

Package ‘lsr’

July 11, 2026

Title Companion to ``Learning Statistics with R"

Version 1.0.0

Description A collection of tools intended to make introductory statistics easier to teach, including wrappers for common hypothesis tests and basic data manipulation. Accompanies the textbook ``Learning Statistics with R: A Tutorial for Psychology Students and Other Beginners" by Navarro.

License MIT + file LICENSE

URL <https://github.com/djnavarro/lsr>, <https://lsr.djnavarro.net/>

BugReports <https://github.com/djnavarro/lsr/issues>

Encoding UTF-8

Language en-GB

Imports graphics, grDevices, methods, stats

Suggests knitr, rmarkdown, testthat (>= 3.0.0), tibble, withr

Config/testthat/edition 3

Config/Needs/website djnavarro/waeponwifestre

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Danielle Navarro [aut, cre] (ORCID:
<<https://orcid.org/0000-0001-7648-6578>>)

Maintainer Danielle Navarro <djnavarro@protonmail.com>

Repository CRAN

Date/Publication 2026-07-11 18:30:11 UTC

Contents

aad	2
associationTest	3
bars	4

ciMean	7
cohensD	8
copy	9
correlate	10
cramersV	12
etaSquared	13
expandFactors	14
goodnessOfFitTest	15
importList	16
independentSamplesTTest	18
longToWide	19
mode	20
oneSampleTTest	21
pairedSamplesTTest	23
permuteLevels	25
posthocPairwiseT	26
print.assocTest	27
print.correlate	27
print.gofTest	28
print.TTest	28
print.whoList	29
quantileCut	29
rmAll	30
sortFrame	31
standardCoefs	32
tFrame	33
unlibrary	34
who	35
wideToLong	36

Index 38

aad	<i>Mean absolute deviation</i>
-----	--------------------------------

Description

Calculates the mean absolute deviation from the sample mean.

Usage

```
aad(x, na.rm = FALSE)
```

Arguments

x	A numeric vector containing the observations.
na.rm	Set to TRUE to remove missing values before computing the deviation. Defaults to FALSE.

Details

Computes the average of the absolute differences between each observation and the sample mean of x , i.e. `mean(abs(x - mean(x)))`.

Value

A single number giving the mean absolute deviation.

See Also

[mean](#), [sd](#), [var](#)

Examples

```
x <- c(1, 3, 6)
aad(x)

# missing values
x <- c(1, 3, NA, 6)
aad(x) # returns NA
aad(x, na.rm = TRUE) # ignores the missing value
```

associationTest	<i>Chi-square test of association / independence</i>
-----------------	--

Description

Runs a chi-square test to check whether two categorical variables are independent of one another.

Usage

```
associationTest(formula, data = NULL)
```

Arguments

formula	A one-sided formula of the form <code>~var1 + var2</code> , specifying the two variables to be tested. Both variables must be factors.
data	An optional data frame containing the variables named in <code>formula</code> . If omitted, the variables are looked up in the workspace.

Details

The test checks whether two categorical variables are statistically independent. Both variables must be factors, and the formula must be one-sided with exactly two variables, e.g. `~gender + answer`.

Missing values are removed before the test is run, and a warning is issued if any cases are dropped. When both variables have only two levels, Yates' continuity correction is applied automatically to the chi-squared statistic (though not to the Cramer's V effect size).

If either variable has unused factor levels (levels with zero observed cases), a warning is issued. Those levels are included in the contingency table with zero observed cases, which may give misleading results. Call [droplevels](#) on the data first if this is not intended.

Value

Prints a summary of the test showing the variable names, null and alternative hypotheses, observed and expected frequency tables, test results (chi-square statistic, degrees of freedom, p-value), and Cramer's V as a measure of effect size. The underlying results are also returned as a list, so the output can be assigned to a variable and inspected if needed.

See Also

[chisq.test](#), [goodnessOfFitTest](#), [cramersV](#)

Examples

```
df <- data.frame(
  gender = factor(c("male", "male", "male", "male", "female", "female", "female")),
  answer = factor(c("heads", "heads", "heads", "heads", "tails", "tails", "heads"))
)

associationTest(~ gender + answer, df)
```

bars

Grouped bar plots with error bars

Description

Creates a bar plot showing group means with error bars showing confidence intervals, broken down by one or two grouping factors.

Usage

```
bars(
  formula,
  data = NULL,
  heightFun = mean,
  errorFun = ciMean,
  yLabel = NULL,
```

```

xLabels = NULL,
main = "",
ylim = NULL,
barFillColour = NULL,
barLineWidth = 2,
barLineColour = "black",
barSpaceSmall = 0.2,
barSpaceBig = 1,
legendLabels = NULL,
legendDownShift = 0,
legendLeftShift = 0,
errorBarLineWidth = 1,
errorBarLineColour = "grey40",
errorBarWhiskerWidth = 0.2
)

```

Arguments

formula	A two-sided formula of the form <code>response ~ group1</code> or <code>response ~ group1 + group2</code> . The response variable must be numeric; grouping variables must be factors.
data	An optional data frame containing the variables named in <code>formula</code> . If omitted, the variables are looked up in the workspace.
heightFun	The function used to calculate bar heights. Defaults to <code>mean</code> . Must return a single number.
errorFun	The function used to calculate error bar positions. Defaults to <code>ciMean</code> . Must return two numbers (lower and upper bounds). Set to <code>FALSE</code> to suppress error bars.
yLabel	The y-axis label. Defaults to the name of the response variable.
xLabels	Labels for the x-axis tick marks. Defaults to the levels of <code>group1</code> .
main	The plot title. Defaults to no title.
ylim	A numeric vector of length 2 giving the y-axis limits. The lower bound defaults to 0; the upper bound is estimated automatically.
barFillColour	A vector of colours used to fill the bars. Defaults to a pastel rainbow palette.
barLineWidth	The width of the bar border lines. Defaults to 2.
barLineColour	The colour of the bar border lines. Defaults to "black".
barSpaceSmall	The gap between bars within a cluster, as a proportion of bar width. Defaults to 0.2.
barSpaceBig	The gap separating clusters of bars, as a proportion of bar width. Defaults to 1.
legendLabels	Labels for the legend entries. Defaults to the levels of <code>group2</code> . Set to <code>FALSE</code> to suppress the legend. No legend is drawn when only one grouping variable is specified.
legendDownShift	How far below the top of the plot to place the legend, as a proportion of plot height. Defaults to 0.

legendLeftShift

How far from the right edge to place the legend, as a proportion of plot width. Defaults to 0.

errorBarLineWidth

The line width for the error bars. Defaults to 1.

errorBarLineColour

The colour of the error bars. Defaults to "grey40".

errorBarWhiskerWidth

The width of the error bar whiskers, as a proportion of bar width. Defaults to 0.2.

