standard errors, no asterisks for p-values. This is equivalent to select = "{estimate} ({se})".
   - "ci_p": Estimates, confidence intervals and asterisks for p-values. This is equivalent to select = "{estimate}{stars} ({ci})".
   - "se_p": Estimates, standard errors and asterisks for p-values. This is equivalent to select = "{estimate}{stars} ({se})"..

- `"ci_p2"`: Estimates, confidence intervals and numeric p-values, in two columns. This is equivalent to `select = "{estimate} ({ci})|{p}"`.
- `"se_p2"`: Estimate, standard errors and numeric p-values, in two columns. This is equivalent to `select = "{estimate} ({se})|{p}"`.

For `model_parameters()`, glue-like syntax is still experimental in the case of more complex models (like mixed models) and may not return expected results.

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

`pretty_names` Can be `TRUE`, which will return "pretty" (i.e. more human readable) parameter names. Or `"labels"`, in which case value and variable labels will be used as parameters names. The latter only works for "labelled" data, i.e. if the data used to fit the model had `"label"` and `"labels"` attributes. See also section *Global Options to Customize Messages when Printing*.

`keep` Character containing a regular expression pattern that describes the parameters that should be included (for `keep`) or excluded (for `drop`) in the returned data frame. `keep` may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If `keep` is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in `keep` will be selected from the returned data frame (and vice versa, all parameter names matching `drop` will be excluded). Furthermore, if `keep` has more than one element, these will be merged with an `OR` operator into a regular expression pattern like this: `"(one|two|three)"`. If `keep` 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()`. Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the `$Parameter` column of the parameters table to get the exact parameter names.

`drop` See `keep`.

`verbose` Toggle warnings and messages.

Details

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. It should also be noted that when including models with interaction terms, not only do the values of the parameters change, but so does their meaning (from main effects, to simple slopes), thereby making such comparisons hard. Therefore, you should not use this function to compare models with interaction terms with models without interaction terms.

Value

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

Examples

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

# custom style
compare_parameters(lm1, lm2, select = "{estimate}{stars} ({se})")

## Not run:
# custom style, in HTML
result <- compare_parameters(lm1, lm2, select = "{estimate}<br>({se})|{p}")
print_html(result)

## End(Not run)

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("linear model" = m1, "logistic reg." = m2)
compare_parameters(m1, m2, column_names = c("linear model", "logistic reg."))

# or as list
compare_parameters(list(m1, m2))
compare_parameters(list("linear model" = m1, "logistic reg." = m2))

## End(Not run)
```

---

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,
  max_per_dimension = 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.
max_per_dimension	Maximum number of variables to keep per dimension.

Value

Converted index.

Examples

```
library(parameters)
data(attitude)
efa <- psych::fa(attitude, nfactors = 3)

model1 <- efa_to_cfa(efa)
model2 <- efa_to_cfa(efa, threshold = 0.3)
model3 <- efa_to_cfa(efa, max_per_dimension = 2)

suppressWarnings(anova(
  lavaan::cfa(model1, data = attitude),
  lavaan::cfa(model2, data = attitude),
  lavaan::cfa(model3, data = attitude)
))
```

`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), `"residual"` 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. See 'Details'.

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).
- `"residual"` 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.
- `"normal"` returns `Inf`.
- `"wald"` returns residual df for models with t-statistic, and `Inf` for all other models.
- `"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

```
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)
```

---

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 = digits,
  p_digits = 3,
  footer_digits = 3,
  ci_brackets = c("(", ")"),
  show_sigma = FALSE,
  show_formula = FALSE,
  zap_small = FALSE,
  font_size = "100%",
  line_padding = 4,
  column_labels = NULL,
  include_reference = FALSE,
  verbose = TRUE,
  ...
)

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

## 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 = digits,
  p_digits = 3,
  ci_width = NULL,
  ci_brackets = NULL,
  zap_small = FALSE,
  format = NULL,
  groups = NULL,
  include_reference = FALSE,
  ...
)

## 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 = digits,
  p_digits = 3,
  footer_digits = 3,
  ci_brackets = c("(", ")"),
  show_sigma = FALSE,
  show_formula = FALSE,
  zap_small = FALSE,
  groups = NULL,
  font_size = "100%",
  line_padding = 4,
  column_labels = NULL,
```

```
  include_reference = FALSE,
  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 = digits,
  p_digits = 3,
  footer_digits = 3,
  ci_brackets = c("(", ")"),
  show_sigma = FALSE,
  show_formula = FALSE,
  zap_small = FALSE,
  groups = NULL,
  include_reference = FALSE,
  verbose = TRUE,
  ...
)
```

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	Can be TRUE, which will return "pretty" (i.e. more human readable) parameter names. Or "labels", in which case value and variable labels will be used as parameters names. The latter only works for "labelled" data, i.e. if the data used to fit the model had "label" and "labels" attributes. See also section *Global Options to Customize Messages when Printing*.
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	Determines which columns and and which layout columns are printed. There are three options for this argument: 1. Selecting columns by name or index select can be a character vector (or numeric index) of column names that

should be printed. There are two pre-defined options for selecting columns: `select = "minimal"` prints coefficients, confidence intervals and p-values, while `select = "short"` prints coefficients, standard errors and p-values.

2. A string expression with layout pattern
`select` is a string with "tokens" enclosed in braces. These tokens will be replaced by their associated columns, where the selected columns will be collapsed into one column. However, it is possible to create multiple columns as well. Following tokens are replaced by the related coefficients or statistics: `{estimate}`, `{se}`, `{ci}` (or `{ci_low}` and `{ci_high}`), `{p}` and `{stars}`. The token `{ci}` will be replaced by `{ci_low}, {ci_high}`. Furthermore, a `|` separates values into new cells/columns. If `format = "html"`, a
 inserts a line break inside a cell. See 'Examples'.
3. A string indicating a pre-defined layout
`select` can be one of the following string values, to create one of the following pre-defined column layouts:
   - `"ci"`: Estimates and confidence intervals, no asterisks for p-values. This is equivalent to `select = "{estimate} ({ci})"`.
   - `"se"`: Estimates and standard errors, no asterisks for p-values. This is equivalent to `select = "{estimate} ({se})"`.
   - `"ci_p"`: Estimates, confidence intervals and asterisks for p-values. This is equivalent to `select = "{estimate}{stars} ({ci})"`.
   - `"se_p"`: Estimates, standard errors and asterisks for p-values. This is equivalent to `select = "{estimate}{stars} ({se})"`..
   - `"ci_p2"`: Estimates, confidence intervals and numeric p-values, in two columns. This is equivalent to `select = "{estimate} ({ci})|{p}"`.
   - `"se_p2"`: Estimate, standard errors and numeric p-values, in two columns. This is equivalent to `select = "{estimate} ({se})|{p}"`.

For `model_parameters()`, glue-like syntax is still experimental in the case of more complex models (like mixed models) and may not return expected results.

`caption` Table caption as string. If `NULL`, depending on the model, either a default caption or no table caption is printed. Use `caption = ""` to suppress the table caption.

`subtitle` Table title (same as caption) and subtitle, as strings. If `NULL`, no title or subtitle is printed, unless it is stored as attributes (`table_title`, or its alias `table_caption`, and `table_subtitle`). If `x` is a list of data frames, `caption` may be a list of table captions, one for each table.

`footer` Can either be `FALSE` or an empty string (i.e. `""`) to suppress the footer, `NULL` to print the default footer, or a string. The latter will combine the string value with the default footer.

`align` Only applies to HTML tables. May be one of `"left"`, `"right"` or `"center"`.

`digits, ci_digits, p_digits`
Number of digits for rounding or significant figures. May also be `"signif"` to return significant figures or `"scientific"` to return scientific notation. Control the number of digits by adding the value as suffix, e.g. `digits = "scientific4"` to have scientific notation with 4 decimal places, or `digits = "signif5"` for 5 significant figures (see also `signif()`).

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.
font_size	For HTML tables, the font size.
line_padding	For HTML tables, the distance (in pixel) between lines.
column_labels	Labels of columns for HTML tables. If NULL, automatic column names are generated. See 'Examples'.
include_reference	Logical, if TRUE, the reference level of factors will be added to the parameters table. This is only relevant for models with categorical predictors. The coefficient for the reference level is always 0 (except when exponentiate = TRUE, then the coefficient will be 1), so this is just for completeness.
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`.

See Also

`print.parameters_model()`

Examples

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

## Not run:
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)
out <- compare_parameters(lm1, lm2, lm3)

print_html(
  out,
  select = "{coef}{stars}|({ci})",
  column_labels = c("Estimate", "95% CI")
)

# line break, unicode minus-sign
print_html(
  out,
  select = "{estimate}{stars}<br>({ci_low} \u2212 {ci_high})",
  column_labels = c("Est. (95% CI)")
)

## End(Not run)
```

---

dominance_analysis *Dominance Analysis*

Description

Computes Dominance Analysis Statistics and Designations

Usage

```
dominance_analysis(
  model,
  sets = NULL,
  all = NULL,
```

```
  conditional = TRUE,
  complete = TRUE,
  quote_args = NULL,
  contrasts = model$contrasts,
  ...
)
```

Arguments

model	A model object supported by `performance::r2()`. See 'Details'.
sets	A (named) list of formula objects with no left hand side/response. If the list has names, the name provided each element will be used as the label for the set. Unnamed list elements will be provided a set number name based on its position among the sets as entered. Predictors in each formula are bound together as a set in the dominance analysis and dominance statistics and designations are computed for the predictors together. Predictors in `sets` must be present in the model submitted to the `model` argument and cannot be in the `all` argument.
all	A formula with no left hand side/response. Predictors in the formula are included in each subset in the dominance analysis and the R2 value associated with them is subtracted from the overall value. Predictors in `all` must be present in the model submitted to the `model` argument and cannot be in the `sets` argument.
conditional	Logical. If `FALSE` then conditional dominance matrix is not computed. If conditional dominance is not desired as an importance criterion, avoiding computing the conditional dominance matrix can save computation time.
complete	Logical. If `FALSE` then complete dominance matrix is not computed. If complete dominance is not desired as an importance criterion, avoiding computing complete dominance designations can save computation time.
quote_args	A character vector of arguments in the model submitted to `model` to `quote()` prior to submitting to the dominance analysis. This is necessary for data masked arguments (e.g., `weights`) to prevent them from being evaluated before being applied to the model and causing an error.
contrasts	A named list of `contrasts` used by the model object. This list is required in order for the correct mapping of parameters to predictors in the output when the model creates indicator codes for factor variables using `insight::get_modelmatrix()`. By default, the `contrast` element from the model object submitted is used. If the model object does not have a `contrast` element the user can supply this named list.
...	Not used at current.

Details

Computes two decompositions of the model's R2 and returns a matrix of designations from which predictor relative importance determinations can be obtained.

Note in the output that the "constant" subset is associated with a component of the model that does not directly contribute to the R2 such as an intercept. The "all" subset is apportioned a component of the fit statistic but is not considered a part of the dominance analysis and therefore does not receive a rank, conditional dominance statistics, or complete dominance designations.

The input model is parsed using `insight::find_predictors()`, does not yet support interactions, transformations, or offsets applied in the R formula, and will fail with an error if any such terms are detected.

The model submitted must accept an formula object as a `formula` argument. In addition, the model object must accept the data on which the model is estimated as a `data` argument. Formulas submitted using object references (i.e., `lm(mtcars$mpg ~ mtcars$vs)`) and functions that accept data as a non-`data` argument (e.g., `survey::svyglm()` uses `design`) will fail with an error.

Models that return `TRUE` for the `insight::model_info()` function's values "is_bayesian", "is_mixed", "is_gam", is_multivariate", "is_zero_inflated", or "is_hurdle" are not supported at current.

When `performance::r2()` returns multiple values, only the first is used by default.

Value

Object of class `"parameters_da"`.

An object of class `"parameters_da"` is a list of `data.frames` composed of the following elements:

`General` A `data.frame` which associates dominance statistics with model parameters. The variables in this `data.frame` include:

- `Parameter` Parameter names.
- `General_Dominance` Vector of general dominance statistics. The R2 ascribed to variables in the `all` argument are also reported here though they are not general dominance statistics.
- `Percent` Vector of general dominance statistics normalized to sum to 1.
- `Ranks` Vector of ranks applied to the general dominance statistics.
- `Subset` Names of the subset to which the parameter belongs in the dominance analysis. Each other `data.frame` returned will refer to these subset names.

`Conditional` A `data.frame` of conditional dominance statistics. Each observation represents a subset and each variable represents an the average increment to R2 with a specific number of subsets in the model. `NULL` if `conditional` argument is `FALSE`.

`Complete` A `data.frame` of complete dominance designations. The subsets in the observations are compared to the subsets referenced in each variable. Whether the subset in each variable dominates the subset in each observation is represented in the logical value. `NULL` if `complete` argument is `FALSE`.

Author(s)

Joseph Luchman

References

- Azen, R., & Budescu, D. V. (2003). The dominance analysis approach for comparing predictors in multiple regression. Psychological Methods, 8(2), 129-148. doi:10.1037/1082-989X.8.2.129

- Budescu, D. V. (1993). Dominance analysis: A new approach to the problem of relative importance of predictors in multiple regression. Psychological Bulletin, 114(3), 542-551. doi:10.1037/0033-2909.114.3.542
- Groemping, U. (2007). Estimators of relative importance in linear regression based on variance decomposition. The American Statistician, 61(2), 139-147. doi:10.1198/000313007X188252

See Also

`domir::domin()`

Examples

```
data(mtcars)

# Dominance Analysis with Logit Regression
model <- glm(vs ~ cyl + carb + mpg, data = mtcars, family = binomial())

performance::r2(model)
dominance_analysis(model)

# Dominance Analysis with Weighted Logit Regression
model_wt <- glm(vs ~ cyl + carb + mpg,
  data = mtcars,
  weights = wt, family = quasibinomial()
)

dominance_analysis(model_wt, quote_args = "weights")
```

---

equivalence_test.lm *Equivalence test*

Description

Compute the (conditional) equivalence test for frequentist models.

Usage

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

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

## S3 method for class 'ggeffects'
equivalence_test(
  x,
  range = "default",
  rule = "classic",
  test = "pairwise",
  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'.
verbose	Toggle warnings and messages.
...	Arguments passed to or from other methods.
effects	Should parameters for fixed effects ("fixed"), random effects ("random"), or both ("all") be returned? Only applies to mixed models. May be abbreviated. If the calculation of random effects parameters takes too long, you may use effects = "fixed".
test	Hypothesis test for computing contrasts or pairwise comparisons. See ?ggeffects::hypothesis_test for details.

Details

In classical null hypothesis significance testing (NHST) within a frequentist framework, it is not possible to accept the null hypothesis, H0 - unlike in Bayesian statistics, where such probability statements are possible. "... one can only reject the null hypothesis if the test statistics falls into the critical region(s), or fail to reject this hypothesis. In the latter case, all we can say is that no significant effect was observed, but one cannot conclude that the null hypothesis is true." (*Pernet 2017*). One way to address this issues without Bayesian methods is *Equivalence Testing*, as implemented

in `equivalence_test()`. While you either can reject the null hypothesis or claim an inconclusive result in NHST, the equivalence test - according to *Pernet* - adds a third category, *"accept"*. Roughly speaking, the idea behind equivalence testing in a frequentist framework is to check whether an estimate and its uncertainty (i.e. confidence interval) falls within a region of "practical equivalence". Depending on the rule for this test (see below), statistical significance does not necessarily indicate whether the null hypothesis can be rejected or not, i.e. the classical interpretation of the p-value may differ from the results returned from the equivalence test.

Calculation of equivalence testing:

- "bayes" - Bayesian rule (Kruschke 2018)
  This rule follows the "HDI+ROPE decision rule" *(Kruschke, 2014, 2018)* used for the `Bayesian counterpart()`. This means, if the confidence intervals are completely outside the ROPE, the "null hypothesis" for this parameter is "rejected". If the ROPE completely covers the CI, the null hypothesis is accepted. Else, it's undecided whether to accept or reject the null hypothesis. Desirable results are low proportions inside the ROPE (the closer to zero the better).
- "classic" - The TOST rule (Lakens 2017)
  This rule follows the "TOST rule", i.e. a two one-sided test procedure (*Lakens 2017*). Following this rule, practical equivalence of an effect (i.e. H0) is *rejected*, when the coefficient is statistically significant *and* the narrow confidence intervals (i.e. `1-2*alpha`) *include* or *exceed* the ROPE boundaries. Practical equivalence is assumed (i.e. H0 "accepted") when the narrow confidence intervals are completely inside the ROPE, no matter if the effect is statistically significant or not. Else, the decision whether to accept or reject practical equivalence 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 (i.e. assume) practical equivalence, 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.

p-Values:

The equivalence p-value is the area of the (cumulative) confidence distribution that is outside of the region of equivalence. It can be interpreted as p-value for *rejecting* the alternative hypothesis and *accepting* the "null hypothesis" (i.e. assuming practical equivalence). That is, a high p-value means we reject the assumption of practical equivalence and accept the alternative hypothesis.

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, Lakens and Delacre 2020)*. It represents the proportion of the confidence interval range that is inside the ROPE.

ROPE range:

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

Value

A data frame.

Note

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

References

- Blume, J. D., D'Agostino McGowan, L., Dupont, W. D., & Greevy, R. A. (2018). Second-generation p-values: Improved rigor, reproducibility, & transparency in statistical analyses. PLOS ONE, 13(3), e0188299. https://doi.org/10.1371/journal.pone.0188299
- Campbell, H., & Gustafson, P. (2018). Conditional equivalence testing: An alternative remedy for publication bias. PLOS ONE, 13(4), e0195145. doi: 10.1371/journal.pone.0195145
- Kruschke, J. K. (2014). Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan. Academic Press
- Kruschke, J. K. (2018). Rejecting or accepting parameter values in Bayesian estimation. Advances in Methods and Practices in Psychological Science, 1(2), 270-280. doi: 10.1177/2515245918771304
- Lakens, D. (2017). Equivalence Tests: A Practical Primer for t Tests, Correlations, and Meta-Analyses. Social Psychological and Personality Science, 8(4), 355–362. doi: 10.1177/1948550617697177
- Lakens, D., & Delacre, M. (2020). Equivalence Testing and the Second Generation P-Value. Meta-Psychology, 4. https://doi.org/10.15626/MP.2018.933
- Pernet, C. (2017). Null hypothesis significance testing: A guide to commonly misunderstood concepts and recommendations for good practice. F1000Research, 4, 621. doi: 10.12688/f1000research.6963.5

See Also

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

Examples

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

# default rule
equivalence_test(model)

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

# plot method
if (require("see", 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

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

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

rotated_data(pca_results, verbose = TRUE)

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

closest_component(pca_results)
```

Arguments

x	A data frame or a statistical model.
n	Number of components to extract. If n="all", then n is set as the number of variables minus 1 (ncol(x)-1). If n="auto" (default) or n=NULL, the number of components is selected through n_factors() 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 psych::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.
sparse	Whether to compute sparse PCA (SPCA, using sparsepca::spca()). SPCA attempts to find sparse loadings (with few nonzero values), which improves interpretability and avoids overfitting. Can be TRUE or "robust" (see sparsepca::robspca()).
pca_results	The output of the principal_components() function.
verbose	Toggle warnings.
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 estimates the optimal number of dimensions to retain.
- `performance::check_factorstructure()` checks the suitability of the data for factor analysis using the sphericity (see `performance::check_sphericity_bartlett()`) and the KMO (see `performance::check_kmo()`) measure.
- `performance::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 "subscales", 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 `predict()` to back-predict scores for each component, to which one can provide `newdata` or a vector of `names` for the components.

