Package: colleyRstats (via r-universe)

June 12, 2026

Type Package

Title Functions to Streamline Statistical Analysis and Reporting

Version 0.1.0

Description Built upon popular R packages such as 'ggstatsplot' and 'ARTool', this collection offers a wide array of tools for simplifying reproducible analyses, generating high-quality visualizations, and producing 'APA'-compliant outputs. The primary goal of this package is to significantly reduce repetitive coding efforts, allowing you to focus on interpreting results. Whether you're dealing with ANOVA assumptions, reporting effect sizes, or creating publication-ready visualizations, this package makes these tasks easier.

URL `https://github.com/M-Colley/colleyRstats`

BugReports `https://github.com/M-Colley/colleyRstats/issues`

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 8.0.0

Language en-US

Depends R (>= 4.6.0), ggplot2 (>= 4.0.3)

Imports dplyr (>= 1.2.1), effectsize (>= 1.0.2), ggpmisc (>= 0.7.0), ggsignif, ggstatsplot (>= 1.0.0), purrr, rlang, see, stats, statsExpressions (>= 2.0.0), stringr, tidyr, utils

Suggests afex, ARTool (>= 0.11.2), BayesFactor, bayestestR, Cairo, car, clipr, conflicted, dunn.test (>= 1.4.1), DT, emmeans, emoa, flexdashboard, FSA, ggtext, Hmisc, knitr, moocore (>= 0.3.1), nparLD, patchwork, pkgload, psych (>= 2.6.3), readxl, report (>= 0.6.4), reporttools, rmarkdown, rstantools, rstatix, scales, sjPlot, stargazer, styler, testthat (>= 3.3.0), tibble, writexl, xtable

Config/testthat/edition 3

Config/Needs/check anthonynorth/roxyglobals

Config/roxyglobals/unique TRUE

Config/testthat/parallel false

VignetteBuilder knitr

NeedsCompilation no

Author Mark Colley [aut, cre, cph] (ORCID: <`https://orcid.org/0000-0001-5207-5029`>)

Maintainer Mark Colley <`mark.colley@yahoo.de`>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-11 22:40:02 UTC

RemoteUrl https://github.com/cran/colleyRstats

RemoteRef HEAD

RemoteSha f1eb8325140ef9cb44804216308ce66205f8c12a

Contents

`add_pareto_emoa_column`

Add `PARETO_EMOA` *Column to a Data Frame*

Description

This function calculates the Pareto front using emoa for a given set of objectives in a data frame and adds a new column, `PARETO_EMOA`, which indicates whether each row in the data frame belongs to the Pareto front.

Usage

```
add_pareto_emoa_column(data, objectives)
```

Arguments

`data`	A data frame containing the data, including the objective columns.
`objectives`	A character vector specifying the names of the objective columns in `data`. These columns should be numeric and will be used to calculate the Pareto front.

Value

A data frame with the same columns as `data`, along with an additional column, `PARETO_EMOA`, which is `TRUE` for rows that are on the Pareto front and `FALSE` otherwise.

Examples

```
# Define objective columns
objectives <- c("trust", "predictability", "perceivedSafety", "Comfort")

# Example data frame
main_df <- data.frame(
  trust = runif(10),
  predictability = runif(10),
  perceivedSafety = runif(10),
  Comfort = runif(10)
)

# Add the Pareto front column
main_df <- add_pareto_emoa_column(data = main_df, objectives)
head(main_df)
```

`add_pareto_moocore_column`

Add `PARETO_MOOCORE` *Column to a Data Frame*

Description

This function calculates the Pareto front using moocore for a given set of objectives in a data frame and adds a new column, `PARETO_MOOCORE`, which indicates whether each row in the data frame belongs to the Pareto front.

Usage

```
add_pareto_moocore_column(data, objectives)
```

Arguments

`data`	A data frame containing the data, including the objective columns.
`objectives`	A character vector specifying the names of the objective columns in `data`. These columns should be numeric and will be used to calculate the Pareto front.

Value

A data frame with the same columns as `data`, along with an additional column, `PARETO_MOOCORE`, which is `TRUE` for rows that are on the Pareto front and `FALSE` otherwise.

Examples

```
# Define objective columns
objectives <- c("trust", "predictability", "perceivedSafety", "Comfort")

# Example data frame
main_df <- data.frame(
```

```
  trust = runif(10),
  predictability = runif(10),
  perceivedSafety = runif(10),
  Comfort = runif(10)
)

# Add the Pareto front column
main_df <- add_pareto_moocore_column(data = main_df, objectives)
head(main_df)
```

`analyze_and_report` *Analyze one dependent variable and produce everything a paper needs*

Description

One-call pipeline for a single dependent variable: checks the assumptions (producing a ready-made methods sentence via `assumption_methods_text()`), builds the matching **ggstatsplot** figure with automatic parametric/non-parametric selection, reports the omnibus test via `reportggstatsplot()`, and – for more than two groups – reports the significant post-hoc comparisons via `reportggstatsplotPostHoc()`.

Usage

```
analyze_and_report(
  data,
  dv,
  iv,
  design = c("between", "within"),
  ylab = dv,
  xlabels = NULL,
  plotType = "boxviolin",
  sink_to = NULL
)
```

Arguments

`data`	the data frame
`dv`	the dependent variable (column name as string)
`iv`	the independent variable (column name as string); coerced to a factor if it is not one already
`design`	`"between"` for between-subjects data (default) or `"within"` for repeated measures
`ylab`	label for the dependent variable; defaults to `dv`
`xlabels`	optional labels for the x-axis
`plotType`	either "box", "violin", or "boxviolin" (default)
`sink_to`	optional path of a `.tex` file; the methods sentence, omnibus result, and post-hoc sentences are written there so a manuscript can `\input{}` them

Value

Invisibly returns a list with components `plot` (the ggplot), `methods` (assumption-check sentence), `text` (omnibus result), `posthoc` (post-hoc sentences, or `NULL` for two groups), and `sentences` (all text combined, in manuscript order).

Examples

```
result <- analyze_and_report(mtcars, dv = "mpg", iv = "cyl")
result$plot
```

`assumption_methods_text` *Methods-section sentence justifying the test selection*

Description

Runs the group-wise Shapiro-Wilk normality check (and optionally Levene's test for homogeneity of variances) and turns the outcome into a ready-made methods-section sentence, including the relevant statistics. This is the justification reviewers expect next to the choice of a parametric or non-parametric test.

Usage

```
assumption_methods_text(data, x, y, include_homogeneity = FALSE)
```

Arguments

`data`	the data frame
`x`	the grouping variable (column name as string)
`y`	the dependent variable (column name as string)
`include_homogeneity`	whether to also report Levene's test. Useful for between-subjects designs. Default `FALSE`.

Value

Invisibly returns the sentence(s) as a single string; the text is also emitted via `message()`.