Details

Plots group means (or the output of heightFun) with error bars (or the output of errorFun) for one or two grouping factors. When two grouping factors are given, group1 determines the primary x-axis grouping and group2 determines the sub-grouping shown as clusters of bars with a legend.

Missing values are removed with a warning. At least 2 complete cases are required per group.

Value

Invisibly returns a data frame containing the factor levels, group summary values, and error bar bounds. This function is primarily used for its side effect of drawing the plot.

See Also

[barplot](#), [ciMean](#)

Examples

```
# one grouping factor
df <- data.frame(
  outcome = c(3, 4, 5, 2, 4, 6, 5, 7, 8),
  group = factor(c("a", "a", "a", "b", "b", "b", "c", "c", "c"))
)
bars(outcome ~ group, data = df)

# two grouping factors
df2 <- data.frame(
  outcome = c(3, 4, 5, 2, 4, 6, 5, 7, 8, 4, 3, 6),
  group1 = factor(rep(c("a", "b"), each = 6)),
  group2 = factor(rep(c("x", "y", "z"), times = 4))
)
bars(outcome ~ group1 + group2, data = df2)
```

ciMean	<i>Confidence interval around the mean</i>
--------	--

Description

Calculates a confidence interval for the mean of a numeric variable (or each numeric variable in a data frame or matrix).

Usage

```
ciMean(x, conf = 0.95, na.rm = FALSE)
```

Arguments

x	A numeric vector, matrix, or data frame.
conf	The confidence level. Defaults to 0.95 for a 95% interval.
na.rm	Set to TRUE to remove missing values before computing the interval. Defaults to FALSE.

Details

Calculates a confidence interval for the mean under the standard assumption that the data are normally distributed. When x is a matrix or data frame, a separate interval is computed for each column. Non-numeric columns in a data frame produce NA rows in the output.

Value

A matrix with one row per variable and two columns giving the lower and upper bounds of the confidence interval. Column names reflect the confidence level (e.g., "2.5%" and "97.5%" for a 95% interval).

See Also

[t.test](#), [mean](#)

Examples

```
x <- c(1, 3, 6)
ciMean(x) # 95% confidence interval
ciMean(x, conf = 0.80) # 80% confidence interval

# for comparison: equivalent result via lm
confint(lm(x ~ 1))

# missing values
x <- c(1, 3, NA, 6)
ciMean(x, na.rm = TRUE)
```

cohensD	<i>Cohen's d</i>
---------	------------------

Description

Calculates the Cohen's d measure of effect size.

Usage

```
cohensD(
  x = NULL,
  y = NULL,
  data = NULL,
  method = "pooled",
  mu = 0,
  formula = NULL
)
```

Arguments

x	A numeric vector of data for group 1, or a formula of the form outcome ~ group (in which case data can be used to supply a data frame).
y	A numeric vector of data for group 2. Omit for a one-sample calculation.
data	An optional data frame containing the variables in x when x is a formula.
method	Which version of Cohen's d to calculate. Options are "pooled" (default), "x.sd", "y.sd", "corrected", "raw", "paired", and "unequal". See Details.
mu	The null value for a one-sample calculation. Almost always 0 (the default).
formula	A formula of the form outcome ~ group. This is an alternative way to supply the formula instead of using x.

Details

The function can be used in two main ways. For two separate numeric vectors, call `cohensD(x = group1, y = group2)`. For data in a data frame with a grouping variable, use a formula: `cohensD(outcome ~ group, data = mydata)`.

The method argument controls how the standard deviation is estimated:

"pooled" Pooled SD from both groups (matches Student's t-test). This is the default.

"corrected" Bias-corrected version of "pooled", multiplied by $(N-3)/(N-2.25)$.

"raw" Like "pooled" but divides by N rather than N-2.

"x.sd" SD of the first group only.

"y.sd" SD of the second group only.

"unequal" Square root of the average of the two group variances (matches Welch's t-test).

"paired" SD of the within-person differences (matches a paired-samples t-test).

For a one-sample calculation, supply only x (and optionally mu). The result is $\text{abs}(\text{mean}(x) - \text{mu}) / \text{sd}(x)$.

Value

A single positive number: the magnitude of the effect size d . The sign of the mean difference is dropped, so the value is always zero or greater.

References

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

Examples

```
# two independent groups supplied as separate vectors
gradesA <- c(55, 65, 65, 68, 70) # 5 students with teacher A
gradesB <- c(56, 60, 62, 66) # 4 students with teacher B
cohensD(gradesA, gradesB)

# the same comparison using a formula and a data frame
grade <- c(55, 65, 65, 68, 70, 56, 60, 62, 66)
teacher <- c("A", "A", "A", "A", "A", "B", "B", "B", "B")
cohensD(grade ~ teacher)

# paired samples: use method = "paired" (SD of within-person differences)
pre <- c(100, 122, 97, 25, 274)
post <- c(104, 125, 99, 29, 277)
cohensD(pre, post, method = "paired")

# equivalent one-sample calculation on the difference scores
cohensD(post - pre)

# formula interface with a data frame
exams <- data.frame(grade, teacher)
cohensD(grade ~ teacher, data = exams)
```

copy

Copy a vector into a matrix

Description

Creates a matrix by stacking multiple copies of a vector as rows (`rowCopy`) or as columns (`colCopy`).

Usage

```
colCopy(x, times, dimnames = NULL)
```

```
rowCopy(x, times, dimnames = NULL)
```

Arguments

<code>x</code>	A vector to be copied.
<code>times</code>	The number of copies to stack together.
<code>dimnames</code>	An optional list specifying row and column names for the output matrix.

Details

These functions are shortcuts for building a matrix where every row (or every column) is the same vector. They are equivalent to calling `matrix` with appropriate `byrow` and dimension arguments.

Value

For `rowCopy`, a matrix with `times` rows and `length(x)` columns, where each row is `x`. For `colCopy`, a matrix with `length(x)` rows and `times` columns, where each column is `x`.

See Also

`matrix`, `rbind`, `cbind`

Examples

```
x <- c(3, 1, 4, 1, 5)

# stack x as rows
rowCopy(x, 4)

# stack x as columns
colCopy(x, 4)

# with custom dimension names
dnames <- list(rows = c("r1", "r2", "r3"), cols = c("c1", "c2", "c3", "c4", "c5"))
rowCopy(x, 3, dnames)
```

`correlate`

Correlation matrices

Description

Computes a correlation matrix, optionally with hypothesis tests and corrections for multiple comparisons.

