Package ‘rempsyc’

July 18, 2022

Title Convenience functions for psychology

Version 0.0.5.1

Description Convenience functions to make your workflow faster and easier. Easily customizable plots (via `ggplot2`), nice APA tables exportable to Word (via `flextable`), easily run statistical tests or check assumptions, and automatize various other tasks.

License GPL (>= 3) + file LICENSE

URL https://github.com/rempsyc/rempsyc

BugReports https://github.com/rempsyc/rempsyc/issues

Depends R (>= 2.10)

Imports boot,
car,
dplyr,
effectsize,
flextable,
ggplot2,
ggrepel,
ggsignif,
lmtest,
qqplotr,
rlang,
methods,
cli

Suggests see,
patchwork,
VennDiagram,
openxlsx,
bootES,
emmeans,
testthat (>= 3.0.0),
knitr,
rmarkdown

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.0

VignetteBuilder knitr

R topics documented:

`cormatrix_excel` *Easy export of correlation matrix to Excel*

Description

Easily output a correlation matrix and export it to Microsoft Excel, with the first row and column frozen, and correlation coefficients colour-coded based on their effect size (0.0-0.3: small (no colour); 0.3-0.6: medium (pink); 0.6-1.0: large (red)).

Usage

```
cormatrix_excel(
  data,
  filename = "mycormatrix",
  overwrite = TRUE,
  use = "pairwise.complete.obs"
)
```

Arguments

data	The data frame
filename	Desired filename (path can be added before hand but no need to specify extension).
overwrite	Whether to allow overwriting previous file.
use	How to handle NA (see ?cor for options).

Examples

```
## Not run:
# Basic example
cormatrix_excel(mtcars)

## End(Not run)
```

`find_mad` *Identify outliers based on 3 MAD*

Description

Identify outliers based on 3 median absolute deviations.

See: Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, *49*(4), 764–766. https://doi.org/10.1016/j.jesp.2013.03.013

Usage

```
find_mad(data, col.list, ID = NULL, criteria = 3)
```

Arguments

data	The data frame.
col.list	List of variables to check for outliers.
ID	ID variable if you would like the outliers to be identified as such.
criteria	How many MAD to use as threshold (similar to standard deviations)

Author(s)

Hugues Leduc, Charles-Étienne Lavoie, Rémi Thériault

Examples

```
find_mad(data = mtcars,
         col.list = names(mtcars),
         criteria = 3)

mtcars2 <- mtcars
mtcars2$car <- row.names(mtcars)
find_mad(data = mtcars2,
         col.list = names(mtcars),
         ID = "car",
         criteria = 3)
```

`format_value` *Easily format p or r values*

Description

Easily format p or r values. Note: converts to character class for use in figures or manuscripts to accommodate e.g., "< .001".

Usage

```
format_value(value, type = "d", ...)

format_p(p, precision = 0.001, prefix = NULL, suffix = NULL, sign = FALSE)

format_r(r, precision = 0.01)

format_d(d, precision = 0.01)
```

Arguments

`value`	Value to be formatted, when using the generic `format_value()`.
`type`	Specify r or p value.
`...`	To specify precision level, if necessary, when using the generic `format_value()`. Simply add the `precision` argument.
`p`	p-value to format.
`precision`	Level of precision desired, if necessary.
`prefix`	To add a prefix before the value.
`suffix`	To add a suffix after the value.
`sign`	Logical. Whether to add an equal sign for p-values higher or equal to .001.
`r`	r-value to format.
`d`	d-value to format.

Examples

```
format_value(0.00041231, "p")
format_value(0.00041231, "r")
format_p(0.0041231)
format_p(0.00041231)
format_r(0.41231)
format_r(0.041231)
```

nice_assumptions *Easy assumptions checks*

Description

Test linear regression assumptions easily with a nice summary table.

Usage

```
nice_assumptions(model, interpretation = TRUE)
```

Arguments

model	The lm object to be passed to the function.
interpretation	Whether to display the interpretation helper or not.

See Also

Other functions useful in assumption testing: nice_density, nice_normality, nice_qq, nice_varplot, nice_var. Tutorial: https://remi-theriault.com/blog_assumptions

Examples

```
# Create a regression model (using data available in R by default)
model <- lm(mpg ~ wt * cyl + gear, data = mtcars)
nice_assumptions(model)

# Multiple dependent variables at once
DV <- names(mtcars[-1])
formulas <- paste(DV, "~ mpg")
models.list <- lapply(X = formulas, FUN = lm, data = mtcars)
assumptions.table <- do.call("rbind", lapply(models.list, nice_assumptions,
                                             interpretation = FALSE))
assumptions.table
```

nice_density *Easy density plots*

Description

Make nice density plots easily.

