Package 'ggpubr'

July 6, 2026

Type Package

Title 'ggplot2' Based Publication Ready Plots

Version 1.0.0

Date 2026-07-06

Author Alboukadel Kassambara [aut, cre],
Laszlo Erdey [ctb] (Faculty of Economics and Business, University of Debrecen, Hungary)

Maintainer Alboukadel Kassambara <alboukadel.kassambara@gmail.com>

Description The 'ggplot2' package is excellent and flexible for elegant data visualization in R. However the default generated plots require some formatting before we can send them for publication. Furthermore, to customize a 'ggplot', the syntax is opaque and this raises the level of difficulty for researchers with no advanced R programming skills. 'ggpubr' provides some easy-to-use functions for creating and customizing 'ggplot2'-based publication ready plots. This version includes modern R ecosystem compatibility updates and customizable p-value formatting presets (APA, AMA, NEJM, Lancet, GraphPad, and scientific notation) for publication workflows, plus robust sparse-subset handling in statistical annotation layers such as 'stat_compare_means()' and 'geom_pwc()', with informative per-group skip diagnostics for non-comparable subsets.

License GPL (>= 2)

LazyData TRUE

Encoding UTF-8

Depends R (>= 4.1.0), ggplot2 (>= 3.5.2)

Imports ggrepel (>= 0.9.2), grid, ggsci, stats, utils, tidyr (>= 1.3.2), purrr, dplyr (>= 1.2.0), cowplot (>= 1.2.0), ggsignif, scales, gridExtra, glue, polynom, rlang (>= 1.1.7), rstatix (>= 1.0.0), tibble, magrittr

Suggests grDevices, knitr, RColorBrewer, gtable, testthat

URL https://rpkgs.datanovia.com/ggpubr/, https://github.com/kassambara/ggpubr

BugReports https://github.com/kassambara/ggpubr/issues

RoxygenNote 7.3.3

Collate 'utilities_color.R' 'utilities_base.R' 'desc_statby.R' 'utilities.R' 'add_summary.R' 'annotate_figure.R' 'as_ggplot.R' 'as_npc.R' 'axis_scale.R' 'background_image.R' 'bgcolor.R' 'border.R' 'p_format_utils.R' 'compare_means.R' 'create_aes.R' 'diff_express.R' 'facet.R' 'font.R' 'gene_citation.R' 'gene_expression.R' 'geom_bracket.R' 'geom_exec.R' 'utils-aes.R' 'utils_stat_test_label.R' 'geom_pwc.R' 'get_breaks.R' 'get_coord.R' 'get_legend.R' 'get_palette.R' 'ggadd.R' 'ggadjust_pvalue.R' 'ggarrange.R' 'ggballoonplot.R' 'ggpar.R' 'ggbarplot.R' 'ggboxplot.R' 'ggdensity.R' 'ggpie.R' 'ggdonutchart.R' 'stat_conf_ellipse.R' 'stat_chull.R' 'ggdotchart.R' 'ggdotplot.R' 'ggecdf.R' 'ggerrorplot.R' 'ggexport.R' 'gghistogram.R' 'ggline.R' 'ggmaplot.R' 'ggpaired.R' 'ggparagraph.R' 'ggpubr-package.R' 'ggpubr_args.R' 'ggpubr_options.R' 'ggqqplot.R' 'utilities_label.R' 'stat_cor.R' 'stat_stars.R' 'ggscatter.R' 'ggscatterhist.R' 'ggstripchart.R' 'ggsummarystats.R' 'ggtext.R' 'ggtexttable.R' 'ggviolin.R' 'gradient_color.R' 'grids.R' 'npc_to_data_coord.R' 'reexports.R' 'rotate.R' 'rotate_axis_text.R' 'rremove.R' 'set_palette.R' 'shared_docs.R' 'show_line_types.R' 'show_point_shapes.R' 'stat_anova_test.R' 'stat_central_tendency.R' 'stat_compare_means.R' 'stat_friedman_test.R' 'stat_kruskal_test.R' 'stat_mean.R' 'stat_overlay_normal_density.R' 'stat_pvalue_manual.R' 'stat_regline_equation.R' 'stat_welch_anova_test.R' 'text_grob.R' 'theme_pubr.R' 'theme_transparent.R' 'utils-geom-signif.R' 'utils-pipe.R' 'utils-tidyr.R' 'zzz.R'

NeedsCompilation no

Repository CRAN

Date/Publication 2026-07-06 15:00:02 UTC

Contents

add_summary 4
annotate_figure 7
as_ggplot 9
as_npc 9
axis_scale 11
background_image 12
bgcolor 13
border 13
clean_lock_files 14
compare_means 15
create_aes 19
create_p_label 20

desc_statby . . . 21
diff_express . . . 22
facet . . . 23
font . . . 25
format_p_value . . . 27
gene_citation . . . 29
gene_expression . . . 30
geom_exec . . . 30
get_breaks . . . 31
get_coord . . . 32
get_legend . . . 34
get_palette . . . 35
get_p_format_style . . . 36
ggadd . . . 37
ggadjust_pvalue . . . 39
ggarrange . . . 42
ggballoonplot . . . 45
ggbarplot . . . 48
ggboxplot . . . 54
ggdensity . . . 59
ggdonutchart . . . 63
ggdotchart . . . 65
ggdotplot . . . 69
ggecdf . . . 73
ggerrorplot . . . 76
ggexport . . . 79
gghistogram . . . 81
ggline . . . 85
ggmaplot . . . 90
ggpaired . . . 93
ggpar . . . 96
ggparagraph . . . 100
ggpie . . . 102
ggpubr_args . . . 104
ggpubr_options . . . 106
ggqqplot . . . 106
ggscatter . . . 109
ggscatterhist . . . 114
ggstripchart . . . 117
ggsummarytable . . . 121
ggtext . . . 125
ggtexttable . . . 127
ggviolin . . . 136
gradient_color . . . 142
grids . . . 143
list_p_format_styles . . . 144
npc_to_data_coord . . . 144
rotate . . . 145

rotate_axis_text . . . 145
rremove . . . 146
set_palette . . . 147
show_line_types . . . 149
show_point_shapes . . . 149
stat_anova_test . . . 150
stat_bracket . . . 156
stat_central_tendency . . . 161
stat_chull . . . 164
stat_compare_means . . . 166
stat_conf_ellipse . . . 172
stat_cor . . . 174
stat_friedman_test . . . 179
stat_kruskal_test . . . 184
stat_mean . . . 189
stat_overlay_normal_density . . . 191
stat_pvalue_manual . . . 193
stat_pwc . . . 197
stat_regline_equation . . . 206
stat_stars . . . 210
stat_welch_anova_test . . . 212
text_grob . . . 216
theme_pubr . . . 218
theme_transparent . . . 219

Index **221**

`add_summary` *Add Summary Statistics onto a ggplot.*

Description

add summary statistics onto a ggplot.

Usage

```
add_summary(
  p,
  fun = "mean_se",
  error.plot = "pointrange",
  color = "black",
  fill = "white",
  group = 1,
  width = NULL,
  shape = 19,
  size = 1,
  linewidth = size,
  linetype = 1,
```

```
  show.legend = NA,
  ci = 0.95,
  data = NULL,
  position = position_dodge(0.8)
)

mean_se_(x, error.limit = "both")

mean_sd(x, error.limit = "both")

mean_ci(x, ci = 0.95, error.limit = "both")

mean_range(x, error.limit = "both")

median_iqr(x, error.limit = "both")

median_hilow_(x, ci = 0.95, error.limit = "both")

median_q1q3(x, error.limit = "both")

median_mad(x, error.limit = "both")

median_range(x, error.limit = "both")
```

Arguments

p	a ggplot on which you want to add summary statistics.
fun	a function that is given the complete data and should return a data frame with variables ymin, y, and ymax. Allowed values are one of: "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range".
error.plot	plot type used to visualize error. Allowed values are one of `c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar","lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange","lower_linerange")`. Default value is "pointrange".
color	point or outline color.
fill	fill color. Used only when `error.plot = "crossbar"`.
group	grouping variable. Allowed values are 1 (for one group) or a character vector specifying the name of the grouping variable. Used only for adding statistical summary per group.
width	numeric value between 0 and 1 specifying bar or box width. Example width = 0.8. Used only when `error.plot` is one of c("crossbar", "errorbar").
shape	point shape. Allowed values can be displayed using the function `show_point_shapes()`.
size	numeric value in [0-1] specifying point and line size.
linewidth	numeric value in [0-1] specifying line width.
linetype	line type.

show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
ci	the percent range of the confidence interval (default is 0.95).
data	a data.frame to be displayed. If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.
position	position adjustment, either as a string, or the result of a call to a position adjustment function. Used to adjust position for multiple groups.
x	a numeric vector.
error.limit	allowed values are one of ("both", "lower", "upper", "none") specifying whether to plot the lower and/or the upper limits of error interval.

Functions

- add_summary(): add summary statistics onto a ggplot.
- mean_se_(): returns the mean and the error limits defined by the standard error. We used the name mean_se_() to avoid masking mean_se().
- mean_sd(): returns the mean and the error limits defined by the standard deviation.
- mean_ci(): returns the mean and the error limits defined by the confidence interval.
- mean_range(): returns the mean and the error limits defined by the range = max - min.
- median_iqr(): returns the median and the error limits defined by the interquartile range.
- median_hilow_(): computes the sample median and a selected pair of outer quantiles having equal tail areas. This function is a reformatted version of Hmisc::smedian.hilow(). The confidence limits are computed as follows: lower.limits = (1-ci)/2 percentiles; upper.limits = (1+ci)/2 percentiles. By default (ci = 0.95), the 2.5th and the 97.5th percentiles are used as the lower and the upper confidence limits, respectively. If you want to use the 25th and the 75th percentiles as the confidence limits, then specify ci = 0.5 or use the function median_q1q3().
- median_q1q3(): computes the sample median and, the 25th and 75th percentiles. Wrapper around the function median_hilow_() using ci = 0.5.
- median_mad(): returns the median and the error limits defined by the median absolute deviation.
- median_range(): returns the median and the error limits defined by the range = max - min.

Examples

```
# Basic violin plot
p <- ggviolin(ToothGrowth, x = "dose", y = "len", add = "none")
p

# Add mean_sd
add_summary(p, "mean_sd")
```

annotate_figure *Annotate Arranged Figure*

Description

Annotate figures including: i) ggplots, ii) arranged ggplots from ggarrange(), grid.arrange() and plot_grid().

Usage

```
annotate_figure(
  p,
  top = NULL,
  bottom = NULL,
  left = NULL,
  right = NULL,
  fig.lab = NULL,
  fig.lab.pos = c("top.left", "top", "top.right", "bottom.left", "bottom",
    "bottom.right"),
  fig.lab.size,
  fig.lab.face,
  column.titles = NULL,
  row.titles = NULL
)
```

Arguments

p	(arranged) ggplots.
top, bottom, left, right	optional string, or grob.
fig.lab	figure label (e.g.: "Figure 1").
fig.lab.pos	position of the figure label, can be one of "top.left", "top", "top.right", "bottom.left", "bottom", "bottom.right". Default is "top.left".
fig.lab.size	optional size of the figure label.
fig.lab.face	optional font face of the figure label. Allowed values include: "plain", "bold", "italic", "bold.italic".
column.titles, row.titles	optional character vector giving one title per column / per row of the arranged figure, adding titles above each column and to the left of each row (useful for publication grids, see ggarrange()). Can also be a list of grobs (e.g. built with text_grob()) for full control over styling; a character vector is rendered in bold using the current theme text settings. The titles are evenly spaced and therefore assume equal-sized columns / rows (the ggarrange() default), with one entry per column / row. row.titles are rotated 90 degrees.

Author(s)

Laszlo Erdey <erdey.laszlo@econ.unideb.hu>

See Also

ggarrange()

Examples

```
data("ToothGrowth")
df <- ToothGrowth
df$dose <- as.factor(df$dose)

# Create some plots
# :::::::::::::::::::::::::::::::::::::::::::::::::
# Box plot
bxp <- ggboxplot(df,
  x = "dose", y = "len",
  color = "dose", palette = "jco"
)
# Dot plot
dp <- ggdotplot(df,
  x = "dose", y = "len",
  color = "dose", palette = "jco"
)
# Density plot
dens <- ggdensity(df, x = "len", fill = "dose", palette = "jco")

# Arrange and annotate
# :::::::::::::::::::::::::::::::::::::::::::::::::
figure <- ggarrange(bxp, dp, dens, ncol = 2, nrow = 2)
annotate_figure(figure,
  top = text_grob("Visualizing Tooth Growth", color = "red", face = "bold", size = 14),
  bottom = text_grob("Data source: \n ToothGrowth data set",
    color = "blue",
    hjust = 1, x = 1, face = "italic", size = 10
  ),
  left = text_grob("Figure arranged using ggpubr", color = "green", rot = 90),
  right = text_grob(bquote("Superscript: (" * kg ~ NH[3] ~ ha^-1 ~ yr^-1 * ")"), rot = 90),
  fig.lab = "Figure 1", fig.lab.face = "bold"
)

# Add a title to each column and each row of an arranged grid
# :::::::::::::::::::::::::::::::::::::::::::::::::
grid <- ggarrange(bxp, dp, dens, bxp, ncol = 2, nrow = 2)
annotate_figure(grid,
  column.titles = c("Column 1", "Column 2"),
  row.titles = c("Row 1", "Row 2")
)
```

as_ggplot *Storing grid.arrange() arrangeGrob() and plots*

Description

Transform the output of `arrangeGrob()` and `grid.arrange()` to an object of class ggplot.

Usage

```
as_ggplot(x)
```

Arguments

x	an object of class gtable or grob as returned by the functions `arrangeGrob()` and `grid.arrange()`.

Value

an object of class ggplot.

Examples

```
# Create some plots
bxp <- ggboxplot(iris, x = "Species", y = "Sepal.Length")
vp <- ggviolin(iris,
  x = "Species", y = "Sepal.Length",
  add = "mean_sd"
)

# Arrange the plots in one page
# Returns a gtable (grob) object
library(gridExtra)
gt <- arrangeGrob(bxp, vp, ncol = 2)

# Transform to a ggplot and print
as_ggplot(gt)
```

as_npc *Convert Character Coordinates into Normalized Parent Coordinates (NPC)*

Description

Convert character coordinates to npc units and shift positions to avoid overlaps when grouping is active. If numeric validate npc values.

Usage

```
as_npc(
  value,
  group = 1L,
  step = 0.1,
  margin.npc = 0.05,
  axis = c("xy", "x", "y")
)

as_npcx(value, group = 1L, step = 0.1, margin.npc = 0.05)

as_npcy(value, group = 1L, step = 0.1, margin.npc = 0.05)
```

Arguments

value	numeric (in [0-1]) or character vector of coordinates. If character, should be one of c('right', 'left', 'bottom', 'top','center', 'centre', 'middle').
group	integer ggplot's group id. Used to shift coordinates to avoid overlaps.
step	numeric value in [0-1]. The step size for shifting coordinates in npc units. Considered as horizontal step for x-axis and vertical step for y-axis. For y-axis, the step value can be negative to reverse the order of groups.
margin.npc	numeric [0-1] The margin added towards the nearest plotting area edge when converting character coordinates into npc.
axis	the concerned axis . Should be one of c("xy", "x", "y").

Details

the as_npc() function is an adaptation from ggpmisc::compute_npc().

Value

A numeric vector with values in the range [0-1] representing npc coordinates.

Functions

- as_npc(): converts x or y coordinate values into npc. Input values should be numeric or one of the following values c('right', 'left','bottom', 'top', 'center', 'centre', 'middle').
- as_npcx(): converts x coordinate values into npc. Input values should be numeric or one of the following values c('right', 'left','center', 'centre', 'middle'). Wrapper around as_npc(axis = "x").
- as_npcy(): converts y coordinate values into npc. Input values should be numeric or one of the following values c('bottom', 'top','center', 'centre', 'middle'). Wrapper around as_npc(axis = "y").

See Also

npc_to_data_coord, get_coord.

Examples

```
as_npc(c("left", "right"))
as_npc(c("top", "right"))
```

`axis_scale` *Change Axis Scale: log2, log10 and more*

Description

Change axis scale.

- `xscale`: change x axis scale.
- `yscale`: change y axis scale.

Usage

```
xscale(.scale, .format = FALSE)

yscale(.scale, .format = FALSE)
```

Arguments

`.scale`	axis scale. Allowed values are one of c("none", "log2", "log10", "sqrt", "percent", "dollar", "scientific"); e.g.: .scale="log2".
`.format`	logical value. If TRUE, axis tick mark labels will be formatted when .scale = "log2" or "log10".

Examples

```
# Basic scatter plots
data(cars)
p <- ggscatter(cars, x = "speed", y = "dist")
p

# Set log scale
p + yscale("log2", .format = TRUE)
```

background_image *Add Background Image to ggplot2*

Description

Add background image to ggplot2.

Usage

```
background_image(raster.img)
```

Arguments

raster.img	raster object to display, as returned by the function readPNG()[in png package] and readJPEG() [in jpeg package].

Author(s)

Laszlo Erdey <erdey.laszlo@econ.unideb.hu>

Examples

```
## Not run:
install.packages("png")

# Import the image
img.file <- system.file(file.path("images", "background-image.png"),
  package = "ggpubr"
)
img <- png::readPNG(img.file)

# Plot with background image
ggplot(iris, aes(Species, Sepal.Length)) +
  background_image(img) +
  geom_boxplot(aes(fill = Species), color = "white") +
  fill_palette("jco")

## End(Not run)
```

bgcolor *Change ggplot Panel Background Color*

Description

Change ggplot panel background color.

Usage

```
bgcolor(color)
```

Arguments

color	background color.

See Also

border().

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot
p <- ggboxplot(df, x = "dose", y = "len")
p

# Change panel background color
p +
  bgcolor("#BFD5E3") +
  border("#BFD5E3")
```

border *Set ggplot Panel Border Line*

Description

Change or set ggplot panel border.

Usage

```
border(color = "black", size = 0.8, linetype = NULL)
```

Arguments

color	border line color.
size	numeric value specifying border line size.
linetype	line type. An integer (0:8), a name (blank, solid, dashed, dotted, dotdash, longdash, twodash). Sess show_line_types.

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot
p <- ggboxplot(df, x = "dose", y = "len")
p

# Add border
p + border()
```

---

clean_lock_files *Clean up stale package lock files*

Description

Removes stale lock directories that can prevent package installation.

Lock files are created during package installation and should be automatically removed when installation completes. If installation is interrupted (e.g., by closing R or a crash), these lock files may remain and block future installations.

Usage

```
clean_lock_files(package = "ggpubr", lib = .libPaths()[1], ask = TRUE)
```

Arguments

package	Character string specifying which package lock to remove. Default is "ggpubr". Use "all" to remove all lock files.
lib	Library path to check. Default is the first library in .libPaths().
ask	Logical. If TRUE (default), asks for confirmation before removing.

Value

Invisibly returns TRUE if files were removed, FALSE otherwise.

Examples

```
## Not run:
# Remove ggpubr lock file
clean_lock_files()

# Remove all lock files
clean_lock_files("all")

# Remove without confirmation
clean_lock_files(ask = FALSE)

## End(Not run)
```

---

compare_means *Comparison of Means*

Description

Performs one or multiple mean comparisons.

Usage

```
compare_means(
  formula,
  data,
  method = "wilcox.test",
  paired = FALSE,
  id = NULL,
  group.by = NULL,
  ref.group = NULL,
  symnum.args = list(),
  p.adjust.method = "holm",
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  signif.cutoffs = NULL,
  signif.symbols = NULL,
  ns.symbol = "ns",
  use.four.stars = FALSE,
  ...
)
```

Arguments

formula	a formula of the form x ~ group where x is a numeric variable giving the data values and group is a factor with one or multiple levels giving the corresponding groups. For example, formula = TP53 ~ cancer_group. It's also possible to perform the test for multiple response variables at the same time. For example, formula = c(TP53, PTEN) ~ cancer_group.
data	a data.frame containing the variables in the formula.
method	the type of test. Default is wilcox.test. Allowed values include: • t.test (parametric) and wilcox.test (non-parametric). Perform comparison between two groups of samples. If the grouping variable contains more than two levels, then a pairwise comparison is performed. • anova (parametric) and kruskal.test (non-parametric). Perform one-way ANOVA test comparing multiple groups.
paired	a logical indicating whether you want a paired test. Used only in t.test and in wilcox.test.
id	optional character string naming a column that identifies matched subjects for a **paired** comparison (method = "t.test" or "wilcox.test"). By default (id = NULL) a paired test pairs observations by *row order*, so a p-value can be wrong if the data are not sorted so that the compared groups align. Providing id pairs the observations by subject id instead (row-order independent), using only the complete pairs (per-comparison pairwise deletion, via **rstatix**). It works for a two-group, a pairwise (more than two groups) and a ref.group comparison; it is an error to combine id with anova/kruskal.test or with ref.group = ".all.".
group.by	a character vector containing the name of grouping variables.
ref.group	a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group). ref.group can be also ".all.". In this case, each of the grouping variable levels is compared to all (i.e. basemean).
symnum.args	a list of arguments to pass to the function symnum for symbolic number coding of p-values. For example, symnum.args <- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns")). In other words, we use the following convention for symbols indicating statistical significance: • ns: p > 0.05 • *: p <= 0.05 • **: p <= 0.01 • ***: p <= 0.001 • ****: p <= 0.0001 Note: If signif.cutoffs is provided, it takes precedence over symnum.args.

p.adjust.method
: method for adjusting p values (see p.adjust). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

  Note that, when the formula contains multiple variables, the p-value adjustment is done independently for each variable.

p.format.style
: character string specifying the p-value formatting style. One of: "default" (backward compatible, uses scientific notation), "apa" (APA style, no leading zero), "nejm" (NEJM style), "lancet" (Lancet style), "ama" (AMA style), "graphpad" (GraphPad style), or "scientific" (scientific notation for GWAS). See list_p_format_styles for details.

p.digits
: integer specifying the number of decimal places for p-values. If provided, overrides the style default.

p.leading.zero
: logical indicating whether to include leading zero before decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

p.min.threshold
: numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold". If NULL, the selected style's default threshold is used; styles without a threshold show exact values. If provided, overrides the style default.

p.decimal.mark
: character string to use as the decimal mark. If NULL, uses getOption("OutDec").

signif.cutoffs
: numeric vector of p-value cutoffs in descending order for assigning significance symbols. For example, c(0.10, 0.05, 0.01) means p < 0.10 gets "*", p < 0.05 gets "**", p < 0.01 gets "***". If use.four.stars = TRUE, can include a fourth level (e.g., c(0.10, 0.05, 0.01, 0.001) for "****" at p < 0.001). Default is NULL, which uses the package defaults (backward compatible).

signif.symbols
: character vector of symbols corresponding to signif.cutoffs. If NULL, auto-generated as "*", "**", "***" (and "****" if use.four.stars = TRUE). Must have the same length as signif.cutoffs.

ns.symbol
: character string for non-significant results. Default is "ns". Use "" (empty string) to show nothing for non-significant results.

use.four.stars
: logical. If TRUE and signif.symbols is NULL, allows four stars (****) for the most significant level when signif.cutoffs has 4 levels. Default is FALSE.

...
: Other arguments to be passed to the test function.

Value

a data frame with the following columns:

- .y.: the y variable used in the test.
- group1,group2: the compared groups in the pairwise tests. Available only when method = "t.test" or method = "wilcox.test".
- p: the p-value.

- p.adj: the adjusted p-value. Default for p.adjust.method = "holm".
- p.format: the formatted p-value.
- p.format.signif: the formatted p-value with significance symbols.
- p.signif: the significance level.
- method: the statistical test used to compare groups.

Significance letters (compact letter display)

To label groups with letters instead of p-values or stars - groups that share a letter are not significantly different - compute the pairwise comparisons and derive the letters with rstatix::add_cld() (available without any extra package, as rstatix is already a dependency), then place them with geom_text():

```
library(ggpubr)
library(rstatix)
library(dplyr)
df <- ToothGrowth
df$dose <- factor(df$dose)

# all-pairwise comparisons, then compact letters (columns: group, cld)
cld <- df %>% tukey_hsd(len ~ dose) %>% add_cld()

# one letter per group, placed above each box
ypos <- df %>% group_by(dose) %>%
  summarise(y.position = max(len) + 2, .groups = "drop")
cld$y.position <- ypos$y.position[match(cld$group, as.character(ypos$dose))]

ggboxplot(df, "dose", "len") +
  geom_text(data = cld, aes(x = group, y = y.position, label = cld))
```

add_cld() expects an all-pairwise result (tukey_hsd(), dunn_test(), games_howell_test(), pairwise wilcox_test()/t_test(), or compare_means()); it is not defined for a single ref.group comparison.

To instead flag each treatment by which of several controls it differs from (e.g. "a" for a negative control and "b" for a positive control), run one comparison per control and assemble the letters:

```
trts <- c("trtA", "trtB", "trtC")
cn <- compare_means(value ~ group, df, ref.group = "neg.ctrl")
cp <- compare_means(value ~ group, df, ref.group = "pos.ctrl")
pv <- function(cc, g) dplyr::filter(cc, group1 == g | group2 == g)$p[1]
lab <- sapply(trts, function(g)
  paste0(if (pv(cn, g) < .05) "a" else "",
         if (pv(cp, g) < .05) "b" else ""))
# then place `lab` above each treatment with geom_text()
```

Examples

```
# Load data
# :::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth

# One-sample test
# ::::::::::::::::::::::::::::::::::::::::::
compare_means(len ~ 1, df, mu = 0)

# Two-samples unpaired test
# ::::::::::::::::::::::::::::::::::::::::::
compare_means(len ~ supp, df)

# Two-samples paired test
# ::::::::::::::::::::::::::::::::::::::::::
compare_means(len ~ supp, df, paired = TRUE)

# Paired test pairing by a subject id column (row-order independent)
# ::::::::::::::::::::::::::::::::::::::::::
df$id <- rep(1:30, 2) # pairs the two supp levels by subject
compare_means(len ~ supp, df, paired = TRUE, id = "id")

# Compare supp levels after grouping the data by "dose"
# ::::::::::::::::::::::::::::::::::::::::::
compare_means(len ~ supp, df, group.by = "dose")

# pairwise comparisons
# ::::::::::::::::::::::::::::::::::::::::::
# As dose contains more thant two levels ==>
# pairwise test is automatically performed.
compare_means(len ~ dose, df)

# Comparison against reference group
# ::::::::::::::::::::::::::::::::::::::::::
compare_means(len ~ dose, df, ref.group = "0.5")

# Comparison against all
# ::::::::::::::::::::::::::::::::::::::::::
compare_means(len ~ dose, df, ref.group = ".all.")

# Anova and kruskal.test
# ::::::::::::::::::::::::::::::::::::::::::
compare_means(len ~ dose, df, method = "anova")
compare_means(len ~ dose, df, method = "kruskal.test")
```

`create_aes` *Create Aes Mapping from a List*

Description

Create aes mapping to make programming easy with ggplot2.

Usage

```
create_aes(.list, parse = TRUE)
```

Arguments

.list	a list of aesthetic arguments; for example .list = list(x = "dose", y = "len", color = "dose").
parse	logical. If TRUE, parse the input as an expression.

Examples

```
# Simple aes creation
create_aes(list(x = "Sepal.Length", y = "Petal.Length"))

# Parse an expression
x <- "log2(Sepal.Length)"
y <- "log2(Petal.Length)"
create_aes(list(x = x, y = y), parse = TRUE)

# Create a ggplot
mapping <- create_aes(list(x = x, y = y), parse = TRUE)
ggplot(iris, mapping) +
  geom_point()
```

---

create_p_label *Create P-Value Label with Proper Prefix*

Description

Creates a p-value label string with proper handling of inequality symbols. When the formatted p-value starts with "<" or ">", uses "p <value" or "p >value" format. Otherwise uses "p = value" format.

Usage

```
create_p_label(p.format, p.signif = NULL)
```

Arguments

p.format	Character string of the formatted p-value (e.g., "0.05", "< 0.001").
p.signif	Optional character string of significance symbol (e.g., "*", "**", "ns"). If provided, it will be appended after the p-value.

Value

A character string with the properly formatted p-value label. Missing values in `p.format` are preserved as `NA_character_`.

Examples

```
## Not run:
create_p_label("0.05")
# Returns: "p = 0.05"

create_p_label("< 0.001")
# Returns: "p < 0.001"

create_p_label("0.01", "**")
# Returns: "p = 0.01 **"

create_p_label("< 0.001", "****")
# Returns: "p < 0.001 ****"

create_p_label(c("0.05", NA_character_))
# Returns: c("p = 0.05", NA)

## End(Not run)
```

---

desc_statby *Descriptive statistics by groups*

Description

Computes descriptive statistics by groups for a measure variable.

Usage

```
desc_statby(data, measure.var, grps, ci = 0.95)
```

Arguments

`data`	a data frame.
`measure.var`	the name of a column containing the variable to be summarized.
`grps`	a character vector containing grouping variables; e.g.: grps = c("grp1", "grp2")
`ci`	the percent range of the confidence interval (default is 0.95).

Value

A data frame containing descriptive statistics, such as:

- **length**: the number of elements in each group
- **min**: minimum
- **max**: maximum
- **median**: median
- **mean**: mean
- **iqr**: interquartile range
- **mad**: median absolute deviation (see ?MAD)
- **sd**: standard deviation of the sample
- **se**: standard error of the mean. It's calculated as the sample standard deviation divided by the root of the sample size.
- **ci**: confidence interval of the mean
- **range**: the range = max - min
- **cv**: coefficient of variation, sd/mean
- **var**: variance, sd^2

Examples

```
# Load data
data("ToothGrowth")

# Descriptive statistics
res <- desc_statby(ToothGrowth,
  measure.var = "len",
  grps = c("dose", "supp")
)
head(res[, 1:10])
```

---

diff_express *Differential gene expression analysis results*

Description

Differential gene expression analysis results obtained from comparing the RNAseq data of two different cell populations using DESeq2

Usage

```
data("diff_express")
```

Format

A data frame with 36028 rows and 5 columns.

`name` gene names

`baseMean` mean expression signal across all samples

`log2FoldChange` log2 fold change

`padj` Adjusted p-value

`detection_call` a numeric vector specifying whether the genes is expressed (value = 1) or not (value = 0).