Usage

```
correlate(
  x,
  y = NULL,
  test = FALSE,
  corr.method = "pearson",
  p.adjust.method = "holm"
)
```

Arguments

<code>x</code>	A numeric vector, matrix, or data frame containing the variables to be correlated.
<code>y</code>	An optional second matrix or data frame. If provided, the variables in <code>x</code> are correlated with the variables in <code>y</code> rather than with each other.
<code>test</code>	Set to TRUE to display p-values and sample sizes alongside the correlations. Defaults to FALSE.
<code>corr.method</code>	The type of correlation to compute: "pearson" (the default), "spearman", or "kendall".
<code>p.adjust.method</code>	The method used to correct p-values for multiple comparisons. Defaults to "holm". All methods supported by p.adjust are accepted.

Details

Calculates a correlation matrix between all pairs of numeric variables. If only `x` is supplied, all pairwise correlations among the variables in `x` are computed. If both `x` and `y` are supplied, variables in `x` are correlated with variables in `y`.

Non-numeric variables (e.g., factors) are silently ignored: they appear in the output with NA in place of correlation values. This makes it convenient to pass an entire data frame without first removing categorical columns.

When `test = TRUE`, hypothesis tests are run for every pair of numeric variables. To reduce the risk of false positives from testing many pairs at once, p-values are adjusted using the Holm method by default. See [p.adjust](#) for other available methods.

Missing data are handled using pairwise complete cases, so sample sizes may differ across pairs of variables. If a particular pair of variables has too few complete observations for [cor.test](#) to run, the corresponding cell in the correlation matrix is left as NA rather than causing the whole call to fail.

Value

Prints the correlation matrix. If `test = TRUE`, also prints a matrix of adjusted p-values and a matrix of sample sizes. The results are also returned as a list with four elements: `correlation` (the correlation matrix), `p.value` (the matrix of p-values), `sample.size` (the matrix of sample sizes), and `args` (a record of the options used). The list can be assigned to a variable and inspected if needed.

See Also

[cor](#), [cor.test](#), [p.adjust](#)

Examples

```
# data frame with factors and missing values
data <- data.frame(
  anxiety = c(1.31, 2.72, 3.18, 4.21, 5.55, NA),
  stress = c(2.01, 3.45, 1.99, 3.25, 4.27, 6.80),
  depression = c(2.51, 1.77, 3.34, 5.83, 9.01, 7.74),
```

```

happiness = c(4.02, 3.66, 5.23, 6.37, 7.83, 1.18),
gender = factor(c("male", "female", "female", "male", "female", "female")),
ssri = factor(c("no", "no", "no", NA, "yes", "yes"))
)

# Pearson correlation matrix (the default)
correlate(data)

# Spearman correlations
correlate(data, corr.method = "spearman")

# correlate two subsets of variables with each other
nervous <- data[, c("anxiety", "stress")]
happy <- data[, c("happiness", "depression")]
correlate(nervous, happy)

# include Holm-corrected p-values and sample sizes
correlate(data, test = TRUE)

```

cramersV

*Cramer's V***Description**

Calculates Cramer's V, a measure of the strength of association for chi-square tests.

Usage

```
cramersV(...)
```

Arguments

... Arguments passed to `chisq.test`, in the same format accepted by that function. The correct argument is always set to FALSE internally and cannot be overridden.

Details

Cramer's V summarises the strength of association from a chi-square test. It is appropriate for both tests of association (two categorical variables) and goodness of fit tests (one variable versus hypothesised probabilities). Values range from 0 (no association) to 1 (perfect association).

Yates' continuity correction is never applied, regardless of the table dimensions. This is intentional: applying the correction reduces the chi-squared statistic, which causes V to fall below 1 even for perfectly associated 2x2 tables — inconsistent with its definition as an effect size on the [0, 1] scale.

Value

A single number giving the value of Cramer's V.

See Also

[chisq.test](#), [associationTest](#), [goodnessOfFitTest](#)

Examples

```
# frequency table for two groups, each choosing from three options
condition1 <- c(30, 20, 50)
condition2 <- c(35, 30, 35)
X <- cbind(condition1, condition2)
rownames(X) <- c("choice1", "choice2", "choice3")

# chi-square test of association
chisq.test(X)

# effect size estimate
cramersV(X)
```

etaSquared

Effect size for ANOVAs

Description

Calculates eta-squared and partial eta-squared effect sizes for an analysis of variance.

Usage

```
etaSquared(x, type = 2, anova = FALSE)
```

Arguments

x	An aov object, as returned by aov .
type	Which type of sums of squares to use: 1 for Type I, 2 for Type II (the default), or 3 for Type III. Type II is recommended for most unbalanced designs.
anova	Set to TRUE to include the full ANOVA table alongside the effect sizes. Defaults to FALSE.

Details

Calculates eta-squared and partial eta-squared, two commonly used measures of effect size in analysis of variance. The input x should be an ANOVA fitted with [aov](#).

For unbalanced designs, Type II sums of squares (type = 2) are recommended and are the default, consistent with the Anova function in the **car** package. Type I (type = 1) matches the output of [anova](#) but tests hypotheses that are often not of interest in unbalanced designs. Type III (type = 3) is also available.

Value

A matrix with one row per term in the ANOVA model and columns for eta-squared (`eta.sq`) and partial eta-squared (`eta.sq.part`). If `anova = TRUE`, additional columns show the sums of squares, mean squares, degrees of freedom, F-statistics, and p-values.

See Also

[aov](#), [summary.aov](#)

Examples

```
outcome <- c(1.4, 2.1, 3.0, 2.1, 3.2, 4.7, 3.5, 4.5, 5.4)
treatment1 <- factor(c(1, 1, 1, 2, 2, 2, 3, 3, 3))

# one-way ANOVA
anova1 <- aov(outcome ~ treatment1)
summary(anova1)
etaSquared(anova1)

# include the full ANOVA table
etaSquared(anova1, anova = TRUE)

# two-way ANOVA
treatment2 <- factor(c(1, 2, 3, 1, 2, 3, 1, 2, 3))
anova2 <- aov(outcome ~ treatment1 + treatment2)
etaSquared(anova2)
```

expandFactors

Expand factors to a set of contrasts

Description

Replaces each factor variable in a data frame with its contrast-coded columns, leaving numeric variables unchanged.

Usage

```
expandFactors(data, ...)
```

Arguments

<code>data</code>	A data frame.
<code>...</code>	Additional arguments passed to model.matrix , such as <code>contrasts.arg</code> for specifying non-default contrast schemes.

Details

Each factor in data is replaced by the numeric contrast columns that `model.matrix` would generate for that factor (using treatment contrasts by default). Numeric variables pass through unchanged. This can be helpful when illustrating the connection between ANOVA and regression.

Value

A data frame with factor columns replaced by numeric contrast columns.

