Package ‘rempsyC’

November 1, 2022

Title Convenience Functions for Psychology

Version 0.1.0

Date 2022-10-27

Description Make your workflow faster and easier. Easily customizable plots (via ‘ggplot2’), nice APA tables (following the style of the *American Psychological Association*) exportable to Word (via ‘flextable’), easily run statistical tests or check assumptions, and automatize various other tasks.

License GPL (>= 3)

URL https://rempsyC.remi-theriault.com

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

Depends R (>= 3.5)

Imports methods, rlang, dplyr (>= 1.0.4), flextable (>= 0.7.1), effectsize, insight (>= 0.18.4), performance (>= 0.10.0)

Suggests ggploT2, lmeTest, ggrepel, boot, bootES, ggSignif, qqplotr, broom, correlation, datawizard (>= 0.5.0), emmeans, ggpubr, interactions, knitr, markdown, openxlsx, openxlsx2, patchwork, psych, report (>= 0.5.1), see, testthat (>= 3.0.0), VennDiagram

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Author Rémi Thériault [aut, cre] (<https://orcid.org/0000-0003-4315-6788>)

Maintainer Rémi Thériault <remi.theriault@mail.mcgill.ca>

Repository CRAN

Date/Publication 2022-11-01 22:15:02 UTC
**R topics documented:**

- `best_duplicate` .................................................. 2
- `cormatrix_excel` ................................................ 3
- `extract_duplicates` .............................................. 4
- `find_mad` ............................................................ 5
- `format_value` ...................................................... 6
- `nice_assumptions` ................................................ 7
- `nice_density` ........................................................ 8
- `nice_lm` .............................................................. 10
- `nice_lm_slopes` .................................................... 11
- `nice_mod` ............................................................ 12
- `nice_na` .............................................................. 13
- `nice_normality` .................................................... 15
- `nice_qq` .............................................................. 17
- `nice_randomize` .................................................... 18
- `nice_reverse` ........................................................ 19
- `nice_scatter` ....................................................... 20
- `nice_slopes` ........................................................ 26
- `nice_table` .......................................................... 28
- `nice_t_test` ........................................................ 31
- `nice_var` ............................................................ 33
- `nice_varplot` ....................................................... 34
- `nice_violin` ........................................................ 35
- `overlap_circle` ..................................................... 40
- `scale_mad` ............................................................ 41
- `sr2` ................................................................. 41
- `winsorize_mad` .................................................... 42

**Index** 43

---

`best_duplicate`  

Choose the best duplicate

**Description**

Chooses the best duplicate, based on the duplicate with the fewer number of NA values. In case of ties, it picks the first duplicate, as it is the one most likely to be valid and authentic, given practice effects.

**Usage**

`best_duplicate(data, id)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>data</code></td>
<td>The data frame.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>The ID variable for which to check for duplicates.</td>
</tr>
</tbody>
</table>
Value

A dataframe, containing only the "best" duplicates.

Examples

df1 <- data.frame(
  id = c(1, 2, 3, 1, 3),
  item1 = c(NA, 1, 1, 2, 3),
  item2 = c(NA, 1, 1, 2, 3),
  item3 = c(NA, 1, 1, 2, 3)
)

best_duplicate(df1, id = "id")

cormatrix_excel

Easy export of correlation matrix to Excel (improved version)

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 effect size (0.0-0.2: small (no colour); 0.2-0.4: medium (pink/light blue); 0.4-1.0: large (red/dark blue)), following Cohen’s suggestions for small (.10), medium (.30), and large (.50) correlation sizes.

Based on the correlation and openxlsx2 packages.

Usage

cormatrix_excel(
  data,
  filename,
  overwrite = TRUE,
  p_adjust = "none",
  print.mat = TRUE,
  ...
)

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.
p_adjust Default p-value adjustment method (default is "none", although correlation::correlation's default is "holm")
print.mat Logical, whether to also print the correlation matrix to console.
... Parameters to be passed to the correlation package (see ?correlation::correlation)
A Microsoft Excel document, containing the colour-coded correlation matrix with significance stars, on the first sheet, and the colour-coded p-values on the second sheet.

