Package ‘sjstats’

November 19, 2022

Type Package

Encoding UTF-8

Title Collection of Convenient Functions for Common Statistical Computations

Version 0.18.2

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 (>= 3.4), utils

Imports bayestestR, broom, datawizard, dplyr, effectsize, emmeans, insight, lme4, magrittr, MASS, modelr, parameters, performance, purrr, rlang, sjlabelled, sjmisc, stats, tidyr

Suggests brms, car, coin, ggplot2, graphics, pscl, pwr, sjPlot, survey, rstan, testthat

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

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

RoxygenNote 7.2.2

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

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

Repository CRAN

Date/Publication 2022-11-19 22:10:02 UTC

R topics documented:

`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)
```

Arguments

model	A fitted anova-model of class aov or anova. Other models are coerced to anova.
digits	Amount of digits for returned values.

Value

A 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.

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

Examples

```
# load sample data
data(efc)

# fit linear model
fit <- aov(
  c12hour ~ as.factor(e42dep) + as.factor(c172code) + c160age,
  data = efc
)
## Not run:
anova_stats(car::Anova(fit, type = 2))

## End(Not run)
```

---

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
if (require("dplyr") && require("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 data frame 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.

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.

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, ...)

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

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 `crosstable_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
crosstable_statistics(efc, e16sex, c161sex)

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

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

# calcilate Spearman's rho, with continuity correction
crosstable_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.

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]]))
```

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)
```

`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/jech-2015207018

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))
```

`means_by_group` *Summary of mean values by group*

Description

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

Usage

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

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, `means_by_group()` 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)
means_by_group(efc, c12hour, e42dep)

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

# also works for grouped data frames
if (require("dplyr")) {
  efc %>%
    group_by(c172code) %>%
    means_by_group(c12hour, e42dep)
}

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

---

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
```

---

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
)

mannwhitney(
  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

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 data frame with one column of conditions and one column with proportions. For grouped data frames, returns a data frame 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 data frame
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'
)
```

r2 — *Deprecated functions*

Description

A list of deprecated functions.

Usage

```
r2(x)

cohens_f(x, ...)

eta_sq(x, ...)

epsilon_sq(x, ...)

omega_sq(x, ...)

scale_weights(x, ...)

robust(x, ...)

icc(x)

p_value(x, ...)

se(x, ...)
```

Arguments

x	An object.
...	Currently not used.

Value

Nothing.

`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.

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.

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)
```

---

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

```
if (require("lme4")) {
  fit <- lmer(Reaction ~ 1 + (1 | Subject), sleepstudy)
  se_ybar(fit)
}
```

---

survey_median — *Weighted statistics for tests and variables*

Description

Weighted statistics for variables

weighted_sd(), weighted_se(), weighted_mean() and weighted_median() compute weighted standard deviation, standard error, mean or median for a variable or for all variables of a data frame. survey_median() computes the median for a variable in a survey-design (see svydesign). weighted_correlation() computes a weighted correlation for a two-sided alternative hypothesis.

Weighted tests

weighted_ttest() computes a weighted t-test, while weighted_mannwhitney() computes a weighted Mann-Whitney-U test or a Kruskal-Wallis test (for more than two groups). weighted_chisqtest() computes a weighted Chi-squared test for contigency tables.

Usage

```
survey_median(x, design)

weighted_chisqtest(data, ...)

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

## S3 method for class 'formula'
```

```
weighted_chisqtest(formula, data, ...)

weighted_correlation(data, ...)

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

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

weighted_mean(x, weights = NULL)

weighted_median(x, weights = NULL)

weighted_mannwhitney(data, ...)

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

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

weighted_sd(x, weights = NULL)

wtd_sd(x, weights = NULL)

weighted_se(x, weights = NULL)

weighted_ttest(data, ...)

## Default S3 method:
weighted_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'
weighted_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 `survey_median()`, `weighted_ttest()`, `weighted_mannwhitney()` and `weighted_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 `weighted_ttest()` and `weighted_mannwhitney()`, currently not used. For `weighted_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

`weighted_chisq()` is a convenient wrapper for `crosstable_statistics`. For a weighted one-way Anova, use `means_by_group()` with `weights`-argument.

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

Examples

```
# weighted sd and se ----
weighted_sd(rnorm(n = 100, mean = 3), runif(n = 100))

data(efc)
weighted_sd(efc[, 1:3], runif(n = nrow(efc)))
weighted_se(efc[, 1:3], runif(n = nrow(efc)))

# survey_median ----
# median for variables from weighted survey designs
if (require("survey")) {
  data(nhanes_sample)

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

  survey_median(total, des)
  survey_median("total", des)
}

# weighted t-test ----
efc$weight <- abs(rnorm(nrow(efc), 1, .3))
weighted_ttest(efc, e17age, weights = weight)
weighted_ttest(efc, e17age, c160age, weights = weight)
weighted_ttest(e17age ~ e16sex + weight, efc)

# weighted Mann-Whitney-U-test ----
weighted_mannwhitney(c12hour ~ c161sex + weight, efc)

# weighted Chi-squared-test ----
weighted_chisqtest(efc, c161sex, e16sex, weights = weight, correct = FALSE)
weighted_chisqtest(c172code ~ c161sex + weight, efc)

# weighted Chi-squared-test for given probabilities ----
weighted_chisqtest(c172code ~ weight, efc, p = c(.33, .33, .34))
```

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

```
if (require("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
}
```

`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
```

---

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