Examples

```
data(diff_express)

# Default plot
ggmaplot(diff_express,
  main = expression("Group 1" %->% "Group 2"),
  fdr = 0.05, fc = 2, size = 0.4,
  palette = c("#B31B21", "#1465AC", "darkgray"),
  genenames = as.vector(diff_express$name),
  legend = "top", top = 20,
  font.label = c("bold", 11),
  font.legend = "bold",
  font.main = "bold",
  ggtheme = ggplot2::theme_minimal()
)

# Add rectangle around labesl
ggmaplot(diff_express,
  main = expression("Group 1" %->% "Group 2"),
  fdr = 0.05, fc = 2, size = 0.4,
  palette = c("#B31B21", "#1465AC", "darkgray"),
  genenames = as.vector(diff_express$name),
  legend = "top", top = 20,
  font.label = c("bold", 11), label.rectangle = TRUE,
  font.legend = "bold",
  font.main = "bold",
  ggtheme = ggplot2::theme_minimal()
)
```

---

facet *Facet a ggplot into Multiple Panels*

Description

Create multi-panel plots of a data set grouped by one or two grouping variables. Wrapper around `facet_wrap`

Usage

```
facet(
  p,
  facet.by,
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  short.panel.labs = TRUE,
  labeller = "label_value",
  panel.labs = NULL,
  panel.labs.background = list(color = NULL, fill = NULL),
  panel.labs.font = list(face = NULL, color = NULL, size = NULL, angle = NULL),
  panel.labs.font.x = panel.labs.font,
  panel.labs.font.y = panel.labs.font,
  strip.position = "top",
  ...
)
```

Arguments

p	a ggplot
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
nrow, ncol	Number of rows and columns in the panel. Used only when the data is faceted by one grouping variable.
scales	should axis scales of panels be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y").
short.panel.labs	logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.
labeller	Character vector. An alternative to the argument short.panel.labs. Possible values are one of "label_both" (panel labelled by both grouping variable names and levels) and "label_value" (panel labelled with only grouping levels).
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).
panel.labs.background	a list to customize the background of panel labels. Should contain the combination of the following elements: • color, linetype, linewidth: background line color, type and width • fill: background fill color. For example, panel.labs.background = list(color = "blue", fill = "pink", linetype = "dashed", linewidth = 0.5). Note: size is deprecated, use linewidth instead.

`panel.labs.font`
: a list of aesthetics indicating the size (e.g.: 14), the face/style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") and the orientation angle (e.g.: 45) of panel labels.

`panel.labs.font.x`, `panel.labs.font.y`
: same as panel.labs.font but for only x and y direction, respectively.

`strip.position`
: (used only in `facet_wrap()`). By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top", "bottom", "left", "right")`

`...`
: not used

Examples

```
p <- ggboxplot(ToothGrowth,
  x = "dose", y = "len",
  color = "supp"
)
print(p)

facet(p, facet.by = "supp")

# Customize
facet(p + theme_bw(),
  facet.by = "supp",
  short.panel.labs = FALSE, # Allow long labels in panels
  panel.labs.background = list(fill = "steelblue", color = "steelblue")
)
```

---

font *Change the Appearance of Titles and Axis Labels*

Description

Change the appearance of the main title, subtitle, caption, axis labels and text, as well as the legend title and texts. Wrapper around `element_text()`.

Usage

```
font(object, size = NULL, color = NULL, face = NULL, family = NULL, ...)
```

Arguments

`object`
: character string specifying the plot components. Allowed values include:

- `"title"` for the main title
- `"subtitle"` for the plot subtitle
- `"caption"` for the plot caption

- "legend.title" for the legend title
- "legend.text" for the legend text
- "x", "xlab", or "x.title" for x axis label
- "y", "ylab", or "y.title" for y axis label
- "xy", "xylab", "xy.title" or "axis.title" for both x and y axis labels
- "x.text" for x axis texts (x axis tick labels)
- "y.text" for y axis texts (y axis tick labels)
- "xy.text" or "axis.text" for both x and y axis texts

size	numeric value specifying the font size, (e.g.: size = 12).
color	character string specifying the font color, (e.g.: color = "red").
face	the font face or style. Allowed values include one of "plain", "bold", "italic", "bold.italic", (e.g.: face = "bold.italic").
family	the font family.
...	other arguments to pass to the function element_text().

Examples

```
# Load data
data("ToothGrowth")

# Basic plot
p <- ggboxplot(ToothGrowth,
  x = "dose", y = "len", color = "dose",
  title = "Box Plot created with ggpubr",
  subtitle = "Length by dose",
  caption = "Source: ggpubr",
  xlab = "Dose (mg)", ylab = "Teeth length"
)
p

# Change the appearance of titles and labels
p +
  font("title", size = 14, color = "red", face = "bold.italic") +
  font("subtitle", size = 10, color = "orange") +
  font("caption", size = 10, color = "orange") +
  font("xlab", size = 12, color = "blue") +
  font("ylab", size = 12, color = "#993333") +
  font("xy.text", size = 12, color = "gray", face = "bold")

# Change the appearance of legend title and texts
p +
  font("legend.title", color = "blue", face = "bold") +
  font("legend.text", color = "red")
```

`format_p_value` *Format P-Values According to Style or Custom Settings*

Description

Formats p-values according to major scientific publication standards (APA, AMA, NEJM, Lancet, etc.) or custom user-defined settings. This function provides flexible control over decimal places, leading zeros, and threshold notation for very small p-values.

Usage

```
format_p_value(
  p,
  style = "default",
  digits = NULL,
  leading.zero = NULL,
  min.threshold = NULL,
  decimal.mark = NULL,
  use.scientific = NULL
)
```

Arguments

`p`	Numeric vector of p-values to format.
`style`	Character string specifying the formatting style. One of: • `"default"`: Current behavior with scientific notation (backward compatible) • `"apa"`: APA Style - no leading zero, 3 decimals, "< .001" threshold • `"nejm"`: NEJM Style - leading zero, 3 decimals, "< 0.001" threshold • `"lancet"`: Lancet Style - leading zero, 4 decimals, "< 0.0001" threshold • `"ama"`: AMA Style - no leading zero, 3 decimals, "< .001" threshold • `"graphpad"`: GraphPad Style - leading zero, 4 decimals, "< 0.0001" threshold • `"scientific"`: Scientific notation for genomics/GWAS studies
`digits`	Integer specifying the number of decimal places. If provided, overrides the style default.
`leading.zero`	Logical indicating whether to include leading zero before decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.
`min.threshold`	Numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold" (e.g., "< 0.001"). If NULL, the selected style's default threshold is used; styles without a threshold show exact values. If provided, overrides the style default. Must be a single positive finite number.
`decimal.mark`	Character string to use as the decimal mark. If NULL, uses `getOption("OutDec")`.
`use.scientific`	Logical indicating whether to force scientific notation. If NULL, uses the style default.

Details

P-value formatting conventions vary across scientific disciplines and journals:

APA Style (Psychology, Social Sciences):

- No leading zero (write ".05" not "0.05")
- 3 decimal places
- Report as "p < .001" for very small values

NEJM/Medical Journals:

- Leading zero (write "0.05" not ".05")
- 3 decimal places
- P < 0.001: report as "< 0.001"

Scientific Notation (GWAS, Genomics):

- Used when very small p-values are meaningful (e.g., 5e-8 threshold)
- Appropriate for high-dimensional data analyses

Value

Character vector of formatted p-values.

See Also

get_p_format_style, list_p_format_styles

Examples

```
# Test p-values
p_vals <- c(0.76404, 0.0432, 0.0043, 0.00018, 1.7e-11)

# Different styles
format_p_value(p_vals, style = "default")
format_p_value(p_vals, style = "apa")
format_p_value(p_vals, style = "nejm")
format_p_value(p_vals, style = "lancet")

# Custom formatting
format_p_value(p_vals, digits = 2, leading.zero = FALSE, min.threshold = 0.01)

# Override style defaults
format_p_value(p_vals, style = "nejm", digits = 4)
```

gene_citation *Gene Citation Index*

Description

Contains the mean citation index of 66 genes obtained by assessing PubMed abstracts and annotations using two key words i) Gene name + b cell differentiation and ii) Gene name + plasma cell differentiation.

Usage

```
data("gene_citation")
```

Format

A data frame with 66 rows and 2 columns.

`gene` gene names

`citation_index` mean citation index

Examples

```
data(gene_citation)

# Some key genes of interest to be highlighted
key.gns <- c(
  "MYC", "PRDM1", "CD69", "IRF4", "CASP3", "BCL2L1", "MYB", "BACH2", "BIM1", "PTEN",
  "KRAS", "FOXP1", "IGF1R", "KLF4", "CDK6", "CCND2", "IGF1", "TNFAIP3", "SMAD3", "SMAD7",
  "BMPR2", "RB1", "IGF2R", "ARNT"
)
# Density distribution
ggdensity(gene_citation,
  x = "citation_index", y = "..count..",
  xlab = "Number of citation",
  ylab = "Number of genes",
  fill = "lightgray", color = "black",
  label = "gene", label.select = key.gns, repel = TRUE,
  font.label = list(color = "citation_index"),
  xticks.by = 20, # Break x ticks by 20
  gradient.cols = c("blue", "red"),
  legend = "bottom",
  legend.title = "" # Hide legend title
)
```

gene_expression *Gene Expression Data*

Description

Gene expression data extracted from TCGA using the 'RTCGA' and 'RTCGA.mRNA' R packages. It contains the mRNA expression for 3 genes - GATA3, PTEN and XBP1- from 3 different datasets: Breast invasive carcinoma (BRCA), Ovarian serous cystadenocarcinoma (OV) and Lung squamous cell carcinoma (LUSC)

Usage

```
data("gene_expression")
```

Format

A data frame with 1305 rows and 5 columns.

`bcr_patient_barcode` sample ID

`dataset` cancer type

`GATA3` GATA3 gene expression

`PTEN` PTEN gene expression

`XBP1` XBP1 gene expression.

Examples

```
data(gene_expression)

ggboxplot(gene_expression,
  x = "dataset",
  y = c("GATA3", "PTEN", "XBP1"),
  combine = TRUE,
  ylab = "Expression",
  color = "dataset", palette = "jco"
)
```

geom_exec *Execute ggplot2 functions*

Description

A helper function used by ggpubr functions to execute any geom_* functions in ggplot2. Useful only when you want to call a geom_* function without worrying about the arguments to put in aes(). Basic users of ggpubr don't need this function.

Usage

```
geom_exec(geomfunc = NULL, data = NULL, position = NULL, ...)
```

Arguments

geomfunc	a ggplot2 function (e.g.: geom_point)
data	a data frame to be used for mapping
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
...	arguments accepted by the function

Value

a plot if geomfunc != NULL or a list(option, mapping) if geomfunc is NULL.

Examples

```
## Not run:
ggplot() +
  geom_exec(geom_point,
    data = mtcars,
    x = "mpg", y = "wt", size = "cyl", color = "cyl"
  )

## End(Not run)
```

---

get_breaks *Easy Break Creation for Numeric Axes*

Description

Creates breaks for numeric axes to be used in the functions scale_x_continuous() and scale_y_continuous(). Can be used to increase the number of x and y ticks by specifying the option n. It's also possible to control axis breaks by specifying a step between ticks. For example, if by = 5, a tick mark is shown on every 5.

Usage

```
get_breaks(n = NULL, by = NULL, from = NULL, to = NULL)
```

Arguments

n	number of breaks.
by	number: the step between breaks.
from	the starting value of breaks. By default, 0 is used for positive variables
to	the end values of breaks. This corresponds generally to the maximum limit of the axis.

Value

a break function

Examples

```
# Generate 5 breaks for a variable x
get_breaks(n = 5)(x = 1:100)

# Generate breaks using an increasing step
get_breaks(by = 10)(x = 1:100)

# Combine with ggplot scale_xx functions
library(ggplot2)

# Create a basic plot
p <- ggscatter(mtcars, x = "wt", y = "mpg")
p

# Increase the number of ticks
p +
  scale_x_continuous(breaks = get_breaks(n = 10)) +
  scale_y_continuous(breaks = get_breaks(n = 10))

# Set ticks according to a specific step, starting from 0
p + scale_x_continuous(
  breaks = get_breaks(by = 1.5, from = 0),
  limits = c(0, 6)
) +
  scale_y_continuous(
    breaks = get_breaks(by = 10, from = 0),
    limits = c(0, 40)
  )
```

---

get_coord *Checks and Returns Data Coordinates from Multiple Input Options*

Description

Checks and returns selected coordinates from multiple input options, which can be either data (x-y) coordinates or npc (normalized parent coordinates).

Helper function internally used in ggpubr function to guess the type of coordinates specified by the user. For example, in the function `stat_cor()`, users can specify either the option `label.x` (data coordinates) or `label.x.npc` (npc coordinates); those coordinates are passed to `get_coord()`, which will make some checking and then return a unique coordinates for the label position.

Usage

```
get_coord(
  group = 1L,
  data.ranges = NULL,
  coord = NULL,
  npc = "left",
  step = 0.1,
  margin.npc = 0.05
)
```

Arguments

group	integer ggplot's group id. Used to shift coordinates to avoid overlaps.
data.ranges	a numeric vector of length 2 containing the data ranges (minimum and the maximum). Should be specified only when coord = NULL and npc is specified. Used to convert npc to data coordinates. Considered only when the argument npc is specified.
coord	data coordinates (i.e., either x or y coordinates).
npc	numeric (in [0-1]) or character vector of coordinates. If character, should be one of c('right', 'left', 'bottom', 'top', 'center', 'centre', 'middle'). Note that, the data.ranges, step and margin.npc, arguments are considered only when npc is specified. The option npc is ignored when the argument coord is specified.
step	numeric value in [0-1]. The step size for shifting coordinates in npc units. Considered as horizontal step for x-axis and vertical step for y-axis. For y-axis, the step value can be negative to reverse the order of groups.
margin.npc	numeric [0-1] The margin added towards the nearest plotting area edge when converting character coordinates into npc.

Value

a numeric vector representing data coordinates.

See Also

as_npc, npc_to_data_coord.

Examples

```
# If npc is specified, it is converted into data coordinates
get_coord(data.ranges = c(2, 20), npc = "left")
get_coord(data.ranges = c(2, 20), npc = 0.1)

# When coord is specified, no transformation is performed
# because this is assumed to be a data coordinate
get_coord(coord = 5)

# For grouped plots
res_top <- get_coord(
```

```
  data.ranges = c(4.2, 36.4), group = c(1, 2, 3),
  npc = "top", step = -0.1, margin.npc = 0
)
res_top
```

---

get_legend *Extract Legends from a ggplot object*

Description

Extract the legend labels from a ggplot object.

Usage

```
get_legend(p, position = NULL)
```

Arguments

p	an object of class ggplot or a list of ggplots. If p is a list, only the first legend is returned.
position	character specifying legend position. Allowed values are one of c("top", "bottom", "left", "right", "none"). To remove the legend use legend = "none".

Value

an object of class gtable.

Examples

```
# Create a scatter plot
p <- ggscatter(iris,
  x = "Sepal.Length", y = "Sepal.Width",
  color = "Species", palette = "jco",
  ggtheme = theme_minimal()
)
p

# Extract the legend. Returns a gtable
leg <- get_legend(p)

# Convert to a ggplot and print
as_ggplot(leg)
```

`get_palette` *Generate Color Palettes*

Description

Generate a palette of k colors from ggsci palettes, RColorBrewer palettes and custom color palettes. Useful to extend RColorBrewer and ggsci to support more colors.

Usage

```
get_palette(palette = "default", k)
```

Arguments

`palette`	Color palette. Allowed values include: • **Grey color palettes**: "grey" or "gray"; • **RColorBrewer palettes**, see `brewer.pal` and details section. Examples of palette names include: "RdBu", "Blues", "Dark2", "Set2", ...; • **Custom color palettes**. For example, palette = c("#00AFBB", "#E7B800", "#FC4E07"); • **ggsci scientific journal palettes**, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
`k`	the number of colors to generate.

Details

RColorBrewer palettes: To display all available color palettes, type this in R:RColorBrewer::display.brewer.all(). Color palette names include:

- **Sequential palettes**, suited to ordered data that progress from low to high. Palette names include: Blues BuGn BuPu GnBu Greens Greys Oranges OrRd PuBu PuBuGn PuRd Purples RdPu Reds YlGn YlGnBu YlOrBr YlOrRd.
- **Diverging palettes**:Gradient colors. Names include: BrBG PiYG PRGn PuOr RdBu RdGy RdYlBu RdYlGn Spectral.
- **Qualitative palettes**: Best suited to representing nominal or categorical data. Names include: Accent, Dark2, Paired, Pastel1, Pastel2, Set1, Set2, Set3.

Value

Returns a vector of color palettes.

Examples

```
data("iris")
iris$Species2 <- factor(rep(c(1:10), each = 15))

# Generate a gradient of 10 colors
ggscatter(iris,
  x = "Sepal.Length", y = "Petal.Length",
  color = "Species2",
  palette = get_palette(c("#00AFBB", "#E7B800", "#FC4E07"), 10)
)

# Scatter plot with default color palette
ggscatter(iris,
  x = "Sepal.Length", y = "Petal.Length",
  color = "Species"
)

# RColorBrewer color palettes
ggscatter(iris,
  x = "Sepal.Length", y = "Petal.Length",
  color = "Species", palette = get_palette("Dark2", 3)
)

# ggsci color palettes
ggscatter(iris,
  x = "Sepal.Length", y = "Petal.Length",
  color = "Species", palette = get_palette("npg", 3)
)

# Custom color palette
ggscatter(iris,
  x = "Sepal.Length", y = "Petal.Length",
  color = "Species",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)

# Or use this
ggscatter(iris,
  x = "Sepal.Length", y = "Petal.Length",
  color = "Species",
  palette = get_palette(c("#00AFBB", "#FC4E07"), 3)
)
```

---

get_p_format_style *Get P-Value Format Style Settings*

Description

Returns the settings for a given p-value format style.

Usage

```
get_p_format_style(style = "default")
```

Arguments

style	Character string specifying the style. One of: "default", "apa", "nejm", "lancet", "ama", "graphpad", "scientific".

Value

A list containing: digits, leading.zero, min.threshold, use.scientific, description.

Examples

```
get_p_format_style("apa")
get_p_format_style("nejm")
```

---

ggadd *Add Summary Statistics or a Geom onto a ggplot*

Description

Add summary statistics or a geometry onto a ggplot.

Usage

```
ggadd(
  p,
  add = NULL,
  color = "black",
  fill = "white",
  group = 1,
  width = 1,
  shape = 19,
  size = NULL,
  alpha = 1,
  jitter = 0.2,
  seed = 123,
  binwidth = NULL,
  dotsize = size,
  linetype = 1,
  linewidth = size,
  show.legend = NA,
  error.plot = "pointrange",
  ci = 0.95,
  outliers = TRUE,
```

```
  outlier.shape = 19,
  data = NULL,
  position = position_dodge(0.8),
  p_geom = ""
)
```

Arguments

p	a ggplot
add	character vector specifying other plot elements to be added. Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range".
color	point or outline color.
fill	fill color. Used only when `error.plot = "crossbar"`.
group	grouping variable. Allowed values are 1 (for one group) or a character vector specifying the name of the grouping variable. Used only for adding statistical summary per group.
width	numeric value between 0 and 1 specifying bar or box width. Example width = 0.8. Used only when `error.plot` is one of c("crossbar", "errorbar").
shape	point shape. Allowed values can be displayed using the function `show_point_shapes()`.
size	numeric value in [0-1] specifying point and line size.
alpha	numeric value specifying fill color transparency. Value should be in [0, 1], where 0 is full transparency and 1 is no transparency.
jitter	a numeric value specifying the amount of jittering. Used only when add contains "jitter".
seed	A random seed to make the jitter reproducible. Default is '123'. Useful if you need to apply the same jitter twice, e.g., for a point and a corresponding label. The random seed is reset after jittering. If 'NA', the seed is initialized with a random value; this makes sure that two subsequent calls start with a different seed. Use NULL to use the current random seed and also avoid resetting (the behaviour of ggplot 2.2.1 and earlier).
binwidth	numeric value specifying bin width. use value between 0 and 1 when you have a strong dense dotplot. For example binwidth = 0.2. Used only when add contains "dotplot".
dotsize	as `size` but applied only to dotplot.
linetype	line type.
linewidth	numeric value in [0-1] specifying line width. Used only when add contains "line" or for adding error bars. Default is `size`.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. `FALSE` never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

error.plot	plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar","lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange","lower_linerange"). Default value is "pointrange".
ci	the percent range of the confidence interval (default is 0.95).
outliers	logical. If TRUE (default), outliers are displayed in boxplots. If FALSE, outliers are not displayed.
outlier.shape	numeric value specifying the shape of outliers in boxplots. Default is 19 (filled circle). if set to NA, outliers are not displayed.
data	a data.frame to be displayed. If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot.
position	position adjustment, either as a string, or the result of a call to a position adjustment function. Used to adjust position for multiple groups.
p_geom	the geometry of the main plot. Ex: p_geom = "geom_line". If NULL, the geometry is extracted from p. Used only by ggline().

Examples

```
# Basic violin plot
data("ToothGrowth")
p <- ggviolin(ToothGrowth, x = "dose", y = "len", add = "none")

# Add mean +/- SD and jitter points
p %>% ggadd(c("mean_sd", "jitter"), color = "dose")

# Add box plot
p %>% ggadd(c("boxplot", "jitter"), color = "dose")
```

---

ggadjust_pvalue *Adjust p-values Displayed on a GGPlot*

Description

Adjust p-values produced by geom_pwc() on a ggplot. This is mainly useful when using facet, where p-values are generally computed and adjusted by panel without taking into account the other panels. In this case, one might want to adjust after the p-values of all panels together.

Usage

```
ggadjust_pvalue(
  p,
  layer = NULL,
  p.adjust.method = "holm",
  label = "p.adj",
  hide.ns = NULL,
```

```
  symnum.args = list(),
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  signif.cutoffs = NULL,
  signif.symbols = NULL,
  ns.symbol = "ns",
  use.four.stars = FALSE,
  output = c("plot", "stat_test")
)
```

Arguments

p	a ggplot
layer	An integer indicating the statistical layer rank in the ggplot (in the order added to the plot).
p.adjust.method	method for adjusting p values (see p.adjust). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Ignored when the specified method is "tukey_hsd" or "games_howell_test" because they come with internal p adjustment method. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".
label	character string specifying label. Can be: • the column containing the label (e.g.: label = "p" or label = "p.adj"), where p is the p-value. Other possible values are "p.signif", "p.adj.signif", "p.format", "p.format.signif", "p.adj.format". • an expression that can be formatted by the glue() package. For example, when specifying label = "Wilcoxon, p = \{p\}", the expression {p} will be replaced by its value. • a combination of plotmath expressions and glue expressions. You may want some of the statistical parameter in italic; for example:label = "Wilcoxon,italic(p)= {p}" .
hide.ns	can be logical value (TRUE or FALSE) or a character vector ("p.adj" or "p").
symnum.args	a list of arguments to pass to the function symnum for symbolic number coding of p-values. For example, symnum.args = list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns")). In other words, we use the following convention for symbols indicating statistical significance: • ns: p > 0.05 • *: p <= 0.05 • **: p <= 0.01

- ***: p <= 0.001
- ****: p <= 0.0001

Note: If symnum.args is provided, it takes precedence over signif.cutoffs.

p.format.style character string specifying the p-value formatting style. One of: "default" (backward compatible, uses scientific notation), "apa" (APA style, no leading zero), "nejm" (NEJM style), "lancet" (Lancet style), "ama" (AMA style), "graphpad" (GraphPad style), or "scientific" (scientific notation for GWAS). See list_p_format_styles for details.

p.digits integer specifying the number of decimal places for p-values. If provided, overrides the style default.

p.leading.zero logical indicating whether to include leading zero before decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

p.min.threshold numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold". If provided, overrides the style default.

p.decimal.mark character string to use as the decimal mark. If NULL, uses getOption("OutDec").

signif.cutoffs numeric vector of p-value cutoffs in descending order for assigning significance symbols. For example, c(0.10, 0.05, 0.01) means p < 0.10 gets "*", p < 0.05 gets "**", p < 0.01 gets "***". Default is NULL, which uses the package defaults.

signif.symbols character vector of symbols corresponding to signif.cutoffs. If NULL, auto-generated as "*", "**", "***" (and "****" if use.four.stars = TRUE).

ns.symbol character string for non-significant results. Default is "ns". Use "" (empty string) to show nothing.

use.four.stars logical. If TRUE, allows four stars (****) for the most significant level. Default is FALSE.

output character. Possible values are one of c("plot","stat_test"). Default is "plot".

Examples

```
# Data preparation
# ::::::::::::::::::::::::::::::::::::::::
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Add a random grouping variable
df$group <- factor(rep(c("grp1", "grp2"), 30))
head(df, 3)

# Boxplot: Two groups by panel
# ::::::::::::::::::::::::::::::::::::::::
# Create a box plot
bxp <- ggboxplot(
  df,
  x = "supp", y = "len", fill = "#00AFBB",
  facet.by = "dose"
)
```

```
# Make facet and add p-values
bxp <- bxp + geom_pwc(method = "t_test")
bxp
# Adjust all p-values together after
ggadjust_pvalue(
  bxp,
  p.adjust.method = "bonferroni",
  label = "{p.adj.format}{p.adj.signif}", hide.ns = TRUE
)


# Boxplot: Three groups by panel
# ::::::::::::::::::::::::::::::::::::::
# Create a box plot
bxp <- ggboxplot(
  df,
  x = "dose", y = "len", fill = "#00AFBB",
  facet.by = "supp"
)
# Make facet and add p-values
bxp <- bxp + geom_pwc(method = "t_test")
bxp
# Adjust all p-values together after
ggadjust_pvalue(
  bxp,
  p.adjust.method = "bonferroni",
  label = "{p.adj.format}{p.adj.signif}"
)
```

---

ggarrange *Arrange Multiple ggplots*

Description

Arrange multiple ggplots on the same page. Wrapper around plot_grid(). Can arrange multiple ggplots over multiple pages, compared to the standard plot_grid(). Can also create a common unique legend for multiple plots.

Usage

```
ggarrange(
  ...,
  plotlist = NULL,
  ncol = NULL,
  nrow = NULL,
  labels = NULL,
  label.x = 0,
  label.y = 1,
  hjust = -0.5,
```

```
  vjust = 1.5,
  font.label = list(size = 14, color = "black", face = "bold", family = NULL),
  align = c("none", "h", "v", "hv"),
  widths = 1,
  heights = 1,
  byrow = TRUE,
  legend = NULL,
  common.legend = FALSE,
  legend.grob = NULL,
  spacing = 0
)
```

Arguments

...	list of plots to be arranged into the grid. The plots can be either ggplot2 plot objects or arbitrary gtables.
plotlist	(optional) list of plots to display.
ncol	(optional) number of columns in the plot grid.
nrow	(optional) number of rows in the plot grid.
labels	(optional) list of labels to be added to the plots. You can also set labels="AUTO" to auto-generate upper-case labels or labels="auto" to auto-generate lower-case labels.
label.x	(optional) Single value or vector of x positions for plot labels, relative to each subplot. Defaults to 0 for all labels. (Each label is placed all the way to the left of each plot.)
label.y	(optional) Single value or vector of y positions for plot labels, relative to each subplot. Defaults to 1 for all labels. (Each label is placed all the way to the top of each plot.)
hjust	Adjusts the horizontal position of each label. More negative values move the label further to the right on the plot canvas. Can be a single value (applied to all labels) or a vector of values (one for each label). Default is -0.5.
vjust	Adjusts the vertical position of each label. More positive values move the label further down on the plot canvas. Can be a single value (applied to all labels) or a vector of values (one for each label). Default is 1.5.
font.label	a list of arguments for customizing labels. Allowed values are the combination of the following elements: size (e.g.: 14), face (e.g.: "plain", "bold", "italic", "bold.italic"), color (e.g.: "red") and family. For example font.label = list(size = 14, face = "bold", color ="red").
align	(optional) Specifies whether graphs in the grid should be horizontally ("h") or vertically ("v") aligned. Options are "none" (default), "hv" (align in both directions), "h", and "v".
widths	(optional) numerical vector of relative columns widths. For example, in a two-column grid, widths = c(2, 1) would make the first column twice as wide as the second column.
heights	same as widths but for column heights.

byrow — logical. If TRUE (default), the plots are filled into the grid by row; set to FALSE to fill by column. Passed to plot_grid().

legend — character specifying legend position. Allowed values are one of c("top", "bottom", "left", "right", "none"). To remove the legend use legend = "none".

common.legend — logical value, or one or several plot indices. Default is FALSE. If TRUE, a single shared legend is used for all the arranged plots. Note that this legend is **not** merged or validated across plots: it is simply the legend of the *first* plot, and the other legends are dropped. It is therefore only correct when every plot shares the same scale (same groups/levels, order and color range). If the first plot's legend is not representative - for example a group is missing in the first plot, or a continuous color scale spans a different range - the shared legend will misrepresent the other plots. In that case you can: (i) give the plots a consistent scale yourself (e.g. scale_fill_manual(limits = ...) or scale_color_continuous(limits = ...)) so a single legend is valid, and/or (ii) choose which plot's legend is shown by passing that plot's index, e.g. common.legend = 2 to use the second plot's legend (equivalent to legend.grob = get_legend(plots[[2]])). You can also pass several indices, e.g. common.legend = c(1, 2), to keep and combine the legends of those plots into a single shared block (side by side for legend = "top"/ "bottom", stacked for "left"/"right") - useful when the plots genuinely need different legends. Note that (ii) only changes *which* legend is displayed; it does not re-map the other plots' color scales, so for the legend keys to match every panel you still need a consistent scale as in (i). When the plots genuinely cannot be described by a single legend (e.g. a discrete fill in one plot and a continuous color bar in another), use common.legend = FALSE to keep a separate legend per plot.

legend.grob — a legend grob as returned by the function get_legend(). If provided, it will be used as the common legend.

spacing — numeric value giving the margin, in text-line units, set uniformly around each plot to increase the gap between the arranged plots. Default is 0, which leaves each plot's own margins untouched (existing arrangements are unchanged). A positive value sets a uniform margin of that many lines around every plot, replacing the plots' default margin; e.g. spacing = 1 puts a one-line margin around each plot.