Author(s)

Adapted from @JanMarvin (JanMarvin/openxlsx2#286) and the original rempsyc::cormatrix_excel (now internal function cormatrix_excel_deprecated)

Examples

# Basic example
cormatrix_excel(mtcars, "cormatrix1")
cormatrix_excel(iris, p_adjust = "none", "cormatrix2")
cormatrix_excel(airquality, method = "spearman", "cormatrix3")

Description

Chooses the best duplicate, based on the duplicate with the fewer number of NA values. In case of ties, it picks the first duplicate, as it is the one most likely to be valid and authentic, given practice effects.

Usage

extract_duplicates(data, id)

Arguments

data The data frame.
id The ID variable for which to check for duplicates.

Value

A dataframe, containing all duplicates, for visual inspection. Note that it also contains the first occurrence of future duplicates, unlike the duplicated() base R function. Also contains an additional column reporting the number of missing values for that row, to help in the decision-making when selecting which duplicates to keep.
Examples

    df1 <- data.frame(
        id = c(1, 2, 3, 1, 3),
        item1 = c(NA, 1, 1, 2, 3),
        item2 = c(NA, 1, 1, 2, 3),
        item3 = c(NA, 1, 1, 2, 3)
    )

    extract_duplicates(df1, id = "id")

    # Filter to exclude duplicates
    df2 <- df1[-c(1, 5),]
    df2

find_mad

Identify outliers based on 3 MAD

Description

Identify outliers based on 3 median absolute deviations (MAD).


Usage

    find_mad(data, col.list, ID = NULL, criteria = 3, mad.scores = TRUE)

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)
- **mad.scores**: Logical, whether to output robust z (MAD) scores (default) or raw scores. Defaults to TRUE.

Value

A list of dataframes of outliers per variable, with row numbers, based on the MAD. When printed, provides the number of outliers, selected variables, and any outlier flagged for more than one variable. More information can be obtained by using the `attributes()` function around the generated object.
Author(s)

Hugues Leduc, Charles-Étienne Lavoie, Rémi Thériault

Examples

```r
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**

```r
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.

#### Value
A formatted p, r, or d value.

#### Examples
```r
format_value(0.00041231, "p")
format_value(0.00041231, "r")
format_value(1.341231, "d")
format_p(0.0041231)
format_p(0.00041231)
format_r(0.41231)
format_r(0.041231)
format_d(1.341231)
format_d(0.341231)
```

---

### nice_assumptions
**Easy assumptions checks**

#### Description
Test linear regression assumptions easily with a nice summary table.

#### Usage
```r
nice_assumptions(model)
```

#### Arguments

- **model**
The `lm` object to be passed to the function.

#### Value
A dataframe, with p-value results for the Shapiro-Wilk, Breusch-Pagan, and Durbin-Watson tests, as well as a diagnostic column reporting how many assumptions are not respected for a given model.

#### See Also
Other functions useful in assumption testing:
- `nice_density`
- `nice_normality`
- `nice_qq`
- `nice_varplot`
- `nice_var`

Tutorial: [https://rempsyc.remi-theriault.com/articles/assumptions](https://rempsyc.remi-theriault.com/articles/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 ))
assumptions.table

nice_density

Easy density plots

Description

Make nice density plots easily. Internally, uses na.rm = TRUE.

Usage

nice_density(
  data,
  variable,
  group,
  colours,
  ytitle = "Density",
  xtitle = variable,
  groups.labels = NULL,
  grid = TRUE,
  shapiro = FALSE,
  title = variable,
  histogram = FALSE,
  breaks.auto = FALSE,
  bins = 30
)

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.
nice_density

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
breaks.auto     If histogram = TRUE, then option to set bins/breaks automatically, mimicking the default behaviour of base R hist() (the Sturges method). Defaults to FALSE.
bins     If histogram = TRUE, then option to change the default bin (30).