Usage

```
nice_density(
  data,
  variable,
  group,
  colours,
  ytitle = "Density",
  xtitle = variable,
  groups.labels = NULL,
  grid = TRUE,
  shapiro = FALSE,
  title = variable,
  histogram = FALSE
)
```

Arguments

`data`	The data frame
`variable`	The dependent variable to be plotted.
`group`	The group by which to plot the variable.
`colours`	Desired colours for the plot, if desired.
`ytitle`	An optional y-axis label, if desired.
`xtitle`	An optional x-axis label, if desired.
`groups.labels`	The groups.labels (might rename to `xlabels` for consistency with other functions)
`grid`	Logical, whether to keep the default background grid or not. APA style suggests not using a grid in the background, though in this case some may find it useful to more easily estimate the slopes of the different groups.
`shapiro`	Logical, whether to include the p-value from the Shapiro-Wilk test on the plot.
`title`	The desired title of the plot. Can be put to `NULL` to remove.
`histogram`	Logical, whether to add an histogram on top of the density plot.

See Also

Other functions useful in assumption testing: `nice_assumptions`, `nice_normality`, `nice_qq`, `nice_varplot`, `nice_var`. Tutorial: `https://remi-theriault.com/blog_assumptions`

Examples

```
# Make the basic plot
nice_density(data = iris,
             variable = "Sepal.Length",
             group = "Species")

# Further customization
nice_density(data = iris,
             variable = "Sepal.Length",
             group = "Species",
             colours = c("#00BA38", "#619CFF", "#F8766D"),
             xtitle = "Sepal Length",
             ytitle = "Density (vs. Normal Distribution)",
             groups.labels = c("(a) Setosa",
                               "(b) Versicolor",
                               "(c) Virginica"),
             grid = FALSE,
             shapiro = TRUE,
             title = "Density (Sepal Length)")
```

nice_lm — *Nice formatting of lm models*

Description

Formats output of `lm` model object for a publication-ready format.

Note: this function uses the `modelEffectSizes` function from the `lmSupport` package to get the sr2 effect sizes.

Usage

```
nice_lm(model, b.label = "b", mod.id = TRUE, ...)
```

Arguments

`model`	The model to be formatted.
`b.label`	What to rename the default "b" column (e.g., to capital B if using standardized data for it to be converted to the Greek beta symbol in the `nice_table` function).
`mod.id`	Logical. Whether to display the model number, when there is more than one model.
`...`	Further arguments to be passed to the `lm` function for the models.

See Also

Checking simple slopes after testing for moderation: `nice_lm_slopes`, `nice_mod`, `nice_slopes`. Tutorial: `https://remi-theriault.com/blog_moderation`

Examples

```
# Make and format model
model <- lm(mpg ~ cyl + wt * hp, mtcars)
nice_lm(model)

# Make and format multiple models
model2 <- lm(qsec ~ disp + drat * carb, mtcars)
my.models <- list(model, model2)
nice_lm(my.models)
```

nice_lm_slopes — *Nice formatting of simple slopes for lm models*

Description

Extracts simple slopes from `lm` model object and format for a publication-ready format.

Note: this function uses the `modelEffectSizes` function from the `lmSupport` package to get the sr2 effect sizes.

Usage

```
nice_lm_slopes(model, predictor, moderator, b.label = "b", mod.id = TRUE, ...)
```

Arguments

`model`	The model to be formatted.
`predictor`	The independent variable.
`moderator`	The moderating variable.
`b.label`	What to rename the default "b" column (e.g., to capital B if using standardized data for it to be converted to the Greek beta symbol in the `nice_table` function).
`mod.id`	Logical. Whether to display the model number, when there is more than one model.
`...`	Further arguments to be passed to the `lm` function for the models.

See Also

Checking for moderation before checking simple slopes: `nice_lm`, `nice_mod`, `nice_slopes`. Tutorial: `https://remi-theriault.com/blog_moderation`

Examples

```
# Make and format model
model <- lm(mpg ~ gear * wt, mtcars)
nice_lm_slopes(model, predictor = "gear", moderator = "wt")

# Make and format multiple models
model2 <- lm(qsec ~ gear * wt, mtcars)
my.models <- list(model, model2)
nice_lm_slopes(my.models, predictor = "gear", moderator = "wt")
```

`nice_mod`	*Easy moderations*

Description

Easily compute moderation analyses, with effect sizes, and format in publication-ready format.

Note: this function uses the `modelEffectSizes` function from the `lmSupport` package to get the sr2 effect sizes.

Usage

```
nice_mod(
  data,
  response,
  predictor,
  moderator,
  moderator2 = NULL,
  covariates = NULL,
  b.label = "b",
  mod.id = TRUE,
  ...
)
```

Arguments

`data`	The data frame
`response`	The dependent variable.
`predictor`	The independent variable.
`moderator`	The moderating variable.
`moderator2`	The second moderating variable, if applicable.
`covariates`	The desired covariates in the model.
`b.label`	What to rename the default "b" column (e.g., to capital B if using standardized data for it to be converted to the Greek beta symbol in the `nice_table` function).
`mod.id`	Logical. Whether to display the model number, when there is more than one model.
`...`	Further arguments to be passed to the `lm` function for the models.

