

# Package ‘rempsyc’

September 15, 2025

**Title** Convenience Functions for Psychology

**Version** 0.2.0

**Date** 2025-09-14

**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 (>= 3.6)

**Imports** rlang, dplyr (>= 1.1.0)

**Suggests** flextable (>= 0.9.1), ggplot2 (>= 3.4.0), effectsize (>= 0.8.5), performance (>= 0.10.0), insight (>= 0.18.4), correlation, datawizard (>= 0.5.0), report (>= 0.5.1), modelbased, see, lmtest, ggrepel, boot, bootES, ggsignif, qqplotr (>= 0.0.6), broom, emmeans, ggpubr, interactions, openxlsx2 (>= 0.8), patchwork, psych, VennDiagram, Rmisc, methods, tidyr, testthat (>= 3.0.0), knitr, markdown, rmarkdown

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**NeedsCompilation** no

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

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

**Repository** CRAN

**Date/Publication** 2025-09-15 00:20:02 UTC

## Contents

best_duplicate . . . . .	2
cormatrix_excel . . . . .	3
extract_duplicates . . . . .	5
find_mad . . . . .	6
format_value . . . . .	7
get_dep_version . . . . .	8
grouped_bar_chart . . . . .	9
install_if_not_installed . . . . .	10
nice_assumptions . . . . .	10
nice_contrasts . . . . .	11
nice_density . . . . .	13
nice_lm . . . . .	15
nice_lm_contrasts . . . . .	17
nice_lm_slopes . . . . .	18
nice_mod . . . . .	20
nice_na . . . . .	22
nice_normality . . . . .	24
nice_qq . . . . .	26
nice_randomize . . . . .	27
nice_reverse . . . . .	28
nice_scatter . . . . .	29
nice_slopes . . . . .	36
nice_table . . . . .	38
nice_t_test . . . . .	41
nice_var . . . . .	43
nice_varplot . . . . .	44
nice_violin . . . . .	46
overlap_circle . . . . .	50
plot_means_over_time . . . . .	52
plot_outliers . . . . .	54
scale_mad . . . . .	56
winsorize_mad . . . . .	57
<b>Index</b>	<b>58</b>

---

best_duplicate	<i>Choose the best duplicate</i>
----------------	----------------------------------

---

### Description

Chooses the best duplicate, based on the duplicate with the smallest number of missing 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, keep.rows = FALSE)
```

**Arguments**

data	The data frame.
id	The ID variable for which to check for duplicates.
keep.rows	Logical, whether to add a column at the beginning of the data frame with the original row indices.

**Details**

For the *easystats* equivalent, see: [datawizard::data\\_duplicated\(\)](#).

**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", keep.rows = TRUE)
```

**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 openxlsx packages.

## Usage

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

## Arguments

<code>data</code>	The data frame
<code>filename</code>	Desired filename (path can be added before hand but no need to specify extension).
<code>overwrite</code>	Whether to allow overwriting previous file.
<code>p_adjust</code>	Default p-value adjustment method (default is "none", although <code>correlation::correlation()</code> 's default is "holm")
<code>print.mat</code>	Logical, whether to also print the correlation matrix to console.
<code>...</code>	Parameters to be passed to the <code>correlation</code> package (see <code>correlation::correlation()</code> )

## Details

For the *easystats* equivalent, see: `correlation::cormatrix_to_excel()`.

## Value

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 `cormatrix_excel` (now imported from `correlation::cormatrix_to_excel`).

## Examples

```
# Basic example  
cormatrix_excel(mtcars, select = c("mpg", "cyl", "disp", "hp", "carb"), filename = "cormatrix1")  
cormatrix_excel(iris, p_adjust = "none", filename = "cormatrix2")  
cormatrix_excel(airquality, method = "spearman", filename = "cormatrix3")
```

---

extract_duplicates	<i>Extract all duplicates</i>
--------------------	-------------------------------

---

### Description

Extract all duplicates, for visual inspection. Note that it also contains the first occurrence of future duplicates, unlike `duplicated()` or `dplyr::distinct()`. 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.

### Usage

```
extract_duplicates(data, id)
```

### Arguments

<code>data</code>	The data frame.
<code>id</code>	The ID variable for which to check for duplicates.

### Details

For the *easystats* equivalent, see: `datawizard::data_unique()`.

### Value

A dataframe, containing all 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)
)

extract_duplicates(df1, id = "id")

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

---

find_mad	<i>Identify outliers based on 3 MAD</i>
----------	---

---

### Description

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

### 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.

### Details

The function internally use [scale\\_mad\(\)](#) to "standardize" the data based on the MAD and median, and then check for any observation greater than the specified criteria (e.g., +/-3).

For the *easystats* equivalent, use: `performance::check_outliers(x, method = "zscore_robust, threshold = 3)`.

### 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.

### References

Ley, 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>

### 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,
  stars = FALSE
)
```

```
format_r(r, precision = 0.01)
```

```
format_d(d, precision = 0.01)
```

## Arguments

value	Value to be formatted, when using the generic <code>format_value()</code> .
type	Specify r or p value.
...	To specify precision level, if necessary, when using the generic <code>format_value()</code> . 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.

stars	Logical. Whether to add asterisks for significant p values.
r	r value to format.
d	d value to format.

### Details

For the *easystats* equivalent, see: [insight::format\\_value\(\)](#).

### Value

A formatted p, r, or d value.

### Examples

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

---

get_dep_version	<i>Get required version of specified package dependency</i>
-----------------	---

---

### Description

Get required version of specified package dependency

### Usage

```
get_dep_version(dep, pkg = utils::packageName())
```

### Arguments

dep	Dependency of the specified package to check
pkg	Package to check the dependency from



---

grouped\_bar\_chart      *Easy grouped bar charts for categorical variables*

---

## Description

Make nice grouped bar charts easily.

## Usage

```
grouped_bar_chart(  
  data,  
  response,  
  label = response,  
  group = "T1_Group",  
  proportion = TRUE,  
  print_table = FALSE  
)
```

## Arguments

data	The data frame.
response	The categorical dependent variable to be plotted.
label	Label of legend describing the dependent variable.
group	The group by which to plot the variable
proportion	Logical, whether to use proportion (default), else, counts.
print_table	Logical, whether to also print the computed proportion or count table.

## Value

A bar plot of class ggplot.

## Examples