Value

an object of class ggarrange, which is a ggplot or a list of ggplots.

Author(s)

Laszlo Erdey <erdey.laszlo@econ.unideb.hu>

See Also

annotate_figure()

Examples

```
data("ToothGrowth")
df <- ToothGrowth
df$dose <- as.factor(df$dose)

# Create some plots
# :::::::::::::::::::::::::::::::::::::::::::::::::
# Box plot
bxp <- ggboxplot(df,
  x = "dose", y = "len",
  color = "dose", palette = "jco"
)
# Dot plot
dp <- ggdotplot(df,
  x = "dose", y = "len",
  color = "dose", palette = "jco"
)
# Density plot
dens <- ggdensity(df, x = "len", fill = "dose", palette = "jco")

# Arrange
# :::::::::::::::::::::::::::::::::::::::::::::::::
ggarrange(bxp, dp, dens, ncol = 2, nrow = 2)
# Use a common legend for multiple plots
ggarrange(bxp, dp, common.legend = TRUE)
```

---

ggballoonplot *Balloon plot*

Description

Plot a graphical matrix where each cell contains a dot whose size reflects the relative magnitude of the corresponding component. Useful to visualize contingency table formed by two categorical variables.

Usage

```
ggballoonplot(
  data,
  x = NULL,
  y = NULL,
  size = "value",
  facet.by = NULL,
  size.range = c(1, 10),
  shape = 21,
  color = "black",
  fill = "gray",
```

```
  show.label = FALSE,
  font.label = list(size = 12, color = "black"),
  rotate.x.text = TRUE,
  ggtheme = theme_minimal(),
  ...
)
```

Arguments

data	a data frame. Can be: • **a standard contingency table** formed by two categorical variables: a data frame with row names and column names. The categories of the first variable are columns and the categories of the second variable are rows. • **a stretched contingency table**: a data frame containing at least three columns corresponding, respectively, to (1) the categories of the first variable, (2) the categories of the second variable, (3) the frequency value. In this case, you should specify the argument x and y in the function `ggballoonplot()` .
x, y	the column names specifying, respectively, the first and the second variable forming the contingency table. Required only when the data is a stretched contingency table.
size	point size. By default, the points size reflects the relative magnitude of the value of the corresponding cell (`size = "value"`). Can be also numeric (`size = 4`).
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
size.range	a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol. Default values are `size.range = c(1, 10)`.
shape	points shape. The default value is 21. Alternative values include 22, 23, 24, 25.
color	point border line color.
fill	point fill color. Default is "lightgray". Considered only for points 21 to 25.
show.label	logical. If TRUE, show the data cell values as point labels.
font.label	a vector of length 3 indicating respectively the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of point labels. For example font.label = c(14, "bold", "red"). To specify only the size and the style, use font.label = c(14, "plain").
rotate.x.text	logical. If TRUE (default), rotate the x axis text.
ggtheme	function, ggplot2 theme name. Default value is `theme_minimal()`. Set `ggtheme = NULL` to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via `theme_set()`.
...	other arguments passed to the function ggpar

Examples

```
# Define color palette
my_cols <- c(
  "#0D0887FF", "#6A00A8FF", "#B12A90FF",
  "#E16462FF", "#FCA636FF", "#F0F921FF"
)

# Standard contingency table
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Read a contingency table: housetasks
# Repartition of 13 housetasks in the couple
data <- read.delim(
  system.file("demo-data/housetasks.txt", package = "ggpubr"),
  row.names = 1
)
data

# Basic ballon plot
ggballoonplot(data)

# Change color and fill
ggballoonplot(data, color = "#0073C2FF", fill = "#0073C2FF")


# Change color according to the value of table cells
ggballoonplot(data, fill = "value") +
  scale_fill_gradientn(colors = my_cols)

# Change the plotting symbol shape
ggballoonplot(data, fill = "value", shape = 23) +
  gradient_fill(c("blue", "white", "red"))


# Set points size to 8, but change fill color by values
# Sow labels
ggballoonplot(data,
  fill = "value", color = "lightgray",
  size = 10, show.label = TRUE
) +
  gradient_fill(c("blue", "white", "red"))

# Stretched contingency table
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# Create an Example Data Frame Containing Car x Color data
carnames <- c("bmw", "renault", "mercedes", "seat")
carcolors <- c("red", "white", "silver", "green")
datavals <- round(rnorm(16, mean = 100, sd = 60), 1)
car_data <- data.frame(
  Car = rep(carnames, 4),
  Color = rep(carcolors, c(4, 4, 4, 4)),
  Value = datavals
```

```
)

car_data

ggballoonplot(car_data,
  x = "Car", y = "Color",
  size = "Value", fill = "Value"
) +
  scale_fill_gradientn(colors = my_cols) +
  guides(size = "none")


# Grouped frequency table
# ::::::::::::::::::::::::::::::::::::::::::::::::::::::::
data("Titanic")
dframe <- as.data.frame(Titanic)
head(dframe)
ggballoonplot(
  dframe,
  x = "Class", y = "Sex",
  size = "Freq", fill = "Freq",
  facet.by = c("Survived", "Age"),
  ggtheme = theme_bw()
) +
  scale_fill_gradientn(colors = my_cols)

# Hair and Eye Color of Statistics Students
data(HairEyeColor)
ggballoonplot(as.data.frame(HairEyeColor),
  x = "Hair", y = "Eye", size = "Freq",
  ggtheme = theme_gray()
) %>%
  facet("Sex")
```

---

ggbarplot *Bar plot*

Description

Create a bar plot.

Usage

```
ggbarplot(
  data,
  x,
  y,
  combine = FALSE,
  merge = FALSE,
```

```
  color = "black",
  fill = "white",
  palette = NULL,
  size = NULL,
  width = NULL,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  select = NULL,
  remove = NULL,
  order = NULL,
  add = "none",
  add.params = list(),
  error.plot = "errorbar",
  label = FALSE,
  lab.col = "black",
  lab.size = 4,
  lab.pos = c("out", "in"),
  lab.vjust = NULL,
  lab.hjust = NULL,
  lab.nb.digits = NULL,
  sort.val = c("none", "desc", "asc"),
  sort.by.groups = TRUE,
  top = Inf,
  position = position_stack(),
  numeric.x.axis = FALSE,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x, y	x and y variables for drawing.
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color, fill	outline and fill colors.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues",

...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".

`size` Numeric value (e.g.: size = 1). change the size of points and outlines.

`width` numeric value between 0 and 1 specifying the width of the plot elements.

`title` plot main title.

`xlab` character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

`ylab` character vector specifying y axis labels. Use ylab = FALSE to hide ylab.

`facet.by` character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.

`panel.labs` a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

`short.panel.labs` logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

`select` character vector specifying which items to display.

`remove` character vector specifying which items to remove from the plot.

`order` character vector specifying the order of items.

`add` character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.

`add.params` parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

`error.plot` plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...

`label` specify whether to add labels on the bar plot. Allowed values are:

- **logical value**: If TRUE, y values are added as labels on the bar plot
- **character vector**: Used as text labels; must be the same length as y.

`lab.col`, `lab.size` text color and size for labels.

`lab.pos` character specifying the position for labels. Allowed values are "out" (for outside) or "in" (for inside). Ignored when lab.vjust != NULL.

`lab.vjust` numeric, vertical justification of labels. Provide negative value (e.g.: -0.4) to put labels outside the bars or positive value to put labels inside (e.g.: 2).

`lab.hjust` numeric, horizontal justification of labels.

lab.nb.digits	integer indicating the number of decimal places (round) to be used.
sort.val	a string specifying whether the value should be sorted. Allowed values are "none" (no sorting), "asc" (for ascending) or "desc" (for descending).
sort.by.groups	logical value. If TRUE the data are sorted by groups. Used only when sort.val != "none".
top	a numeric value specifying the number of top elements to be shown.
position	position adjustment, either as a string, or the result of a call to a position adjustment function (e.g. position_dodge(0.8)). Used to control the spacing between grouped elements.
numeric.x.axis	logical. If TRUE, x axis will be treated as numeric. Default is FALSE. Useful, for example, to plot bars at their numeric x positions (e.g. a time axis) instead of at equally-spaced discrete categories. Ignored when order is set or sort.val != "none", which require a discrete x axis.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to be passed to ggpar().

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

Faceting a summarized bar plot

When the bars show a computed summary (e.g. add = "mean_se"), facet the plot with the facet.by argument - **not** by appending + facet_wrap() / + facet_grid(). The summaries are pre-computed, grouping by x, color/fill and facet.by; a facet added afterwards is not part of that grouping, so the bars (and, for stacked bars, the error bars) are pooled over the whole data set and repeated identically in every panel. Use ggbarplot(..., facet.by = "group") for correct per-panel summaries.

See Also

ggpar, ggline

Examples

```
# Data
df <- data.frame(
  dose = c("D0.5", "D1", "D2"),
  len = c(4.2, 10, 29.5)
)
print(df)

# Basic plot with label outsite
# ++++++++++++++++++++++++++++
ggbarplot(df,
  x = "dose", y = "len",
  label = TRUE, label.pos = "out"
)

# Change width
ggbarplot(df, x = "dose", y = "len", width = 0.5)

# Change the plot orientation: horizontal
ggbarplot(df, "dose", "len", orientation = "horiz")

# Change the default order of items
ggbarplot(df, "dose", "len",
  order = c("D2", "D1", "D0.5")
)


# Change colors
# ++++++++++++++++++++++++++++

# Change fill and outline color
# add labels inside bars
ggbarplot(df, "dose", "len",
  fill = "steelblue", color = "steelblue",
  label = TRUE, lab.pos = "in", lab.col = "white"
)

# Change colors by groups: dose
# Use custom color palette
ggbarplot(df, "dose", "len",
  color = "dose",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)

# Change fill and outline colors by groups
ggbarplot(df, "dose", "len",
  fill = "dose", color = "dose",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)


# Plot with multiple groups
```

```
# ++++++++++++++++++++

# Create some data
df2 <- data.frame(
  supp = rep(c("VC", "OJ"), each = 3),
  dose = rep(c("D0.5", "D1", "D2"), 2),
  len = c(6.8, 15, 33, 4.2, 10, 29.5)
)
print(df2)

# Plot "len" by "dose" and change color by a second group: "supp"
# Add labels inside bars
ggbarplot(df2, "dose", "len",
  fill = "supp", color = "supp", palette = "Paired",
  label = TRUE, lab.col = "white", lab.pos = "in"
)

# Change position: Interleaved (dodged) bar plot
ggbarplot(df2, "dose", "len",
  fill = "supp", color = "supp", palette = "Paired",
  label = TRUE,
  position = position_dodge(0.9)
)

# Add points and errors
# ++++++++++++++++++++++++++

# Data: ToothGrowth data set we'll be used.
df3 <- ToothGrowth
head(df3, 10)

# It can be seen that for each group we have
# different values
ggbarplot(df3, x = "dose", y = "len")

# Visualize the mean of each group
ggbarplot(df3,
  x = "dose", y = "len",
  add = "mean"
)

# Add error bars: mean_se
# (other values include: mean_sd, mean_ci, median_iqr, ....)
# Add labels
ggbarplot(df3,
  x = "dose", y = "len",
  add = "mean_se", label = TRUE, lab.vjust = -1.6
)

# Use only "upper_errorbar"
ggbarplot(df3,
  x = "dose", y = "len",
  add = "mean_se", error.plot = "upper_errorbar"
```

```
)

# Change error.plot to "pointrange"
ggbarplot(df3,
  x = "dose", y = "len",
  add = "mean_se", error.plot = "pointrange"
)

# Add jitter points and errors (mean_se)
ggbarplot(df3,
  x = "dose", y = "len",
  add = c("mean_se", "jitter")
)

# Add dot and errors (mean_se)
ggbarplot(df3,
  x = "dose", y = "len",
  add = c("mean_se", "dotplot")
)

# Multiple groups with error bars and jitter point
ggbarplot(df3,
  x = "dose", y = "len", color = "supp",
  add = "mean_se", palette = c("#00AFBB", "#E7B800"),
  position = position_dodge()
)
#
```

---

ggboxplot *Box plot*

Description

Create a box plot with points. Box plots display a group of numerical data through their quartiles.

Usage

```
ggboxplot(
  data,
  x,
  y,
  combine = FALSE,
  merge = FALSE,
  color = "black",
  fill = "white",
  palette = NULL,
  title = NULL,
  xlab = NULL,
```

```
  ylab = NULL,
  bxp.errorbar = FALSE,
  bxp.errorbar.width = 0.4,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  linetype = "solid",
  size = NULL,
  linewidth = NULL,
  width = 0.7,
  notch = FALSE,
  outliers = TRUE,
  outlier.shape = 19,
  select = NULL,
  remove = NULL,
  order = NULL,
  add = "none",
  add.params = list(),
  error.plot = "pointrange",
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  position = position_dodge(0.8),
  ggtheme = theme_pubr(),
  show.n = FALSE,
  ...
)
```

Arguments

data	a data frame
x	character string containing the name of x variable.
y	character vector containing one or more variables to plot
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	outline color.
fill	fill color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues",

	...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
bxp.errorbar	logical value. If TRUE, shows error bars of box plots.
bxp.errorbar.width	numeric value specifying the width of box plot error bars. Default is 0.4.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).
short.panel.labs	logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.
linetype	line types.
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
linewidth	constant value specifying the line width.
width	numeric value between 0 and 1 specifying the width of the plot elements.
notch	If FALSE (default) make a standard box plot. If TRUE, make a notched box plot. Notches are used to compare groups; if the notches of two boxes do not overlap, this suggests that the medians are significantly different.
outliers	Whether to display (TRUE) or discard (FALSE) outliers from the plot. Hiding or discarding outliers can be useful when, for example, raw data points need to be displayed on top of the boxplot. By discarding outliers, the axis limits will adapt to the box and whiskers only, not the full data range. If outliers need to be hidden and the axes needs to show the full data range, please use outlier.shape = NA instead.
outlier.shape	point shape of outlier. Default is 19. To hide outlier, specify outlier.shape = NA. When jitter is added, then outliers will be automatically hidden.
select	character vector specifying which items to display.
remove	character vector specifying which items to remove from the plot.
order	character vector specifying the order of items.
add	character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.

`add.params` parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

`error.plot` plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...

`label` the name of the column containing point labels. Can be also a character vector with length = nrow(data).

`font.label` a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

`label.select` can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - `top.up` and `top.down`: to display the labels of the top up/down points. For example, `label.select = list(top.up = 10, top.down = 4)`.
  - `criteria`: to filter, for example, by x and y variables values, use this: `label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')")`.

`repel` a logical value, whether to use ggrepel to avoid overplotting text labels or not.

`label.rectangle` logical value. If TRUE, add rectangle underneath the text, making it easier to read.

`position` position adjustment, either as a string, or the result of a call to a position adjustment function (e.g. `position_dodge(0.8)`). Used to control the spacing between grouped elements.

`ggtheme` function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

`show.n` logical. If TRUE, displays the number of observations ("n = <count>") at the top of each group. Off by default. When the groups are dodged (a `color`/`fill` grouping with a dodging `position`), one count is shown per group; otherwise a single count is shown per x-axis tick. Counts respect `select`/`remove` and are computed per facet.

`...` other arguments to be passed to `geom_boxplot`, `ggpar` and `facet`.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab

- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

Suggestions for the argument "add"

Suggested values are one of c("dotplot", "jitter").

See Also

ggpar, ggviolin, ggdotplot and ggstripchart.

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot
# ++++++++++++++++++++++++++++
# width: change box plots width
ggboxplot(df, x = "dose", y = "len", width = 0.8)

# Change orientation: horizontal
ggboxplot(df, "dose", "len", orientation = "horizontal")

# Notched box plot
ggboxplot(df,
  x = "dose", y = "len",
  notch = TRUE
)

# Add dots
# ++++++++++++++++++++++++++
ggboxplot(df,
  x = "dose", y = "len",
  add = "dotplot"
)

# Add jitter points and change the shape by groups
ggboxplot(df,
  x = "dose", y = "len",
  add = "jitter", shape = "dose"
)


# Select and order items
# ++++++++++++++++++++++++++++++
```

```
# Select which items to display: "0.5" and "2"
ggboxplot(df, "dose", "len",
  select = c("0.5", "2")
)

# Change the default order of items
ggboxplot(df, "dose", "len",
  order = c("2", "1", "0.5")
)


# Change colors
# ++++++++++++++++++++++++++
# Change outline and fill colors
ggboxplot(df, "dose", "len",
  color = "black", fill = "gray"
)

# Change outline colors by groups: dose
# Use custom color palette
# Add jitter points and change the shape by groups
ggboxplot(df, "dose", "len",
  color = "dose", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "jitter", shape = "dose"
)

# Change fill color by groups: dose
ggboxplot(df, "dose", "len",
  fill = "dose", palette = c("#00AFBB", "#E7B800", "#FC4E07")
)


# Box plot with multiple groups
# ++++++++++++++++++++
# fill or color box plot by a second group : "supp"
ggboxplot(df, "dose", "len",
  color = "supp",
  palette = c("#00AFBB", "#E7B800")
)
```

---

ggdensity *Density plot*

Description

Create a density plot.

Usage

```
ggdensity(
  data,
  x,
  y = "density",
  combine = FALSE,
  merge = FALSE,
  color = "black",
  fill = NA,
  palette = NULL,
  size = NULL,
  linewidth = NULL,
  linetype = "solid",
  alpha = 0.5,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  add = c("none", "mean", "median"),
  add.params = list(linetype = "dashed"),
  rug = FALSE,
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	variable to be drawn.
y	one of "density" or "count".
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color, fill	density line color and fill color.

`palette` the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".

`size` Numeric value (e.g.: size = 1). change the size of points and outlines.

`linewidth` numeric value specifying line width. This is the preferred parameter for ggplot2 3.4.0+. If `size` is also specified, `linewidth` takes precedence.

`linetype` line type. See `show_line_types`.

`alpha` numeric value specifying fill color transparency. Value should be in [0, 1], where 0 is full transparency and 1 is no transparency.

`title` plot main title.

`xlab` character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

`ylab` character vector specifying y axis labels. Use ylab = FALSE to hide ylab.

`facet.by` character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.

`panel.labs` a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

`short.panel.labs` logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

`add` allowed values are one of "mean" or "median" (for adding mean or median line, respectively).

`add.params` parameters (color, size, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

`rug` logical value. If TRUE, add marginal rug.

`label` the name of the column containing point labels. Can be also a character vector with length = nrow(data).

`font.label` a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

`label.select` can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - `top.up` and `top.down`: to display the labels of the top up/down points. For example, `label.select = list(top.up = 10, top.down = 4)`.
  - `criteria`: to filter, for example, by x and y variables values, use this: `label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')")`.

repel	a logical value, whether to use ggrepel to avoid overplotting text labels or not.
label.rectangle	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to geom_density and ggpar.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

gghistogram and ggpar.

Examples

```
# Create some data format
set.seed(1234)
wdata <- data.frame(
  sex = factor(rep(c("F", "M"), each = 200)),
  weight = c(rnorm(200, 55), rnorm(200, 58))
)

head(wdata, 4)

# Basic density plot
# Add mean line and marginal rug
ggdensity(wdata,
  x = "weight", fill = "lightgray",
  add = "mean", rug = TRUE
)

# Change outline colors by groups ("sex")
# Use custom palette
ggdensity(wdata,
  x = "weight",
  add = "mean", rug = TRUE,
```

```
  color = "sex", palette = c("#00AFBB", "#E7B800")
)


# Change outline and fill colors by groups ("sex")
# Use custom palette
ggdensity(wdata,
  x = "weight",
  add = "mean", rug = TRUE,
  color = "sex", fill = "sex",
  palette = c("#00AFBB", "#E7B800")
)
```

---

ggdonutchart *Donut chart*

Description

Create a donut chart.

Usage

```
ggdonutchart(
  data,
  x,
  label = x,
  lab.pos = c("out", "in"),
  lab.adjust = 0,
  lab.font = c(4, "plain", "black"),
  label.repel = FALSE,
  font.family = "",
  color = "black",
  fill = "white",
  palette = NULL,
  size = NULL,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	variable containing values for drawing.
label	variable specifying the label of each slice.
lab.pos	character specifying the position for labels. Allowed values are "out" (for outside) or "in" (for inside).

lab.adjust	numeric value, used to adjust label position when lab.pos = "in". Increase or decrease this value to see the effect.
lab.font	a vector of length 3 indicating respectively the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of label font. For example *lab.font= c(4, "bold", "red")*.
label.repel	logical. Default is FALSE. If TRUE, the slice labels are placed around the chart with `ggrepel::geom_text_repel()` and connected to their slice with leader lines, so that the labels of many small slices no longer overlap. When TRUE, `lab.pos` is ignored.
font.family	character vector specifying font family.
color, fill	outline and fill colors.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to be passed to ggpar().

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar, ggpie

Examples

```
# Data: Create some data
# +++++++++++++++++++++++++++++

df <- data.frame(
  group = c("Male", "Female", "Child"),
```

```
  value = c(25, 25, 50)
)

head(df)


# Basic pie charts
# ++++++++++++++++++++++++++++++++

ggdonutchart(df, "value", label = "group")


# Change color
# ++++++++++++++++++++++++++++++++

# Change fill color by group
# set line color to white
# Use custom color palette
ggdonutchart(df, "value",
  label = "group",
  fill = "group", color = "white",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)


# Change label
# ++++++++++++++++++++++++++++++++

# Show group names and value as labels
labs <- paste0(df$group, " (", df$value, "%)")
ggdonutchart(df, "value",
  label = labs,
  fill = "group", color = "white",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)

# Change the position and font color of labels
ggdonutchart(df, "value",
  label = labs,
  lab.pos = "in", lab.font = "white",
  fill = "group", color = "white",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)
```

---

ggdotchart *Cleveland's Dot Plots*

Description

Draw a Cleveland dot plot.

Usage

```
ggdotchart(
  data,
  x,
  y,
  group = NULL,
  combine = FALSE,
  color = "black",
  palette = NULL,
  shape = 19,
  size = NULL,
  dot.size = size,
  sorting = c("ascending", "descending", "none"),
  add = c("none", "segment"),
  add.params = list(),
  x.text.col = TRUE,
  rotate = FALSE,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  select = NULL,
  remove = NULL,
  order = NULL,
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  position = "identity",
  ggtheme = theme_pubr(),
  ...
)

theme_cleveland(rotate = TRUE)
```

Arguments

data	a data frame
x, y	x and y variables for drawing.
group	an optional column name indicating how the elements of x are grouped.
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
color, size	points color and size.

palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
shape	point shape. See show_point_shapes.
dot.size	numeric value specifying the dot size.
sorting	a character vector for sorting into ascending or descending order. Allowed values are one of "descending", "ascending" and "none". Partial match are allowed (e.g. sorting = "desc" or "asc"). Default is "descending".
add	character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.
add.params	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").
x.text.col	logical. If TRUE (default), x axis texts are colored by groups.
rotate	logical value. If TRUE, rotate the graph by setting the plot orientation to horizontal.
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).
short.panel.labs	logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.
select	character vector specifying which items to display.
remove	character vector specifying which items to remove from the plot.
order	character vector specifying the order of items.
label	the name of the column containing point labels.
font.label	a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").
label.select	can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - `top.up` and `top.down`: to display the labels of the top up/down points. For example, `label.select = list(top.up = 10, top.down = 4)`.
  - `criteria`: to filter, for example, by x and y variables values, use this: `label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')")`.

`repel` a logical value, whether to use ggrepel to avoid overplotting text labels or not.

`label.rectangle`
logical value. If TRUE, add rectangle underneath the text, making it easier to read.

`position` Position adjustment, either as a string, or the result of a call to a position adjustment function.

`ggtheme` function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

`...` other arguments to be passed to `geom_point` and `ggpar`.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

`ggpar`

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)
df$name <- rownames(df)
head(df[, c("wt", "mpg", "cyl")], 3)

# Basic plot
ggdotchart(df,
  x = "name", y = "mpg",
```

```
  ggtheme = theme_bw()
)

# Change colors by  group cyl
ggdotchart(df,
  x = "name", y = "mpg",
  group = "cyl", color = "cyl",
  palette = c("#999999", "#E69F00", "#56B4E9"),
  rotate = TRUE,
  sorting = "descending",
  ggtheme = theme_bw(),
  y.text.col = TRUE
)


# Plot with multiple groups
# ++++++++++++++++++++
# Create some data
df2 <- data.frame(
  supp = rep(c("VC", "OJ"), each = 3),
  dose = rep(c("D0.5", "D1", "D2"), 2),
  len = c(6.8, 15, 33, 4.2, 10, 29.5)
)
print(df2)

ggdotchart(df2,
  x = "dose", y = "len",
  color = "supp", size = 3,
  add = "segment",
  add.params = list(color = "lightgray", linewidth = 1.5),
  position = position_dodge(0.3),
  palette = "jco",
  ggtheme = theme_pubclean()
)
```

---

ggdotplot *Dot plot*

Description

Create a dot plot.

Usage

```
ggdotplot(
  data,
  x,
  y,
  combine = FALSE,
```

```
  merge = FALSE,
  color = "black",
  fill = "lightgray",
  palette = NULL,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  size = NULL,
  binwidth = NULL,
  select = NULL,
  remove = NULL,
  order = NULL,
  add = "mean_se",
  add.params = list(),
  error.plot = "pointrange",
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	character string containing the name of x variable.
y	character vector containing one or more variables to plot
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	outline color.
fill	fill color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".

`title` plot main title.

`xlab` character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

`ylab` character vector specifying y axis labels. Use ylab = FALSE to hide ylab.

`facet.by` character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.

`panel.labs` a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

`short.panel.labs` logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

`size` Numeric value (e.g.: size = 1). change the size of points and outlines.

`binwidth` numeric value specifying bin width. use value between 0 and 1 when you have a strong dense dotplot. For example binwidth = 0.2.

`select` character vector specifying which items to display.

`remove` character vector specifying which items to remove from the plot.

`order` character vector specifying the order of items.

`add` character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.

`add.params` parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

`error.plot` plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...

`label` the name of the column containing point labels. Can be also a character vector with length = nrow(data).

`font.label` a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

`label.select` can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - `top.up` and `top.down`: to display the labels of the top up/down points. For example, `label.select = list(top.up = 10, top.down = 4)`.

- criteria: to filter, for example, by x and y variables values, use this: label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')").

repel — a logical value, whether to use ggrepel to avoid overplotting text labels or not.

label.rectangle — logical value. If TRUE, add rectangle underneath the text, making it easier to read.

ggtheme — function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

... — other arguments to be passed to geom_dotplot, ggpar and facet.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar, ggviolin, ggboxplot and ggstripchart.

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot with summary statistics : mean_sd
# ++++++++++++++++++++++++++++
ggdotplot(df,
  x = "dose", y = "len",
  add = "mean_sd"
)

# Change error.plot to "crossbar"
ggdotplot(df,
  x = "dose", y = "len",
  add = "mean_sd", add.params = list(width = 0.5),
  error.plot = "crossbar"
)
```

```
# Add box plot
ggdotplot(df,
  x = "dose", y = "len",
  add = "boxplot"
)

# Add violin + mean_sd
ggdotplot(df,
  x = "dose", y = "len",
  add = c("violin", "mean_sd")
)


# Change colors
# ++++++++++++++++++++++++++++
# Change fill and outline colors by groups: dose
# Use custom color palette
ggdotplot(df, "dose", "len",
  add = "boxplot",
  color = "dose", fill = "dose",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)


# Plot with multiple groups
# +++++++++++++++++++++
# Change color by a second group : "supp"
ggdotplot(df, "dose", "len",
  fill = "supp", color = "supp",
  palette = c("#00AFBB", "#E7B800")
)
```

---

ggecdf *Empirical cumulative density function*

Description

Empirical Cumulative Density Function (ECDF).

Usage

```
ggecdf(
  data,
  x,
  combine = FALSE,
  merge = FALSE,
  color = "black",
```

```
  palette = NULL,
  size = NULL,
  linetype = "solid",
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	variable to be drawn.
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	line and point color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
size	line and point size.
linetype	line type. See show_line_types.
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).
short.panel.labs	logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to stat_ecdf and ggpar.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar

Examples

```
# Create some data format
set.seed(1234)
wdata <- data.frame(
  sex = factor(rep(c("F", "M"), each = 200)),
  weight = c(rnorm(200, 55), rnorm(200, 58))
)

head(wdata, 4)

# Basic ECDF plot
ggecdf(wdata, x = "weight")

# Change colors and linetype by groups ("sex")
# Use custom palette
ggecdf(wdata,
  x = "weight",
  color = "sex", linetype = "sex",
  palette = c("#00AFBB", "#E7B800")
)
```

ggerrorplot *Visualizing Error*

Description

Visualizing error.