See Also

Checking simple slopes after testing for moderation: `nice_slopes`, `nice_lm`, `nice_lm_slopes`. Tutorial: `https://remi-theriault.com/blog_moderation`

Examples

```
# Make the basic table
nice_mod(data = mtcars,
         response = "mpg",
         predictor = "gear",
         moderator = "wt")
```

```
# Multiple dependent variables at once
nice_mod(data = mtcars,
         response = c("mpg", "disp", "hp"),
         predictor = "gear",
         moderator = "wt")

# Add covariates
nice_mod(data = mtcars,
         response = "mpg",
         predictor = "gear",
         moderator = "wt",
         covariates = c("am", "vs"))

# Three-way interaction
nice_mod(data = mtcars,
         response = "mpg",
         predictor = "gear",
         moderator = "wt",
         moderator2 = "am")
```

`nice_na` *Report missing values according to guidelines*

Description

Nicely reports NA values according to existing guidelines. This function reports both absolute and percentage values of specified column lists. Some authors recommend reporting item-level missing item per scale, as well as participant's maximum number of missing items by scale. For example, Parent (2013) writes:

I recommend that authors (a) state their tolerance level for missing data by scale or subscale (e.g., "We calculated means for all subscales on which participants gave at least 75% complete data") and then (b) report the individual missingness rates by scale per data point (i.e., the number of missing values out of all data points on that scale for all participants) and the maximum by participant (e.g., "For Attachment Anxiety, a total of 4 missing data points out of 100 were observed, with no participant missing more than a single data point").

See: Parent, M. C. (2013). Handling item-level missing data: Simpler is just as good. The *Counseling Psychologist*, *41*(4), 568-600. https://doi.org/10.1177%2F0011000012445176

Usage

```
nice_na(data, vars, scales)
```

Arguments

`data`	The data frame.
`vars`	Variable (or lists of variables) to check for NAs.
`scales`	The scale names to check for NAs (single character string).

Examples

```
# Use whole data frame
nice_na(airquality)

# Use selected columns
nice_na(airquality,
          vars = list(c("Ozone", "Solar.R", "Wind"),
                      c("Temp", "Month", "Day")))

# If the questionnaire items start with the same name, e.g.,
set.seed(15)
fun <- function() {sample(c(NA, 1:10), replace = TRUE)}
df <- data.frame(scale1_Q1 = fun(), scale1_Q2 = fun(), scale1_Q3 = fun(),
                 scale2_Q1 = fun(), scale2_Q2 = fun(), scale2_Q3 = fun(),
                 scale3_Q1 = fun(), scale3_Q2 = fun(), scale3_Q3 = fun())
# One can list the scale names directly:
nice_na(df, scales = c("scale1", "scale2", "scale3"))
```

`nice_normality` *Easy normality check per group*

Description

Easily make nice per-group density and QQ plots through a wrapper around the `ggplot2` and `qqplotr` packages.

Usage

```
nice_normality(
  data,
  variable,
  group,
  colours,
  groups.labels,
  grid = TRUE,
  shapiro = FALSE,
  title = NULL,
  histogram = FALSE,
  ...
)
```

Arguments

`data`	The data frame.
`variable`	The dependent variable to be plotted.
`group`	The group by which to plot the variable.
`colours`	Desired colours for the plot, if desired.
`groups.labels`	How to label the groups.

grid	Logical, whether to keep the default background grid or not. APA style suggests not using a grid in the background, though in this case some may find it useful to more easily estimate the slopes of the different groups.
shapiro	Logical, whether to include the p-value from the Shapiro-Wilk test on the plot.
title	An optional title, if desired.
histogram	Logical, whether to add an histogram on top of the density plot.
...	Further arguments from `nice_qq()` and `nice_density()` to be passed to `nice_normality()`

See Also

Other functions useful in assumption testing: `nice_assumptions`, `nice_density`, `nice_qq`, `nice_var`, `nice_varplot`. Tutorial: `https://remi-theriault.com/blog_assumptions`

Examples

```
# Make the basic plot
nice_normality(data = iris,
               variable = "Sepal.Length",
               group = "Species")

# Further customization
nice_normality(data = iris,
               variable = "Sepal.Length",
               group = "Species",
               colours = c("#00BA38",
                           "#619CFF",
                           "#F8766D"),
               groups.labels = c("(a) Setosa",
                                 "(b) Versicolor",
                                 "(c) Virginica"),
               grid = FALSE,
               shapiro = TRUE)
```

---

nice_qq *Easy QQ plots per group*

Description

Easily make nice per-group QQ plots through a wrapper around the `ggplot2` and `qqplotr` packages.

Usage

```
nice_qq(
  data,
  variable,
  group,
  colours,
  groups.labels = NULL,
  grid = TRUE,
  shapiro = FALSE,
  title = variable
)
```

Arguments

`data`	The data frame.
`variable`	The dependent variable to be plotted.
`group`	The group by which to plot the variable.
`colours`	Desired colours for the plot, if desired.
`groups.labels`	How to label the groups.
`grid`	Logical, whether to keep the default background grid or not. APA style suggests not using a grid in the background, though in this case some may find it useful to more easily estimate the slopes of the different groups.
`shapiro`	Logical, whether to include the p-value from the Shapiro-Wilk test on the plot.
`title`	An optional title, if desired.

