Package 'sjstats'

November 14, 2019

Type Package

Encoding UTF-8

Title Collection of Convenient Functions for Common Statistical Computations

Version 0.17.7

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

Description Collection of convenient functions for common statistical computations, which are not directly provided by R's base or stats packages. This package aims at providing, first, shortcuts for statistical measures, which otherwise could only be calculated with additional effort (like Cramer's V, Phi, or effect size statistics like Eta or Omega squared), or for which currently no functions available. Second, another focus lies on weighted variants of common statistical measures and tests like weighted standard error, mean, t-test, correlation, and more.

License GPL-3

Depends R (>= 3.2), utils

Imports bayestestR (>= 0.4.0), broom, dplyr (>= 0.8.1), emmeans, insight (>= 0.6.0), lme4, magrittr, MASS, modelr, parameters (>= 0.2.0), performance (>= 0.4.0), purrr, rlang, sjlabelled (>= 1.1.1), sjmisc (>= 2.8.2), stats, tidyr

Suggests brms, car, coin, ggplot2, graphics, httr, knitr, mediation, nlme, pbkrtest (>= 0.4-7), pscl, pwr, sandwich, sjPlot, survey, rstan, rstanarm, VGAM, Zelig, testthat

URL https://github.com/strengejacke/sjstats, https://strengejacke.github.io/sjstats

BugReports https://github.com/strengejacke/sjstats/issues

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Daniel Lüdecke [aut, cre] (<https://orcid.org/0000-0002-8895-3206>)

Repository CRAN

Date/Publication 2019-11-14 15:00:02 UTC

R topics documented:

`sjstats-package` *Collection of Convenient Functions for Common Statistical Computations*

Description

Collection of convenient functions for common statistical computations, which are not directly provided by R's base or stats packages.

This package aims at providing, first, shortcuts for statistical measures, which otherwise could only be calculated with additional effort (like standard errors or root mean squared errors).

Second, these shortcut functions are generic (if appropriate), and can be applied not only to vectors, but also to other objects as well (e.g., the Coefficient of Variation can be computed for vectors, linear models, or linear mixed models; the `r2()`-function returns the r-squared value for `lm`, `glm`, `merMod`, `glmmTMB`, or `lme` and other objects).

Most functions of this package are designed as *summary functions*, i.e. they do not transform the input vector; rather, they return a summary, which is sometimes a vector and sometimes a tidy data frame. he focus of most functions lies on summary statistics or fit measures for regression models, including generalized linear models, mixed effects models or Bayesian models. However, some of the functions deal with other statistical measures, like Cronbach's Alpha, Cramer's V, Phi etc.

The comprised tools include:

- For regression and mixed models: Coefficient of Variation, Root Mean Squared Error, Residual Standard Error, Coefficient of Discrimination, R-squared and pseudo-R-squared values, standardized beta values
- Especially for mixed models: Design effect, ICC, sample size calculation and convergence tests
- Especially for Bayesian models: Highest Density Interval, region of practical equivalence (rope), Monte Carlo Standard Errors, ratio of number of effective samples, mediation analysis, Test for Practical Equivalence
- Fit and accuracy measures for regression models: Overdispersion tests, accuracy of predictions, test/training-error comparisons, error rate and binned residual plots for logistic regression models
- For anova-tables: Eta-squared, Partial Eta-squared, Omega-squared and Partial Omega-squared statistics

Furthermore, **sjstats** has functions to access information from model objects, which either support more model objects than their **stats** counterparts, or provide easy access to model attributes, like:

- `model_frame()` to get the model frame
- `model_family()` to get information about the model family, link functions etc.
- `link_inverse()` to get the link-inverse function
- `pred_vars()` and `resp_var()` to get the names of either the dependent or independent variables, or

- `var_names()` to get the "cleaned" variables names from a model object (cleaned means, things like `s()` or `log()` are removed from the returned character vector with variable names.)

Other statistics:

- Cramer's V, Cronbach's Alpha, Mean Inter-Item-Correlation, Mann-Whitney-U-Test, Item-scale reliability tests

---

`anova_stats` *Effect size statistics for anova*

Description

Returns the (partial) eta-squared, (partial) omega-squared, epsilon-squared statistic or Cohen's F for all terms in an anovas. `anova_stats()` returns a tidy summary, including all these statistics and power for each term.

Usage

```
anova_stats(model, digits = 3)

cohens_f(model)

epsilon_sq(model, ci.lvl = NULL, n = 1000, method = c("dist",
  "quantile"))

eta_sq(model, partial = FALSE, ci.lvl = NULL, n = 1000,
  method = c("dist", "quantile"))

omega_sq(model, partial = FALSE, ci.lvl = NULL, n = 1000,
  method = c("dist", "quantile"))
```

Arguments

`model`	A fitted anova-model of class `aov` or `anova`. Other models are coerced to `anova`.
`digits`	Number of decimal points in the returned data frame.
`ci.lvl`	Scalar between 0 and 1. If not `NULL`, returns a data frame with effect sizes including lower and upper confidence intervals.
`n`	Number of bootstraps to be generated.
`method`	Character vector, indicating if confidence intervals should be based on bootstrap standard error, multiplied by the value of the quantile function of the t-distribution (default), or on sample quantiles of the bootstrapped values. See 'Details' in `boot_ci()`. May be abbreviated.
`partial`	Logical, if `TRUE`, the partial eta-squared is returned.

Details

For `eta_sq()` (with `partial = FALSE`), due to non-symmetry, confidence intervals are based on bootstrap-methods. In this case, `n` indicates the number of bootstrap samples to be drawn to compute the confidence intervals. Confidence intervals for partial omega-squared and epsilon-squared is also based on bootstrapping.

Since bootstrapped confidence intervals are based on the bootstrap standard error (i.e. `mean(x) +/-qt(.975,df = length(x) -1) * sd(x)`), bounds of the confidence interval may be negative. Use `method = "quantile"` to make sure that the confidence intervals are always positive.

Value

A data frame with the term name(s) and effect size statistics; if `ci.lvl` is not `NULL`, a data frame including lower and upper confidence intervals is returned. For `anova_stats()`, a tidy data frame with all statistics is returned (excluding confidence intervals).

References

Levine TR, Hullett CR (2002): Eta Squared, Partial Eta Squared, and Misreporting of Effect Size in Communication Research (pdf)

Tippey K, Longnecker MT (2016): An Ad Hoc Method for Computing Pseudo-Effect Size for Mixed Model. (pdf)

Examples

```
# load sample data
data(efc)

# fit linear model
fit <- aov(
  c12hour ~ as.factor(e42dep) + as.factor(c172code) + c160age,
  data = efc
)

eta_sq(fit)
omega_sq(fit)
eta_sq(fit, partial = TRUE)
eta_sq(fit, partial = TRUE, ci.lvl = .8)

anova_stats(car::Anova(fit, type = 2))
```

`auto_prior` *Create default priors for brms-models*

Description

This function creates default priors for brms-regression models, based on the same automatic prior-scale adjustment as in **rstanarm**.

Usage

```
auto_prior(formula, data, gaussian, locations = NULL)
```

Arguments

formula	A formula describing the model, which just needs to contain the model terms, but no notation of interaction, splines etc. Usually, you want only those predictors in the formula, for which automatic priors should be generated. Add informative priors afterwards to the returned brmsprior-object.
data	The data that will be used to fit the model.
gaussian	Logical, if the outcome is gaussian or not.
locations	A numeric vector with location values for the priors. If locations = NULL, 0 is used as location parameter.

Details

auto_prior() is a small, convenient function to create some default priors for brms-models with automatically adjusted prior scales, in a similar way like **rstanarm** does. The default scale for the intercept is 10, for coefficients 2.5. If the outcome is gaussian, both scales are multiplied with sd(y). Then, for categorical variables, nothing more is changed. For numeric variables, the scales are divided by the standard deviation of the related variable.

All prior distributions are *normal* distributions. auto_prior() is intended to quickly create default priors with feasible scales. If more precise definitions of priors is necessary, this needs to be done directly with brms-functions like set_prior().

Value

A brmsprior-object.

Note

As auto_prior() also sets priors on the intercept, the model formula used in brms::brm() must be rewritten to something like y ~ 0 + intercept ..., see set_prior.

Examples

```
library(sjmisc)
data(efc)
efc$c172code <- as.factor(efc$c172code)
efc$c161sex <- to_label(efc$c161sex)

mf <- formula(neg_c_7 ~ c161sex + c160age + c172code)
```

```
if (requireNamespace("brms", quietly = TRUE))
  auto_prior(mf, efc, TRUE)

## compare to
# library(rstanarm)
# m <- stan_glm(mf, data = efc, chains = 2, iter = 200)
# ps <- prior_summary(m)
# ps$prior_intercept$adjusted_scale
# ps$prior$adjusted_scale

## usage
# ap <- auto_prior(mf, efc, TRUE)
# brm(mf, data = efc, priors = ap)

# add informative priors
mf <- formula(neg_c_7 ~ c161sex + c172code)

if (requireNamespace("brms", quietly = TRUE)) {
  auto_prior(mf, efc, TRUE) +
    brms::prior(normal(.1554, 40), class = "b", coef = "c160age")
}

# example with binary response
efc$neg_c_7d <- ifelse(efc$neg_c_7 < median(efc$neg_c_7, na.rm = TRUE), 0, 1)
mf <- formula(neg_c_7d ~ c161sex + c160age + c172code + e17age)

if (requireNamespace("brms", quietly = TRUE))
  auto_prior(mf, efc, FALSE)
```

`bootstrap` *Generate nonparametric bootstrap replications*

Description

Generates `n` bootstrap samples of `data` and returns the bootstrapped data frames as list-variable.