```
# Make the basic plot  
iris2 <- iris  
iris2$plant <- c(  
  rep("yes", 45),  
  rep("no", 45),  
  rep("maybe", 30),  
  rep("NA", 30)  
)  
grouped_bar_chart(  
  data = iris2,  
  response = "plant",  
  group = "Species"  
)
```

---

```
install_if_not_installed
```

*Install package if not already installed*

---

**Description**

Install package if not already installed

**Usage**

```
install_if_not_installed(pkgs)
```

**Arguments**

pkgs                    Packages to install if not already installed

---

```
nice_assumptions
```

*Easy assumptions checks*

---

**Description**

Test linear regression assumptions easily with a nice summary table.

**Usage**

```
nice_assumptions(model)
```

**Arguments**

model                    The `lm()` object to be passed to the function.

**Details**

Interpretation: (p) values < .05 imply assumptions are not respected. Diagnostic is how many assumptions are not respected for a given model or variable.

**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. Shapiro-Wilk is set to NA if  $n < 3$  or  $n > 5000$ .

**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>

**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
model2 <- lm(qsec ~ disp + drat * carb, mtcars)
my.models <- list(model, model2)
nice_assumptions(my.models)
```

---

nice\_contrasts                      *Easy planned contrasts*

---

**Description**

Easily compute planned contrast analyses (pairwise comparisons similar to t-tests but more powerful when more than 2 groups), and format in publication-ready format. In this particular case, the confidence intervals are bootstrapped on chosen effect size (default to Cohen's d).

**Usage**

```
nice_contrasts(
  response,
  group,
  covariates = NULL,
  data,
  effect.type = "cohens.d",
  bootstraps = 2000,
  ...
)
```

**Arguments**

response	The dependent variable.
group	The group for the comparison.
covariates	The desired covariates in the model.
data	The data frame.
effect.type	What effect size type to use. One of "cohens.d" (default), "akp.robust.d", "unstandardized", "hedges.g", "cohens.d.sigma", or "r".
bootstraps	The number of bootstraps to use for the confidence interval
...	Arguments passed to <code>bootES::bootES</code> .

## Details

Statistical power is lower with the standard  $t$  test compared than it is with the planned contrast version for two reasons: a) the sample size is smaller with the  $t$  test, because only the cases in the two groups are selected; and b) in the planned contrast the error term is smaller than it is with the standard  $t$  test because it is based on all the cases ([source](#)).

The effect size and confidence interval are calculated via `bootES::bootES`, and correct for contrasts but not for covariates and other predictors. Because this method uses bootstrapping, it is recommended to set a seed before using for reproducibility reasons (e.g., `set.seed(100)`).

Does not for the moment support nested comparisons for marginal means, only a comparison of all groups. For nested comparisons, please use `emmeans::contrast()` directly, or for the *easystats* equivalent, `modelbased::estimate_contrasts()`.

When using `nice_lm_contrasts()`, please use `as.factor()` outside the `lm()` formula, or it will lead to an error.

## Value

A dataframe, with the selected dependent variable(s), comparisons of interest, degrees of freedom, t-values, p-values, Cohen's d, and the lower and upper 95% confidence intervals of the effect size (i.e., dR).

## See Also

`nice_lm_contrasts`, Tutorial: <https://rempsyc.remi-theriault.com/articles/contrasts>

## Examples

```
# Basic example
set.seed(100)
nice_contrasts(
  data = mtcars,
  response = "mpg",
  group = "cyl",
  bootstraps = 200
)

set.seed(100)
nice_contrasts(
  data = mtcars,
  response = "disp",
  group = "gear"
)

# Multiple dependent variables
set.seed(100)
nice_contrasts(
  data = mtcars,
  response = c("mpg", "disp", "hp"),
  group = "cyl"
)
```

```
# Adding covariates
set.seed(100)
nice_contrasts(
  data = mtcars,
  response = "mpg",
  group = "cyl",
  covariates = c("disp", "hp")
)

# Now supports more than 3 levels
mtcars2 <- mtcars
mtcars2$carb <- as.factor(mtcars2$carb)
set.seed(100)
nice_contrasts(
  data = mtcars,
  response = "mpg",
  group = "carb",
  bootstraps = 200
)
```

---

nice\_density

*Easy density plots*

---

## Description

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

## Usage

```
nice_density(
  data,
  variable,
  group = NULL,
  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.
xtitle	An optional x-axis label, if desired.
groups.labels	The groups.labels (might rename to xlabel 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 <code>hist()</code> (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://rempsysc.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)",
histogram = TRUE
)

```

---

nice\_lm

*Nice formatting of lm models*


---

### Description

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

### Usage

```

nice_lm(
  model,
  b.label = "b",
  standardize = FALSE,
  mod.id = TRUE,
  ci.alternative = "two.sided",
  ...
)

```

### 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 <code>nice_table</code> function). Now attempts to automatically detect whether the variables were standardized, and if so, sets <code>b.label = "B"</code> automatically. Factor variables or dummy variables (only two numeric values) are ignored when checking for standardization. <i>This argument is now deprecated, please use argument <code>standardize</code> directly instead.</i>
standardize	Logical, whether to standardize the data before refitting the model. If TRUE, automatically sets <code>b.label = "B"</code> . Defaults to FALSE. Note that if you have factor variables, these will be pseudo-betas, so these coefficients could be interpreted more like Cohen's <i>d</i> .
mod.id	Logical. Whether to display the model number, when there is more than one model.

`ci.alternative` Alternative for the confidence interval of the `sr2`. It can be either "two.sided" (the default in this package), "greater", or "less".

... Further arguments to be passed to the `effectsize::r2_semipartial` function for the effect size.

## Details

The effect size, `sr2` (semi-partial correlation squared, also known as delta R<sup>2</sup>), is computed through `effectsize::r2_semipartial`. Please read the documentation for that function, especially regarding the interpretation of the confidence interval. In `rempsysc`, instead of using the default one-sided alternative ("greater"), we use the two-sided alternative ("greater").