See Also

[model.matrix](#), [contrasts](#)

Examples

```
grading <- data.frame(
  teacher = factor(c("Amy", "Amy", "Ben", "Ben", "Cat")),
  gender   = factor(c("male", "female", "female", "male", "male")),
  grade    = c(75, 80, 45, 50, 65)
)

# expand using the default contrasts (treatment contrasts)
expandFactors(grading)

# specify different contrasts via contrasts.arg
my.contrasts <- list(teacher = "contr.helmert", gender = "contr.treatment")
expandFactors(grading, contrasts.arg = my.contrasts)
```

goodnessOfFitTest	<i>Chi-square goodness of fit test</i>
-------------------	--

Description

Runs a chi-square goodness of fit test to check whether the observed frequencies in a categorical variable match a set of hypothesised probabilities.

Usage

```
goodnessOfFitTest(x, p = NULL)
```

Arguments

x	A factor variable containing the observed outcomes.
p	A numeric vector of hypothesised probabilities, one per level of x. The values must sum to 1. If named, the names must match the levels of x (order does not matter). If omitted, all outcomes are assumed to be equally likely.

Details

The test checks whether the observed frequencies for a categorical variable are consistent with the probabilities specified in `p`.

Missing values in `x` are removed before the test is run, and a warning is issued if any cases are dropped. If the probabilities in `p` do not sum exactly to 1, they are rescaled with a warning.

If `x` has unused factor levels (levels with zero observed cases), a warning is issued. Those levels are included in the test with zero observed cases, which changes the degrees of freedom and may give misleading results. Call [droplevels](#) on the data first if this is not intended.

Value

Prints a summary of the test showing the variable name, null and alternative hypotheses, a table of observed frequencies, expected frequencies, and hypothesised probabilities, and the test results (chi-square statistic, degrees of freedom, p-value). The underlying results are also returned as a list, so the output can be assigned to a variable and inspected if needed.

See Also

[chisq.test](#), [associationTest](#), [cramersV](#)

Examples

```
# raw data
gender <- factor(
  c(
    "male", "male", "male", "male", "female", "female",
    "female", "male", "male", "male"
  )
)

# goodness of fit test against the hypothesis that males and
# females occur with equal frequency
goodnessOfFitTest(gender)

# goodness of fit test against the hypothesis that males appear
# with probability .6 and females with probability .4.
goodnessOfFitTest(gender, p = c(.4, .6))
goodnessOfFitTest(gender, p = c(female = .4, male = .6))
goodnessOfFitTest(gender, p = c(male = .6, female = .4))
```

importList

Import a list into the workspace

Description

Copies each element of a list into a separate variable in the workspace, using the list element names as variable names.

Usage

```
importList(x, ask = TRUE)
```

Arguments

x	A list or data frame whose elements are to be imported as individual variables.
ask	Set to TRUE (the default) to display the names of the variables that will be created and ask for confirmation before proceeding. Set to FALSE to import silently.

Details

Creates one variable per list element in the calling environment (usually the global workspace). All elements of x must be named; passing an unnamed or partially-named list is an error. Element names that are not valid R variable names are automatically converted using [make.names](#). An empty list produces a message and returns invisibly without creating any variables.

Value

Called primarily for its side effect of creating variables in the workspace. Invisibly returns 1 if variables were created, 0 if the user declined.

See Also

[unlist](#), [attach](#)

Examples

```
values <- c(1, 2, 3, 4, 5)
group <- c("group A", "group A", "group B", "group B", "group B")

# split() returns a named list: one element per group
grp_list <- split(values, group)

# import silently (no confirmation prompt)
importList(grp_list, ask = FALSE)

if (FALSE) {
  # interactive: shows variable names and asks for confirmation
  importList(grp_list)
}
```

`independentSamplesTTest`*Independent samples t-test*

Description

Runs an independent-samples t-test and prints the results in a readable format.

Usage

```
independentSamplesTTest(  
  formula,  
  data = NULL,  
  var.equal = FALSE,  
  one.sided = FALSE,  
  conf.level = 0.95  
)
```

Arguments

<code>formula</code>	A formula of the form <code>outcome ~ group</code> , where <code>outcome</code> is the numeric variable being measured and <code>group</code> is a factor with exactly two levels.
<code>data</code>	An optional data frame containing the variables named in <code>formula</code> . Tibbles are accepted and converted automatically. If <code>data</code> is omitted the variables are looked up in the workspace.
<code>var.equal</code>	Set to <code>TRUE</code> to run Student's t-test, which assumes equal group variances. The default (<code>FALSE</code>) runs Welch's t-test, which is safer when variances may differ between groups.
<code>one.sided</code>	Set to <code>FALSE</code> (default) for a two-sided test. Set to the name of the group expected to have the larger mean for a one-sided test (e.g., <code>one.sided = "group2"</code>).
<code>conf.level</code>	The confidence level for the confidence interval. The default is <code>0.95</code> for a 95% interval.

Details

Runs an independent-samples t-test comparing the means of two groups, and prints the results in a beginner-friendly format. The calculations are done by `t.test` and `cohensD`. When `var.equal = TRUE`, Cohen's d uses a pooled standard deviation; when `var.equal = FALSE` (Welch's test), it uses the "unequal" method. Cases with missing values are removed with a warning.

Value

Prints a summary showing the outcome and grouping variable names, group means and standard deviations, null and alternative hypotheses, test results (t-statistic, degrees of freedom, p-value), a confidence interval, and Cohen's d as a measure of effect size. The underlying results are also returned as a list, so the output can be assigned to a variable and inspected if needed.

See Also

[t.test](#), [oneSampleTTest](#), [pairedSamplesTTest](#), [cohensD](#)

Examples

```
df <- data.frame(
  rt = c(451, 562, 704, 324, 505, 600, 829),
  cond = factor(x = c(1, 1, 1, 2, 2, 2, 2), labels = c("group1", "group2"))
)

# Welch's t-test (the default, does not assume equal variances)
independentSamplesTTest(rt ~ cond, df)

# Student's t-test (assumes equal variances)
independentSamplesTTest(rt ~ cond, df, var.equal = TRUE)

# one-sided test: is group1 larger?
independentSamplesTTest(rt ~ cond, df, one.sided = "group1")

# missing values are removed with a warning
df$rt[1] <- NA
df$cond[7] <- NA
independentSamplesTTest(rt ~ cond, df)
```

longToWide	<i>Reshape from long to wide</i>
------------	----------------------------------

Description

Reshapes a data frame from long form (one row per observation) to wide form (one row per subject), using a formula to specify the structure.

Usage

```
longToWide(data, formula, sep = "_")
```

Arguments

data	A long-form data frame with one row per observation.
formula	A two-sided formula of the form <code>measure ~ within</code> , listing the measured variable(s) on the left and the within-subject variable(s) on the right. All other variables in data are treated as between-subject variables. Multiple variables are supported on each side, e.g. <code>rt + accuracy ~ day + session</code> .
sep	The separator string used to construct wide-form variable names. Defaults to <code>"_"</code> . For example, with <code>sep = "_"</code> and a measure called <code>accuracy</code> at levels <code>t1</code> and <code>t2</code> , the output columns are named <code>accuracy_t1</code> and <code>accuracy_t2</code> .