Value
A density plot of class ggplot, by group (if provided), along a reference line representing a matched normal distribution.

See Also
Other functions useful in assumption testing: nice_assumptions, nice_normality, nice_qq, nice_varplot, nice_var. Tutorial: https://rempsyc.remi-theriault.com/articles/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

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.

Value

A formatted dataframe of the specified lm model, with DV, IV, degrees of freedom, regression coefficient, t-value, p-value, and the effect size, the semi-partial correlation squared.

See Also

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

Value

A formatted dataframe of the simple slopes of the specified lm model, with DV, levels of IV, degrees of freedom, regression coefficient, t-value, p-value, and the effect size, the semi-partial correlation squared.

See Also


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.

Value

A formatted dataframe of the specified lm model, with DV, IV, degrees of freedom, regression coefficient, t-value, p-value, and the effect size, the semi-partial correlation squared.

See Also

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

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").

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).

Value

A dataframe, with:

- var: variables selected
- items: number of items for selected variables
- na: number of missing cell values for those variables (e.g., 2 missing values for first participant + 2 missing values for second participant = total of 4 missing values)
- cells: total number of cells (i.e., number of participants multiplied by number of variables, items)
- na_percent: the percentage of missing values (number of missing cells, na, divided by total number of cells, cells)
- na_max: The amount of missing values for the participant with the most missing values for the selected variables
- na_max_percent: The amount of missing values for the participant with the most missing values for the selected variables, in percentage (i.e., na_max divided by the number of selected variables, items)
- all_na: the number of participants missing 100% of items for that scale (the selected variables)

References


Examples

# Use whole data frame
nice_na(airquality)

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

# If the questionnaire items start with the same name, e.g.,
set.seed(15)
fun <- function() {
  c(sample(c(NA, 1:10), replace = TRUE), NA, NA, NA)
}

df <- data.frame(
  ID = c("idz", NA),
  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("ID", "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,
  breaks.auto = FALSE,
  ...
)

Arguments

data The data frame.
variable The dependent variable to be plotted.
nice_normality

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.
breaks.auto If histogram = TRUE, then option to set bins/breaks automatically, mimicking the default behaviour of base R hist() (the Sturges method). Defaults to FALSE.
... Further arguments from nice_qq() and nice_density() to be passed to nice_normality()

Value
A plot of classes patchwork and ggplot, containing two plots, resulting from nice_density and nice_qq.

See Also
Other functions useful in assumption testing: nice_assumptions, nice_density, nice_qq, nice_var, nice_varplot. Tutorial: https://rempsyc.remi-theriault.com/articles/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


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.

Value

A qq plot of class ggplot, by group (if provided), along a reference interpretation helper, the 95% confidence band.

See Also

Other functions useful in assumption testing: nice_assumptions, nice_density, nice_normality, nice_var, nice_varplot. Tutorial: https://rempsyc.remi-theriault.com/articles/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 using between.

condition.names  
The names of the randomized conditions.

col.names  
The desired additional column names for a runsheet.
Value

A dataframe, with participant ID and randomized condition, based on selected design.

See Also

Tutorial: https://rempsy.c.remi-theriault.com/articles/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 minimum score on the scale (optional unless it isn’t 1).
- **warning**: Logical. Whether to show the warning about the minimum not being 1.

## Value

A numeric vector, of reversed scores.

## Examples

```r
# 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)
```

## Description

Make nice scatter plots easily.