Usage

```
ggerrorplot(
  data,
  x,
  y,
  desc_stat = "mean_se",
  numeric.x.axis = FALSE,
  combine = FALSE,
  merge = FALSE,
  color = "black",
  fill = "white",
  palette = NULL,
  size = NULL,
  width = NULL,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  select = NULL,
  remove = NULL,
  order = NULL,
  add = "none",
  add.params = list(),
  error.plot = "pointrange",
  ci = 0.95,
  position = position_dodge(),
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x, y	x and y variables for drawing.
desc_stat	descriptive statistics to be used for visualizing errors. Default value is "mean_se". Allowed values are one of , "mean", "mean_se", "mean_sd", "mean_ci", "mean_range",

	"median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see desc_statby for more details.
numeric.x.axis	logical. If TRUE, x axis will be treated as numeric. Default is FALSE.
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color, fill	outline and fill colors.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
width	numeric value between 0 and 1 specifying the width of the plot elements.
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).
short.panel.labs	logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.
select	character vector specifying which items to display.
remove	character vector specifying which items to remove from the plot.
order	character vector specifying the order of items. Considered only when x axis is a factor variable.
add	character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.
add.params	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

error.plot	plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...
ci	the percent range of the confidence interval (default is 0.95).
position	position adjustment, either as a string, or the result of a call to a position adjustment function (e.g. position_dodge(0.8)). Used to control the spacing between grouped elements.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to be passed to ggpar().

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar, ggline

Examples

```
# Data: ToothGrowth data set we'll be used.
df <- ToothGrowth
head(df, 10)

# Plot mean_se
ggerrorplot(df, x = "dose", y = "len")


# Change desc_stat to mean_sd
# (other values include: mean_sd, mean_ci, median_iqr, ....)
# Add labels
ggerrorplot(df,
  x = "dose", y = "len",
  desc_stat = "mean_sd"
)
```

```
# Change error.plot to "errorbar" and add mean point
# Visualize the mean of each group
ggerrorplot(df,
  x = "dose", y = "len",
  add = "mean", error.plot = "errorbar"
)

# Horizontal plot
ggerrorplot(df,
  x = "dose", y = "len",
  add = "mean", error.plot = "errorbar",
  orientation = "horizontal"
)


# Change error.plot to "crossbar"
ggerrorplot(df,
  x = "dose", y = "len",
  error.plot = "crossbar", width = 0.5
)


# Add jitter points and errors (mean_se)
ggerrorplot(df,
  x = "dose", y = "len",
  add = "jitter"
)

# Add dot and errors (mean_se)
ggerrorplot(df,
  x = "dose", y = "len",
  add = "dotplot"
)

# Multiple groups with error bars and jitter point
ggerrorplot(df,
  x = "dose", y = "len",
  color = "supp", palette = "Paired",
  error.plot = "pointrange",
  position = position_dodge(0.5)
)
#
```

---

ggexport *Export ggplots*

Description

Export ggplots

Usage

```
ggexport(
  ...,
  plotlist = NULL,
  filename = NULL,
  ncol = NULL,
  nrow = NULL,
  width = 480,
  height = 480,
  pointsize = 12,
  res = NA,
  verbose = TRUE
)
```

Arguments

...	list of plots to be arranged into the grid. The plots can be either ggplot2 plot objects, arbitrary gtables or an object of class ggarrange.
plotlist	(optional) list of plots to display.
filename	File name to create on disk.
ncol	(optional) number of columns in the plot grid.
nrow	(optional) number of rows in the plot grid.
width, height	plot width and height, respectively (example, width = 800, height = 800). Applied only to raster plots: "png", "jpeg", "jpg", "bmp" and "tiff".
pointsize	the default pointsize of plotted text (example, pointsize = 8). Used only for raster plots.
res	the resolution in ppi (example, res = 250). Used only for raster plots.
verbose	logical. If TRUE, show message.

Author(s)

Laszlo Erdey <erdey.laszlo@econ.unideb.hu>

Examples

```
## Not run:
require("magrittr")
# Load data
data("ToothGrowth")
df <- ToothGrowth
df$dose <- as.factor(df$dose)

# Box plot
bxp <- ggboxplot(df,
  x = "dose", y = "len",
  color = "dose", palette = "jco"
)
```

```
# Dot plot
dp <- ggdotplot(df,
  x = "dose", y = "len",
  color = "dose", palette = "jco"
)
# Density plot
dens <- ggdensity(df, x = "len", fill = "dose", palette = "jco")

# Export to pdf
ggarrange(bxp, dp, dens, ncol = 2) %>%
  ggexport(filename = "test.pdf")

# Export to png
ggarrange(bxp, dp, dens, ncol = 2) %>%
  ggexport(filename = "test.png")

## End(Not run)
```

---

gghistogram *Histogram plot*

Description

Create a histogram plot.

Usage

```
gghistogram(
  data,
  x,
  y = "count",
  combine = FALSE,
  merge = FALSE,
  weight = NULL,
  color = "black",
  fill = NA,
  palette = NULL,
  size = NULL,
  linetype = "solid",
  linewidth = NULL,
  alpha = 0.5,
  bins = NULL,
  binwidth = NULL,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
```

```
  panel.labs = NULL,
  short.panel.labs = TRUE,
  add = c("none", "mean", "median"),
  add.params = list(linetype = "dashed"),
  rug = FALSE,
  add_density = FALSE,
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  position = position_identity(),
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	variable to be drawn.
y	one of "density" or "count".
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
weight	a variable name available in the input data for creating a weighted histogram.
color, fill	histogram line color and fill color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
linetype	line type. See show_line_types.
linewidth	numeric value specifying the line width.
alpha	numeric value specifying fill color transparency. Value should be in [0, 1], where 0 is full transparency and 1 is no transparency.
bins	Number of bins. Defaults to 30.
binwidth	numeric value specifying bin width. use value between 0 and 1 when you have a strong dense dotplot. For example binwidth = 0.2.
title	plot main title.

`xlab` character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

`ylab` character vector specifying y axis labels. Use ylab = FALSE to hide ylab.

`facet.by` character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.

`panel.labs` a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

`short.panel.labs` logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

`add` allowed values are one of "mean" or "median" (for adding mean or median line, respectively).

`add.params` parameters (color, size, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

`rug` logical value. If TRUE, add marginal rug.

`add_density` logical value. If TRUE, add density curves.

`label` the name of the column containing point labels. Can be also a character vector with length = nrow(data).

`font.label` a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

`label.select` can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - `top.up` and `top.down`: to display the labels of the top up/down points. For example, `label.select = list(top.up = 10, top.down = 4)`.
  - `criteria`: to filter, for example, by x and y variables values, use this: `label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')")`.

`repel` a logical value, whether to use ggrepel to avoid overplotting text labels or not.

`label.rectangle` logical value. If TRUE, add rectangle underneath the text, making it easier to read.

`position` Position adjustment, either as a string, or the result of a call to a position adjustment function. Allowed values include "identity", "stack", "dodge".

`ggtheme` function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

`...` other arguments to be passed to `geom_histogram` and `ggpar`.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggdensity and ggpar

Examples

```
# Create some data format
set.seed(1234)
wdata <- data.frame(
  sex = factor(rep(c("F", "M"), each = 200)),
  weight = c(rnorm(200, 55), rnorm(200, 58))
)

head(wdata, 4)

# Basic density plot
# Add mean line and marginal rug
gghistogram(wdata,
  x = "weight", fill = "lightgray",
  add = "mean", rug = TRUE
)

# Change outline colors by groups ("sex")
# Use custom color palette
gghistogram(wdata,
  x = "weight",
  add = "mean", rug = TRUE,
  color = "sex", palette = c("#00AFBB", "#E7B800")
)

# Change outline and fill colors by groups ("sex")
# Use custom color palette
gghistogram(wdata,
  x = "weight",
  add = "mean", rug = TRUE,
  color = "sex", fill = "sex",
  palette = c("#00AFBB", "#E7B800")
)
```

```
# Combine histogram and density plots
gghistogram(wdata,
  x = "weight",
  add = "mean", rug = TRUE,
  fill = "sex", palette = c("#00AFBB", "#E7B800"),
  add_density = TRUE
)

# Weighted histogram
gghistogram(iris, x = "Sepal.Length", weight = "Petal.Length")
```

---

ggline *Line plot*

Description

Create a line plot.

Usage

```
ggline(
  data,
  x,
  y,
  group = 1,
  numeric.x.axis = FALSE,
  combine = FALSE,
  merge = FALSE,
  color = "black",
  palette = NULL,
  linetype = "solid",
  plot_type = c("b", "l", "p"),
  size = NULL,
  linewidth = NULL,
  shape = 19,
  stroke = NULL,
  point.size = NULL,
  point.color = color,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  select = NULL,
  remove = NULL,
  order = NULL,
  add = "none",
```

```
  add.params = list(),
  error.plot = "errorbar",
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  show.line.label = FALSE,
  position = "identity",
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x, y	x and y variables for drawing.
group	grouping variable to connect points by line. Allowed values are 1 (for one line, one group) or a character vector specifying the name of the grouping variable (case of multiple lines).
numeric.x.axis	logical. If TRUE, x axis will be treated as numeric. Default is FALSE.
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	line colors.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
linetype	line type.
plot_type	plot type. Allowed values are one of "b" for both line and point; "l" for line only; and "p" for point only. Default is "b".
size	line size. Deprecated in ggplot2 v >= 3.4.0, use `linewidth` instead.
linewidth	line width. Default is 0.5. Recommended parameter for ggplot2 version >= 3.4.0. If both `size` and `linewidth` are specified, an error is thrown.
shape	point shapes.
stroke	point stroke. Used only for shapes 21-24 to control the thickness of points border.
point.size	point size.

point.color — point color.

title — plot main title.

xlab — character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

ylab — character vector specifying y axis labels. Use ylab = FALSE to hide ylab.

facet.by — character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.

panel.labs — a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

short.panel.labs — logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

select — character vector specifying which items to display.

remove — character vector specifying which items to remove from the plot.

order — character vector specifying the order of items.

add — character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.

add.params — parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

error.plot — plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...

label — the name of the column containing point labels. Can be also a character vector with length = nrow(data).

font.label — a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

label.select — can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - top.up and top.down: to display the labels of the top up/down points. For example, label.select = list(top.up = 10, top.down = 4).
  - criteria: to filter, for example, by x and y variables values, use this: label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')").

repel	a logical value, whether to use ggrepel to avoid overplotting text labels or not.
label.rectangle	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
show.line.label	logical value. If TRUE, shows line labels.
position	position adjustment, either as a string, or the result of a call to a position adjustment function (e.g. position_dodge(0.8)). Used to control the spacing between grouped elements.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to geom_dotplot.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar, ggbarplot

Examples

```
# Data
df <- data.frame(
  dose = c("D0.5", "D1", "D2"),
  len = c(4.2, 10, 29.5)
)
print(df)

# Basic plot
# ++++++++++++++++++++++++++++
ggline(df, x = "dose", y = "len")


# Plot with multiple groups
# ++++++++++++++++++++
```

```
# Create some data
df2 <- data.frame(
  supp = rep(c("VC", "OJ"), each = 3),
  dose = rep(c("D0.5", "D1", "D2"), 2),
  len = c(6.8, 15, 33, 4.2, 10, 29.5)
)
print(df2)

# Plot "len" by "dose" and
# Change line types and point shapes by a second groups: "supp"
ggline(df2, "dose", "len",
  linetype = "supp", shape = "supp"
)


# Change colors
# ++++++++++++++++++++

# Change color by group: "supp"
# Use custom color palette
ggline(df2, "dose", "len",
  linetype = "supp", shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800")
)


# Add points and errors
# ++++++++++++++++++++++++++

# Data: ToothGrowth data set we'll be used.
df3 <- ToothGrowth
head(df3, 10)

# It can be seen that for each group we have
# different values
ggline(df3, x = "dose", y = "len")

# Visualize the mean of each group
ggline(df3,
  x = "dose", y = "len",
  add = "mean"
)

# Add error bars: mean_se
# (other values include: mean_sd, mean_ci, median_iqr, ....)
# Add labels
ggline(df3, x = "dose", y = "len", add = "mean_se")

# Change error.plot to "pointrange"
ggline(df3,
  x = "dose", y = "len",
  add = "mean_se", error.plot = "pointrange"
)
```

```
# Add jitter points and errors (mean_se)
ggline(df3,
  x = "dose", y = "len",
  add = c("mean_se", "jitter")
)

# Add dot and errors (mean_se)
ggline(df3,
  x = "dose", y = "len",
  add = c("mean_se", "dotplot"), color = "steelblue"
)

# Add violin and errors (mean_se)
ggline(df3,
  x = "dose", y = "len",
  add = c("mean_se", "violin"), color = "steelblue"
)

# Multiple groups with error bars
# ++++++++++++++++++++++

ggline(df3,
  x = "dose", y = "len", color = "supp",
  add = "mean_se", palette = c("#00AFBB", "#E7B800")
)

# Add jitter
ggline(df3,
  x = "dose", y = "len", color = "supp",
  add = c("mean_se", "jitter"), palette = c("#00AFBB", "#E7B800")
)

# Add dot plot
ggline(df3,
  x = "dose", y = "len", color = "supp",
  add = c("mean_se", "dotplot"), palette = c("#00AFBB", "#E7B800")
)
#
```

---

ggmaplot *MA-plot from means and log fold changes*

Description

Make MA-plot which is a scatter plot of log2 fold changes (M, on the y-axis) versus the average expression signal (A, on the x-axis). `M = log2(x/y)` and `A = (log2(x) + log2(y))/2 = log2(xy)*1/2`, where x and y are respectively the mean of the two groups being compared.

Usage

```
ggmaplot(
  data,
  fdr = 0.05,
  fc = 1.5,
  genenames = NULL,
  detection_call = NULL,
  size = NULL,
  alpha = 1,
  seed = 42,
  font.label = c(12, "plain", "black"),
  label.rectangle = FALSE,
  palette = c("#B31B21", "#1465AC", "darkgray"),
  top = 15,
  select.top.method = c("padj", "fc"),
  label.select = NULL,
  facet.by = NULL,
  main = NULL,
  xlab = "Log2 mean expression",
  ylab = "Log2 fold change",
  line.color = "black",
  ggtheme = theme_classic(),
  ...
)
```

Arguments

data	an object of class DESeqResults, get_diff, DE_Results, matrix or data frame containing the columns baseMean (or baseMeanLog2), log2FoldChange, and padj. Rows are genes. Two possible formats are accepted for the input data: • 1/ `baseMean | log2FoldChange | padj`. This is a typical output from DESeq2 pipeline. Here, we'll use log2(baseMean) as the x-axis variable. • 2/ `baseMeanLog2 | log2FoldChange | padj`. Here, baseMeanLog2 is assumed to be the mean of logged values; so we'll use it as the x-axis variable without any transformation. This is the real A in MA plot. In other words, it is the average of two log-scales values: `A = (log2(x) + log2(y))/2 = log2(xy)*1/2` Terminology: • baseMean: the mean expression of genes in the two groups. • log2FoldChange: the log2 fold changes of group 2 compared to group 1 • padj: the adjusted p-value of the used statistical test.
fdr	Accepted false discovery rate for considering genes as differentially expressed.
fc	the fold change threshold. Only genes with a fold change >= fc and padj <= fdr are considered as significantly differentially expressed.

genenames	a character vector of length nrow(data) specifying gene names corresponding to each row. Used for point labels.
detection_call	a numeric vector with length = nrow(data), specifying if the genes is expressed (value = 1) or not (value = 0). For example detection_call = c(1, 1, 0, 1, 0, 1). Default is NULL. If detection_call column is available in data, it will be used.
size	points size.
alpha	numeric value between 0 and 1 specifying point alpha for controlling transparency. For example, use alpha = 0.5.
seed	Random seed passed to set.seed. if NA, set.seed will not be called. Default is 42 for reproducibility.
font.label	a vector of length 3 indicating respectively the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of point labels. For example *font.label = c(14, "bold", "red")*.
label.rectangle	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
top	the number of top genes to be shown on the plot. Use top = 0 to hide to gene labels.
select.top.method	methods to be used for selecting top genes. Allowed values include "padj" and "fc" for selecting by adjusted p values or fold changes, respectively.
label.select	character vector specifying some labels to show.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels (one MA plot per group). The variable(s) must be columns of data (supplied as a data frame). The top genes are selected *per panel*; point colors use the same significance thresholds in every panel. Default is NULL (no faceting), in which case the output is unchanged.
main	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
line.color	color of the horizontal threshold lines (the central line at 0 and the two fold-change cutoff lines). Default is "black".
ggtheme	function, ggplot2 theme name. Default value is theme_classic(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set().
...	other arguments to be passed to ggpar.

Value

a ggplot.

Examples

```
data(diff_express)

# Default plot
ggmaplot(diff_express,
  main = expression("Group 1" %->% "Group 2"),
  fdr = 0.05, fc = 2, size = 0.4,
  palette = c("#B31B21", "#1465AC", "darkgray"),
  genenames = as.vector(diff_express$name),
  legend = "top", top = 20,
  font.label = c("bold", 11),
  font.legend = "bold",
  font.main = "bold",
  ggtheme = ggplot2::theme_minimal()
)

# Add rectangle around labels
ggmaplot(diff_express,
  main = expression("Group 1" %->% "Group 2"),
  fdr = 0.05, fc = 2, size = 0.4,
  palette = c("#B31B21", "#1465AC", "darkgray"),
  genenames = as.vector(diff_express$name),
  legend = "top", top = 20,
  font.label = c("bold", 11), label.rectangle = TRUE,
  font.legend = "bold",
  font.main = "bold",
  ggtheme = ggplot2::theme_minimal()
)

# Select specific genes to show
# set top = 0, then specify genes using label.select argument
ggmaplot(diff_express,
  main = expression("Group 1" %->% "Group 2"),
  fdr = 0.05, fc = 2, size = 0.4,
  genenames = as.vector(diff_express$name),
  ggtheme = ggplot2::theme_minimal(),
  top = 0, label.select = c("BUB1", "CD83")
)
```

---

ggpaired *Plot Paired Data*

Description

Plot paired data.

Usage

```
ggpaired(
  data,
  cond1,
  cond2,
  x = NULL,
  y = NULL,
  id = NULL,
  color = "black",
  fill = "white",
  palette = NULL,
  width = 0.5,
  point.size = 1.2,
  line.size = 0.5,
  line.color = "black",
  linetype = "solid",
  title = NULL,
  xlab = "Condition",
  ylab = "Value",
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  ggtheme = theme_pubr(),
  jitter = 0,
  ...
)
```

Arguments

data	a data frame
cond1	variable name corresponding to the first condition.
cond2	variable name corresponding to the second condition.
x, y	x and y variables, where x is a grouping variable and y contains values for each group. Considered only when cond1 and cond2 are missing.
id	variable name corresponding to paired samples' id. Used to connect paired points with lines.
color	points and box plot colors. To color by conditions, use color = "condition".
fill	box plot fill color. To change fill color by conditions, use fill = "condition".
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes

from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".

`width` box plot width.

`point.size, line.size` point and line size, respectively.

`line.color` line color.

`linetype` line type.

`title` plot main title.

`xlab` character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

`ylab` character vector specifying y axis labels. Use ylab = FALSE to hide ylab.

`facet.by` character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.

`panel.labs` a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

`short.panel.labs` logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

`label` the name of the column containing point labels. Can be also a character vector with length = nrow(data).

`font.label` a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

`label.select` can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - `top.up` and `top.down`: to display the labels of the top up/down points. For example, `label.select = list(top.up = 10, top.down = 4)`.
  - `criteria`: to filter, for example, by x and y variables values, use this: `label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')")`.

`repel` a logical value, whether to use ggrepel to avoid overplotting text labels or not.

`label.rectangle` logical value. If TRUE, add rectangle underneath the text, making it easier to read.

`ggtheme` function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

jitter	numeric value (default 0, no jitter) giving the amount of horizontal jitter added to the paired points to reduce overlap. Points are nudged sideways within `[-jitter, jitter]`; each subject (`id`) gets a single offset so its two points move together and the connecting line stays intact. Only the horizontal positions change (the values are never moved). Typical values are small relative to the box `width` (e.g. `jitter = 0.05` to `0.1`). `jitter = 0` leaves the plot unchanged.
...	other arguments to be passed to be passed to ggpar().

Examples

```
# Example 1
# ::::::::::::::::::::::::::::::::::::::::
before <- c(200.1, 190.9, 192.7, 213, 241.4, 196.9, 172.2, 185.5, 205.2, 193.7)
after <- c(392.9, 393.2, 345.1, 393, 434, 427.9, 422, 383.9, 392.3, 352.2)

d <- data.frame(before = before, after = after)
ggpaired(d,
  cond1 = "before", cond2 = "after",
  fill = "condition", palette = "jco"
)

# Example 2
# ::::::::::::::::::::::::::::::::::::::::
ggpaired(ToothGrowth,
  x = "supp", y = "len",
  color = "supp", line.color = "gray", line.size = 0.4,
  palette = "npg"
)
```

---

ggpar *Graphical parameters*

Description

Graphical parameters

Usage

```
ggpar(
  p,
  palette = NULL,
  gradient.cols = NULL,
  main = NULL,
  submain = NULL,
  caption = NULL,
  xlab = NULL,
  ylab = NULL,
```

```
  title = NULL,
  subtitle = NULL,
  font.main = NULL,
  font.submain = NULL,
  font.x = NULL,
  font.y = NULL,
  font.caption = NULL,
  font.title = NULL,
  font.subtitle = NULL,
  font.family = "",
  xlim = NULL,
  ylim = NULL,
  xscale = c("none", "log2", "log10", "sqrt"),
  yscale = c("none", "log2", "log10", "sqrt"),
  format.scale = FALSE,
  legend = NULL,
  legend.direction = NULL,
  legend.title = NULL,
  font.legend = NULL,
  ticks = TRUE,
  tickslab = TRUE,
  font.tickslab = NULL,
  font.xtickslab = font.tickslab,
  font.ytickslab = font.tickslab,
  x.text.angle = NULL,
  y.text.angle = NULL,
  xtickslab.rt = x.text.angle,
  ytickslab.rt = y.text.angle,
  xticks.by = NULL,
  yticks.by = NULL,
  rotate = FALSE,
  orientation = c("vertical", "horizontal", "reverse"),
  ggtheme = NULL,
  ...
)
```

Arguments

p	an object of class ggplot or a list of ggplots
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty". Can be also a numeric vector of length(groups); in this case a basic color palette is created using the function palette.
gradient.cols	vector of colors to use for n-colour gradient. Allowed values include brewer and ggsci color palettes.
main	plot main title.

submain, subtitle
: plot subtitle.

caption
: plot caption.

xlab
: character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

ylab
: character vector specifying y axis labels. Use ylab = FALSE to hide ylab.

title
: plot main title.

font.main, font.submain, font.caption, font.x, font.y
: a vector of length 3 indicating respectively the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of main title, subtitle, caption, xlab and ylab, respectively. For example *font.x = c(14, "bold", "red")*. Use font.x = 14, to change only font size; or use font.x = "bold", to change only font face.

font.title, font.subtitle
: alias of font.submain and font.submain, respectively.

font.family
: character vector specifying font family.

xlim, ylim
: a numeric vector of length 2, specifying x and y axis limits (minimum and maximum), respectively. e.g.: ylim = c(0, 50).

xscale, yscale
: x and y axis scale, respectively. Allowed values are one of c("none", "log2", "log10", "sqrt"); e.g.: yscale="log2".

format.scale
: logical value. If TRUE, axis tick mark labels will be formatted when xscale or yscale = "log2" or "log10".

legend
: character specifying legend position. Allowed values are one of c("top", "bottom", "left", "right", "none"). To remove the legend use legend = "none". Legend position can be also specified using a numeric vector c(x, y); see details section.

legend.direction
: character specifying the layout of legend items. Allowed values are "horizontal" or "vertical". Useful to override the default direction, e.g. to stack items vertically with `legend = "bottom"`.

legend.title
: legend title, e.g.: `legend.title = "Species"`. Can be also a list, `legend.title = list(color = "Species", linetype = "Species", shape = "Species")`.

font.legend
: legend text font style; e.g.: font.legend = c(10, "plain", "black").

ticks
: logical value. Default is TRUE. If FALSE, hide axis tick marks.

tickslab
: logical value. Default is TRUE. If FALSE, hide axis tick labels.

font.tickslab, font.xtickslab, font.ytickslab
: Font style (size, face, color) for tick labels, e.g.: c(14, "bold", "red").

x.text.angle, y.text.angle
: Numeric value specifying the rotation angle of x and y axis tick labels, respectively. Default value is NULL. For vertical x axis texts use x.text.angle = 90.

xtickslab.rt, ytickslab.rt
: Same as x.text.angle and y.text.angle, respectively. Will be deprecated in the near future.

xticks.by, yticks.by
: numeric value controlling x and y axis breaks, respectively. For example, if yticks.by = 5, a tick mark is shown on every 5. Default value is NULL.

rotate
: logical value. If TRUE, rotate the graph by setting the plot orientation to horizontal.

orientation
: change the orientation of the plot. Allowed values are one of c("vertical", "horizontal", "reverse"). Partial match is allowed.

ggtheme
: function, ggplot2 theme name. The default is set by each function's ggtheme argument; see the function usage for the actual default. Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

...
: not used

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic box plot
# ++++++++++++++++++++++++++++

p <- ggboxplot(df, x = "dose", y = "len")

# Change the plot orientation: horizontal
ggpar(p, orientation = "horiz")


# Change main title and axis labels
# ++++++++++++++++++++++++++++

ggpar(p,
  main = "Plot of length \n by dose",
  xlab = "Dose (mg)", ylab = "Length"
)

# Title font styles: 'plain', 'italic', 'bold', 'bold.italic'
ggpar(p,
  main = "Length by dose",
  font.main = c(14, "bold.italic", "red"),
  font.x = c(14, "bold", "#2E9FDF"),
  font.y = c(14, "bold", "#E7B800")
)

# Hide axis labels
ggpar(p, xlab = FALSE, ylab = FALSE)


# Change colors
```

```
# ++++++++++++++++++++++

# Change outline colors by groups: dose
p2 <- ggboxplot(df, "dose", "len", color = "dose")
p2

# Use custom color palette
ggpar(p2, palette = c("#00AFBB", "#E7B800", "#FC4E07"))

# Use brewer palette
ggpar(p2, palette = "Dark2")

# Use grey palette
ggpar(p2, palette = "grey")

# Use scientific journal palette from ggsci package
ggpar(p2, palette = "npg") # nature

# Axis ticks, limits, scales
# +++++++++++++++++++++++++

# Axis ticks labels and rotation
ggpar(p,
  font.tickslab = c(14, "bold", "#993333"),
  xtickslab.rt = 45, ytickslab.rt = 45
)
# Hide axis ticks and tick labels
ggpar(p, ticks = FALSE, tickslab = FALSE)

# Axis limits
ggpar(p, ylim = c(0, 50))

# Axis scale
ggpar(p, yscale = "log2")

# Format axis scale
ggpar(p, yscale = "log2", format.scale = TRUE)

# Legends
# ++++++++++++++++++
# Change legend position and title
ggpar(p2,
  legend = "right", legend.title = "Dose (mg)",
  font.legend = c(10, "bold", "red")
)
```

---

ggparagraph *Draw a Paragraph of Text*

Description

Draw a paragraph o text. Splits a long text into multiple lines (by inserting line breaks) so that the output will fit within the current viewport.

Usage

```
ggparagraph(
  text,
  color = NULL,
  size = NULL,
  face = NULL,
  family = NULL,
  lineheight = NULL
)

## S3 method for class 'splitText'
drawDetails(x, recording)
```

Arguments

text	the text to plot.
color	font color, example: color = "black"
size	font size, example: size = 12
face	font face. Allowed values are one of "plain", "italic", "bold", "bold.italic".
family	font family
lineheight	Line height, example: lineheight = 2.
x	a grid grob
recording	a logical value indicating whether a grob is being added to the display list or redrawn from the display list.

Author(s)

Laszlo Erdey <erdey.laszlo@econ.unideb.hu>

Examples

```
# Density plot
density.p <- ggdensity(iris,
  x = "Sepal.Length",
  fill = "Species", palette = "jco"
)

# Text plot
text <- paste("iris data set gives the measurements in cm",
  "of the variables sepal length and width",
  "and petal length and width, respectively,",
  "for 50 flowers from each of 3 species of iris.",
  "The species are Iris setosa, versicolor, and virginica.",
```

```
  sep = " "
)
text.p <- ggparagraph(text, face = "italic", size = 12)

# Arrange the plots on the same page
ggarrange(density.p, text.p,
  ncol = 1, nrow = 2,
  heights = c(1, 0.3)
)
```

---

ggpie *Pie chart*

Description

Create a pie chart.

Usage

```
ggpie(
  data,
  x,
  label = x,
  lab.pos = c("out", "in"),
  lab.adjust = 0,
  lab.font = c(4, "plain", "black"),
  font.family = "",
  color = "black",
  fill = "white",
  palette = NULL,
  size = NULL,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	variable containing values for drawing.
label	variable specifying the label of each slice.
lab.pos	character specifying the position for labels. Allowed values are "out" (for outside) or "in" (for inside).
lab.adjust	numeric value, used to adjust label position when lab.pos = "in". Increase or decrease this value to see the effect.

lab.font	a vector of length 3 indicating respectively the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of label font. For example *lab.font= c(4, "bold", "red")*.
font.family	character vector specifying font family.
color, fill	outline and fill colors.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to be passed to ggpar().

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar, ggline

Examples

```
# Data: Create some data
# +++++++++++++++++++++++++++++++

df <- data.frame(
  group = c("Male", "Female", "Child"),
  value = c(25, 25, 50)
)

head(df)


# Basic pie charts
```

```
# +++++++++++++++++++++++++++++++++

ggpie(df, "value", label = "group")

# Reducing margins around the pie chart
ggpie(df, "value", label = "group") +
  theme(plot.margin = unit(c(-.75, -.75, -.75, -.75), "cm"))


# Change color
# +++++++++++++++++++++++++++++++++

# Change fill color by group
# set line color to white
# Use custom color palette
ggpie(df, "value",
  label = "group",
  fill = "group", color = "white",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)


# Change label
# +++++++++++++++++++++++++++++++++

# Show group names and value as labels
labs <- paste0(df$group, " (", df$value, "%)")
ggpie(df, "value",
  label = labs,
  fill = "group", color = "white",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)

# Change the position and font color of labels
ggpie(df, "value",
  label = labs,
  lab.pos = "in", lab.font = "white",
  fill = "group", color = "white",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)
```

---

ggpubr_args *ggpubr General Arguments Description*

Description

ggpubr General Arguments Description

Arguments

data	a data frame
x	character string containing the name of x variable.
y	character vector containing one or more variables to plot
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	outline color.
fill	fill color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
linetype	line types.
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
select	character vector specifying which items to display.
remove	character vector specifying which items to remove from the plot.
order	character vector specifying the order of items.
add	character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_mad", "median_range"; see ?desc_statby for more details.
add.params	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").
error.plot	plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...
font.label	a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.

ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

---

ggpubr_options *Global Options for ggpubr*

Description

Displays allowed global options in ggpubr.

Usage

```
ggpubr_options()
```

Examples

```
ggpubr_options()
```

---

ggqqplot *QQ Plots*

Description

Quantile-Quantile plot.

Usage

```
ggqqplot(
  data,
  x,
  combine = FALSE,
  merge = FALSE,
  color = "black",
  palette = NULL,
  size = NULL,
  shape = NULL,
  add = c("qqline", "none"),
  add.params = list(linetype = "solid"),
  conf.int = TRUE,
  conf.int.level = 0.95,
```

```
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	variable to be drawn.
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	point color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
size	point size.
shape	point shape.
add	character vector. Allowed values are one of "none" and "qqline" (for adding qqline).
add.params	parameters (color, size, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").
conf.int	logical value. If TRUE, confidence interval is added.
conf.int.level	the confidence level. Default value is 0.95.
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

short.panel.labs
: logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

ggtheme
: function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

...
: other arguments to be passed to ggpar.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar

Examples

```
# Create some data format
set.seed(1234)
wdata <- data.frame(
  sex = factor(rep(c("F", "M"), each = 200)),
  weight = c(rnorm(200, 55), rnorm(200, 58))
)

head(wdata, 4)

# Basic QQ plot
ggqqplot(wdata, x = "weight")

# Change colors and shape by groups ("sex")
# Use custom palette
ggqqplot(wdata,
  x = "weight",
  color = "sex", palette = c("#00AFBB", "#E7B800")
)
```

ggscatter *Scatter plot*

Description

Create a scatter plot.