Usage

```
bootstrap(data, n, size)
```

Arguments

`data`	A data frame.
`n`	Number of bootstraps to be generated.
`size`	Optional, size of the bootstrap samples. May either be a number between 1 and `nrow(data)` or a value between 0 and 1 to sample a proportion of observations from `data` (see 'Examples').

Details

By default, each bootstrap sample has the same number of observations as `data`. To generate bootstrap samples without resampling same observations (i.e. sampling without replacement), use `size` to get bootstrapped data with a specific number of observations. However, specifying the `size`-argument is much less memory-efficient than the bootstrap with replacement. Hence, it is recommended to ignore the `size`-argument, if it is not really needed.

Value

A data frame with one column: a list-variable `strap`, which contains resample-objects of class `sj_resample`. These resample-objects are lists with three elements:

1. the original data frame, `data`
2. the rownmumbers `id`, i.e. rownumbers of `data`, indicating the resampled rows with replacement
3. the `resample.id`, indicating the index of the resample (i.e. the position of the `sj_resample`-object in the list `strap`)

Note

This function applies nonparametric bootstrapping, i.e. the function draws samples with replacement.

There is an `as.data.frame`- and a `print`-method to get or print the resampled data frames. See 'Examples'. The `as.data.frame`- method automatically applies whenever coercion is done because a data frame is required as input. See 'Examples' in `boot_ci`.

See Also

`boot_ci` to calculate confidence intervals from bootstrap samples.

Examples

```
data(efc)
bs <- bootstrap(efc, 5)

# now run models for each bootstrapped sample
lapply(bs$strap, function(x) lm(neg_c_7 ~ e42dep + c161sex, data = x))

# generate bootstrap samples with 600 observations for each sample
bs <- bootstrap(efc, 5, 600)

# generate bootstrap samples with 70% observations of the original sample size
bs <- bootstrap(efc, 5, .7)

# compute standard error for a simple vector from bootstraps
# use the `as.data.frame()`-method to get the resampled
# data frame
bs <- bootstrap(efc, 100)
bs$c12hour <- unlist(lapply(bs$strap, function(x) {
```

```
  mean(as.data.frame(x)$c12hour, na.rm = TRUE)
}))

# or as tidyverse-approach
library(dplyr)
library(purrr)
bs <- efc %>%
  bootstrap(100) %>%
  mutate(
    c12hour = map_dbl(strap, ~mean(as.data.frame(.x)$c12hour, na.rm = TRUE))
  )

# bootstrapped standard error
boot_se(bs, c12hour)
```

`boot_ci` *Standard error and confidence intervals for bootstrapped estimates*

Description

Compute nonparametric bootstrap estimate, standard error, confidence intervals and p-value for a vector of bootstrap replicate estimates.

Usage

```
boot_ci(data, ..., method = c("dist", "quantile"), ci.lvl = 0.95)

boot_se(data, ...)

boot_p(data, ...)

boot_est(data, ...)
```

Arguments

`data`	A data frame that containts the vector with bootstrapped estimates, or directly the vector (see 'Examples').
`...`	Optional, unquoted names of variables with bootstrapped estimates. Required, if either `data` is a data frame (and no vector), and only selected variables from `data` should be processed. You may also use functions like : or tidyselect's `select_helpers`.
`method`	Character vector, indicating if confidence intervals should be based on bootstrap standard error, multiplied by the value of the quantile function of the t-distribution (default), or on sample quantiles of the bootstrapped values. See 'Details' in `boot_ci()`. May be abbreviated.
`ci.lvl`	Numeric, the level of the confidence intervals.

Details

The methods require one or more vectors of bootstrap replicate estimates as input.

- `boot_est()` returns the bootstrapped estimate, simply by computing the mean value of all bootstrap estimates.
- `boot_se()` computes the nonparametric bootstrap standard error by calculating the standard deviation of the input vector.
- The mean value of the input vector and its standard error is used by `boot_ci()` to calculate the lower and upper confidence interval, assuming a t-distribution of bootstrap estimate replicates (for `method = "dist"`, the default, which is `mean(x) +/-qt(.975,df = length(x) -1) * sd(x)`); for `method = "quantile"`, 95% sample quantiles are used to compute the confidence intervals (`quantile(x,probs = c(.025,.975))`). Use `ci.lvl` to change the level for the confidence interval.
- P-values from `boot_p()` are also based on t-statistics, assuming normal distribution.

Value

A `tibble` with either bootstrap estimate, standard error, the lower and upper confidence intervals or the p-value for all bootstrapped estimates.

References

Carpenter J, Bithell J. Bootstrap confdence intervals: when, which, what? A practical guide for medical statisticians. Statist. Med. 2000; 19:1141-1164

See Also

`bootstrap` to generate nonparametric bootstrap samples.

Examples

```
library(dplyr)
library(purrr)
data(efc)
bs <- bootstrap(efc, 100)

# now run models for each bootstrapped sample
bs$models <- map(bs$strap, ~lm(neg_c_7 ~ e42dep + c161sex, data = .x))

# extract coefficient "dependency" and "gender" from each model
bs$dependency <- map_dbl(bs$models, ~coef(.x)[2])
bs$gender <- map_dbl(bs$models, ~coef(.x)[3])

# get bootstrapped confidence intervals
boot_ci(bs$dependency)

# compare with model fit
fit <- lm(neg_c_7 ~ e42dep + c161sex, data = efc)
confint(fit)[2, ]
```

```
# alternative function calls.
boot_ci(bs$dependency)
boot_ci(bs, dependency)
boot_ci(bs, dependency, gender)
boot_ci(bs, dependency, gender, method = "q")


# compare coefficients
mean(bs$dependency)
boot_est(bs$dependency)
coef(fit)[2]


# bootstrap() and boot_ci() work fine within pipe-chains
efc %>%
  bootstrap(100) %>%
  mutate(
    models = map(strap, ~lm(neg_c_7 ~ e42dep + c161sex, data = .x)),
    dependency = map_dbl(models, ~coef(.x)[2])
  ) %>%
  boot_ci(dependency)

# check p-value
boot_p(bs$gender)
summary(fit)$coefficients[3, ]

## Not run:
# 'spread_coef()' from the 'sjmisc'-package makes it easy to generate
# bootstrapped statistics like confidence intervals or p-values
library(dplyr)
library(sjmisc)
efc %>%
  # generate bootstrap replicates
  bootstrap(100) %>%
  # apply lm to all bootstrapped data sets
  mutate(
    models = map(strap, ~lm(neg_c_7 ~ e42dep + c161sex + c172code, data = .x))
  ) %>%
  # spread model coefficient for all 100 models
  spread_coef(models) %>%
  # compute the CI for all bootstrapped model coefficients
  boot_ci(e42dep, c161sex, c172code)

# or...
efc %>%
  # generate bootstrap replicates
  bootstrap(100) %>%
  # apply lm to all bootstrapped data sets
  mutate(
    models = map(strap, ~lm(neg_c_7 ~ e42dep + c161sex + c172code, data = .x))
  ) %>%
  # spread model coefficient for all 100 models
  spread_coef(models, append = FALSE) %>%
```

```
  # compute the CI for all bootstrapped model coefficients
  boot_ci()
## End(Not run)
```

---

chisq_gof *Compute model quality*

Description

For logistic regression models, performs a Chi-squared goodness-of-fit-test.

Usage

```
chisq_gof(x, prob = NULL, weights = NULL)
```

Arguments

x	A numeric vector or a glm-object.
prob	Vector of probabilities (indicating the population probabilities) of the same length as x's amount of categories / factor levels. Use nrow(table(x)) to determine the amount of necessary values for prob. Only used, when x is a vector, and not a glm-object.
weights	Vector with weights, used to weight x.
...	More fitted model objects, to compute multiple coefficients of variation at once.

Details

For vectors, this function is a convenient function for the chisq.test(), performing goodness-of-fit test. For glm-objects, this function performs a goodness-of-fit test. A well-fitting model shows *no* significant difference between the model and the observed data, i.e. the reported p-values should be greater than 0.05.

Value

For vectors, returns the object of the computed chisq.test. For glm-objects, an object of class chisq_gof with following values: p.value, the p-value for the goodness-of-fit test; z.score, the standardized z-score for the goodness-of-fit test; rss, the residual sums of squares term and chisq, the pearson chi-squared statistic.

References

Hosmer, D. W., & Lemeshow, S. (2000). Applied Logistic Regression. Hoboken, NJ, USA: John Wiley & Sons, Inc. doi: 10.1002/0471722146

Examples

```
data(efc)
efc$neg_c_7d <- ifelse(efc$neg_c_7 < median(efc$neg_c_7, na.rm = TRUE), 0, 1)
m <- glm(
  neg_c_7d ~ c161sex + barthtot + c172code,
  data = efc,
  family = binomial(link = "logit")
)

# goodness-of-fit test for logistic regression
chisq_gof(m)

# goodness-of-fit test for vectors against probabilities
# differing from population
chisq_gof(efc$e42dep, c(0.3,0.2,0.22,0.28))

# equal to population
chisq_gof(efc$e42dep, prop.table(table(efc$e42dep)))
```

---

cramer *Measures of association for contingency tables*

Description

This function calculates various measure of association for contingency tables and returns the statistic and p-value. Supported measures are Cramer's V, Phi, Spearman's rho, Kendall's tau and Pearson's r.