Examples

```
set.seed(1)
d <- data.frame(g = rep(c("A", "B"), each = 20), v = rnorm(40))
assumption_methods_text(d, x = "g", y = "v")
```

`check_homogeneity_by_group` *Check homogeneity of variances across groups*

Description

Check homogeneity of variances across groups

Usage

```
check_homogeneity_by_group(data, x, y)
```

Arguments

`data`	the data frame
`x`	the grouping variable (column name as string)
`y`	the dependent variable (column name as string)

Value

TRUE if Levene's test is non-significant (p >= .05), FALSE otherwise. The Levene test result (columns `df1`, `df2`, `statistic`, `p`) is attached in the `"test"` attribute, e.g. for use in a methods section via `assumption_methods_text()`.

`check_normality_by_group` *Check normality for groups*

Description

Check normality for groups

Usage

```
check_normality_by_group(data, x, y)
```

Arguments

`data`	the data frame
`x`	the x column
`y`	the y column

Value

TRUE if all groups are normal, FALSE otherwise. The per-group Shapiro-Wilk statistics are attached as a data frame in the `"tests"` attribute (columns: group, `W`, `p_value`), e.g. for use in a methods section via `assumption_methods_text()`. For groups with more than 5000 non-missing values, Shapiro-Wilk is computed on a random sample of 5000 observations (a warning is emitted); the returned value still reflects that sampled test. Because the sample is drawn randomly, results for such large groups are not reproducible unless a seed is set beforehand.

`checkAssumptionsForAnova` *Check the assumptions for an ANOVA with a variable number of factors: Normality and Homogeneity of variance assumption.*

Description

Check the assumptions for an ANOVA with a variable number of factors: Normality and Homogeneity of variance assumption.

Usage

```
checkAssumptionsForAnova(data, y, factors)

check_assumptions_anova(data, y, factors)
```

Arguments

`data`	the data frame
`y`	The dependent variable for which assumptions should be checked
`factors`	A character vector of factor names

Value

A message indicating whether to use parametric or non-parametric ANOVA

Examples

```
set.seed(123)

main_df <- data.frame(
  tlx_mental      = rnorm(40),
  Video           = factor(rep(c("A", "B"), each = 20)),
  DriverPosition  = factor(rep(c("Left", "Right"), times = 20))
)

checkAssumptionsForAnova(
  data    = main_df,
  y       = "tlx_mental",
```

```
  factors = c("Video", "DriverPosition")
)
```

`cite_methods` *Citations and methods boilerplate for the analyses used*

Description

Prints a ready-made methods phrase plus the BibTeX entries for the R packages behind the requested analysis methods, so a manuscript's methods section and bibliography can be filled in one step.

Usage

```
cite_methods(methods = c("ggstatsplot", "effectsize"), bibtex = TRUE)
```

Arguments

`methods`	Character vector of analysis methods to cite. Any of `"art"` (Aligned Rank Transform via ARTool), `"dunn"` (Dunn's test via FSA), `"nparld"` (nparLD), `"ggstatsplot"`, `"effectsize"`, and `"colleyrstats"` (this package).
`bibtex`	whether to include the BibTeX entries. Default `TRUE`.

Value

Invisibly returns the generated lines as a character vector; the text is also emitted via `message()`. Methods whose package is not installed are skipped with a message.

Examples

```
cite_methods("ggstatsplot", bibtex = FALSE)
```

`colleyRstats_setup` *Configure Global R Environment for colleyRstats*

Description

Sets ggplot2 themes and conflict preferences to match the standards used in the colleyRstats workflow.

Usage

```
colleyRstats_setup(
  set_options = FALSE,
  set_theme = TRUE,
  set_conflicts = TRUE,
  print_citation = TRUE,
  verbose = TRUE
)
```

Arguments

`set_options`	Logical. If `TRUE`, prints a notice that global options are no longer changed automatically. Default is `FALSE`.
`set_theme`	Logical. If `TRUE`, sets the default `ggplot2` theme to `see::theme_lucid` with custom modifications. Default is `TRUE`.
`set_conflicts`	Logical. If `TRUE`, sets `conflicted` preferences to favor `dplyr` and other tidyverse packages. Default is `TRUE`.
`print_citation`	Logical. If `TRUE`, prints the citation information for this package. Default is `TRUE`.
`verbose`	Logical. If `TRUE`, emit informational messages. Default is `TRUE`.

Value

Invisibly returns `NULL`.

Examples

```
# Runs everywhere, no extra packages, no session side effects
colleyRstats::colleyRstats_setup(
  set_options = FALSE,
  set_theme = FALSE,
  set_conflicts = FALSE,
  print_citation = FALSE,
  verbose = FALSE
)


# Full setup (requires suggested packages; changes session defaults)
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("see", quietly = TRUE)) {
  local({
    old_theme <- ggplot2::theme_get()
    on.exit(ggplot2::theme_set(old_theme), add = TRUE)

    colleyRstats::colleyRstats_setup(
      set_options = FALSE,
      set_conflicts = FALSE,   # avoid persisting conflict prefs in checks
      print_citation = FALSE,
      verbose = TRUE
```

```
    )

    ggplot2::ggplot(mtcars, ggplot2::aes(mpg, wt)) +
      ggplot2::geom_point()
  })
}
```

`data the data frame` *Replace values across a data frame*

Description

Replace all occurrences of given values in all columns of a data frame.

The `data` data frame contains a collection of records, with attributes organized in columns. It may include various types of values, such as numerical, categorical, or textual data.

Usage

```
replace_values(data, to_replace, replace_with)
```

Arguments

`data`	The input data frame to be modified.
`to_replace`	A vector of values to be replaced within the data frame. This must be the same length as `replace_with`.
`replace_with`	A vector of corresponding replacement values. This must be the same length as `to_replace`.

Value

Modified data frame with specified values replaced.

Examples

```
data <- data.frame(
  q1 = c("neg2", "neg1", "0"),
  q2 = c("1", "neg2", "neg1")
)

replace_values(
  data,
  to_replace = c("neg2", "neg1"),
  replace_with = c("-2", "-1")
)
```

`debug_contr_error` *Debug contrast errors in ANOVA-like models*

Description

Debug contrast errors in ANOVA-like models

Usage

```
debug_contr_error(dat, subset_vec = NULL)
```

Arguments

`dat`	A data frame of predictors.
`subset_vec`	Optional logical or numeric index vector used to subset rows before checks.

Value

A list with two elements:

nlevels Integer vector giving the number of levels for each factor variable in `dat`.

levels List of factor level labels for each factor variable in `dat`.

Examples

```
dat <- data.frame(
  group = factor(rep(letters[1:3], each = 3)),
  score = rnorm(9)
)

debug_contr_error(dat = dat)
```

`generateEffectPlot` *Function to define a plot, either showing the main or interaction effect in bold.*

Description

Function to define a plot, either showing the main or interaction effect in bold.