## Usage

```r
nice_scatter(
    data,
    predictor,
    response,
    xtitle = predictor,
    ytitle = response,
    has.points = TRUE,
    has.jitter = FALSE,
    alpha = 0.7,
    has.line = TRUE,
    has.confband = FALSE,
    has.fullrange = FALSE,
    has.linetype = FALSE,
)```
nice_scatter

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
predictor
response
xtitle
ytitle
has.points
has.jitter
alpha
has.line
has.confband
has.fullrange
has.linetype
has.shape
xmin
xmax
xby
ymin
ymax

The data frame.
The independent variable to be plotted.
The dependent variable to be plotted.
An optional y-axis label, if desired.
An optional x-axis label, if desired.
Whether to plot the individual observations or not.
Alternative to has.points. "Jitters" the observations to avoid overlap (overplotting). Use one or the other, not both.
The desired level of transparency.
Whether to plot the regression line(s).
Logical. Whether to display the confidence band around the slope.
Logical. Whether to extend the slope beyond the range of observations.
Logical. Whether to change line types as a function of group.
Logical. Whether to change shape of observations as a function of group.
The minimum score on the x-axis scale.
The maximum score on the x-axis scale.
How much to increase on each “tick” on the x-axis scale.
The minimum score on the y-axis scale.
The maximum score on the y-axis scale.
### nice_scatter

- **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.

### Value

A scatter plot of class ggplot.

### See Also


### Examples

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

# Save a high-resolution image file to specified directory
ggplot2::ggsave("nicescatterplothere.pdf", width = 7,
  height = 7, unit = "in", dpi = 300
) # change for your own desired path

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

)  

# 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",
  has.confband = TRUE,
  has.jitter = TRUE,
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(
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.legend = TRUE,
    legend.title = "cylinders"
  )
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 $r^2$ 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.
nice_slopes

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.

Value
   A formatted dataframe of the simple slopes of the specified lm model, with DV, levels of IV, degrees of freedom, regression coefficient, t-value, p-value, and the effect size, the semi-partial correlation squared.

See Also
   Checking for moderation before checking simple slopes: nice_mod, nice_lm, nice_lm_slopes.
   Tutorial: https://rempsyc.remi-theriault.com/articles/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"
   )
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,
  col.format.ci,
  format.custom,
  col.format.custom,
  width = 1,
  broom = "",
  report = "",
  short = FALSE,
  title,
  footnote,
  separate.header
)

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.

### col.format.ci
Applies 95% confidence interval formatting to selected columns (e.g., when reporting more than one interval).

### 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.

### title
Optional, to add a table header, if desired.

### footnote
Optional, to add a table footnote (or more), if desired.

### separate.header
Logical, whether to separate headers based on name delimiters (i.e., periods ".").

### Value
An APA-formatted table of class "flextable" (and "nice_table").

### See Also
Tutorial: [https://rempsyc.remi-theriault.com/articles/table](https://rempsyc.remi-theriault.com/articles/table)

### Examples

```r
# Make the basic table
my_table <- nice_table(mtcars[1:3, ],
  title = "Motor Trend Car Road Tests",
  footnote = "1974 Motor Trend US magazine."
)
my_table

# Save table to word
mypath <- tempfile(fileext = ".docx")
save_as_docx(my_table, path = mypath)
```
# 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"
)
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")
```

# Separate headers based on periods
```
header.data <- structure(list(
  Variable = c(
    "Sepal.Length",
    "Sepal.Width", "Petal.Length"
  ),
  setosa.M = c(5.01, 3.43, 1.46),
  setosa.SD = c(0.35, 0.38, 0.17),
  versicolor.M = c(5.94, 2.77, 4.26),
  versicolor.SD = c(0.52, 0.31, 0.47)
),
  row.names = c(NA, -3L), class = "data.frame"
)
```
nice_t_test

)  
nice_table(header.data,  
  separate.header = TRUE,  
  italics = 2:4  
)

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).

Value

A formatted dataframe of the specified model, with DV, degrees of freedom, t-value, p-value, the effect size, Cohen’s d, and its 95% confidence interval lower and upper bounds.
See Also

Tutorial: https://rempsyc.remi-theriault.com/articles/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.