See Also

Other functions useful in assumption testing: `nice_assumptions`, `nice_density`, `nice_normality`, `nice_var`, `nice_varplot`. Tutorial: `https://remi-theriault.com/blog_assumptions`

Examples

```
# Make the basic plot
nice_qq(data = iris,
        variable = "Sepal.Length",
        group = "Species")

# Further customization
nice_qq(data = iris,
        variable = "Sepal.Length",
        group = "Species",
        colours = c("#00BA38", "#619CFF", "#F8766D"),
        groups.labels = c("(a) Setosa", "(b) Versicolor", "(c) Virginica"),
        grid = FALSE,
        shapiro = TRUE,
        title = NULL)
```

`nice_randomize` *Easily randomization*

Description

Randomize easily with different designs.

Usage

```
nice_randomize(
  design = "between",
  Ncondition = 3,
  n = 9,
  condition.names = c("a", "b", "c"),
  col.names = c("id", "Condition")
)
```

Arguments

`design`	The design: either between-subject (different groups) or within-subject (repeated-measures on same people).
`Ncondition`	The number of conditions you want to randomize.
`n`	The desired sample size. Note that it needs to be a multiple of your number of groups if you are usingbetween.
`condition.names`	The names of the randomized conditions.
`col.names`	The desired additional column names for a runsheet.

See Also

Tutorial: https://remi-theriault.com/blog_randomize

Examples

```
# Specify design, number of conditions, number of
# participants, and names of conditions:
nice_randomize(design = "between", Ncondition = 4, n = 8,
               condition.names = c("BP","CX","PZ","ZL"))

# Within-Group Design
nice_randomize(design = "within", Ncondition = 4, n = 6,
               condition.names = c("SV","AV","ST","AT"))

# Make a quick runsheet
randomized <- nice_randomize(design = "within", Ncondition = 4, n = 128,
                             condition.names = c("SV","AV","ST","AT"),
                             col.names = c("id", "Condition", "Date/Time",
                             "SONA ID", "Age/Gd.", "Handedness",
                             "Tester", "Notes"))
head(randomized)
```

`nice_reverse` *Easily recode scores*

Description

Easily recode scores (reverse-score), typically for questionnaire answers.

Usage

```
nice_reverse(x, max, min = 1, warning = TRUE)
```

Arguments

`x`	The score to reverse.
`max`	The maximum score on the scale.
`min`	The miminum score on the scale (optional unless it isn't 1).
`warning`	Logical. Whether to show the warning about the minimum not being 1.

Examples

```
# Reverse score of 5 with a maximum score of 5
nice_reverse(5, 5)

# Reverse several scores at once
nice_reverse(1:5, 5)

# Reverse scores with maximum = 4 and minimum = 0
nice_reverse(1:4, 4, min = 0)

# Reverse scores with maximum = 3 and minimum = -3
nice_reverse(-3:3, 3, min = -3)
```

`nice_scatter` *Easy scatter plots*

Description

Make nice scatter plots easily.

Usage

```
nice_scatter(
  data,
  predictor,
  response,
  xtitle = ggplot2::waiver(),
  ytitle = ggplot2::waiver(),
  has.points = TRUE,
  has.jitter = FALSE,
  alpha = 0.7,
  has.line = TRUE,
  has.confband = FALSE,
  has.fullrange = FALSE,
  has.linetype = FALSE,
  has.shape = FALSE,
  xmin,
  xmax,
  xby = 1,
  ymin,
  ymax,
  yby = 1,
  has.legend = FALSE,
  legend.title = "",
  group = NULL,
  colours = "#619CFF",
  groups.order = NULL,
  groups.labels = NULL,
  groups.alpha = NULL,
  has.r = FALSE,
  r.x = Inf,
  r.y = -Inf,
  has.p = FALSE,
  p.x = Inf,
  p.y = -Inf
)
```

Arguments

`data`	The data frame.
`predictor`	The independent variable to be plotted.
`response`	The dependent variable to be plotted.
`xtitle`	An optional y-axis label, if desired.
`ytitle`	An optional x-axis label, if desired.
`has.points`	Whether to plot the individual observations or not.
`has.jitter`	Alternative to `has.points`. "Jitters" the observations to avoid overlap (overplotting). Use one or the other, not both.
`alpha`	The desired level of transparency.
`has.line`	Whether to plot the regression line(s).
`has.confband`	Logical. Whether to display the confidence band around the slope.
`has.fullrange`	Logical. Whether to extend the slope beyond the range of observations.
`has.linetype`	Logical. Whether to change line types as a function of group.
`has.shape`	Logical. Whether to change shape of observations as a function of group.
`xmin`	The minimum score on the x-axis scale.
`xmax`	The maximum score on the x-axis scale.
`xby`	How much to increase on each "tick" on the x-axis scale.
`ymin`	The minimum score on the y-axis scale.
`ymax`	The maximum score on the y-axis scale.
`yby`	How much to increase on each "tick" on the y-axis scale.
`has.legend`	Logical. Whether to display the legend or not.
`legend.title`	The desired legend title.
`group`	The group by which to plot the variable
`colours`	Desired colours for the plot, if desired.
`groups.order`	Specifies the desired display order of the groups.
`groups.labels`	Changes groups names (labels). Note: This applies after changing order of level.
`groups.alpha`	The manually specified transparency desired for the groups slopes. Use only when plotting groups separately.
`has.r`	Whether to display the correlation coefficient, the r-value.
`r.x`	The x-axis coordinates for the r-value.
`r.y`	The y-axis coordinates for the r-value.
`has.p`	Whether to display the p-value.
`p.x`	The x-axis coordinates for the p-value.
`p.y`	The y-axis coordinates for the p-value.