Usage

```
generateEffectPlot(
  data,
  x,
  y,
  fillColourGroup,
  ytext = "testylab",
  xtext = "testxlab",
  legendPos = c(0.1, 0.23),
  legendHeading = NULL,
  shownEffect = "main",
  effectLegend = FALSE,
  effectDescription = NULL,
  xLabelsOverwrite = NULL,
  useLatexMarkup = FALSE,
  numberColors = 6
)

plot_effect(
  data,
  x,
  y,
  fillColourGroup,
  ytext = "testylab",
  xtext = "testxlab",
  legendPos = c(0.1, 0.23),
  legendHeading = NULL,
  shownEffect = "main",
  effectLegend = FALSE,
  effectDescription = NULL,
  xLabelsOverwrite = NULL,
  useLatexMarkup = FALSE,
  numberColors = 6
)
```

Arguments

`data`	the data frame
`x`	factor shown on the x-axis
`y`	dependent variable
`fillColourGroup`	group to color
`ytext`	label for y-axis
`xtext`	label for x-axis
`legendPos`	position for legend
`legendHeading`	custom heading for legend

`shownEffect`	either "main" or "interaction"
`effectLegend`	TRUE: show legend for effect (Default: FALSE)
`effectDescription`	custom label for effect
`xLabelsOverwrite`	custom labels for x-axis
`useLatexMarkup`	use latex font and markup
`numberColors`	number of colors

Value

a plot

Examples

```
set.seed(123)
main_df <- data.frame(
  strategy    = factor(rep(c("A", "B"), each = 20)),
  Emotion     = factor(rep(c("Happy", "Sad"), times = 20)),
  trust_mean  = rnorm(40, mean = 5, sd = 1)
)

generateEffectPlot(
  data = main_df,
  x = "strategy",
  y = "trust_mean",
  fillColourGroup = "Emotion",
  ytext = "Trust",
  xtext = "Strategy",
  legendPos = c(0.1, 0.23)
)
```

`generateMoboPlot` *Generate a Multi-objective Optimization Plot*

Description

This function generates a multi-objective optimization plot using `ggplot2`. The plot visualizes the relationship between the `x` and `y` variables, grouping and coloring by a fill variable, with the option to customize legend position, labels, and annotation of sampling and optimization phases.

Usage

```
generateMoboPlot(
  data,
  x,
  y,
```

```
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  numberSamplingSteps = 5,
  labelPosFormulaY = "top",
  verticalLinePosY = 0.75
)

plot_mobo(
  data,
  x,
  y,
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  numberSamplingSteps = 5,
  labelPosFormulaY = "top",
  verticalLinePosY = 0.75
)
```

Arguments

`data`	A data frame containing the data to be plotted.
`x`	A string representing the column name in `data` to be used for the x-axis. Can be either numeric or factor.
`y`	A string representing the column name in `data` to be used for the y-axis. This should be a numeric variable.
`fillColourGroup`	A string representing the column name in `data` that defines the fill color grouping for the plot. Default is `"ConditionID"`.
`ytext`	A custom label for the y-axis. If not provided, the y-axis label will be the title-cased version of `y`.
`legendPos`	A numeric vector of length 2 specifying the position of the legend inside the plot. Default is `c(0.65, 0.85)`.
`numberSamplingSteps`	An integer specifying the number of initial sampling steps before the optimization phase begins. Default is 5.
`labelPosFormulaY`	A string specifying the vertical position of the polynomial equation label in the plot. Acceptable values are `"top"`, `"center"`, or `"bottom"`. Default is `"top"`.
`verticalLinePosY`	A numeric value of the y-coordinate where the "sampling" and "optimization" line should be drawn.

Value

A `ggplot` object representing the multi-objective optimization plot, ready to be rendered.

Examples

```
library(ggplot2)
library(ggpmisc)

# Example with numeric x-axis
df <- data.frame(
  x = 1:20,
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10)
)
generateMoboPlot(df, x = "x", y = "y")

# Example with factor x-axis
df <- data.frame(
  x = factor(rep(1:5, each = 4)),
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10)
)
generateMoboPlot(df, x = "x", y = "y", numberSamplingSteps = 3)
```

`generateMoboPlot2` *Generate a Multi-objective Optimization Plot*

Description

This function generates a multi-objective optimization plot using `ggplot2`. The plot visualizes the relationship between the `x` and `y` variables, grouping and coloring by a fill variable, with the option to customize legend position, labels, and annotation of sampling and optimization phases. Appropriate if you use https://github.com/Pascal-Jansen/Bayesian-Optimization-for-Unity in version 1.1.0 or higher.

Usage

```
generateMoboPlot2(
  data,
  x = "Iteration",
  y,
  phaseCol = "Phase",
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  labelPosFormulaY = "top",
  labelPosFormulaX = "left",
  horizontalLinePosY = 0.75,
  horizontalLineDistToText = 0.3,
  fillLabels = NULL,
  annotationTextSize = 5
)
```

```
plot_mobo2(
  data,
  x = "Iteration",
  y,
  phaseCol = "Phase",
  fillColourGroup = "ConditionID",
  ytext,
  legendPos = c(0.65, 0.85),
  labelPosFormulaY = "top",
  labelPosFormulaX = "left",
  horizontalLinePosY = 0.75,
  horizontalLineDistToText = 0.3,
  fillLabels = NULL,
  annotationTextSize = 5
)
```

Arguments

`data`	A data frame containing the data to be plotted.
`x`	A string representing the column name in `data` to be used for the x-axis. Can be either numeric or factor. Default is `"Iteration"`.
`y`	A string representing the column name in `data` to be used for the y-axis. This should be a numeric variable.
`phaseCol`	the name of the column for the color of the phase (sampling or optimization)
`fillColourGroup`	A string representing the column name in `data` that defines the fill color grouping for the plot. Default is `"ConditionID"`.
`ytext`	A custom label for the y-axis. If not provided, the y-axis label will be the title-cased version of `y`.
`legendPos`	A numeric vector of length 2 specifying the position of the legend inside the plot. Default is `c(0.65, 0.85)`.
`labelPosFormulaY`	A string specifying the vertical position of the polynomial equation label in the plot. Acceptable values are `"top"`, `"center"`, or `"bottom"`. Default is `"top"`.
`labelPosFormulaX`	A string specifying the position of the polynomial equation label in the plot. Acceptable values are `"left"`, `"center"`, or `"right"`. Default is `"left"`.
`horizontalLinePosY`	A numeric value of the y-coordinate where the "sampling" and "optimization" line should be drawn. Default is `0.75`
`horizontalLineDistToText`	A numeric value of the y-coordinate where the "sampling" and "optimization" text should be drawn below the line. Default is `0.3`
`fillLabels`	An optional named character vector mapping raw factor levels to display labels for the fill/colour legend (e.g. `c("value_only" = "Value Only", "llm_only" = "LLM Only")`). If `NULL` (default), the original factor levels are used as-is.