Usage

```
cramer(tab, ...)

## S3 method for class 'formula'
cramer(formula, data, ci.lvl = NULL, n = 1000,
  method = c("dist", "quantile"), ...)

phi(tab, ...)

xtab_statistics(data, x1 = NULL, x2 = NULL, statistics = c("auto",
  "cramer", "phi", "spearman", "kendall", "pearson", "fisher"),
  weights = NULL, ...)
```

Arguments

tab	A table or ftable. Tables of class xtabs and other will be coerced to ftable objects.

...	Other arguments, passed down to the statistic functions chisq.test, fisher.test or cor.test.
formula	A formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor giving the corresponding groups.
data	A data frame or a table object. If a table object, x1 and x2 will be ignored. For Kendall's *tau*, Spearman's *rho* or Pearson's product moment correlation coefficient, data needs to be a data frame. If x1 and x2 are not specified, the first two columns of the data frames are used as variables to compute the crosstab.
ci.lvl	Scalar between 0 and 1. If not NULL, returns a data frame including lower and upper confidence intervals.
n	Number of bootstraps to be generated.
method	Character vector, indicating if confidence intervals should be based on bootstrap standard error, multiplied by the value of the quantile function of the t-distribution (default), or on sample quantiles of the bootstrapped values. See 'Details' in boot_ci(). May be abbreviated.
x1	Name of first variable that should be used to compute the contingency table. If data is a table object, this argument will be irgnored.
x2	Name of second variable that should be used to compute the contingency table. If data is a table object, this argument will be irgnored.
statistics	Name of measure of association that should be computed. May be one of "auto", "cramer", "phi", "spearman", "kendall", "pearson" or "fisher". See 'Details'.
weights	Name of variable in x that indicated the vector of weights that will be applied to weight all observations. Default is NULL, so no weights are used.

Details

The p-value for Cramer's V and the Phi coefficient are based on chisq.test(). If any expected value of a table cell is smaller than 5, or smaller than 10 and the df is 1, then fisher.test() is used to compute the p-value, unless statistics = "fisher"; in this case, the use of fisher.test() is forced to compute the p-value. The test statistic is calculated with cramer() resp. phi().

Both test statistic and p-value for Spearman's rho, Kendall's tau and Pearson's r are calculated with cor.test().

When statistics = "auto", only Cramer's V or Phi are calculated, based on the dimension of the table (i.e. if the table has more than two rows or columns, Cramer's V is calculated, else Phi).

Value

For phi(), the table's Phi value. For cramer(), the table's Cramer's V.

For xtab_statistics(), a list with following components:

estimate the value of the estimated measure of association.

p.value the p-value for the test.

`statistic` the value of the test statistic.

`stat.name` the name of the test statistic.

`stat.html` if applicable, the name of the test statistic, in HTML-format.

`df` the degrees of freedom for the contingency table.

`method` character string indicating the name of the measure of association.

`method.html` if applicable, the name of the measure of association, in HTML-format.

`method.short` the short form of association measure, equals the `statistics`-argument.

`fisher` logical, if Fisher's exact test was used to calculate the p-value.

Examples

```
# Phi coefficient for 2x2 tables
tab <- table(sample(1:2, 30, TRUE), sample(1:2, 30, TRUE))
phi(tab)

# Cramer's V for nominal variables with more than 2 categories
tab <- table(sample(1:2, 30, TRUE), sample(1:3, 30, TRUE))
cramer(tab)

# formula notation
data(efc)
cramer(e16sex ~ c161sex, data = efc)

# bootstrapped confidence intervals
cramer(e16sex ~ c161sex, data = efc, ci.lvl = .95, n = 100)

# 2x2 table, compute Phi automatically
xtab_statistics(efc, e16sex, c161sex)

# more dimensions than 2x2, compute Cramer's V automatically
xtab_statistics(efc, c172code, c161sex)

# ordinal data, use Kendall's tau
xtab_statistics(efc, e42dep, quol_5, statistics = "kendall")

# calcilate Spearman's rho, with continuity correction
xtab_statistics(efc,
  e42dep,
  quol_5,
  statistics = "spearman",
  exact = FALSE,
  continuity = TRUE
)
```

cv *Compute model quality*

Description

Compute the coefficient of variation.

Usage

```
cv(x, ...)
```

Arguments

x	Fitted linear model of class lm, merMod (**lme4**) or lme (**nlme**).
...	More fitted model objects, to compute multiple coefficients of variation at once.

Details

The advantage of the cv is that it is unitless. This allows coefficient of variation to be compared to each other in ways that other measures, like standard deviations or root mean squared residuals, cannot be.

"It is interesting to note the differences between a model's CV and R-squared values. Both are unitless measures that are indicative of model fit, but they define model fit in two different ways: CV evaluates the relative closeness of the predictions to the actual values while R-squared evaluates how much of the variability in the actual values is explained by the model." *(source: UCLA-FAQ)*

Value

Numeric, the coefficient of variation.

Examples

```
data(efc)
fit <- lm(barthtot ~ c160age + c12hour, data = efc)
cv(fit)
```

cv_error *Test and training error from model cross-validation*

Description

`cv_error()` computes the root mean squared error from a model fitted to kfold cross-validated test-training-data. `cv_compare()` does the same, for multiple formulas at once (by calling `cv_error()` for each formula).

Usage

```
cv_error(data, formula, k = 5)

cv_compare(data, formulas, k = 5)
```

Arguments

`data`	A data frame.
`formula`	The formula to fit the linear model for the test and training data.
`k`	The number of folds for the kfold-crossvalidation.
`formulas`	A list of formulas, to fit linear models for the test and training data.

Details

`cv_error()` first generates cross-validated test-training pairs, using `crossv_kfold` and then fits a linear model, which is described in `formula`, to the training data. Then, predictions for the test data are computed, based on the trained models. The *training error* is the mean value of the `rmse` for all *trained* models; the *test error* is the rmse based on all residuals from the test data.

Value

A data frame with the root mean squared errors for the training and test data.

Examples

```
data(efc)
cv_error(efc, neg_c_7 ~ barthtot + c161sex)

cv_compare(efc, formulas = list(
  neg_c_7 ~ barthtot + c161sex,
  neg_c_7 ~ barthtot + c161sex + e42dep,
  neg_c_7 ~ barthtot + c12hour
))
```

`design_effect` *Design effects for two-level mixed models*

Description

Compute the design effect (also called *Variance Inflation Factor*) for mixed models with two-level design.

Usage

```
design_effect(n, icc = 0.05)
```

Arguments

`n`	Average number of observations per grouping cluster (i.e. level-2 unit).
`icc`	Assumed intraclass correlation coefficient for multilevel-model.

Details

The formula for the design effect is simply `(1 + (n -1) * icc)`.

Value

The design effect (Variance Inflation Factor) for the two-level model.

References

Bland JM. 2000. Sample size in guidelines trials. Fam Pract. (17), 17-20.

Hsieh FY, Lavori PW, Cohen HJ, Feussner JR. 2003. An Overview of Variance Inflation Factors for Sample-Size Calculation. Evaluation and the Health Professions 26: 239-257. doi: 10.1177/0163278703255230

Snijders TAB. 2005. Power and Sample Size in Multilevel Linear Models. In: Everitt BS, Howell DC (Hrsg.). Encyclopedia of Statistics in Behavioral Science. Chichester, UK: John Wiley and Sons, Ltd. doi: 10.1002/0470013192.bsa492

Thompson DM, Fernald DH, Mold JW. 2012. Intraclass Correlation Coefficients Typical of Cluster-Randomized Studies: Estimates From the Robert Wood Johnson Prescription for Health Projects. The Annals of Family Medicine;10(3):235-40. doi: 10.1370/afm.1347

Examples

```
# Design effect for two-level model with 30 observations per
# cluster group (level-2 unit) and an assumed intraclass
# correlation coefficient of 0.05.
design_effect(n = 30)
```

```
# Design effect for two-level model with 24 observation per cluster
# group and an assumed intraclass correlation coefficient of 0.2.
design_effect(n = 24, icc = 0.2)
```

---

efc *Sample dataset from the EUROFAMCARE project*

Description

German data set from the European study on family care of older people.

References

Lamura G, Döhner H, Kofahl C, editors. Family carers of older people in Europe: a six-country comparative study. Münster: LIT, 2008.

---

find_beta *Determining distribution parameters*

Description

`find_beta()`, `find_normal()` and `find_cauchy()` find the shape, mean and standard deviation resp. the location and scale parameters to describe the beta, normal or cauchy distribution, based on two percentiles. `find_beta2()` finds the shape parameters for a Beta distribution, based on a probability value and its standard error or confidence intervals.