To interpret the `sr2`, use `effectsize::interpret_r2_semipartial()`.

For the `easystats` equivalent, use `report::report()` on the `lm()` model object.

## 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, and its confidence interval.

## See Also

Checking simple slopes after testing for moderation: [nice\\_lm\\_slopes](#), [nice\\_mod](#), [nice\\_slopes](#).  
Tutorial: <https://rempsysc.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)
x <- nice_lm(my.models)
x

# Get interpretations
cbind(x, Interpretation = effectsize::interpret_r2_semipartial(x$sr2))
```



---

nice\_lm\_contrasts      *Easy planned contrasts using lm models*

---

## Description

Easily compute planned contrast analyses (pairwise comparisons similar to t-tests but more powerful when more than 2 groups), and format in publication-ready format. In this particular case, the confidence intervals are bootstrapped on chosen effect size (default to Cohen's d).

## Usage

```
nice_lm_contrasts(
  model,
  group,
  data,
  p_adjust = "none",
  effect.type = "cohens.d",
  bootstraps = 2000,
  ...
)
```

## Arguments

model	The model to be formatted.
group	The group for the comparison.
data	The data frame.
p_adjust	Character: adjustment method (e.g., "bonferroni") – added to options
effect.type	What effect size type to use. One of "cohens.d" (default), "akp.robust.d", "unstandardized", "hedges.g", "cohens.d.sigma", or "r".
bootstraps	The number of bootstraps to use for the confidence interval
...	Arguments passed to <a href="#">bootES::bootES</a> .

## Details

Statistical power is lower with the standard  $t$  test compared than it is with the planned contrast version for two reasons: a) the sample size is smaller with the  $t$  test, because only the cases in the two groups are selected; and b) in the planned contrast the error term is smaller than it is with the standard  $t$  test because it is based on all the cases ([source](#)).

The effect size and confidence interval are calculated via [bootES::bootES](#), and correct for contrasts but not for covariates and other predictors. Because this method uses bootstrapping, it is recommended to set a seed before using for reproducibility reasons (e.g., `sed.set(100)`).

Does not for the moment support nested comparisons for marginal means, only a comparison of all groups. For nested comparisons, please use [emmeans::contrast\(\)](#) directly, or for the *easystats* equivalent, [modelbased::estimate\\_contrasts\(\)](#).

When using `nice_lm_contrasts()`, please use `as.factor()` outside the `lm()` formula, or it will lead to an error.

**Value**

A dataframe, with the selected dependent variable(s), comparisons of interest, degrees of freedom, t-values, p-values, Cohen's d, and the lower and upper 95% confidence intervals of the effect size (i.e., dR).

**See Also**

[nice\\_contrasts](https://rempsysc.remi-theriault.com/articles/contrasts), Tutorial: <https://rempsysc.remi-theriault.com/articles/contrasts>

**Examples**

```
# Make and format model (group need to be a factor)
mtcars2 <- mtcars
mtcars2$cyl <- as.factor(mtcars2$cyl)
model <- lm(mpg ~ cyl + wt * hp, mtcars2)
set.seed(100)
nice_lm_contrasts(model, group = "cyl", data = mtcars, bootstraps = 500)

# Several models at once
mtcars2$gear <- as.factor(mtcars2$gear)
model2 <- lm(qsec ~ cyl, data = mtcars2)
my.models <- list(model, model2)
set.seed(100)
nice_lm_contrasts(my.models, group = "cyl", data = mtcars, bootstraps = 500)

# Now supports more than 3 levels
mtcars2$carb <- as.factor(mtcars2$carb)
model <- lm(mpg ~ carb + wt * hp, mtcars2)
set.seed(100)
nice_lm_contrasts(model, group = "carb", data = mtcars2, bootstraps = 500)
```

---

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.

**Usage**

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

```

    ci.alternative = "two.sided",
    ...
)

```

### 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 <code>nice_table()</code> function). Now attempts to automatically detect whether the variables were standardized, and if so, sets <code>b.label = "B"</code> automatically. Factor variables or dummy variables (only two numeric values) are ignored when checking for standardization. <i>This argument is now deprecated, please use argument <code>standardize</code> directly instead.</i>
standardize	Logical, whether to standardize the data before refitting the model. If TRUE, automatically sets <code>b.label = "B"</code> . Defaults to FALSE. Note that if you have factor variables, these will be pseudo-betas, so these coefficients could be interpreted more like Cohen's <i>d</i> .
mod.id	Logical. Whether to display the model number, when there is more than one model.
ci.alternative	Alternative for the confidence interval of the <code>sr2</code> . It can be either "two.sided" (the default in this package), "greater", or "less".
...	Further arguments to be passed to the <code>lm()</code> function for the models.

### Details

The effect size, `sr2` (semi-partial correlation squared, also known as delta R<sup>2</sup>), is computed through `effectsize::r2_semipartial`. Please read the documentation for that function, especially regarding the interpretation of the confidence interval. In `rempsysc`, instead of using the default one-sided alternative ("greater"), we use the two-sided alternative.

To interpret the `sr2`, use `effectsize::interpret_r2_semipartial()`.

For the *easystats* equivalent, use `report::report()` on the `lm()` model object.

### 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, and its confidence interval.

### See Also

Checking for moderation before checking simple slopes: [nice\\_lm](#), [nice\\_mod](#), [nice\\_slopes](#). Tutorial: <https://rempsysc.remi-theriault.com/articles/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)
x <- nice_lm_slopes(my.models, predictor = "gear", moderator = "wt")
x

# Get interpretations
cbind(x, Interpretation = effectsize::interpret_r2_semipartial(x$sr2))
```

---

nice\_mod

*Easy moderations*


---

**Description**

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

**Usage**

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

**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 <code>nice_table()</code> function). Now attempts to automatically detect whether the variables were standardized, and if so, sets <code>b.label = "B"</code> automatically. Factor variables or dummy variables (only two numeric values) are ignored when checking for standardization. <i>This argument is now deprecated, please use argument <code>standardize</code> directly instead.</i>
standardize	Logical, whether to standardize the data before fitting the model. If TRUE, automatically sets <code>b.label = "B"</code> . Defaults to TRUE.
mod.id	Logical. Whether to display the model number, when there is more than one model.
ci.alternative	Alternative for the confidence interval of the <code>sr2</code> . It can be either "two.sided" (the default in this package), "greater", or "less".
...	Further arguments to be passed to the <code>lm()</code> function for the models.