See Also

Visualize group differences via violin plots: `nice_violin`. Tutorial: `https://remi-theriault.com/blog_scatter`

Examples

```
# Make the basic plot
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg")

## Not run:
# Save a high-resolution image file to specified directory
ggsave('nicescatterplothere.pdf', width = 7, height = 7, unit = 'in',
       dpi = 300, path = "/") # change for your own desired path

## End(Not run)

# Change x- and y- axis labels
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             ytitle = "Miles/(US) gallon",
             xtitle = "Weight (1000 lbs)")

# Have points "jittered"
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             has.jitter = TRUE)

# Change the transparency of the points
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             alpha = 1)

# Remove points
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             has.points = FALSE,
             has.jitter = FALSE)

# Add confidence band
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             has.confband = TRUE)

# Set x- and y- scales manually
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             xmin = 1,
             xmax = 6,
             xby = 1,
             ymin = 10,
             ymax = 35,
             yby = 5)

# Change plot colour
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             colours = "blueviolet")

# Add correlation coefficient to plot and p-value
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             has.r = TRUE,
             has.p = TRUE)

# Change location of correlation coefficient or p-value
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             has.r = TRUE,
             r.x = 4,
             r.y = 25,
             has.p = TRUE,
             p.x = 5,
             p.y = 20)

# Plot by group
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl")

# Use full range on the slope/confidence band
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.fullrange = TRUE)

# Remove lines
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.line = FALSE)

# Add a legend
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.legend = TRUE)

# Change order of labels on the legend
```

```
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.legend = TRUE,
             groups.order = c(8,4,6))

# Change legend labels
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.legend = TRUE,
             groups.labels = c("Weak","Average","Powerful"))
# Warning: This applies after changing order of level

# Add a title to legend
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.legend = TRUE,
             legend.title = "cylinders")

# Plot by group + manually specify colours
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             colours = c("burlywood", "darkgoldenrod", "chocolate"))

# Plot by group + use different line types for each group
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.linetype = TRUE)

# Plot by group + use different point shapes for each group
nice_scatter(data = mtcars,
             predictor = "wt",
             response = "mpg",
             group = "cyl",
             has.shape = TRUE)
```

`nice_slopes` *Easy simple slopes*

Description

Easily compute simple slopes in moderation analysis, with effect sizes, and format in publication-ready format.

Note: this function uses the `modelEffectSizes` function from the `lmSupport` package to get the sr2 effect sizes.

Usage

```
nice_slopes(
  data,
  response,
  predictor,
  moderator,
  moderator2 = NULL,
  covariates = NULL,
  b.label,
  mod.id = TRUE,
  ...
)
```

Arguments

`data`	The data frame
`response`	The dependent variable.
`predictor`	The independent variable
`moderator`	The moderating variable.
`moderator2`	The second moderating variable, if applicable.
`covariates`	The desired covariates in the model.
`b.label`	What to rename the default "b" column (e.g., to capital B if using standardized data for it to be converted to the Greek beta symbol in the `nice_table` function).
`mod.id`	Logical. Whether to display the model number, when there is more than one model.
`...`	Further arguments to be passed to the `lm` function for the models.

See Also

Checking for moderation before checking simple slopes: `nice_mod`, `nice_lm`, `nice_lm_slopes`. Tutorial: `https://remi-theriault.com/blog_moderation`

Examples

```
# Make the basic table
nice_slopes(data = mtcars,
            response = "mpg",
            predictor = "gear",
            moderator = "wt")

# Multiple dependent variables at once
nice_slopes(data = mtcars,
            response = c("mpg", "disp", "hp"),
            predictor = "gear",
            moderator = "wt")

# Add covariates
nice_slopes(data = mtcars,
            response = "mpg",
            predictor = "gear",
            moderator = "wt",
            covariates = c("am", "vs"))
```

```
# Three-way interaction (continuous moderator and binary
# second moderator required)
nice_slopes(data = mtcars,
            response = "mpg",
            predictor = "gear",
            moderator = "wt",
            moderator2 = "am")
```

nice_table *Easily make nice APA tables*

Description

Make nice APA tables easily through a wrapper around the `flextable` package with sensical defaults and automatic formatting features.