Usage

```
find_beta(x1, p1, x2, p2)

find_beta2(x, se, ci, n)

find_cauchy(x1, p1, x2, p2)

find_normal(x1, p1, x2, p2)
```

Arguments

x1	Value for the first percentile.
p1	Probability of the first percentile.
x2	Value for the second percentile.
p2	Probability of the second percentile.

x	Numeric, a probability value between 0 and 1. Typically indicates a prevalence rate of an outcome of interest; Or an integer value with the number of observed events. In this case, specify n to indicate the toral number of observations.
se	The standard error of x. Either se or ci must be specified.
ci	The upper limit of the confidence interval of x. Either se or ci must be specified.
n	Numeric, number of total observations. Needs to be specified, if x is an integer (number of observed events), and no probability. See 'Examples'.

Details

These functions can be used to find parameter for various distributions, to define prior probabilities for Bayesian analyses. x1, p1, x2 and p2 are parameters that describe two quantiles. Given this knowledge, the distribution parameters are returned.

Use find_beta2(), if the known parameters are, e.g. a prevalence rate or similar probability, and its standard deviation or confidence interval. In this case. x should be a probability, for example a prevalence rate of a certain event. se then needs to be the standard error for this probability. Alternatively, ci can be specified, which should indicate the upper limit of the confidence interval od the probability (prevalence rate) x. If the number of events out of a total number of trials is known (e.g. 12 heads out of 30 coin tosses), x can also be the number of observed events, while n indicates the total amount of trials (in the above example, the function call would be: find_beta2(x = 12,n = 30)).

Value

A list of length two, with the two distribution parameters than can be used to define the distribution, which (best) describes the shape for the given input parameters.

References

Cook JD. Determining distribution parameters from quantiles. 2010: Department of Biostatistics, Texas (PDF)

Examples

```
# example from blogpost:
# https://www.johndcook.com/blog/2010/01/31/parameters-from-percentiles/
# 10% of patients respond within 30 days of treatment
# and 80% respond within 90 days of treatment
find_normal(x1 = 30, p1 = .1, x2 = 90, p2 = .8)
find_cauchy(x1 = 30, p1 = .1, x2 = 90, p2 = .8)

parms <- find_normal(x1 = 30, p1 = .1, x2 = 90, p2 = .8)
curve(
  dnorm(x, mean = parms$mean, sd = parms$sd),
  from = 0, to = 200
)

parms <- find_cauchy(x1 = 30, p1 = .1, x2 = 90, p2 = .8)
curve(
```

```
  dcauchy(x, location = parms$location, scale = parms$scale),
  from = 0, to = 200
)


find_beta2(x = .25, ci = .5)

shapes <- find_beta2(x = .25, ci = .5)
curve(dbeta(x, shapes[[1]], shapes[[2]]))

# find Beta distribution for 3 events out of 20 observations
find_beta2(x = 3, n = 20)

shapes <- find_beta2(x = 3, n = 20)
curve(dbeta(x, shapes[[1]], shapes[[2]]))
```

`fish` *Sample dataset*

Description

Sample data from the UCLA idre website.

References

https://stats.idre.ucla.edu/r/dae/zip/

`gmd` *Gini's Mean Difference*

Description

`gmd()` computes Gini's mean difference for a numeric vector or for all numeric vectors in a data frame.

Usage

```
gmd(x, ...)
```

Arguments

`x`	A vector or data frame.
`...`	Optional, unquoted names of variables that should be selected for further processing. Required, if `x` is a data frame (and no vector) and only selected variables from `x` should be processed. You may also use functions like `:` or tidyselect's `select_helpers`.

Value

For numeric vectors, Gini's mean difference. For non-numeric vectors or vectors of length < 2, returns NA.

Note

Gini's mean difference is defined as the mean absolute difference between any two distinct elements of a vector. Missing values from x are silently removed.

References

David HA. Gini's mean difference rediscovered. Biometrika 1968(55): 573-575

Examples

```
data(efc)
gmd(efc$e17age)
gmd(efc, e17age, c160age, c12hour)
```

---

grpmean *Summary of mean values by group*

Description

Computes mean, sd and se for each sub-group (indicated by grp) of dv.

Usage

```
grpmean(x, dv, grp, weights = NULL, digits = 2, out = c("txt",
  "viewer", "browser"), encoding = "UTF-8", file = NULL)
```

Arguments

x	A (grouped) data frame.
dv	Name of the dependent variable, for which the mean value, grouped by grp, is computed.
grp	Factor with the cross-classifying variable, where dv is grouped into the categories represented by grp. Numeric vectors are coerced to factors.
weights	Name of variable in x that indicated the vector of weights that will be applied to weight all observations. Default is NULL, so no weights are used.
digits	Numeric, amount of digits after decimal point when rounding estimates and values.
out	Character vector, indicating whether the results should be printed to console (out = "txt") or as HTML-table in the viewer-pane (out = "viewer") or browser (out = "browser"), of if the results should be plotted (out = "plot", only applies to certain functions). May be abbreviated.

encoding	Character vector, indicating the charset encoding used for variable and value labels. Default is "UTF-8". Only used when out is not "txt".
file	Destination file, if the output should be saved as file. Only used when out is not "txt".

Details

This function performs a One-Way-Anova with dv as dependent and grp as independent variable, by calling lm(count ~ as.factor(grp)). Then contrast is called to get p-values for each subgroup. P-values indicate whether each group-mean is significantly different from the total mean.

Value

For non-grouped data frames, grpmean() returns a data frame with following columns: term, mean, N, std.dev, std.error and p.value. For grouped data frames, returns a list of such data frames.

Examples

```
data(efc)
grpmean(efc, c12hour, e42dep)

data(iris)
grpmean(iris, Sepal.Width, Species)

# also works for grouped data frames
library(dplyr)
efc %>%
  group_by(c172code) %>%
  grpmean(c12hour, e42dep)

# weighting
efc$weight <- abs(rnorm(n = nrow(efc), mean = 1, sd = .5))
grpmean(efc, c12hour, e42dep, weights = weight)
```

---

inequ_trend *Compute trends in status inequalities*

Description

This method computes the proportional change of absolute (rate differences) and relative (rate ratios) inequalities of prevalence rates for two different status groups, as proposed by Mackenbach et al. (2015).

Usage

```
inequ_trend(data, prev.low, prev.hi)
```

Arguments

data	A data frame that contains the variables with prevalence rates for both low and high status groups (see 'Examples').
prev.low	The name of the variable with the prevalence rates for the low status groups.
prev.hi	The name of the variable with the prevalence rates for the hi status groups.

Details

Given the time trend of prevalence rates of an outcome for two status groups (e.g. the mortality rates for people with lower and higher socioeconomic status over 40 years), this function computes the proportional change of absolute and relative inequalities, expressed in changes in rate differences and rate ratios. The function implements the algorithm proposed by *Mackenbach et al. 2015*.

Value

A data frame with the prevalence rates as well as the values for the proportional change in absolute (rd) and relative (rr) ineqqualities.

References

Mackenbach JP, Martikainen P, Menvielle G, de Gelder R. 2015. The Arithmetic of Reducing Relative and Absolute Inequalities in Health: A Theoretical Analysis Illustrated with European Mortality Data. Journal of Epidemiology and Community Health 70(7): 730-36. doi: 10.1136/jech2015207018

Examples

```
# This example reproduces Fig. 1 of Mackenbach et al. 2015, p.5

# 40 simulated time points, with an initial rate ratio of 2 and
# a rate difference of 100 (i.e. low status group starts with a
# prevalence rate of 200, the high status group with 100)

# annual decline of prevalence is 1% for the low, and 3% for the
# high status group

n <- 40
time <- seq(1, n, by = 1)
lo <- rep(200, times = n)
for (i in 2:n) lo[i] <- lo[i - 1] * .99

hi <- rep(100, times = n)
for (i in 2:n) hi[i] <- hi[i - 1] * .97

prev.data <- data.frame(lo, hi)

# print values
inequ_trend(prev.data, lo, hi)

# plot trends - here we see that the relative inequalities
```

```
# are increasing over time, while the absolute inequalities
# are first increasing as well, but later are decreasing
# (while rel. inequ. are still increasing)
plot(inequ_trend(prev.data, lo, hi))
```

`is_prime` *Find prime numbers*

Description

This functions checks whether a number is, or numbers in a vector are prime numbers.

Usage

```
is_prime(x)
```

Arguments

`x`	An integer, or a vector of integers.

Value

`TRUE` for each prime number in `x`, `FALSE` otherwise.

Examples

```
is_prime(89)
is_prime(15)
is_prime(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
```

`mean_n` *Row means with min amount of valid values*

Description

This function is similar to the SPSS `MEAN.n` function and computes row means from a `data.frame` or `matrix` if at least `n` values of a row are valid (and not `NA`).

Usage

```
mean_n(dat, n, digits = 2)
```

Arguments

dat	A data frame with at least two columns, where row means are applied.
n	May either be • a numeric value that indicates the amount of valid values per row to calculate the row mean; • or a value between 0 and 1, indicating a proportion of valid values per row to calculate the row mean (see 'Details'). If a row's sum of valid values is less than n, NA will be returned as row mean value.
digits	Numeric value indicating the number of decimal places to be used for rounding mean value. Negative values are allowed (see 'Details').

Details

Rounding to a negative number of digits means rounding to a power of ten, so for example mean_n(df, 3, digits = -2) rounds to the nearest hundred.

