Package 'sjstats'

February 4, 2018

Type Package

Encoding UTF-8

Title Collection of Convenient Functions for Common Statistical Computations

Version 0.14.1

Date 2018-02-05

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

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 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' or 'lme' objects). The focus of most functions lies on summary statistics or fit measures for regression models, including generalized linear models and mixed effects models. However, some of the functions also deal with other statistical measures, like Cronbach's Alpha, Cramer's V, Phi etc.

License GPL-3

Depends R (>= 3.2), stats, utils

Imports bayesplot, broom, cli, coin, crayon, dplyr (>= 0.7.1), emmeans, lme4 (>= 1.1-12), lmtest (>= 0.9-34), glmmTMB, magrittr, MASS, Matrix, modelr, nlme, prediction, purrr (>= 0.2.2), pwr, rlang, sandwich (>= 2.3-4), sjlabelled (>= 1.0.6), sjmisc (>= 2.6.3), tidyr (>= 0.7.0), tidyselect, tibble (>= 1.3.3)

Suggests AER, arm, brms, car, ggplot2, graphics, Hmisc, knitr, lmerTest, pbkrtest (>= 0.4-7), pROC, psych, sjPlot, survey, rstan, rstanarm, tidyverse, Zelig

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

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

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2018-02-04 15:31:34 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 or lme 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. The focus of most functions lies on summary statistics or fit measures for regression models, including generalized linear models and mixed effects 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
- Fit and accuracy measures for regression models: Overdispersion tests, accuracy of predictions, test/training-error comparisons
- For anova-tables: Eta-squared, Partial Eta-squared and Omega-squared statistics

Other statistics:

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

---

`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 `tibble` 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(tidyverse)
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)
# standard error of original variable
se(efc$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"))

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'. May be abbreviated.

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))).
- 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(tidyverse)
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)
```

`check_assumptions` *Check model assumptions*

Description

- `outliers()` detects outliers in (generalized) linear models.
- `heteroskedastic()` checks a linear model for (non-)constant error variance.
- `autocorrelation()` checks for independence of errors.
- `normality()` checks linear models for (non-)normality of residuals.
- `multicollin()` checks predictors of linear models for multicollinearity.
- `check_assumptions()` checks all of the above assumptions.

Usage

```
check_assumptions(x, model.column = NULL, as.logical = FALSE, ...)

outliers(x, iterations = 5)

heteroskedastic(x, model.column = NULL)

autocorrelation(x, model.column = NULL, ...)

normality(x, model.column = NULL)

multicollin(x, model.column = NULL)
```

Arguments

x	Fitted lm (for outliers(), may also be a glm model), or a (nested) data frame with a list-variable that contains fitted model objects.
model.column	Name or index of the list-variable that contains the fitted model objects. Only applies, if x is a nested data frame (e.g with models fitted to bootstrap replicates).
as.logical	Logical, if TRUE, the values returned by check_assumptions() are TRUE or FALSE, indicating whether each violation of model assumotion holds true or not. If FALSE (the default), the p-value of the respective test-statistics is returned.
...	Other arguments, passed down to durbinWatsonTest.
iterations	Numeric, indicates the number of iterations to remove outliers.

Details

These functions are wrappers that compute various test statistics, however, each of them returns a tibble instead of a list of values. Furthermore, all functions can also be applied to multiples models in stored in *list-variables* (see 'Examples').

outliers() wraps outlierTest and iteratively removes outliers for iterations times, or if the r-squared value (for glm: the AIC) did not improve after removing outliers. The function returns a tibble with r-squared and AIC statistics for the original and updated model, as well as the update model itself ($updated.model), the number ($removed.count) and indices of the removed observations ($removed.obs).

heteroskedastic() wraps ncvTest and returns the p-value of the test statistics as tibble. A p-value < 0.05 indicates a non-constant variance (heteroskedasticity).

autocorrelation() wraps durbinWatsonTest and returns the p-value of the test statistics as tibble. A p-value < 0.05 indicates autocorrelated residuals. In such cases, robust standard errors (see robust return more accurate results for the estimates, or maybe a mixed model with error term for the cluster groups should be used.

normality() calls shapiro.test and checks the standardized residuals for normal distribution.

The p-value of the test statistics is returned as tibble. A p-value < 0.05 indicates a significant deviation from normal distribution. Note that this formal test almost always yields significant results for the distribution of residuals and visual inspection (e.g. qqplots) are preferable (see `sjp.lm` with `type = "ma"`).

`multicollin()` wraps `vif` and returns the logical result as tibble. `TRUE`, if multicollinearity exists, else not. In case of multicollinearity, the names of independent variables that vioalte contribute to multicollinearity are printed to the console.

`check_assumptions()` runs all of the above tests and returns a tibble with all test statistics included. In case the p-values are too confusing, use the `as.logical` argument, where all p-values are replaced with either `TRUE` (in case of violation) or `FALSE` (in case of model conforms to assumption of linar regression).

Value

A tibble with the respective statistics.

Note

These formal tests are very strict and in most cases violation of model assumptions are alerted, though the model is actually ok. It is preferable to check model assumptions based on visual inspection (see `sjp.lm` with `type = "ma"`).

Examples

```
data(efc)

fit <- lm(barthtot ~ c160age + c12hour + c161sex + c172code, data = efc)
outliers(fit)
heteroskedastic(fit)
autocorrelation(fit)
normality(fit)
check_assumptions(fit)

fit <- lm(barthtot ~ c160age + c12hour + c161sex + c172code + neg_c_7,
          data = efc)
outliers(fit)
check_assumptions(fit, as.logical = TRUE)

# apply function to multiple models in list-variable
library(purrr)
library(dplyr)
tmp <- efc %>%
  bootstrap(50) %>%
  mutate(
    models = map(strap, ~lm(neg_c_7 ~ e42dep + c12hour + c161sex, data = .x))
  )

# for list-variables, argument 'model.column' is the
# quoted name of the list-variable with fitted models
```

```
tmp %>% normality("models")
tmp %>% heteroskedastic("models")

# Durbin-Watson-Test from package 'car' takes a little bit longer due
# to simulation of p-values...
## Not run:
tmp %>% check_assumptions("models", as.logical = TRUE, reps = 100)
## End(Not run)
```

---

chisq_gof *Chi-square goodness-of-fit-test*

Description

This method performs a Chi-square goodness-of-fit-test (GOF) either on a numeric vector against probabilities, or a Goodness-of-fit test for glm-objects for binary data.

Usage

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

Arguments

x	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.

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
- X2 the pearson chi-squared statistic

Note

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 based on the `X2GOFtest` function of the **binomTools** package. 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.

Examples

```
data(efc)
# 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)))