Usage

```
ggscatter(
  data,
  x,
  y,
  combine = FALSE,
  merge = FALSE,
  color = "black",
  fill = "lightgray",
  palette = NULL,
  shape = 19,
  size = 2,
  point = TRUE,
  rug = FALSE,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  add = c("none", "reg.line", "loess"),
  add.params = list(),
  conf.int = FALSE,
  conf.int.level = 0.95,
  fullrange = FALSE,
  ellipse = FALSE,
  ellipse.level = 0.95,
  ellipse.type = "norm",
  ellipse.alpha = 0.1,
  ellipse.border.remove = FALSE,
  mean.point = FALSE,
  mean.point.size = ifelse(is.numeric(size), 2 * size, size),
  star.plot = FALSE,
  star.plot.lty = 1,
  star.plot.lwd = NULL,
  label = NULL,
  font.label = c(12, "plain"),
  font.family = "",
  label.select = NULL,
```

```
  repel = FALSE,
  label.rectangle = FALSE,
  parse = FALSE,
  cor.coef = FALSE,
  cor.coeff.args = list(),
  cor.method = "pearson",
  cor.coef.coord = c(NULL, NULL),
  cor.coef.size = 4,
  ggp = NULL,
  show.legend.text = NA,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x	x variables for drawing.
y	y variables for drawing.
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color, fill	point colors.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
shape	point shape. See show_point_shapes.
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
point	logical value. If TRUE, show points.
rug	logical value. If TRUE, add marginal rug.
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).

short.panel.labs
: logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.

add
: allowed values are one of "none", "reg.line" (for adding linear regression line) or "loess" (for adding local regression fitting).

add.params
: parameters (color, size, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

conf.int
: logical value. If TRUE, adds confidence interval.

conf.int.level
: Level controlling confidence region. Default is 95%. Used only when add != "none" and conf.int = TRUE.

fullrange
: should the fit span the full range of the plot, or just the data. Used only when add != "none".

ellipse
: logical value. If TRUE, draws ellipses around points.

ellipse.level
: the size of the concentration ellipse in normal probability.

ellipse.type
: Character specifying frame type. Possible values are "convex", "confidence" or types supported by stat_ellipse() including one of c("t", "norm", "euclid") for plotting concentration ellipses.

  - "convex": plot convex hull of a set o points.
  - "confidence": plot confidence ellipses around group mean points as FactoMineR::coord.ellipse
  - "t": assumes a multivariate t-distribution.
  - "norm": assumes a multivariate normal distribution.
  - "euclid": draws a circle with the radius equal to level, representing the euclidean distance from the center. This ellipse probably won't appear circular unless coord_fixed() is applied.

ellipse.alpha
: Alpha for ellipse specifying the transparency level of fill color. Use alpha = 0 for no fill color.

ellipse.border.remove
: logical value. If TRUE, remove ellipse border lines.

mean.point
: logical value. If TRUE, group mean points are added to the plot.

mean.point.size
: numeric value specifying the size of mean points.

star.plot
: logical value. If TRUE, a star plot is generated.

star.plot.lty, star.plot.lwd
: line type and line width (size) for star plot, respectively.

label
: the name of the column containing point labels. Can be also a character vector with length = nrow(data).

font.label
: a vector of length 3 indicating respectively the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of point labels. For example *font.label = c(14, "bold", "red")*. To specify only the size and the style, use font.label = c(14, "plain").

font.family
: character vector specifying font family.

label.select
: character vector specifying some labels to show.

repel	a logical value, whether to use ggrepel to avoid overplotting text labels or not.
label.rectangle	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
cor.coef	logical value. If TRUE, correlation coefficient with the p-value will be added to the plot.
cor.coeff.args	a list of arguments to pass to the function stat_cor for customizing the displayed correlation coefficients. For example: cor.coeff.args = list(method = "pearson",label.x.npc = "right", label.y.npc = "top").
cor.method	method for computing correlation coefficient. Allowed values are one of "pearson", "kendall", or "spearman".
cor.coef.coord	numeric vector, of length 2, specifying the x and y coordinates of the correlation coefficient. Default values are NULL.
cor.coef.size	correlation coefficient text font size.
ggp	a ggplot. If not NULL, points are added to an existing plot.
show.legend.text	logical. Should text be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to geom_point and ggpar.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

stat_cor, stat_stars, stat_conf_ellipse and ggpar.

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)
head(df[, c("wt", "mpg", "cyl")], 3)

# Basic plot
# ++++++++++++++++++++++++++++
ggscatter(df,
  x = "wt", y = "mpg",
  color = "black", shape = 21, size = 3, # Points color, shape and size
  add = "reg.line", # Add regressin line
  add.params = list(color = "blue", fill = "lightgray"), # Customize reg. line
  conf.int = TRUE, # Add confidence interval
  cor.coef = TRUE, # Add correlation coefficient. see ?stat_cor
  cor.coeff.args = list(method = "pearson", label.x = 3, label.sep = "\n")
)

# loess method: local regression fitting
ggscatter(df,
  x = "wt", y = "mpg",
  add = "loess", conf.int = TRUE
)


# Control point size by continuous variable values ("qsec")
ggscatter(df,
  x = "wt", y = "mpg",
  color = "#00AFBB", size = "qsec"
)


# Change colors
# ++++++++++++++++++++++++++++
# Use custom color palette
# Add marginal rug
ggscatter(df,
  x = "wt", y = "mpg", color = "cyl",
  palette = c("#00AFBB", "#E7B800", "#FC4E07")
)


# Add group ellipses and mean points
# Add stars
# ++++++++++++++++++++
ggscatter(df,
  x = "wt", y = "mpg",
  color = "cyl", shape = "cyl",
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  ellipse = TRUE, mean.point = TRUE,
  star.plot = TRUE
```

```
)


# Textual annotation
# ++++++++++++++++++
df$name <- rownames(df)
ggscatter(df,
  x = "wt", y = "mpg",
  color = "cyl", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  label = "name", repel = TRUE
)
```

---

ggscatterhist *Scatter Plot with Marginal Histograms*

Description

Create a scatter plot with marginal histograms, density plots or box plots.

Usage

```
ggscatterhist(
  data,
  x,
  y,
  group = NULL,
  color = "black",
  fill = NA,
  palette = NULL,
  shape = 19,
  size = 2,
  linetype = "solid",
  bins = 30,
  margin.plot = c("density", "histogram", "boxplot"),
  margin.params = list(),
  margin.ggtheme = theme_void(),
  margin.space = FALSE,
  main.plot.size = 2,
  margin.plot.size = 1,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  legend = "top",
  ggtheme = theme_pubr(),
  print = TRUE,
  ...
)
```

```
## S3 method for class 'ggscatterhist'
print(
  x,
  margin.space = FALSE,
  main.plot.size = 2,
  margin.plot.size = 1,
  title = NULL,
  legend = "top",
  ...
)
```

Arguments

data	a data frame
x	an object of class ggscatterhist.
y	y variables for drawing.
group	a grouping variable. Change points color and shape by groups if the options color and shape are missing. Should be also specified when you want to create a marginal box plot that is grouped.
color, fill	point colors.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
shape	point shape. See show_point_shapes.
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
linetype	line type ("solid", "dashed", ...)
bins	Number of histogram bins. Defaults to 30. Pick a better value that fit to your data.
margin.plot	the type of the marginal plot. Default is "hist".
margin.params	parameters to be applied to the marginal plots.
margin.ggtheme	the theme of the marginal plot. Default is theme_void().
margin.space	logical value. If TRUE, adds space between the main plot and the marginal plot.
main.plot.size	the width of the main plot. Default is 2.
margin.plot.size	the width of the marginal plot. Default is 1.
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
legend	specify the legend position. Allowed values include: "top", "bottom", "left", "right".

ggtheme	the theme to be used for the scatter plot. Default is theme_pubr().
print	logical value. If TRUE (default), print the plot.
...	other arguments passed to the function ggscatter().

Value

an object of class ggscatterhist, which is a list of ggplots, including the following elements:

- sp: main scatter plot;
- xplot: marginal x-axis plot;
- yplot: marginal y-axis plot.

.

User can modify each of plot before printing.

Examples

```
# Basic scatter plot with marginal density plot
ggscatterhist(iris,
  x = "Sepal.Length", y = "Sepal.Width",
  color = "#00AFBB",
  margin.params = list(fill = "lightgray")
)


# Grouped data
ggscatterhist(
  iris,
  x = "Sepal.Length", y = "Sepal.Width",
  color = "Species", size = 3, alpha = 0.6,
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  margin.params = list(fill = "Species", color = "black", size = 0.2)
)

# Use boxplot as marginal
ggscatterhist(
  iris,
  x = "Sepal.Length", y = "Sepal.Width",
  color = "Species", size = 3, alpha = 0.6,
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  margin.plot = "boxplot",
  ggtheme = theme_bw()
)

# Add vertical and horizontal line to a ggscatterhist
plots <- ggscatterhist(iris, x = "Sepal.Length", y = "Sepal.Width", print = FALSE)
plots$sp <- plots$sp +
  geom_hline(yintercept = 3, linetype = "dashed", color = "blue") +
  geom_vline(xintercept = 6, linetype = "dashed", color = "red")
plots
```

ggstripchart *Stripcharts*

Description

Create a stripchart, also known as one dimensional scatter plots. These plots are suitable compared to box plots when sample sizes are small.

Usage

```
ggstripchart(
  data,
  x,
  y,
  combine = FALSE,
  merge = FALSE,
  color = "black",
  fill = "white",
  palette = NULL,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
  short.panel.labs = TRUE,
  shape = 19,
  size = NULL,
  select = NULL,
  remove = NULL,
  order = NULL,
  add = "mean_se",
  add.params = list(),
  error.plot = "pointrange",
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  jitter = 0.2,
  position = position_jitter(jitter, seed = 123),
  ggtheme = theme_pubr(),
  show.n = FALSE,
  ...
)
```

Arguments

data	a data frame

x	character string containing the name of x variable.
y	character vector containing one or more variables to plot
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	outline color.
fill	fill color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).
short.panel.labs	logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.
shape	point shape
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
select	character vector specifying which items to display.
remove	character vector specifying which items to remove from the plot.
order	character vector specifying the order of items.
add	character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.
add.params	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

`error.plot` plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...

`label` the name of the column containing point labels. Can be also a character vector with length = nrow(data).

`font.label` a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

`label.select` can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - `top.up` and `top.down`: to display the labels of the top up/down points. For example, `label.select = list(top.up = 10, top.down = 4)`.
  - `criteria`: to filter, for example, by x and y variables values, use this: `label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')")`.

`repel` a logical value, whether to use ggrepel to avoid overplotting text labels or not.

`label.rectangle` logical value. If TRUE, add rectangle underneath the text, making it easier to read.

`jitter` the amount of jitter.

`position` position adjustment, either as a string, or the result of a call to a position adjustment function. Used to adjust position for multiple groups.

`ggtheme` function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

`show.n` logical. If TRUE, displays the number of observations ("n = <count>") at the top of each group. Off by default. When the groups are dodged (a `color`/`fill` grouping with a dodging `position`), one count is shown per group; otherwise a single count is shown per x-axis tick. Counts respect `select`/`remove` and are computed per facet.

`...` other arguments to be passed to `geom_jitter`, `ggpar` and `facet`.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))

- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar, ggviolin, ggdotplot and ggboxplot.

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot with summary statistics: mean_se
# ++++++++++++++++++++++++++++
# Change point shapes by groups: "dose"
ggstripchart(df,
  x = "dose", y = "len",
  shape = "dose", size = 3,
  add = "mean_se"
)

# Use mean_sd
# Change error.plot to "crossbar"
ggstripchart(df,
  x = "dose", y = "len",
  shape = "dose", size = 3,
  add = "mean_sd", add.params = list(width = 0.5),
  error.plot = "crossbar"
)


# Add summary statistics
# ++++++++++++++++++++++++++

# Add box plot
ggstripchart(df,
  x = "dose", y = "len",
  shape = "dose", add = "boxplot"
)

# Add violin + mean_sd
ggstripchart(df,
  x = "dose", y = "len",
  shape = "dose", add = c("violin", "mean_sd")
)


# Change colors
```

```
# ++++++++++++++++++++++++++++
# Change colors by groups: dose
# Use custom color palette
ggstripchart(df, "dose", "len",
  shape = "dose",
  color = "dose", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "mean_sd"
)


# Plot with multiple groups
# ++++++++++++++++++++
# Change shape and color by a second group : "supp"
ggstripchart(df, "dose", "len",
  shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800")
)

# Adjust point position
ggstripchart(df, "dose", "len",
  shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800"),
  position = position_dodge(0.8)
)

# You can also use position_jitterdodge()
# but fill aesthetic is required
ggstripchart(df, "dose", "len",
  shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800"),
  position = position_jitterdodge()
)

# Add boxplot
ggstripchart(df, "dose", "len",
  shape = "supp",
  color = "supp", palette = c("#00AFBB", "#E7B800"),
  add = "boxplot", add.params = list(color = "black")
)
```

`ggsummarytable` *GGPlot with Summary Stats Table Under the Plot*

Description

Create a ggplot with summary stats (n, median, mean, iqr) table under the plot. Read more: How to Create a Beautiful Plots in R with Summary Statistics Labels.

Usage

```
ggsummarytable(
  data,
  x,
  y,
  digits = 0,
  size = 3,
  color = "black",
  palette = NULL,
  facet.by = NULL,
  labeller = "label_value",
  position = "identity",
  angle = 0,
  ggtheme = theme_pubr(),
  ...
)

ggsummarystats(
  data,
  x,
  y,
  summaries = c("n", "median", "iqr"),
  ggfunc = ggboxplot,
  color = "black",
  fill = "white",
  palette = NULL,
  facet.by = NULL,
  free.panels = FALSE,
  labeller = "label_value",
  heights = c(0.8, 0.2),
  digits = 0,
  table.font.size = 3,
  ggtheme = theme_pubr(),
  ...
)

## S3 method for class 'ggsummarystats'
print(x, heights = c(0.8, 0.2), ...)

## S3 method for class 'ggsummarystats_list'
print(x, heights = c(0.8, 0.2), legend = NULL, ...)
```

Arguments

data	a data frame
x	a list of ggsummarystats.
y	character vector containing one or more variables to plot

digits	integer indicating the number of decimal places (round) to be used.
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
color	outline color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
labeller	Character vector. An alternative to the argument short.panel.labs. Possible values are one of "label_both" (panel labelled by both grouping variable names and levels) and "label_value" (panel labelled with only grouping levels).
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
angle	numeric value specifying the rotation angle (in degrees) of the summary table text. Default is 0.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments passed to the function ggpar(), facet() or ggarrange() when printing the plot.
summaries	summary stats to display in the table. Possible values are those returned by the function get_summary_stats(), including: "n", "min", "max", "median", "q1", "q2", "q3", "mad","mean", "sd", "se", "ci".
ggfunc	a ggpubr function, including: ggboxplot, ggviolin, ggdotplot, ggbarplot, ggline, etc. Can be any other ggplot function that accepts the following arguments data, x, color, fill, palette, ggtheme,facet.by.
fill	fill color.
free.panels	logical. If TRUE, create free plot panels when the argument facet.by is specified.
heights	a numeric vector of length 2, specifying the heights of the main and the summary table, respectively.
table.font.size	the summary table font size.
legend	character specifying legend position. Allowed values are one of c("top", "bottom", "left", "right", "none"). To remove the legend use legend = "none".

Functions

- ggsummarytable(): Create a table of summary stats
- ggsummarystats(): Create a ggplot with a summary stat table under the plot.

Examples

```
# Data preparation
# ::::::::::::::::::::::::::::::::::::::::::::::::
data("ToothGrowth")
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Add random QC column
set.seed(123)
qc <- rep(c("pass", "fail"), 30)
df$qc <- as.factor(sample(qc, 60))
# Inspect the data
head(df)


# Basic summary stats
# ::::::::::::::::::::::::::::::::::::::::::::::::
# Compute summary statistics
summary.stats <- df %>%
  group_by(dose) %>%
  get_summary_stats(type = "common")
summary.stats

# Visualize summary table
ggsummarytable(
  summary.stats,
  x = "dose", y = c("n", "median", "iqr"),
  ggtheme = theme_bw()
)


# Create plots with summary table under the plot
# ::::::::::::::::::::::::::::::::::::::::::::::::
# Basic plot
ggsummarystats(
  df,
  x = "dose", y = "len",
  ggfunc = ggboxplot, add = "jitter"
)

# Color by groups
ggsummarystats(
  df,
  x = "dose", y = "len",
  ggfunc = ggboxplot, add = "jitter",
  color = "dose", palette = "npg"
)

# Create a barplot
ggsummarystats(
  df,
  x = "dose", y = "len",
  ggfunc = ggbarplot, add = c("jitter", "median_iqr"),
```

```
  color = "dose", palette = "npg"
)

# Facet
# ::::::::::::::::::::::::::::::::::::::::::::::
# Specify free.panels = TRUE for free panels
ggsummarystats(
  df,
  x = "dose", y = "len",
  ggfunc = ggboxplot, add = "jitter",
  color = "dose", palette = "npg",
  facet.by = c("supp", "qc"),
  labeller = "label_both"
)
```

---

ggtext *Text*

Description

Add text to a plot.

Usage

```
ggtext(
  data,
  x = NULL,
  y = NULL,
  label = NULL,
  color = "black",
  palette = NULL,
  size = 11,
  face = "plain",
  family = "",
  show.legend = NA,
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  parse = FALSE,
  grouping.vars = NULL,
  position = "identity",
  ggp = NULL,
  ggtheme = theme_pubr(),
  ...
)
```

Arguments

data	a data frame
x, y	x and y variables for drawing.
label	the name of the column containing point labels. Can be also a character vector with length = nrow(data).
color	text font color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
size	text font size.
face	text font style. Allowed values are one of c("plain", "bold", "italic", "bold.italic").
family	character vector specifying font family.
show.legend	logical. Should text be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes.
label.select	can be of two formats: • a character vector specifying some labels to show. • a list containing one or the combination of the following components: – top.up and top.down: to display the labels of the top up/down points. For example, label.select = list(top.up = 10, top.down = 4). – criteria: to filter, for example, by x and y variables values, use this: label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')").
repel	a logical value, whether to use ggrepel to avoid overplotting text labels or not.
label.rectangle	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
grouping.vars	grouping variables to sort the data by, when the user wants to display the top n up/down labels.
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
ggp	a ggplot. If not NULL, points are added to an existing plot.
ggtheme	function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...
...	other arguments to be passed to ggpar.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"

See Also

ggpar

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)
df$name <- rownames(df)
head(df[, c("wt", "mpg", "cyl")], 3)

# Textual annotation
# ++++++++++++++++++
ggtext(df,
  x = "wt", y = "mpg",
  color = "cyl", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  label = "name", repel = TRUE
)

# Add rectangle around label
ggtext(df,
  x = "wt", y = "mpg",
  color = "cyl", palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  label = "name", repel = TRUE, label.rectangle = TRUE
)
```

---

ggtexttable *Draw a Textual Table*

Description

Draw a textual table.

- `ggtexttable()`: draw a textual table.
- `ttheme()`: customize table theme.

- `rownames_style()`, `colnames_style()`, `tbody_style()`: helper functions to customize the table row names, column names and body.
- `table_cell_font()`: access to a table cell for changing the text font (size and face).
- `table_cell_bg()`: access to a table cell for changing the background (fill, color, linewidth).
- `tab_cell_crossout()`: cross out a table cell.
- `tab_ncol()`, `tab_nrow()`: returns, respectively, the number of columns and rows in a ggtexttable.
- `tab_add_hline()`: Creates horizontal lines or separators at the top or the bottom side of a given specified row.
- `tab_add_vline()`: Creates vertical lines or separators at the right or the left side of a given specified column.
- `tab_add_border()`, `tbody_add_border()`, `thead_add_border()`: Add borders to table; tbody is for table body and thead is for table head.
- `tab_add_title()`,`tab_add_footnote()`: Add title, subtitle and footnote to a table.

Usage

```
ggtexttable(
  x,
  rows = rownames(x),
  cols = colnames(x),
  vp = NULL,
  theme = ttheme(),
  ...
)

ttheme(
  base_style = "default",
  base_size = 11,
  base_colour = "black",
  padding = unit(c(4, 4), "mm"),
  colnames.style = colnames_style(size = base_size),
  rownames.style = rownames_style(size = base_size),
  tbody.style = tbody_style(size = base_size)
)

colnames_style(
  color = "black",
  face = "bold",
  size = 12,
  fill = "grey80",
  linewidth = 1,
  linecolor = "white",
  parse = FALSE,
  ...
)
```

```
  rownames_style(
    color = "black",
    face = "italic",
    size = 12,
    fill = NA,
    linewidth = 1,
    linecolor = "white",
    parse = FALSE,
    ...
  )

  tbody_style(
    color = "black",
    face = "plain",
    size = 12,
    fill = c("grey95", "grey90"),
    linewidth = 1,
    linecolor = "white",
    parse = FALSE,
    ...
  )

  table_cell_font(tab, row, column, face = NULL, size = NULL, color = NULL)

  table_cell_bg(
    tab,
    row,
    column,
    fill = NULL,
    color = NULL,
    linewidth = NULL,
    alpha = NULL
  )

  tab_cell_crossout(
    tab,
    row,
    column,
    linetype = 1,
    linewidth = 1,
    linecolor = "black",
    reduce.size.by = 0
  )

  tab_ncol(tab)

  tab_nrow(tab)
```

```
tab_add_hline(
  tab,
  at.row = 2:tab_nrow(tab),
  row.side = c("bottom", "top"),
  from.column = 1,
  to.column = tab_ncol(tab),
  linetype = 1,
  linewidth = 1,
  linecolor = "black"
)

tab_add_vline(
  tab,
  at.column = 2:tab_ncol(tab),
  column.side = c("left", "right"),
  from.row = 1,
  to.row = tab_nrow(tab),
  linetype = 1,
  linewidth = 1,
  linecolor = "black"
)

tab_add_border(
  tab,
  from.row = 2,
  to.row = tab_nrow(tab),
  from.column = 1,
  to.column = tab_ncol(tab),
  linetype = 1,
  linewidth = 1,
  linecolor = "black"
)

tbody_add_border(
  tab,
  from.row = 2,
  to.row = tab_nrow(tab),
  from.column = 1,
  to.column = tab_ncol(tab),
  linetype = 1,
  linewidth = 1,
  linecolor = "black"
)

thead_add_border(
  tab,
  from.row = 1,
```

```
  to.row = 1,
  from.column = 1,
  to.column = tab_ncol(tab),
  linetype = 1,
  linewidth = 1,
  linecolor = "black"
)

tab_add_title(
  tab,
  text,
  face = NULL,
  size = NULL,
  color = NULL,
  family = NULL,
  padding = unit(1.5, "line"),
  just = "left",
  hjust = NULL,
  vjust = NULL
)

tab_add_footnote(
  tab,
  text,
  face = NULL,
  size = NULL,
  color = NULL,
  family = NULL,
  padding = unit(1.5, "line"),
  just = "right",
  hjust = NULL,
  vjust = NULL
)
```

Arguments

x	a data.frame or matrix.
rows	optional vector to specify row names
cols	optional vector to specify column names
vp	optional viewport
theme	a list, as returned by the function ttheme(), defining the parameters of the table theme. Allowed values include one of ttheme() and ttheme_clean().
...	extra parameters for text justification, e.g.: hjust and x. Default is "centre" for the body and header, and "right" for the row names. Left justification: hjust = 0, x = 0.1. Right justification: hjust = 1, x = 0.9.
base_style	character string the table style/theme. The available themes are illustrated in the ggtexttable-theme.pdf file. Allowed values include one of c("default",

	"blank", "classic","minimal", "light", "lBlack", "lBlue", "lRed", "lGreen", "lViolet", "lCyan","lOrange", "lBlackWhite", "lBlueWhite", "lRedWhite", "lGreenWhite","lVioletWhite", "lCyanWhite", "lOrangeWhite", "mBlack", "mBlue", "mRed","mGreen", "mViolet", "mCyan", "mOrange", "mBlackWhite", "mBlueWhite","mRedWhite", "mGreenWhite", "mVioletWhite", "mCyanWhite", "mOrangeWhite"). Note that, l = "light"; m = "medium".
base_size	default font size
base_colour	default font colour
padding	length-2 unit vector specifying the horizontal and vertical padding of text within each cell
colnames.style	a list, as returned by the function colnames_style(), defining the style of the table column names. Considered only when base_size = "default".
rownames.style	a list, as returned by the function rownames_style(), defining the style of the table row names. Considered only when base_size = "default".
tbody.style	a list, as returned by the function tbody_style(), defining the style of the table body. Considered only when base_size = "default".
color, face, size	text font color, face and size, respectively. Allowed values for face include c("plain", "bold", "italic", "bold.italic").
fill	background color.
linewidth, linecolor	line width and color, respectively.
parse	logical, default behaviour for parsing text as plotmath
tab	an object from ggtexttable or from gridExtra::tableGrob().
row, column	an integer specifying the row and the column numbers for the cell of interest.
alpha	numeric value specifying fill color transparency. Value should be in [0, 1], where 0 is full transparency and 1 is no transparency.
linetype	line type
reduce.size.by	Numeric value in [0, 1] to reduce the size by.
at.row	a numeric vector of row indexes; for example at.row = c(1, 2).
row.side	row side to which the horizontal line should be added. Can be one of c("bottom", "top").
from.column	integer indicating the column from which to start drawing the horizontal line.
to.column	integer indicating the column to which the horizontal line should end.
at.column	a numeric vector of column indexes; for example at.column = c(1, 2).
column.side	column side to which the vertical line should be added. Can be one of c("left", "right").
from.row	integer indicating the row from which to start drawing the horizontal line.
to.row	integer indicating the row to which the vertical line should end.
text	text to be added as title or footnote.
family	font family

just	The justification of the text relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left (bottom) alignment and 1 means right (top) alignment.
hjust	A numeric vector specifying horizontal justification. If specified, overrides the just setting.
vjust	A numeric vector specifying vertical justification. If specified, overrides the just setting.

Value

an object of class ggplot.

Examples

```
# data
df <- head(iris)

# Default table
# Remove row names using rows = NULL
ggtexttable(df, rows = NULL)

# Text justification for individual cells/rows/columns (#335)
# First column is left justified i.e., hjust = 0 , x = 0.1
# Remaining columns are right justified i.e., hjust = 1 , x = 0.9
table_theme <- ttheme(
  tbody.style = tbody_style(
    hjust = as.vector(matrix(c(0, 1, 1, 1, 1), ncol = 5, nrow = nrow(df), byrow = TRUE)),
    x = as.vector(matrix(c(.1, .9, .9, .9, .9), ncol = 5, nrow = nrow(df), byrow = TRUE))
  )
)
ggtexttable(df, rows = NULL, theme = table_theme)

# Blank theme
ggtexttable(df, rows = NULL, theme = ttheme("blank"))

# light theme
ggtexttable(df, rows = NULL, theme = ttheme("light"))

# Column names border only
ggtexttable(df, rows = NULL, theme = ttheme("blank")) %>%
  tab_add_hline(at.row = 1:2, row.side = "top", linewidth = 2)

# classic theme
ggtexttable(df, rows = NULL, theme = ttheme("classic"))

# minimal theme
ggtexttable(df, rows = NULL, theme = ttheme("minimal"))

# Medium blue (mBlue) theme
```

```
ggtexttable(df, rows = NULL, theme = ttheme("mBlue"))


# Customize the table as you want
ggtexttable(df,
  rows = NULL,
  theme = ttheme(
    colnames.style = colnames_style(color = "white", fill = "#8cc257"),
    tbody.style = tbody_style(color = "black", fill = c("#e8f3de", "#d3e8bb"))
  )
)

# Use RColorBrewer palette
# Provide as many fill color as there are rows in the table body, here nrow = 6
ggtexttable(df,
  theme = ttheme(
    colnames.style = colnames_style(fill = "white"),
    tbody.style = tbody_style(fill = get_palette("RdBu", 6))
  )
)

# Text justification
# :::::::::::::::::::::::::::::::::::::::::::::
# Default is "centre" for the body and header, and "right" for the row names.
# Left justification: hjust=0, x=0.1
# Right justification: hjust=1, x=0.9
tbody.style <- tbody_style(
  color = "black",
  fill = c("#e8f3de", "#d3e8bb"), hjust = 1, x = 0.9
)
ggtexttable(head(iris),
  rows = NULL,
  theme = ttheme(
    colnames.style = colnames_style(color = "white", fill = "#8cc257"),
    tbody.style = tbody.style
  )
)

# Access and modify the font and
# the background of table cells
# :::::::::::::::::::::::::::::::::::::::::::::
tab <- ggtexttable(head(iris),
  rows = NULL,
  theme = ttheme("classic")
)
tab <- table_cell_font(tab,
  row = 3, column = 2,
  face = "bold"
)
tab <- table_cell_bg(tab,
  row = 4, column = 3, linewidth = 5,
  fill = "darkolivegreen1", color = "darkolivegreen4"
)
```

```
tab

# Change table cells background and font for column 3,
# Spaning from row 2 to the last row in the data
tab <- ggtexttable(df, rows = NULL, theme = ttheme("classic"))
tab %>%
  table_cell_bg(row = 2:tab_nrow(tab), column = 3, fill = "darkblue") %>%
  table_cell_font(row = 2:tab_nrow(tab), column = 3, face = "italic", color = "white")

# Add separators and borders
# :::::::::::::::::::::::::::::::::::::::::::::::::
# Table with blank theme
tab <- ggtexttable(df, theme = ttheme("blank"), rows = NULL)
# Add horizontal and vertical lines
tab %>%
  tab_add_hline(at.row = c(1, 2), row.side = "top", linewidth = 3, linetype = 1) %>%
  tab_add_hline(at.row = c(7), row.side = "bottom", linewidth = 3, linetype = 1) %>%
 tab_add_vline(at.column = 2:tab_ncol(tab), column.side = "left", from.row = 2, linetype = 2)

# Add borders to table body and header
# Cross out some cells
tab %>%
  tbody_add_border() %>%
  thead_add_border() %>%
  tab_cell_crossout(
    row = c(2, 4), column = 3, linecolor = "red",
    reduce.size.by = 0.6
  )

# Add titles andd footnote
# :::::::::::::::::::::::::::::::::::::::::::::::::
# Add titles and footnote
# Wrap subtitle into multiple lines using strwrap()
main.title <- "Edgar Anderson's Iris Data"
subtitle <- paste0(
  "This famous (Fisher's or Anderson's) iris data set gives the measurements",
  " in centimeters of the variables sepal length and width and petal length and width,",
  " respectively, for 50 flowers from each of 3 species of iris.",
  " The species are Iris setosa, versicolor, and virginica."
) %>%
  strwrap(width = 80) %>%
  paste(collapse = "\n")

tab <- ggtexttable(head(iris), theme = ttheme("light"))
tab %>%
  tab_add_title(text = subtitle, face = "plain", size = 10) %>%
  tab_add_title(text = main.title, face = "bold", padding = unit(0.1, "line")) %>%
  tab_add_footnote(text = "*Table created using ggpubr", size = 10, face = "italic")


# Combine density plot and summary table
# :::::::::::::::::::::::::::::::::::
# Density plot of "Sepal.Length"
```

```
density.p <- ggdensity(iris,
  x = "Sepal.Length",
  fill = "Species", palette = "jco"
)

# Draw the summary table of Sepal.Length
# Descriptive statistics by groups
stable <- desc_statby(iris,
  measure.var = "Sepal.Length",
  grps = "Species"
)
stable <- stable[, c("Species", "length", "mean", "sd")]
stable.p <- ggtexttable(stable,
  rows = NULL,
  theme = ttheme("mOrange")
)

# Arrange the plots on the same page
ggarrange(density.p, stable.p,
  ncol = 1, nrow = 2,
  heights = c(1, 0.5)
)
```

---

ggviolin *Violin plot*

Description

Create a violin plot with error bars. Violin plots are similar to box plots, except that they also show the kernel probability density of the data at different values.