`annotationTextSize`
: numeric. The font size for embedded text annotations inside the plot (e.g., "Sampling", "Optimization" labels, and the regression equations). Default is `5.0`.

Value

A `ggplot` object representing the multi-objective optimization plot, ready to be rendered.

Examples

```
library(ggplot2)
library(ggpmisc)

# Example with numeric x-axis
df <- data.frame(
  x = 1:20,
  y = rnorm(20),
  ConditionID = rep(c("A", "B"), 10),
  Phase = rep(c("Sampling", "Optimization"), 10)
)
generateMoboPlot2(data = df, x = "x", y = "y")
```

`ggbetweenstatsWithPriorNormalityCheck`

Check the data's distribution. If non-normal, take the non-parametric variant of ggbetweenstats*. x and y have to be in parentheses, e.g., "ConditionID".*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggbetweenstats*. x and y have to be in parentheses, e.g., "ConditionID".

Usage

```
ggbetweenstatsWithPriorNormalityCheck(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)

plot_between_stats(
  data,
  x,
  y,
```

```
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)
```

Arguments

`data`	the data frame
`x`	the independent variable, most likely "ConditionID"
`y`	the dependent variable under investigation
`ylab`	label to be shown for the dependent variable
`xlabels`	labels to be used for the x-axis
`showPairwiseComp`	whether to show pairwise comparisons, TRUE as default
`plotType`	either "box", "violin", or "boxviolin" (default)

Value

A `ggplot` object produced by `ggstatsplot::ggbetweenstats`, which can be printed or further modified with +.

Examples

```
set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggbetweenstatsWithPriorNormalityCheck(
  data = main_df,
  x = "CondID",
  y = "tlx_mental", ylab = "Mental Demand",
  xlabels = labels_xlab,
  showPairwiseComp = TRUE
)
```

`ggbetweenstatsWithPriorNormalityCheckAsterisk` *Check the data's distribution. If non-normal, take the non-parametric variant of* ggbetweenstats. *x and y have to be in parentheses, e.g., "ConditionID".*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggbetweenstats*. x and y have to be in parentheses, e.g., "ConditionID".

Usage

```
ggbetweenstatsWithPriorNormalityCheckAsterisk(
  data,
  x,
  y,
  ylab,
  xlabels,
  plotType = "boxviolin"
)

plot_between_stats_asterisk(data, x, y, ylab, xlabels, plotType = "boxviolin")
```

Arguments

`data`	the data frame
`x`	the independent variable, most likely "ConditionID"
`y`	the dependent variable under investigation
`ylab`	label to be shown for the dependent variable
`xlabels`	labels to be used for the x-axis
`plotType`	either "box", "violin", or "boxviolin" (default)

Value

A `ggplot` object produced by `ggstatsplot::ggbetweenstats` with additional significance annotations, which can be printed or modified.

Examples

```
set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
```

```
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggbetweenstatsWithPriorNormalityCheckAsterisk(
  data = main_df,
  x = "CondID", y = "tlx_mental", ylab = "Mental Demand", xlabels = labels_xlab
)
```

`ggwithinstatsWithPriorNormalityCheck` *Check the data's distribution. If non-normal, take the non-parametric variant of* ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID".*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID".

Usage

```
ggwithinstatsWithPriorNormalityCheck(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)

plot_within_stats(
  data,
  x,
  y,
  ylab,
  xlabels = NULL,
  showPairwiseComp = TRUE,
  plotType = "boxviolin"
)
```

Arguments

`data`	the data frame
`x`	the independent variable, most likely "ConditionID"
`y`	the dependent variable under investigation
`ylab`	label to be shown for the dependent variable
`xlabels`	labels to be used for the x-axis
`showPairwiseComp`	whether to show pairwise comparisons, TRUE as default
`plotType`	either "box", "violin", or "boxviolin" (default)

Value

A `ggplot` object produced by `ggstatsplot::ggwithinstats` with additional significance annotations, which can be printed or modified.

Examples

```
#'   set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggwithinstatsWithPriorNormalityCheck(
  data = main_df,
  x = "CondID", y = "tlx_mental",
  ylab = "Mental Demand",
  xlabels = labels_xlab,
  showPairwiseComp = TRUE
)
```

`ggwithinstatsWithPriorNormalityCheckAsterisk`

Check the data's distribution. If non-normal, take the non-parametric variant of ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID". Add Asterisks instead of p-values.*

Description

Check the data's distribution. If non-normal, take the non-parametric variant of *ggwithinstats*. x and y have to be in parentheses, e.g., "ConditionID". Add Asterisks instead of p-values.

Usage

```
ggwithinstatsWithPriorNormalityCheckAsterisk(
  data,
  x,
  y,
  ylab,
  xlabels,
  plotType = "boxviolin"
)

plot_within_stats_asterisk(data, x, y, ylab, xlabels, plotType = "boxviolin")
```

Arguments

`data`	the data frame
`x`	the independent variable, most likely "ConditionID"
`y`	the dependent variable under investigation
`ylab`	label to be shown for the dependent variable
`xlabels`	labels to be used for the x-axis
`plotType`	either "box", "violin", or "boxviolin" (default)

Value

A `ggplot` object produced by `ggstatsplot::ggwithinstats` with additional significance annotations, which can be printed or modified.

Examples

```
set.seed(123)

# Toy within-subject style data
main_df <- data.frame(
  Participant = factor(rep(1:20, each = 3)),
  CondID      = factor(rep(c("A", "B", "C"), times = 20)),
  tlx_mental  = rnorm(60, mean = 50, sd = 10)
)

# Custom x-axis labels
labels_xlab <- c("Condition A", "Condition B", "Condition C")


ggwithinstatsWithPriorNormalityCheckAsterisk(
  data = main_df,
```

```
  x = "CondID", y = "tlx_mental",
  ylab = "Mental Demand", xlabels = labels_xlab
)
```

`latex_preamble` *LaTeX preamble required by the report functions*

Description

All report functions emit LaTeX text that relies on a small set of custom commands. This helper prints the complete set, ready to paste into a manuscript preamble, or writes it to a file that can be included with `\input{}` (or renamed to `.sty` and loaded via `\usepackage`).

Usage

```
latex_preamble(path = NULL)
```

Arguments

`path`	Optional path of a `.tex` file to write the definitions to.

Value

Invisibly returns the macro definitions as a character vector; the text is also emitted via `message()`.

Examples

```
latex_preamble()
```

`latexify_report` *Transform text from* `report::report()` *into LaTeX-friendly output.*

Description

This function transforms the text output from `report::report()` by performing several substitutions to prepare the text for LaTeX typesetting. In particular, it replaces instances of `R2`, `%`, and `~` with the corresponding LaTeX code. Additionally, it provides options to:

- Omit bullet items marked as "non-significant" (when `only_sig = TRUE`).
- Remove a concluding note about standardized parameters (when `remove_std = TRUE`).
- Wrap bullet items in a LaTeX `itemize` environment or leave them as plain text (controlled by `itemize`).