For n, must be a numeric value from 0 to ncol(dat). If a *row* in dat has at least n non-missing values, the row mean is returned. If n is a non-integer value from 0 to 1, n is considered to indicate the proportion of necessary non-missing values per row. E.g., if n = .75, a row must have at least ncol(dat) * n non-missing values for the row mean to be calculated. See 'Examples'.

Value

A vector with row mean values of df for those rows with at least n valid values. Else, NA is returned.

References

r4stats.com

Examples

```
dat <- data.frame(c1 = c(1,2,NA,4),
                  c2 = c(NA,2,NA,5),
                  c3 = c(NA,4,NA,NA),
                  c4 = c(2,3,7,8))

# needs at least 4 non-missing values per row
mean_n(dat, 4) # 1 valid return value

# needs at least 3 non-missing values per row
mean_n(dat, 3) # 2 valid return values

# needs at least 2 non-missing values per row
mean_n(dat, 2)

# needs at least 1 non-missing value per row
mean_n(dat, 1) # all means are shown
```

```
# needs at least 50% of non-missing values per row
mean_n(dat, .5) # 3 valid return values

# needs at least 75% of non-missing values per row
mean_n(dat, .75) # 2 valid return values
```

`mediation` *Summary of Bayesian multivariate-response mediation-models*

Description

`mediation()` is a short summary for multivariate-response mediation-models.

Usage

```
mediation(x, ...)

## S3 method for class 'brmsfit'
mediation(x, treatment, mediator, prob = 0.9,
  typical = "median", ...)
```

Arguments

`x`	A `stanreg`, `stanfit`, or `brmsfit` object.
`...`	Not used.
`treatment`	Character, name of the treatment variable (or direct effect) in a (multivariate response) mediator-model. If missing, `mediation()` tries to find the treatment variable automatically, however, this may fail.
`mediator`	Character, name of the mediator variable in a (multivariate response) mediator-model. If missing, `mediation()` tries to find the treatment variable automatically, however, this may fail.
`prob`	Vector of scalars between 0 and 1, indicating the mass within the credible interval that is to be estimated.
`typical`	The typical value that will represent the Bayesian point estimate. By default, the posterior median is returned. See `typical_value` for possible values for this argument.

Details

`mediation()` returns a data frame with information on the *direct effect* (mean value of posterior samples from `treatment` of the outcome model), *mediator effect* (mean value of posterior samples from `mediator` of the outcome model), *indirect effect* (mean value of the multiplication of the posterior samples from `mediator` of the outcome model and the posterior samples from `treatment` of the mediation model) and the total effect (mean value of sums of posterior samples used for the direct and indirect effect). The *proportion mediated* is the indirect effect divided by the total effect.

For all values, the 90% HDIs are calculated by default. Use prob to calculate a different interval.

The arguments treatment and mediator do not necessarily need to be specified. If missing, mediation() tries to find the treatment and mediator variable automatically. If this does not work, specify these variables.

Value

A data frame with direct, indirect, mediator and total effect of a multivariate-response mediation-model, as well as the proportion mediated. The effect sizes are mean values of the posterior samples.

---

mwu *Mann-Whitney-U-Test*

Description

This function performs a Mann-Whitney-U-Test (or Wilcoxon rank sum test, see wilcox.test and wilcox_test) for x, for each group indicated by grp. If grp has more than two categories, a comparison between each combination of two groups is performed.

The function reports U, p and Z-values as well as effect size r and group-rank-means.

Usage

```
mwu(data, x, grp, distribution = "asymptotic", out = c("txt", "viewer",
  "browser"), encoding = "UTF-8", file = NULL)
```

Arguments

data	A data frame.
x	Bare (unquoted) variable name, or a character vector with the variable name.
grp	Bare (unquoted) name of the cross-classifying variable, where x is grouped into the categories represented by grp, or a character vector with the variable name.
distribution	Indicates how the null distribution of the test statistic should be computed. May be one of "exact", "approximate" or "asymptotic" (default). See wilcox_test for details.
out	Character vector, indicating whether the results should be printed to console (out = "txt") or as HTML-table in the viewer-pane (out = "viewer") or browser (out = "browser"), of if the results should be plotted (out = "plot", only applies to certain functions). May be abbreviated.
encoding	Character vector, indicating the charset encoding used for variable and value labels. Default is "UTF-8". Only used when out is not "txt".
file	Destination file, if the output should be saved as file. Only used when out is not "txt".

Value

(Invisibly) returns a data frame with U, p and Z-values for each group-comparison as well as effect-size r; additionally, group-labels and groups' n's are also included.

Note

This function calls the `wilcox_test` with formula. If `grp` has more than two groups, additionally a Kruskal-Wallis-Test (see `kruskal.test`) is performed.

Interpretation of effect sizes, as a rule-of-thumb:

- small effect >= 0.1
- medium effect >= 0.3
- large effect >= 0.5

Examples

```
data(efc)
# Mann-Whitney-U-Tests for elder's age by elder's dependency.
mwu(efc, e17age, e42dep)
```

---

nhanes_sample *Sample dataset from the National Health and Nutrition Examination Survey*

Description

Selected variables from the National Health and Nutrition Examination Survey that are used in the example from Lumley (2010), Appendix E. See `svyglm.nb` for examples.

References

Lumley T (2010). Complex Surveys: a guide to analysis using R. Wiley

odds_to_rr *Get relative risks estimates from logistic regressions or odds ratio values*

Description

`odds_to_rr()` converts odds ratios from a logistic regression model (including mixed models) into relative risks; `or_to_rr()` converts a single odds ratio estimate into a relative risk estimate.

Usage

```
odds_to_rr(fit)

or_to_rr(or, p0)
```

Arguments

`fit`	A fitted binomial generalized linear (mixed) model with logit-link function (logistic (multilevel) regression model).
`or`	Numeric, an odds ratio estimate.
`p0`	Numeric, the risk of having a positive outcome in the control or unexposed group (reference group), i.e. the number of outcome or "successes" in the control divided by the total number of observations in the control group.

Details

This function extracts the odds ratios (exponentiated model coefficients) from logistic regressions (fitted with `glm` or `glmer`) and their related confidence intervals, and transforms these values into relative risks (and their related confidence intervals).

The formula for transformation is based on Zhang and Yu (1998), Wang (2013) and Grant (2014): `RR <-OR / (1 -P0 + (P0 * OR))`, where `OR` is the odds ratio and `P0` indicates the proportion of the incidence in the outcome variable for the control group (reference group).

Value

A data frame with relative risks and lower/upper confidence interval for the relative risks estimates; for `or_to_rr()`, the risk ratio estimate.

References

Grant RL. 2014. Converting an odds ratio to a range of plausible relative risks for better communication of research findings. BMJ 348:f7450. doi: 10.1136/bmj.f7450

Wang Z. 2013. Converting Odds Ratio to Relative Risk in Cohort Studies with Partial Data Information. J Stat Soft 2013;55. doi: 10.18637/jss.v055.i05

Zhang J, Yu KF. 1998. What's the Relative Risk? A Method of Correcting the Odds Ratio in Cohort Studies of Common Outcomes. JAMA; 280(19): 1690-1. doi: 10.1001/jama.280.19.1690

Examples

```
library(sjmisc)
library(lme4)
# create binary response
sleepstudy$Reaction.dicho <- dicho(sleepstudy$Reaction, dich.by = "median")
# fit model
fit <- glmer(Reaction.dicho ~ Days + (Days | Subject),
             data = sleepstudy, family = binomial("logit"))
# convert to relative risks
odds_to_rr(fit)


data(efc)
# create binary response
y <- ifelse(efc$neg_c_7 < median(na.omit(efc$neg_c_7)), 0, 1)
# create data frame for fitted model
mydf <- data.frame(
  y = as.factor(y),
  sex = to_factor(efc$c161sex),
  dep = to_factor(efc$e42dep),
  barthel = efc$barthtot,
  education = to_factor(efc$c172code)
)
# fit model
fit <- glm(y ~., data = mydf, family = binomial(link = "logit"))
# convert to relative risks
odds_to_rr(fit)

# replicate OR/RR for coefficient "sex" from above regression
# p0 ~ .44, or ~ 1.914
prop.table(table(mydf$y, mydf$sex))
or_to_rr(1.914, 0.1055 / (.1324 + .1055))
```

`overdisp` *Deprecated functions*

Description

A list of deprecated functions.

Usage

```
overdisp(x, ...)

zero_count(x, ...)
```

```
pca(x, ...)

pca_rotate(x, ...)

r2(x)

icc(x)

p_value(x, ...)

se(x, ...)
```

Arguments

x	An object.
...	Currently not used.

Value

Nothing.

---

prop *Proportions of values in a vector*

Description

`prop()` calculates the proportion of a value or category in a variable. `props()` does the same, but allows for multiple logical conditions in one statement. It is similar to `mean()` with logical predicates, however, both `prop()` and `props()` work with grouped data frames.

Usage

```
prop(data, ..., weights = NULL, na.rm = TRUE, digits = 4)

props(data, ..., na.rm = TRUE, digits = 4)
```

Arguments

data	A data frame. May also be a grouped data frame (see 'Examples').
...	One or more value pairs of comparisons (logical predicates). Put variable names the left-hand-side and values to match on the right hand side. Expressions may be quoted or unquoted. See 'Examples'.
weights	Vector of weights that will be applied to weight all observations. Must be a vector of same length as the input vector. Default is `NULL`, so no weights are used.

na.rm	Logical, whether to remove NA values from the vector when the proportion is calculated. na.rm = FALSE gives you the raw percentage of a value in a vector, na.rm = TRUE the valid percentage.
digits	Amount of digits for returned values.