### Details

The effect size, `sr2` (semi-partial correlation squared, also known as  $\Delta R^2$ ), is computed through `effectsize::r2_semipartial`. Please read the documentation for that function, especially regarding the interpretation of the confidence interval. In `remsyc`, instead of using the default one-sided alternative ("greater"), we use the two-sided alternative.

To interpret the `sr2`, use `effectsize::interpret_r2_semipartial()`.

For the *easystats* equivalent, use `report::report()` on the `lm()` model object.

### 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, and its confidence interval.

### See Also

Checking simple slopes after testing for moderation: `nice_slopes`, `nice_lm`, `nice_lm_slopes`.  
Tutorial: <https://remsyc.remi-theriault.com/articles/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
x <- nice_mod(
  data = mtcars,
  response = "mpg",
  predictor = "gear",
  moderator = "wt",
  moderator2 = "am"
)
x

# Get interpretations
cbind(x, Interpretation = effectsize::interpret_r2_semipartial(x$sr2))

```

---

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").*

## Usage

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

**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**

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>

**Examples**

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

# Use selected columns explicitly
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() {
  c(sample(c(NA, 1:10), replace = TRUE), NA, NA, NA)
```

```

}
df <- data.frame(
  ID = c("idz", NA),
  open_1 = fun(), open_2 = fun(), open_3 = fun(),
  extrovert_1 = fun(), extrovert_2 = fun(), extrovert_3 = fun(),
  agreeable_1 = fun(), agreeable_2 = fun(), agreeable_3 = fun()
)

# One can list the scale names directly:
nice_na(df, scales = c("ID", "open", "extrovert", "agreeable"))

```

---

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 = NULL,
  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.
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 <code>hist()</code> (the Sturges method). Defaults to FALSE.
...	Further arguments from <code>nice_qq()</code> and <code>nice_density()</code> to be passed to <code>nice_normality()</code>

### 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`*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 = NULL,  
  colours,  
  groups.labels = NULL,  
  grid = TRUE,  
  shapiro = FALSE,  
  title = variable  
)
```

## Arguments

<code>data</code>	The data frame.
<code>variable</code>	The dependent variable to be plotted.
<code>group</code>	The group by which to plot the variable.
<code>colours</code>	Desired colours for the plot, if desired.
<code>groups.labels</code>	How to label the groups.
<code>grid</code>	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.
<code>shapiro</code>	Logical, whether to include the p-value from the Shapiro-Wilk test on the plot.
<code>title</code>	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://rempsysc.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://rempsyc.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.

For the *easystats* equivalent, see: [datawizard::reverse\(\)](#).

**Usage**

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

**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).

**Value**

A numeric vector, of reversed scores.

**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 = predictor,
  ytitle = response,
  has.points = TRUE,
  has.jitter = FALSE,
  alpha = 0.7,
  has.line = TRUE,
  method = "lm",
  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 = "none",
groups.labels = NULL,
groups.alpha = NULL,
has.r = FALSE,
r.x = Inf,
r.y = -Inf,
has.p = FALSE,
p.x = Inf,
p.y = -Inf,
has.ids = FALSE,
id.column = NULL,
has.group.r = FALSE,
group.r.x = NULL,
group.r.y = NULL,
has.group.p = FALSE,
group.p.x = NULL,
group.p.y = NULL
)

```

### Arguments

<code>data</code>	The data frame.
<code>predictor</code>	The independent variable to be plotted.
<code>response</code>	The dependent variable to be plotted.
<code>xtitle</code>	An optional y-axis label, if desired.
<code>ytitle</code>	An optional x-axis label, if desired.
<code>has.points</code>	Whether to plot the individual observations or not.
<code>has.jitter</code>	Alternative to <code>has.points</code> . "Jitters" the observations to avoid overlap (overplotting). Use one or the other, not both.
<code>alpha</code>	The desired level of transparency.
<code>has.line</code>	Whether to plot the regression line(s).
<code>method</code>	Which method to use for the regression line, either "lm" (default) or "loess".
<code>has.confband</code>	Logical. Whether to display the confidence band around the slope.
<code>has.fullrange</code>	Logical. Whether to extend the slope beyond the range of observations.

<code>has.linetype</code>	Logical. Whether to change line types as a function of group.
<code>has.shape</code>	Logical. Whether to change shape of observations as a function of group.
<code>xmin</code>	The minimum score on the x-axis scale.
<code>xmax</code>	The maximum score on the x-axis scale.
<code>xby</code>	How much to increase on each "tick" on the x-axis scale.
<code>ymin</code>	The minimum score on the y-axis scale.
<code>ymax</code>	The maximum score on the y-axis scale.
<code>yby</code>	How much to increase on each "tick" on the y-axis scale.
<code>has.legend</code>	Logical. Whether to display the legend or not.
<code>legend.title</code>	The desired legend title.
<code>group</code>	The group by which to plot the variable
<code>colours</code>	Desired colours for the plot, if desired.
<code>groups.order</code>	Specifies the desired display order of the groups on the legend. Either provide the levels directly, or a string: "increasing" or "decreasing", to order based on the average value of the variable on the y axis, or "string.length", to order from the shortest to the longest string (useful when working with long string names). "Defaults to "none".
<code>groups.labels</code>	Changes groups names (labels). Note: This applies after changing order of level.
<code>groups.alpha</code>	The manually specified transparency desired for the groups slopes. Use only when plotting groups separately.
<code>has.r</code>	Whether to display the correlation coefficient, the r-value.
<code>r.x</code>	The x-axis coordinates for the r-value.
<code>r.y</code>	The y-axis coordinates for the r-value.
<code>has.p</code>	Whether to display the p-value.
<code>p.x</code>	The x-axis coordinates for the p-value.
<code>p.y</code>	The y-axis coordinates for the p-value.
<code>has.ids</code>	Whether to display point IDs/labels on the plot.
<code>id.column</code>	The column name to use for point labels. If not specified, row names will be used.
<code>has.group.r</code>	Whether to display correlation coefficients for each group separately when using grouping.
<code>group.r.x</code>	The x-axis coordinates for group correlation coefficients. If NULL (default), will be positioned at the left side of the plot.
<code>group.r.y</code>	The y-axis coordinates for group correlation coefficients. If NULL (default), will be positioned at the top of the plot.
<code>has.group.p</code>	Whether to display p-values for each group separately.
<code>group.p.x</code>	The x-axis coordinates for group p-values. If NULL (default), will be positioned at the right side of the plot.
<code>group.p.y</code>	The y-axis coordinates for group p-values. If NULL (default), will be positioned at the top of the plot.