Usage

```
ggviolin(
  data,
  x,
  y,
  combine = FALSE,
  merge = FALSE,
  color = "black",
  fill = "white",
  palette = NULL,
  alpha = 1,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  facet.by = NULL,
  panel.labs = NULL,
```

```
  short.panel.labs = TRUE,
  linetype = "solid",
  trim = FALSE,
  drop = TRUE,
  size = NULL,
  linewidth = NULL,
  width = 1,
  quantiles = NULL,
  quantile.linetype = NULL,
  quantile.type = NULL,
  quantile.alpha = NULL,
  quantile.colour = NULL,
  quantile.color = NULL,
  quantile.linewidth = NULL,
  quantile.size = NULL,
  draw_quantiles = NULL,
  select = NULL,
  remove = NULL,
  order = NULL,
  add = "mean_se",
  add.params = list(),
  error.plot = "pointrange",
  label = NULL,
  font.label = list(size = 11, color = "black"),
  label.select = NULL,
  repel = FALSE,
  label.rectangle = FALSE,
  position = position_dodge(0.8),
  ggtheme = theme_pubr(),
  show.n = FALSE,
  ...
)
```

Arguments

data	a data frame
x	character string containing the name of x variable.
y	character vector containing one or more variables to plot
combine	logical value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of y variables.
merge	logical or character value. Default is FALSE. Used only when y is a vector containing multiple variables to plot. If TRUE, merge multiple y variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If merge = "flip", then y variables are used as x tick labels and the x variable is used as grouping variable.
color	outline color.

fill	fill color.
palette	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
alpha	color transparency. Values should be between 0 and 1.
title	plot main title.
xlab	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
ylab	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
facet.by	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
panel.labs	a list of one or two character vectors to modify facet panel labels. For example, panel.labs = list(sex = c("Male", "Female")) specifies the labels for the "sex" variable. For two grouping variables, you can use for example panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2")).
short.panel.labs	logical value. Default is TRUE. If TRUE, create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.
linetype	line types.
trim	If TRUE (default), trim the tails of the violins to the range of the data. If FALSE, don't trim the tails.
drop	logical, passed to geom_violin(), controlling whether grouped sub-samples with fewer than two data points (for which no density can be computed) are dropped from the plot and the dodge position. By default ggviolin() keeps grouped violins aligned with their other layers (e.g. added boxplots or dot plots) automatically: when you do not set drop or position and a grouped sub-group has a single data point, the empty dodge lane is reserved (equivalent to drop = FALSE together with position = position_dodge(0.8, preserve = "single")) so the violins stay aligned. Balanced, ungrouped, and faceted plots are unaffected. Setting drop or position explicitly turns off this automatic handling and uses exactly what you supply (#381).
size	Numeric value (e.g.: size = 1). change the size of points and outlines.
linewidth	constant value specifying the line width.
width	violin width.
quantiles	numeric vector of quantiles to draw on the violin.
quantile.linetype	linetype for quantile lines; set to draw quantiles with ggplot2 >= 4.0.0.
quantile.type	quantile algorithm passed to ggplot2.
quantile.alpha, quantile.size	quantile.colour, quantile.color, quantile.linewidth, aesthetics for quantile lines.

draw_quantiles **[Deprecated]** Previous specification of drawing quantiles.

select character vector specifying which items to display.

remove character vector specifying which items to remove from the plot.

order character vector specifying the order of items.

add character vector for adding another plot element (e.g.: dot plot or error bars). Allowed values are one or the combination of: "none", "dotplot", "jitter", "boxplot", "point", "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_hilow", "median_q1q3", "median_mad", "median_range"; see ?desc_statby for more details.

add.params parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").

error.plot plot type used to visualize error. Allowed values are one of c("pointrange", "linerange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ...

label the name of the column containing point labels. Can be also a character vector with length = nrow(data).

font.label a list which can contain the combination of the following elements: the size (e.g.: 14), the style (e.g.: "plain", "bold", "italic", "bold.italic") and the color (e.g.: "red") of labels. For example font.label = list(size = 14, face = "bold", color ="red"). To specify only the size and the style, use font.label = list(size = 14, face = "plain").

label.select can be of two formats:

- a character vector specifying some labels to show.
- a list containing one or the combination of the following components:
  - top.up and top.down: to display the labels of the top up/down points. For example, label.select = list(top.up = 10, top.down = 4).
  - criteria: to filter, for example, by x and y variables values, use this: label.select = list(criteria = "`y` > 2 & `y` < 5 & `x` %in% c('A', 'B')").

repel a logical value, whether to use ggrepel to avoid overplotting text labels or not.

label.rectangle

logical value. If TRUE, add rectangle underneath the text, making it easier to read.

position position adjustment, either as a string, or the result of a call to a position adjustment function (e.g. position_dodge(0.8)). Used to control the spacing between grouped elements.

ggtheme function, ggplot2 theme name. Default value is theme_pubr(). Set ggtheme = NULL to skip applying a ggpubr theme, so the plot keeps ggplot2 default theme or the theme set globally via theme_set(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ...

show.n	logical. If TRUE, displays the number of observations ("n = <count>") at the top of each group. Off by default. When the groups are dodged (a color/fill grouping with a dodging position), one count is shown per group; otherwise a single count is shown per x-axis tick. Counts respect select/remove and are computed per facet.
...	other arguments to be passed to geom_violin, ggpar and facet.

Details

The plot can be easily customized using the function ggpar(). Read ?ggpar for changing:

- main title and axis labels: main, xlab, ylab
- axis limits: xlim, ylim (e.g.: ylim = c(0, 30))
- axis scales: xscale, yscale (e.g.: yscale = "log2")
- color palettes: palette = "Dark2" or palette = c("gray", "blue", "red")
- legend title, labels and position: legend = "right"
- plot orientation : orientation = c("vertical", "horizontal", "reverse")

See Also

ggpar

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot
# ++++++++++++++++++++++++++++
ggviolin(df, x = "dose", y = "len")
# Change the plot orientation: horizontal
ggviolin(df, "dose", "len", orientation = "horiz")

# Add summary statistics
# ++++++++++++++++++++++++++
# Draw quantiles
ggviolin(df, "dose", "len",
  add = "none",
  quantiles = 0.5, quantile.linetype = "dashed"
)

# Add box plot
ggviolin(df,
  x = "dose", y = "len",
  add = "boxplot"
)

ggviolin(df,
  x = "dose", y = "len",
```

```
  add = "dotplot"
)

# Add jitter points and
# change point shape by groups ("dose")
ggviolin(df,
  x = "dose", y = "len",
  add = "jitter", shape = "dose"
)


# Add mean_sd + jittered points
ggviolin(df,
  x = "dose", y = "len",
  add = c("jitter", "mean_sd")
)

# Change error.plot to "crossbar"
ggviolin(df,
  x = "dose", y = "len",
  add = "mean_sd", error.plot = "crossbar"
)


# Change colors
# ++++++++++++++++++++++++++
# Change outline and fill colors
ggviolin(df, "dose", "len",
  color = "black", fill = "gray"
)

# Change outline colors by groups: dose
# Use custom color palette and add boxplot
ggviolin(df, "dose", "len",
  color = "dose",
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "boxplot"
)

# Change fill color by groups: dose
# add boxplot with white fill color
ggviolin(df, "dose", "len",
  fill = "dose",
  palette = c("#00AFBB", "#E7B800", "#FC4E07"),
  add = "boxplot", add.params = list(fill = "white")
)


# Plot with multiple groups
# ++++++++++++++++++++
# fill or color box plot by a second group : "supp"
ggviolin(df, "dose", "len",
  color = "supp",
```

```
  palette = c("#00AFBB", "#E7B800"), add = "boxplot"
)
```

`gradient_color` *Set Gradient Color*

Description

Change gradient color.

- `gradient_color()`: Change gradient color.
- `gradient_fill()`: Change gradient fill.

Usage

```
gradient_color(palette)

gradient_fill(palette)
```

Arguments

`palette`	the color palette to be used for coloring or filling by groups. Allowed values include "grey" for grey color palettes; brewer palettes e.g. "RdBu", "Blues", ...; or custom color palette e.g. c("blue", "red"); and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty". Can be also a numeric vector; in this case a basic color palette is created using the function palette.

See Also

set_palette.

Examples

```
df <- mtcars
p <- ggscatter(df,
  x = "wt", y = "mpg",
  color = "mpg"
)

# Change gradient color
# Use one custom color
p + gradient_color("red")

# Two colors
p + gradient_color(c("blue", "red"))

# Three colors
```

```
p + gradient_color(c("blue", "white", "red"))

# Use RColorBrewer palette
p + gradient_color("RdYlBu")

# Use ggsci color palette
p + gradient_color("npg")
```

`grids` *Add Grids to a ggplot*

Description

Add grids to ggplot.

Usage

```
grids(axis = c("xy", "x", "y"), color = "grey92", size = NULL, linetype = NULL)
```

Arguments

`axis`	axis for which grid should be added. Allowed values include `c("xy", "x", "y")`.
`color`	grid line color.
`size`	numeric value specifying grid line size.
`linetype`	line type. An integer (0:8), a name (blank, solid, dashed, dotted, dotdash, longdash, twodash). Sess `show_line_types`.

Examples

```
# Load data
data("ToothGrowth")

# Basic plot
p <- ggboxplot(ToothGrowth, x = "dose", y = "len")
p

# Add border
p + grids(linetype = "dashed")
```

list_p_format_styles *List Available P-Value Format Styles*

Description

Returns a data frame describing all available p-value formatting styles.

Usage

```
list_p_format_styles()
```

Value

A data frame with columns: style, digits, leading.zero, min.threshold, use.scientific, description.

Examples

```
list_p_format_styles()
```

npc_to_data_coord *Convert NPC to Data Coordinates*

Description

Convert NPC (Normalized Parent Coordinates) into data coordinates.

Usage

```
npc_to_data_coord(npc, data.ranges)
```

Arguments

npc	a numeric vector. Each value should be in [0-1]
data.ranges	a numeric vector of length 2 containing the data ranges (minimum and the maximum)

Value

a numeric vector representing data coordinates.

See Also

as_npc, get_coord.

Examples

```
npc_to_data_coord(npc = c(0.2, 0.95), data.ranges = c(1, 20))
as_npc(c("top", "right")) %>%
  npc_to_data_coord(data.ranges = c(1, 20))
```

---

rotate *Rotate a ggplot Horizontally*

Description

Rotate a ggplot to create horizontal plots. Wrapper around coord_flip.

Usage

```
rotate(...)
```

Arguments

...	other arguments to pass to coord_flip.

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot
p <- ggboxplot(df,
  x = "dose", y = "len",
  color = "dose", palette = "jco"
)
p
# Create horizontal plots
p + rotate()
```

---

rotate_axis_text *Rotate Axes Text*

Description

Rotate the x-axis text (tick mark labels).

- rotate_x_text(): Rotate x axis text.
- rotate_y_text(): Rotate y axis text.

Usage

```
rotate_x_text(angle = 90, hjust = NULL, vjust = NULL, ...)

rotate_y_text(angle = 90, hjust = NULL, vjust = NULL, ...)
```

Arguments

angle	numeric value specifying the rotation angle. Default is 90 for vertical x-axis text.
hjust	horizontal justification (in [0, 1]).
vjust	vertical justification (in [0, 1]).
...	other arguments to pass to the function element_text().

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot
p <- ggboxplot(df, x = "dose", y = "len")
p
# Vertical x axis text
p + rotate_x_text()
# Set rotation angle to 45
p + rotate_x_text(45)
p + rotate_y_text(45)
```

---

rremove *Remove a ggplot Component*

Description

Remove a specific component from a ggplot.

Usage

```
rremove(object)
```

Arguments

object	character string specifying the plot components. Allowed values include: • "grid" for both x and y grids • "x.grid" for x axis grids • "y.grid" for y axis grids

- "axis" for both x and y axes
- "x.axis" for x axis
- "y.axis" for y axis
- "xlab", or "x.title" for x axis label
- "ylab", or "y.title" for y axis label
- "xylab", "xy.title" or "axis.title" for both x and y axis labels
- "x.text" for x axis texts (x axis tick labels)
- "y.text" for y axis texts (y axis tick labels)
- "xy.text" or "axis.text" for both x and y axis texts
- "ticks" for both x and y ticks
- "x.ticks" for x ticks
- "y.ticks" for y ticks
- "legend.title" for the legend title
- "legend" for the legend

Examples

```
# Load data
data("ToothGrowth")

# Basic plot
p <- ggboxplot(ToothGrowth,
  x = "dose", y = "len",
  ggtheme = theme_gray()
)
p

# Remove all grids
p + rremove("grid")

# Remove only x grids
p + rremove("x.grid")
```

---

set_palette *Set Color Palette*

Description

- change_palette(), set_palette(): Change both color and fill palettes.
- color_palette(): change color palette only.
- fill_palette(): change fill palette only.

Usage

```
set_palette(p, palette)

change_palette(p, palette)

color_palette(palette = NULL, ...)

fill_palette(palette = NULL, ...)
```

Arguments

p	a ggplot
palette	Color palette. Allowed values include: • **Grey color palettes**: "grey" or "gray"; • **RColorBrewer palettes**, see brewer.pal and details section. Examples of palette names include: "RdBu", "Blues", "Dark2", "Set2", ...; • **Custom color palettes**. For example, palette = c("#00AFBB", "#E7B800", "#FC4E07"); • **ggsci scientific journal palettes**, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
...	other arguments passed to ggplot2 scale_color_xxx() and scale_fill_xxx() functions.

See Also

get_palette.

Examples

```
# Load data
data("ToothGrowth")
df <- ToothGrowth

# Basic plot
p <- ggboxplot(df,
  x = "dose", y = "len",
  color = "dose"
)
p

# Change the color palette
set_palette(p, "jco")
```

show_line_types *Line types available in R*

Description

Show line types available in R.

Usage

```
show_line_types()
```

Value

a ggplot.

See Also

ggpar and ggline.

Examples

```
show_line_types() +
  theme_minimal()
```

show_point_shapes *Point shapes available in R*

Description

Show point shapes available in R.

Usage

```
show_point_shapes()
```

Value

a ggplot.

See Also

ggpar and ggline.

Examples

```
show_point_shapes() +
  theme_minimal()
```

stat_anova_test *Add Anova Test P-values to a GGPlot*

Description

Adds automatically one-way and two-way ANOVA test p-values to a ggplot, such as box plots, dot plots and stripcharts.

Usage

```
stat_anova_test(
  mapping = NULL,
  data = NULL,
  method = c("one_way", "one_way_repeated", "two_way", "two_way_repeated",
    "two_way_mixed"),
  wid = NULL,
  group.by = NULL,
  type = NULL,
  effect.size = "ges",
  error = NULL,
  correction = c("auto", "GG", "HF", "none"),
  label = "{method}, p = {p.format}",
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  step.increase = 0.1,
  p.adjust.method = "holm",
  significance = list(),
  signif.cutoffs = NULL,
  signif.symbols = NULL,
  ns.symbol = "ns",
  use.four.stars = FALSE,
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  geom = "text",
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  parse = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
method	ANOVA test methods. Possible values are one of c("one_way", "one_way_repeated", "two_way", "two_way_repeated","two_way_mixed").
wid	(factor) column name containing individuals/subjects identifier. Should be unique per individual. Required only for repeated measure tests ("one_way_repeated", "two_way_repeated", "friedman_test", etc).
group.by	(optional) character vector specifying the grouping variable; it should be used only for grouped plots. Possible values are : • "x.var": Group by the x-axis variable and perform the test between legend groups. In other words, the p-value is compute between legend groups at each x position • "legend.var": Group by the legend variable and perform the test between x-axis groups. In other words, the test is performed between the x-groups for each legend level.
type	the type of sums of squares for ANOVA. Allowed values are either 1, 2 or 3. type = 2 is the default because this will yield identical ANOVA results as type = 1 when data are balanced but type = 2 will additionally yield various assumption tests where appropriate. When the data are unbalanced the type = 3 is used by popular commercial software including SPSS.
effect.size	the effect size to compute and to show in the ANOVA results. Allowed values can be either "ges" (generalized eta squared) or "pes" (partial eta squared) or both. Default is "ges".
error	(optional) for a linear model, an lm model object from which the overall error sum of squares and degrees of freedom are to be calculated. Read more in Anova() documentation.
correction	character. Used only in repeated measures ANOVA test to specify which correction of the degrees of freedom should be reported for the within-subject factors. Possible values are: • "GG": applies Greenhouse-Geisser correction to all within-subjects factors even if the assumption of sphericity is met (i.e., Mauchly's test is not significant, $p > 0.05$). • "HF": applies Huynh-Feldt correction to all within-subjects factors even if the assumption of sphericity is met,

- "none": returns the ANOVA table without any correction and
- "auto": apply automatically GG correction to only within-subjects factors violating the sphericity assumption (i.e., Mauchly's test p-value is significant, p <= 0.05).

`label` character string specifying label. Can be:

- the column containing the label (e.g.: `label = "p"` or `label = "p.adj"`), where p is the p-value. Other possible values are `"p.signif"`, `"p.adj.signif"`, `"p.format"`, `"p.format.signif"`, `"p.adj.format"`.
- an expression that can be formatted by the `glue()` package. For example, when specifying `label = "Anova, p = \{p\}"`, the expression {p} will be replaced by its value.
- a combination of plotmath expressions and glue expressions. You may want some of the statistical parameter in italic; for example:`label = "Anova, italic(p) = {p}"`.
- a constant: `label = "as_italic"`: display statistical parameters in italic; `label = "as_detailed"`: detailed plain text; `label = "as_detailed_expression"` or `label = "as_detailed_italic"`: detailed plotmath expression. Statistical parameters will be displayed in italic.

.

`label.x.npc, label.y.npc`

can be `numeric` or `character` vector of the same length as the number of groups and/or panels. If too short they will be recycled.

- If `numeric`, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".
- If `character`, allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

`label.x, label.y`

`numeric` Coordinates (in data units) to be used for absolute positioning of the label. If too short they will be recycled.

`step.increase` numeric value in with the increase in fraction of total height for every additional comparison to minimize overlap. The step value can be negative to reverse the order of groups.

`p.adjust.method`

method for adjusting p values (see `p.adjust`). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

`significance` a list of arguments specifying the significance cutpoints and symbols. For example, `significance <- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns"))`.

In other words, we use the following convention for symbols indicating statistical significance:

- ns: p > 0.05

- *: p <= 0.05
- **: p <= 0.01
- ***: p <= 0.001
- ****: p <= 0.0001

Note: `significance` is kept for backward compatibility. For functions that also expose `signif.cutoffs` and related parameters, `significance` takes precedence when provided.

`signif.cutoffs` numeric vector of p-value cutoffs in descending order for assigning significance symbols. For example, `c(0.10, 0.05, 0.01)` means p < 0.10 gets "*", p < 0.05 gets "**", p < 0.01 gets "***". If `use.four.stars = TRUE`, can include a fourth level. Default is NULL, which uses the package defaults.

`signif.symbols` character vector of symbols corresponding to `signif.cutoffs`. If NULL, auto-generated as "*", "**", "***" (and "****" if `use.four.stars = TRUE`).

`ns.symbol` character string for non-significant results. Default is "ns". Use "" (empty string) to show nothing.

`use.four.stars` logical. If TRUE, allows four stars (****) for the most significant level. Default is FALSE.

`p.format.style` character string specifying the p-value formatting style. One of: `"default"` (backward compatible, uses scientific notation), `"apa"` (APA style, no leading zero), `"nejm"` (NEJM style), `"lancet"` (Lancet style), `"ama"` (AMA style), `"graphpad"` (GraphPad style), or `"scientific"` (scientific notation for GWAS). See `list_p_format_styles` for details.

`p.digits` integer specifying the number of decimal places for p-values. If provided, overrides the style default.

`p.leading.zero` logical indicating whether to include leading zero before decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

`p.min.threshold`

numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold". If provided, overrides the style default.

`p.decimal.mark` character string to use as the decimal mark. If NULL, uses `getOption("OutDec")`.

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE` (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to `TRUE` to inherit aesthetics from the parent ggplot layer.

`parse` If TRUE, the labels will be parsed into expressions and displayed as described in `?plotmath`.

`...` other arguments to pass to `geom_text`, such as:

- `hjust`: horizontal justification of the text. Move the text left or right and
- `vjust`: vertical justification of the text. Move the text up or down.

Computed variables

- DFn: Degrees of Freedom in the numerator (i.e. DF effect).
- DFd: Degrees of Freedom in the denominator (i.e., DF error).
- ges: Generalized Eta-Squared measure of effect size. Computed only when the option `effect.size = "ges"`.
- pes: Partial Eta-Squared measure of effect size. Computed only when the option `effect.size = "pes"`.
- F: F-value.
- p: p-value.
- p.adj: Adjusted p-values.
- p.signif: P-value significance.
- p.adj.signif: Adjusted p-value significance.
- p.format: Formatted p-value.
- p.format.signif: Formatted p-value with significance symbols.
- p.adj.format: Formatted adjusted p-value.
- n: number of samples.

Examples

```
# Data preparation
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Transform `dose` into factor variable
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Add individuals id
df$id <- rep(1:10, 6)
# Add a random grouping variable
set.seed(123)
df$group <- sample(factor(rep(c("grp1", "grp2", "grp3"), 20)))
df$len <- ifelse(df$group == "grp2", df$len + 2, df$len)
df$len <- ifelse(df$group == "grp3", df$len + 7, df$len)
head(df, 3)


# Basic boxplot
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a basic boxplot
# Add 5% and 10% space to the plot bottom and the top, respectively
bxp <- ggboxplot(df, x = "dose", y = "len") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))

# Add the p-value to the boxplot
bxp + stat_anova_test()

## Not run:
# Change the label position
# Using coordinates in data units
bxp + stat_anova_test(label.x = "1", label.y = 10, hjust = 0)

## End(Not run)

# Format the p-value differently
custom_p_format <- function(p) {
  rstatix::p_format(p, accuracy = 0.0001, digits = 3, leading.zero = FALSE)
}
bxp + stat_anova_test(
  label = "Anova, italic(p) = {custom_p_format(p)}{p.signif}"
)

# Show a detailed label in italic
bxp + stat_anova_test(label = "as_detailed_italic")


# Faceted plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a ggplot facet
bxp <- ggboxplot(df, x = "dose", y = "len", facet.by = "supp") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))
# Add p-values
bxp + stat_anova_test()
```

```
# Grouped plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bxp2 <- ggboxplot(df, x = "group", y = "len", color = "dose", palette = "npg")

# For each x-position, computes tests between legend groups
bxp2 + stat_anova_test(aes(group = dose), label = "p = {p.format}{p.signif}")

#  For each legend group, computes tests between x variable groups
bxp2 + stat_anova_test(aes(group = dose, color = dose), group.by = "legend.var")

## Not run:
# Two-way ANOVA: Independent measures
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Visualization: box plots with p-values
# Two-way interaction p-values between x and legend (group) variables
bxp3 <- ggboxplot(
  df,
  x = "supp", y = "len",
  color = "dose", palette = "jco"
)
bxp3 + stat_anova_test(aes(group = dose), method = "two_way")

# One-way repeatead measures ANOVA
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
df$id <- as.factor(c(rep(1:10, 3), rep(11:20, 3)))
ggboxplot(df, x = "dose", y = "len") +
  stat_anova_test(method = "one_way_repeated", wid = "id")

# Two-way repeatead measures ANOVA
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
df$id <- as.factor(rep(1:10, 6))
ggboxplot(df, x = "dose", y = "len", color = "supp", palette = "jco") +
  stat_anova_test(aes(group = supp), method = "two_way_repeated", wid = "id")

# Grouped one-way repeated measures ANOVA
ggboxplot(df, x = "dose", y = "len", color = "supp", palette = "jco") +
  stat_anova_test(aes(group = supp, color = supp),
    method = "one_way_repeated", wid = "id", group.by = "legend.var"
  )

## End(Not run)
```

---

stat_bracket *Add Brackets with Labels to a GGPlot*

Description

add brackets with label annotation to a ggplot. Helpers for adding p-value or significance levels to a plot.

Usage

```
stat_bracket(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  label = NULL,
  type = c("text", "expression"),
  y.position = NULL,
  xmin = NULL,
  xmax = NULL,
  step.increase = 0,
  step.group.by = NULL,
  tip.length = 0.03,
  tip.length.ref = c("data", "axis"),
  bracket.nudge.y = 0,
  bracket.shorten = 0,
  size = 0.3,
  linewidth = size,
  label.size = 3.88,
  family = "",
  vjust = 0,
  ...
)

geom_bracket(
  mapping = NULL,
  data = NULL,
  stat = "bracket",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  label = NULL,
  type = c("text", "expression"),
  y.position = NULL,
  xmin = NULL,
  xmax = NULL,
  step.increase = 0,
  step.group.by = NULL,
  tip.length = 0.03,
  tip.length.ref = c("data", "axis"),
  bracket.nudge.y = 0,
  bracket.shorten = 0,
  size = 0.3,
  linewidth = size,
```

```
  label.size = 3.88,
  family = "",
  vjust = 0,
  coord.flip = FALSE,
  orientation = c("horizontal", "vertical"),
  ymin = NULL,
  ymax = NULL,
  x.position = NULL,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

`label` character vector with alternative label, if not null test is ignored

`type` the label type. Can be one of "text" and "expression" (for parsing plotmath expression).

`y.position` numeric vector with the y positions of the brackets

`xmin` numeric vector with the positions of the left sides of the brackets

`xmax` numeric vector with the positions of the right sides of the brackets

`step.increase` numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.

`step.group.by` a variable name for grouping brackets before adding step.increase. Useful to group bracket by facet panel.

`tip.length` numeric vector with the fraction that the bracket tips go down to indicate the precise column. Interpreted relative to the reference set by `tip.length.ref`. Default is 0.03. Can be of the same length as the number of brackets to adjust each tip specifically, e.g. `tip.length = c(0.01, 0.03)`. If too short it is recycled.

`tip.length.ref` character string specifying what `tip.length` is a fraction of. Allowed values are:

- `"data"` (default): fraction of the trained data range. Tips scale with the data, so plots with different data ranges get different absolute tip lengths (historical behavior; keeps existing plots unchanged).
- `"axis"`: fraction of the y-axis range (the limits set via `ylim`/`scale_y_*`). This renders at the same physical fraction across plots and therefore gives visually constant tip lengths regardless of the data - useful to keep tips consistent across facets or across separate plots with different scales (#362).

`bracket.nudge.y` Vertical adjustment to nudge brackets by. Useful to move up or move down the bracket. If positive value, brackets will be moved up; if negative value, brackets are moved down.

`bracket.shorten` a small numeric value in [0-1] for shortening the with of bracket.

`size` change the width of the lines of the bracket. Deprecated, use `linewidth` instead.

`linewidth` change the width of the lines of the bracket

`label.size` change the size of the label text

`family` change the font used for the text

`vjust` move the text up or down relative to the bracket

`...` other arguments passed on to `layer()`). These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.

- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`coord.flip` logical. If `TRUE`, flip x and y coordinates so that horizontal becomes vertical, and vertical, horizontal. When adding the p-values to a horizontal ggplot (generated using `coord_flip()`), you need to specify the option `coord.flip = TRUE`.

`orientation` the bracket orientation. Either `"horizontal"` (the default, a bracket spanning the x axis with the bar at `y.position`) or `"vertical"` (a bracket spanning the y axis with the bar at `x.position`, tips pointing left and the label rotated to the side). Vertical brackets are useful to annotate plots where the comparison runs along the y axis, e.g. Kaplan-Meier curves. They are designed for a continuous x axis; specify the span with `ymin`/`ymax` and the position with `x.position`. Cannot be combined with `coord.flip = TRUE`.

`ymin, ymax` numeric vectors with the positions, along the y axis, of the two ends of each vertical bracket. Used only when `orientation = "vertical"` (the vertical-bracket analogue of `xmin`/`xmax`).

`x.position` numeric vector with the x positions of the vertical brackets. Used only when `orientation = "vertical"` (the vertical-bracket analogue of `y.position`).

Examples

```
df <- ToothGrowth
df$dose <- factor(df$dose)

# Add bracket with labels
ggboxplot(df, x = "dose", y = "len") +
  geom_bracket(
    xmin = "0.5", xmax = "1", y.position = 30,
    label = "t-test, p < 0.05"
  )

# Customize bracket tip.length tip.length
ggboxplot(df, x = "dose", y = "len") +
  geom_bracket(
    xmin = "0.5", xmax = "1", y.position = 30,
    label = "t-test, p < 0.05", tip.length = c(0.2, 0.02)
  )

# Using plotmath expression
ggboxplot(df, x = "dose", y = "len") +
  geom_bracket(
    xmin = "0.5", xmax = "1", y.position = 30,
    label = "list(~italic(p)<=0.001)", type = "expression",
    tip.length = c(0.2, 0.02)
  )

# Specify multiple brackets manually
ggboxplot(df, x = "dose", y = "len") +
```

```
  geom_bracket(
    xmin = c("0.5", "1"), xmax = c("1", "2"),
    y.position = c(30, 35), label = c("***", "**"),
    tip.length = 0.01
  )

# Compute statistical tests and add p-values
stat.test <- compare_means(len ~ dose, ToothGrowth, method = "t.test")
ggboxplot(df, x = "dose", y = "len") +
  geom_bracket(
    aes(xmin = group1, xmax = group2, label = signif(p, 2)),
    data = stat.test, y.position = 35
  )

# Increase step length between brackets
ggboxplot(df, x = "dose", y = "len") +
  geom_bracket(
    aes(xmin = group1, xmax = group2, label = signif(p, 2)),
    data = stat.test, y.position = 35, step.increase = 0.1
  )

# Or specify the positions of each comparison
ggboxplot(df, x = "dose", y = "len") +
  geom_bracket(
    aes(xmin = group1, xmax = group2, label = signif(p, 2)),
    data = stat.test, y.position = c(32, 35, 38)
  )

# Vertical bracket (e.g. to annotate a plot with a continuous x axis)
ggscatter(df, x = "len", y = "len") +
  geom_bracket(
    orientation = "vertical",
    ymin = 10, ymax = 25, x.position = 34,
    label = "p < 0.05"
  )
```

---

stat_central_tendency *Add Central Tendency Measures to a GGPlot*

Description

Add central tendency measures (mean, median, mode) to density and histogram plots created using ggplots.

Note that, normally, the mode is used for categorical data where we wish to know which is the most common category. Therefore, we can have two or more values that share the highest frequency. This might be problematic for a continuous variable.

For a continuous variable, we can consider using mean or median as the measures of the central tendency.

Usage

```
stat_central_tendency(
  mapping = NULL,
  data = NULL,
  geom = c("line", "point"),
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  type = c("mean", "median", "mode"),
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".