Usage

```
latexify_report(
  x,
  print_result = TRUE,
  only_sig = FALSE,
  remove_std = FALSE,
  itemize = TRUE
)
```

Arguments

`x`	Character vector or a single string containing the report text.
`print_result`	Logical. If `TRUE` (default), the formatted text is printed to the console.
`only_sig`	Logical. If `TRUE`, bullet items containing "non-significant" are omitted. Default is `FALSE`.
`remove_std`	Logical. If `TRUE`, the final standardized parameters note is removed. Default is `FALSE`.
`itemize`	Logical. If `TRUE` (default), bullet items are wrapped in a LaTeX `itemize` environment; otherwise the bullet markers are simply removed.

Value

A single string with the LaTeX-friendly formatted report text.

Examples

```
if (requireNamespace("report", quietly = TRUE)) {
  # Simple linear model on the iris dataset
  model <- stats::lm(
    Sepal.Length ~ Sepal.Width + Petal.Length,
    data = datasets::iris
  )

  # Format the report output, showing only significant items, removing the
  # standard note, and wrapping bullet items in an itemize environment.
  report_text <- try(report::report(model), silent = TRUE)
  if (!inherits(report_text, "try-error")) {
    latexify_report(
      report_text,
      only_sig = TRUE,
      remove_std = TRUE,
      itemize = TRUE
    )
  }
}
```

`n_fun` *Build a median/size label for plot annotations*

Description

Build a median/size label for plot annotations

Usage

```
n_fun(x)
```

Arguments

`x`	A numeric vector.

Value

A data frame with the median and label.

`na.zero` *Replace NA values with zero*

Description

Replace NA values with zero

Usage

```
na.zero(x)
```

Arguments

`x`	A vector.

Value

A vector with NAs replaced by zeros.

Examples

```
na.zero(c(NA, 1, NA, 2))
```

`normalize` *This function normalizes the values in a vector to the range [new_min, new_max] based on their original range [old_min, old_max].*

Description

This function normalizes the values in a vector to the range [new_min, new_max] based on their original range [old_min, old_max].

Usage

```
normalize(x_vector, old_min, old_max, new_min, new_max)
```

Arguments

`x_vector`	A numeric vector that you want to normalize.
`old_min`	The minimum value in the original scale of the data.
`old_max`	The maximum value in the original scale of the data.
`new_min`	The minimum value in the new scale to which you want to normalize the data.
`new_max`	The maximum value in the new scale to which you want to normalize the data.

Value

A numeric vector with the normalized values.

Examples

```
normalize(c(1, 2, 3, 4, 5), 1, 5, 0, 1)
```

`not_empty` *Ensure input is not empty*

Description

Stops execution if x is NULL, empty, or contains only NAs.

Usage

```
not_empty(x, msg = "Input must not be empty.")
```

Arguments

`x`	The object to check
`msg`	The error message to display

Value

Invisible TRUE if valid.

`not_in` *Negate `%in%` membership*

Description

Negate `%in%` membership

Usage

```
not_in(x, y)

x %!in% y
```

Arguments

`x`	Vector of values to test.
`y`	Vector of values to match against.

Value

Logical vector indicating non-membership.

`pathPrep` *Convert Windows paths to R-friendly format*

Description

Convert Windows paths to R-friendly format

Usage

```
pathPrep(path = "clipboard", read_fn = NULL, write_fn = NULL)
```

Arguments

`path`	Path to convert or the string "clipboard" to read from the clipboard.
`read_fn`	Optional custom function to read from the clipboard.
`write_fn`	Optional custom function to write to the clipboard.

Value

A normalized path string.

`remove_outliers_REI` *Flag suspicious survey responses via the Response Entropy Index (REI)*

Description

This function takes a data frame, optional header information, variables to consider, and a range for a Likert scale. It then calculates the Response Entropy Index (REI) and flags suspicious entries based on percentiles. Note that no rows are removed; entries are only flagged via the `Suspicious` column.

Usage

```
remove_outliers_REI(df, header = FALSE, variables = "", range = c(1, 5))
```

Arguments

`df`	Data frame containing the data.
`header`	Logical indicating if the data frame has a header. Defaults to FALSE.
`variables`	Character string specifying which variables to consider, separated by commas.
`range`	Numeric vector of length 2 specifying the range of the Likert scale (used to sanity-check the responses). Defaults to c(1, 5).

Details

Missing responses are ignored when tallying answers. Responses outside the declared Likert `range` trigger a warning (they often indicate mis-coded data) but are still included in the REI computation.

For more information on the REI method, refer to: Response Entropy Index Method

Value

A data frame with calculated REI, percentile, and a 'Suspicious' flag.

Examples

```
df <- data.frame(var1 = c(1, 2, 3), var2 = c(2, 3, 4))
result <- remove_outliers_REI(df, TRUE, "var1,var2", c(1, 5))
```

`report_all` *Analyze and report several dependent variables at once*

Description

Runs `analyze_and_report()` for each dependent variable (e.g., all questionnaire scales of a study) and additionally returns a summary table of the omnibus tests with Holm-adjusted p-values across the dependent variables, plus – when **patchwork** is installed – a combined figure.

Usage

```
report_all(
  data,
  dvs,
  iv,
  design = c("between", "within"),
  labels = NULL,
  xlabels = NULL,
  plotType = "boxviolin",
  sink_dir = NULL
)
```

Arguments

`data`	the data frame
`dvs`	character vector of dependent variable column names
`iv`	the independent variable (column name as string)
`design`	`"between"` (default) or `"within"`
`labels`	optional named character vector mapping a dv name to its axis label, e.g. `c(tlx_mental = "Mental Demand")`
`xlabels`	optional labels for the x-axis, passed to every plot
`plotType`	either "box", "violin", or "boxviolin" (default)
`sink_dir`	optional directory; each dv's sentences are written to `<sink_dir>/<dv>.tex` so a manuscript can `\input{}` them

Value

Invisibly returns a list with components `results` (named list of `analyze_and_report()` results), `summary` (data frame with one row per dv: method, statistic, p.value, and Holm-adjusted `p.holm`), and `combined_plot` (a patchwork figure, or `NULL` when patchwork is not installed).