Details

This function is the companion to [wideToLong](#). It reshapes a long-form data frame into wide form by spreading the within-subject observations across columns, with column names constructed from the measure name and factor level(s) joined by sep.

Value

A wide-form data frame with one row per subject (or experimental unit). Column names for the repeated measures follow the naming convention used by [wideToLong](#): the measure name followed by the within-subject factor level(s), separated by sep.

See Also

[wideToLong](#), [reshape](#)

Examples

```
long <- data.frame(
  id      = c(1, 2, 3, 1, 2, 3, 1, 2, 3),
  time    = c("t1", "t1", "t1", "t2", "t2", "t2", "t3", "t3", "t3"),
  accuracy = c(.50, .03, .72, .94, .63, .49, .78, .71, .16)
)

longToWide(long, accuracy ~ time)
```

mode	<i>Sample mode</i>
------	--------------------

Description

Calculate the most frequently occurring value(s) in a sample (`modeOf`) or the frequency of the most common value (`maxFreq`).

Usage

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

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

Arguments

- x A vector or factor containing the observations.
- na.rm Set to TRUE (the default) to remove missing values before computing the mode. Set to FALSE to treat NA as a possible modal value (see Details).

Details

When `na.rm = FALSE`, missing values are treated as a distinct value that can itself be the mode. If the number of NAs exceeds the frequency of every other value, `modeOf` returns NA and `maxFreq` returns the count of missing values.

Because of this ambiguity, the default is `na.rm = TRUE`, unlike most other functions in this package.

Value

`modeOf` returns the most frequently observed value. If multiple values are tied for the highest frequency, all of them are returned as a vector. If the input has no non-missing values, `modeOf` issues a warning and returns NA. `maxFreq` returns the modal frequency as a single number, or NA with a warning if the input has no non-missing values.

See Also

[mean](#), [median](#), [table](#)

Examples

```
eyes <- c("green", "green", "brown", "brown", "blue")
modeOf(eyes) # returns c("green", "brown") -- a tie
maxFreq(eyes) # returns 2

# with missing data
eyes <- c("green", "green", "brown", "brown", "blue", NA, NA, NA)

# na.rm = FALSE: NA is the most frequent "value"
modeOf(eyes, na.rm = FALSE)
maxFreq(eyes, na.rm = FALSE)

# na.rm = TRUE: missing values ignored
modeOf(eyes, na.rm = TRUE)
maxFreq(eyes, na.rm = TRUE)
NULL
```

oneSampleTTest

One sample t-test

Description

Runs a one-sample t-test and prints the results in a readable format.

Usage

```
oneSampleTTest(x, mu, one.sided = FALSE, conf.level = 0.95)
```

Arguments

<code>x</code>	A numeric vector containing the data to be tested.
<code>mu</code>	The hypothesised population mean to test against.
<code>one.sided</code>	Set to FALSE (default) for a two-sided test. Set to "greater" if you expect the population mean to be above mu, or "less" if you expect it to be below.
<code>conf.level</code>	The confidence level for the confidence interval. The default is 0.95 for a 95% interval.

Details

Runs a one-sample t-test comparing the mean of `x` to the hypothesised value `mu`, and prints the results in a beginner-friendly format. The calculations are done by [t.test](#) and [cohensD](#). Missing values in `x` are removed with a warning.

Value

Prints a summary showing the variable name, descriptive statistics, null and alternative hypotheses, test results (t-statistic, degrees of freedom, p-value), a confidence interval, and Cohen's d as a measure of effect size. The underlying results are also returned as a list, so the output can be assigned to a variable and inspected if needed.

See Also

[t.test](#), [pairedSamplesTTest](#), [independentSamplesTTest](#), [cohensD](#)

Examples

```
likert <- c(3, 1, 4, 1, 4, 6, 7, 2, 6, 6, 7)

# two-sided test (the default)
oneSampleTTest(x = likert, mu = 4)

# one-sided test: is the mean greater than 4?
oneSampleTTest(x = likert, mu = 4, one.sided = "greater")

# wider confidence interval
oneSampleTTest(x = likert, mu = 4, conf.level = 0.99)

# missing values are removed with a warning
likert <- c(3, NA, 4, NA, 4, 6, 7, NA, 6, 6, 7)
oneSampleTTest(x = likert, mu = 4)
```

pairedSamplesTTest	<i>Paired samples t-test</i>
--------------------	------------------------------

Description

Runs a paired-samples t-test and prints the results in a readable format.

Usage

```
pairedSamplesTTest(  
  formula,  
  data = NULL,  
  id = NULL,  
  one.sided = FALSE,  
  conf.level = 0.95  
)
```

Arguments

formula	A formula describing the data. For wide-format data use a one-sided formula such as <code>~ time1 + time2</code> . For long-format data use <code>outcome ~ group + (id)</code> , or <code>outcome ~ group</code> together with the <code>id</code> argument.
data	An optional data frame containing the variables named in formula. Tibbles are accepted and converted automatically. If data is omitted the variables are looked up in the workspace.
id	The name of the participant ID variable as a character string (e.g., <code>id = "subject"</code>). Required when using long-format data with a plain <code>outcome ~ group</code> formula instead of <code>outcome ~ group + (id)</code> .
one.sided	Set to <code>FALSE</code> (default) for a two-sided test. Set to the name of the group or variable expected to have the larger mean for a one-sided test (e.g., <code>one.sided = "time2"</code>).
conf.level	The confidence level for the confidence interval. The default is <code>0.95</code> for a 95% interval.

Details

Runs a paired-samples t-test and prints the results in a beginner-friendly format. The calculations are done by [t.test](#) and [cohensD](#).

There are two ways to supply data. If the data are in **wide format** (one row per participant, with the two measurements in separate columns), use a one-sided formula such as `~ time1 + time2`. The first row of `time1` is paired with the first row of `time2`, and so on.

If the data are in **long format** (two rows per participant), use a two-sided formula. The recommended style is `outcome ~ group + (id)`, where the participant ID variable is enclosed in parentheses. Alternatively, use the plain formula `outcome ~ group` and supply the ID variable name via

the `id` argument. The lme4-style notation `outcome ~ group + (1|id)` is also accepted as equivalent to `outcome ~ group + (id)`.

Participants with missing measurements are removed with a warning.

Value

Prints a summary showing the variable names, descriptive statistics (including the mean and standard deviation of the differences), null and alternative hypotheses, test results (t-statistic, degrees of freedom, p-value), a confidence interval, and Cohen's d as a measure of effect size. The underlying results are also returned as a list, so the output can be assigned to a variable and inspected if needed.