# goodness-of-fit test for logistic regression
efc$services <- ifelse(efc$tot_sc_e > 0, 1, 0)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep, data = efc,
           family = binomial(link = "logit"))
chisq_gof(fit)
```

---

cod *Tjur's Coefficient of Discrimination*

Description

This method calculates the Coefficient of Discrimination `D` for generalized linear (mixed) models for binary data. It is an alternative to other Pseudo-R-squared values like Nakelkerke's R2 or Cox-Snell R2.

Usage

```
cod(x)
```

Arguments

`x`	Fitted `glm` or `glmer` model.

Value

The `D` Coefficient of Discrimination, also known as Tjur's R-squared value.

Note

The Coefficient of Discrimination `D` can be read like any other (Pseudo-)R-squared value.

References

Tjur T (2009) Coefficients of determination in logistic regression models - a new proposal: The coefficient of discrimination. The American Statistician, 63(4): 366-372

See Also

r2 for Nagelkerke's and Cox and Snell's pseudo r-squared coefficients.

Examples

```
library(sjmisc)
data(efc)

# Tjur's R-squared value
efc$services <- ifelse(efc$tot_sc_e > 0, 1, 0)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep,
           data = efc, family = binomial(link = "logit"))
cod(fit)
```

---

converge_ok *Convergence test for mixed effects models*

Description

converge_ok() provides an alternative convergence test for merMod-objects; is_singular() checks post-fitting convergence warnings. If the model fit is singular, warning about negative eigenvalues of the Hessian can most likely be ignored.

Usage

```
converge_ok(x, tolerance = 0.001)

is_singular(x, tolerance = 1e-05)
```

Arguments

x	A merMod-object. For is_singluar(), may also be a glmmTMB-object.
tolerance	Indicates up to which value the convergence result is accepted. The smaller tolerance is, the stricter the test will be.

Details

converge_ok() provides an alternative convergence test for merMod-objects, as discussed here and suggested by Ben Bolker in this comment.

is_singular() checks if a model fit is singular, and can be used in case of post-fitting convergence warnings, such as warnings about negative eigenvalues of the Hessian. If the fit is singular (i.e. is_singular() returns TRUE), these warnings can most likely be ignored.

Value

For converge_ok(), a logical vector, which is TRUE if convergence is fine and FALSE if convergence is suspicious. Additionally, the convergence value is returned as return value's name. is_singluar() returns TRUE if the model fit is singular.

Examples

```
library(sjmisc)
library(lme4)
data(efc)
# create binary response
efc$hi_qol <- dicho(efc$quol_5)
# prepare group variable
efc$grp = as.factor(efc$e15relat)
# data frame for fitted model
mydf <- data.frame(hi_qol = as.factor(efc$hi_qol),
                   sex = as.factor(efc$c161sex),
                   c12hour = as.numeric(efc$c12hour),
                   neg_c_7 = as.numeric(efc$neg_c_7),
                   grp = efc$grp)
# fit glmer
fit <- glmer(hi_qol ~ sex + c12hour + neg_c_7 + (1|grp),
             data = mydf, family = binomial("logit"))

converge_ok(fit)
```

---

cv *Coefficient of Variation*

Description

Compute coefficient of variation for single variables (standard deviation divided by mean) or for fitted linear (mixed effects) models (root mean squared error (RMSE) divided by mean of dependent variable).

Usage

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

Arguments

x	(Numeric) vector or a 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

The coefficient of variation of `x`.

References

Everitt, Brian (1998). The Cambridge Dictionary of Statistics. Cambridge, UK New York: Cambridge University Press

See Also

`rmse`

Examples

```
data(efc)
cv(efc$e17age)

fit <- lm(neg_c_7 ~ e42dep, data = efc)
cv(fit)

library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
cv(fit)

library(nlme)
fit <- lme(Reaction ~ Days, random = ~ Days | Subject, data = sleepstudy)
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 tibble with the root mean squared errors for the training and test data.

See Also

`pred_accuracy`

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
))
```

---

deff *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

```
deff(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 & 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 & 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.
deff(n = 30)

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

efc *Sample dataset from the EUROFAMCARE project*

Description

A SPSS sample data set, imported with the `read_spss` function.

Examples

```
# Attach EFC-data
data(efc)

# Show structure
str(efc)

# show first rows
head(efc)

# show variables
## Not run:
library(sjmisc)
library(sjPlot)
view_df(efc)

# show variable labels
get_label(efc)

# plot efc-data frame summary
sjt.df(efc, alternateRowColor = TRUE)
## End(Not run)
```

eta_sq *Effect size statistics for anova*

Description

Returns the (partial) eta-squared, omega-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

```
eta_sq(model, partial = FALSE)

omega_sq(model)

cohens_f(model)

anova_stats(model, digits = 3)
```

Arguments

model	A fitted anova-model of class aov or anova. Other models are coerced to anova.
partial	Logical, if TRUE, the partial eta-squared is returned.
digits	Number of decimal points in the returned data frame.

Value

A numeric vector with the effect size statistics; for anova_stats(), a tidy data frame with all these statistics.

References

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

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

---

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)
gmd(efc, tidyselect::contains("cop"))
```

---

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, weight.by = NULL, digits = 2, out = c("txt", "viewer",
  "browser"))
```

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.
weight.by	Vector of weights that will be applied to weight all cases. Must be a vector of same length as the input vector. 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"`).

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 sub-group. 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)
```

---

hdi *Compute statistics for MCMC samples*

Description

`hdi()` computes the high density interval for values from MCMC samples. `rope()` calculates the proportion of a posterior distribution that lies within a region of practical equivalence. `n_eff()` calculates the number of effective samples (effective sample size). `mcse()` returns the Monte Carlo standard error.

Usage

```
hdi(x, prob = 0.9, trans = NULL, type = c("fixed", "random", "all"))

mcse(x, type = c("fixed", "random", "all"))
```

```
n_eff(x, type = c("fixed", "random", "all"))

rope(x, rope, trans = NULL, type = c("fixed", "random", "all"))
```

Arguments

x	A stanreg, stanfit, or brmsfit object. For hdi() and rope(), may also be a vector of values from a probability distribution (e.g., posterior probabilities from MCMC sampling).
prob	Scalar between 0 and 1, indicating the mass within the credible interval that is to be estimated.
trans	Name of a function or character vector naming a function, used to apply transformations on the returned HDI-values resp. (for rope()) on the values of the posterior distribution, before calculating the rope based on the boundaries given in rope. Note that the values in rope are not transformed.
type	For mixed effects models, specify the type of effects that should be returned. type = "fixed" returns fixed effects only, type = "random" the random effects and type = "all" returns both fixed and random effects.
rope	Vector of length two, indicating the lower and upper limit of a range around zero, which indicates the region of practical equivalence. Values of the posterior distribution within this range are considered as being "practically equivalent to zero".

Details

Computation for HDI is based on the code from Kruschke 2015, pp. 727f. For default sampling in Stan (4000 samples), the 90% intervals for HDI are more stable than, for instance, 95% intervals. An effective sample size (see n_eff()) of at least 10.000 is recommended if 95% intervals should be computed (see Kruschke 2015, p. 183ff).

The formula for mcse() is from Kruschke 2015, p. 187.

For n_eff(), the ratio of effective number of samples ranges from 0 to 1, and should be close to 1. The closer this ratio comes to zero means that the chains may be inefficient, but possibly still okay.

Value

For hdi(), if x is a vector, returns a vector of length two with the lower and upper limit of the HDI; if x is a stanreg, stanfit or brmsfit object, returns a tibble with lower and upper HDI-limits for each predictor. For rope(), returns the proportion of values from x that are within the boundaries of rope. mcse() and n_eff() return a tibble with two columns: one with the term names and one with the related statistic.

References

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

Examples