Examples

```
out <- report_all(mtcars, dvs = c("mpg", "disp"), iv = "cyl")
out$summary
```

`reportART` *Generate the Latex-text based on the ARTool (see* https://github.com/mjskay/ARTool*). The ART result must be piped into an anova(). Only significant main and interaction effects are reported. P-values are rounded for the third digit. Attention: Effect sizes are not calculated! Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).*

Description

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\F}[3]{$F({#1},{#2})={#3}$}` `\newcommand{\p}{\textit{p=}}` `\newcommand{\pminor}{\text`

Usage

```
reportART(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_art(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

`model`	the model of the art
`dv`	the name of the dependent variable that should be reported
`write_to_clipboard`	whether to write to the clipboard
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE)) {
  set.seed(123)
```

```
  main_df <- data.frame(
    tlx_mental = stats::rnorm(80),
    Video      = factor(rep(c("A", "B"), each = 40)),
    gesture    = factor(rep(c("G1", "G2"), times = 40)),
    eHMI       = factor(rep(c("On", "Off"), times = 40)),
    UserID     = factor(rep(1:20, each = 4))
  )

  art_model <- ARTool::art(
    tlx_mental ~ Video * gesture * eHMI +
      Error(UserID / (gesture * eHMI)),
    data = main_df
  )

  model_anova <- stats::anova(art_model)
  reportART(model_anova, dv = "mental demand")
}
```

`reportArtCon` *Report significant ART contrasts (art.con) as LaTeX text*

Description

Companion to `reportDunnTest()` for aligned-rank-transform (ART) models. It extracts the significant pairwise comparisons produced by `ARTool::art.con()` (an **emmeans** contrast grid), computes the mean and standard deviation of the groups involved from the raw data, and prints LaTeX-formatted sentences.

Usage

```
reportArtCon(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  sink_to = NULL
)

report_art_con(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  sink_to = NULL
)
```

Arguments

`ac`	the contrast object returned by `ARTool::art.con()` (or its `summary()`)
`data`	the raw data frame used to fit the model
`iv`	independent variable (the contrasted factor)
`dv`	dependent variable
`paired`	whether to compute the rank-biserial effect size for paired (within-subjects) data. Defaults to `FALSE`. When `TRUE`, `id` is required.
`id`	the subject/pairing column, used only when `paired = TRUE`. Replicate trials per subject and condition are averaged before pairing.
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Details

The p-values are taken as-is from the contrast object, i.e. they are already adjusted by whatever `adjust` was passed to `art.con()` (e.g. `"holm"`). The effect size is the rank-biserial correlation computed from the raw data. ART is most often used for within-subjects designs; pass `paired = TRUE` together with `id` (the subject column) to obtain the paired rank-biserial effect size.

Attention: `ac` must be a pairwise contrast over a single factor `iv` (e.g. `art.con(model, ~ interaction_mode, adjust = "holm")`).

Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_` `\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE) &&
  requireNamespace("emmeans", quietly = TRUE)) {
  set.seed(123)
  n <- 20
  df <- data.frame(
    UserID = factor(rep(seq_len(n), times = 3)),
    mode   = factor(rep(c("Hand", "Eye", "Both"), each = n)),
    prime  = factor(rep(rep(c("A", "B"), each = n / 2), times = 3))
  )
  df$score <- as.numeric(df$mode) * 2 + stats::rnorm(nrow(df))

  m  <- ARTool::art(score ~ mode * prime + Error(UserID / mode), data = df)
  ac <- ARTool::art.con(m, ~ mode, adjust = "holm")
  reportArtCon(ac, data = df, iv = "mode", dv = "score", paired = TRUE, id = "UserID")
}
```

`reportArtConTable` *Report ART contrasts (art.con) as a LaTeX table. Customizable with sensible defaults. Companion to* `reportDunnTestTable()`.

Description

Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_`
`\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Usage

```
reportArtConTable(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  sink_to = NULL
)

report_art_con_table(
  ac,
  data,
  iv = "testiv",
  dv = "testdv",
  paired = FALSE,
  id = NULL,
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  sink_to = NULL
)
```

Arguments

`ac`	the contrast object returned by `ARTool::art.con()` (or its `summary()`)
`data`	the raw data frame used to fit the model
`iv`	independent variable (the contrasted factor)
`dv`	dependent variable

`paired`	whether to compute the rank-biserial effect size for paired (within-subjects) data. Defaults to `FALSE`. When `TRUE`, `id` is required.
`id`	the subject/pairing column, used only when `paired = TRUE`. Replicate trials per subject and condition are averaged before pairing.
`orderByP`	whether to order by the p value
`numberDigitsForPValue`	the number of digits to show
`latexSize`	which size for the text
`orderText`	whether to order the comparisons alphabetically; ignored when `orderByP = TRUE`
`sink_to`	optional path of a `.tex` file to write the table to, so a manuscript can `\input{}` it

Value

Invisibly returns the rendered LaTeX table as a string (or `NULL` when xtable is unavailable); the table is also printed.

Examples

```
if (requireNamespace("ARTool", quietly = TRUE) &&
  requireNamespace("emmeans", quietly = TRUE)) {
  set.seed(123)
  n <- 20
  df <- data.frame(
    UserID = factor(rep(seq_len(n), times = 3)),
    mode   = factor(rep(c("Hand", "Eye", "Both"), each = n)),
    prime  = factor(rep(rep(c("A", "B"), each = n / 2), times = 3))
  )
  df$score <- as.numeric(df$mode) * 2 + stats::rnorm(nrow(df))

  m  <- ARTool::art(score ~ mode * prime + Error(UserID / mode), data = df)
  ac <- ARTool::art.con(m, ~ mode, adjust = "holm")
  reportArtConTable(ac, data = df, iv = "mode", dv = "score", paired = TRUE, id = "UserID")
}
```

`reportDunnTest` *Report dunnTest as text. Required commands in LaTeX:* `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Description

Report dunnTest as text. Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Usage

```
reportDunnTest(d, data, iv = "testiv", dv = "testdv", sink_to = NULL)

report_dunn_test(d, data, iv = "testiv", dv = "testdv", sink_to = NULL)
```

Arguments

`d`	the dunn test object
`data`	the data frame
`iv`	independent variable
`dv`	dependent variable
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Examples

```
if (requireNamespace("FSA", quietly = TRUE)) {
  # Use built-in iris data
  data(iris)

  # Dunn test on Sepal.Length by Species
  d <- FSA::dunnTest(Sepal.Length ~ Species,
    data   = iris,
    method = "holm"
  )

  # Report the Dunn test
  reportDunnTest(d,
    data = iris,
    iv   = "Species",
    dv   = "Sepal.Length"
  )
}
```

`reportDunnTestTable` *report Dunn test as a table. Customizable with sensible defaults. Required commands in LaTeX:* `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Description

report Dunn test as a table. Customizable with sensible defaults. Required commands in LaTeX: `\newcommand{\padjminor}{\textit{p$_{adj}<$}}` `\newcommand{\padj}{\textit{p$_{adj}$=}}` `\newcommand{\rankbiserial}[1]{$r_{rb} = #1$}`

Usage