### Value

A dataframe, with the values of the selected variables for each group, their max variance ratio 
(maximum variance divided by the minimum variance), the selected decision criterion, and whether 
the data are considered heteroscedastic according to the decision criterion.

### See Also

Other functions useful in assumption testing: nice_assumptions, nice_density, nice_normality, 

### Examples

```r
# 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 = 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.
ytitle  An optional y-axis label, if desired.

Value

A scatter plot of class ggplot attempting to display the group variances. Also includes the max variance ratio (maximum variance divided by the minimum variance).

See Also

Other functions useful in assumption testing: nice_assumptions, nice_density, nice_normality, nice.qq, nice.var. Tutorial: https://rempyc.remi-theriault.com/articles/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% (possibly bootstrapped) confidence intervals.

**Usage**

nice_violin(
  data,
  group,
  response,
  boot = FALSE,
  bootstraps = 2000,
  colours,
  xlabels = NULL,
  ytitle = response,
  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,
CI.cap.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 for the confidence interval 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.

Details

Using boot = TRUE uses bootstrapping (for the confidence intervals only) with the BCa method, using the rcompanion_groupwiseMean function.

Value

A violin plot of class ggplot, by group.

See Also


Examples

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

# Save a high-resolution image file to specified directory
ggplot2::ggsave("niceviolinplotthere.pdf", width = 7, height = 7, unit = "in", dpi = 300)
# Change x- and y- axes labels

```r
nice_violin(
  data = ToothGrowth,
  group = "dose",
  response = "len",
  ytitle = "Length of Tooth",
  xtitle = "Vitamin C Dosage"
)
```

# See difference between two groups

```r
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

```r
nice_violin(
  data = ToothGrowth,
  group = "dose",
  response = "len",
  signif_annotation = c("*", "**", "***"),
  signif_yposition = c(30, 35, 40),
  signif_xmin = c(1, 2, 1),
  signif_xmax = c(2, 3, 3)
)
```

# Where should the right-sided brackets end (x)

# Set the colours manually

```r
nice_violin(
  data = ToothGrowth,
  group = "dose",
  response = "len",
  colours = c("darkseagreen", "cadetblue", "darkslateblue")
)
```

# Changing the names of the x-axis labels

```r
nice_violin(
  data = ToothGrowth,
  group = "dose",
  response = "len",
  xtitle = "Vitamin C Dosage"
)
```
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,
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.

Value

A plot of class gList, displaying overlapping circles relative to the selected score.

See Also

Tutorial: https://rempsyc.remi-theriault.com/articles/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"))

# Saving to file (PDF or PNG)
mypath <- tempfile(fileext = ".pdf")
plot <- overlap_circle(3.5)
ggplot2::ggsave(plot, file = mypath, 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.

**Usage**

```r
scale_mad(x)
```

**Arguments**

- `x` The vector to be scaled.

**Value**

A numeric vector of standardized data.

**Author(s)**

Hugues Leduc, Charles-Étienne Lavoie

**References**


**Examples**

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

---

**sr2**

*Semi-Partial Correlation Squared (Delta R2)*

**Description**

Compute the semi-partial correlation squared (also known as the delta R2), for a `lm` model.

**Usage**

```r
sr2(model, ...)
```
Arguments

- **model**: An `lm` model.
- **...**: Arguments passed to `lm`. these can be `subset` and `na.action`.

Value

A data frame with the effect size.