Explained Variance and Eingenvalues:

Use `summary()` to get the Eigenvalues and the explained 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 explain the variance of the data along the new feature axes.

Value

A data frame of loadings.

References

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

Examples

```
library(parameters)


# Principal Component Analysis (PCA) -------------------
principal_components(mtcars[, 1:7], n = "all", threshold = 0.2)

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

# Sparse PCA
principal_components(mtcars[, 1:7], n = 4, sparse = TRUE)
principal_components(mtcars[, 1:7], n = 4, sparse = "robust")

# Rotated PCA
```

```
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)


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

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, verbose = FALSE)


# 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_df_adjust` *Format the name of the degrees-of-freedom adjustment methods*

Description

Format the name of the degrees-of-freedom adjustment methods.

Usage

```
format_df_adjust(
  method,
  approx_string = "-approximated",
  dof_string = " degrees of freedom"
)
```

Arguments

method	Name of the method.
approx_string, dof_string	Suffix added to the name of the method in the returned string.

Value

A formatted string.

Examples

```
library(parameters)

format_df_adjust("kenward")
format_df_adjust("kenward", approx_string = "", dof_string = " DoF")
```

---

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 insight::format_value().
...	Arguments to be passed to insight::format_value() if textual is FALSE.

Value

A formatted string.

Examples

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

`format_parameters` *Parameter names formatting*

Description

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**, **ggeffects**, or **marginaleffects**. 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

```
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

```
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

```
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

```
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

```
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
}
```

model_parameters *Model Parameters*

Description

Compute and extract model parameters. The available options and arguments depend on the modeling **package** and model `class`. Follow one of these links to read the model-specific documentation:

- Default method: `lm`, `glm`, **stats**, **censReg**, **MASS**, **survey**, ...
- Additive models: **bamlss**, **gamlss**, **mgcv**, **scam**, **VGAM**, `Gam`, `gamm`, ...
- ANOVA: **afex**, `aov`, `anova`, ...
- Bayesian: **BayesFactor**, **blavaan**, **brms**, **MCMCglmm**, **posterior**, **rstanarm**, `bayesQR`, `bcplm`, `BGGM`, `blmrm`, `blrm`, `mcmc.list`, `MCMCglmm`, ...
- Clustering: **hclust**, **kmeans**, **mclust**, **pam**, ...
- Correlations, t-tests, etc.: **lmtest**, `htest`, `pairwise.htest`, ...
- Meta-Analysis: **metaBMA**, **metafor**, **metaplus**, ...
- Mixed models: **cplm**, **glmmTMB**, **lme4**, **lmerTest**, **nlme**, **ordinal**, **robustlmm**, **spaMM**, `mixed`, `MixMod`, ...
- Multinomial, ordinal and cumulative link: **brglm2**, **DirichletReg**, **nnet**, **ordinal**, `mlm`, ...
- Multiple imputation: **mice**
- PCA, FA, CFA, SEM: **FactoMineR**, **lavaan**, **psych**, `sem`, ...
- Zero-inflated and hurdle: **cplm**, **mhurdle**, **pscl**, ...
- Other models: **aod**, **bbmle**, **betareg**, **emmeans**, **epiR**, **ggeffects**, **glmx**, **ivfixed**, **ivprobit**, **JRM**, **lmodel2**, **logitsf**, **marginaleffects**, **margins**, **maxLik**, **mediation**, **mfx**, **multcomp**, **mvord**, **plm**, **PMCMRplus**, **quantreg**, **selection**, **systemfit**, **tidymodels**, **varEST**, **WRS2**, `bfsl`, `deltaMethod`, `fitdistr`, `mjoint`, `mle`, `model.avg`, ...

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. If `s_value = TRUE`, the p-value will be replaced by the S-value in the output (cf. *Rafi and Greenland 2020*). pd adds an additional column with the *probability of direction* (see `bayestestR::p_direction()` for details). `groups` can be used to group coefficients. It will be passed to the print-method, or can directly be used in `print()`, see documentation in `print.parameters_model()`.

Furthermore, see 'Examples' in `model_parameters.default()`. For developers, whose interest mainly is to get a "tidy" data frame of model summaries, it is recommended to set `pretty_names = FALSE` to speed up computation of the summary table.

Value

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

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"`.

Standardization Methods

- **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). This method is particularly recommended for complex models that include interactions or transformations (e.g., polynomial or spline terms). The `robust` (default to `FALSE`) argument enables a robust standardization of data, i.e., based on the `median` and `MAD` instead of the `mean` and `SD`. **See `standardize()` for more details. Note** that `standardize_parameters(method = "refit")` may not return the same results as fitting a model on data that has been standardized with `standardize()`; `standardize_parameters()` used the data used by the model fitting function, which might not be same data if there are missing values. see the `remove_na` argument in `standardize()`.
- **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`) of the outcome (which becomes their expression 'unit'). Then, the coefficients related to numeric variables are additionally multiplied by the standard deviation (or MAD if `robust`) of the related terms, so that they correspond to changes of 1 SD of the predictor (e.g., "A change in 1 SD of `x` is related to a change of 0.24 of the SD of `y`). This does not apply to binary variables or factors, so the coefficients are still related to changes in levels. This method is not accurate and tend to give aberrant results when interactions are specified.
- **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()`.
- **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 if `robust`) 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.

- **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 `performance::check_heterogeneity_bias` Predictors are standardized based on their SD at level of prediction (see also `datawizard::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.

See also package vignette.

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.

Confidence intervals and approximation of degrees of freedom

There are different ways of approximating the degrees of freedom depending on different assumptions about the nature of the model and its sampling distribution. The `ci_method` argument modulates the method for computing degrees of freedom (df) that are used to calculate confidence intervals (CI) and the related p-values. Following options are allowed, depending on the model class:

Classical methods:

Classical inference is generally based on the **Wald method**. The Wald approach to inference computes a test statistic by dividing the parameter estimate by its standard error (Coefficient / SE), then comparing this statistic against a t- or normal distribution. This approach can be used to compute CIs and p-values.

`"wald"`:

- Applies to *non-Bayesian models*. For *linear models*, CIs computed using the Wald method (SE and a *t-distribution with residual df*); p-values computed using the Wald method with a *t-distribution with residual df*. For other models, CIs computed using the Wald method (SE and a *normal distribution*); p-values computed using the Wald method with a *normal distribution*.

`"normal"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a normal distribution.

`"residual"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a *t-distribution with residual df* when possible. If the residual df for a model cannot be determined, a normal distribution is used instead.

Methods for mixed models:

Compared to fixed effects (or single-level) models, determining appropriate df for Wald-based inference in mixed models is more difficult. See the R GLMM FAQ for a discussion.

Several approximate methods for computing df are available, but you should also consider instead using profile likelihood (`"profile"`) or bootstrap (`"boot"`) CIs and p-values instead.

`"satterthwaite"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with Satterthwaite df*); p-values computed using the Wald method with a *t-distribution with Satterthwaite df*.

`"kenward"`

- Applies to *linear mixed models*. CIs computed using the Wald method (*Kenward-Roger SE* and a *t-distribution with Kenward-Roger df*); p-values computed using the Wald method with *Kenward-Roger SE and t-distribution with Kenward-Roger df*.

`"ml1"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with m-l-1 approximated df*); p-values computed using the Wald method with a *t-distribution with m-l-1 approximated df*. See `ci_ml1()`.

`"betwithin"`

- Applies to *linear mixed models* and *generalized linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with between-within df*); p-values computed using the Wald method with a *t-distribution with between-within df*. See `ci_betwithin()`.

Likelihood-based methods:

Likelihood-based inference is based on comparing the likelihood for the maximum-likelihood estimate to the the likelihood for models with one or more parameter values changed (e.g., set to zero or a range of alternative values). Likelihood ratios for the maximum-likelihood and alternative models are compared to a χ-squared distribution to compute CIs and p-values.

`"profile"`

- Applies to *non-Bayesian models* of class `glm`, `polr`, `merMod` or `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using linear interpolation to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

`"uniroot"`

- Applies to *non-Bayesian models* of class `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using root finding to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

Methods for bootstrapped or Bayesian models:

Bootstrap-based inference is based on **resampling** and refitting the model to the resampled datasets. The distribution of parameter estimates across resampled datasets is used to approximate the parameter's sampling distribution. Depending on the type of model, several different methods for bootstrapping and constructing CIs and p-values from the bootstrap distribution are available.

For Bayesian models, inference is based on drawing samples from the model posterior distribution.

`"quantile"` (or `"eti"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *equal tailed intervals* using the quantiles of the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::eti()`.

`"hdi"`

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *highest density intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::hdi()`.

`"bci"` (or `"bcai"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *bias corrected and accelerated intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::bci()`.

`"si"`

- Applies to *Bayesian models* with proper priors. CIs computed as *support intervals* comparing the posterior samples against the prior samples; p-values are based on the *probability of direction*. See `bayestestR::si()`.

`"boot"`

- Applies to *non-Bayesian models* of class `merMod`. CIs computed using *parametric bootstrapping* (simulating data from the fitted model); p-values computed using the Wald method with a *normal-distribution)* (note: this might change in a future update!).

For all iteration-based methods other than `"boot"` (`"hdi"`, `"quantile"`, `"ci"`, `"eti"`, `"si"`, `"bci"`, `"bcai"`), p-values are based on the probability of direction (`bayestestR::p_direction()`), which is converted into a p-value using `bayestestR::pd_to_p()`.

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**, **ggeffects**, or **marginaleffects**. 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()`.

Global Options to Customize Messages and Tables when Printing

The `verbose` argument can be used to display or silence messages and warnings for the different functions in the **parameters** package. However, some messages providing additional information can be displayed or suppressed using `options()`:

- `parameters_summary`: `options(parameters_summary = TRUE)` will override the `summary` argument in `model_parameters()` and always show the model summary for non-mixed models.
- `parameters_mixed_summary`: `options(parameters_mixed_summary = TRUE)` will override the `summary` argument in `model_parameters()` for mixed models, and will then always show the model summary.
- `parameters_cimethod`: `options(parameters_cimethod = TRUE)` will show the additional information about the approximation method used to calculate confidence intervals and p-values. Set to `FALSE` to hide this message when printing `model_parameters()` objects.
- `parameters_exponentiate`: `options(parameters_exponentiate = TRUE)` will show the additional information on how to interpret coefficients of models with log-transformed response variables or with log-/logit-links when the `exponentiate` argument in `model_parameters()` is not `TRUE`. Set this option to `FALSE` to hide this message when printing `model_parameters()` objects.

There are further options that can be used to modify the default behaviour for printed outputs:

- `parameters_labels`: `options(parameters_labels = TRUE)` will use variable and value labels for pretty names, if data is labelled. If no labels available, default pretty names are used.
- `parameters_interaction`: `options(parameters_interaction = <character>)` will replace the interaction mark (by default, `*`) with the related character.
- `parameters_select`: `options(parameters_select = <value>)` will set the default for the `select` argument. See argument's documentation for available options.

Note

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

For developers, if speed performance is an issue, you can use the (undocumented) `pretty_names` argument, e.g. `model_parameters(..., pretty_names = FALSE)`. This will skip the formatting of the coefficient names and make `model_parameters()` faster.

References

- Hoffman, L. (2015). Longitudinal analysis: Modeling within-person fluctuation and change. Routledge.
- Neter, J., Wasserman, W., & Kutner, M. H. (1989). Applied linear regression models.
- Rafi Z, Greenland S. Semantic and cognitive tools to aid statistical science: replace confidence and significance by compatibility and surprise. BMC Medical Research Methodology (2020) 20:244.

See Also

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

`model_parameters.aov` *Parameters from ANOVAs*

Description

Parameters from ANOVAs

Usage

```
## S3 method for class 'aov'
model_parameters(
  model,
  type = NULL,
  df_error = NULL,
  ci = NULL,
  alternative = NULL,
  test = NULL,
  power = FALSE,
  effectsize_type = NULL,
  keep = NULL,
  drop = NULL,
  table_wide = FALSE,
  verbose = TRUE,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  ...
)

## S3 method for class 'anova'
model_parameters(
  model,
  type = NULL,
  df_error = NULL,
  ci = NULL,
  alternative = NULL,
  test = NULL,
  power = FALSE,
  effectsize_type = NULL,
  keep = NULL,
  drop = NULL,
  table_wide = FALSE,
  verbose = TRUE,
```

```
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  ...
)

## S3 method for class 'aovlist'
model_parameters(
  model,
  type = NULL,
  df_error = NULL,
  ci = NULL,
  alternative = NULL,
  test = NULL,
  power = FALSE,
  effectsize_type = NULL,
  keep = NULL,
  drop = NULL,
  table_wide = FALSE,
  verbose = TRUE,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  ...
)

## S3 method for class 'afex_aov'
model_parameters(
  model,
  effectsize_type = NULL,
  df_error = NULL,
  type = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'anova.rms'
model_parameters(
  model,
  type = NULL,
  df_error = NULL,
  ci = NULL,
  alternative = NULL,
  test = NULL,
  power = FALSE,
  effectsize_type = NULL,
```

```
  keep = NULL,
  drop = NULL,
  table_wide = FALSE,
  verbose = TRUE,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  ...
)

## S3 method for class 'Anova.mlm'
model_parameters(
  model,
  type = NULL,
  df_error = NULL,
  ci = NULL,
  alternative = NULL,
  test = NULL,
  power = FALSE,
  effectsize_type = NULL,
  keep = NULL,
  drop = NULL,
  table_wide = FALSE,
  verbose = TRUE,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  ...
)

## S3 method for class 'maov'
model_parameters(
  model,
  type = NULL,
  df_error = NULL,
  ci = NULL,
  alternative = NULL,
  test = NULL,
  power = FALSE,
  effectsize_type = NULL,
  keep = NULL,
  drop = NULL,
  table_wide = FALSE,
  verbose = TRUE,
  omega_squared = NULL,
  eta_squared = NULL,
  epsilon_squared = NULL,
  ...
```

)

Arguments

model	Object of class aov(), anova(), aovlist, Gam, manova(), Anova.mlm, afex_aov or maov.
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.)
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.)
ci	Confidence Interval (CI) level for effect sizes specified in effectsize_type. The default, NULL, will compute no confidence intervals. ci should be a scalar between 0 and 1.
alternative	A character string specifying the alternative hypothesis; Controls the type of CI returned: "two.sided" (default, two-sided CI), "greater" or "less" (one-sided CI). Partial matching is allowed (e.g., "g", "l", "two"...). See section *One-Sided CIs* in the effectsize_CIs vignette.
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.
effectsize_type	The effect size of interest. Not that possibly not all effect sizes are applicable to the model object. See 'Details'. For Anova models, can also be a character vector with multiple effect size names.
keep	Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.
drop	See keep.
table_wide	Logical that decides whether the ANOVA table should be in wide format, i.e. should the numerator and denominator degrees of freedom be in the same row. Default: FALSE.

verbose — Toggle warnings and messages.

omega_squared, eta_squared, epsilon_squared — Deprecated. Please use effectsize_type.

... — Arguments passed to effectsize::effectsize(). For example, to calculate *partial* effect sizes types, use partial = TRUE. For objects of class htest or BFBayesFactor, adjust = TRUE can be used to return bias-corrected effect sizes, which is advisable for small samples and large tables. See also ?effectsize::eta_squared for arguments partial and generalized; ?effectsize::phi for adjust; and ?effectsize::oddratio for log.

Details

- For an object of class htest, data is extracted via insight::get_data(), and passed to the relevant function according to:
  - A **t-test** depending on type: "cohens_d" (default), "hedges_g", or one of "p_superiority", "u1", "u2", "u3", "overlap".
  - A **Chi-squared tests of independence** or **Fisher's Exact Test**, depending on type: "cramers_v" (default), "tschuprows_t", "phi", "cohens_w", "pearsons_c", "cohens_h", "oddsratio", "riskratio", "arr", or "nnt".
  - A **Chi-squared tests of goodness-of-fit**, depending on type: "fei" (default) "cohens_w", "pearsons_c"
  - A **One-way ANOVA test**, depending on type: "eta" (default), "omega" or "epsilon" -squared, "f", or "f2".
  - A **McNemar test** returns *Cohen's g*.
  - A **Wilcoxon test** depending on type: returns "rank_biserial" correlation (default) or one of "p_superiority", "vda", "u2", "u3", "overlap".
  - A **Kruskal-Wallis test** depending on type: "epsilon" (default) or "eta".
  - A **Friedman test** returns *Kendall's W*. (Where applicable, ci and alternative are taken from the htest if not otherwise provided.)
- For an object of class BFBayesFactor, using bayestestR::describe_posterior(),
  - A **t-test** depending on type: "cohens_d" (default) or one of "p_superiority", "u1", "u2", "u3", "overlap".
  - A **correlation test** returns *r*.
  - A **contingency table test**, depending on type: "cramers_v" (default), "phi", "tschuprows_t", "cohens_w", "pearsons_c", "cohens_h", "oddsratio", or "riskratio", "arr", or "nnt".
  - A **proportion test** returns *p*.
- Objects of class anova, aov, aovlist or afex_aov, depending on type: "eta" (default), "omega" or "epsilon" -squared, "f", or "f2".
- Other objects are passed to parameters::standardize_parameters().

For statistical models it is recommended to directly use the listed functions, for the full range of options they provide.

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

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

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

model_parameters(model, effectsize_type = c("omega", "eta"), ci = 0.9)

model <- anova(lm(Sepal.Length ~ Sepal.Big, data = df))
model_parameters(model)
model_parameters(
  model,
  effectsize_type = c("omega", "eta", "epsilon"),
  alternative = "greater"
)

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


## Not run:
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,
  effectsize_type = "eta",
  ci = 0.9,
  df_error = dof_satterthwaite(mm)[2:3]
)

## End(Not run)
```

`model_parameters.befa` *Parameters from Bayesian Exploratory Factor Analysis*

Description

Format Bayesian Exploratory Factor Analysis objects from the BayesFM package.