**Value**

A scatter plot of class `ggplot`.

**See Also**

Visualize group differences via violin plots: [nice\\_violin](#). Tutorial: <https://rempsys.remi-theriault.com/articles/scatter>

**Examples**

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

# Save a high-resolution image file to specified directory
ggplot2::ggsave("nicescatterplotthere.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)"
)

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

# 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  
)  
  
# Change order of labels on the legend  
nice_scatter(  
  data = mtcars,  
  predictor = "wt",  
  response = "mpg",  
  group = "cyl",  
  groups.order = c(8, 4, 6)  
)  
  
# Change legend labels  
nice_scatter(  
  data = mtcars,  
  predictor = "wt",  
  response = "mpg",  
  group = "cyl",  
  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",
    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
)

# Display point IDs/labels
nice_scatter(
  data = mtcars,
  predictor = "wt",
  response = "mpg",
  has.ids = TRUE
)

# Display group correlations separately
nice_scatter(
  data = mtcars,
  predictor = "wt",
  response = "mpg",
  group = "cyl",
  has.group.r = TRUE,
  has.group.p = TRUE
)
```

---

 nice\_slopes
 

---

*Easy simple slopes*


---

### Description

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

### Usage

```
nice_slopes(
  data,
  response,
  predictor,
  moderator,
  moderator2 = NULL,
  covariates = NULL,
  b.label = "b",
  standardize = TRUE,
  mod.id = TRUE,
  ci.alternative = "two.sided",
  ...
)
```

### Arguments

data	The data frame
response	The dependent variable.
predictor	The independent variable
moderator	The moderating variable.
moderator2	The second moderating variable, if applicable. At this time, the second moderator variable can only be a binary variable of the form $c(0, 1)$ .
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 <code>nice_table()</code> function). Now attempts to automatically detect whether the variables were standardized, and if so, sets <code>b.label = "B"</code> automatically. Factor variables or dummy variables (only two numeric values) are ignored when checking for standardization. <i>This argument is now deprecated, please use argument <code>standardize</code> directly instead.</i>
standardize	Logical, whether to standardize the data before fitting the model. If TRUE, automatically sets <code>b.label = "B"</code> . Defaults to TRUE.
mod.id	Logical. Whether to display the model number, when there is more than one model.

`ci.alternative` Alternative for the confidence interval of the `sr2`. It can be either "two.sided" (the default in this package), "greater", or "less".

... Further arguments to be passed to the `lm()` function for the models.

## Details

The effect size, `sr2` (semi-partial correlation squared, also known as delta R<sup>2</sup>), is computed through `effectsize::r2_semipartial`. Please read the documentation for that function, especially regarding the interpretation of the confidence interval. In `rempsysc`, instead of using the default one-sided alternative ("greater"), we use the two-sided alternative.

To interpret the `sr2`, use `effectsize::interpret_r2_semipartial()`.

For the *easystats* equivalent, use `report::report()` on the `lm()` model object.

## 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, and its confidence interval.

## See Also

Checking for moderation before checking simple slopes: [nice\\_mod](#), [nice\\_lm](#), [nice\\_lm\\_slopes](#).  
Tutorial: <https://rempsysc.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)
x <- nice_slopes(
  data = mtcars,
  response = "mpg",
  predictor = "gear",
  moderator = "wt",
  moderator2 = "am"
)
x

# Get interpretations
cbind(x, Interpretation = effectsize::interpret_r2_semipartial(x$sr2))
```

---

nice\_table

*Easily make nice APA tables*

---

## Description

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

## Usage

```
nice_table(
  data,
  highlight = FALSE,
  stars = TRUE,
  italics,
  col.format.p,
  col.format.r,
  col.format.ci,
  format.custom,
  col.format.custom,
  width = NULL,
  spacing = 2,
  broom = NULL,
  report = NULL,
  short = FALSE,
  title,
  note,
  separate.header
)
```

**Arguments**

<code>data</code>	The data frame, to be converted to a flextable. The data frame cannot have duplicate column names.
<code>highlight</code>	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).
<code>stars</code>	Logical. Whether to add asterisks for significant p values.
<code>italics</code>	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.
<code>col.format.p</code>	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.
<code>col.format.r</code>	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.
<code>col.format.ci</code>	Applies 95% confidence interval formatting to selected columns (e.g., when reporting more than one interval).
<code>format.custom</code>	Applies custom formatting to columns selected via the <code>col.format.custom</code> 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.
<code>col.format.custom</code>	Which columns to apply the custom function to. Select with numerical range, e.g., 1:3.
<code>width</code>	Width of the table, in percentage of the total width, when exported e.g., to Word. For full width, use <code>width = 1</code> .
<code>spacing</code>	Spacing of the rows (1 = single space, 2 = double space)
<code>broom</code>	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").
<code>report</code>	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").
<code>short</code>	Logical. Whether to return an abbreviated version of the tables made by the report package.
<code>title</code>	Optional, to add a table header, if desired.
<code>note</code>	Optional, to add one or more table footnote (APA note), if desired.
<code>separate.header</code>	Logical, whether to separate headers based on name delimiters (i.e., periods ".").

**Details**

The resulting flextable objects can be opened in Word with `print(table, preview="docx")`, or saved to Word with the `flextable::save_as_docx()` function.