Details

prop() only allows one logical statement per comparison, while props() allows multiple logical statements per comparison. However, prop() supports weighting of variables before calculating proportions, and comparisons may also be quoted. Hence, prop() also processes comparisons, which are passed as character vector (see 'Examples').

Value

For one condition, a numeric value with the proportion of the values inside a vector. For more than one condition, a tibble with one column of conditions and one column with proportions. For grouped data frames, returns a tibble with one column per group with grouping categories, followed by one column with proportions per condition.

Examples

```
data(efc)

# proportion of value 1 in e42dep
prop(efc, e42dep == 1)

# expression may also be completely quoted
prop(efc, "e42dep == 1")

# use "props()" for multiple logical statements
props(efc, e17age > 70 & e17age < 80)

# proportion of value 1 in e42dep, and all values greater
# than 2 in e42dep, including missing values. will return a tibble
prop(efc, e42dep == 1, e42dep > 2, na.rm = FALSE)

# for factors or character vectors, use quoted or unquoted values
library(sjmisc)
# convert numeric to factor, using labels as factor levels
efc$e16sex <- to_label(efc$e16sex)
efc$n4pstu <- to_label(efc$n4pstu)

# get proportion of female older persons
prop(efc, e16sex == female)

# get proportion of male older persons
prop(efc, e16sex == "male")

# "props()" needs quotes around non-numeric factor levels
props(efc,
  e17age > 70 & e17age < 80,
```

```
  n4pstu == 'Care Level 1' | n4pstu == 'Care Level 3'
)

# also works with pipe-chains
library(dplyr)
efc %>% prop(e17age > 70)
efc %>% prop(e17age > 70, e16sex == 1)

# and with group_by
efc %>%
  group_by(e16sex) %>%
  prop(e42dep > 2)

efc %>%
  select(e42dep, c161sex, c172code, e16sex) %>%
  group_by(c161sex, c172code) %>%
  prop(e42dep > 2, e16sex == 1)

# same for "props()"
efc %>%
  select(e42dep, c161sex, c172code, c12hour, n4pstu) %>%
  group_by(c161sex, c172code) %>%
  props(
    e42dep > 2,
    c12hour > 20 & c12hour < 40,
    n4pstu == 'Care Level 1' | n4pstu == 'Care Level 3'
  )
```

`robust` *Robust standard errors for regression models*

Description

`robust()` computes robust standard error for regression models. This method calls one of the `vcov*()`-functions from the **sandwich**-package for robust covariance matrix estimators. Results are returned as tidy data frame.

`svy()` is intended to compute standard errors for survey designs (complex samples) fitted with regular `lm` or `glm` functions, as alternative to the **survey**-package. It simulates sampling weights by adjusting the residual degrees of freedom based on the precision weights used to fit `x`, and then calls `robust()` with the adjusted model.

Usage

```
robust(x, vcov.fun = "vcovHC", vcov.type = c("HC3", "const", "HC",
  "HC0", "HC1", "HC2", "HC4", "HC4m", "HC5"), vcov.args = NULL,
  conf.int = FALSE, exponentiate = FALSE)
```

```
svy(x, vcov.fun = "vcovHC", vcov.type = c("HC1", "const", "HC", "HC0",
  "HC3", "HC2", "HC4", "HC4m", "HC5"), vcov.args = NULL,
  conf.int = FALSE, exponentiate = FALSE)
```

Arguments

x	A fitted model of any class that is supported by the vcov*()-functions from the **sandwich** package. For svy(), x must be lm object, fitted with weights.
vcov.fun	String, indicating the name of the vcov*()-function from the **sandwich**-package, e.g. vcov.fun = "vcovCL".
vcov.type	Character vector, specifying the estimation type for the robust covariance matrix estimation (see vcovHC for details).
vcov.args	List of named vectors, used as additional arguments that are passed down to vcov.fun.
conf.int	Logical, TRUE if confidence intervals based on robust standard errors should be included.
exponentiate	Logical, whether to exponentiate the coefficient estimates and confidence intervals (typical for logistic regression).

Value

A summary of the model, including estimates, robust standard error, p-value and - optionally - the confidence intervals.

Note

svy() simply calls robust(), but first adjusts the residual degrees of freedom based on the model weights. Hence, for svy(), x should be fitted with weights. This simulates *sampling weights* like in survey designs, though lm and glm implement *precision weights*. The results from svy() are usually more accurate than simple weighted standard errors for complex samples. However, results from the **survey** package are still more exactly, especially regarding the estimates.

vcov.type for svy() defaults to "HC1", because standard errors with this estimation type come closest to the standard errors from the **survey**-package.

Currently, svy() only works for objects of class lm.

Examples

```
data(efc)
fit <- lm(barthtot ~ c160age + c12hour + c161sex + c172code, data = efc)
summary(fit)
robust(fit)

confint(fit)
robust(fit, conf.int = TRUE)
robust(fit, vcov.type = "HC1", conf.int = TRUE) # "HC1" should be Stata default

library(sjmisc)
```

```
# dichtomozize service usage by "service usage yes/no"
efc$services <- sjmisc::dicho(efc$tot_sc_e, dich.by = 0)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep,
           data = efc, family = binomial(link = "logit"))

robust(fit)
robust(fit, conf.int = TRUE, exponentiate = TRUE)
```

`samplesize_mixed` *Sample size for linear mixed models*

Description

Compute an approximated sample size for linear mixed models (two-level-designs), based on power-calculation for standard design and adjusted for design effect for 2-level-designs.

Usage

```
samplesize_mixed(eff.size, df.n = NULL, power = 0.8,
  sig.level = 0.05, k, n, icc = 0.05)

smpsize_lmm(eff.size, df.n = NULL, power = 0.8, sig.level = 0.05, k,
  n, icc = 0.05)
```

Arguments

`eff.size`	Effect size.
`df.n`	Optional argument for the degrees of freedom for numerator. See 'Details'.
`power`	Power of test (1 minus Type II error probability).
`sig.level`	Significance level (Type I error probability).
`k`	Number of cluster groups (level-2-unit) in multilevel-design.
`n`	Optional, number of observations per cluster groups (level-2-unit) in multilevel-design.
`icc`	Expected intraclass correlation coefficient for multilevel-model.

Details

The sample size calculation is based on a power-calculation for the standard design. If `df.n` is not specified, a power-calculation for an unpaired two-sample t-test will be computed (using `pwr.t.test` of the **pwr**-package). If `df.n` is given, a power-calculation for general linear models will be computed (using `pwr.f2.test` of the **pwr**-package). The sample size of the standard design is then adjusted for the design effect of two-level-designs (see `design_effect`). Thus, the sample size calculation is appropriate in particular for two-level-designs (see *Snijders 2005*). Models that additionally include repeated measures (three-level-designs) may work as well, however, the computed sample size may be less accurate.

Value

A list with two values: The number of subjects per cluster, and the total sample size for the linear mixed model.

References

Cohen J. 1988. Statistical power analysis for the behavioral sciences (2nd ed.). Hillsdale,NJ: Lawrence Erlbaum.

Hsieh FY, Lavori PW, Cohen HJ, Feussner JR. 2003. An Overview of Variance Inflation Factors for Sample-Size Calculation. Evaluation and the Health Professions 26: 239-257. doi: 10.1177/0163278703255230

Snijders TAB. 2005. Power and Sample Size in Multilevel Linear Models. In: Everitt BS, Howell DC (Hrsg.). Encyclopedia of Statistics in Behavioral Science. Chichester, UK: John Wiley and Sons, Ltd. doi: 10.1002/0470013192.bsa492

Examples

```
# Sample size for multilevel model with 30 cluster groups and a small to
# medium effect size (Cohen's d) of 0.3. 27 subjects per cluster and
# hence a total sample size of about 802 observations is needed.
samplesize_mixed(eff.size = .3, k = 30)

# Sample size for multilevel model with 20 cluster groups and a medium
# to large effect size for linear models of 0.2. Five subjects per cluster and
# hence a total sample size of about 107 observations is needed.
samplesize_mixed(eff.size = .2, df.n = 5, k = 20, power = .9)
```

---

scale_weights *Rescale design weights for multilevel analysis*

Description

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

Usage

```
scale_weights(x, cluster.id, pweight)
```

Arguments

x	A data frame.
cluster.id	Variable indicating the grouping structure (strata) of the survey data (level-2-cluster variable).
pweight	Variable indicating the probability (design or sampling) weights of the survey data (level-1-weight).

Details

Rescaling is based on two methods: For svywght_a, the sample weights pweight are adjusted by a factor that represents the proportion of cluster size divided by the sum of sampling weights within each cluster. The adjustment factor for svywght_b is the sum of sample weights within each cluster devided by the sum of squared sample weights within each cluster (see Carle (2009), Appendix B).

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

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

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

Value

x, with two new variables: svywght_a and svywght_b, which represent the rescaled design weights to use in multilevel models (use these variables for the weights argument).