Usage

```
## S3 method for class 'befa'
model_parameters(
  model,
  sort = FALSE,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  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"` (see `map_estimate()`), `"trimmed"` (which is just `mean(x, trim = threshold)`), `"mode"` or `"all"`.
`dispersion`	Logical, if `TRUE`, computes indices of dispersion related to the estimate(s) (`SD` and `MAD` for `mean` and `median`, respectively). Dispersion is not available for `"MAP"` or `"mode"` centrality indices.
`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 `"ETI"` (default, see `eti()`), `"HDI"` (see `hdi()`), `"BCI"` (see `bci()`), `"SPI"` (see `spi()`), 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 warnings.
`...`	Arguments passed to or from other methods.

Value

A data frame of loadings.

Examples

```
library(parameters)

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

---

model_parameters.BFBayesFactor

Parameters from BayesFactor objects

Description

Parameters from `BFBayesFactor` objects from {`BayesFactor`} package.

Usage

```
## S3 method for class 'BFBayesFactor'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  priors = TRUE,
  effectsize_type = NULL,
  include_proportions = FALSE,
  verbose = TRUE,
  cohens_d = NULL,
  cramers_v = NULL,
  ...
)
```

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" (see map_estimate()), "trimmed" (which is just mean(x, trim = threshold)), "mode" or "all".
dispersion	Logical, if TRUE, computes indices of dispersion related to the estimate(s) (SD and MAD for mean and median, respectively). Dispersion is not available for "MAP" or "mode" centrality indices.
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 "ETI" (default, see eti()), "HDI" (see hdi()), "BCI" (see bci()), "SPI" (see spi()), 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.
effectsize_type	The effect size of interest. Not that possibly not all effect sizes are applicable to the model object. See 'Details'. For Anova models, can also be a character vector with multiple effect size names.
include_proportions	Logical that decides whether to include posterior cell proportions/counts for Bayesian contingency table analysis (from BayesFactor::contingencyTableBF()). Defaults to FALSE, as this information is often redundant.
verbose	Toggle warnings and messages.
cohens_d, cramers_v	Deprecated. Please use effectsize_type.
...	Additional arguments to be passed to or from methods.

Details

The meaning of the extracted parameters:

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

Value

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

Examples

```
if (require("BayesFactor")) {
  # Bayesian t-test
  model <- ttestBF(x = rnorm(100, 1, 1))
  model_parameters(model)
  model_parameters(model, cohens_d = TRUE, ci = .9)

  # Bayesian contingency table analysis
  data(raceDolls)
  bf <- contingencyTableBF(raceDolls, sampleType = "indepMulti", fixedMargin = "cols")
  model_parameters(bf,
    centrality = "mean",
    dispersion = TRUE,
    verbose = FALSE,
    effectsize_type = "cramers_v"
  )
}
```

---

model_parameters.cgam *Parameters from Generalized Additive (Mixed) Models*

Description

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

Usage

```
## S3 method for class 'cgam'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "residual",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
```

```
)

## S3 method for class 'gam'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "residual",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'gamlss'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "residual",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'gamm'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  verbose = TRUE,
  ...
)

## S3 method for class 'Gam'
model_parameters(
  model,
  effectsize_type = NULL,
```

```
  df_error = NULL,
  type = NULL,
  table_wide = FALSE,
  verbose = TRUE,
  ...
)

## S3 method for class 'scam'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "residual",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'vgam'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "residual",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

model	A gam/gamm model.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
ci_method	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin", "hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confi-*

dence intervals and approximation of degrees of freedom in model_parameters() for further details. When ci_method=NULL, in most cases "wald" is used then.

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 NULL (default; no standardization), "refit" (for re-fitting the model on standardized data) or one of "basic", "posthoc", "smart", "pseudo". See 'Details' in standardize_parameters(). **Importantly**:

- The "refit" method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use standardize="basic" or standardize the data with datawizard::standardize(force=TRUE) *before* fitting the model.
- For mixed models, when using methods other than "refit", only the fixed effects will be standardized.
- Robust estimation (i.e., vcov set to a value other than NULL) 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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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 stats::p.adjust() for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from **emmeans**).

keep — Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.

drop — See keep.

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().

effectsize_type — The effect size of interest. Not that possibly not all effect sizes are applicable to the model object. See 'Details'. For Anova models, can also be a character vector with multiple effect size names.

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.)

table_wide — Logical that decides whether the ANOVA table should be in wide format, i.e. should the numerator and denominator degrees of freedom be in the same row. Default: FALSE.

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

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

Examples

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

model_parameters.cpglmm *Parameters from Mixed Models*

Description

Parameters from (linear) mixed models.

Usage

```
## S3 method for class 'cpglmm'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  ci_random = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  include_sigma = FALSE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'glmmTMB'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "wald",
  ci_random = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  component = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  wb_component = TRUE,
  summary = getOption("parameters_mixed_summary", FALSE),
  keep = NULL,
```

```
  drop = NULL,
  verbose = TRUE,
  include_sigma = FALSE,
  ...
)

## S3 method for class 'merMod'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  ci_random = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  wb_component = TRUE,
  summary = getOption("parameters_mixed_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  include_sigma = FALSE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)

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

## S3 method for class 'mixed'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "wald",
  ci_random = NULL,
```

```
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  component = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  wb_component = TRUE,
  summary = getOption("parameters_mixed_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  include_sigma = FALSE,
  ...
)

## S3 method for class 'MixMod'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "wald",
  ci_random = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  component = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  wb_component = TRUE,
  summary = getOption("parameters_mixed_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  include_sigma = FALSE,
  ...
)

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

```
  exponentiate = FALSE,
  include_sigma = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'lme'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  ci_random = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  wb_component = TRUE,
  summary = getOption("parameters_mixed_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  include_sigma = FALSE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)

## S3 method for class 'clmm2'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "scale"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
```

```
)

## S3 method for class 'clmm'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  ci_random = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  include_sigma = FALSE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'rlmerMod'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  ci_random = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  effects = "all",
  group_level = FALSE,
  exponentiate = FALSE,
  p_adjust = NULL,
  include_sigma = FALSE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'HLfit'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
```

```
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)
```

Arguments

model	A mixed model.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
ci_method	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin", "hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confidence intervals and approximation of degrees of freedom* in model_parameters() for further details. When ci_method=NULL, in most cases "wald" is used then.
ci_random	Logical, if TRUE, includes the confidence intervals for random effects parameters. Only applies if effects is not "fixed" and if ci is not NULL. Set ci_random = FALSE if computation of the model summary is too much time consuming. By default, ci_random = NULL, which uses a heuristic to guess if computation of confidence intervals for random effects is fast enough or not. For models with larger sample size and/or more complex random effects structures, confidence intervals will not be computed by default, for simpler models or fewer observations, confidence intervals will be included. Set explicitly to TRUE or FALSE to enforce or omit calculation of confidence intervals.
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 NULL (default; no standardization), "refit" (for re-fitting the model on standardized data) or one of "basic", "posthoc", "smart", "pseudo". See 'Details' in standardize_parameters(). **Importantly**: • The "refit" method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use standardize="basic" or standardize the data with datawizard::standardize(force=TRUE) *before* fitting the model. • For mixed models, when using methods other than "refit", only the fixed effects will be standardized.

- Robust estimation (i.e., vcov set to a value other than NULL) 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. If the calculation of random effects parameters takes too long, you may use effects = "fixed".

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 coefficients (and related confidence intervals). This is typical for logistic regression, or more generally speaking, for models with log or logit links. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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 stats::p.adjust() for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from **emmeans**).

include_sigma — Logical, if TRUE, includes the residual standard deviation. For mixed models, this is defined as the sum of the distribution-specific variance and the variance for the additive overdispersion term (see insight::get_variance() for details). Defaults to FALSE for mixed models due to the longer computation time.

keep — Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.

drop — See keep.

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().

component — Should all parameters, parameters for the conditional model, for the zero-inflation part of the model, or the dispersion model be returned? Applies to models with zero-inflation and/or dispersion 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 datawizard::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-inflation component of the model.

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

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

- A covariance matrix
- A function which returns a covariance matrix (e.g., stats::vcov())
- A string which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC.
  - Cluster-robust: "vcovCR", "CR0", "CR1", "CR1p", "CR1S", "CR2", "CR3". See ?clubSandwich::vcovCR.
  - Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See ?sandwich::vcovBS.
  - Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".

vcov_args — List of arguments to be passed to the function identified by the vcov argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., ?sandwich::vcovHAC) to see the list of available arguments.

Value

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

Confidence intervals for random effect variances

For models of class merMod and glmmTMB, confidence intervals for random effect variances can be calculated.

- For models of from package **lme4**, when ci_method is either "profile" or "boot", and effects is either "random" or "all", profiled resp. bootstrapped confidence intervals are computed for the random effects.

- For all other options of `ci_method`, and only when the **merDeriv** package is installed, confidence intervals for random effects are based on normal-distribution approximation, using the delta-method to transform standard errors for constructing the intervals around the log-transformed SD parameters. These are than back-transformed, so that random effect variances, standard errors and confidence intervals are shown on the original scale. Due to the transformation, the intervals are asymmetrical, however, they are within the correct bounds (i.e. no negative interval for the SD, and the interval for the correlations is within the range from -1 to +1).
- For models of class `glmmTMB`, confidence intervals for random effect variances always use a Wald t-distribution approximation.

Dispersion parameters in *glmmTMB*

For some models from package **glmmTMB**, both the dispersion parameter and the residual variance from the random effects parameters are shown. Usually, these are the same but presented on different scales, e.g.

```
model <- glmmTMB(Sepal.Width ~ Petal.Length + (1|Species), data = iris)
exp(fixef(model)$disp) # 0.09902987
sigma(model)^2         # 0.09902987
```

For models where the dispersion parameter and the residual variance are the same, only the residual variance is shown in the output.

Confidence intervals and approximation of degrees of freedom

There are different ways of approximating the degrees of freedom depending on different assumptions about the nature of the model and its sampling distribution. The `ci_method` argument modulates the method for computing degrees of freedom (df) that are used to calculate confidence intervals (CI) and the related p-values. Following options are allowed, depending on the model class:

Classical methods:

Classical inference is generally based on the **Wald method**. The Wald approach to inference computes a test statistic by dividing the parameter estimate by its standard error (Coefficient / SE), then comparing this statistic against a t- or normal distribution. This approach can be used to compute CIs and p-values.

`"wald"`:

- Applies to *non-Bayesian models*. For *linear models*, CIs computed using the Wald method (SE and a *t-distribution with residual df*); p-values computed using the Wald method with a *t-distribution with residual df*. For other models, CIs computed using the Wald method (SE and a *normal distribution*); p-values computed using the Wald method with a *normal distribution*.

`"normal"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a normal distribution.

`"residual"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a *t-distribution with residual df* when possible. If the residual df for a model cannot be determined, a normal distribution is used instead.

Methods for mixed models:

Compared to fixed effects (or single-level) models, determining appropriate df for Wald-based inference in mixed models is more difficult. See the R GLMM FAQ for a discussion.

Several approximate methods for computing df are available, but you should also consider instead using profile likelihood (`"profile"`) or bootstrap (`"boot"`) CIs and p-values instead.

`"satterthwaite"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with Satterthwaite df*); p-values computed using the Wald method with a *t-distribution with Satterthwaite df*.

`"kenward"`

- Applies to *linear mixed models*. CIs computed using the Wald method (*Kenward-Roger SE* and a *t-distribution with Kenward-Roger df*); p-values computed using the Wald method with *Kenward-Roger SE and t-distribution with Kenward-Roger df*.

`"ml1"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with m-l-1 approximated df*); p-values computed using the Wald method with a *t-distribution with m-l-1 approximated df*. See `ci_ml1()`.

`"betwithin"`

- Applies to *linear mixed models* and *generalized linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with between-within df*); p-values computed using the Wald method with a *t-distribution with between-within df*. See `ci_betwithin()`.

Likelihood-based methods:

Likelihood-based inference is based on comparing the likelihood for the maximum-likelihood estimate to the the likelihood for models with one or more parameter values changed (e.g., set to zero or a range of alternative values). Likelihood ratios for the maximum-likelihood and alternative models are compared to a χ-squared distribution to compute CIs and p-values.

`"profile"`

- Applies to *non-Bayesian models* of class `glm`, `polr`, `merMod` or `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using linear interpolation to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

`"uniroot"`

- Applies to *non-Bayesian models* of class `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using root finding to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

Methods for bootstrapped or Bayesian models:

Bootstrap-based inference is based on **resampling** and refitting the model to the resampled datasets. The distribution of parameter estimates across resampled datasets is used to approximate the parameter's sampling distribution. Depending on the type of model, several different methods for bootstrapping and constructing CIs and p-values from the bootstrap distribution are available.

For Bayesian models, inference is based on drawing samples from the model posterior distribution.

`"quantile"` (or `"eti"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *equal tailed intervals* using the quantiles of the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::eti()`.

`"hdi"`

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *highest density intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::hdi()`.

`"bci"` (or `"bcai"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *bias corrected and accelerated intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::bci()`.

`"si"`

- Applies to *Bayesian models* with proper priors. CIs computed as *support intervals* comparing the posterior samples against the prior samples; p-values are based on the *probability of direction*. See `bayestestR::si()`.

`"boot"`

- Applies to *non-Bayesian models* of class `merMod`. CIs computed using *parametric bootstrapping* (simulating data from the fitted model); p-values computed using the Wald method with a *normal-distribution)* (note: this might change in a future update!).

For all iteration-based methods other than `"boot"` (`"hdi"`, `"quantile"`, `"ci"`, `"eti"`, `"si"`, `"bci"`, `"bcai"`), p-values are based on the probability of direction (`bayestestR::p_direction()`), which is converted into a p-value using `bayestestR::pd_to_p()`.

Note

If the calculation of random effects parameters takes too long, you may use `effects = "fixed"`. There is also a `plot()`-method implemented in the **see**-package.

See Also

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

Examples

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

if (require("glmmTMB")) {
  data(Salamanders)
  model <- glmmTMB(
    count ~ spp + mined + (1 | site),
    ziformula = ~mined,
    family = poisson(),
    data = Salamanders
  )
  model_parameters(model, effects = "all")
}

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

---

model_parameters.dbscan

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

Description

Format cluster models obtained for example by `kmeans()`.

Usage

```
## S3 method for class 'dbscan'
model_parameters(model, data = NULL, clusters = NULL, ...)

## S3 method for class 'hclust'
model_parameters(model, data = NULL, clusters = NULL, ...)

## S3 method for class 'pvclust'
model_parameters(model, data = NULL, clusters = NULL, ci = 0.95, ...)

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

## S3 method for class 'hkmeans'
```

```
model_parameters(model, ...)

## S3 method for class 'Mclust'
model_parameters(model, data = NULL, clusters = NULL, ...)

## S3 method for class 'pam'
model_parameters(model, data = NULL, clusters = NULL, ...)
```

Arguments

model	Cluster model.
data	A data.frame.
clusters	A vector with clusters assignments (must be same length as rows in data).
...	Arguments passed to or from other methods.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).