See Also

[t.test](#), [oneSampleTTest](#), [independentSamplesTTest](#), [cohensD](#)

Examples

```
# long-format data: one row per participant per time point
df <- data.frame(
  id = factor(
    x = c(1, 1, 2, 2, 3, 3, 4, 4),
    labels = c("alice", "bob", "chris", "diana")
  ),
  time = factor(
    x = c(1, 2, 1, 2, 1, 2, 1, 2),
    labels = c("time1", "time2")
  ),
  wm = c(3, 4, 6, 6, 9, 12, 7, 9)
)

# wide-format data: one row per participant
df2 <- longToWide(df, wm ~ time)

# three equivalent ways to run the same test
pairedSamplesTTest(formula = wm ~ time, data = df, id = "id")
pairedSamplesTTest(formula = wm ~ time + (id), data = df)
pairedSamplesTTest(formula = ~ wm_time1 + wm_time2, data = df2)

# one-sided test: is time2 larger than time1?
pairedSamplesTTest(formula = wm ~ time, data = df, id = "id", one.sided = "time2")

# missing value: that participant is removed with a warning
df$wm[1] <- NA
pairedSamplesTTest(formula = wm ~ time, data = df, id = "id")

# missing row: that participant is also removed with a warning
df <- df[-1, ]
pairedSamplesTTest(formula = wm ~ time, data = df, id = "id")
```

permuteLevels	<i>Permute the levels of a factor</i>
---------------	---------------------------------------

Description

Reorder the levels of a factor into any order you specify.

Usage

```
permuteLevels(x, perm, ordered = is.ordered(x), invert = FALSE)
```

Arguments

x	A factor.
perm	An integer vector of the same length as nlevels(x). When invert = FALSE (the default), perm[k] gives the index of the old level that should appear in position k of the new ordering. When invert = TRUE, perm[k] gives the new position to assign to the k-th old level.
ordered	Set to TRUE to return an ordered factor. Defaults to is.ordered(x), preserving the ordered status of the input.
invert	Set to TRUE to apply the inverse of perm. See the perm description above and the examples below.

Details

Similar to [relevel](#), but more general: relevel can only move one level to the front, whereas permuteLevels can place the levels in any order. This is useful when you want to control the order in which levels appear on a plot axis or in a table.

Value

A factor with the same values as x but with the levels in the new order specified by perm.

See Also

[factor](#), [relevel](#), [order](#)

Examples

```
# factor with levels a, b, c, d, e, f (in that order)
x <- factor(c(1, 4, 2, 2, 3, 3, 5, 5, 6, 6), labels = letters[1:6])
levels(x)

# move level e to position 1, c to position 2, b to 3, a to 4, d to 5, f to 6
permuteLevels(x, perm = c(5, 3, 2, 1, 4, 6))

# using invert = TRUE: move level a to position 5, b to 3, c to 2, etc.
permuteLevels(x, perm = c(5, 3, 2, 1, 4, 6), invert = TRUE)
```

posthocPairwiseT	<i>Post-hoc pairwise t-tests for ANOVA</i>
------------------	--

Description

Runs pairwise t-tests for a one-way analysis of variance, with corrections for multiple comparisons.

Usage

```
posthocPairwiseT(x, ...)
```

Arguments

x	An aov object, as returned by aov . Only one-way ANOVA models are supported.
...	Additional arguments passed to pairwise.t.test , such as <code>p.adjust.method</code> .

Details

Takes a fitted one-way ANOVA object and runs pairwise t-tests for all pairs of groups, applying a correction for multiple comparisons. This is a simpler alternative to [TukeyHSD](#) that uses the same correction methods (e.g., Holm, Bonferroni) as [pairwise.t.test](#).

Value

Prints a table of p-values for all pairwise group comparisons. The underlying result is also returned as a list (with the same structure as [pairwise.t.test](#)) so it can be assigned to a variable and inspected if needed.

See Also

[pairwise.t.test](#), [TukeyHSD](#), [aov](#)

Examples

```
dataset <- data.frame(
  outcome = c(1, 2, 3, 2, 3, 4, 5, 6, 7),
  group = factor(c("a", "a", "a", "b", "b", "b", "c", "c", "c"))
)

anova1 <- aov(outcome ~ group, data = dataset)
summary(anova1)

# post-hoc pairwise comparisons with Holm correction (the default)
posthocPairwiseT(anova1)

# Bonferroni correction instead
posthocPairwiseT(anova1, p.adjust.method = "bonferroni")
```

print.assocTest	<i>Print chi-square association test results</i>
-----------------	--

Description

Prints the results of a chi-square test of association in a readable format. This function is called automatically whenever a result from [associationTest](#) is displayed.

Usage

```
## S3 method for class 'assocTest'  
print(x, ...)
```

Arguments

x	An association test result, as returned by associationTest .
...	Additional arguments (unused, included for compatibility).

Value

Invisibly returns x unchanged.

print.correlate	<i>Print correlation matrix results</i>
-----------------	---

Description

Prints the results of a correlation analysis in a readable format. This function is called automatically whenever a result from [correlate](#) is displayed.

Usage

```
## S3 method for class 'correlate'  
print(x, ...)
```

Arguments

x	A correlation result, as returned by correlate .
...	Additional arguments (unused, included for compatibility).

Value

Invisibly returns x unchanged.

print.gofTest	<i>Print goodness of fit test results</i>
---------------	---

Description

Prints the results of a chi-square goodness of fit test in a readable format. This function is called automatically whenever a result from [goodnessOfFitTest](#) is displayed.

Usage

```
## S3 method for class 'gofTest'
print(x, ...)
```

Arguments

x	A goodness of fit test result, as returned by goodnessOfFitTest .
...	Additional arguments (unused, included for compatibility).

Value

Invisibly returns x unchanged.

print.TTest	<i>Print t-test results</i>
-------------	-----------------------------

Description

Prints the results of a t-test in a readable, beginner-friendly format. This function is called automatically whenever a result from [oneSampleTTest](#), [independentSamplesTTest](#), or [pairedSamplesTTest](#) is displayed.

Usage

```
## S3 method for class 'TTest'
print(x, ...)
```

Arguments

x	A t-test result, as returned by oneSampleTTest , independentSamplesTTest , or pairedSamplesTTest .
...	Additional arguments (unused, included for compatibility).

Value

Invisibly returns x unchanged.

print.whoList	<i>Print workspace summary</i>
---------------	--------------------------------

Description

Prints a workspace summary in a readable format. This function is called automatically whenever a result from [who](#) is displayed.

Usage

```
## S3 method for class 'whoList'
print(x, ...)
```

Arguments

x	A workspace summary, as returned by who .
...	Additional arguments (unused, included for compatibility).

Value

Invisibly returns x unchanged.

quantileCut	<i>Cut by quantiles</i>
-------------	-------------------------

Description

Divides a numeric variable into n categories that each contain approximately the same number of observations.