**Value**

An APA-formatted table of class "flextable"

**See Also**

Tutorial: <https://rempsyc.remi-theriault.com/articles/table>

**Examples**

```
# Make the basic table
my_table <- nice_table(
  mtcars[1:3, ],
  title = c("Table 1", "Motor Trend Car Road Tests"),
  note = c(
    "The data was extracted from the 1974 Motor Trend US magazine.",
    "* p < .05, ** p < .01, *** p < .001"
  )
)
my_table
```

```
# Save table to word
mypath <- tempfile(fileext = ".docx")
flextable::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"
)
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")

# 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_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.

### Usage

```

nice_t_test(
  data,
  response,
  group = NULL,
  correction = "none",
  paired = FALSE,

```

```

    verbose = 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".
paired	Whether to use a paired t-test.
verbose	Whether to display the Welch test warning or not.
...	Further arguments to be passed to the <code>t.test()</code> 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).

### Details

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`.

Note that for paired *t* tests, you need to use `paired = TRUE`, and you also need data in "long" format rather than wide format (like for the `ToothGrowth` data set). In this case, the `group` argument refers to the participant ID for example, so the same group/participant is measured several times, and thus has several rows. Note also that R  $\geq$  4.4.0 has stopped supporting the `paired` argument for the formula method used internally here.

For the *easystats* equivalent, use: `report::report()` on the `t.test()` object.

### 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://rempsysc.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  
)  
  
# 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](#), [nice\\_qq](#), [nice\\_varplot](#). Tutorial: <https://rempsysc.remi-theriault.com/articles/assumptions>

**Examples**

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

# Try on multiple variables
nice_var(
  data = iris,
  variable = names(iris[1:4]),
  group = "Species"
)
```

---

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://rempsyc.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"
  ),

```

```
yttitle = "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,
  response,
  group = NULL,
  boot = FALSE,
  bootstraps = 2000,
  colours,
  xlabel = NULL,
  yttitle = response,
  xttitle = 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,
  groups.order = "none",
```

```

    xlabel.angle = 0
  )

```

## Arguments

data	The data frame.
response	The dependent variable to be plotted.
group	The group by which to plot the variable.
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.
yttitle	An optional y-axis label, if desired.
xttitle	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. The type of plotting can also be specified, either "dotplot" (same as obs = TRUE for backward compatibility) or "jitter", useful when there are a lot of observations.

<code>alpha</code>	The transparency of the plot.
<code>border.colour</code>	The colour of the violins border.
<code>border.size</code>	The size of the violins border.
<code>has.d</code>	Whether to display the d-value.
<code>d.x</code>	The x-axis coordinates for the d-value.
<code>d.y</code>	The y-axis coordinates for the d-value.
<code>groups.order</code>	How to order the group factor levels on the x-axis. Either "increasing" or "decreasing", to order based on the value of the variable on the y axis, or "string.length", to order from the shortest to the longest string (useful when working with long string names). "Defaults to "none".
<code>xlabels.angle</code>	How much to tilt the labels of the x-axis. Useful when working with long string names. "Defaults to 0.

### Details

Using `boot = TRUE` uses bootstrapping (for the confidence intervals only) with the BCa method, using the [`rcompanion\_groupwiseMean`](#) function.

For the *easystats* equivalent, see: [`see::geom\_violindot\(\)`](#).

### Value

A violin plot of class `ggplot`, by group.

### See Also

Visualize group differences via scatter plots: [`nice\_scatter`](#). Tutorial: <https://rempsyc.remi-theriault.com/articles/violin>

### Examples

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

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

# 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",
  xlabel = 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
)
```

## Description

Interpolating the Inclusion of the Other in the Self Scale (IOS; self-other merging) easily. The user provides the IOS score, from 1 to 7, and the function will provide a percentage of actual area overlap between the two circles (i.e., not linear overlap), so it is possible to say, e.g., that experimental group 1 had an average overlap of X% with the other person, whereas experimental group 2 had an average overlap of X% with the other person.

## Usage

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

## Arguments

response	The variable to plot: requires IOS scores ranging from 1 to 7 (when scoring = "IOS").
categories	The desired category names of the two overlapping circles for display on the plot.
scoring	One of c("IOS", "percentage", "direct"). If scoring = "IOS", response needs to be a value between 1 to 7. If set to "percentage" or "direct", responses need to be between 0 and 100. If set to "direct", must provide exactly three values that represent the area from the first circle, the middle overlapping area, and area from the second circle.

## Details

The circles are generated through the `VennDiagram::draw.pairwise.venn()` function and the desired percentage overlap is passed to its `cross.area` argument ("The size of the intersection between the sets"). The percentage overlap values are interpolated from this reference grid: Score of 1 = 0%, 2 = 10%, 3 = 20%, 4 = 30%, 5 = 55%, 6 = 65%, 7 = 85%.

## Value

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

## See Also

Tutorial: <https://rempsyc.remi-theriault.com/articles/circles>

For a javascript web plugin of a continuous version of the Inclusion of Other in the Self (IOS) task (instead of the pen and paper version), for experiments during data collection, rather than data analysis, please see: <https://github.com/jspsych/jspsych-contrib/tree/main/packages/plugin-ios>

## 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)
plot <- overlap_circle(3.5)
ggplot2::ggsave(plot,
  file = tempfile(fileext = ".pdf"), width = 7,
  height = 7, unit = "in", dpi = 300
)
# Change for your own desired path

```

---

plot\_means\_over\_time *Easy scatter plots over multiple times (T1, T2, T3)*

---

## Description

Make nice scatter plots over multiple times (T1, T2, T3) easily.