Examples

```
# DBSCAN ---------------------------
if (require("dbscan", quietly = TRUE)) {
  model <- dbscan::dbscan(iris[1:4], eps = 1.45, minPts = 10)

  rez <- model_parameters(model, iris[1:4])
  rez

  # Get clusters
  predict(rez)

  # Clusters centers in long form
  attributes(rez)$means

  # Between and Total Sum of Squares
  attributes(rez)$Sum_Squares_Total
  attributes(rez)$Sum_Squares_Between

  # HDBSCAN
  model <- dbscan::hdbscan(iris[1:4], minPts = 10)
  model_parameters(model, iris[1:4])
}

#
# Hierarchical clustering (hclust) ----------------------------
data <- iris[1:4]
model <- hclust(dist(data))
clusters <- cutree(model, 3)

rez <- model_parameters(model, data, clusters)
rez
```

```
# Get clusters
predict(rez)

# Clusters centers in long form
attributes(rez)$means

# Between and Total Sum of Squares
attributes(rez)$Total_Sum_Squares
attributes(rez)$Between_Sum_Squares

#
# pvclust (finds "significant" clusters) ---------------------------
if (require("pvclust", quietly = TRUE)) {
  data <- iris[1:4]
  # NOTE: pvclust works on transposed data
  model <- pvclust::pvclust(datawizard::data_transpose(data, verbose = FALSE),
    method.dist = "euclidean",
    nboot = 50,
    quiet = TRUE
  )

  rez <- model_parameters(model, data, ci = 0.90)
  rez

  # Get clusters
  predict(rez)

  # Clusters centers in long form
  attributes(rez)$means

  # Between and Total Sum of Squares
  attributes(rez)$Sum_Squares_Total
  attributes(rez)$Sum_Squares_Between
}

## Not run:
#
# K-means ------------------------------
model <- kmeans(iris[1:4], centers = 3)
rez <- model_parameters(model)
rez

# Get clusters
predict(rez)

# Clusters centers in long form
attributes(rez)$means

# Between and Total Sum of Squares
attributes(rez)$Sum_Squares_Total
attributes(rez)$Sum_Squares_Between

## End(Not run)
```

```
## Not run:
#
# Hierarchical K-means (factoextra::hkclust) ----------------------
if (require("factoextra", quietly = TRUE)) {
  data <- iris[1:4]
  model <- factoextra::hkmeans(data, k = 3)

  rez <- model_parameters(model)
  rez

  # Get clusters
  predict(rez)

  # Clusters centers in long form
  attributes(rez)$means

  # Between and Total Sum of Squares
  attributes(rez)$Sum_Squares_Total
  attributes(rez)$Sum_Squares_Between
}

## End(Not run)
if (require("mclust", quietly = TRUE)) {
  model <- mclust::Mclust(iris[1:4], verbose = FALSE)
  model_parameters(model)
}
## Not run:
#
# K-Medoids (PAM and HPAM) ==============
if (require("cluster", quietly = TRUE)) {
  model <- cluster::pam(iris[1:4], k = 3)
  model_parameters(model)
}
if (require("fpc", quietly = TRUE)) {
  model <- fpc::pamk(iris[1:4], criterion = "ch")
  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

```
## Default S3 method:
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)

## S3 method for class 'glm'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  vcov = NULL,
  vcov_args = NULL,
  verbose = TRUE,
  ...
)

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

```
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)

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

## S3 method for class 'polr'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  vcov = NULL,
  vcov_args = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'negbin'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  vcov = NULL,
  vcov_args = NULL,
  verbose = TRUE,
```

```
  ...
)
```

Arguments

model	Model object.
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
ci_method	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin", "hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confidence intervals and approximation of degrees of freedom* in model_parameters() for further details. When ci_method=NULL, in most cases "wald" is used then.
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 NULL (default; no standardization), "refit" (for re-fitting the model on standardized data) or one of "basic", "posthoc", "smart", "pseudo". See 'Details' in standardize_parameters(). **Importantly**: • The "refit" method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use standardize="basic" or standardize the data with datawizard::standardize(force=TRUE) *before* fitting the model. • For mixed models, when using methods other than "refit", only the fixed effects will be standardized. • Robust estimation (i.e., vcov set to a value other than NULL) 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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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 stats::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).

keep — Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.

drop — See keep.

verbose — Toggle warnings and messages.

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

- A covariance matrix
- A function which returns a covariance matrix (e.g., stats::vcov())
- A string which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC.
  - Cluster-robust: "vcovCR", "CR0", "CR1", "CR1p", "CR1S", "CR2", "CR3". See ?clubSandwich::vcovCR.
  - Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See ?sandwich::vcovBS.
  - Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".

vcov_args — List of arguments to be passed to the function identified by the vcov argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., ?sandwich::vcovHAC) to see the list of available arguments.

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

Value

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

Confidence intervals and approximation of degrees of freedom

There are different ways of approximating the degrees of freedom depending on different assumptions about the nature of the model and its sampling distribution. The `ci_method` argument modulates the method for computing degrees of freedom (df) that are used to calculate confidence intervals (CI) and the related p-values. Following options are allowed, depending on the model class:

Classical methods:

Classical inference is generally based on the **Wald method**. The Wald approach to inference computes a test statistic by dividing the parameter estimate by its standard error (Coefficient / SE), then comparing this statistic against a t- or normal distribution. This approach can be used to compute CIs and p-values.

`"wald"`:

- Applies to *non-Bayesian models*. For *linear models*, CIs computed using the Wald method (SE and a *t-distribution with residual df*); p-values computed using the Wald method with a *t-distribution with residual df*. For other models, CIs computed using the Wald method (SE and a *normal distribution*); p-values computed using the Wald method with a *normal distribution*.

`"normal"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a normal distribution.

`"residual"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a *t-distribution with residual df* when possible. If the residual df for a model cannot be determined, a normal distribution is used instead.

Methods for mixed models:

Compared to fixed effects (or single-level) models, determining appropriate df for Wald-based inference in mixed models is more difficult. See the R GLMM FAQ for a discussion.

Several approximate methods for computing df are available, but you should also consider instead using profile likelihood (`"profile"`) or bootstrap (`"boot"`) CIs and p-values instead.

`"satterthwaite"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with Satterthwaite df*); p-values computed using the Wald method with a *t-distribution with Satterthwaite df*.

`"kenward"`

- Applies to *linear mixed models*. CIs computed using the Wald method (*Kenward-Roger SE* and a *t-distribution with Kenward-Roger df*); p-values computed using the Wald method with *Kenward-Roger SE and t-distribution with Kenward-Roger df*.

`"ml1"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with m-l-1 approximated df*); p-values computed using the Wald method with a *t-distribution with m-l-1 approximated df*. See `ci_ml1()`.

`"betwithin"`

- Applies to *linear mixed models* and *generalized linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with between-within df*); p-values computed using the Wald method with a *t-distribution with between-within df*. See `ci_betwithin()`.

Likelihood-based methods:

Likelihood-based inference is based on comparing the likelihood for the maximum-likelihood estimate to the the likelihood for models with one or more parameter values changed (e.g., set to zero or a range of alternative values). Likelihood ratios for the maximum-likelihood and alternative models are compared to a χ-squared distribution to compute CIs and p-values.

`"profile"`

- Applies to *non-Bayesian models* of class `glm`, `polr`, `merMod` or `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using linear interpolation to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

`"uniroot"`

- Applies to *non-Bayesian models* of class `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using root finding to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

Methods for bootstrapped or Bayesian models:

Bootstrap-based inference is based on **resampling** and refitting the model to the resampled datasets. The distribution of parameter estimates across resampled datasets is used to approximate the parameter's sampling distribution. Depending on the type of model, several different methods for bootstrapping and constructing CIs and p-values from the bootstrap distribution are available.

For Bayesian models, inference is based on drawing samples from the model posterior distribution.

`"quantile"` (or `"eti"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *equal tailed intervals* using the quantiles of the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::eti()`.

`"hdi"`

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *highest density intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::hdi()`.

`"bci"` (or `"bcai"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *bias corrected and accelerated intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::bci()`.

"si"

- Applies to *Bayesian models* with proper priors. CIs computed as *support intervals* comparing the posterior samples against the prior samples; p-values are based on the *probability of direction*. See bayestestR::si().

"boot"

- Applies to *non-Bayesian models* of class merMod. CIs computed using *parametric bootstrapping* (simulating data from the fitted model); p-values computed using the Wald method with a *normal-distribution)* (note: this might change in a future update!).

For all iteration-based methods other than "boot" ("hdi", "quantile", "ci", "eti", "si", "bci", "bcai"), p-values are based on the probability of direction (bayestestR::p_direction()), which is converted into a p-value using bayestestR::pd_to_p().

See Also

insight::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")

# robust, heteroskedasticity-consistent standard errors
model_parameters(model, vcov = "HC3")

model_parameters(model,
  vcov = "vcovCL",
  vcov_args = list(cluster = mtcars$cyl)
)

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

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

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

```
# bias-corrected logistic regression with penalized maximum likelihood
model <- glm(
  vs ~ wt + cyl,
  data = mtcars,
  family = "binomial",
  method = "brglmFit"
)
model_parameters(model)
```

`model_parameters.DirichletRegModel`
Parameters from multinomial or cumulative link models

Description

Parameters from multinomial or cumulative link models

Usage

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

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

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

```
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'clm2'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "scale"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)
```

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` Should all parameters, parameters for the conditional model, for the zero-inflation part of the model, or the dispersion model be returned? Applies to models with zero-inflation and/or dispersion component. `component` may be one of `"conditional"`, `"zi"`, `"zero-inflated"`, `"dispersion"` or `"all"` (default). May be abbreviated.

`standardize` The method used for standardizing the parameters. Can be `NULL` (default; no standardization), `"refit"` (for re-fitting the model on standardized data) or one of `"basic"`, `"posthoc"`, `"smart"`, `"pseudo"`. See 'Details' in `standardize_parameters()`. **Importantly**:

- The `"refit"` method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use `standardize="basic"` or standardize the data with `datawizard::standardize(force=TRUE)` *before* fitting the model.
- For mixed models, when using methods other than `"refit"`, only the fixed effects will be standardized.
- Robust estimation (i.e., `vcov` set to a value other than `NULL`) 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. It is also recommended to use `exponentiate = TRUE` for models with log-transformed response values. **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 `stats::p.adjust()` for details. Further possible adjustment methods are `"tukey"`, `"scheffe"`, `"sidak"` and `"none"` to explicitly disable adjustment for `emmGrid` objects (from **emmeans**).

`keep` Character containing a regular expression pattern that describes the parameters that should be included (for `keep`) or excluded (for `drop`) in the returned data frame. `keep` may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If `keep` is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in `keep` will be selected from the returned data frame (and vice versa, all parameter names matching `drop` will be excluded). Furthermore, if `keep` has more than one element, these will be merged with an `OR` operator into a regular expression pattern like this: `"(one|two|three)"`. If `keep` 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()`. Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the `$Parameter` column of the parameters table to get the exact parameter names.

`drop` See `keep`.

`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()`.

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

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

- A covariance matrix
- A function which returns a covariance matrix (e.g., `stats::vcov()`)
- A string which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"vcovHC"`, `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`.
  - Cluster-robust: `"vcovCR"`, `"CR0"`, `"CR1"`, `"CR1p"`, `"CR1S"`, `"CR2"`, `"CR3"`. See `?clubSandwich::vcovCR`.
  - Bootstrap: `"vcovBS"`, `"xy"`, `"residual"`, `"wild"`, `"mammen"`, `"webb"`. See `?sandwich::vcovBS`.
  - Other `sandwich` package functions: `"vcovHAC"`, `"vcovPC"`, `"vcovCL"`, `"vcovPL"`.

`vcov_args` List of arguments to be passed to the function identified by the `vcov` argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., `?sandwich::vcovHAC`) to see the list of available arguments.

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

`insight::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

```
## S3 method for class 'htest'
model_parameters(
  model,
  ci = 0.95,
  alternative = NULL,
  bootstrap = FALSE,
  effectsize_type = NULL,
  verbose = TRUE,
  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,
  ...
)

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

```
## S3 method for class 'coeftest'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "wald",
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

model	Object of class htest or pairwise.htest.
ci	Level of confidence intervals for effect size statistic. Currently only applies to objects from chisq.test() or oneway.test().
alternative	A character string specifying the alternative hypothesis; Controls the type of CI returned: "two.sided" (default, two-sided CI), "greater" or "less" (one-sided CI). Partial matching is allowed (e.g., "g", "l", "two"...). See section *One-Sided CIs* in the effectsize_CIs vignette.
bootstrap	Should estimates be bootstrapped?
effectsize_type	The effect size of interest. Not that possibly not all effect sizes are applicable to the model object. See 'Details'. For Anova models, can also be a character vector with multiple effect size names.
verbose	Toggle warnings and messages.
cramers_v, phi, cohens_g, standardized_d, hedges_g, omega_squared, eta_squared, epsilon_squared, rank_b	Deprecated. Please use effectsize_type.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like type or parallel are passed down to bootstrap_model().
ci_method	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin", "hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confidence intervals and approximation of degrees of freedom* in model_parameters() for further details. When ci_method=NULL, in most cases "wald" is used then.
keep	Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a

regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.

drop See keep.

Details

- For an object of class htest, data is extracted via insight::get_data(), and passed to the relevant function according to:
  - A **t-test** depending on type: "cohens_d" (default), "hedges_g", or one of "p_superiority", "u1", "u2", "u3", "overlap".
  - A **Chi-squared tests of independence** or **Fisher's Exact Test**, depending on type: "cramers_v" (default), "tschuprows_t", "phi", "cohens_w", "pearsons_c", "cohens_h", "oddsratio", "riskratio", "arr", or "nnt".
  - A **Chi-squared tests of goodness-of-fit**, depending on type: "fei" (default) "cohens_w", "pearsons_c"
  - A **One-way ANOVA test**, depending on type: "eta" (default), "omega" or "epsilon" -squared, "f", or "f2".
  - A **McNemar test** returns *Cohen's g*.
  - A **Wilcoxon test** depending on type: returns "rank_biserial" correlation (default) or one of "p_superiority", "vda", "u2", "u3", "overlap".
  - A **Kruskal-Wallis test** depending on type: "epsilon" (default) or "eta".
  - A **Friedman test** returns *Kendall's W*. (Where applicable, ci and alternative are taken from the htest if not otherwise provided.)
- For an object of class BFBayesFactor, using bayestestR::describe_posterior(),
  - A **t-test** depending on type: "cohens_d" (default) or one of "p_superiority", "u1", "u2", "u3", "overlap".
  - A **correlation test** returns *r*.
  - A **contingency table test**, depending on type: "cramers_v" (default), "phi", "tschuprows_t", "cohens_w", "pearsons_c", "cohens_h", "oddsratio", or "riskratio", "arr", or "nnt".
  - A **proportion test** returns *p*.
- Objects of class anova, aov, aovlist or afex_aov, depending on type: "eta" (default), "omega" or "epsilon" -squared, "f", or "f2".
- Other objects are passed to parameters::standardize_parameters().

For statistical models it is recommended to directly use the listed functions, for the full range of options they provide.

Value

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

Examples

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

model <- t.test(iris$Sepal.Width, iris$Sepal.Length)
model_parameters(model, effectsize_type = "hedges_g")

model <- t.test(mtcars$mpg ~ mtcars$vs)
model_parameters(model, effectsize_type = "hedges_g")

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

data(airquality)
airquality$Month <- factor(airquality$Month, labels = month.abb[5:9])
model <- pairwise.t.test(airquality$Ozone, airquality$Month)
model_parameters(model)

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

model <- suppressWarnings(chisq.test(table(mtcars$am, mtcars$cyl)))
model_parameters(model, effectsize_type = "cramers_v")
```

---

model_parameters.MCMCglmm

Parameters from Bayesian Models

Description

Parameters from Bayesian models.

Usage

```
## S3 method for class 'MCMCglmm'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  bf_prior = NULL,
```

```
  diagnostic = c("ESS", "Rhat"),
  priors = TRUE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'bamlss'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  component = "all",
  exponentiate = FALSE,
  standardize = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'data.frame'
model_parameters(model, as_draws = FALSE, verbose = TRUE, ...)

## S3 method for class 'bayesQR'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  bf_prior = NULL,
  diagnostic = c("ESS", "Rhat"),
  priors = TRUE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)
```

```
## S3 method for class 'brmsfit'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  bf_prior = NULL,
  diagnostic = c("ESS", "Rhat"),
  priors = FALSE,
  effects = "fixed",
  component = "all",
  exponentiate = FALSE,
  standardize = NULL,
  group_level = FALSE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'mcmc.list'
model_parameters(model, as_draws = FALSE, verbose = TRUE, ...)