- For more information and other ways to specify the position, see the layer position documentation.

na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.
type	the type of central tendency measure to be used. Possible values include: "mean", "median", "mode".
...	other arguments to pass to geom_line.

See Also

ggdensity

Examples

```
# Simple density plot
data("mtcars")
ggdensity(mtcars, x = "mpg", fill = "red") +
  scale_x_continuous(limits = c(-1, 50)) +
  stat_central_tendency(type = "mean", linetype = "dashed")

# Color by groups
data(iris)
ggdensity(iris, "Sepal.Length", color = "Species") +
  stat_central_tendency(aes(color = Species), type = "median", linetype = 2)

# Use geom = "point" for central tendency
data(iris)
ggdensity(iris, "Sepal.Length", color = "Species") +
  stat_central_tendency(
    aes(color = Species),
    type = "median",
    geom = "point", size = 4
  )

# Facet
ggdensity(iris, "Sepal.Length", facet.by = "Species") +
  stat_central_tendency(type = "mean", color = "red", linetype = 2) +
  stat_central_tendency(type = "median", color = "blue", linetype = 2)
```

stat_chull *Plot convex hull of a set of points*

Description

Plot convex hull of a set of points.

Usage

```
stat_chull(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
geom	The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following: • A `Geom` ggproto subclass, for example `GeomPoint`. • A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`. • For more information and other ways to specify the geom, see the layer geom documentation.

position — A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a geom_*() function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is geom_area(stat = "density", adjust = 0.5). The stat's documentation lists which parameters it can accept.
- The key_glyph argument of layer() may also be passed on through .... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

See Also

ggpar, ggscatter

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)

# scatter plot with convex hull
ggscatter(df, x = "wt", y = "mpg", color = "cyl") +
  stat_chull(aes(color = cyl))

ggscatter(df, x = "wt", y = "mpg", color = "cyl") +
  stat_chull(aes(color = cyl, fill = cyl), alpha = 0.1, geom = "polygon")
```

---

stat_compare_means *Add Mean Comparison P-values to a ggplot*

Description

Add mean comparison p-values to a ggplot, such as box plots, dot plots and stripcharts.

Usage

```
stat_compare_means(
  mapping = NULL,
  data = NULL,
  method = NULL,
  paired = FALSE,
  id = NULL,
  method.args = list(),
  ref.group = NULL,
  comparisons = NULL,
  hide.ns = FALSE,
  label.sep = ", ",
  label = NULL,
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  vjust = 0,
  tip.length = 0.03,
  bracket.size = 0.3,
  step.increase = 0,
  symnum.args = list(),
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
```

```
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  signif.cutoffs = NULL,
  signif.symbols = NULL,
  ns.symbol = "ns",
  use.four.stars = FALSE,
  show.signif = TRUE,
  geom = "text",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
method	a character string indicating which method to be used for comparing means.
paired	a logical indicating whether you want a paired test. Used only in t.test and in wilcox.test.
id	optional character string naming a column that identifies matched subjects for a **paired** comparison (paired = TRUE, with method = "t.test" or "wilcox.test", and without comparisons). Without it the paired test pairs observations by row order, so the p-value is wrong when the data are not sorted so that the compared groups align by subject. Providing id pairs the observations by subject id instead (row-order independent); see compare_means.
method.args	a list of additional arguments used for the test method. For example one might use method.args = list(alternative = "greater") for wilcoxon test.
ref.group	a character string specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group). ref.group can be also ".all.". In this case, each of the grouping variable levels is compared to all (i.e. basemean).

`comparisons` A list of length-2 vectors. The entries in the vector are either the names of 2 values on the x-axis or the 2 integers that correspond to the index of the groups of interest, to be compared.

Note: when `comparisons` is specified, each pairwise test is computed independently and the displayed p-values are **not** adjusted for multiple comparisons. For p-values corrected for multiple testing, use `geom_pwc()`, or `stat_pvalue_manual()` together with `compare_means(..., p.adjust.method = )`.

`hide.ns` logical value. If TRUE, hide ns symbol when displaying significance levels.

`label.sep` a character string to separate the terms. Default is ", ", to separate the test method name and the p-value.

`label` character string specifying label type. Allowed values include `"p.signif"` (shows the significance levels), `"p.format"` (shows the formatted p-value), and `"p.format.signif"` (shows the formatted p-value followed by significance stars, e.g., "p = 0.01 **").

`label.x.npc, label.y.npc`

can be `numeric` or `character` vector of the same length as the number of groups and/or panels. If too short they will be recycled.

- If `numeric`, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".
- If `character`, allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

`label.x, label.y`

`numeric` Coordinates (in data units) to be used for absolute positioning of the label. If too short they will be recycled.

`vjust` move the text up or down relative to the bracket.

`tip.length` numeric vector with the fraction that the bracket tips go down to indicate the precise column. Default is 0.03. Can be of same length as the number of comparisons to adjust specifically the tip length of each comparison. For example tip.length = c(0.01, 0.03).

If too short they will be recycled.

Note: when `comparisons` is set, brackets are drawn via `ggsignif::geom_signif()` and `tip.length` is a fraction of the trained data range, so the absolute tip length scales with your data (plots with different scales get different tip lengths). To obtain visually constant tip lengths across plots, compute the tests with `compare_means()` and draw them with `stat_pvalue_manual(..., tip.length.ref = "axis")` (which makes `tip.length` a fraction of the y-axis range). Passing `tip.length.ref` directly to `stat_compare_means()` has no effect and emits an informative message.

`bracket.size` Width of the lines of the bracket.

`step.increase` numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.

`symnum.args` a list of arguments to pass to the function `symnum` for symbolic number coding of p-values. For example, `symnum.args <- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns"))`.

In other words, we use the following convention for symbols indicating statistical significance:

- ns: p > 0.05
- *: p <= 0.05
- **: p <= 0.01
- ***: p <= 0.001
- ****: p <= 0.0001

Note: If `signif.cutoffs` is provided, it takes precedence over `symnum.args`.

`p.format.style` character string specifying the p-value formatting style. One of: `"default"` (backward compatible, uses scientific notation), `"apa"` (APA style, no leading zero), `"nejm"` (NEJM style), `"lancet"` (Lancet style), `"ama"` (AMA style), `"graphpad"` (GraphPad style), or `"scientific"` (scientific notation for GWAS). See `list_p_format_styles` for details.

`p.digits` integer specifying the number of decimal places for p-values. If provided, overrides the style default.

`p.leading.zero` logical indicating whether to include leading zero before decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

`p.min.threshold`

numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold". If provided, overrides the style default.

`p.decimal.mark` character string to use as the decimal mark. If NULL, uses `getOption("OutDec")`.

`signif.cutoffs` numeric vector of p-value cutoffs in descending order for assigning significance symbols. For example, `c(0.10, 0.05, 0.01)` means p < 0.10 gets "*", p < 0.05 gets "**", p < 0.01 gets "***". If `use.four.stars = TRUE`, can include a fourth level. Default is NULL, which uses the package defaults.

`signif.symbols` character vector of symbols corresponding to `signif.cutoffs`. If NULL, auto-generated as "*", "**", "***" (and "****" if `use.four.stars = TRUE`).

`ns.symbol` character string for non-significant results. Default is "ns". Use "" (empty string) to show nothing.

`use.four.stars` logical. If TRUE, allows four stars (****) for the most significant level. Default is FALSE.

`show.signif` logical. If TRUE (default), shows significance symbols when using `label = "p.format.signif"`. If FALSE, falls back to showing only the p-value (equivalent to `label = "p.format"`) with a warning.

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

position — A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm — If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

... — other arguments to pass to geom_text or geom_label.

Details

For grouped plots, if one or more subsets do not contain enough levels to identify the requested comparison, those subsets are skipped and valid subsets are still tested. This avoids fatal layer failures in sparse grouped data settings (Issue #663).

See Also

compare_means

Examples

```
# Load data
data("ToothGrowth")
head(ToothGrowth)

# Two independent groups
# :::::::::::::::::::::::::::::::::::::::::::::
p <- ggboxplot(ToothGrowth,
  x = "supp", y = "len",
  color = "supp", palette = "npg", add = "jitter"
)

#  Add p-value
p + stat_compare_means()
```

```
# Change method
p + stat_compare_means(method = "t.test")

# Paired samples
# ++++++++++++++++++++++++++++++++++++++++++++++
ggpaired(ToothGrowth,
  x = "supp", y = "len",
  color = "supp", line.color = "gray", line.size = 0.4,
  palette = "npg"
) +
  stat_compare_means(paired = TRUE)

# Paired samples matched by a subject id column (row-order independent):
# `id` gives the correct p-value even when the rows are not in subject order.
# ++++++++++++++++++++++++++++++++++++++++++++++
tg <- ToothGrowth
tg$id <- c(1:30, 30:1) # subjects, with the second group in reverse order
ggpaired(tg, x = "supp", y = "len", color = "supp",
  line.color = "gray", palette = "npg") +
  stat_compare_means(paired = TRUE, id = "id")

# More than two groups
# ++++++++++++++++++++++++++++++++++++++++++++++
# Pairwise comparisons: Specify the comparisons you want
my_comparisons <- list(c("0.5", "1"), c("1", "2"), c("0.5", "2"))
ggboxplot(ToothGrowth,
  x = "dose", y = "len",
  color = "dose", palette = "npg"
) +
  # Add pairwise comparisons p-value
  stat_compare_means(comparisons = my_comparisons, label.y = c(29, 35, 40)) +
  stat_compare_means(label.y = 45) # Add global Anova p-value

# Multiple pairwise test against a reference group
ggboxplot(ToothGrowth,
  x = "dose", y = "len",
  color = "dose", palette = "npg"
) +
  stat_compare_means(method = "anova", label.y = 40) + # Add global p-value
  stat_compare_means(aes(label = after_stat(p.signif)),
    method = "t.test", ref.group = "0.5"
  )

# Multiple grouping variables
# ++++++++++++++++++++++++++++++++++++++++++++++
# Box plot facetted by "dose"
p <- ggboxplot(ToothGrowth,
  x = "supp", y = "len",
  color = "supp", palette = "npg",
  add = "jitter",
  facet.by = "dose", short.panel.labs = FALSE
)
# Use only p.format as label. Remove method name.
```

```
p + stat_compare_means(
  aes(label = paste0("p = ", after_stat(p.format)))
)
```

`stat_conf_ellipse` *Plot confidence ellipses.*

Description

Plot confidence ellipses around barycenters. The method for computing confidence ellipses has been modified from `FactoMineR::coord.ellipse()`.

Usage

```
stat_conf_ellipse(
  mapping = NULL,
  data = NULL,
  geom = "path",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  level = 0.95,
  npoint = 100,
  bary = TRUE,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.
`data`	The data to be displayed in this layer. There are three options: If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`. A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created. A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).
`geom`	The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point".
- For more information and other ways to specify the geom, see the layer geom documentation.

position — A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

level — confidence level used to construct the ellipses. By default, 0.95.

npoint — number of points used to draw the ellipses.

bary — logical value. If TRUE, the coordinates of the ellipse around the barycentre of individuals are calculated.

... — Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a stat_*() function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is stat_density(geom = "area", outline.type = "both"). The geom's documentation lists which parameters it can accept.

- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

See Also

`stat_conf_ellipse`

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)

# scatter plot with confidence ellipses
ggscatter(df, x = "wt", y = "mpg", color = "cyl") +
  stat_conf_ellipse(aes(color = cyl))

ggscatter(df, x = "wt", y = "mpg", color = "cyl") +
  stat_conf_ellipse(aes(color = cyl, fill = cyl), alpha = 0.1, geom = "polygon")
```

---

stat_cor *Add Correlation Coefficients with P-values to a Scatter Plot*

Description

Add correlation coefficients with p-values to a scatter plot. Can be also used to add 'R2'.

Usage

```
stat_cor(
  mapping = NULL,
  data = NULL,
  method = "pearson",
  alternative = "two.sided",
  cor.coef.name = c("R", "rho", "tau"),
  label.sep = ", ",
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  label.y.step = 1.4,
```

```
  label.anchor = c("data", "panel"),
  output.type = "expression",
  digits = 2,
  r.digits = digits,
  p.digits = digits,
  r.accuracy = NULL,
  p.accuracy = NULL,
  r.leading.zero = NULL,
  p.format.style = "default",
  p.leading.zero = NULL,
  p.decimal.mark = NULL,
  p.coef.name = "p",
  conf.level = 0.95,
  geom = "text",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
method	a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default), "kendall", or "spearman".
alternative	a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
cor.coef.name	character. Can be one of "R" (pearson coef), "rho" (spearman coef) and "tau" (kendall coef). Uppercase and lowercase are allowed.
label.sep	a character string to separate the terms. Default is ", ", to separate the correlation coefficient and the p.value.
label.x.npc, label.y.npc	can be numeric or character vector of the same length as the number of groups and/or panels. If too short they will be recycled.

- If numeric, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".
- If character, allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

If too short they will be recycled.

label.x, label.y

numeric Coordinates (in data units) to be used for absolute positioning of the label. If too short they will be recycled.

label.y.step numeric value giving the vertical spacing (in text-line units) between the labels of successive groups when several groups are present. Default is 1.4 (unchanged behavior). Set label.y.step = 0 to stop the per-group vertical shift, so that labels align across facet panels when a factor is mapped to an aesthetic that also defines the facets. This places the labels flush with the top of each panel; it is similar in spirit to the aes(group = 1) workaround, which instead leaves them one text line lower.

label.anchor character. How label.x.npc/label.y.npc are interpreted. "data" (default) converts them to data coordinates using the data range, so the label follows the data; the output is unchanged from previous versions. "panel" places the label at the true panel-relative position (npc), so the labels stay aligned across panels/facets whose axis ranges differ - e.g. with facet_wrap(scales = "free_y"), with geom_smooth() extending each panel by a different amount, or across separate plots combined with ggarrange(). An explicit label.x/label.y always stays in data units regardless of label.anchor. Requires ggplot2 >= 3.5.0 (already a dependency).

output.type character One of "expression", "latex", "tex" or "text".

digits, r.digits, p.digits

integer indicating the number of decimal places (round) or significant digits (signif) to be used for the correlation coefficient and the p-value, respectively. In the default p-value label style, the displayed p-value keeps the legacy raw value unless p.accuracy is set; p.digits is used by non-default p.format.style outputs and the computed p variable.

r.accuracy a real value specifying the number of decimal places of precision for the correlation coefficient. Default is NULL. Use (e.g.) 0.01 to show 2 decimal places of precision. If specified, then r.digits is ignored.

p.accuracy a real value specifying the number of decimal places of precision for the p-value. Default is NULL. Use (e.g.) 0.0001 to show 4 decimal places of precision. If specified, then p.digits is ignored.

r.leading.zero logical. Whether to include the leading zero before the decimal point in the correlation coefficient (e.g., "0.73" vs ".73"). Default (NULL) keeps the leading zero; set to FALSE for APA-style reporting.

p.format.style character specifying the p-value formatting style. One of "default", "apa", "nejm", "lancet", "ama", "graphpad", "scientific". Default is "default" for backward compatibility.

p.leading.zero logical. Whether to include leading zero before decimal point (e.g., "0.05" vs ".05"). If NULL, uses the style's default setting.

p.decimal.mark character string to use as the decimal mark. If NULL, uses getOption("OutDec").

p.coef.name character. Symbol used for the p-value label. Default is "p"; use "P" for an uppercase p-value label. For output.type = "expression" this should be a single valid plotmath symbol (e.g. "P"), since it is parsed as an expression.

conf.level confidence level for the confidence interval of the correlation coefficient, used to compute the conf.int.low, conf.int.high and conf.int.label computed variables (see the **Computed variables** section). Default is 0.95. A confidence interval is only available for method = "pearson" (with at least 4 complete observations); for "spearman"/"kendall" the confidence-interval variables are NA.

geom The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point".
- For more information and other ways to specify the geom, see the layer geom documentation.

position A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

... other arguments to pass to geom_text or geom_label.

Computed variables

r correlation coefficient

rr correlation coefficient squared

rmse root mean square deviation (RMSE/RMSD) between `x` and `y`, computed on the complete pairs; meaningful when `x` and `y` are on the same scale (e.g. predicted vs. reference values). Rounded and formatted with the coefficient settings (`r.digits` / `r.accuracy`).

r.label formatted label for the correlation coefficient

rr.label formatted label for the squared correlation coefficient

p.label label for the p-value

rmse.label formatted label for the RMSE/RMSD

conf.int.low, conf.int.high lower and upper bounds of the confidence interval of the correlation coefficient (Pearson only; `NA` for Spearman/Kendall), at the level given by `conf.level`

conf.int.label formatted label for the confidence interval, e.g. `"95% CI [0.21, 0.75]"` (`NA` when the interval is unavailable)

label default label displayed by `stat_cor()`

See Also

`ggscatter`. For an alternative implementation with more control over label positioning (native NPC coordinates, per-group vertical and horizontal steps), see `ggpmisc::stat_correlation()`.

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)

# Scatter plot with correlation coefficient
# ::::::::::::::::::::::::::::::::::::::::::::::::
sp <- ggscatter(df,
  x = "wt", y = "mpg",
  add = "reg.line", # Add regressin line
  add.params = list(color = "blue", fill = "lightgray"), # Customize reg. line
  conf.int = TRUE # Add confidence interval
)
# Add correlation coefficient
sp + stat_cor(method = "pearson", label.x = 3, label.y = 30)

# Specify the number of decimal places of precision for p and r
# Using 3 decimal places for the p-value and
# 2 decimal places for the correlation coefficient (r)
sp + stat_cor(p.accuracy = 0.001, r.accuracy = 0.01)

# Show only the r.label but not the p.label
sp + stat_cor(aes(label = after_stat(r.label)), label.x = 3)

# Use R2 instead of R
ggscatter(df, x = "wt", y = "mpg", add = "reg.line") +
  stat_cor(
    aes(label = paste(after_stat(rr.label), after_stat(p.label), sep = "~`,`~")),
```

```
    label.x = 3
  )

# Show the RMSE/RMSD (root mean square deviation) between x and y
# (useful for agreement between paired measurements on the same scale)
sp + stat_cor(aes(label = after_stat(rmse.label)), label.x = 3)

# Combine the correlation coefficient and the RMSE (comma-separated)
sp + stat_cor(
  aes(label = paste(after_stat(r.label), after_stat(rmse.label), sep = "~`,`~")),
  label.x = 3
)

# Show the confidence interval of the correlation coefficient (Pearson)
sp + stat_cor(aes(label = after_stat(conf.int.label)), label.x = 3)

# Correlation coefficient with its confidence interval
sp + stat_cor(
  aes(label = paste(after_stat(r.label), after_stat(conf.int.label), sep = "~`,`~")),
  label.x = 1.5
)

# Color by groups and facet
# ::::::::::::::::::::::::::::::::::::::::::::::::::
sp <- ggscatter(df,
  x = "wt", y = "mpg",
  color = "cyl", palette = "jco",
  add = "reg.line", conf.int = TRUE
)
sp + stat_cor(aes(color = cyl), label.x = 3)
```

`stat_friedman_test` *Add Friedman Test P-values to a GGPlot*

Description

Add automatically Friedman test p-values to a ggplot, such as box plots, dot plots and stripcharts.

Usage

```
stat_friedman_test(
  mapping = NULL,
  data = NULL,
  wid = NULL,
  group.by = NULL,
  label = "{method}, p = {p.format}",
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
```

```
  label.y = NULL,
  step.increase = 0.1,
  p.adjust.method = "holm",
  significance = list(),
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  geom = "text",
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  parse = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
wid	(factor) column name containing individuals/subjects identifier. Should be unique per individual. Required only for repeated measure tests ("one_way_repeated", "two_way_repeated", "friedman_test", etc).
group.by	(optional) character vector specifying the grouping variable; it should be used only for grouped plots. Possible values are : • "x.var": Group by the x-axis variable and perform the test between legend groups. In other words, the p-value is compute between legend groups at each x position • "legend.var": Group by the legend variable and perform the test between x-axis groups. In other words, the test is performed between the x-groups for each legend level.
label	the column containing the label (e.g.: label = "p" or label = "p.adj"), where p is the p-value. Can be also an expression that can be formatted by the glue() package. For example, when specifying label = "t-test, p = {p}", the expression {p} will be replaced by its value. Values inserted through a glue expression

are taken from the raw data column and are **not** rounded by `p.digits`; to round within a glue expression, wrap the value, e.g. `label = "p = \{signif(p, 3)\}"` (or `{format_p_value(p)}` for the p-value house style).

`label.x.npc`, `label.y.npc`

can be `numeric` or `character` vector of the same length as the number of groups and/or panels. If too short they will be recycled.

- If `numeric`, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".
- If `character`, allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

`label.x`, `label.y`

`numeric` Coordinates (in data units) to be used for absolute positioning of the label. If too short they will be recycled.

`step.increase` numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.

`p.adjust.method`

method for adjusting p values (see `p.adjust`). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

`significance` a list of arguments specifying the significance cutpoints and symbols. For example, `significance <- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns"))`.

In other words, we use the following convention for symbols indicating statistical significance:

- `ns`: p > 0.05
- `*`: p <= 0.05
- `**`: p <= 0.01
- `***`: p <= 0.001
- `****`: p <= 0.0001

Note: `significance` is kept for backward compatibility. For functions that also expose `signif.cutoffs` and related parameters, `significance` takes precedence when provided.

`p.format.style` character string specifying the p-value formatting style applied to recognized numeric p-value label columns (see `p.digits`). One of: `"default"` (backward compatible, uses scientific notation), `"apa"` (APA style, no leading zero), `"nejm"` (NEJM style), `"lancet"` (Lancet style), `"ama"` (AMA style), `"graphpad"` (GraphPad style), or `"scientific"` (scientific notation for GWAS). See `list_p_format_styles` for details. Default is `"default"`, which leaves the rendered p-value labels unchanged.

`p.digits` integer indicating the number of digits used to format recognized **numeric** p-value label columns (`label` is one of `"p"`, `"p.adj"`, `"p.value"`, `"p.val"`, `"pval"`, `"padj"`). Default is 3. Set to `NULL` to print the raw value without rounding (this

also disables p.format.style and the related formatting arguments). All other labels are left unchanged: other numeric columns (e.g. "statistic", "n", effect sizes), significance symbols ("p.signif", "p.adj.signif"), already-formatted strings, and glue expressions. A p-named column whose values fall outside [0, 1] is also left as-is. Uses the same formatting engine (format_p_value()) as stat_anova_test() for consistency across layers. The style of this formatting is further controlled by p.format.style, p.leading.zero, p.min.threshold and p.decimal.mark (same arguments as stat_compare_means()).

p.leading.zero logical indicating whether to include the leading zero before the decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

p.min.threshold
numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold" (e.g. p.min.threshold = 0.001 renders very small p-values as "< 0.001"). If provided, overrides the style default.

p.decimal.mark character string to use as the decimal mark. If NULL, uses getOption("OutDec").

geom The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point".
- For more information and other ways to specify the geom, see the layer geom documentation.

position A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
...	other arguments passed to the function geom_bracket() or geom_text()

Computed variables

- statistic: the value of the test statistic (Chi-squared).
- df: the degrees of freedom of the approximate chi-squared distribution of the test statistic.
- p: p-value.
- p.adj: Adjusted p-values.
- p.signif: P-value significance.
- p.adj.signif: Adjusted p-value significance.
- p.format: Formatted p-value.
- p.format.signif: Formatted p-value with significance symbols.
- p.adj.format: Formatted adjusted p-value.
- n: number of samples.

Examples

```
# Data preparation
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Transform `dose` into factor variable
df <- ToothGrowth
df$dose <- as.factor(df$dose)
df$id <- as.factor(c(rep(1:10, 3), rep(11:20, 3)))
# Add a random grouping variable
set.seed(123)
df$group <- sample(factor(rep(c("grp1", "grp2", "grp3"), 20)))
df$len <- ifelse(df$group == "grp2", df$len + 2, df$len)
df$len <- ifelse(df$group == "grp3", df$len + 7, df$len)
head(df, 3)


# Basic boxplot
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a basic boxplot
# Add 5% and 10% space to the plot bottom and the top, respectively
bxp <- ggboxplot(df, x = "dose", y = "len") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))

# Add the p-value to the boxplot
bxp + stat_friedman_test(aes(wid = id))

# Change the label position
# Using coordinates in data units
bxp + stat_friedman_test(aes(wid = id), label.x = "1", label.y = 10, hjust = 0)

# Format the p-value differently
```

```
custom_p_format <- function(p) {
  rstatix::p_format(p, accuracy = 0.0001, digits = 3, leading.zero = FALSE)
}
bxp + stat_friedman_test(
  aes(wid = id),
  label = "Friedman test, italic(p) = {custom_p_format(p)}{p.signif}"
)

# Show a detailed label in italic
bxp + stat_friedman_test(aes(wid = id), label = "as_detailed_italic")


# Faceted plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a ggplot facet
df$id <- rep(1:10, 6)
bxp <- ggboxplot(df, x = "dose", y = "len", facet.by = "supp") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))
# Add p-values
bxp + stat_friedman_test(aes(wid = id))


# Grouped plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
df$id <- rep(1:10, 6)
bxp <- ggboxplot(df, x = "dose", y = "len", color = "supp", palette = "jco")

# For each legend group, computes tests within x variable groups
bxp + stat_friedman_test(aes(wid = id, group = supp, color = supp), within = "x")

# For each x-position, computes tests within legend variable groups
bxp + stat_friedman_test(
  aes(wid = id, group = supp, color = supp),
  within = "group", label = "p = {p.format}"
)
```

`stat_kruskal_test` *Add Kruskal-Wallis Test P-values to a GGPlot*

Description

Add Kruskal-Wallis test p-values to a ggplot, such as box plots, dot plots and stripcharts.