References

Carle AC. *Fitting multilevel models in complex survey data with design weights: Recommendations* BMC Medical Research Methodology 2009, 9(49): 1-13

Asparouhov T. *General Multi-Level Modeling with Sampling Weights* Communications in Statistics - Theory and Methods 2006, 35: 439-460

Examples

```
data(nhanes_sample)
scale_weights(nhanes_sample, SDMVSTRA, WTINT2YR)

library(lme4)
nhanes_sample <- scale_weights(nhanes_sample, SDMVSTRA, WTINT2YR)
glmer(
  total ~ factor(RIAGENDR) * (log(age) + factor(RIDRETH1)) + (1 | SDMVPSU),
```

```
  family = poisson(),
  data = nhanes_sample,
  weights = svywght_a
)
```

`se_ybar` *Standard error of sample mean for mixed models*

Description

Compute the standard error for the sample mean for mixed models, regarding the extent to which clustering affects the standard errors. May be used as part of the multilevel power calculation for cluster sampling (see *Gelman and Hill 2007, 447ff*).

Usage

```
se_ybar(fit)
```

Arguments

`fit`	Fitted mixed effects model (`merMod`-class).

Value

The standard error of the sample mean of `fit`.

References

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

Examples

```
library(lme4)
fit <- lmer(Reaction ~ 1 + (1 | Subject), sleepstudy)
se_ybar(fit)
```

`std_beta` *Standardized beta coefficients and CI of linear and mixed models*

Description

Returns the standardized beta coefficients, std. error and confidence intervals of a fitted linear (mixed) models.

Usage

```
std_beta(fit, ...)

## S3 method for class 'merMod'
std_beta(fit, ci.lvl = 0.95, ...)

## S3 method for class 'lm'
std_beta(fit, type = "std", ci.lvl = 0.95, ...)

## S3 method for class 'gls'
std_beta(fit, type = "std", ci.lvl = 0.95, ...)
```

Arguments

`fit`	Fitted linear (mixed) model of class `lm`, `merMod` (**lme4** package), `gls` or `stanreg`.
`...`	Currently not used.
`ci.lvl`	Numeric, the level of the confidence intervals.
`type`	If `fit` is of class `lm`, normal standardized coefficients are computed by default. Use `type = "std2"` to follow Gelman's (2008) suggestion, rescaling the estimates by deviding them by two standard deviations, so resulting coefficients are directly comparable for untransformed binary predictors.

Details

"Standardized coefficients refer to how many standard deviations a dependent variable will change, per standard deviation increase in the predictor variable. Standardization of the coefficient is usually done to answer the question of which of the independent variables have a greater effect on the dependent variable in a multiple regression analysis, when the variables are measured in different units of measurement (for example, income measured in dollars and family size measured in number of individuals)" *(Source: Wikipedia)*

Value

A `tibble` with term names, standardized beta coefficients, standard error and confidence intervals of `fit`.

Note

For gls-objects, standardized beta coefficients may be wrong for categorical variables (factors), because the model.matrix for gls objects returns the original data of the categorical vector, and not the 'dummy' coded vectors as for other classes. See, as example:

```
head(model.matrix(lm(neg_c_7 ~ as.factor(e42dep),data = efc,na.action = na.omit)))
```

and

```
head(model.matrix(nlme::gls(neg_c_7 ~ as.factor(e42dep),data = efc,na.action = na.omit))).
```

In such cases, use to_dummy to create dummies from factors.

References

Wikipedia: Standardized coefficient

Gelman A. 2008. Scaling regression inputs by dividing by two standard deviations. Statistics in Medicine 27: 2865-2873 http://www.stat.columbia.edu/~gelman/research/published/standardizing7.pdf

Examples

```
# fit linear model
fit <- lm(Ozone ~ Wind + Temp + Solar.R, data = airquality)
# print std. beta coefficients
std_beta(fit)

# print std. beta coefficients and ci, using
# 2 sd and center binary predictors
std_beta(fit, type = "std2")

# std. beta for mixed models
library(lme4)
fit1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
std_beta(fit)
```

---

svyglm.nb *Survey-weighted negative binomial generalised linear model*

Description

svyglm.nb() is an extension to the **survey**-package to fit survey-weighted negative binomial models. It uses svymle to fit sampling-weighted maximum likelihood estimates, based on starting values provided by glm.nb, as proposed by *Lumley (2010, pp249)*.

Usage

```
svyglm.nb(formula, design, ...)
```

Arguments

formula	An object of class formula, i.e. a symbolic description of the model to be fitted. See 'Details' in glm.
design	An object of class svydesign, providing a specification of the survey design.
...	Other arguments passed down to glm.nb.

Details

For details on the computation method, see Lumley (2010), Appendix E (especially 254ff.)

sjstats implements following S3-methods for svyglm.nb-objects: family(), model.frame(), formula(), print(), predict() and residuals(). However, these functions have some limitations:

- family() simply returns the family-object from the underlying glm.nb-model.
- The predict()-method just re-fits the svyglm.nb-model with glm.nb, overwrites the $coefficients from this model-object with the coefficients from the returned svymle-object and finally calls predict.glm to compute the predicted values.
- residuals() re-fits the svyglm.nb-model with glm.nb and then computes the Pearson-residuals from the glm.nb-object.

Value

An object of class svymle and svyglm.nb, with some additional information about the model.

References

Lumley T (2010). Complex Surveys: a guide to analysis using R. Wiley

Examples

```
# ----------------------------------------
# This example reproduces the results from
# Lumley 2010, figure E.7 (Appendix E, p256)
# ----------------------------------------
library(survey)
data(nhanes_sample)

# create survey design
des <- svydesign(
  id = ~SDMVPSU,
  strat = ~SDMVSTRA,
  weights = ~WTINT2YR,
  nest = TRUE,
  data = nhanes_sample
)
```

```
# fit negative binomial regression
fit <- svyglm.nb(total ~ factor(RIAGENDR) * (log(age) + factor(RIDRETH1)), des)

# print coefficients and standard errors
fit
```

---

svyglm.zip *Survey-weighted zero-inflated Poisson model*

Description

svyglm.zip() is an extension to the **survey**-package to fit survey-weighted zero-inflated Poisson models. It uses svymle to fit sampling-weighted maximum likelihood estimates, based on starting values provided by zeroinfl.

Usage

```
svyglm.zip(formula, design, ...)
```

Arguments

formula	An object of class formula, i.e. a symbolic description of the model to be fitted. See 'Details' in zeroinfl.
design	An object of class svydesign, providing a specification of the survey design.
...	Other arguments passed down to zeroinfl.

Details

Code modified from https://notstatschat.rbind.io/2015/05/26/zero-inflated-poisson-from-complex-samples/.

Value

An object of class svymle and svyglm.zip, with some additional information about the model.

Examples

```
library(survey)
data(nhanes_sample)
set.seed(123)
nhanes_sample$malepartners <- rpois(nrow(nhanes_sample), 2)
nhanes_sample$malepartners[sample(1:2992, 400)] <- 0

# create survey design
des <- svydesign(
  id = ~SDMVPSU,
```

```
  strat = ~SDMVSTRA,
  weights = ~WTINT2YR,
  nest = TRUE,
  data = nhanes_sample
)

# fit negative binomial regression
fit <- svyglm.zip(
  malepartners ~ age + factor(RIDRETH1) | age + factor(RIDRETH1),
  des
)

# print coefficients and standard errors
fit
```

---

svy_md *Weighted statistics for tests and variables*

Description

Weighted statistics for variables

`wtd_sd()`, `wtd_se()`, `wtd_mean()` and `wtd_median()` compute weighted standard deviation, standard error, mean or median for a variable or for all variables of a data frame. `svy_md()` computes the median for a variable in a survey-design (see `svydesign`). `wtd_cor()` computes a weighted correlation for a two-sided alternative hypothesis.

Weighted tests

`wtd_ttest()` computes a weighted t-test, while `wtd_mwu()` computes a weighted Mann-Whitney-U test or a Kruskal-Wallis test (for more than two groups). `wtd_chisqtest()` computes a weighted Chi-squared test for contigency tables.