## S3 method for class 'bcplm'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  bf_prior = NULL,
  diagnostic = c("ESS", "Rhat"),
  priors = TRUE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'blrm'
```

```
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  bf_prior = NULL,
  diagnostic = c("ESS", "Rhat"),
  priors = TRUE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'draws'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'stanfit'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  diagnostic = c("ESS", "Rhat"),
  effects = "fixed",
  exponentiate = FALSE,
  standardize = NULL,
```

```
  group_level = FALSE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'stanreg'
model_parameters(
  model,
  centrality = "median",
  dispersion = FALSE,
  ci = 0.95,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  bf_prior = NULL,
  diagnostic = c("ESS", "Rhat"),
  priors = TRUE,
  effects = "fixed",
  exponentiate = FALSE,
  standardize = NULL,
  group_level = FALSE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

model	Bayesian model (including SEM from **blavaan**. May also be a data frame with posterior samples, however, as_draws must be set to TRUE (else, for data frames NULL is returned).
centrality	The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: "median", "mean", "MAP" (see map_estimate()), "trimmed" (which is just mean(x, trim = threshold)), "mode" or "all".
dispersion	Logical, if TRUE, computes indices of dispersion related to the estimate(s) (SD and MAD for mean and median, respectively). Dispersion is not available for "MAP" or "mode" centrality indices.
ci	Credible Interval (CI) level. Default to 0.95 (95%). See bayestestR::ci() for further details.
ci_method	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin",

	"hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confidence intervals and approximation of degrees of freedom* in model_parameters() for further details. When ci_method=NULL, in most cases "wald" is used then.
test	The indices of effect existence to compute. Character (vector) or list with one or more of these options: "p_direction" (or "pd"), "rope", "p_map", "equivalence_test" (or "equitest"), "bayesfactor" (or "bf") or "all" to compute all tests. For each "test", the corresponding **bayestestR** function is called (e.g. rope() or p_direction()) and its results included in the summary output.
rope_range	ROPE's lower and higher bounds. Should be a list of two values (e.g., c(-0.1, 0.1)) or "default". If "default", the bounds are set to x +- 0.1*SD(response).
rope_ci	The Credible Interval (CI) probability, corresponding to the proportion of HDI, to use for the percentage in ROPE.
bf_prior	Distribution representing a prior for the computation of Bayes factors / SI. Used if the input is a posterior, otherwise (in the case of models) ignored.
diagnostic	Diagnostic metrics to compute. Character (vector) or list with one or more of these options: "ESS", "Rhat", "MCSE" or "all".
priors	Add the prior used for each parameter.
keep	Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.
drop	See keep.
verbose	Toggle messages and warnings.
...	Currently not used.
component	Which type of parameters to return, such as parameters for the conditional model, the zero-inflation part of the model, the dispersion term, or other auxiliary parameters be returned? Applies to models with zero-inflation and/or dispersion formula, or if parameters such as sigma should be included. May be abbreviated. Note that the *conditional* component is also called *count* or *mean* component, depending on the model. There are three convenient shortcuts: component = "all" returns all possible parameters. If component = "location", location parameters such as conditional, zero_inflated, or smooth_terms, are returned (everything that are fixed or random effects - depending on the

effects argument - but no auxiliary parameters). For component = "distributional" (or "auxiliary"), components like sigma, dispersion, or beta (and other auxiliary parameters) are returned.

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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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.

standardize — The method used for standardizing the parameters. Can be NULL (default; no standardization), "refit" (for re-fitting the model on standardized data) or one of "basic", "posthoc", "smart", "pseudo". See 'Details' in standardize_parameters(). **Importantly**:

- The "refit" method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use standardize="basic" or standardize the data with datawizard::standardize(force=TRUE) *before* fitting the model.
- For mixed models, when using methods other than "refit", only the fixed effects will be standardized.
- Robust estimation (i.e., vcov set to a value other than NULL) of standardized parameters only works when standardize="refit".

as_draws — Logical, if TRUE and model is of class data.frame, the data frame is treated as posterior samples and handled similar to Bayesian models. All arguments in ... are passed to model_parameters.draws().

effects — Should results for fixed effects, random effects or both 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.

Value

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

Confidence intervals and approximation of degrees of freedom

There are different ways of approximating the degrees of freedom depending on different assumptions about the nature of the model and its sampling distribution. The ci_method argument modulates the method for computing degrees of freedom (df) that are used to calculate confidence intervals (CI) and the related p-values. Following options are allowed, depending on the model class:

Classical methods:

Classical inference is generally based on the **Wald method**. The Wald approach to inference computes a test statistic by dividing the parameter estimate by its standard error (Coefficient / SE), then comparing this statistic against a t- or normal distribution. This approach can be used to compute CIs and p-values.

`"wald"`:

- Applies to *non-Bayesian models*. For *linear models*, CIs computed using the Wald method (SE and a *t-distribution with residual df*); p-values computed using the Wald method with a *t-distribution with residual df*. For other models, CIs computed using the Wald method (SE and a *normal distribution*); p-values computed using the Wald method with a *normal distribution*.

`"normal"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a normal distribution.

`"residual"`

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a *t-distribution with residual df* when possible. If the residual df for a model cannot be determined, a normal distribution is used instead.

Methods for mixed models:

Compared to fixed effects (or single-level) models, determining appropriate df for Wald-based inference in mixed models is more difficult. See the R GLMM FAQ for a discussion.

Several approximate methods for computing df are available, but you should also consider instead using profile likelihood (`"profile"`) or bootstrap (`"boot"`) CIs and p-values instead.

`"satterthwaite"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with Satterthwaite df*); p-values computed using the Wald method with a *t-distribution with Satterthwaite df*.

`"kenward"`

- Applies to *linear mixed models*. CIs computed using the Wald method (*Kenward-Roger SE* and a *t-distribution with Kenward-Roger df*); p-values computed using the Wald method with *Kenward-Roger SE and t-distribution with Kenward-Roger df*.

`"ml1"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with m-l-1 approximated df*); p-values computed using the Wald method with a *t-distribution with m-l-1 approximated df*. See `ci_ml1()`.

`"betwithin"`

- Applies to *linear mixed models* and *generalized linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with between-within df*); p-values computed using the Wald method with a *t-distribution with between-within df*. See `ci_betwithin()`.

Likelihood-based methods:

Likelihood-based inference is based on comparing the likelihood for the maximum-likelihood estimate to the the likelihood for models with one or more parameter values changed (e.g., set to zero or a range of alternative values). Likelihood ratios for the maximum-likelihood and alternative models are compared to a χ-squared distribution to compute CIs and p-values.

`"profile"`

- Applies to *non-Bayesian models* of class `glm`, `polr`, `merMod` or `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using linear interpolation to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

`"uniroot"`

- Applies to *non-Bayesian models* of class `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using root finding to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

Methods for bootstrapped or Bayesian models:

Bootstrap-based inference is based on **resampling** and refitting the model to the resampled datasets. The distribution of parameter estimates across resampled datasets is used to approximate the parameter's sampling distribution. Depending on the type of model, several different methods for bootstrapping and constructing CIs and p-values from the bootstrap distribution are available.

For Bayesian models, inference is based on drawing samples from the model posterior distribution.

`"quantile"` (or `"eti"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *equal tailed intervals* using the quantiles of the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::eti()`.

`"hdi"`

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *highest density intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::hdi()`.

`"bci"` (or `"bcai"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *bias corrected and accelerated intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::bci()`.

`"si"`

- Applies to *Bayesian models* with proper priors. CIs computed as *support intervals* comparing the posterior samples against the prior samples; p-values are based on the *probability of direction*. See `bayestestR::si()`.

"boot"

- Applies to *non-Bayesian models* of class merMod. CIs computed using *parametric bootstrapping* (simulating data from the fitted model); p-values computed using the Wald method with a *normal-distribution)* (note: this might change in a future update!).

For all iteration-based methods other than "boot" ("hdi", "quantile", "ci", "eti", "si", "bci", "bcai"), p-values are based on the probability of direction (bayestestR::p_direction()), which is converted into a p-value using bayestestR::pd_to_p().

Note

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

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

See Also

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

Examples

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

## End(Not run)
```

---

model_parameters.mipo *Parameters from multiply imputed repeated analyses*

Description

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

Usage

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

```
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

Arguments

model	An object of class mira or mipo.
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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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 stats::p.adjust() for details. Further possible adjustment methods are "tukey", "scheffe", "sidak" and "none" to explicitly disable adjustment for emmGrid objects (from **emmeans**).
keep	Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.
drop	See keep.
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

```
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)
}
## Not run:
# model_parameters() also works for models that have no "tidy"-method in mice
if (require("mice", quietly = TRUE) && require("gee", quietly = TRUE)) {
  data(warpbreaks)
  set.seed(1234)
  warpbreaks$tension[sample(1:nrow(warpbreaks), size = 10)] <- NA
  imp <- mice(warpbreaks)
  fit <- with(data = imp, expr = gee(breaks ~ tension, id = wool))

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

  model_parameters(fit)
}

## End(Not run)



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

  model_parameters(pooled)
```

```
}
```

`model_parameters.PCA` *Parameters from PCA, FA, CFA, SEM*

Description

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

Usage

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

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

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

## S3 method for class 'sem'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)
```

Arguments

model	Model object.
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.
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").

keep — Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.

drop — See keep.

ci_method — Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin", "hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confidence intervals and approximation of degrees of freedom* in model_parameters() for further details. When ci_method=NULL, in most cases "wald" is used then.

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.

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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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 stats::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).

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

- A covariance matrix
- A function which returns a covariance matrix (e.g., stats::vcov())
- A string which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC.
  - Cluster-robust: "vcovCR", "CR0", "CR1", "CR1p", "CR1S", "CR2", "CR3". See ?clubSandwich::vcovCR.
  - Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See ?sandwich::vcovBS.
  - Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".

vcov_args — List of arguments to be passed to the function identified by the vcov argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., ?sandwich::vcovHAC) to see the list of available arguments.

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 indices or loadings.

Note

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

References

- Kaiser, H.F. and Rice. J. (1974). Little jiffy, mark iv. Educational and Psychological Measurement, 34(1):111–117
- Pettersson, E., and Turkheimer, E. (2010). Item selection, evaluation, and simple structure in personality data. Journal of research in personality, 44(4), 407-420.
- Revelle, W. (2016). How To: Use the psych package for Factor Analysis and data reduction.
- Tabachnick, B. G., and Fidell, L. S. (2013). Using multivariate statistics (6th ed.). Boston: Pearson Education.
- Rosseel Y (2012). lavaan: An R Package for Structural Equation Modeling. Journal of Statistical Software, 48(2), 1-36.
- Merkle EC , Rosseel Y (2018). blavaan: Bayesian Structural Equation Models via Parameter Expansion. Journal of Statistical Software, 85(4), 1-30. http://www.jstatsoft.org/v85/i04/

Examples

```
library(parameters)
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))
  )


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


# lavaan
```

```
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.PMCMR

Parameters from special models

Description

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

Parameters from Hypothesis Testing.

Usage

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

## S3 method for class 'glimML'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("conditional", "random", "dispersion", "all"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'mle2'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
```

```
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  vcov = NULL,
  vcov_args = 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,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'bfsl'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "residual",
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'deltaMethod'
model_parameters(model, p_adjust = NULL, verbose = TRUE, ...)

## S3 method for class 'emmGrid'
model_parameters(
  model,
  ci = 0.95,
```

```
  centrality = "median",
  dispersion = FALSE,
  ci_method = "eti",
  test = "pd",
  rope_range = "default",
  rope_ci = 0.95,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'epi.2by2'
model_parameters(model, verbose = TRUE, ...)

## S3 method for class 'fitdistr'
model_parameters(model, exponentiate = FALSE, verbose = TRUE, ...)

## S3 method for class 'ggeffects'
model_parameters(model, keep = NULL, drop = NULL, verbose = TRUE, ...)

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

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

## S3 method for class 'ivFixed'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "wald",
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'ivprobit'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "wald",
  keep = NULL,
```

```
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'lmodel2'
model_parameters(
  model,
  ci = 0.95,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'logistf'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  vcov = NULL,
  vcov_args = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'marginaleffects'
```

```
model_parameters(model, ci = 0.95, exponentiate = FALSE, ...)

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

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

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

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

## S3 method for class 'predictions'
model_parameters(model, ci = 0.95, exponentiate = TRUE, ...)

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

## S3 method for class 'maxLik'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)

## S3 method for class 'maxim'
model_parameters(
```

```
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  vcov = NULL,
  vcov_args = NULL,
  ...
)

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

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

## S3 method for class 'meta_random'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "eti",
  exponentiate = FALSE,
  include_studies = TRUE,
  verbose = TRUE,
  ...
)

## S3 method for class 'meta_fixed'
model_parameters(
```

```
  model,
  ci = 0.95,
  ci_method = "eti",
  exponentiate = FALSE,
  include_studies = TRUE,
  verbose = TRUE,
  ...
)

## S3 method for class 'meta_bma'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "eti",
  exponentiate = FALSE,
  include_studies = TRUE,
  verbose = TRUE,
  ...
)

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

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

## S3 method for class 'negbinirr'
model_parameters(
```

```
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = TRUE,
  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,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'probitmfx'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
```

```
  iterations = 1000,
  component = c("all", "conditional", "marginal"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'betaor'
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 'betamfx'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "precision", "marginal"),
```

```
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'mjoint'
model_parameters(
  model,
  ci = 0.95,
  effects = "fixed",
  component = c("all", "conditional", "survival"),
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'model_fit'
model_parameters(
  model,
  ci = 0.95,
  effects = "fixed",
  component = "conditional",
  ci_method = "profile",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  verbose = TRUE,
  ...
)

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

```
)

## S3 method for class 'mvord'
model_parameters(
  model,
  ci = 0.95,
  component = c("all", "conditional", "thresholds", "correlation"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'rqss'
model_parameters(
  model,
  ci = 0.95,
  ci_method = "residual",
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'rqs'
model_parameters(
```

```
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'selection'
model_parameters(
  model,
  ci = 0.95,
  component = c("all", "selection", "outcome", "auxiliary"),
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

## S3 method for class 'mle'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = getOption("parameters_summary", FALSE),
  keep = NULL,
  drop = NULL,
  vcov = NULL,
  vcov_args = NULL,
  verbose = TRUE,
  ...
)
```

```
## S3 method for class 'systemfit'
model_parameters(
  model,
  ci = 0.95,
  ci_method = NULL,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  summary = FALSE,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

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

## S3 method for class 'med1way'
model_parameters(model, verbose = TRUE, ...)

## S3 method for class 'dep.effect'
model_parameters(model, keep = NULL, verbose = TRUE, ...)

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

Arguments

model	Object from WRS2 package.
...	Arguments passed to or from other methods.
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 NULL (default; no standardization), "refit" (for re-fitting the model on standardized data) or one of "basic", "posthoc", "smart", "pseudo". See 'Details' in standardize_parameters(). **Importantly**:

- The "refit" method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use standardize="basic" or standardize the data with datawizard::standardize(force=TRUE) *before* fitting the model.
- For mixed models, when using methods other than "refit", only the fixed effects will be standardized.
- Robust estimation (i.e., vcov set to a value other than NULL) 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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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 stats::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).

keep — Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a

regular expression pattern like this: `"(one|two|three)"`. If `keep` 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()`. Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the `$Parameter` column of the parameters table to get the exact parameter names.

`drop` See `keep`.

`verbose` Toggle warnings and messages.

`ci_method` Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): `"residual"`, `"normal"`, `"likelihood"`, `"satterthwaite"`, `"kenward"`, `"wald"`, `"profile"`, `"boot"`, `"uniroot"`, `"ml1"`, `"betwithin"`, `"hdi"`, `"quantile"`, `"ci"`, `"eti"`, `"si"`, `"bci"`, or `"bcai"`. See section *Confidence intervals and approximation of degrees of freedom* in `model_parameters()` for further details. When `ci_method=NULL`, in most cases `"wald"` is used then.

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

- A covariance matrix
- A function which returns a covariance matrix (e.g., `stats::vcov()`)
- A string which indicates the kind of uncertainty estimates to return.
  - Heteroskedasticity-consistent: `"vcovHC"`, `"HC"`, `"HC0"`, `"HC1"`, `"HC2"`, `"HC3"`, `"HC4"`, `"HC4m"`, `"HC5"`. See `?sandwich::vcovHC`.
  - Cluster-robust: `"vcovCR"`, `"CR0"`, `"CR1"`, `"CR1p"`, `"CR1S"`, `"CR2"`, `"CR3"`. See `?clubSandwich::vcovCR`.
  - Bootstrap: `"vcovBS"`, `"xy"`, `"residual"`, `"wild"`, `"mammen"`, `"webb"`. See `?sandwich::vcovBS`.
  - Other `sandwich` package functions: `"vcovHAC"`, `"vcovPC"`, `"vcovCL"`, `"vcovPL"`.

`vcov_args` List of arguments to be passed to the function identified by the `vcov` argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., `?sandwich::vcovHAC`) to see the list of available arguments.

`centrality` The point-estimates (centrality indices) to compute. Character (vector) or list with one or more of these options: `"median"`, `"mean"`, `"MAP"` (see `map_estimate()`), `"trimmed"` (which is just `mean(x, trim = threshold)`), `"mode"` or `"all"`.

`dispersion` Logical, if `TRUE`, computes indices of dispersion related to the estimate(s) (`SD` and `MAD` for `mean` and `median`, respectively). Dispersion is not available for `"MAP"` or `"mode"` centrality indices.

`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.

`effects` Should results for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.

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

Value

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

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

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

See Also

`insight::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)
}

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 <- suppressWarnings(
    kwAllPairsConoverTest(count ~ spray, data = InsectSprays)
  )
  model_parameters(model)
}

if (require("WRS2") && packageVersion("WRS2") >= "1.1.3") {
  model <- t1way(libido ~ dose, data = viagra)
  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

```
## S3 method for class 'rma'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  standardize = NULL,
  exponentiate = FALSE,
  include_studies = TRUE,
  keep = NULL,
  drop = NULL,
  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 `NULL` (default; no standardization), `"refit"` (for re-fitting the model on standardized data) or one of `"basic"`, `"posthoc"`, `"smart"`, `"pseudo"`. See 'Details' in `standardize_parameters()`. **Importantly**:

- The `"refit"` method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use `standardize="basic"` or standardize the data with `datawizard::standardize(force=TRUE)` *before* fitting the model.
- For mixed models, when using methods other than `"refit"`, only the fixed effects will be standardized.
- Robust estimation (i.e., `vcov` set to a value other than `NULL`) 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. It is also recommended to use `exponentiate = TRUE` for models with log-transformed response values. **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.

`keep` Character containing a regular expression pattern that describes the parameters that should be included (for `keep`) or excluded (for `drop`) in the returned data frame. `keep` may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If `keep` is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in `keep` will be selected from the returned data frame (and vice versa, all parameter names matching `drop` will be excluded). Furthermore, if `keep` has more than one element, these will be merged with an `OR` operator into a regular expression pattern like this: `"(one|two|three)"`. If `keep` 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()`. Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the `$Parameter` column of the parameters table to get the exact parameter names.

`drop` See `keep`.