```
## Not run:
if (require("rstanarm")) {
  fit <- stan_glm(mpg ~ wt + am, data = mtcars, chains = 1)
  hdi(fit)

  # fit logistic regression model
  fit <- stan_glm(
    vs ~ wt + am,
    data = mtcars,
    family = binomial("logit"),
    chains = 1
  )
  # compute hdi, transform on "odds ratio scale"
  hdi(fit, trans = exp)

  # compute rope, on scale of linear predictor. finds proportion
  # of posterior distribution values between -1 and 1.
  rope(fit, rope = c(-1, 1))

  # compute rope, boundaries as "odds ratios". finds proportion of
  # posterior distribution values, which - after being exponentiated -
  # are between .8 and 1.25 (about -.22 and .22 on linear scale)
  rope(fit, rope = c(.8, 1.25), trans = exp)
}
## End(Not run)
```

---

hoslem_gof *Hosmer-Lemeshow Goodness-of-fit-test*

Description

This method performs a Hosmer-Lemeshow goodness-of-fit-test for generalized linear (mixed) models for binary data.

Usage

```
hoslem_gof(x, g = 10)
```

Arguments

x	Fitted glm or glmer model.
g	Number of bins to divide the data. Default is 10.

Value

An object of class `hoslem_test` with following values:

- `chisq` the Hosmer-Lemeshow chi-squared statistic
- `df` degrees of freedom
- `p.value` the p-value for the goodness-of-fit test

Note

A well-fitting model shows no significant difference between the model and the observed data, i.e. the reported p-value should be greater than 0.05.

References

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

See Also

`r2`

Examples

```
data(efc)
# goodness-of-fit test for logistic regression
efc$services <- ifelse(efc$tot_sc_e > 0, 1, 0)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep, data = efc,
           family = binomial(link = "logit"))
hoslem_gof(fit)
```

---

icc *Intraclass-Correlation Coefficient*

Description

This function calculates the intraclass-correlation (icc) - sometimes also called *variance partition coefficient* (vpc) - for random intercepts of mixed effects models. Currently, `merMod`, `glmmTMB` `stanreg` and `brmsfit` objects are supported.

Usage

```
icc(x, ...)
```

Arguments

`x`	Fitted mixed effects model (of class `merMod`, `glmmTMB`, `stanreg` or `brmsfit`).
`...`	More fitted model objects, to compute multiple intraclass-correlation coefficients at once.

Details

The ICC is calculated by dividing the between-group-variance (random intercept variance) by the total variance (i.e. sum of between-group-variance and within-group (residual) variance).

The calculation of the ICC for generalized linear mixed models with binary outcome is based on *Wu et al. (2012)*. For Poisson multilevel models, please refere to *Stryhn et al. (2006)*. *Aly et al. (2014)* describe computation of ICC for negative binomial models.

There is a `print`-method that prints the variance parameters using the `comp`-argument set to `"var"`: `print(x, comp = "var")` (see 'Examples'). The `re_var`-function is a convenient wrapper.

The random effect variances indicate the between- and within-group variances as well as random-slope variance and random-slope-intercept correlation. The components are denoted as following:

- Within-group (residual) variance: sigma_2
- Between-group-variance: tau.00 (variation between individual intercepts and average intercept)
- Random-slope-variance: tau.11 (variation between individual slopes and average slope)
- Random-Intercept-Slope-covariance: tau.01
- Random-Intercept-Slope-correlation: rho.01

Value

A numeric vector with all random intercept intraclass-correlation-coefficients, or a list of numeric vectors, when more than one model were used as arguments. Furthermore, between- and within-group variances as well as random-slope variance are returned as attributes.

Note

Some notes on why the ICC is useful, based on *Grace-Martin*:

- It can help you determine whether or not a linear mixed model is even necessary. If you find that the correlation is zero, that means the observations within clusters are no more similar than observations from different clusters. Go ahead and use a simpler analysis technique.
- It can be theoretically meaningful to understand how much of the overall variation in the response is explained simply by clustering. For example, in a repeated measures psychological study you can tell to what extent mood is a trait (varies among people, but not within a person on different occasions) or state (varies little on average among people, but varies a lot across occasions).
- It can also be meaningful to see how the ICC (as well as the between and within cluster variances) changes as variable are added to the model.

In short, the ICC can be interpreted as "the proportion of the variance explained by the grouping structure in the population" *(Hox 2002: 15)*.

Usually, the ICC is calculated for the null model ("unconditional model"). However, according to *Raudenbush and Bryk (2002)* or *Rabe-Hesketh and Skrondal (2012)* it is also feasible to compute the ICC for full models with covariates ("conditional models") and compare how much a level-2

variable explains the portion of variation in the grouping structure (random intercept).

Caution: For three-level-models, depending on the nested structure of the model, the ICC only reports the proportion of variance explained for each grouping level. However, the proportion of variance for specific levels related to each other (e.g., similarity of level-1-units within level-2-units or level-2-units within level-3-units) must be computed manually. Use `get_re_var` to get the between-group-variances and residual variance of the model, and calculate the ICC for the various level correlations.

For example, for the ICC between level 1 and 2:

```
sum(get_re_var(fit)) / (sum(get_re_var(fit)) + get_re_var(fit, "sigma_2"))
```

or for the ICC between level 2 and 3:

```
get_re_var(fit)[2] / sum(get_re_var(fit))
```

References

- Aguinis H, Gottfredson RK, Culpepper SA. 2013. Best-Practice Recommendations for Estimating Cross-Level Interaction Effects Using Multilevel Modeling. Journal of Management 39(6): 1490–1528 (doi: 10.1177/0149206313478188)
- Aly SS, Zhao J, Li B, Jiang J. 2014. Reliability of environmental sampling culture results using the negative binomial intraclass correlation coefficient. Springerplus 14(3) (doi: 10.1186/21931801340)
- Grace-Martion K. The Intraclass Correlation Coefficient in Mixed Models, web
- Hox J. 2002. Multilevel analysis: techniques and applications. Mahwah, NJ: Erlbaum
- Rabe-Hesketh S, Skrondal A. 2012. Multilevel and longitudinal modeling using Stata. 3rd ed. College Station, Tex: Stata Press Publication
- Raudenbush SW, Bryk AS. 2002. Hierarchical linear models: applications and data analysis methods. 2nd ed. Thousand Oaks: Sage Publications
- Stryhn H, Sanchez J, Morley P, Booker C, Dohoo IR. 2006. Interpretation of variance parameters in multilevel Poisson regression models. Proceedings of the 11th International Symposium on Veterinary Epidemiology and Economics, 2006 Available at `http://www.sciquest.org.nz/node/64294`
- Wu S, Crespi CM, Wong WK. 2012. Comparison of methods for estimating the intraclass correlation coefficient for binary responses in cancer prevention cluster randomized trials. Contempory Clinical Trials 33: 869-880 (doi: 10.1016/j.cct.2012.05.004)

Further helpful online-ressources:

- CrossValidated (2012) *Intraclass correlation (ICC) for an interaction?*
- CrossValidated (2014) *Interpreting the random effect in a mixed-effect model*
- CrossValidated (2014) *how to partition the variance explained at group level and individual level*

See Also

`re_var`

Examples

```
library(lme4)
fit0 <- lmer(Reaction ~ 1 + (1 | Subject), sleepstudy)
icc(fit0)

fit1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
icc(fit1)

sleepstudy$mygrp <- sample(1:45, size = 180, replace = TRUE)
fit2 <- lmer(Reaction ~ Days + (1 | mygrp) + (Days | Subject), sleepstudy)
icc(fit2)

# return icc for all models at once
icc(fit0, fit1, fit2)

icc1 <- icc(fit1)
icc2 <- icc(fit2)

print(icc1, comp = "var")
print(icc2, comp = "var")
```

---

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
```

---

mn *Sum, mean and median for vectors*

Description

mn(), md() and sm() calculate the mean, median or sum of values in a vector, but have set argument na.rm to TRUE by default.

Usage

```
mn(x, na.rm = TRUE)

sm(x, na.rm = TRUE)

md(x, na.rm = TRUE)
```

Arguments

x	A vector.
na.rm	Logical, whether to remove NA values from x before computing mean, median or sum.

Value

The mean, median or sum of x.

Examples

```
data(efc)
md(efc$neg_c_7)
mn(efc$neg_c_7)
mean(efc$neg_c_7)
sm(efc$neg_c_7 > 15)
```

---

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(x, grp, distribution = "asymptotic", weight.by = NULL)
```

Arguments

x	Numeric vector or variable.
grp	Grouping variable indicating the groups that should be used for comparison.

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.
weight.by	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.

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$e17age, efc$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, proportion of the incidence in the outcome variable (base line risk).

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) 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.

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

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

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

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 = 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
or_to_rr(1.913887, .5516)
```

`overdisp` *Check overdispersion of GL(M)M's*

Description

`overdisp()` checks generalized linear (mixed) models for overdispersion, while `zero_count()` checks whether models from poisson-families are over- or underfitting zero-counts in the outcome.

Usage

```
overdisp(x, trafo = NULL)

zero_count(x, tolerance = 0.05)
```

Arguments

`x`	Fitted GLMM (`merMod`-class) or `glm` model.

trafo — A specification of the alternative, can be numeric or a (positive) function or NULL (the default). See 'Details' in dispersiontest in package **AER**. Does not apply to merMod objects.

tolerance — The tolerance for the ratio of observed and predicted zeros to considered as over- or underfitting zero-counts. A ratio between 1 +/- tolerance are considered as OK, while a ratio beyond or below this treshold would indicate over- or underfitting.

Details

For merMod- and glmmTMB-objects, overdisp() is based on the code in the GLMM FAQ, section *How can I deal with overdispersion in GLMMs?*. Note that this function only returns an *approximate* estimate of an overdispersion parameter, and is probably inaccurate for zero-inflated mixed models (fitted with glmmTMB).

For glm's, overdisp() simply wraps the dispersiontest from the **AER**-package.

Value

For overdisp(), information on the overdispersion test; for zero_count(), the amount of predicted and observed zeros in the outcome, as well as the ratio between these two values.

Note

For overdispersoion test, a p-value < .05 indicates overdispersion.

For zero_count(), a model that is underfitting zero-counts indicates a zero-inflation in the data, i.e. it is recommended to use negative binomial or zero-inflated models then.

References

Bolker B et al. (2017): GLMM FAQ.

Examples

```
library(sjmisc)
data(efc)

# response has many zero-counts, poisson models
# might be overdispersed
barplot(table(efc$tot_sc_e))

fit <- glm(tot_sc_e ~ neg_c_7 + e42dep + c160age,
           data = efc, family = poisson)
overdisp(fit)
zero_count(fit)

library(lme4)
efc$e15relat <- to_factor(efc$e15relat)
fit <- glmer(tot_sc_e ~ neg_c_7 + e42dep + c160age + (1 | e15relat),
             data = efc, family = poisson)
```

```
overdisp(fit)
zero_count(fit)
```

---

pca *Tidy summary of Principal Component Analysis*

Description

...

Usage

```
pca(x)

pca_rotate(x, nf = NULL, rotation = c("varimax", "oblimin"))
```

Arguments

x	A data frame or a prcomp object.
nf	Number of components to extract. If rotation = "varmiax" and nf = NULL, number of components is based on the Kaiser-criteria.
rotation	Rotation of the factor loadings. May be "varimax" for orthogonal rotation or "oblimin" for oblique transformation.

Details

The print()-method for pca_rotate() has a cutoff-argument, which is a scalar between 0 and 1, indicating which (absolute) values from the loadings should be blank in the output. By default, all loadings below .1 (or -.1) are not shown.

Value

A tidy data frame with either all loadings of principal components (for pca()) or a rotated loadings matrix (for pca_rotate()).

Examples

```
data(efc)
# recveive first item of COPE-index scale
start <- which(colnames(efc) == "c82cop1")
# recveive last item of COPE-index scale
end <- which(colnames(efc) == "c90cop9")

# extract principal components
pca(efc[, start:end])
```

```
# extract principal components, varimax-rotation.
# number of components based on Kaiser-criteria
pca_rotate(efc[, start:end])
```

---

phi *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

```
phi(tab)

cramer(tab)

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

Arguments

tab	A table or ftable. Tables of class xtabs and other will be coerced to ftable objects.
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.
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" or "pearson". See 'Details'.
...	Other arguments, passed down to the statistic functions chisq.test, fisher.test or cor.test.

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. 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`-aergument.

`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)

data(efc)
# 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
)
```

---

pred_accuracy *Accuracy of predictions from model fit*

Description

This function calculates the predictive accuracy of linear or logistic regression models.