Examples

```r
m <- lm(mpg ~ cyl + disp + hp * drat, data = mtcars)
sr2(m)
```

---

**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.

Usage

```r
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)

Value

A numeric vector of winsorized data.

Author(s)

Hugues Leduc, Charles-Étienne Lavoie

References


Examples

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

* APA
  nice_table, 28
* Excel
  cormatrix_excel, 3
* MAD
  find_mad, 5
  scale_mad, 41
  winsorize_mad, 42
* NA
  nice_na, 13
* QQ
  nice_normality, 15
  nice_qq, 17
* Venn
  overlap_circle, 40
* allocation
  nice_randomize, 18
* assumptions
  nice_assumptions, 7
* conditions
  nice_randomize, 18
* correlation
  cormatrix_excel, 3
  format_value, 6
* density
  nice_density, 8
  nice_normality, 15
* design
  nice_randomize, 18
* diagrams
  overlap_circle, 40
* differences
  nice_t_test, 31
* distribution
  nice_normality, 15
  nice_qq, 17
* duplicates
  best_duplicate, 2
  extract_duplicates, 4
* effect size correlation
  sr2, 41
* experimental
  nice_randomize, 18
* formatting
  format_value, 6
* group
  nice_t_test, 31
* guidelines
  nice_na, 13
* interaction
  nice_lm, 10
  nice_lm_slopes, 11
  nice_mod, 12
  nice_slopes, 26
* linear
  nice_assumptions, 7
* matrix
  cormatrix_excel, 3
* mean
  find_mad, 5
  scale_mad, 41
  winsorize_mad, 42
* median
  find_mad, 5
  scale_mad, 41
  winsorize_mad, 42
* merging
  overlap_circle, 40
* missing
  nice_na, 13
* moderation
  nice_lm, 10
  nice_lm_slopes, 11
  nice_mod, 12
  nice_slopes, 26
* normality
  nice_density, 8
  nice_normality, 15
nice_qq, 17
* normalization
  find_mad, 5
  scale_mad, 41
  winsorize_mad, 42
* outliers
  find_mad, 5
  scale_mad, 41
  winsorize_mad, 42
* overlap
  overlap_circle, 40
* p-value
  format_value, 6
* plots
  nice_normality, 15
  nice_qq, 17
  nice_scatter, 20
  nice_violin, 35
* psychology
  overlap_circle, 40
* r-value
  format_value, 6
* randomization
  nice_randomize, 18
* random
  nice_randomize, 18
* regression
  nice_assumptions, 7
  nice_lm, 10
  nice_lm_slopes, 11
  nice_mod, 12
  nice_slopes, 26
* reverse
  nice_reverse, 19
* scatter
  nice_scatter, 20
* scoring
  nice_reverse, 19
* self-other
  overlap_circle, 40
* simple
  nice_slopes, 26
* slopes
  nice_slopes, 26
* social
  overlap_circle, 40
* standardization
  find_mad, 5
  scale_mad, 41
  winsorize_mad, 42
* statistical
  nice_assumptions, 7
* style
  nice_table, 28
* t-test
  nice_t_test, 31
* table
  nice_table, 28
* values
  nice_na, 13
* variance
  nice_var, 33
  nice_varplot, 34
* violations
  nice_assumptions, 7
* violin
  nice_violin, 35

best_duplicate, 2
cormatrix_excel, 3
extract_duplicates, 4
find_mad, 5
format_d(format_value), 6
format_p(format_value), 6
format_r(format_value), 6
format_value, 6

nice_assumptions, 7, 9, 16, 17, 33, 34
nice_density, 7, 8, 16, 17, 33, 34
nice_lm, 10, 11, 12, 27
nice_lm_slopes, 10, 11, 12, 27
nice_mod, 10, 11, 12, 27
nice_na, 13
nice_normality, 7, 9, 15, 17, 33, 34
nice_qq, 7, 9, 16, 17, 33, 34
nice_randomize, 18
nice_reverse, 19
nice_scatter, 20, 37
nice_slopes, 10–12, 26
nice_t_test, 31
nice_table, 28
nice_var, 7, 9, 16, 17, 33, 34
nice_varplot, 7, 9, 16, 17, 33, 34
nice_violin, 22, 35
INDEX

overlap_circle, 40
scale_mad, 41
sr2, 41
winsorize_mad, 42