`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()`.

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")
  d <<- dat
  model <- rma(yi, vi, mods = ~alloc, data = d, digits = 3, slab = author)
  model_parameters(model)
}

if (require("metaBMA", quietly = TRUE)) {
  data(towels)
  m <- suppressWarnings(meta_random(logOR, SE, study, data = towels))
  model_parameters(m)
}

## End(Not run)
```

---

model_parameters.zcpglm

Parameters from Zero-Inflated Models

Description

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

Usage

```
## 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,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  summary = getOption("parameters_summary", FALSE),
  verbose = TRUE,
  ...
)

## S3 method for class 'mhurdle'
model_parameters(
  model,
  ci = 0.95,
 component = c("all", "conditional", "zi", "zero_inflated", "infrequent_purchase", "ip",
    "auxiliary"),
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

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

```
## S3 method for class 'hurdle'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "zi", "zero_inflated"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  summary = getOption("parameters_summary", FALSE),
  verbose = TRUE,
  ...
)

## S3 method for class 'zerocount'
model_parameters(
  model,
  ci = 0.95,
  bootstrap = FALSE,
  iterations = 1000,
  component = c("all", "conditional", "zi", "zero_inflated"),
  standardize = NULL,
  exponentiate = FALSE,
  p_adjust = NULL,
  keep = NULL,
  drop = NULL,
  summary = getOption("parameters_summary", FALSE),
  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	Should all parameters, parameters for the conditional model, for the zero-inflation part of the model, or the dispersion model be returned? Applies to models with zero-inflation and/or dispersion component. component may be one of "conditional", "zi", "zero-inflated", "dispersion" or "all" (default). May be abbreviated.

`standardize` The method used for standardizing the parameters. Can be `NULL` (default; no standardization), `"refit"` (for re-fitting the model on standardized data) or one of `"basic"`, `"posthoc"`, `"smart"`, `"pseudo"`. See 'Details' in `standardize_parameters()`. **Importantly**:

- The `"refit"` method does *not* standardize categorical predictors (i.e. factors), which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). to mimic such behaviours, either use `standardize="basic"` or standardize the data with `datawizard::standardize(force=TRUE)` *before* fitting the model.
- For mixed models, when using methods other than `"refit"`, only the fixed effects will be standardized.
- Robust estimation (i.e., `vcov` set to a value other than `NULL`) 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. It is also recommended to use `exponentiate = TRUE` for models with log-transformed response values. **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 `stats::p.adjust()` for details. Further possible adjustment methods are `"tukey"`, `"scheffe"`, `"sidak"` and `"none"` to explicitly disable adjustment for `emmGrid` objects (from **emmeans**).

`keep` Character containing a regular expression pattern that describes the parameters that should be included (for `keep`) or excluded (for `drop`) in the returned data frame. `keep` may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If `keep` is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in `keep` will be selected from the returned data frame (and vice versa, all parameter names matching `drop` will be excluded). Furthermore, if `keep` has more than one element, these will be merged with an `OR` operator into a regular expression pattern like this: `"(one|two|three)"`. If `keep` 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()`. Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the `$Parameter` column of the parameters table to get the exact parameter names.

`drop` See `keep`.

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

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().

Value

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

See Also

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

Examples

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

---

n_clusters *Find number of clusters in your data*

Description

Similarly to n_factors() for factor / principal component analysis, n_clusters() is the main function to find out the optimal numbers of clusters present in the data based on the maximum consensus of a large number of methods.

Essentially, there exist many methods to determine the optimal number of clusters, each with pros and cons, benefits and limitations. The main n_clusters function proposes to run all of them, and find out the number of clusters that is suggested by the majority of methods (in case of ties, it will select the most parsimonious solution with fewer clusters).

Note that we also implement some specific, commonly used methods, like the Elbow or the Gap method, with their own visualization functionalities. See the examples below for more details.

Usage

```
n_clusters(
  x,
  standardize = TRUE,
  include_factors = FALSE,
  package = c("easystats", "NbClust", "mclust"),
  fast = TRUE,
  nbclust_method = "kmeans",
  n_max = 10,
  ...
```

```
)

n_clusters_elbow(
  x,
  standardize = TRUE,
  include_factors = FALSE,
  clustering_function = stats::kmeans,
  n_max = 10,
  ...
)

n_clusters_gap(
  x,
  standardize = TRUE,
  include_factors = FALSE,
  clustering_function = stats::kmeans,
  n_max = 10,
  gap_method = "firstSEmax",
  ...
)

n_clusters_silhouette(
  x,
  standardize = TRUE,
  include_factors = FALSE,
  clustering_function = stats::kmeans,
  n_max = 10,
  ...
)

n_clusters_dbscan(
  x,
  standardize = TRUE,
  include_factors = FALSE,
  method = c("kNN", "SS"),
  min_size = 0.1,
  eps_n = 50,
  eps_range = c(0.1, 3),
  ...
)

n_clusters_hclust(
  x,
  standardize = TRUE,
  include_factors = FALSE,
  distance_method = "correlation",
  hclust_method = "average",
  ci = 0.95,
```

```
  iterations = 100,
  ...
)
```

Arguments

x	A data frame.
standardize	Standardize the dataframe before clustering (default).
include_factors	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 "easystats", "NbClust", "mclust", 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.
nbclust_method	The clustering method (passed to NbClust::NbClust() as method).
n_max	Maximal number of clusters to test.
...	Arguments passed to or from other methods. For instance, when bootstrap = TRUE, arguments like type or parallel are passed down to bootstrap_model().
clustering_function, gap_method	Other arguments passed to other functions. clustering_function is used by fviz_nbclust() and can be kmeans, cluster::pam, cluster::clara, cluster::fanny, and more. gap_method is used by cluster::maxSE to extract the optimal numbers of clusters (see its method argument).
method, min_size, eps_n, eps_range	Arguments for DBSCAN algorithm.
distance_method	The distance method (passed to dist()). Used by algorithms relying on the distance matrix, such as hclust or dbscan.
hclust_method	The hierarchical clustering method (passed to hclust()).
ci	Confidence Interval (CI) level. Default to 0.95 (95%).
iterations	The number of bootstrap replicates. This only apply in the case of bootstrapped frequentist models.

Note

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

Examples

```
## Not run:
library(parameters)
```

```
# The main 'n_clusters' function ================================
if (require("mclust", quietly = TRUE) && require("NbClust", quietly = TRUE) &&
  require("cluster", quietly = TRUE) && require("see", quietly = TRUE)) {
  n <- n_clusters(iris[, 1:4], package = c("NbClust", "mclust")) # package can be "all"
  n
  summary(n)
  as.data.frame(n) # Duration is the time elapsed for each method in seconds
  plot(n)

  # The following runs all the method but it significantly slower
  # n_clusters(iris[1:4], standardize = FALSE, package = "all", fast = FALSE)
}

## End(Not run)


x <- n_clusters_elbow(iris[1:4])
x
as.data.frame(x)
plot(x)



#
# Gap method --------------------
if (require("see", quietly = TRUE) &&
  require("cluster", quietly = TRUE) &&
  require("factoextra", quietly = TRUE)) {
  x <- n_clusters_gap(iris[1:4])
  x
  as.data.frame(x)
  plot(x)
}


#
# Silhouette method -------------------------
if (require("factoextra", quietly = TRUE)) {
  x <- n_clusters_silhouette(iris[1:4])
  x
  as.data.frame(x)
  plot(x)
}


#
if (require("dbscan", quietly = TRUE)) {
  # DBSCAN method ------------------------
  # NOTE: This actually primarily estimates the 'eps' parameter, the number of
  # clusters is a side effect (it's the number of clusters corresponding to
  # this 'optimal' EPS parameter).
  x <- n_clusters_dbscan(iris[1:4], method = "kNN", min_size = 0.05) # 5 percent
  x
```

```
  head(as.data.frame(x))
  plot(x)

  x <- n_clusters_dbscan(iris[1:4], method = "SS", eps_n = 100, eps_range = c(0.1, 2))
  x
  head(as.data.frame(x))
  plot(x)
}


#
# hclust method ------------------------------
if (require("pvclust", quietly = TRUE)) {
  # iterations should be higher for real analyses
  x <- n_clusters_hclust(iris[1:4], iterations = 50, ci = 0.90)
  x
  head(as.data.frame(x), n = 10) # Print 10 first rows
  plot(x)
}
```

`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"),
  cor = NULL,
  safe = TRUE,
  n_max = NULL,
  ...
)

n_components(
  x,
  type = "PCA",
  rotation = "varimax",
```

```
  algorithm = "default",
  package = c("nFactors", "psych"),
  cor = NULL,
  safe = TRUE,
  ...
)
```

Arguments

x	A data frame.
type	Can be "FA" or "PCA", depending on what you want to do.
rotation	Only used for VSS (Very Simple Structure criterion, see psych::VSS()). The rotation to apply. Can be "none", "varimax", "quartimax", "bentlerT", "equamax", "varimin", "geominT" and "bifactor" for orthogonal rotations, and "promax", "oblimin", "simplimax", "bentlerQ", "geominQ", "biquartimin" and "cluster" for oblique transformations.
algorithm	Factoring method used by VSS. Can be "pa" for Principal Axis Factor Analysis, "minres" for minimum residual (OLS) factoring, "mle" for Maximum Likelihood FA and "pc" for Principal Components. "default" will select "minres" if type = "FA" and "pc" if type = "PCA".
package	Package from which respective methods are used. Can be "all" or a vector containing "nFactors", "psych", "PCDimension", "fit" or "EGAnet". Note that "fit" (which actually also relies on the psych package) and "EGAnet" can be very slow for bigger datasets. Thus, the default is c("nFactors", "psych"). You must have the respective packages installed for the methods to be used.
cor	An optional correlation matrix that can be used (note that the data must still be passed as the first argument). If NULL, will compute it by running cor() on the passed data.
safe	If TRUE, 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.
n_max	If set to a value (e.g., 10), will drop from the results all methods that suggest a higher number of components. The interpretation becomes 'from all the methods that suggested a number lower than n_max, the results are ...'.
...	Arguments passed to or from other methods.

Details

n_components() is actually an alias for 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

- Bartlett, M. S. (1950). Tests of significance in factor analysis. British Journal of statistical psychology, 3(2), 77-85.
- Bentler, P. M., & Yuan, K. H. (1996). Test of linear trend in eigenvalues of a covariance matrix with application to data analysis. British Journal of Mathematical and Statistical Psychology, 49(2), 299-312.
- Cattell, R. B. (1966). The scree test for the number of factors. Multivariate behavioral research, 1(2), 245-276.
- Finch, W. H. (2019). Using Fit Statistic Differences to Determine the Optimal Number of Factors to Retain in an Exploratory Factor Analysis. Educational and Psychological Measurement.
- Zoski, K. W., & Jurs, S. (1996). An objective counterpart to the visual scree test for factor analysis: The standard error scree. Educational and Psychological Measurement, 56(3), 443-451.
- Zoski, K., & Jurs, S. (1993). Using multiple regression to determine the number of factors to retain in factor analysis. Multiple Linear Regression Viewpoints, 20(1), 5-9.
- Nasser, F., Benson, J., & Wisenbaker, J. (2002). The performance of regression-based variations of the visual scree for determining the number of common factors. Educational and psychological measurement, 62(3), 397-419.
- Golino, H., Shi, D., Garrido, L. E., Christensen, A. P., Nieto, M. D., Sadana, R., & Thiyagarajan, J. A. (2018). Investigating the performance of Exploratory Graph Analysis and traditional techniques to identify the number of latent factors: A simulation and tutorial.
- Golino, H. F., & Epskamp, S. (2017). Exploratory graph analysis: A new approach for estimating the number of dimensions in psychological research. PloS one, 12(6), e0174035.
- Revelle, W., & Rocklin, T. (1979). Very simple structure: An alternative procedure for estimating the optimal number of interpretable factors. Multivariate Behavioral Research, 14(4), 403-414.
- Velicer, W. F. (1976). Determining the number of components from the matrix of partial correlations. Psychometrika, 41(3), 321-327.

Examples

```
library(parameters)
n_factors(mtcars, type = "PCA")

result <- n_factors(mtcars[1:5], type = "FA")
as.data.frame(result)
summary(result)
## Not run:
# Setting package = 'all' will increase the number of methods (but is slow)
n_factors(mtcars, type = "PCA", package = "all")
n_factors(mtcars, type = "FA", algorithm = "mle", package = "all")

## End(Not run)
```

`parameters_type` *Type of model parameters*

Description

In a regression model, the parameters do not all have the meaning. For instance, the intercept has to be interpreted as theoretical outcome value under some conditions (when predictors are set to 0), whereas other coefficients are to be interpreted as amounts of change. Others, such as interactions, represent changes in another of the parameter. The `parameters_type` function attempts to retrieve information and meaning of parameters. It outputs a dataframe of information for each parameters, such as the `Type` (whether the parameter corresponds to a factor or a numeric predictor, or whether it is a (regular) interaction or a nested one), the `Link` (whether the parameter can be interpreted as a mean value, the slope of an association or a difference between two levels) and, in the case of interactions, which other parameters is impacted by which parameter.

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

```
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. It is also recommended to use `exponentiate = TRUE` for models with log-transformed response values. **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. If the calculation of random effects parameters takes too long, you may use effects = "fixed".
component	Should all parameters, parameters for the conditional model, for the zero-inflation part of the model, or the dispersion model be returned? Applies to models with zero-inflation and/or dispersion component. component may be one of "conditional", "zi", "zero-inflated", "dispersion" or "all" (default). May be abbreviated.
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-inflation 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-inflation model parts would be combined. Therefore, the component argument defaults to "conditional" to avoid this.

Some model objects do not return standard errors (e.g. objects of class htest). For these models, no pooled confidence intervals nor p-values are returned.

References

Barnard, J. and Rubin, D.B. (1999). Small sample degrees of freedom with multiple imputation. Biometrika, 86, 948-955. Rubin, D.B. (1987). Multiple Imputation for Nonresponse in Surveys. New York: John Wiley and Sons.

Examples

```
# 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))
}
```

`predict.parameters_clusters` *Predict method for parameters_clusters objects*

Description

Predict method for parameters_clusters objects

Usage

```
## S3 method for class 'parameters_clusters'
predict(object, newdata = NULL, names = NULL, ...)
```

Arguments

`object`	a model object for which prediction is desired.
`newdata`	data.frame
`names`	character vector or list
`...`	additional arguments affecting the predictions produced.

`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,
  footer = NULL,
  digits = 2,
  ci_digits = digits,
  p_digits = 3,
```

```
  footer_digits = 3,
  show_sigma = FALSE,
  show_formula = FALSE,
  zap_small = FALSE,
  groups = NULL,
  column_width = NULL,
  ci_brackets = c("[", "]"),
  include_reference = FALSE,
  ...
)

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

Arguments

x, object	An object returned by model_parameters().
pretty_names	Can be TRUE, which will return "pretty" (i.e. more human readable) parameter names. Or "labels", in which case value and variable labels will be used as parameters names. The latter only works for "labelled" data, i.e. if the data used to fit the model had "label" and "labels" attributes. See also section *Global Options to Customize Messages when Printing*.
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	Determines which columns and and which layout columns are printed. There are three options for this argument: 1. Selecting columns by name or index select can be a character vector (or numeric index) of column names that should be printed. There are two pre-defined options for selecting columns: select = "minimal" prints coefficients, confidence intervals and p-values, while select = "short" prints coefficients, standard errors and p-values. 2. A string expression with layout pattern select is a string with "tokens" enclosed in braces. These tokens will be replaced by their associated columns, where the selected columns will be collapsed into one column. However, it is possible to create multiple columns as well. Following tokens are replaced by the related coefficients or statistics: {estimate}, {se}, {ci} (or {ci_low} and {ci_high}), {p} and {stars}. The token {ci} will be replaced by {ci_low}, {ci_high}. Furthermore, a | separates values into new cells/columns. If format = "html", a inserts a line break inside a cell. See 'Examples'. 3. A string indicating a pre-defined layout select can be one of the following string values, to create one of the following pre-defined column layouts: • "ci": Estimates and confidence intervals, no asterisks for p-values. This is equivalent to select = "{estimate} ({ci})".

- "se": Estimates and standard errors, no asterisks for p-values. This is equivalent to select = "{estimate} ({se})".
- "ci_p": Estimates, confidence intervals and asterisks for p-values. This is equivalent to select = "{estimate}{stars} ({ci})".
- "se_p": Estimates, standard errors and asterisks for p-values. This is equivalent to select = "{estimate}{stars} ({se})"..
- "ci_p2": Estimates, confidence intervals and numeric p-values, in two columns. This is equivalent to select = "{estimate} ({ci})|{p}".
- "se_p2": Estimate, standard errors and numeric p-values, in two columns. This is equivalent to select = "{estimate} ({se})|{p}".

For model_parameters(), glue-like syntax is still experimental in the case of more complex models (like mixed models) and may not return expected results.

caption — Table caption as string. If NULL, depending on the model, either a default caption or no table caption is printed. Use caption = "" to suppress the table caption.

footer — Can either be FALSE or an empty string (i.e. "") to suppress the footer, NULL to print the default footer, or a string. The latter will combine the string value with the default footer.

digits, ci_digits, p_digits — Number of digits for rounding or significant figures. May also be "signif" to return significant figures or "scientific" to return scientific notation. Control the number of digits by adding the value as suffix, e.g. digits = "scientific4" to have scientific notation with 4 decimal places, or digits = "signif5" for 5 significant figures (see also signif()).

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.

column_width — Width of table columns. Can be either NULL, a named numeric vector, or "fixed". If NULL, the width for each table column is adjusted to the minimum required width. If a named numeric vector, value names are matched against column names, and for each match, the specified width is used. If "fixed", and table is split into multiple components, columns across all table components are adjusted to have the same width.

ci_brackets — Logical, if TRUE (default), CI-values are encompassed in square brackets (else in parentheses).

include_reference
: Logical, if TRUE, the reference level of factors will be added to the parameters table. This is only relevant for models with categorical predictors. The coefficient for the reference level is always 0 (except when exponentiate = TRUE, then the coefficient will be 1), so this is just for completeness.

...
: 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.

Global Options to Customize Messages and Tables when Printing

The verbose argument can be used to display or silence messages and warnings for the different functions in the **parameters** package. However, some messages providing additional information can be displayed or suppressed using options():

- parameters_summary: options(parameters_summary = TRUE) will override the summary argument in model_parameters() and always show the model summary for non-mixed models.
- parameters_mixed_summary: options(parameters_mixed_summary = TRUE) will override the summary argument in model_parameters() for mixed models, and will then always show the model summary.
- parameters_cimethod: options(parameters_cimethod = TRUE) will show the additional information about the approximation method used to calculate confidence intervals and p-values. Set to FALSE to hide this message when printing model_parameters() objects.
- parameters_exponentiate: options(parameters_exponentiate = TRUE) will show the additional information on how to interpret coefficients of models with log-transformed response variables or with log-/logit-links when the exponentiate argument in model_parameters() is not TRUE. Set this option to FALSE to hide this message when printing model_parameters() objects.

There are further options that can be used to modify the default behaviour for printed outputs:

- parameters_labels: options(parameters_labels = TRUE) will use variable and value labels for pretty names, if data is labelled. If no labels available, default pretty names are used.
- parameters_interaction: options(parameters_interaction = <character>) will replace the interaction mark (by default, *) with the related character.
- parameters_select: options(parameters_select = <value>) will set the default for the select argument. See argument's documentation for available options.

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**, **ggeffects**, or **marginaleffects**. 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()`. See also `display()`.

Examples

```
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

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")
))



# custom column layouts ------

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

# custom style
result <- compare_parameters(lm1, lm2, select = "{estimate}{stars} ({se})")
print(result)

## Not run:
# custom style, in HTML
result <- compare_parameters(lm1, lm2, select = "{estimate}<br>({se})|{p}")
print_html(result)

## End(Not run)
```

`p_calibrate` *Calculate calibrated p-values.*

Description

Compute calibrated p-values that can be interpreted probabilistically, i.e. as posterior probability of H0 (given that H0 and H1 have equal prior probabilities).

Usage

```
p_calibrate(x, ...)