Usage

```
pred_accuracy(data, fit, method = c("cv", "boot"), k = 5, n = 1000)
```

Arguments

data	A data frame.
fit	Fitted model object of class lm or glm, the latter being a logistic regression model (binary response).
method	Character string, indicating whether crossvalidation (method = "cv") or bootstrapping (method = "boot") is used to compute the accuracy values.
k	The number of folds for the kfold-crossvalidation.
n	Number of bootstraps to be generated

Details

For linar models, the accuracy is the correlation coefficient between the actual and the predicted value of the outcome. For logistic regression models, the accuracy corresponds to the AUC-value, calculated with the auc-function.

The accuracy is the mean value of multiple correlation resp. AUC-values, which are either computed with crossvalidation or nonparametric bootstrapping (see argument method). The standard error is the standard deviation of the computed correlation resp. AUC-values.

Value

A list with two values: The accuracy of the model predictions, i.e. the proportion of accurately predicted values from the model and its standard error, std.error.

See Also

cv_error

Examples

```
data(efc)
fit <- lm(neg_c_7 ~ barthtot + c161sex, data = efc)

# accuracy for linear model, with crossvalidation
pred_accuracy(efc, fit)

# accuracy for linear model, with bootstrapping
pred_accuracy(efc, fit, method = "boot", n = 100)

# accuracy for logistic regression, with crossvalidation
efc$services <- sjmisc::dicho(efc$tot_sc_e, dich.by = 0, as.num = TRUE)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep,
           data = efc, family = binomial(link = "logit"))
pred_accuracy(efc, fit)
```

---

pred_vars *Access information from model objects*

Description

Several functions to retrieve information from model objects, like variable names, link-inverse function, model frame etc.

Usage

```
pred_vars(x)

resp_var(x)

resp_val(x)

link_inverse(x)

model_frame(x, fe.only = TRUE)

var_names(x)
```

Arguments

x	A fitted model; for var_names(), x may also be a character vector.
fe.only	Logical, if TRUE (default) and x is a mixed effects model, returns the model frame for fixed effects only.

Value

For pred_vars() and resp_var(), the name(s) of the response or predictor variables from x as character vector. resp_val() returns the values from x's response vector. link_inverse() returns, if known, the inverse link function from x; else NULL for those models where the inverse link function can't be identified. model_frame() is similar to model.frame(), but should also work for model objects that don't have a S3-generic for model.frame(). var_names() returns the "cleaned" variable names, i.e. things like s() for splines or log() are removed.

Examples

```
data(efc)
fit <- lm(neg_c_7 ~ e42dep + c161sex, data = efc)

pred_vars(fit)
resp_var(fit)
resp_val(fit)

link_inverse(fit)(2.3)

# example from ?stats::glm
counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
outcome <- gl(3, 1, 9)
treatment <- gl(3, 3)
m <- glm(counts ~ outcome + treatment, family = poisson())

link_inverse(m)(.3)
# same as
exp(.3)

outcome <- as.numeric(outcome)
m <- glm(counts ~ log(outcome) + as.factor(treatment), family = poisson())
var_names(m)
```

---

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, ..., weight.by = 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'.
weight.by	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'
  )
```

---

p_value *Get p-values from regression model objects*

Description

This function returns the p-values for fitted model objects.

Usage

```
p_value(fit, p.kr = FALSE)
```

Arguments

fit	A fitted model object of class lm, glm, merMod, merModLmerTest, pggls or gls. Other classes may work as well.
p.kr	Logical, if TRUE, the computation of p-values is based on conditional F-tests with Kenward-Roger approximation for the df (see 'Details').

Details

For linear mixed models (lmerMod-objects), the computation of p-values (if p.kr = TRUE) is based on conditional F-tests with Kenward-Roger approximation for the df, using the **pbkrtest**-package. If **pbkrtest** is not available or p.kr = FALSE, or if x is a glmerMod-object, computation of p-values is based on normal-distribution assumption, treating the t-statistics as Wald z-statistics.

If p-values already have been computed (e.g. for merModLmerTest-objects from the **lmerTest**-package), these will be returned.

Value

A tibble with the model coefficients' names (term), p-values (p.value) and standard errors (std.error).

Examples

```
data(efc)
# linear model fit
fit <- lm(neg_c_7 ~ e42dep + c172code, data = efc)
p_value(fit)

# Generalized Least Squares fit
library(nlme)
fit <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
p_value(fit)

# lme4-fit
library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), data = sleepstudy)
p_value(fit, p.kr = TRUE)
```

---

r2 *Compute r-squared of (generalized) linear (mixed) models*

Description

Compute R-squared values of linear (mixed) models, or pseudo-R-squared values for generalized linear (mixed) models.

Usage

```
r2(x, n = NULL)
```

Arguments

x	Fitted model of class lm, glm, lmerMod/lme or glmerMod.
n	Optional, a lmerMod object, representing the fitted null-model (unconditional model) to x. If n is given, the pseudo-r-squared for random intercept and random slope variances are computed (*Kwok et al. 2008*) as well as the Omega squared value (*Xu 2003*). See 'Examples' and 'Details'.

Details

For linear models, the r-squared and adjusted r-squared value is returned, as provided by the summary-function.

For linear mixed models, an r-squared approximation by computing the correlation between the fitted and observed values, as suggested by *Byrnes (2008)*, is returned as well as a simplified version of the Omega-squared value (1 - (residual variance / response variance), *Xu (2003)*, *Nakagawa, Schielzeth 2013*), unless n is specified.

If n is given, for linear mixed models pseudo r-squared measures based on the variances of random intercept (tau 00, between-group-variance) and random slope (tau 11, random-slope-variance), as well as the r-squared statistics as proposed by *Snijders and Bosker 2012* and the Omega-squared value (1 - (residual variance full model / residual variance null model)) as suggested by *Xu (2003)* are returned.

For generalized linear models, Cox & Snell's and Nagelkerke's pseudo r-squared values are returned.

For generalized linear mixed models, the coefficient of determination as suggested by *Tjur (2009)* (see also cod). Note that *Tjur's D* is restricted to models with binary response.

More ways to compute coefficients of determination are shown in this great GLMM faq. Furthermore, see r.squaredGLMM or rsquared for conditional and marginal r-squared values for GLMM's.

Value

- For linear models, the r-squared and adjusted r-squared values.
- For linear mixed models, the r-squared and Omega-squared values.
- For glm objects, Cox & Snell's and Nagelkerke's pseudo r-squared values.
- For glmerMod objects, Tjur's coefficient of determination.

Note

If n is given, the Pseudo-R2 statistic is the proportion of explained variance in the random effect after adding co-variates or predictors to the model, or in short: the proportion of the explained variance in the random effect of the full (conditional) model x compared to the null (unconditional)

model n.

The Omega-squared statistics, if n is given, is 1 - the proportion of the residual variance of the full model compared to the null model's residual variance, or in short: the the proportion of the residual variation explained by the covariates.

The r-squared statistics for linear mixed models, if the unconditional model is also specified (see n), is the difference of the total variance of the null and full model divided by the total variance of the null model.

Alternative ways to assess the "goodness-of-fit" is to compare the ICC of the null model with the ICC of the full model (see icc).

References

- DRAFT r-sig-mixed-models FAQ
- Bolker B et al. (2017): GLMM FAQ.
- Byrnes, J. 2008. Re: Coefficient of determination (R^2) when using lme() (https://stat.ethz.ch/pipermail/r-sig-mixed-models/2008q2/000713.html)
- Kwok OM, Underhill AT, Berry JW, Luo W, Elliott TR, Yoon M. 2008. Analyzing Longitudinal Data with Multilevel Models: An Example with Individuals Living with Lower Extremity Intra-Articular Fractures. Rehabilitation Psychology 53(3): 370–86. doi: 10.1037/a0012765
- Nakagawa S, Schielzeth H. 2013. A general and simple method for obtaining R2 from generalized linear mixed-effects models. Methods in Ecology and Evolution, 4(2):133–142. doi: 10.1111/j.2041210x.2012.00261.x
- Rabe-Hesketh S, Skrondal A. 2012. Multilevel and longitudinal modeling using Stata. 3rd ed. College Station, Tex: Stata Press Publication
- Raudenbush SW, Bryk AS. 2002. Hierarchical linear models: applications and data analysis methods. 2nd ed. Thousand Oaks: Sage Publications
- Snijders TAB, Bosker RJ. 2012. Multilevel analysis: an introduction to basic and advanced multilevel modeling. 2nd ed. Los Angeles: Sage
- Xu, R. 2003. Measuring explained variation in linear mixed effects models. Statist. Med. 22:3527-3541. doi: 10.1002/sim.1572
- Tjur T. 2009. Coefficients of determination in logistic regression models - a new proposal: The coefficient of discrimination. The American Statistician, 63(4): 366-372

See Also

rmse for more methods to assess model quality.

Examples