Usage

```
quantileCut(x, n, ...)
```

Arguments

x	A numeric vector.
n	The number of categories to create.
...	Additional arguments passed to cut , such as labels.

Details

Unlike [cut](#), which creates categories of equal width, [quantileCut](#) uses [quantile](#) to find breakpoints that produce roughly equal-sized groups. This can be useful in exploratory analysis, but the resulting categories are data-driven and may not have a clear interpretation. Using them as grouping variables in an ANOVA is generally not recommended, as the breakpoints are arbitrary and the groups will typically not have equal variances.

Value

A factor with n levels. Level labels follow the same convention as `cut` and can be overridden with the `labels` argument.

See Also

`cut`, `quantile`

Examples

```
# the data are unevenly spread, so equal-width bins would be unbalanced
x <- c(0, 1, 2, 3, 4, 5, 7, 10, 15)

# quantileCut creates equal-frequency bins
bins_eq_freq <- quantileCut(x, 3)
table(bins_eq_freq)

# compare to cut(), which creates equal-width bins
bins_eq_width <- cut(x, 3)
table(bins_eq_width)
```

rmAll

Remove all objects from the workspace

Description

Deletes all objects from the workspace, with an optional confirmation prompt.

Usage

```
rmAll(ask = TRUE)
```

Arguments

ask	Set to TRUE (the default) to display the current workspace contents and ask for confirmation before deleting. Set to FALSE to delete immediately without prompting.
-----	---

Details

Removes all objects from the workspace. When `ask = TRUE`, the list of objects is printed and the user must type `y` to confirm before anything is deleted. This is similar to `rm(list = objects())`, but with an interactive safety check.

Value

Invisibly returns 1 if objects were deleted, 0 if the user declined or the workspace was already empty.

See Also[rm](#), [objects](#)**Examples**

```

if (FALSE) {
  # interactive: displays workspace contents and asks for confirmation
  rmAll()

  # non-interactive: deletes immediately without prompting
  rmAll(ask = FALSE)
}

```

sortFrame	<i>Sort a data frame</i>
-----------	--------------------------

Description

Sorts a data frame by one or more of its variables.

Usage

```
sortFrame(x, ..., alphabetical = TRUE)
```

Arguments

<code>x</code>	A data frame to be sorted.
<code>...</code>	One or more unquoted variable names to sort by, in order of priority. Prefix a variable name with <code>-</code> to sort in descending order (e.g., <code>sortFrame(x, -a, b)</code> sorts descending by <code>a</code> , then ascending by <code>b</code>).
<code>alphabetical</code>	Set to <code>TRUE</code> (the default) to sort character variables case-insensitively in alphabetical order. Set to <code>FALSE</code> to use the locale-dependent ordering used by sort .

Details

Sorts the rows of `x` by the variables listed in `...`. Numeric variables and factors are sorted by their numeric values (for factors, this corresponds to level order). Character variables are sorted alphabetically by default, ignoring case; prefix with `-` for reverse alphabetical order.

Simple expressions combining variables are also accepted (e.g., `sortFrame(x, a + b)` sorts by the sum of `a` and `b`), though care is required: the sort uses [xtfrm](#) internally to convert variables to sortable numeric codes, which can produce unexpected results for character variables when expressions other than `-` are used.

Value

The sorted data frame.

See Also[order](#), [sort](#), [xtfrm](#)**Examples**

```
dataset <- data.frame(
  txt = c("bob", "Clare", "clare", "bob", "eve", "eve"),
  num1 = c(3, 1, 2, 0, 0, 2),
  num2 = c(1, 1, 3, 0, 3, 2),
  stringsAsFactors = FALSE
)

sortFrame(dataset, num1) # sort by num1 ascending
sortFrame(dataset, num1, num2) # sort by num1 then num2
sortFrame(dataset, -num1) # sort by num1 descending
sortFrame(dataset, txt) # sort alphabetically (case-insensitive)
```

standardCoefs

*Standardised regression coefficients***Description**

Calculates standardised regression coefficients (beta weights) for a linear model.

Usage

```
standardCoefs(x)
```

Arguments

x A linear model, as returned by [lm](#).

Details

Standardised coefficients are the regression coefficients that would result from fitting the model after scaling all predictors and the outcome to have mean 0 and variance 1. They can be useful for comparing the relative magnitude of predictors measured on different scales, though this comparison should be interpreted with care.

Note that when a model contains interaction terms, the interaction column is also standardised as a whole, rather than being constructed from standardised versions of the constituent predictors.

Value

A matrix with one row per predictor (excluding the intercept) and two columns: b (unstandardised coefficient) and beta (standardised coefficient).

See Also[lm](#), [coefficients](#)

Examples

```

X1 <- c(0.69, 0.77, 0.92, 1.72, 1.79, 2.37, 2.64, 2.69, 2.84, 3.41)
Y <- c(3.28, 4.23, 3.34, 3.73, 5.33, 6.02, 5.16, 6.49, 6.49, 6.05)

# simple linear regression
model1 <- lm(Y ~ X1)
coefficients(model1) # unstandardised
standardCoefs(model1) # unstandardised and standardised side by side

# multiple regression
X2 <- c(0.19, 0.22, 0.95, 0.43, 0.51, 0.04, 0.12, 0.44, 0.38, 0.33)
model2 <- lm(Y ~ X1 + X2)
standardCoefs(model2)

# model with an interaction term
model3 <- lm(Y ~ X1 * X2)
standardCoefs(model3)

```

tFrame

*Transpose a data frame***Description**

Transposes a data frame, swapping rows and columns, and returns the result as a data frame.

Usage

```
tFrame(x)
```

Arguments

x A data frame to be transposed.

Details

Equivalent to `as.data.frame(t(x))`. Unlike applying `t` directly, `tFrame` ensures the result is a data frame rather than a matrix. This makes sense when the rows of `x` can be meaningfully treated as variables — for example, when each row represents a measurement type and each column represents a participant.

Value

The transposed data frame.

See Also

[t](#)

Examples

```
dataset <- data.frame(  
  Gf = c(105, 119, 121, 98), # fluid intelligence  
  Gc = c(110, 115, 119, 103), # crystallised intelligence  
  Gs = c(112, 102, 108, 99) # processing speed  
)  
rownames(dataset) <- paste0("person", 1:4)  
dataset  
  
tFrame(dataset)
```

unlibrary

Unload a package

Description

Removes a package from the search path, using the same naming convention as [library](#).

Usage

```
unlibrary(package)
```

Arguments

package The name of the package to unload, with or without quotes.

Details

Calls [detach](#) on the named package. Unlike [detach](#), which requires the full "package:name" string, [unlibrary](#) accepts the bare package name (with or without quotes), matching the syntax of [library](#). Only the named package is unloaded; dependencies are not affected.

Value

Called for its side effect of removing the package from the search path. Returns the result of [detach](#) invisibly.