## Default S3 method:
p_calibrate(x, type = "frequentist", verbose = TRUE, ...)
```

Arguments

`x`	A numeric vector of p-values, or a regression model object.
`...`	Currently not used.
`type`	Type of calibration. Can be `"frequentist"` or `"bayesian"`. See 'Details'.
`verbose`	Toggle warnings.

Details

The Bayesian calibration, i.e. when `type = "bayesian"`, can be interpreted as the lower bound of the Bayes factor for H0 to H1, based on the data. The full Bayes factor would then require multiplying by the prior odds of H0 to H1. The frequentist calibration also has a Bayesian interpretation; it is the posterior probability of H0, assuming that H0 and H1 have equal prior probabilities of 0.5 each (*Sellke et al. 2001*).

The calibration only works for p-values lower than or equal to `1/e`.

Value

A data frame with p-values and calibrated p-values.

References

Thomas Sellke, M. J Bayarri and James O Berger (2001) Calibration of p Values for Testing Precise Null Hypotheses, The American Statistician, 55:1, 62-71, doi:10.1198/000313001300339950

Examples

```
model <- lm(mpg ~ wt + as.factor(gear) + am, data = mtcars)
p_calibrate(model, verbose = FALSE)
```

p_function *p-value or consonance function*

Description

Compute p-values and compatibility (confidence) intervals for statistical models, at different levels. This function is also called consonance function. It allows to see which estimates are compatible with the model at various compatibility levels. Use `plot()` to generate plots of the *p* resp. *consonance* function and compatibility intervals at different levels.

Usage

```
p_function(
  model,
  ci_levels = c(0.25, 0.5, 0.75, emph = 0.95),
  exponentiate = FALSE,
  effects = "fixed",
  component = "all",
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

consonance_function(
  model,
  ci_levels = c(0.25, 0.5, 0.75, emph = 0.95),
  exponentiate = FALSE,
  effects = "fixed",
  component = "all",
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)

confidence_curve(
  model,
  ci_levels = c(0.25, 0.5, 0.75, emph = 0.95),
  exponentiate = FALSE,
  effects = "fixed",
  component = "all",
  keep = NULL,
  drop = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

model	Statistical Model.
ci_levels	Vector of scalars, indicating the different levels at which compatibility intervals should be printed or plotted. In plots, these levels are highlighted by vertical lines. It is possible to increase thickness for one or more of these lines by providing a names vector, where the to be highlighted values should be named "emph", e.g ci_levels = c(0.25, 0.5, emph = 0.95).
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. It is also recommended to use exponentiate = TRUE for models with log-transformed response values. **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. If the calculation of random effects parameters takes too long, you may use effects = "fixed".
component	Should all parameters, parameters for the conditional model, for the zero-inflation part of the model, or the dispersion model be returned? Applies to models with zero-inflation and/or dispersion component. component may be one of "conditional", "zi", "zero-inflated", "dispersion" or "all" (default). May be abbreviated.
keep	Character containing a regular expression pattern that describes the parameters that should be included (for keep) or excluded (for drop) in the returned data frame. keep may also be a named list of regular expressions. All non-matching parameters will be removed from the output. If keep is a character vector, every parameter name in the *"Parameter"* column that matches the regular expression in keep will be selected from the returned data frame (and vice versa, all parameter names matching drop will be excluded). Furthermore, if keep has more than one element, these will be merged with an OR operator into a regular expression pattern like this: "(one|two|three)". If keep 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(). Note that the regular expression pattern should match the parameter names as they are stored in the returned data frame, which can be different from how they are printed. Inspect the $Parameter column of the parameters table to get the exact parameter names.
drop	See keep.
verbose	Toggle warnings and messages.

... 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. If s_value = TRUE, the p-value will be replaced by the S-value in the output (cf. *Rafi and Greenland 2020*). pd adds an additional column with the *probability of direction* (see bayestestR::p_direction() for details). groups can be used to group coefficients. It will be passed to the print-method, or can directly be used in print(), see documentation in print.parameters_model(). Furthermore, see 'Examples' in model_parameters.default(). For developers, whose interest mainly is to get a "tidy" data frame of model summaries, it is recommended to set pretty_names = FALSE to speed up computation of the summary table.

Details

Compatibility intervals and continuous *p*-values for different estimate values:

p_function() only returns the compatibility interval estimates, not the related *p*-values. The reason for this is because the *p*-value for a given estimate value is just 1 - CI_level. The values indicating the lower and upper limits of the intervals are the related estimates associated with the *p*-value. E.g., if a parameter x has a 75% compatibility interval of (0.81, 1.05), then the *p*-value for the estimate value of 0.81 would be 1 - 0.75, which is 0.25. This relationship is more intuitive and better to understand when looking at the plots (using plot()).

Conditional versus unconditional interpretation of *p*-values and intervals:

p_function(), and in particular its plot() method, aims at re-interpreting *p*-values and confidence intervals (better named: *compatibility* intervals) in *unconditional* terms. Instead of referring to the long-term property and repeated trials when interpreting interval estimates (so-called "aleatory probability", *Schweder 2018*), and assuming that all underlying assumptions are correct and met, p_function() interprets *p*-values in a Fisherian way as "*continuous* measure of evidence against the very test hypothesis *and* entire model (all assumptions) used to compute it" (*P-Values Are Tough and S-Values Can Help*, lesslikely.com/statistics/s-values; see also *Amrhein and Greenland 2022*).

This interpretation as a continuous measure of evidence against the test hypothesis and the entire model used to compute it can be seen in the figure below (taken from *P-Values Are Tough and S-Values Can Help*, lesslikely.com/statistics/s-values). The "conditional" interpretation of *p*-values and interval estimates (A) implicitly assumes certain assumptions to be true, thus the interpretation is "conditioned" on these assumptions (i.e. assumptions are taken as given). The unconditional interpretation (B), however, questions all these assumptions.

"Emphasizing unconditional interpretations helps avoid overconfident and misleading inferences in light of uncertainties about the assumptions used to arrive at the statistical results." (*Greenland et al. 2022*).

Note: The term "conditional" as used by Rafi and Greenland probably has a slightly different meaning than normally. "Conditional" in this notion means that all model assumptions are taken as given - it should not be confused with terms like "conditional probability". See also *Greenland et al. 2022* for a detailed elaboration on this issue.

In other words, the term compatibility interval emphasizes "the dependence of the *p*-value on the assumptions as well as on the data, recognizing that $p<0.05$ can arise from assumption violations even if the effect under study is null" (*Gelman/Greenland 2019*).

Probabilistic interpretation of compatibility intervals:

Schweder (2018) resp. Schweder and Hjort (2016) (and others) argue that confidence curves (as produced by `p_function()`) have a valid probabilistic interpretation. They distinguish between *aleatory probability*, which describes the aleatory stochastic element of a distribution *ex ante*, i.e. before the data are obtained. This is the classical interpretation of confidence intervals following the Neyman-Pearson school of statistics. However, there is also an *ex post* probability, called *epistemic* probability, for confidence curves. The shift in terminology from *confidence* intervals to *compatibility* intervals may help emphasizing this interpretation.

In this sense, the probabilistic interpretation of *p*-values and compatibility intervals is "conditional" - on the data *and* model assumptions (which is in line with the "unconditional" interpretation in the sense of Rafi and Greenland).

"The realized confidence distribution is clearly an epistemic probability distribution" (*Schweder 2018*). In Bayesian words, compatibility intervals (or confidence distributons, or consonance curves) are "posteriors without priors" (*Schweder, Hjort, 2003*). In this regard, interpretation of *p*-values might be guided using `bayestestR::p_to_pd()`.

Compatibility intervals - is their interpretation conditional or not?:

The fact that the term "conditional" is used in different meanings, is confusing and unfortunate. Thus, we would summarize the probabilistic interpretation of compatibility intervals as follows: The intervals are built from the data *and* our modeling assumptions. The accuracy of the intervals depends on our model assumptions. If a value is outside the interval, that might be because (1) that parameter value isn't supported by the data, or (2) the modeling assumptions are a poor fit for the situation. When we make bad assumptions, the compatibility interval might be too wide or (more commonly and seriously) too narrow, making us think we know more about the parameter than is warranted.

When we say "there is a 95% chance the true value is in the interval", that is a statement of *epistemic probability* (i.e. description of uncertainty related to our knowledge or belief). When we talk about repeated samples or sampling distributions, that is referring to *aleatoric* (physical properties) probability. Frequentist inference is built on defining estimators with known *aleatoric* probability properties, from which we can draw *epistemic* probabilistic statements of uncertainty (*Schweder and Hjort 2016*).

Value

A data frame with p-values and compatibility intervals.

Note

Curently, `p_function()` computes intervals based on Wald t- or z-statistic. For certain models (like mixed models), profiled intervals may be more accurate, however, this is currently not supported.

References

- Amrhein V, Greenland S. Discuss practical importance of results based on interval estimates and p-value functions, not only on point estimates and null p-values. Journal of Information Technology 2022;37:316–20. doi:10.1177/02683962221105904
- Fraser DAS. The P-value function and statistical inference. The American Statistician. 2019;73(sup1):135-147. doi:10.1080/00031305.2018.1556735

- Gelman A, Greenland S. Are confidence intervals better termed "uncertainty intervals"? BMJ (2019)l5381. doi:10.1136/bmj.l5381
- Greenland S, Rafi Z, Matthews R, Higgs M. To Aid Scientific Inference, Emphasize Unconditional Compatibility Descriptions of Statistics. (2022) https://arxiv.org/abs/1909.08583v7 (Accessed November 10, 2022)
- Rafi Z, Greenland S. Semantic and cognitive tools to aid statistical science: Replace confidence and significance by compatibility and surprise. BMC Medical Research Methodology. 2020;20(1):244. doi:10.1186/s12874020011059
- Schweder T. Confidence is epistemic probability for empirical science. Journal of Statistical Planning and Inference (2018) 195:116–125. doi:10.1016/j.jspi.2017.09.016
- Schweder T, Hjort NL. Confidence and Likelihood. Scandinavian Journal of Statistics. 2002;29(2):309-332. doi:10.1111/14679469.00285
- Schweder T, Hjort NL. Frequentist analogues of priors and posteriors. In Stigum, B. (ed.), Econometrics and the Philosophy of Economics: Theory Data Confrontation in Economics, pp. 285-217. Princeton University Press, Princeton, NJ, 2003
- Schweder T, Hjort NL. Confidence, Likelihood, Probability: Statistical inference with confidence distributions. Cambridge University Press, 2016.

Examples

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

if (requireNamespace("see") && packageVersion("see") > "0.7.3") {
  model <- lm(mpg ~ wt + as.factor(gear) + am, data = mtcars)
  result <- p_function(model)

  # single panels
  plot(result, n_columns = 2)

  # integrated plot, the default
  plot(result)
}
```

---

p_value *p-values*

Description

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

- 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,
  dof = NULL,
  method = NULL,
  component = "all",
  vcov = NULL,
  vcov_args = NULL,
  verbose = TRUE,
  ...
)

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

Arguments

model	A statistical model.
...	Additional arguments
dof	Number of degrees of freedom to be used when calculating confidence intervals. If NULL (default), the degrees of freedom are retrieved by calling degrees_of_freedom() with approximation method defined in method. If not NULL, use this argument to override the default degrees of freedom used to compute confidence intervals.
method	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin", "hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confidence intervals and approximation of degrees of freedom* in model_parameters() for further details.
component	Model component for which parameters should be shown. See the documentation for your object's class in model_parameters() or p_value() for further details.
vcov	Variance-covariance matrix used to compute uncertainty estimates (e.g., for robust standard errors). This argument accepts a covariance matrix, a function which returns a covariance matrix, or a string which identifies the function to be used to compute the covariance matrix. • A covariance matrix • A function which returns a covariance matrix (e.g., stats::vcov()) • A string which indicates the kind of uncertainty estimates to return. – Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC.

- Cluster-robust: "vcovCR", "CR0", "CR1", "CR1p", "CR1S", "CR2", "CR3". See ?clubSandwich::vcovCR.
- Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See ?sandwich::vcovBS.
- Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".

vcov_args — List of arguments to be passed to the function identified by the vcov argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., ?sandwich::vcovHAC) to see the list of available arguments.

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.

Confidence intervals and approximation of degrees of freedom

There are different ways of approximating the degrees of freedom depending on different assumptions about the nature of the model and its sampling distribution. The ci_method argument modulates the method for computing degrees of freedom (df) that are used to calculate confidence intervals (CI) and the related p-values. Following options are allowed, depending on the model class:

Classical methods:

Classical inference is generally based on the **Wald method**. The Wald approach to inference computes a test statistic by dividing the parameter estimate by its standard error (Coefficient / SE), then comparing this statistic against a t- or normal distribution. This approach can be used to compute CIs and p-values.

"wald":

- Applies to *non-Bayesian models*. For *linear models*, CIs computed using the Wald method (SE and a *t-distribution with residual df*); p-values computed using the Wald method with a *t-distribution with residual df*. For other models, CIs computed using the Wald method (SE and a *normal distribution*); p-values computed using the Wald method with a *normal distribution*.

"normal"

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a normal distribution.

"residual"

- Applies to *non-Bayesian models*. Compute Wald CIs and p-values, but always use a *t-distribution with residual df* when possible. If the residual df for a model cannot be determined, a normal distribution is used instead.

Methods for mixed models:

Compared to fixed effects (or single-level) models, determining appropriate df for Wald-based inference in mixed models is more difficult. See the R GLMM FAQ for a discussion.

Several approximate methods for computing df are available, but you should also consider instead using profile likelihood (`"profile"`) or bootstrap (`"boot"`) CIs and p-values instead.

`"satterthwaite"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with Satterthwaite df*); p-values computed using the Wald method with a *t-distribution with Satterthwaite df*.

`"kenward"`

- Applies to *linear mixed models*. CIs computed using the Wald method (*Kenward-Roger SE* and a *t-distribution with Kenward-Roger df*); p-values computed using the Wald method with *Kenward-Roger SE and t-distribution with Kenward-Roger df*.

`"ml1"`

- Applies to *linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with m-l-1 approximated df*); p-values computed using the Wald method with a *t-distribution with m-l-1 approximated df*. See `ci_ml1()`.

`"betwithin"`

- Applies to *linear mixed models* and *generalized linear mixed models*. CIs computed using the Wald method (SE and a *t-distribution with between-within df*); p-values computed using the Wald method with a *t-distribution with between-within df*. See `ci_betwithin()`.

Likelihood-based methods:

Likelihood-based inference is based on comparing the likelihood for the maximum-likelihood estimate to the the likelihood for models with one or more parameter values changed (e.g., set to zero or a range of alternative values). Likelihood ratios for the maximum-likelihood and alternative models are compared to a χ-squared distribution to compute CIs and p-values.

`"profile"`

- Applies to *non-Bayesian models* of class `glm`, `polr`, `merMod` or `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using linear interpolation to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

`"uniroot"`

- Applies to *non-Bayesian models* of class `glmmTMB`. CIs computed by *profiling the likelihood curve for a parameter*, using root finding to find where likelihood ratio equals a critical value; p-values computed using the Wald method with a *normal-distribution* (note: this might change in a future update!)

Methods for bootstrapped or Bayesian models:

Bootstrap-based inference is based on **resampling** and refitting the model to the resampled datasets. The distribution of parameter estimates across resampled datasets is used to approximate the parameter's sampling distribution. Depending on the type of model, several different methods for bootstrapping and constructing CIs and p-values from the bootstrap distribution are available.

For Bayesian models, inference is based on drawing samples from the model posterior distribution.

`"quantile"` (or `"eti"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *equal tailed intervals* using the quantiles of the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::eti()`.

`"hdi"`

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *highest density intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::hdi()`.

`"bci"` (or `"bcai"`)

- Applies to *all models (including Bayesian models)*. For non-Bayesian models, only applies if `bootstrap = TRUE`. CIs computed as *bias corrected and accelerated intervals* for the bootstrap or posterior samples; p-values are based on the *probability of direction*. See `bayestestR::bci()`.

`"si"`

- Applies to *Bayesian models* with proper priors. CIs computed as *support intervals* comparing the posterior samples against the prior samples; p-values are based on the *probability of direction*. See `bayestestR::si()`.

`"boot"`

- Applies to *non-Bayesian models* of class `merMod`. CIs computed using *parametric bootstrapping* (simulating data from the fitted model); p-values computed using the Wald method with a *normal-distribution)* (note: this might change in a future update!).

For all iteration-based methods other than `"boot"` (`"hdi"`, `"quantile"`, `"ci"`, `"eti"`, `"si"`, `"bci"`, `"bcai"`), p-values are based on the probability of direction (`bayestestR::p_direction()`), which is converted into a p-value using `bayestestR::pd_to_p()`.

Examples

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

p_value.BFBayesFactor *p-values for Bayesian Models*

Description

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

Usage

```
## S3 method for class 'BFBayesFactor'
p_value(model, ...)
```

Arguments

model	A statistical model.
...	Additional arguments

Details

For Bayesian models, the p-values corresponds to the *probability of direction* (bayestestR::p_direction()), which is converted to a p-value using bayestestR::convert_pd_to_p().

Value

The p-values.

Examples

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

p_value.DirichletRegModel *p-values for Models with Special Components*

Description

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

Usage

```
## 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"),
  verbose = TRUE,
  ...
)

## 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).
...	Additional arguments
verbose	Toggle warnings and messages.