```
library(sjmisc)
library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
r2(fit)
```

```
data(efc)
fit <- lm(barthtot ~ c160age + c12hour, data = efc)
r2(fit)

# Pseudo-R-squared values
efc$services <- ifelse(efc$tot_sc_e > 0, 1, 0)
fit <- glm(services ~ neg_c_7 + c161sex + e42dep,
           data = efc, family = binomial(link = "logit"))
r2(fit)

# Pseudo-R-squared values for random effect variances
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
fit.null <- lmer(Reaction ~ 1 + (Days | Subject), sleepstudy)
r2(fit, fit.null)
```

`reliab_test` *Check internal consistency of a test or questionnaire*

Description

These function compute various measures of internal consistencies for tests or item-scales of questionnaires.

Usage

```
reliab_test(x, scale.items = FALSE, digits = 3, out = c("txt", "viewer",
  "browser"))

split_half(x, digits = 3)

cronb(x)

mic(x, cor.method = c("pearson", "spearman", "kendall"))
```

Arguments

`x`	Depending on the function, `x` may be a `matrix` as returned by the `cor`-function, or a data frame with items (e.g. from a test or questionnaire).
`scale.items`	Logical, if `TRUE`, the data frame's vectors will be scaled. Recommended, when the variables have different measures / scales.
`digits`	Amount of digits for returned 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"`).
`cor.method`	Correlation computation method. May be one of `"spearman"` (default), `"pearson"` or `"kendall"`. You may use initial letter only.

Details

`reliab_test()` This function calculates the item discriminations (corrected item-total correlations for each item of `x` with the remaining items) and the Cronbach's alpha for each item, if it was deleted from the scale.

`split_half()` This function calculates the split-half reliability for items in the data frame `x`, including the Spearman-Brown adjustment. Splitting is done by selecting odd versus even columns in `x`.

`cronb()` The Cronbach's Alpha value for `x`.

`mic()` This function calculates a mean inter-item-correlation, i.e. a correlation matrix of `x` will be computed (unless `x` is already a matrix as returned by the `cor`-function) and the mean of the sum of all item's correlation values is returned. Requires either a data frame or a computed `cor`-object.

Value

`reliab_test()` A data frame with the corrected item-total correlations (item discrimination, column `item.discr`) and Cronbach's alpha (if item deleted, column `alpha.if.deleted`) for each item of the scale, or `NULL` if data frame had too less columns.

`split_half()` A list with two values: the split-half reliability `splithalf` and the Spearman-Brown corrected split-half reliability `spearmanbrown`.

`cronb()` The Cronbach's Alpha value for `x`.

`mic()` The mean inter-item-correlation value for `x`.

Note

`reliab_test()` is similar to a basic reliability test in SPSS. The correlations in the Item-Total-Statistic are a computed correlation of each item against the sum of the remaining items (which are thus treated as one item).

For `split_half()` and `cronb()`, a value closer to 1 indicates greater internal consistency.

For the mean inter-item-correlation: "Ideally, the average inter-item correlation for a set of items should be between .20 and .40, suggesting that while the items are reasonably homogenous, they do contain sufficiently unique variance so as to not be isomorphic with each other. When values are lower than .20, then the items may not be representative of the same content domain. If values are higher than .40, the items may be only capturing a small bandwidth of the construct." *(Piedmont 2014)*

References

Spearman C. 1910. Correlation calculated from faulty data. British Journal of Psychology (3): 271–295. doi: 10.1111/j.20448295.1910.tb00206.x

Brown W. 1910. Some experimental results in the correlation of mental abilities. British Journal of Psychology (3): 296–322. doi: 10.1111/j.20448295.1910.tb00207.x

Piedmont RL. 2014. Inter-item Correlations. In: Michalos AC (eds) Encyclopedia of Quality of

Life and Well-Being Research. Dordrecht: Springer, 3303-3304. doi: 10.1007/978940070753-5_1493

Examples

```
library(sjlabelled)
# Data from the EUROFAMCARE sample dataset
data(efc)

# retrieve variable and value labels
varlabs <- get_label(efc)

# recveive first item of COPE-index scale
start <- which(colnames(efc) == "c82cop1")
# recveive last item of COPE-index scale
end <- which(colnames(efc) == "c90cop9")

# create data frame with COPE-index scale
x <- efc[, c(start:end)]
colnames(x) <- varlabs[c(start:end)]

# reliability tests
reliab_test(x)

# split-half-reliability
split_half(x)

# cronbach's alpha
cronb(x)

# mean inter-item-correlation
mic(x)

## Not run:
library(sjPlot)
sjt.df(reliab_test(x), describe = FALSE, show.cmmn.row = TRUE,
       string.cmmn = sprintf("Cronbach's &alpha;=%.2f", cronb(x)))