```
reportDunnTestTable(
  d = NULL,
  data,
  iv = "testiv",
  dv = "testdv",
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  sink_to = NULL
)

report_dunn_test_table(
  d = NULL,
  data,
  iv = "testiv",
  dv = "testdv",
  orderByP = FALSE,
  numberDigitsForPValue = 4,
  latexSize = "small",
  orderText = TRUE,
  sink_to = NULL
)
```

Arguments

`d`	the dunn test object
`data`	the data frame
`iv`	independent variable
`dv`	dependent variable
`orderByP`	whether to order by the p value
`numberDigitsForPValue`	the number of digits to show
`latexSize`	which size for the text
`orderText`	whether to order the comparisons alphabetically; ignored when `orderByP = TRUE`
`sink_to`	optional path of a `.tex` file to write the table to, so a manuscript can `\input{}` it

Value

Invisibly returns the rendered LaTeX table as a string (or `NULL` when xtable is unavailable); the table is also printed.

Examples

```
if (requireNamespace("FSA", quietly = TRUE)) {
  # Use built-in iris data
  data(iris)

  # Dunn test on Sepal.Length by Species
  d <- FSA::dunnTest(Sepal.Length ~ Species,
    data   = iris,
    method = "holm"
  )

  # Report the Dunn test
  reportDunnTestTable(d,
    data = iris,
    iv   = "Species",
    dv   = "Sepal.Length"
  )
}
```

`reportggstatsplot` *Report statistical details for ggstatsplot.*

Description

Report statistical details for ggstatsplot.

Usage

```
reportggstatsplot(
  p,
  iv = "independent",
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_ggstatsplot(
  p,
  iv = "independent",
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

`p`	the object returned by ggwithinstats or ggbetweenstats
`iv`	the independent variable
`dv`	the dependent variable
`write_to_clipboard`	whether to write to the clipboard
`sink_to`	optional path of a `.tex` file to write the sentence to, so a manuscript can `\input{}` it

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Examples

```
library(ggstatsplot)
library(dplyr)

# Generate a plot
plt <- ggbetweenstats(mtcars, am, mpg)

reportggstatsplot(plt, iv = "am", dv = "mpg")
```

`reportggstatsplotPostHoc`

Report significant post-hoc pairwise comparisons

Description

This function extracts significant pairwise comparisons from a `ggstatsplot` object, calculates the mean and standard deviation for the groups involved using the raw data, and prints LaTeX-formatted sentences reporting the results.

Usage

```
reportggstatsplotPostHoc(
  data,
  p,
  iv = "testiv",
  dv = "testdv",
  label_mappings = NULL,
  sink_to = NULL
)

report_ggstatsplot_posthoc(
```

```
  data,
  p,
  iv = "testiv",
  dv = "testdv",
  label_mappings = NULL,
  sink_to = NULL
)
```

Arguments

`data`	A data frame containing the raw data used to generate the plot.
`p`	A `ggstatsplot` object (e.g., returned by `ggbetweenstats`) containing the pairwise comparison statistics.
`iv`	Character string. The column name of the independent variable (grouping variable).
`dv`	Character string. The column name of the dependent variable.
`label_mappings`	Optional named list or vector. Used to rename factor levels in the output text (e.g., `list("old_name" = "New Label")`).
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

LaTeX Requirements

To easily copy and paste the results to your manuscript, the following commands (or similar) must be defined in your LaTeX preamble, as the function outputs commands taking arguments (e.g., `\m{value}`):

```
\newcommand{\m}[1]{\textit{M}=#1}
\newcommand{\sd}[1]{\textit{SD}=#1}
\newcommand{\padj}[1]{$p_{adj}=#1$}
\newcommand{\padjminor}[1]{$p_{adj}<#1$}
```

Examples

```
library(ggstatsplot)
library(dplyr)

# Generate a plot
plt <- ggbetweenstats(mtcars, am, mpg)

# Report stats
reportggstatsplotPostHoc(
  data = mtcars,
  p = plt,
  iv = "am",
```

```
  dv = "mpg",
  label_mappings = list("0" = "Automatic", "1" = "Manual")
)
```

---

reportMeanAndSD *Report the mean and standard deviation of a dependent variable for all levels of an independent variable rounded to the 2nd digit.*

Description

#' To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\m}{\textit{M=}}` `\newcommand{\sd}{\textit{SD=}}`

Usage

```
reportMeanAndSD(data, iv = "testiv", dv = "testdv", sink_to = NULL)

report_mean_sd(data, iv = "testiv", dv = "testdv", sink_to = NULL)
```

Arguments

`data`	the data frame
`iv`	the independent variable
`dv`	the dependent variable
`sink_to`	optional path of a `.tex` file to write the lines to, so a manuscript can `\input{}` them

Value

Invisibly returns the formatted lines as a character vector; the text is also emitted via `message()`.

Examples

```
example_data <- data.frame(Condition = rep(c("A", "B", "C"),
each = 10), TLX1 = stats::rnorm(30))

reportMeanAndSD(example_data, iv = "Condition", dv = "TLX1")
```

`reportNparLD` *Report the model produced by nparLD. The model provided must be the model generated by the command 'nparLD'* `nparLD` *(see* `https://CRAN.R-project.org/package=nparLD`*).*

Description

#' Only significant main and interaction effects are reported. P-values are rounded for the third digit and relative treatment effects (RTE) are included when available. Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).

Usage

```
reportNparLD(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)

report_nparld(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

`model`	the model
`dv`	the dependent variable
`write_to_clipboard`	whether to write to the clipboard
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Details

#' To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\F}{\textit{F=}}` `\newcommand{\df}{\textit{df=}}` `\newcommand{\p}{\textit{p=}}` `\newcommand{\pminor}{\textit{p$<$}}`

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Examples

```
if (requireNamespace("nparLD", quietly = TRUE)) {
  # Small toy data set for nparLD
  set.seed(123)
  example_data <- data.frame(
    Subject = factor(rep(1:10, each = 3)),
    Time    = factor(rep(c("T1", "T2", "T3"), times = 10)),
    TLX1    = stats::rnorm(30, mean = 50, sd = 10)
  )

  # Fit nparLD model
  model <- nparLD::nparLD(
    TLX1 ~ Time,
    data        = example_data,
    subject     = "Subject",
    description = FALSE
  )

  # Report the nparLD result
  reportNparLD(model, dv = "TLX1")
  }
```

`reportNPAV` *Generate the Latex-text based on the NPAV by Lüpsen (see https://www.uni-koeln.de/~{}luepsen/R/). Only significant main and interaction effects are reported. P-values are rounded for the third digit and partial eta squared values are provided when possible. Attention: the independent variables of the formula and the term specifying the participant must be factors (i.e., use as.factor()).*

Description

Deprecated: `reportNPAV()` will be removed in a future release. Use `reportART()` with ARTool instead.

Usage

```
reportNPAV(
  model,
  dv = "Testdependentvariable",
  write_to_clipboard = FALSE,
  sink_to = NULL
)
```

Arguments

`model`	the model of the np.anova

`dv`	the name of the dependent variable that should be reported
`write_to_clipboard`	whether to write to the clipboard
`sink_to`	optional path of a `.tex` file to write the sentences to, so a manuscript can `\input{}` them

Details

To easily copy and paste the results to your manuscript, the following commands must be defined in Latex: `\newcommand{\F}[3]{$F({#1},{#2})={#3}$}` `\newcommand{\p}{\textit{p=}}` `\newcommand{\pminor}{\text`

Value

Invisibly returns the reported sentence(s) as a character vector; the text is also emitted via `message()`.

Examples