Usage

```
nice_table(
  data,
  highlight = FALSE,
  italics,
  col.format.p,
  col.format.r,
  format.custom,
  col.format.custom,
  width = 1,
  broom = "",
  report = "",
  short = FALSE
)
```

Arguments

`data`	The data frame, to be converted to a flextable. The data frame cannot have duplicate column names.
`highlight`	Highlight rows with statistically significant results? Requires a column named "p" containing p-values. Can either accept logical (TRUE/FALSE) OR a numeric value for a custom critical p-value threshold (e.g., 0.10 or 0.001).
`italics`	Which columns headers should be italic? Useful for column names that should be italic but that are not picked up automatically by the function. Select with numerical range, e.g., 1:3.
`col.format.p`	Applies p-value formatting to columns that cannot be named "p" (for example for a data frame full of p-values, also because it is not possible to have more than one column named "p"). Select with numerical range, e.g., 1:3.
`col.format.r`	Applies r-value formatting to columns that cannot be named "r" (for example for a data frame full of r-values, also because it is not possible to have more than one column named "r"). Select with numerical range, e.g., 1:3.

`format.custom`	Applies custom formatting to columns selected via the `col.format.custom` argument. This is useful if one wants custom formatting other than for p- or r-values. It can also be used to transform (e.g., multiply) certain values or print a specific symbol along the values for instance.
`col.format.custom`	Which columns to apply the custom function to. Select with numerical range, e.g., 1:3.
`width`	Width of the table, in percentage of the total width, when exported e.g., to Word.
`broom`	If providing a tidy table produced with the `broom` package, which model type to use if one wants automatic formatting (options are "t.test", "lm", "cor.test", and "wilcox.test").
`report`	If providing an object produced with the `report` package, which model type to use if one wants automatic formatting (options are "t.test", "lm", and "cor.test").
`short`	Logical. Whether to return an abbreviated version of the tables made by the `report` package.

See Also

Tutorial: https://remi-theriault.com/blog_table

Examples

```
# Make the basic table
my_table <- nice_table(head(mtcars))
my_table

## Not run:
# Save table to word
save_as_docx(my_table, path = "nicetablehere.docx")

## End(Not run)

# Publication-ready tables
mtcars.std <- lapply(mtcars, scale)
model <- lm(mpg ~ cyl + wt * hp, mtcars.std)
stats.table <- as.data.frame(summary(model)$coefficients)
CI <- confint(model)
stats.table <- cbind(row.names(stats.table),
                     stats.table, CI)
names(stats.table) <- c("Term", "B", "SE", "t", "p",
                        "CI_lower", "CI_upper")
nice_table(stats.table, highlight = TRUE)

# Test different column names
test <- head(mtcars)
names(test) <- c("dR", "N", "M", "SD", "b", "np2",
                 "ges", "p", "r", "R2", "sr2")
test[, 10:11] <- test[, 10:11]/10
nice_table(test)

# Custom cell formatting (such as p or r)
nice_table(test[8:11], col.format.p = 2:4, highlight = .001)

nice_table(test[8:11], col.format.r = 1:4)
```

```
# Apply custom functions to cells
fun <- function(x) {x+11.1}
nice_table(test[8:11], col.format.custom = 2:4, format.custom = "fun")

fun <- function(x) {paste("x", x)}
nice_table(test[8:11], col.format.custom = 2:4, format.custom = "fun")
```

nice_t_test *Easy t-tests*

Description

Easily compute t-test analyses, with effect sizes, and format in publication-ready format. The 95% confidence interval is for the effect size, Cohen's d, both provided by the `effectsize` package.

This function relies on the base R `t.test` function, which uses the Welch t-test per default (see why here: https://daniellakens.blogspot.com/2015/01/always-use-welchs-t-test-instead-of.html). To use the Student t-test, simply add the following argument: `var.equal = TRUE`.

Usage

```
nice_t_test(
  data,
  response,
  group = NULL,
  correction = "none",
  warning = TRUE,
  ...
)
```

Arguments

`data`	The data frame.
`response`	The dependent variable.
`group`	The group for the comparison.
`correction`	What correction for multiple comparison to apply, if any. Default is "none" and the only other option (for now) is "bonferroni".
`warning`	Whether to display the Welch test warning or not.
`...`	Further arguments to be passed to the `t.test` function (e.g., to use Student instead of Welch test, to change from two-tail to one-tail, or to do a paired-sample t-test instead of independent samples).