Usage

```
svy_md(x, design)

survey_median(x, design)

wtd_chisqtest(data, ...)

## Default S3 method:
wtd_chisqtest(data, x, y, weights, ...)

## S3 method for class 'formula'
wtd_chisqtest(formula, data, ...)
```

```
wtd_cor(data, ...)

## Default S3 method:
wtd_cor(data, x, y, weights, ci.lvl = 0.95, ...)

## S3 method for class 'formula'
wtd_cor(formula, data, ci.lvl = 0.95, ...)

wtd_mean(x, weights = NULL)

wtd_median(x, weights = NULL)

wtd_mwu(data, ...)

## Default S3 method:
wtd_mwu(data, x, grp, weights, ...)

## S3 method for class 'formula'
wtd_mwu(formula, data, ...)

wtd_sd(x, weights = NULL)

wtd_se(x, weights = NULL)

wtd_ttest(data, ...)

## Default S3 method:
wtd_ttest(data, x, y = NULL, weights, mu = 0,
  paired = FALSE, ci.lvl = 0.95, alternative = c("two.sided", "less",
  "greater"), ...)

## S3 method for class 'formula'
wtd_ttest(formula, data, mu = 0, paired = FALSE,
  ci.lvl = 0.95, alternative = c("two.sided", "less", "greater"), ...)
```

Arguments

x	(Numeric) vector or a data frame. For `svy_md()`, `wtd_ttest()`, `wtd_mwu()` and `wtd_chisqtest()` the bare (unquoted) variable name, or a character vector with the variable name.
design	An object of class `svydesign`, providing a specification of the survey design.
data	A data frame.
...	For `wtd_ttest()` and `wtd_mwu()`, currently not used. For `wtd_chisqtest()`, further arguments passed down to `chisq.test`.
y	Optional, bare (unquoted) variable name, or a character vector with the variable name.
weights	Bare (unquoted) variable name, or a character vector with the variable name

	of the numeric vector of weights. If `weights = NULL`, unweighted statistic is reported.
`formula`	A formula of the form `lhs ~ rhs1 + rhs2` where `lhs` is a numeric variable giving the data values and `rhs1` a factor with two levels giving the corresponding groups and `rhs2` a variable with weights.
`ci.lvl`	Confidence level of the interval.
`grp`	Bare (unquoted) name of the cross-classifying variable, where `x` is grouped into the categories represented by `grp`, or a character vector with the variable name.
`mu`	A number indicating the true value of the mean (or difference in means if you are performing a two sample test).
`paired`	Logical, whether to compute a paired t-test.
`alternative`	A character string specifying the alternative hypothesis, must be one of `"two.sided"` (default), `"greater"` or `"less"`. You can specify just the initial letter.

Value

The weighted (test) statistic.

Note

`wtd_chisq()` is a convenient wrapper for `xtab_statistics`. For a weighted one-way Anova, use `grpmean()` with `weights`-argument.

`wtd_ttest()` assumes unequal variance between the two groups.

Examples

```
# weighted sd and se ----

wtd_sd(rnorm(n = 100, mean = 3), runif(n = 100))

data(efc)
wtd_sd(efc[, 1:3], runif(n = nrow(efc)))
wtd_se(efc[, 1:3], runif(n = nrow(efc)))

# svy_md ----

# median for variables from weighted survey designs
library(survey)
data(nhanes_sample)

des <- svydesign(
  id = ~SDMVPSU,
  strat = ~SDMVSTRA,
  weights = ~WTINT2YR,
  nest = TRUE,
  data = nhanes_sample
)
```

```
svy_md(total, des)
svy_md("total", des)

# weighted t-test ----

efc$weight <- abs(rnorm(nrow(efc), 1, .3))
wtd_ttest(efc, e17age, weights = weight)
wtd_ttest(efc, e17age, c160age, weights = weight)
wtd_ttest(e17age ~ e16sex + weight, efc)

# weighted Mann-Whitney-U-test ----

wtd_mwu(c12hour ~ c161sex + weight, efc)

# weighted Chi-squared-test ----

wtd_chisqtest(efc, c161sex, e16sex, weights = weight, correct = FALSE)
wtd_chisqtest(c172code ~ c161sex + weight, efc)
```

`table_values` *Expected and relative table values*

Description

This function calculates a table's cell, row and column percentages as well as expected values and returns all results as lists of tables.

Usage

```
table_values(tab, digits = 2)
```

Arguments

`tab`	Simple `table` or `ftable` of which cell, row and column percentages as well as expected values are calculated. Tables of class `xtabs` and other will be coerced to `ftable` objects.
`digits`	Amount of digits for the table percentage values.

Value

(Invisibly) returns a list with four tables:

1. `cell` a table with cell percentages of `tab`
2. `row` a table with row percentages of `tab`
3. `col` a table with column percentages of `tab`
4. `expected` a table with expected values of `tab`

Examples

```
tab <- table(sample(1:2, 30, TRUE), sample(1:3, 30, TRUE))
# show expected values
table_values(tab)$expected
# show cell percentages
table_values(tab)$cell
```

`tidy_stan` *Tidy summary output for stan models*

Description

Returns a tidy summary output for stan models.

Usage

```
tidy_stan(x, prob = 0.89, typical = "median", trans = NULL,
  effects = c("all", "fixed", "random"), component = c("all",
  "conditional", "zero_inflated", "zi"), digits = 2)
```

Arguments

`x`	A `stanreg`, `stanfit` or `brmsfit` object.
`prob`	Vector of scalars between 0 and 1, indicating the mass within the credible interval that is to be estimated.
`typical`	The typical value that will represent the Bayesian point estimate. By default, the posterior median is returned. See `typical_value` for possible values for this argument.
`trans`	Name of a function or character vector naming a function, used to apply transformations on the estimates and uncertainty intervals. The values for standard errors are *not* transformed! If `trans` is not `NULL`, *credible intervals* instead of *HDI* are computed, due to the possible asymmetry of the HDI.
`effects`	Should results for fixed effects, random effects or both be returned? Only applies to mixed models. May be abbreviated.
`component`	Should results for all parameters, parameters for the conditional model or the zero-inflated part of the model be returned? May be abbreviated. Only applies to **brms**-models.
`digits`	Amount of digits to round numerical values in the output.

Details

The returned data frame has an additonal class-attribute, `tidy_stan`, to pass the result to its own `print()`-method. The `print()`-method creates a cleaner output, especially for multilevel, zero-inflated or multivariate response models, where - for instance - the conditional part of a model is printed separately from the zero-inflated part, or random and fixed effects are printed separately.

The returned data frame gives information on:

- The Bayesian point estimate (column *estimate*, which is by default the posterior median; other statistics are also possible, see argument `typical`).
- The standard error (which is actually the *median absolute deviation*).
- The HDI. Computation for HDI is based on the code from Kruschke 2015, pp. 727f.
- The Probability of Direction (pd), which is an index for "effect significance" (see *Makowski et al. 2019*). A value of 95% or higher indicates a "significant" (i.e. statistically clear) effect.
- The effective numbers of samples, *ESS*.
- The Rhat statistics. When Rhat is above 1, it usually indicates that the chain has not yet converged, indicating that the drawn samples might not be trustworthy. Drawing more iteration may solve this issue.
- The Monte Carlo standard error (see `mcse`). It is defined as standard deviation of the chains divided by their effective sample size and "provides a quantitative suggestion of how big the estimation noise is" (*Kruschke 2015, p.187*).

Value

A data frame, summarizing `x`, with consistent column names. To distinguish multiple HDI values, column names for the HDI get a suffix when `prob` has more than one element.

References

Kruschke JK. *Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan* 2nd edition. Academic Press, 2015

Gelman A, Carlin JB, Stern HS, Dunson DB, Vehtari A, Rubin DB. *Bayesian data analysis* 3rd ed. Boca Raton: Chapman and Hall/CRC, 2013

Gelman A, Rubin DB. *Inference from iterative simulation using multiple sequences* Statistical Science 1992;7: 457-511

Makowski D, Ben-Shachar MS, Lüdecke D. bayestestR: Describing Effects and their Uncertainty, Existence and Significance within the Bayesian Framework. Journal of Open Source Software 2019;4:1541. doi: 10.21105/joss.01541

McElreath R. *Statistical Rethinking. A Bayesian Course with Examples in R and Stan* Chapman and Hall, 2015

Examples

```
## Not run:
if (require("rstanarm")) {
  fit <- stan_glm(mpg ~ wt + am, data = mtcars, chains = 1)
  tidy_stan(fit)
  tidy_stan(fit, prob = c(.89, .5))
}
## End(Not run)
```

---

var_pop *Calculate population variance and standard deviation*

Description

Calculate the population variance or standard deviation of a vector.

Usage

```
var_pop(x)

sd_pop(x)
```

Arguments

x	(Numeric) vector.

Details

Unlike var, which returns the sample variance, var_pop() returns the population variance. sd_pop() returns the standard deviation based on the population variance.

Value

The population variance or standard deviation of x.

Examples

```
data(efc)

# sampling variance
var(efc$c12hour, na.rm = TRUE)
# population variance
var_pop(efc$c12hour)

# sampling sd
sd(efc$c12hour, na.rm = TRUE)
# population sd
sd_pop(efc$c12hour)
```

`weight` *Weight a variable*

Description

These functions weight the variable `x` by a specific vector of `weights`.

Usage

```
weight(x, weights, digits = 0)

weight2(x, weights)
```

Arguments

`x`	(Unweighted) variable.
`weights`	Vector with same length as `x`, which contains weight factors. Each value of `x` has a specific assigned weight in `weights`.
`digits`	Numeric value indicating the number of decimal places to be used for rounding the weighted values. By default, this value is `0`, i.e. the returned values are integer values.

Details

`weight2()` sums up all `weights` values of the associated categories of `x`, whereas `weight()` uses a `xtabs` formula to weight cases. Thus, `weight()` may return a vector of different length than `x`.

Value

The weighted `x`.

Note

The values of the returned vector are in sorted order, whereas the values' order of the original `x` may be spread randomly. Hence, `x` can't be used, for instance, for further cross tabulation. In case you want to have weighted contingency tables or (grouped) box plots etc., use the `weightBy` argument of most functions.

Examples

```
v <- sample(1:4, 20, TRUE)
table(v)
w <- abs(rnorm(20))
table(weight(v, w))
table(weight2(v, w))

set.seed(1)
x <- sample(letters[1:5], size = 20, replace = TRUE)
```

```
w <- runif(n = 20)

table(x)
table(weight(x, w))
```

Index