## Usage

```

plot_means_over_time(
  data,
  response,
  group,
  groups.order = "none",
  error_bars = TRUE,
  ytitle = NULL,
  legend.title = "",
  significance_stars,
  significance_stars_x,
  significance_stars_y,
  significance_bars_x,
  print_table = FALSE,
  verbose = FALSE
)

```

## Arguments

data	The data frame.
response	The dependent variable to be plotted (e.g., c("variable_T1", "variable_T2", "variable_T3"), etc.).

group	The group by which to plot the variable
groups.order	Specifies the desired display order of the groups on the legend. Either provide the levels directly, or a string: "increasing" or "decreasing", to order based on the average value of the variable on the y axis, or "string.length", to order from the shortest to the longest string (useful when working with long string names). Defaults to "none".
errorBars	Logical, whether to include 95% confidence intervals for means.
ylabel	An optional x-axis label, if desired. If NULL, will take the variable name of the first variable in response, and keep only the part of the string before an underscore or period.
legend.title	The desired legend title.
significance_stars	Vector of significance stars to display on the plot (e.g., c("*", "**", "***")).
significance_stars_x	Vector of where on the x-axis significance stars should appear on the plot (e.g., c(2.2, 3.2, 4.2)).
significance_stars_y	Vector of where on the y-axis significance stars should appear on the plot. The logic here is different than previous arguments. Rather than providing actual coordinates, we provide a list object with structure group 1, group 2, and time of comparison, e.g., list(c("group1", "group2", time = 2), c("group1", "group3", time = 3), c("group2", "group3", time = 4)).
significance_bars_x	Vector of where on the x-axis vertical significance bars should appear on the plot (e.g., c(2:4)).
printTable	Logical, whether to also print the computed table.
verbose	Logical, whether to also print a note regarding the meaning of the error bars.

## Details

Error bars are calculated using the method of Morey (2008) through `Rmisc::summarySEwithin()`, but raw means are plotted instead of the normed means. For more information, visit: [http://www.cookbook-r.com/Graphs/Plotting\\_means\\_and\\_error\\_bars\\_\(ggplot2\)](http://www.cookbook-r.com/Graphs/Plotting_means_and_error_bars_(ggplot2)).

## Value

A scatter plot of class `ggplot`.

## References

Morey, R. D. (2008). Confidence intervals from normalized data: A correction to Cousineau (2005). *Tutorials in Quantitative Methods for Psychology*, 4(2), 61-64. doi:10.20982/tqmp.04.2.p061

**Examples**

```

data <- mtcars
names(data)[6:3] <- paste0("T", 1:4, "_var")
plot_means_over_time(
  data = data,
  response = names(data)[6:3],
  group = "cyl",
  groups.order = "decreasing"
)

# Add significance stars/bars
plot_means_over_time(
  data = data,
  response = names(data)[6:3],
  group = "cyl",
  significance_bars_x = c(3.15, 4.15),
  significance_stars = c("*", "***"),
  significance_stars_x = c(3.25, 4.5),
  significance_stars_y = list(
    c("4", "8", time = 3),
    c("4", "8", time = 4)
  )
)
# significance_stars_y: List with structure: list(c("group1", "group2", time))

```

---

plot\_outliers

*Visually check outliers (dot plot)*


---

**Description**

Easily and visually check outliers through a dot plot with accompanying reference lines at  $\pm 3$  MAD or SD. When providing a group, data are group-mean centered and standardized (based on MAD or SD); if no group is provided, data are simply standardized.

**Usage**

```

plot_outliers(
  data,
  group = NULL,
  response,
  method = "mad",
  criteria = 3,
  colours,
  xlabel = NULL,
  ylabel = NULL,
  xtitle = NULL,
  has.ylabels = TRUE,

```

```

    has.xlabels = TRUE,
    ymin,
    ymax,
    yby = 1,
    ...
  )

```

### Arguments

data	The data frame.
group	The group by which to plot the variable.
response	The dependent variable to be plotted.
method	Method to identify outliers, either (e.g., 3) median absolute deviations ("mad", default) or standard deviations ("sd").
criteria	How many MADs (or standard deviations) to use as threshold (default is 3).
colours	Desired colours for the plot, if desired.
xlabels	The individual group labels on the x-axis.
ytile	An optional y-axis label, if desired.
xtile	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.
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.
...	Other arguments passed to <code>ggplot2::geom_dotplot</code> .

### Value

A dot plot of class `ggplot`, by group.

### See Also

Other functions useful in assumption testing: Tutorial: <https://rempsysc.remi-theriault.com/articles/assumptions>

### Examples

```

# Make the basic plot
plot_outliers(
  airquality,
  group = "Month",
  response = "Ozone"
)

plot_outliers(
  airquality,

```

```
response = "Ozone",  
method = "sd"  
)
```

---

scale\_mad

*Standardize based on the absolute median deviation*

---

### Description

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

### Usage

```
scale_mad(x)
```

### Arguments

x                    The vector to be scaled.

### Details

The function subtracts the median to each observation, and then divides the outcome by the MAD. This is analogous to regular standardization which subtracts the mean to each observation, and then divides the outcome by the standard deviation.

For the *easystats* equivalent, use: `datawizard::standardize(x, robust = TRUE)`.

### Value

A numeric vector of standardized data.

### References

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>

### 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.

**Usage**

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

**Arguments**

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

**Details**

For the *easystats* equivalent, use: `datawizard::winsorize(x, method = "zscore", threshold = 3, robust = TRUE)`.

**Value**

A numeric vector of winsorized data.

**References**

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>

**Examples**

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

# Index

- \* **APA**
  - nice\_table, 38
- \* **Excel**
  - cormatrix\_excel, 3
- \* **MAD**
  - find\_mad, 6
  - scale\_mad, 56
  - winsorize\_mad, 57
- \* **NA**
  - nice\_na, 22
- \* **QQ**
  - nice\_normality, 24
  - nice\_qq, 26
- \* **Venn**
  - overlap\_circle, 51
- \* **allocation**
  - nice\_randomize, 27
- \* **assumptions**
  - nice\_assumptions, 10
- \* **comparisons**
  - nice\_contrasts, 11
  - nice\_lm\_contrasts, 17
- \* **conditions**
  - nice\_randomize, 27
- \* **contrasts**
  - nice\_contrasts, 11
  - nice\_lm\_contrasts, 17
- \* **correlation**
  - cormatrix\_excel, 3
  - format\_value, 7
- \* **density**
  - nice\_density, 13
  - nice\_normality, 24
- \* **design**
  - nice\_randomize, 27
- \* **diagrams**
  - overlap\_circle, 51
- \* **differences**
  - nice\_contrasts, 11
  - nice\_lm\_contrasts, 17
  - nice\_t\_test, 41
- \* **distribution**
  - nice\_normality, 24
  - nice\_qq, 26
- \* **dotplot**
  - plot\_outliers, 54
- \* **duplicates**
  - best\_duplicate, 2
  - extract\_duplicates, 5
- \* **experimental**
  - nice\_randomize, 27
- \* **formatting**
  - format\_value, 7
- \* **group**
  - nice\_contrasts, 11
  - nice\_lm\_contrasts, 17
  - nice\_t\_test, 41
- \* **guidelines**
  - nice\_na, 22
- \* **interaction**
  - nice\_lm, 15
  - nice\_lm\_slopes, 18
  - nice\_mod, 20
  - nice\_slopes, 36
- \* **linear**
  - nice\_assumptions, 10
- \* **matrix**
  - cormatrix\_excel, 3
- \* **mean**
  - find\_mad, 6
  - scale\_mad, 56
  - winsorize\_mad, 57
- \* **median**
  - find\_mad, 6
  - scale\_mad, 56
  - winsorize\_mad, 57
- \* **merging**
  - overlap\_circle, 51