Value

The p-values.

---

p_value.poissonmfx *p-values for Marginal Effects Models*

Description

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

Usage

```
## 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

```
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")
}
```

---

p_value.zcpglm *p-values for Models with Zero-Inflation*

Description

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

Usage

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

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

Arguments

model	A statistical model.
component	Model component for which parameters should be shown. See the documentation for your object's class in model_parameters() or p_value() for further details.
...	Additional arguments
method	Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): "residual", "normal", "likelihood", "satterthwaite", "kenward", "wald", "profile", "boot", "uniroot", "ml1", "betwithin", "hdi", "quantile", "ci", "eti", "si", "bci", or "bcai". See section *Confidence intervals and approximation of degrees of freedom* in model_parameters() for further details.
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 `datawizard::demean()` function. Health-related quality of life from cancer-patients was measured at three time points (pre-surgery, 6 and 12 months after surgery).

Format

A data frame with 564 rows and 7 variables:

ID Patient ID

QoL Quality of Life Score

time Timepoint of measurement

age Age in years

phq4 Patients' Health Questionnaire, 4-item version

hospital Hospital ID, where patient was treated

education Patients' educational level

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, for the zero-inflation part of the model, or the dispersion model be returned? Applies to models with zero-inflation and/or dispersion component. `component` may be one of `"conditional"`, `"zi"`, `"zero-inflated"`, `"dispersion"` or `"all"` (default). May be abbreviated.

Details

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

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

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

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

Random slope-intercept correlation:
The random slope-intercept correlation (ρ_{01}) is obtained from `VarCorr()`. This measure is only available for mixed models with random intercepts and slopes.
Note: For the within-group and between-group variance, variance and standard deviations (which are simply the square root of the variance) are shown.

Value

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

Examples

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

---

reduce_parameters *Dimensionality reduction (DR) / Features Reduction*

Description

This function performs a reduction in the 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).
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

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

Examples

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

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

---

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

```
if (require("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, ...)
```

Arguments

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

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 `cAIC4::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.

Value

The model refitted with optimal number of parameters.

Examples

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

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

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

---

simulate_model *Simulated draws from model coefficients*

Description

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

Usage

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

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

Arguments

model	Statistical model (no Bayesian models).
iterations	The number of draws to simulate/bootstrap.
...	Arguments passed to insight::get_varcov(), e.g. to allow simulated draws to be based on heteroscedasticity consistent variance covariance matrices.

component — Should all parameters, parameters for the conditional model, for the zero-inflation part of the model, or the dispersion model be returned? Applies to models with zero-inflation and/or dispersion 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::mvrnorm()) with mean mu = coef(model) and variance Sigma = vcov(model).

Models with Zero-Inflation Component:

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

Value

A data frame.

See Also

simulate_parameters(), bootstrap_model(), bootstrap_parameters()

Examples

```
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

```
## 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",
  ...
)
```

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" (see map_estimate()), "trimmed" (which is just mean(x, trim = threshold)), "mode" or "all".
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 "ETI" (default, see eti()), "HDI" (see hdi()), "BCI" (see bci()), "SPI" (see spi()), 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 insight::get_varcov(), e.g. to allow simulated draws to be based on heteroscedasticity consistent variance covariance matrices.

Details

Technical Details:

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

Models with Zero-Inflation Component:

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

Value

A data frame with simulated parameters.

Note

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

References

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

See Also

bootstrap_model(), bootstrap_parameters(), simulate_model()

Examples

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

## Not run:
if (require("glmmTMB", quietly = TRUE)) {
  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)
```

`sort_parameters` *Sort parameters by coefficient values*

Description

Sort parameters by coefficient values

Usage

```
sort_parameters(x, ...)

## Default S3 method:
sort_parameters(x, sort = "none", column = "Coefficient", ...)
```

Arguments

`x`	A data frame or a `parameters_model` object.
`...`	Arguments passed to or from other methods.
`sort`	If `"none"` (default) do not sort, `"ascending"` sort by increasing coefficient value, or `"descending"` sort by decreasing coefficient value.
`column`	The column containing model parameter estimates. This will be `"Coefficient"` (default) in *easystats* packages, `"estimate"` in *broom* package, etc.

Value

A sorted data frame or original object.

Examples

```
# creating object to sort (can also be a regular data frame)
mod <- model_parameters(stats::lm(wt ~ am * cyl, data = mtcars))

# original output
mod

# sorted outputs
sort_parameters(mod, sort = "ascending")
sort_parameters(mod, sort = "descending")
```

`standardize_info` *Get Standardization Information*

Description

This function extracts information, such as the deviations (SD or MAD) from parent variables, that are necessary for post-hoc standardization of parameters. This function gives a window on how standardized are obtained, i.e., by what they are divided. The "basic" method of standardization uses.

Usage

```
standardize_info(model, ...)

## Default S3 method:
standardize_info(
  model,
  robust = FALSE,
  two_sd = FALSE,
  include_pseudo = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`model`	A statistical model.
`...`	Arguments passed to or from other methods.
`robust`	Logical, if `TRUE`, centering is done by subtracting the median from the variables and dividing it by the median absolute deviation (MAD). If `FALSE`, variables are standardized by subtracting the mean and dividing it by the standard deviation (SD).
`two_sd`	If `TRUE`, the variables are scaled by two times the deviation (SD or MAD depending on `robust`). This method can be useful to obtain model coefficients of continuous parameters comparable to coefficients related to binary predictors, when applied to **the predictors** (not the outcome) (Gelman, 2008).
`include_pseudo`	(For (G)LMMs) Should Pseudo-standardized information be included?
`verbose`	Toggle warnings and messages on or off.

Value

A data frame with information on each parameter (see `parameters_type()`), and various standardization coefficients for the post-hoc methods (see `standardize_parameters()`) for the predictor and the response.

See Also

Other standardize: standardize_parameters()

Examples

```
model <- lm(mpg ~ ., data = mtcars)
standardize_info(model)
standardize_info(model, robust = TRUE)
standardize_info(model, two_sd = TRUE)
```

---

standardize_parameters

Parameters standardization

Description

Compute standardized model parameters (coefficients).

Usage

```
standardize_parameters(
  model,
  method = "refit",
  ci = 0.95,
  robust = FALSE,
  two_sd = FALSE,
  include_response = TRUE,
  verbose = TRUE,
  ...
)

standardize_posteriors(
  model,
  method = "refit",
  robust = FALSE,
  two_sd = FALSE,
  include_response = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

model	A statistical model.
method	The method used for standardizing the parameters. Can be "refit" (default), "posthoc", "smart", "basic", "pseudo" or "sdy". See Details'.
ci	Confidence Interval (CI) level

robust — Logical, if TRUE, centering is done by subtracting the median from the variables and dividing it by the median absolute deviation (MAD). If FALSE, variables are standardized by subtracting the mean and dividing it by the standard deviation (SD).

two_sd — If TRUE, the variables are scaled by two times the deviation (SD or MAD depending on robust). This method can be useful to obtain model coefficients of continuous parameters comparable to coefficients related to binary predictors, when applied to **the predictors** (not the outcome) (Gelman, 2008).

include_response — If TRUE (default), the response value will also be standardized. If FALSE, only the predictors will be standardized. For GLMs the response value will never be standardized (see *Generalized Linear Models* section).

verbose — Toggle warnings and messages on or off.

... — For standardize_parameters(), arguments passed to model_parameters(), such as:

- ci_method, centrality for Mixed models and Bayesian models...
- exponentiate, ...
- etc.

Details

Standardization Methods:

- **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). This method is particularly recommended for complex models that include interactions or transformations (e.g., polynomial or spline terms). The robust (default to FALSE) argument enables a robust standardization of data, i.e., based on the median and MAD instead of the mean and SD. **See standardize() for more details.**
  - **Note** that standardize_parameters(method = "refit") may not return the same results as fitting a model on data that has been standardized with standardize(); standardize_parameters() used the data used by the model fitting function, which might not be same data if there are missing values. see the remove_na argument in standardize().
- **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) of the outcome (which becomes their expression 'unit'). Then, the coefficients related to numeric variables are additionally multiplied by the standard deviation (or MAD if robust) of the related terms, so that they correspond to changes of 1 SD of the predictor (e.g., "A change in 1 SD of x is related to a change of 0.24 of the SD of y). This does not apply to binary variables or factors, so the coefficients are still related to changes in levels. This method is not accurate and tend to give aberrant results when interactions are specified.
- **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()`.

- **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 if `robust`) 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.
- **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 `performance::check_heterogeneity_bias`). Predictors are standardized based on their SD at level of prediction (see also `datawizard::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.
- **sdy** (*for logistic regression models only*): This y-standardization is useful when comparing coefficients of logistic regression models across models for the same sample. Unobserved heterogeneity varies across models with different independent variables, and thus, odds ratios from the same predictor of different models cannot be compared directly. The y-standardization makes coefficients "comparable across models by dividing them with the estimated standard deviation of the latent variable for each model" (Mood 2010). Thus, whenever one has multiple logistic regression models that are fit to the same data and share certain predictors (e.g. nested models), it can be useful to use this standardization approach to make log-odds or odds ratios comparable.

Transformed Variables:

When the model's formula contains transformations (e.g. `y ~ exp(X)`) `method = "refit"` will give different results compared to `method = "basic"` (`"posthoc"` and `"smart"` do not support such transformations): While `"refit"` standardizes the data *prior* to the transformation (e.g. equivalent to `exp(scale(X))`), the `"basic"` method standardizes the transformed data (e.g. equivalent to `scale(exp(X))`).

See the *Transformed Variables* section in `standardize.default()` for more details on how different transformations are dealt with when `method = "refit"`.

Confidence Intervals:

The returned confidence intervals are re-scaled versions of the unstandardized confidence intervals, and not "true" confidence intervals of the standardized coefficients (cf. Jones & Waller, 2015).

Generalized Linear Models:

Standardization for generalized linear models (GLM, GLMM, etc) is done only with respect to the predictors (while the outcome remains as-is, unstandardized) - maintaining the interpretability of the coefficients (e.g., in a binomial model: the exponent of the standardized parameter is the OR of a change of 1 SD in the predictor, etc.)

Dealing with Factors:

`standardize(model)` or `standardize_parameters(model, method = "refit")` do *not* standardize categorical predictors (i.e. factors) / their dummy-variables, which may be a different behaviour compared to other R packages (such as **lm.beta**) or other software packages (like SPSS). To mimic such behaviours, either use `standardize_parameters(model, method = "basic")` to obtain post-hoc standardized parameters, or standardize the data with `datawizard::standardize(data, force = TRUE)` *before* fitting the model.

Value

A data frame with the standardized parameters (`Std_*`, depending on the model type) and their CIs (`CI_low` and `CI_high`). Where applicable, standard errors (SEs) are returned as an attribute (`attr(x, "standard_error")`).

References

- Hoffman, L. (2015). Longitudinal analysis: Modeling within-person fluctuation and change. Routledge.
- Jones, J. A., & Waller, N. G. (2015). The normal-theory and asymptotic distribution-free (ADF) covariance matrix of standardized regression coefficients: theoretical extensions and finite sample behavior. Psychometrika, 80(2), 365-378.
- Neter, J., Wasserman, W., & Kutner, M. H. (1989). Applied linear regression models.
- Gelman, A. (2008). Scaling regression inputs by dividing by two standard deviations. Statistics in medicine, 27(15), 2865-2873.
- Mood C. Logistic Regression: Why We Cannot Do What We Think We Can Do, and What We Can Do About It. European Sociological Review (2010) 26:67–82.

See Also

See also package vignette.

Other standardize: `standardize_info()`

Examples

```
model <- lm(len ~ supp * dose, data = ToothGrowth)
standardize_parameters(model, method = "refit")

standardize_parameters(model, method = "posthoc")
standardize_parameters(model, method = "smart")
standardize_parameters(model, method = "basic")

# Robust and 2 SD
standardize_parameters(model, robust = TRUE)
standardize_parameters(model, two_sd = TRUE)

model <- glm(am ~ cyl * mpg, data = mtcars, family = "binomial")
standardize_parameters(model, method = "refit")
standardize_parameters(model, method = "posthoc")
standardize_parameters(model, method = "basic", exponentiate = TRUE)
```

```
m <- lme4::lmer(mpg ~ cyl + am + vs + (1 | cyl), mtcars)
standardize_parameters(m, method = "pseudo", ci_method = "satterthwaite")
```

```
model <- rstanarm::stan_glm(rating ~ critical + privileges, data = attitude, refresh = 0)
standardize_posteriors(model, method = "refit", verbose = FALSE)
standardize_posteriors(model, method = "posthoc", verbose = FALSE)
standardize_posteriors(model, method = "smart", verbose = FALSE)
head(standardize_posteriors(model, method = "basic", verbose = FALSE))
```

`standard_error` *Standard Errors*

Description

`standard_error()` attempts to return standard errors of model parameters.

Usage

```
standard_error(model, ...)

## Default S3 method:
standard_error(
  model,
  component = "all",
  vcov = NULL,
  vcov_args = NULL,
  verbose = TRUE,
  ...
)

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

## S3 method for class 'glmmTMB'
standard_error(
  model,
  effects = "fixed",
  component = "all",
  verbose = TRUE,
  ...
```

```
)

## S3 method for class 'merMod'
standard_error(
  model,
  effects = "fixed",
  method = NULL,
  vcov = NULL,
  vcov_args = NULL,
  ...
)
```

Arguments

model	A model.
...	Arguments passed to or from other methods.
component	Model component for which standard errors should be shown. See the documentation for your object's class in model_parameters() or p_value() for further details.
vcov	Variance-covariance matrix used to compute uncertainty estimates (e.g., for robust standard errors). This argument accepts a covariance matrix, a function which returns a covariance matrix, or a string which identifies the function to be used to compute the covariance matrix. • A covariance matrix • A function which returns a covariance matrix (e.g., stats::vcov()) • A string which indicates the kind of uncertainty estimates to return. – Heteroskedasticity-consistent: "vcovHC", "HC", "HC0", "HC1", "HC2", "HC3", "HC4", "HC4m", "HC5". See ?sandwich::vcovHC. – Cluster-robust: "vcovCR", "CR0", "CR1", "CR1p", "CR1S", "CR2", "CR3". See ?clubSandwich::vcovCR. – Bootstrap: "vcovBS", "xy", "residual", "wild", "mammen", "webb". See ?sandwich::vcovBS. – Other sandwich package functions: "vcovHAC", "vcovPC", "vcovCL", "vcovPL".
vcov_args	List of arguments to be passed to the function identified by the vcov argument. This function is typically supplied by the **sandwich** or **clubSandwich** packages. Please refer to their documentation (e.g., ?sandwich::vcovHAC) to see the list of available arguments.
verbose	Toggle warnings and messages.
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 ("fixed"), random effects ("random"), or both ("all") 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.

`method` Method for computing degrees of freedom for confidence intervals (CI) and the related p-values. Allowed are following options (which vary depending on the model class): `"residual"`, `"normal"`, `"likelihood"`, `"satterthwaite"`, `"kenward"`, `"wald"`, `"profile"`, `"boot"`, `"uniroot"`, `"ml1"`, `"betwithin"`, `"hdi"`, `"quantile"`, `"ci"`, `"eti"`, `"si"`, `"bci"`, or `"bcai"`. See section *Confidence intervals and approximation of degrees of freedom* in `model_parameters()` for further details.

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)

if (require("sandwich") && require("clubSandwich")) {
  standard_error(model, vcov = "HC3")

  standard_error(model,
    vcov = "vcovCL",
    vcov_args = list(cluster = iris$Species)
  )
}
```

Index