```
model <- data.frame(
  Df = c(1, 1, 10),
  `F value` = c(6.12, 5.01, NA),
  `Pr(>F)` = c(0.033, 0.045, NA),
  check.names = FALSE
)
rownames(model) <- c("Video", "gesture:eHMI", "Residuals")
reportNPAV(model, dv = "mental workload")
```

`reshape_data` *Reshape Excel Data Based on Custom Markers and Include Custom ID Column*

Description

This function takes an Excel file with data in a wide format and transforms it to a long format. It includes a customizable "ID" column in the first position and repeats it for each slice. The function identifies sections of columns between markers that start with a user-defined string (default is "videoinfo") and appends those sections under the first section, aligning by column index.

Usage

```
reshape_data(
  input_filepath,
  sheetName = "Results",
  marker = "videoinfo",
  id_col = "ID",
  output_filepath
)
```

Arguments

`input_filepath`	String, the file path of the input Excel file.
`sheetName`	String, the name of the sheet to read from the Excel file. Default is "Results".
`marker`	String, the string that identifies the start of a new section of columns. Default is "videoinfo".
`id_col`	String, the name of the column to use as the ID column. Default is "ID".
`output_filepath`	String, the file path for the output Excel file.

Details

Relevant if you receive data in wide-format but cannot use built-in functionality due to naming (e.g., in LimeSurvey)

Value

None, writes the reshaped data to an Excel file specified by output_filepath.

Examples

```
if (requireNamespace(c("write_xlsx", "readxl"), quietly = TRUE)) {
  tmp_in  <- tempfile(fileext = ".xlsx")
  tmp_out <- tempfile(fileext = ".xlsx")

  # Minimal toy input that includes your required pieces:
  # an ID column and something that contains the marker value.
  toy <- data.frame(
    ID = c(1, 1, 2, 2),
    section = c("videoinfo", "videoinfo", "videoinfo", "videoinfo"),
    key = c("fps", "duration_s", "fps", "duration_s"),
    value = c(30, 12.3, 25, 9.8),
    stringsAsFactors = FALSE
  )

  writexl::write_xlsx(toy, tmp_in)

  reshape_data(
    input_filepath = tmp_in,
    marker = "videoinfo",
    id_col = "ID",
    output_filepath = tmp_out
  )

  out <- readxl::read_excel(tmp_out)
  print(out)
}
```

`rFromNPAV` *Calculation based on Rosenthal's formula (1994). N stands for the* number of measurements. *Necessary command:*

Description

Calculation based on Rosenthal's formula (1994). N stands for the *number of measurements*. Necessary command:

Usage

```
rFromNPAV(pvalue, N)
```

Arguments

`pvalue`	p value
`N`	number of measurements in the experiment

Value

Invisibly returns a list with components:

- `r`: effect size as a numeric scalar.
- `z`: corresponding z-statistic.
- `text`: LaTeX-formatted character string that is also sent to the console.

Examples

```
rFromNPAV(0.02, N = 180)
```

`rFromWilcox` *Calculation based on Rosenthal's formula (1994). N stands for the* number of measurements.

Description

Calculation based on Rosenthal's formula (1994). N stands for the *number of measurements*.

Usage

```
rFromWilcox(wilcoxModel, N)
```

Arguments

`wilcoxModel`	the Wilcox model
`N`	number of measurements in the experiment

Value

Invisibly returns a list with components:

- `r`: effect size as a numeric scalar.
- `z`: corresponding z-statistic.
- `text`: character string that is also sent to the console.

Examples

```
set.seed(1)
d <- data.frame(
  group = rep(c("A", "B"), each = 10),
  value = rnorm(20)
)
w <- stats::wilcox.test(value ~ group, data = d, exact = FALSE)
rFromWilcox(w, N = nrow(d))
```

`rFromWilcoxAdjusted` *rFromWilcoxAdjusted*

Description

rFromWilcoxAdjusted

Usage

```
rFromWilcoxAdjusted(wilcoxModel, N, adjustFactor)
```

Arguments

`wilcoxModel`	the Wilcox model
`N`	number of measurements in the experiment
`adjustFactor`	ad adjustment factor

Value

Invisibly returns a list with components:

- `r`: adjusted effect size as a numeric scalar.
- `z`: adjusted z-statistic.
- `text`: character string that is also sent to the console.

Examples

```
set.seed(1)
d <- data.frame(
  group = rep(c("A", "B"), each = 10),
  value = rnorm(20)
)
w <- stats::wilcox.test(value ~ group, data = d, exact = FALSE)
rFromWilcoxAdjusted(w, N = nrow(d), adjustFactor = 2)
```

`save_paper_figure` *Save a plot with publication-ready defaults*

Description

Saves a ggplot with sizes matching common two-column conference/journal layouts (e.g., ACM): a single-column figure is 3.33 in wide, a full-width figure 7 in. PDFs are rendered with `grDevices::cairo_pdf` so that fonts are embedded and unicode glyphs survive.

Usage

```
save_paper_figure(
  plot = ggplot2::last_plot(),
  filename,
  columns = 1,
  width = NULL,
  height = NULL,
  dpi = 300
)
```

Arguments

`plot`	The plot to save (defaults to the last plot displayed).
`filename`	Output path; the extension selects the device (`.pdf` is recommended for LaTeX).
`columns`	1 for a single-column figure, 2 for a full-width figure. Ignored when `width` is given.
`width`	Figure width in inches; overrides `columns`.
`height`	Figure height in inches. Defaults to 2/3 of the width.
`dpi`	Resolution for raster output. Default 300.

Value

Invisibly returns `filename`.

Examples

```
p <- ggplot2::ggplot(mtcars, ggplot2::aes(factor(cyl), mpg)) +
  ggplot2::geom_boxplot()
save_paper_figure(p, file.path(tempdir(), "cyl-mpg.pdf"), columns = 1)
```

`stat_sum_df` *Generating the sum and adding a crossbar.*

Description

Generating the sum and adding a crossbar.

Usage

```
stat_sum_df(fun, geom = "crossbar", ...)
```

Arguments

`fun`	function
`geom`	geom to be shown
`...`	Additional arguments passed to stat_summary

Value

A `ggplot2` layer that can be added to a ggplot object.

Examples

```
  # Simple summary function: use the mean as y, ymin, and ymax
  mean_fun <- function(x) {
    m <- mean(x, na.rm = TRUE)
    data.frame(y = m, ymin = m, ymax = m)
  }

  ggplot2::ggplot(mtcars, ggplot2::aes(x = factor(cyl), y = mpg)) +
    stat_sum_df(mean_fun)
```

Index