# Compute PCA on Cope-Index, and perform a
# reliability check on each extracted factor.
factors <- sjt.pca(x)$factor.index
findex <- sort(unique(factors))
library(sjPlot)
for (i in seq_len(length(findex))) {
 rel.df <- subset(x, select = which(factors == findex[i]))
 if (ncol(rel.df) >= 3) {
   sjt.df(reliab_test(rel.df), describe = FALSE, show.cmmn.row = TRUE,
          use.viewer = FALSE, title = "Item-Total-Statistic",
          string.cmmn = sprintf("Scale's overall Cronbach's &alpha;=%.2f",
                                cronb(rel.df)))
   }
 }
## End(Not run)
```

re_var *Random effect variances*

Description

These functions extracts random effect variances as well as random-intercept-slope-correlation of mixed effects models. Currently, merMod, glmmTMB, stanreg and brmsfit objects are supported.

Usage

```
re_var(x)

get_re_var(x, comp = c("tau.00", "tau.01", "tau.11", "rho.01", "sigma_2"))
```

Arguments

x	Fitted mixed effects model (of class merMod, glmmTMB, stanreg or brmsfit). get_re_var() also accepts an object of class icc.lme4, as returned by the icc function.
comp	Name of the variance component to be returned. See 'Details'.

Details

The random effect variances indicate the between- and within-group variances as well as random-slope variance and random-slope-intercept correlation. Use following values for comp to get the particular variance component:

"sigma_2" Within-group (residual) variance

"tau.00" Between-group-variance (variation between individual intercepts and average intercept)

"tau.11" Random-slope-variance (variation between individual slopes and average slope)

"tau.01" Random-Intercept-Slope-covariance

"rho.01" Random-Intercept-Slope-correlation

The within-group-variance is affected by factors at level one, i.e. by the lower-level direct effects. Level two factors (i.e. cross-level direct effects) affect the between-group-variance. Cross-level interaction effects are group-level factors that explain the variance in random slopes (Aguinis et al. 2013).

Value

get_re_var() returns the value of the requested variance component, re_var() returns all random effects variances.

References

Aguinis H, Gottfredson RK, Culpepper SA. 2013. Best-Practice Recommendations for Estimating Cross-Level Interaction Effects Using Multilevel Modeling. Journal of Management 39(6): 1490–1528 (doi: 10.1177/0149206313478188)

See Also

icc

Examples

```
library(lme4)
fit1 <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)

# all random effect variance components
re_var(fit1)

# just the rand. slope-intercept covariance
get_re_var(fit1, "tau.01")

sleepstudy$mygrp <- sample(1:45, size = 180, replace = TRUE)
fit2 <- lmer(Reaction ~ Days + (1 | mygrp) + (Days | Subject), sleepstudy)
re_var(fit2)
```

---

rmse *Compute model quality*

Description

Compute root mean squared error, residual standard error or mean square error of fitted linear (mixed effects) models.

Usage

```
rmse(fit, normalized = FALSE)

rse(fit)

mse(fit)
```

Arguments

fit	Fitted linear model of class lm, merMod (**lme4**) or lme (**nlme**).
normalized	Logical, use TRUE if normalized rmse should be returned.

Note

Root Mean Square Error The RMSE is the square root of the variance of the residuals and indicates the absolute fit of the model to the data (difference between observed data to model's predicted values). "RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit. RMSE is a good measure of how accurately the model predicts the response, and is the most important criterion for fit if the main purpose of the model is prediction." *(Grace-Martin K: Assessing the Fit of Regression Models)*

The normalized RMSE is the proportion of the RMSE related to the range of the response variable. Hence, lower values indicate less residual variance.

Residual Standard Error The residual standard error is the square root of the residual sum of squares divided by the residual degrees of freedom.

Mean Square Error The mean square error is the mean of the sum of squared residuals, i.e. it measures the average of the squares of the errors. Lower values (closer to zero) indicate better fit.

References

Grace-Martin K: Assessing the Fit of Regression Models

See Also

r2 for R-squared or pseude-R-squared values, and cv for the coefficient of variation.

Examples

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

library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
rmse(fit)
mse(fit)

# normalized RMSE
library(nlme)
fit <- lme(distance ~ age, data = Orthodont)
rmse(fit, normalized = TRUE)
```

robust *Robust standard errors for regression models*

Description

`robust()` computes robust standard error for regression models. This method wraps the `coeftest`-function with robust covariance matrix estimators based on the `vcovHC`-function, and returns the result 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 = c("HC3", "const", "HC", "HC0", "HC1", "HC2", "HC4", "HC4m",
  "HC5"), conf.int = FALSE, exponentiate = FALSE)

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

Arguments

`x`	A fitted model of any class that is supported by the `coeftest()`-function. For `svy()`, `x` must be `lm` object, fitted with weights.
`vcov`	Character vector, specifying the estimation type for the heteroskedasticity-consistent covariance matrix estimation (see `vcovHC` for details).
`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 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 = "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)
```

---

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.

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)
```

---

se *Standard Error for variables or coefficients*

Description

Compute standard error for a variable, for all variables of a data frame, for joint random and fixed effects coefficients of (non-/linear) mixed models, the adjusted standard errors for generalized linear (mixed) models, or for intraclass correlation coefficients (ICC).

Usage

```
se(x, nsim = 100, type = c("fe", "re"))
```

Arguments

x	(Numeric) vector, a data frame, an lm or glm-object, a merMod-object as returned by the functions from the **lme4**-package, an ICC object (as obtained by the icc-function) or a list with estimate and p-value. For the latter case, the list must contain elements named estimate and p.value (see 'Examples' and 'Details').
nsim	Numeric, the number of simulations for calculating the standard error for intra-class correlation coefficients, as obtained by the icc-function.
type	Type of standard errors for generalized linear mixed models. type = "fe" returns the standard errors for fixed effects, based on the delta-method-approximation. type = "re" returns the standard errors for joint random and fixed effects, which are on the scale of the link function. See 'Details'.

Details

For linear mixed models, and generalized linear mixed models **with** type = "re", this function computes the standard errors for joint (sums of) random and fixed effects coefficients (unlike se.coef, which returns the standard error for fixed and random effects separately). Hence, se() returns the appropriate standard errors for coef.merMod.

For generalized linear models or generalized linear mixed models, approximated standard errors, using the delta method for transformed regression parameters are returned (Oehlert 1992). For generalized linear mixed models, by default, the standard errors refer to the fixed effects only. Use type = "re" to compute standard errors for joint random and fixed effects coefficients. However, computation for the latter *is not* based on the delta method, so standard errors from type = "re" are on the scale of the link-function (and not back transformed).

The standard error for the icc is based on bootstrapping, thus, the nsim-argument is required. See 'Examples'.

se() also returns the standard error of an estimate (regression coefficient) and p-value, assuming a normal distribution to compute the z-score from the p-value (formula in short: b / qnorm(p / 2)). See 'Examples'.

Value

The standard error of x.

Note

Computation of standard errors for coefficients of mixed models is based on this code. Standard errors for generalized linear (mixed) models, if type = "re", are approximations based on the delta method (Oehlert 1992).