See Also

Tutorial: https://remi-theriault.com/blog_t-test

Examples

```
# Make the basic table
nice_t_test(data = mtcars,
            response = "mpg",
            group = "am")

# Multiple dependent variables at once
nice_t_test(data = mtcars,
            response = names(mtcars)[1:7],
            group = "am")

# Can be passed some of the regular arguments
# of base `t.test()`

# Student t-test (instead of Welch)
nice_t_test(data = mtcars,
            response = "mpg",
            group = "am",
            var.equal = TRUE)

# One-sided instead of two-sided
nice_t_test(data = mtcars,
            response = "mpg",
            group = "am",
            alternative = "less")

# One-sample t-test
nice_t_test(data = mtcars,
            response = "mpg",
            mu = 10)

# Paired t-test instead of independent samples
nice_t_test(data = ToothGrowth,
            response = "len",
            group = "supp",
            paired = TRUE)
# Make sure cases appear in the same order for
# both levels of the grouping factor
```

`nice_var` *Obtain variance per group*

Description

Obtain variance per group as well as check for the rule of thumb of one group having variance four times bigger than any of the other groups. Variance ratio is calculated as Max / Min.

Usage

```
nice_var(data, variable, group, criteria = 4)
```

Arguments

`data`	The data frame
`variable`	The dependent variable to be plotted.
`group`	The group by which to plot the variable.
`criteria`	Desired threshold if one wants something different than four times the variance.

See Also

Other functions useful in assumption testing: `nice_assumptions`, `nice_density`, `nice_normality`, `nice_qq`, `nice_varplot`. Tutorial: `https://remi-theriault.com/blog_assumptions`

Examples

```
# Make the basic table
nice_var(data = iris,
         variable = "Sepal.Length",
         group = "Species")

# Try on multiple variables
DV <- names(iris[1:4])
var.table <- do.call("rbind", lapply(DV, nice_var,
                     data = iris, group = "Species"))
var.table
```

`nice_varplot` *Attempt to visualize variance per group*

Description

Attempt to visualize variance per group.

Usage

```
nice_varplot(
  data,
  variable,
  group,
  colours,
  groups.labels,
  grid = TRUE,
  shapiro = FALSE,
  ytitle = ggplot2::waiver()
)
```

Arguments

`data`	The data frame
`variable`	The dependent variable to be plotted.
`group`	The group by which to plot the variable.

`colours`	Desired colours for the plot, if desired.
`groups.labels`	How to label the groups.
`grid`	Logical, whether to keep the default background grid or not. APA style suggests not using a grid in the background, though in this case some may find it useful to more easily estimate the slopes of the different groups.
`shapiro`	Logical, whether to include the p-value from the Shapiro-Wilk test on the plot.
`ytitle`	An optional y-axis label, if desired.

See Also

Other functions useful in assumption testing: `nice_assumptions`, `nice_density`, `nice_normality`, `nice_qq`, `nice_var`. Tutorial: `https://remi-theriault.com/blog_assumptions`

Examples

```
# Make the basic plot
nice_varplot(data = iris,
             variable = "Sepal.Length",
             group = "Species")

# Further customization
nice_varplot(data = iris,
             variable = "Sepal.Length",
             group = "Species",
             colours = c("#00BA38",
                         "#619CFF",
                         "#F8766D"),
             ytitle = "Sepal Length",
             groups.labels = c("(a) Setosa",
                               "(b) Versicolor",
                               "(c) Virginica"))
```

`nice_violin` *Easy violin plots*

Description

Make nice violin plots easily with 95% bootstrapped confidence intervals.

Usage

```
nice_violin(
  data,
  group,
  response,
  boot = TRUE,
  bootstraps = 2000,
  colours,
  xlabels = NULL,
  ytitle = ggplot2::waiver(),
  xtitle = NULL,
  has.ylabels = TRUE,
  has.xlabels = TRUE,
  comp1 = 1,
  comp2 = 2,
  signif_annotation = NULL,
  signif_yposition = NULL,
  signif_xmin = NULL,
  signif_xmax = NULL,
  ymin,
  ymax,
  yby = 1,
  CIcap.width = 0.1,
  obs = FALSE,
  alpha = 1,
  border.colour = "black",
  border.size = 2,
  has.d = FALSE,
  d.x = mean(c(comp1, comp2)) * 1.1,
  d.y = mean(data[[response]]) * 1.3
)
```

Arguments

`data`	The data frame.
`group`	The group by which to plot the variable.
`response`	The dependent variable to be plotted.
`boot`	Logical, whether to use bootstrapping or not.
`bootstraps`	How many bootstraps to use.
`colours`	Desired colours for the plot, if desired.
`xlabels`	The individual group labels on the x-axis.
`ytitle`	An optional y-axis label, if desired.
`xtitle`	An optional x-axis label, if desired.
`has.ylabels`	Logical, whether the x-axis should have labels or not.
`has.xlabels`	Logical, whether the y-axis should have labels or not.
`comp1`	The first unit of a pairwise comparison, if the goal is to compare two groups. Automatically displays `*`, `**`, or `***` depending on significance of the difference. Can take either a numeric value (based on the group number) or the name of the group directly. Must be provided along with argument `comp2`.
`comp2`	The second unit of a pairwise comparison, if the goal is to compare two groups. Automatically displays "", ""**, or** "" depending on significance of the difference. Can take either a numeric value (based on the group number) or the name of the group directly. Must be provided along with argument `comp1`.
`signif_annotation`	Manually provide the required annotations/numbers of stars (as character strings). Useful if the automatic pairwise comparison annotation does not work as expected, or yet if one wants more than one pairwise comparison. Must be provided along with arguments `signif_yposition`, `signif_xmin`, and `signif_xmax`.
`signif_yposition`	Manually provide the vertical position of the annotations/stars, based on the y-scale.