See Also

[library](#), [detach](#)

Examples

```
if (FALSE) {  
  # after loading a package with library(), unload it with unlibrary()  
  unlibrary(MASS)  
}
```

Description

Prints a summary of all objects in the workspace, showing each object's name, class, and size.

Usage

```
who(expand = FALSE)
```

Arguments

expand	Set to TRUE to also list the variables inside any data frames in the workspace. Defaults to FALSE.
--------	--

Details

Shows each object's name, class, and size. For objects with explicit dimensions (e.g., data frames, matrices) the size is shown as rows x columns; for other objects it is the length. Size is only shown for objects whose mode is numeric, character, logical, complex, or list.

Shows more information than [objects](#) (especially for variables inside data frames) but less detail than [ls.str](#).

Value

Prints the workspace summary and invisibly returns the underlying data (a data frame with columns Name, Class, and Size), which can be assigned to a variable and inspected if needed.

See Also

[objects](#), [ls.str](#)

Examples

```
cats <- 4
mood <- "happy"
who()

dataset <- data.frame(
  hi = c("hello", "cruel", "world"),
  pi = c(3, 1, 4)
)

who()
who(expand = TRUE)
```

wideToLong	<i>Reshape from wide to long</i>
------------	----------------------------------

Description

Reshapes a data frame from wide form (one row per subject) to long form (one row per observation), using variable names to determine the structure.

Usage

```
wideToLong(data, within = "within", sep = "_", split = TRUE)
```

Arguments

<code>data</code>	A wide-form data frame with one row per subject (or experimental unit). Variables whose names contain <code>sep</code> are treated as repeated measures; all others are treated as between-subject variables.
<code>within</code>	A character string, or vector of strings, giving the name(s) to use for the within-subject factor column(s) in the output. Defaults to "within".
<code>sep</code>	The separator string used in the wide-form variable names to separate the measure name from the factor level(s). Defaults to "_". The separator must not appear anywhere else in the variable names.
<code>split</code>	Set to TRUE (the default) to split multiple within-subject factors into separate columns in the output. Set to FALSE to keep them combined into a single column.

Details

This function is the companion to [longToWide](#). It determines the reshape structure from the variable names rather than requiring an explicit formula.

The naming scheme for repeated-measures variables places the measure name first, followed by the factor level(s), all joined by `sep`. For example, variables named `accuracy_t1` and `accuracy_t2` indicate a measure called `accuracy` recorded at two time points (`t1` and `t2`). After reshaping, the long-form output contains one column called `accuracy` and a factor column (named by the `within` argument) with levels `t1` and `t2`.

Designs with multiple within-subject factors are supported. For example, `MRT_cond1_day1` encodes measure `MRT` at level `cond1` of one factor and `day1` of another. Supply `within = c("condition", "day")` to name both output columns. Multiple measured variables per observation (e.g., both `MRT` and `PC`) are also supported.

Value

A long-form data frame with one row per observation.

See Also

[longToWide](#), [reshape](#)

Examples

```
# simple design: accuracy measured at two time points for 4 participants
wide <- data.frame(
  id      = 1:4,
  accuracy_t1 = c(.15, .50, .78, .55),
  accuracy_t2 = c(.55, .32, .99, .60)
)
wideToLong(wide, "time")

# complex design: two measures (MRT, PC), two conditions, two days
wide2 <- data.frame(
  id      = 1:4,
  gender  = factor(c("male", "male", "female", "female")),
  MRT_cond1_day1 = c(415, 500, 478, 550),
  MRT_cond2_day1 = c(455, 532, 499, 602),
  MRT_cond1_day2 = c(400, 490, 468, 502),
  MRT_cond2_day2 = c(450, 518, 474, 588),
  PC_cond1_day1  = c(79, 83, 91, 75),
  PC_cond2_day1  = c(82, 86, 90, 78),
  PC_cond1_day2  = c(88, 92, 98, 89),
  PC_cond2_day2  = c(93, 97, 100, 95)
)

# default: condition and day become separate columns
wideToLong(wide2, within = c("condition", "day"))

# alternative: keep condition and day as one combined column
wideToLong(wide2, split = FALSE)
```

Index

aad, [2](#)
anova, [13](#)
aov, [13](#), [14](#), [26](#)
associationTest, [3](#), [13](#), [16](#), [27](#)
attach, [17](#)

barplot, [6](#)
bars, [4](#)

cbind, [10](#)
chisq.test, [4](#), [12](#), [13](#), [16](#)
ciMean, [5](#), [6](#), [7](#)
coefficients, [32](#)
cohensD, [8](#), [18](#), [19](#), [22–24](#)
colCopy (copy), [9](#)
contrasts, [15](#)
copy, [9](#)
cor, [11](#)
cor.test, [11](#)
correlate, [10](#), [27](#)
cramersV, [4](#), [12](#), [16](#)
cut, [29](#), [30](#)

detach, [34](#)
droplevels, [4](#), [16](#)

etaSquared, [13](#)
expandFactors, [14](#)

factor, [25](#)

goodnessOfFitTest, [4](#), [13](#), [15](#), [28](#)

importList, [16](#)
independentSamplesTTest, [18](#), [22](#), [24](#), [28](#)

library, [34](#)
lm, [32](#)
longToWide, [19](#), [36](#)
ls.str, [35](#)
make.names, [17](#)

matrix, [10](#)
maxFreq (mode), [20](#)
mean, [3](#), [7](#), [21](#)
median, [21](#)
mode, [20](#)
model.matrix, [14](#), [15](#)
modeOf (mode), [20](#)

objects, [31](#), [35](#)
oneSampleTTest, [19](#), [21](#), [24](#), [28](#)
order, [25](#), [32](#)

p.adjust, [11](#)
pairedSamplesTTest, [19](#), [22](#), [23](#), [28](#)
pairwise.t.test, [26](#)
permuteLevels, [25](#)
posthocPairwiseT, [26](#)
print.assocTest, [27](#)
print.correlate, [27](#)
print.gofTest, [28](#)
print.TTest, [28](#)
print.whoList, [29](#)

quantile, [29](#), [30](#)
quantileCut, [29](#)

rbind, [10](#)
relevel, [25](#)
reshape, [20](#), [36](#)
rm, [31](#)
rmAll, [30](#)
rowCopy (copy), [9](#)

sd, [3](#)
sort, [31](#), [32](#)
sortFrame, [31](#)
standardCoefs, [32](#)
summary.aov, [14](#)

t, [33](#)
t.test, [7](#), [18](#), [19](#), [22–24](#)

table, [21](#)
tFrame, [33](#)
TukeyHSD, [26](#)

unlibrary, [34](#)
unlist, [17](#)

var, [3](#)

who, [29](#), [35](#)
wideToLong, [20](#), [36](#)

xtfrm, [31](#), [32](#)