A remark on standard errors: "Standard error represents variation in the point estimate, but confidence interval has usual Bayesian interpretation only with flat prior." (Gelman 2017)

References

Oehlert GW. 1992. A note on the delta method. American Statistician 46(1).

Gelman A 2017. How to interpret confidence intervals? http://andrewgelman.com/2017/03/04/interpret-confidence-intervals/

Examples

```
# compute standard error for vector
se(rnorm(n = 100, mean = 3))

# compute standard error for each variable in a data frame
data(efc)
se(efc[, 1:3])

# compute standard error for merMod-coefficients
library(lme4)
fit <- lmer(Reaction ~ Days + (Days | Subject), sleepstudy)
se(fit)

# compute odds-ratio adjusted standard errors, based on delta method
# with first-order Taylor approximation.
data(efc)
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"))
se(fit)

# compute odds-ratio adjusted standard errors for generalized
# linear mixed model, also based on delta method
library(lme4)
library(sjmisc)
# create binary response
sleepstudy$Reaction.dicho <- dicho(sleepstudy$Reaction, dich.by = "median")
fit <- glmer(Reaction.dicho ~ Days + (Days | Subject),
             data = sleepstudy, family = binomial("logit"))
se(fit)

# compute standard error from regression coefficient and p-value
se(list(estimate = .3, p.value = .002))

## Not run:
# compute standard error of ICC for the linear mixed model
icc(fit)
se(icc(fit))

# the standard error for the ICC can be computed manually in this way,
# taking the fitted model example from above
library(tidyverse)
dummy <- sleepstudy %>%
  # generate 100 bootstrap replicates of dataset
  bootstrap(100) %>%
```

```
  # run mixed effects regression on each bootstrap replicate
  # and compute ICC for each "bootstrapped" regression
  mutate(
    models = map(strap, ~lmer(Reaction ~ Days + (Days | Subject), data = .x)),
    icc = map_dbl(models, ~icc(.x))
  )

# now compute SE and p-values for the bootstrapped ICC, values
# may differ from above example due to random seed
boot_se(dummy, icc)
boot_p(dummy, icc)
## End(Not run)
```

---

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)
```

`smpsize_lmm` *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

```
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 `deff`). 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 & 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 & 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.
smpsize_lmm(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.
smpsize_lmm(eff.size = .2, df.n = 5, k = 20, power = .9)
```

---

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, type = "std", ci.lvl = 0.95)
```

Arguments

`fit`	Fitted linear (mixed) model of class `lm` or `merMod` (**lme4** package).
`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.
`ci.lvl`	Numeric, the level of the confidence intervals.

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() and predict(). However, these functions have some limitations: family() simply returns the family-object from the underlying glm.nb-model. The predict()-methods simply recomputes the 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.

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
```

---

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, probs = 0.89, typical = "median", trans = NULL,
  type = c("fixed", "random", "all"), digits = 3)
```

Arguments

x	A stanreg, stanfit or brmsfit object.
probs	Vector of scalars between 0 and 1, indicating the mass within the credible interval that is to be estimated. See hdi.
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 estimate and HDI-values. The values for standard errors are *not* transformed!
type	For mixed effects models, specify the type of effects that should be returned. type = "fixed" returns fixed effects only, type = "random" the random effects and type = "all" returns both fixed and random effects.
digits	Amount of digits to round numerical values in the output.

Details

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 `typical`), the standard error (which are actually *median absolute deviations*), the HDI, the ratio of effective numbers of samples, *n_eff*, (i.e. effective number of samples divided by total number of samples) and Rhat statistics.

The ratio of effective number of samples ranges from 0 to 1, and should be close to 1. The closer this ratio comes to zero means that the chains may be inefficient, but possibly still okay.

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.

Computation for HDI is based on the code from Kruschke 2015, pp. 727f.

Value

A tidy data frame, summarizing `x`, with consistent column names. To distinguish multiple HDI values, column names for the HDI get a suffix when `probs` 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 & Hall/CRC, 2013

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

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

See Also

`hdi`

Examples

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

typical_value *Return the typical value of a vector*

Description

This function returns the "typical" value of a variable.

Usage

```
typical_value(x, fun = "mean", weight.by = NULL, ...)
```

Arguments

x	A variable.
fun	Character vector, naming the function to be applied to x. Currently, "mean", "weighted.mean", "median" and "mode" are supported, which call the corresponding R functions (except "mode", which calls an internal function to compute the most common value). "zero" simply returns 0. **Note:** By default, if x is a factor, only fun = "mode" is applicable; for all other functions (including the default, "mean") the reference level of x is returned. For character vectors, only the mode is returned. You can use a named vector to apply other different functions to numeric and categorical x, where factors are first converted to numeric vectors, e.g. fun = c(numeric = "median", factor = "mean"). See 'Examples'.
weight.by	Vector of weights that will be applied to weight all cases. Must be a vector of same length as the input vector. Default is NULL, so no weights are used.
...	Further arguments, passed down to fun.

Details

By default, for numeric variables, typical_value() returns the mean value of x (unless changed with the fun-argument).

For factors, the reference level is returned or the most common value (if fun = "mode"), unless fun is a named vector. If fun is a named vector, specify the function for numeric and categorical variables as element names, e.g. fun = c(numeric = "median", factor = "mean"). In this case, factors are converted to numeric values (using to_value) and the related function is applied. You may abbreviate the names fun = c(n = "median", f = "mean"). See also 'Examples'.

For character vectors the most common value (mode) is returned.

Value

The "typical" value of x.

Examples

```
data(iris)
typical_value(iris$Sepal.Length)

library(purrr)
map(iris, ~ typical_value(.x))

# example from ?stats::weighted.mean
wt <- c(5,  5,  4,  1) / 15
x <- c(3.7, 3.3, 3.5, 2.8)

typical_value(x, "weighted.mean")
typical_value(x, "weighted.mean", weight.by = wt)

# for factors, return either reference level or mode value
set.seed(123)
x <- sample(iris$Species, size = 30, replace = TRUE)
typical_value(x)
typical_value(x, fun = "mode")

# for factors, use a named vector to apply other functions than "mode"
map(iris, ~ typical_value(.x, fun = c(n = "median", f = "mean")))
```

---

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))
```

---

wtd_sd *Weighted statistics for variables*

Description

wtd_sd() and wtd_se() compute weighted standard deviation or standard error 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).

Usage

```
wtd_sd(x, weights = NULL)

wtd_se(x, weights = NULL)

svy_md(x, design)
```

Arguments

x	(Numeric) vector or a data frame. For svy_md(), the bare (unquoted) variable name, or a character vector with the variable name.
weights	Numeric vector of weights.
design	An object of class svydesign, providing a specification of the survey design.

Value

The weighted standard deviation or standard error of x, or for each variable if x is a data frame.

Examples

```
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)))


# 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)
```

Index