- \* **missing**
    - nice\_na, 22
  - \* **moderation**
    - nice\_lm, 15
    - nice\_lm\_slopes, 18
    - nice\_mod, 20
    - nice\_slopes, 36
  - \* **normality**
    - nice\_density, 13
    - nice\_normality, 24
    - nice\_qq, 26
  - \* **normalization**
    - find\_mad, 6
    - scale\_mad, 56
    - winsorize\_mad, 57
  - \* **outliers**
    - find\_mad, 6
    - scale\_mad, 56
    - winsorize\_mad, 57
  - \* **overlap**
    - overlap\_circle, 51
  - \* **pairwise**
    - nice\_contrasts, 11
    - nice\_lm\_contrasts, 17
  - \* **planned**
    - nice\_contrasts, 11
    - nice\_lm\_contrasts, 17
  - \* **plots**
    - nice\_normality, 24
    - nice\_qq, 26
    - nice\_scatter, 29
    - nice\_violin, 46
    - plot\_outliers, 54
  - \* **psychology**
    - overlap\_circle, 51
  - \* **p**
    - format\_value, 7
  - \* **randomization**
    - nice\_randomize, 27
  - \* **random**
    - nice\_randomize, 27
  - \* **regression**
    - nice\_assumptions, 10
    - nice\_lm, 15
    - nice\_lm\_slopes, 18
    - nice\_mod, 20
    - nice\_slopes, 36
  - \* **reverse**
    - nice\_reverse, 28
  - \* **r**
    - format\_value, 7
  - \* **scatter**
    - nice\_scatter, 29
  - \* **scoring**
    - nice\_reverse, 28
  - \* **self-other**
    - overlap\_circle, 51
  - \* **simple**
    - nice\_slopes, 36
  - \* **slopes**
    - nice\_slopes, 36
  - \* **social**
    - overlap\_circle, 51
  - \* **standardization**
    - find\_mad, 6
    - scale\_mad, 56
    - winsorize\_mad, 57
  - \* **statistical**
    - nice\_assumptions, 10
  - \* **style**
    - nice\_table, 38
  - \* **t-test**
    - nice\_t\_test, 41
  - \* **table**
    - nice\_table, 38
  - \* **values**
    - nice\_na, 22
  - \* **value**
    - format\_value, 7
  - \* **variance**
    - nice\_var, 43
    - nice\_varplot, 44
  - \* **violations**
    - nice\_assumptions, 10
  - \* **violin**
    - nice\_violin, 46
- attributes(), 6
- best\_duplicate, 2
- bootES::bootES, 11, 12, 17
- cormatrix\_excel, 3, 4
- correlation::cormatrix\_to\_excel, 4
- correlation::cormatrix\_to\_excel(), 4
- correlation::correlation(), 4
- datawizard::data\_duplicated(), 3

datawizard::data\_unique(), 5  
 datawizard::reverse(), 28  
 dplyr::distinct(), 5  
 duplicated(), 5  
  
 effectsize::interpret\_r2\_semipartial(),  
     16, 19, 21, 37  
 effectsize::r2\_semipartial, 16, 19, 21,  
     37  
 emmeans::contrast(), 12, 17  
 extract\_duplicates, 5  
  
 find\_mad, 6  
 format\_d(format\_value), 7  
 format\_p(format\_value), 7  
 format\_r(format\_value), 7  
 format\_value, 7  
 format\_value(), 7  
  
 get\_dep\_version, 8  
 ggplot2::geom\_dotplot, 55  
 grouped\_bar\_chart, 9  
  
 hist(), 14  
  
 insight::format\_value(), 8  
 install\_if\_not\_installed, 10  
  
 lm(), 10, 15, 16, 18, 19, 21, 37  
  
 modelbased::estimate\_contrasts(), 12,  
     17  
  
 nice\_assumptions, 10, 14, 25, 26, 44, 45  
 nice\_contrasts, 11, 18  
 nice\_density, 10, 13, 25, 26, 44, 45  
 nice\_density(), 25  
 nice\_lm, 15, 19, 21, 37  
 nice\_lm\_contrasts, 12, 17  
 nice\_lm\_slopes, 16, 18, 21, 37  
 nice\_mod, 16, 19, 20, 37  
 nice\_na, 22  
 nice\_normality, 10, 14, 24, 26, 44, 45  
 nice\_normality(), 25  
 nice\_qq, 10, 14, 25, 26, 44, 45  
 nice\_qq(), 25  
 nice\_randomize, 27  
 nice\_reverse, 28  
 nice\_scatter, 29, 48  
 nice\_slopes, 16, 19, 21, 36  
  
 nice\_t\_test, 41  
 nice\_table, 15, 38  
 nice\_table(), 19, 21, 36  
 nice\_var, 10, 14, 25, 26, 43, 45  
 nice\_varplot, 10, 14, 25, 26, 44, 44  
 nice\_violin, 32, 46  
  
 overlap\_circle, 50  
  
 plot\_means\_over\_time, 52  
 plot\_outliers, 54  
  
 rcompanion\_groupwiseMean, 48  
 report::report(), 16, 19, 21, 37, 42  
 Rmisc::summarySEwithin(), 53  
  
 scale\_mad, 56  
 scale\_mad(), 6  
 see::geom\_violindot(), 48  
  
 t.test(), 42  
  
 winsorize\_mad, 57