signif_xmin	Manually provide the first part of the horizontal position of the annotations/stars (start of the left-sided bracket), based on the x-scale.
signif_xmax	Manually provide the second part of the horizontal position of the annotations/stars (end of the right-sided bracket), based on the x-scale.
ymin	The minimum score on the y-axis scale.
ymax	The maximum score on the y-axis scale.
yby	How much to increase on each "tick" on the y-axis scale.
CIcap.width	The width of the confidence interval cap.
obs	Logical, whether to plot individual observations or not.
alpha	The transparency of the plot.
border.colour	The colour of the violins border.
border.size	The size of the violins border.
has.d	Whether to display the d-value.
d.x	The x-axis coordinates for the d-value.
d.y	The y-axis coordinates for the d-value.

See Also

Visualize group differences via scatter plots: nice_scatter. Tutorial: https://remi-theriault.com/blog_violin

Examples

```
# Make the basic plot
nice_violin(data = ToothGrowth,
           group = "dose",
           response = "len")

## Not run:
# Save a high-resolution image file to specified directory
ggsave('niceviolinplothere.pdf', width = 7, height = 7, unit = 'in',
       dpi = 300, path = "/") # change for your own desired path

## End(Not run)

# Change x- and y- axes labels
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            ytitle = "Length of Tooth",
            xtitle = "Vitamin C Dosage")

# See difference between two groups
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            comp1 = "0.5",
            comp2 = "2")

nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            comp1 = 2,
            comp2 = 3)

# Compare all three groups
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            signif_annotation = c("*","**","***"),
                # manually enter the number of stars
            signif_yposition = c(30,35,40),
                # What height (y) should the stars appear
            signif_xmin = c(1,2,1),
                # Where should the left-sided brackets start (x)
            signif_xmax = c(2,3,3))
                # Where should the right-sided brackets end (x)

# Set the colours manually
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            colours = c("darkseagreen","cadetblue","darkslateblue"))

# Changing the names of the x-axis labels
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            xlabels = c("Low", "Medium", "High"))

# Removing the x-axis or y-axis titles
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            ytitle = NULL,
            xtitle = NULL)

# Removing the x-axis or y-axis labels (for whatever purpose)
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            has.ylabels = FALSE,
            has.xlabels = FALSE)

# Set y-scale manually
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            ymin = 5,
            ymax = 35,
            yby = 5)

# Plotting individual observations
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            obs = TRUE)

# Micro-customizations
```

```
nice_violin(data = ToothGrowth,
            group = "dose",
            response = "len",
            CIcap.width = 0,
            alpha = .70,
            border.size = 1,
            border.colour = "white",
            comp1 = 1,
            comp2 = 2,
            has.d = TRUE)
```

`overlap_circle` *Interpolate the Inclusion of the Other in the Self Scale*

Description

Interpolating the Inclusion of the Other in the Self Scale (self-other merging) easily.

Usage

```
overlap_circle(response, categories = c("Self", "Other"))
```

Arguments

`response`	The variable to plot.
`categories`	The desired categories of the two overlapping circles.

See Also

Tutorial: https://remi-theriault.com/blog_circles

Examples

```
# Score of 1 (0% overlap)
overlap_circle(1)

# Score of 3.5 (25% overlap)
overlap_circle(3.5)

# Score of 6.84 (81.8% overlap)
overlap_circle(6.84)

# Changing labels
overlap_circle(3.12, categories = c("Humans","Animals"))

##  NOT RUN
# Saving to file (PDF or PNG)
## plot <- overlap_circle(3.5)
## ggplot2::ggsave(plot, file=NULL, width=7, height=7, unit='in', dpi=300)
# Change for your own desired path
```

`scale_mad` *Standardize based on the absolute median deviation*

Description

Scale and center ("standardize") data based on the median absolute deviation.

See: Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, *49*(4), 764–766. https://doi.org/10.1016/j.jesp.2013.03.013

Usage

```
scale_mad(x)
```

Arguments

`x`	The vector to be scaled.

Author(s)

Hugues Leduc, Charles-Étienne Lavoie

Examples

```
scale_mad(mtcars$mpg)
```

`winsorize_mad` *Winsorize based on the absolute median deviation*

Description

Winsorize (bring extreme observations to usually +/- 3 standard deviations) data based on median absolute deviations instead of standard deviations.

See: Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, *49*(4), 764–766. https://doi.org/10.1016/j.jesp.2013.03.013

Usage

```
winsorize_mad(x, criteria = 3)
```

Arguments

`x`	The vector to be winsorized based on the MAD.
`criteria`	How many MAD to use as threshold (similar to standard deviations)

Author(s)

Hugues Leduc, Charles-Étienne Lavoie

Examples

```
winsorize_mad(mtcars$qsec, criteria = 2)
```

Index