Usage

```
stat_kruskal_test(
  mapping = NULL,
  data = NULL,
  group.by = NULL,
```

```
  label = "{method}, p = {p.format}",
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  step.increase = 0.1,
  p.adjust.method = "holm",
  significance = list(),
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  geom = "text",
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  parse = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
group.by	(optional) character vector specifying the grouping variable; it should be used only for grouped plots. Possible values are : • "x.var": Group by the x-axis variable and perform the test between legend groups. In other words, the p-value is compute between legend groups at each x position • "legend.var": Group by the legend variable and perform the test between x-axis groups. In other words, the test is performed between the x-groups for each legend level.
label	the column containing the label (e.g.: label = "p" or label = "p.adj"), where p is the p-value. Can be also an expression that can be formatted by the glue() package. For example, when specifying label = "t-test, p = {p}", the expression

{p} will be replaced by its value. Values inserted through a glue expression are taken from the raw data column and are **not** rounded by p.digits; to round within a glue expression, wrap the value, e.g. label = "p = \{signif(p, 3)\}" (or {format_p_value(p)} for the p-value house style).

label.x.npc, label.y.npc

can be numeric or character vector of the same length as the number of groups and/or panels. If too short they will be recycled.

- If numeric, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".
- If character, allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

label.x, label.y

numeric Coordinates (in data units) to be used for absolute positioning of the label. If too short they will be recycled.

step.increase numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.

p.adjust.method

method for adjusting p values (see p.adjust). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

significance a list of arguments specifying the significance cutpoints and symbols. For example, significance <- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns")).

In other words, we use the following convention for symbols indicating statistical significance:

- ns: p > 0.05
- *: p <= 0.05
- **: p <= 0.01
- ***: p <= 0.001
- ****: p <= 0.0001

Note: significance is kept for backward compatibility. For functions that also expose signif.cutoffs and related parameters, significance takes precedence when provided.

p.format.style character string specifying the p-value formatting style applied to recognized numeric p-value label columns (see p.digits). One of: "default" (backward compatible, uses scientific notation), "apa" (APA style, no leading zero), "nejm" (NEJM style), "lancet" (Lancet style), "ama" (AMA style), "graphpad" (GraphPad style), or "scientific" (scientific notation for GWAS). See list_p_format_styles for details. Default is "default", which leaves the rendered p-value labels unchanged.

p.digits integer indicating the number of digits used to format recognized **numeric** p-value label columns (label is one of "p", "p.adj", "p.value", "p.val", "pval",

"padj"). Default is 3. Set to NULL to print the raw value without rounding (this also disables p.format.style and the related formatting arguments). All other labels are left unchanged: other numeric columns (e.g. "statistic", "n", effect sizes), significance symbols ("p.signif", "p.adj.signif"), already-formatted strings, and glue expressions. A p-named column whose values fall outside [0, 1] is also left as-is. Uses the same formatting engine (format_p_value()) as stat_anova_test() for consistency across layers. The style of this formatting is further controlled by p.format.style, p.leading.zero, p.min.threshold and p.decimal.mark (same arguments as stat_compare_means()).

p.leading.zero logical indicating whether to include the leading zero before the decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

p.min.threshold
numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold" (e.g. p.min.threshold = 0.001 renders very small p-values as "< 0.001"). If provided, overrides the style default.

p.decimal.mark character string to use as the decimal mark. If NULL, uses getOption("OutDec").

geom The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point".
- For more information and other ways to specify the geom, see the layer geom documentation.

position A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that

	define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.
parse	If TRUE, the labels will be parsed into expressions and displayed as described in ?plotmath.
...	other arguments passed to the function geom_bracket() or geom_text()

Computed variables

- statistic: the Kruskal-Wallis rank sum chi-squared statistic used to compute the p-value.
- p: p-value.
- p.adj: Adjusted p-values.
- p.signif: P-value significance.
- p.adj.signif: Adjusted p-value significance.
- p.format: Formatted p-value.
- p.format.signif: Formatted p-value with significance symbols.
- p.adj.format: Formatted adjusted p-value.
- n: number of samples.

Examples

```
# Data preparation
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Transform `dose` into factor variable
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Add a random grouping variable
set.seed(123)
df$group <- sample(factor(rep(c("grp1", "grp2", "grp3"), 20)))
df$len <- ifelse(df$group == "grp2", df$len + 2, df$len)
df$len <- ifelse(df$group == "grp3", df$len + 7, df$len)
head(df, 3)


# Basic boxplot
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a basic boxplot
# Add 5% and 10% space to the plot bottom and the top, respectively
bxp <- ggboxplot(df, x = "dose", y = "len") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))

# Add the p-value to the boxplot
bxp + stat_kruskal_test()

# Change the label position
# Using coordinates in data units
bxp + stat_kruskal_test(label.x = "1", label.y = 10, hjust = 0)

# Format the p-value differently
```

```
custom_p_format <- function(p) {
  rstatix::p_format(p, accuracy = 0.0001, digits = 3, leading.zero = FALSE)
}
bxp + stat_kruskal_test(
  label = "Kruskal-Wallis, italic(p) = {custom_p_format(p)}{p.signif}"
)

# Show a detailed label in italic
bxp + stat_kruskal_test(label = "as_detailed_italic")


# Faceted plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a ggplot facet
bxp <- ggboxplot(df, x = "dose", y = "len", facet.by = "supp") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))
# Add p-values
bxp + stat_kruskal_test()


# Grouped plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bxp2 <- ggboxplot(df, x = "group", y = "len", color = "dose", palette = "npg")

# For each x-position, computes tests between legend groups
bxp2 + stat_kruskal_test(aes(group = dose), label = "p = {p.format}{p.signif}")

#  For each legend group, computes tests between x variable groups
bxp2 + stat_kruskal_test(aes(group = dose, color = dose), group.by = "legend.var")
```

`stat_mean` *Draw group mean points*

Description

Draw the mean point of each group.

Usage

```
stat_mean(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.
...	other arguments to pass to geom_point.

See Also

stat_conf_ellipse, stat_chull and ggscatter

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)

# Scatter plot with ellipses and group mean points
ggscatter(df,
  x = "wt", y = "mpg",
  color = "cyl", shape = "cyl", ellipse = TRUE
) +
  stat_mean(aes(color = cyl, shape = cyl), size = 4)
```

---

stat_overlay_normal_density

Overlay Normal Density Plot

Description

Overlay normal density plot (with the same mean and SD) to the density distribution of 'x'. This is useful for visually inspecting the degree of deviance from normality.

Usage

```
stat_overlay_normal_density(
  mapping = NULL,
  data = NULL,
  geom = "line",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
na.rm	If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.
...	other arguments to pass to geom_line.

See Also

ggdensity

Examples

```
# Simpledensity plot
data("mtcars")
ggdensity(mtcars, x = "mpg", fill = "red") +
  scale_x_continuous(limits = c(-1, 50)) +
  stat_overlay_normal_density(color = "red", linetype = "dashed")

# Color by groups
data(iris)
ggdensity(iris, "Sepal.Length", color = "Species") +
  stat_overlay_normal_density(aes(color = Species), linetype = "dashed")


# Facet
ggdensity(iris, "Sepal.Length", facet.by = "Species") +
  stat_overlay_normal_density(color = "red", linetype = "dashed")
```

---

stat_pvalue_manual *Add P-values Manually to a ggplot*

Description

Add p-values manually to a ggplot, such as box plots, dot plots and stripcharts. Frequently asked questions are available on Datanovia ggpubr FAQ page, for example:

- How to Add P-Values onto Basic GGPlots
- How to Add Adjusted P-values to a Multi-Panel GGPlot
- How to Add P-values to GGPlot Facets
- How to Add P-Values Generated Elsewhere to a GGPlot
- How to Add P-Values onto a Grouped GGPlot using the ggpubr R Package
- How to Create Stacked Bar Plots with Error Bars and P-values
- How to Add P-Values onto Horizontal GGPlots

Usage

```
stat_pvalue_manual(
  data,
  label = NULL,
  y.position = "y.position",
  xmin = "group1",
  xmax = "group2",
  x = NULL,
  size = 3.88,
  label.size = size,
  p.digits = 3,
```

```
  p.format.style = "default",
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  bracket.size = 0.3,
  bracket.nudge.y = 0,
  bracket.shorten = 0,
  color = "black",
  linetype = 1,
  tip.length = 0.03,
  tip.length.ref = c("data", "axis"),
  remove.bracket = FALSE,
  step.increase = 0,
  step.group.by = NULL,
  hide.ns = FALSE,
  vjust = 0,
  coord.flip = FALSE,
  position = "identity",
  inherit.aes = FALSE,
  ...
)
```

Arguments

`data`	a data frame containing statistical test results. The expected default format should contain the following columns: `group1 | group2 | p | y.position | etc`. `group1` and `group2` are the groups that have been compared. `p` is the resulting p-value. `y.position` is the y coordinates of the p-values in the plot.
`label`	the column containing the label (e.g.: label = "p" or label = "p.adj"), where `p` is the p-value. Can be also an expression that can be formatted by the `glue()` package. For example, when specifying label = "t-test, p = {p}", the expression {p} will be replaced by its value. Values inserted through a glue expression are taken from the raw data column and are **not** rounded by `p.digits`; to round within a glue expression, wrap the value, e.g. `label = "p = \{signif(p, 3)\}"` (or `{format_p_value(p)}` for the p-value house style).
`y.position`	column containing the coordinates (in data units) to be used for absolute positioning of the label. Default value is "y.position". Can be also a numeric vector.
`xmin`	column containing the position of the left sides of the brackets. Default value is "group1".
`xmax`	(optional) column containing the position of the right sides of the brackets. Default value is "group2". If NULL, the p-values are plotted as a simple text.
`x`	x position of the p-value. Should be used only when you want plot the p-value as text (without brackets).
`size, label.size`	size of label text.
`p.digits`	integer indicating the number of digits used to format recognized **numeric** p-value label columns (`label` is one of `"p"`, `"p.adj"`, `"p.value"`, `"p.val"`, `"pval"`,

	"padj"). Default is 3. Set to NULL to print the raw value without rounding (this also disables p.format.style and the related formatting arguments). All other labels are left unchanged: other numeric columns (e.g. "statistic", "n", effect sizes), significance symbols ("p.signif", "p.adj.signif"), already-formatted strings, and glue expressions. A p-named column whose values fall outside [0, 1] is also left as-is. Uses the same formatting engine (format_p_value()) as stat_anova_test() for consistency across layers. The style of this formatting is further controlled by p.format.style, p.leading.zero, p.min.threshold and p.decimal.mark (same arguments as stat_compare_means()).
p.format.style	character string specifying the p-value formatting style applied to recognized numeric p-value label columns (see p.digits). One of: "default" (backward compatible, uses scientific notation), "apa" (APA style, no leading zero), "nejm" (NEJM style), "lancet" (Lancet style), "ama" (AMA style), "graphpad" (GraphPad style), or "scientific" (scientific notation for GWAS). See list_p_format_styles for details. Default is "default", which leaves the rendered p-value labels unchanged.
p.leading.zero	logical indicating whether to include the leading zero before the decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.
p.min.threshold	numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold" (e.g. p.min.threshold = 0.001 renders very small p-values as "< 0.001"). If provided, overrides the style default.
p.decimal.mark	character string to use as the decimal mark. If NULL, uses getOption("OutDec").
bracket.size	Width of the lines of the bracket.
bracket.nudge.y	Vertical adjustment to nudge brackets by. Useful to move up or move down the bracket. If positive value, brackets will be moved up; if negative value, brackets are moved down.
bracket.shorten	a small numeric value in [0-1] for shortening the with of bracket.
color	text and line color. Can be variable name in the data for coloring by groups.
linetype	linetype. Can be variable name in the data for changing linetype by groups.
tip.length	numeric vector with the fraction that the bracket tips go down to indicate the precise column. Interpreted relative to the reference set by tip.length.ref. Default is 0.03.
tip.length.ref	character string specifying what tip.length is a fraction of. Either "data" (default): fraction of the trained data range (tips scale with the data; existing plots unchanged); or "axis": fraction of the y-axis range (ylim/scale_y_*), giving visually constant tip lengths across plots with different scales (#362).
remove.bracket	logical, if TRUE, brackets are removed from the plot. Considered only in the situation, where comparisons are performed against reference group or against "all".
step.increase	numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.
step.group.by	a variable name for grouping brackets before adding step.increase. Useful to group bracket by facet panel.

hide.ns	can be logical value or a character vector. • Case when logical value. If TRUE, hide ns symbol when displaying significance levels. Filter is done by checking the column p.adj.signif, p.signif, p.adj and p. • Case when character value. Possible values are "p" or "p.adj", for filtering out non significant.
vjust	move the text up or down relative to the bracket. Can be also a column name available in the data.
coord.flip	logical. If TRUE, flip x and y coordinates so that horizontal becomes vertical, and vertical, horizontal. When adding the p-values to a horizontal ggplot (generated using coord_flip()), you need to specify the option coord.flip = TRUE.
position	position adjustment, either as a string, or the result of a call to a position adjustment function.
inherit.aes	If FALSE (the default), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. stat_pvalue_manual(). Most layers should use TRUE (the **ggplot2** default), but stat_pvalue_manual() uses its own data with different column names than the parent plot data, so FALSE avoids conflicts between parent plot aesthetics (like fill or color) and the p-value annotation data.
...	other arguments passed to the function geom_bracket() or geom_text()

See Also

stat_compare_means

Examples

```
# T-test
stat.test <- compare_means(
  len ~ dose,
  data = ToothGrowth,
  method = "t.test"
)
stat.test

# Create a simple box plot
p <- ggboxplot(ToothGrowth, x = "dose", y = "len")
p

# Perform a t-test between groups
stat.test <- compare_means(
  len ~ dose,
  data = ToothGrowth,
  method = "t.test"
)
stat.test

# Add p-values manually from stat.test data
```

```
# First specify the y.position of each comparison
stat.test <- stat.test %>%
  mutate(y.position = c(29, 35, 39))
p + stat_pvalue_manual(stat.test, label = "p.adj")

# Customize the label with glue expression
# (https://github.com/tidyverse/glue)
p + stat_pvalue_manual(stat.test, label = "p = {p.adj}")


# Grouped bar plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ToothGrowth$dose <- as.factor(ToothGrowth$dose)
# Comparisons against reference
stat.test <- compare_means(
  len ~ dose,
  data = ToothGrowth, group.by = "supp",
  method = "t.test", ref.group = "0.5"
)
stat.test
# Plot
bp <- ggbarplot(ToothGrowth,
  x = "supp", y = "len",
  fill = "dose", palette = "jco",
  add = "mean_sd", add.params = list(group = "dose"),
  position = position_dodge(0.8)
)
bp + stat_pvalue_manual(
  stat.test,
  x = "supp", y.position = 33,
  label = "p.signif",
  position = position_dodge(0.8)
)
```

---

stat_pwc *Add Pairwise Comparisons P-values to a GGPlot*

Description

add pairwise comparison p-values to a ggplot such as box plots, dot plots and stripcharts.

Usage

```
stat_pwc(
  mapping = NULL,
  data = NULL,
  method = "wilcox_test",
  method.args = list(),
  ref.group = NULL,
```

```
  label = "p.format",
  y.position = NULL,
  group.by = NULL,
  dodge = 0.8,
  bracket.nudge.y = 0.05,
  bracket.shorten = 0,
  bracket.group.by = c("x.var", "legend.var"),
  step.increase = 0.12,
  tip.length = 0.03,
  size = 0.3,
  label.size = 3.88,
  family = "",
  vjust = 0,
  hjust = 0.5,
  p.adjust.method = "holm",
  p.adjust.by = c("group", "panel"),
  p.adjust.n = NULL,
  symnum.args = list(),
  hide.ns = FALSE,
  remove.bracket = FALSE,
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  signif.cutoffs = NULL,
  signif.symbols = NULL,
  ns.symbol = "ns",
  use.four.stars = FALSE,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  parse = FALSE,
  ...
)

geom_pwc(
  mapping = NULL,
  data = NULL,
  stat = "pwc",
  method = "wilcox_test",
  method.args = list(),
  ref.group = NULL,
  label = "p.format",
  y.position = NULL,
  group.by = NULL,
  dodge = 0.8,
```

```
  stack = FALSE,
  step.increase = 0.12,
  tip.length = 0.03,
  bracket.nudge.y = 0.05,
  bracket.shorten = 0,
  bracket.group.by = c("x.var", "legend.var"),
  size = 0.3,
  label.size = 3.88,
  family = "",
  vjust = 0,
  hjust = 0.5,
  p.adjust.method = "holm",
  p.adjust.by = c("group", "panel"),
  p.adjust.n = NULL,
  symnum.args = list(),
  hide.ns = FALSE,
  remove.bracket = FALSE,
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  signif.cutoffs = NULL,
  signif.symbols = NULL,
  ns.symbol = "ns",
  use.four.stars = FALSE,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  parse = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

`method` a character string indicating which method to be used for pairwise comparisons. Default is `"wilcox_test"`. Allowed methods include pairwise comparisons methods implemented in the `rstatix` R package. These methods are: `"wilcox_test"`, `"t_test"`, `"sign_test"`,`"dunn_test"`, `"emmeans_test"`, `"tukey_hsd"`, `"games_howell_test"`.

`method.args` a list of additional arguments used for the test method. For example one might use `method.args = list(alternative = "greater")` for wilcoxon test.

`ref.group` a character string or a numeric value specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group).

`ref.group` can be also `"all"`. In this case, each of the grouping variable levels is compared to all (i.e. basemean).

Allowed values can be:

- **numeric value**: specifying the rank of the reference group. For example, use `ref.group = 1` when the first group is the reference; use `ref.group = 2` when the second group is the reference, and so on. This works for all situations, including i) when comparisons are performed between x-axis groups and ii) when comparisons are performed between legend groups.
- **character value**: For example, you can use `ref.group = "ctrl"` instead of using the numeric rank value of the "ctrl" group.
- **"all"**: In this case, each of the grouping variable levels is compared to all (i.e. basemean).

`label` character string specifying label. Can be:

- the column containing the label (e.g.: `label = "p"` or `label = "p.adj"`), where p is the p-value. Other possible values are `"p.signif"`, `"p.adj.signif"`, `"p.format"`, `"p.format.signif"`, `"p.adj.format"`.
- an expression that can be formatted by the `glue()` package. For example, when specifying `label = "Wilcoxon, p = \{p\}"`, the expression {p} will be replaced by its value.
- a combination of plotmath expressions and glue expressions. You may want some of the statistical parameter in italic; for example:`label = "Wilcoxon,italic(p)= {p}"`

.

`y.position` numeric vector with the y positions of the brackets

`group.by` (optional) character vector specifying the grouping variable; it should be used only for grouped plots. Possible values are :

- `"x.var"`: Group by the x-axis variable and perform the test between legend groups. In other words, the p-value is compute between legend groups at each x position
- `"legend.var"`: Group by the legend variable and perform the test between x-axis groups. In other words, the test is performed between the x-groups for each legend level.

`dodge` dodge width for grouped ggplot/test. Default is 0.8. It's used to dodge the brackets position when `group.by = "legend.var"`.

`bracket.nudge.y`
: Vertical adjustment to nudge brackets by (in fraction of the total height). Useful to move up or move down the bracket. If positive value, brackets will be moved up; if negative value, brackets are moved down.

`bracket.shorten`
: a small numeric value in [0-1] for shortening the width of bracket.

`bracket.group.by`
: (optional); a variable name for grouping brackets before adding step.increase. Useful for grouped plots. Possible values include `"x.var"` and `"legend.var"`.

`step.increase`
: numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.

`tip.length`
: numeric vector with the fraction of total height that the bar goes down to indicate the precise column/

`size`
: change the width of the lines of the bracket

`label.size`
: change the size of the label text

`family`
: change the font used for the text

`vjust`
: move the text up or down relative to the bracket.

`hjust`
: move the text left or right relative to the bracket.

`p.adjust.method`
: method for adjusting p values (see `p.adjust`). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Ignored when the specified method is `"tukey_hsd"` or `"games_howell_test"` because they come with internal p adjustment method. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

`p.adjust.by`
: possible value is one of `c("group", "panel")`. Default is `"group"`: for a grouped data, if pairwise test is performed, then the p-values are adjusted for each group level independently. P-values are adjusted by panel when `p.adjust.by = "panel"`.

`p.adjust.n`
: optional single number giving the number of comparisons to use for the p-value adjustment, passed as `n` to `p.adjust`. Default is `NULL`, which uses the number of p-values actually computed (the standard behavior). Set it when the displayed comparisons are a subset of a larger family and the adjustment should reflect that larger family size. As required by `p.adjust`, `p.adjust.n` must be greater than or equal to the number of p-values being adjusted. It affects the adjustment that `geom_pwc()` performs itself: the panel-level adjustment (`p.adjust.by = "panel"`) and cases where the test returns a single set of p-values (a single comparison, or grouped comparisons with two levels per group). It does not alter the adjustment that pairwise tests already perform internally per group.

`symnum.args`
: a list of arguments to pass to the function `symnum` for symbolic number coding of p-values. For example, `symnum.args = list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns"))`. In other words, we use the following convention for symbols indicating statistical significance:

- ns: p > 0.05
- *: p <= 0.05
- **: p <= 0.01
- ***: p <= 0.001
- ****: p <= 0.0001

Note: If symnum.args is provided, it takes precedence over signif.cutoffs.

hide.ns can be logical value (TRUE or FALSE) or a character vector ("p.adj" or "p").

remove.bracket logical, if TRUE, brackets are removed from the plot.

- Case when logical value. If TRUE, hide ns symbol when displaying significance levels. Filter is done by checking the column p.adj.signif, p.signif, p.adj and p.
- Case when character value. Possible values are "p" or "p.adj", for filtering out non significant.

p.format.style character string specifying the p-value formatting style. One of: "default" (backward compatible, uses scientific notation), "apa" (APA style, no leading zero), "nejm" (NEJM style), "lancet" (Lancet style), "ama" (AMA style), "graphpad" (GraphPad style), or "scientific" (scientific notation for GWAS). See list_p_format_styles for details.

p.digits integer specifying the number of decimal places for p-values. If provided, overrides the style default.

p.leading.zero logical indicating whether to include leading zero before decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

p.min.threshold

numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold". If provided, overrides the style default.

p.decimal.mark character string to use as the decimal mark. If NULL, uses getOption("OutDec").

signif.cutoffs numeric vector of p-value cutoffs in descending order for assigning significance symbols. For example, c(0.10, 0.05, 0.01) means p < 0.10 gets "*", p < 0.05 gets "**", p < 0.01 gets "***". Default is NULL, which uses the package defaults.

signif.symbols character vector of symbols corresponding to signif.cutoffs. If NULL, auto-generated as "*", "**", "***" (and "****" if use.four.stars = TRUE).

ns.symbol character string for non-significant results. Default is "ns". Use "" (empty string) to show nothing.

use.four.stars logical. If TRUE, allows four stars (****) for the most significant level. Default is FALSE.

position A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".

- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

`show.legend` logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

`parse` logical for parsing plotmath expression.

`...` other arguments passed on to `layer()`). These are often aesthetics, used to set an aesthetic to a fixed value, like `color = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`stack` logical value. Default is FALSE; should be set to TRUE for stacked bar plots or line plots. If TRUE, then the brackets are automatically removed and the dodge value is set to zero.

Details

Notes on adjusted p-values and facet. When using the ggplot facet functions, the p-values are computed and adjusted by panel, without taking into account the other panels. This is by design in ggplot2.

In this case, when there is only one computed p-value by panel, then using 'label = "p"' or 'label = "p.adj"' will give the same results using 'geom_pwc()'. Again, p-value computation and adjustment in a given facet panel is done independently to the other panels.

For grouped plots with sparse data, if some subsets do not contain enough levels or observations for the requested test, those subsets are skipped while preserving valid inferential comparisons from comparable subsets. Informative messages list skipped grouped subsets and their skip reasons (for example, missing `ref.group` or fewer than two comparison levels).

One might want to adjust the p-values of all the facet panels together. There are two solutions for that:

- Using ggadjust_pvalue(p) after creating the plot p
- or adding the adjusted p-value manually using stat_pvalue_manual(). Read more at:
  - How to Add P-values to GGPlot Facets
  - Add P-values to GGPlot Facets with Different Scales

See Also

ggadjust_pvalue

Examples

```
df <- ToothGrowth
df$dose <- factor(df$dose)

# Data preparation
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Transform `dose` into factor variable
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Add a random grouping variable
df$group <- factor(rep(c("grp1", "grp2"), 30))
head(df, 3)


# Two groups by x position
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

# Create a box plot
# Add 10% spaces between the p-value labels and the plot border
bxp <- ggboxplot(
  df,
  x = "dose", y = "len",
  color = "supp", palette = c("#00AFBB", "#E7B800")
) +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.10)))


# Add p-values onto the box plots
# label can be "p.format"  or "p.adj.format"
bxp + geom_pwc(
  aes(group = supp),
  tip.length = 0,
  method = "t_test", label = "p.format"
)

# Show adjusted p-values and significance levels
# Hide ns (non-significant)
bxp + geom_pwc(
  aes(group = supp),
  tip.length = 0,
  method = "t_test", label = "{p.adj.format}{p.adj.signif}",
  p.adjust.method = "bonferroni", p.adjust.by = "panel",
```

```
  hide.ns = TRUE
)

# Complex cases
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# 1. Add p-values of OJ vs VC at each dose group
bxp.complex <- bxp +
  geom_pwc(
    aes(group = supp),
    tip.length = 0,
    method = "t_test", label = "p.adj.format",
    p.adjust.method = "bonferroni", p.adjust.by = "panel"
  )
# 2. Add pairwise comparisons between dose levels
# Nudge up the brackets by 20% of the total height
bxp.complex <- bxp.complex +
  geom_pwc(
    method = "t_test", label = "p.adj.format",
    p.adjust.method = "bonferroni",
    bracket.nudge.y = 0.2
  )
# 3. Display the plot
bxp.complex


# Three groups by x position
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

# Simple plots
# ____________________________________

# Box plots with p-values
bxp <- ggboxplot(
  df,
  x = "supp", y = "len", fill = "dose",
  palette = "npg"
)
bxp +
  geom_pwc(
    aes(group = dose),
    tip.length = 0,
    method = "t_test", label = "p.adj.format",
    bracket.nudge.y = -0.08
  ) +
  scale_y_continuous(expand = expansion(mult = c(0, 0.1)))

# Bar plots with p-values
bp <- ggbarplot(
  df,
  x = "supp", y = "len", fill = "dose",
  palette = "npg", add = "mean_sd",
  position = position_dodge(0.8)
)
```

```
bp +
  geom_pwc(
    aes(group = dose),
    tip.length = 0,
    method = "t_test", label = "p.adj.format",
    bracket.nudge.y = -0.08
  ) +
  scale_y_continuous(expand = expansion(mult = c(0, 0.1)))
```

`stat_regline_equation` *Add Regression Line Equation and R-Square to a GGPlot.*

Description

Add regression line equation and R^2 to a ggplot. Regression model is fitted using the function `lm`.

Usage

```
stat_regline_equation(
  mapping = NULL,
  data = NULL,
  formula = y ~ x,
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  label.y.step = 1.4,
  label.anchor = c("data", "panel"),
  output.type = "expression",
  decreasing = TRUE,
  coef.digits = 2,
  rr.digits = 2,
  geom = "text",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

`data` The data to be displayed in this layer. There are three options:

If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A `function` will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A `function` can be created from a `formula` (e.g. `~ head(.x, 10)`).

`formula` a formula object

`label.x.npc, label.y.npc`

can be `numeric` or `character` vector of the same length as the number of groups and/or panels. If too short they will be recycled.

- If `numeric`, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".
- If `character`, allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

If too short they will be recycled.

`label.x, label.y`

`numeric` Coordinates (in data units) to be used for absolute positioning of the label. If too short they will be recycled.

`label.y.step` numeric value giving the vertical spacing (in text-line units) between the labels of successive groups. Default is 1.4 (unchanged behavior). Set `label.y.step = 0` to stop the per-group vertical shift so labels align across facet panels.

`label.anchor` character. How `label.x.npc`/`label.y.npc` are interpreted. `"data"` (default) converts them to data coordinates (the previous behavior). `"panel"` places the label at the true panel-relative position (npc), so labels stay aligned across panels/facets with different axis ranges (e.g. `scales = "free_y"` or `geom_smooth()` extending each panel). An explicit `label.x`/`label.y` always stays in data units. Requires ggplot2 >= 3.5.0 (already a dependency).

`output.type` character One of "expression", "latex" or "text".

`decreasing` logical. If `TRUE` (the default), the equation is formatted in standard mathematical convention with terms in decreasing order of powers (e.g., "y = 2*x + 1"). If `FALSE`, terms are in increasing order (e.g., "y = 1 + 2*x").

`coef.digits` integer indicating the number of significant digits to use for the regression equation coefficients. Default is 2.

`rr.digits` integer indicating the number of significant digits to use for R2 and adjusted R2. Default is 2.

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.

- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE` (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to `TRUE` to inherit aesthetics from the parent ggplot layer.

`...` other arguments to pass to `geom_text` or `geom_label`.

Computed variables

x x position for left edge

y y position near upper edge

eq.label equation for the fitted polynomial as a character string to be parsed

rr.label R^2 of the fitted model as a character string to be parsed

adj.rr.label Adjusted R^2 of the fitted model as a character string to be parsed

AIC.label AIC for the fitted model.

BIC.label BIC for the fitted model.

hjust Set to zero to override the default of the "text" geom.

References

`stat_regline_equation()` was adapted from the source code of `stat_poly_eq()` in the 'ggpmisc' package by Pedro J. Aphalo. See `ggpmisc::stat_poly_eq()` as an alternative with additional features.

See Also

ggscatter

Examples

```
# Simple scatter plot with correlation coefficient and
# regression line
# ::::::::::::::::::::::::::::::::::::::::::::::::::
ggscatter(mtcars, x = "wt", y = "mpg", add = "reg.line") +
  stat_cor(label.x = 3, label.y = 34) +
  stat_regline_equation(label.x = 3, label.y = 32)


# Groupped scatter plot
# ::::::::::::::::::::::::::::::::::::::::::::::::::
ggscatter(
  iris,
  x = "Sepal.Length", y = "Sepal.Width",
  color = "Species", palette = "jco",
  add = "reg.line"
) +
  facet_wrap(~Species) +
  stat_cor(label.y = 4.4) +
  stat_regline_equation(label.y = 4.2)

# Polynomial equation
# ::::::::::::::::::::::::::::::::::::::::::::::::::

# Demo data
set.seed(4321)
x <- 1:100
y <- (x + x^2 + x^3) + rnorm(length(x), mean = 0, sd = mean(x^3) / 4)
my.data <- data.frame(x, y,
  group = c("A", "B"),
  y2 = y * c(0.5, 2), block = c("a", "a", "b", "b")
)

# Fit polynomial regression line and add labels
formula <- y ~ poly(x, 3, raw = TRUE)
p <- ggplot(my.data, aes(x, y2, color = group)) +
  geom_point() +
  stat_smooth(aes(fill = group, color = group), method = "lm", formula = formula) +
  stat_regline_equation(
    aes(label = paste(..eq.label.., ..adj.rr.label.., sep = "~~~~")),
    formula = formula
  ) +
  theme_bw()
ggpar(p, palette = "jco")
```

stat_stars *Add Stars to a Scatter Plot*

Description

Create a star plot by drawing segments from group centroid to each points.

Usage

```
stat_stars(
  mapping = NULL,
  data = NULL,
  geom = "segment",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
geom	The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following: • A Geom ggproto subclass, for example GeomPoint. • A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point". • For more information and other ways to specify the geom, see the layer geom documentation.

position — A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter".
- For more information and other ways to specify the position, see the layer position documentation.

na.rm — If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.

inherit.aes — If FALSE (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to TRUE to inherit aesthetics from the parent ggplot layer.

... — other arguments to pass to geom_segment.

See Also

ggscatter

Examples

```
# Load data
data("mtcars")
df <- mtcars
df$cyl <- as.factor(df$cyl)

# Scatter plot with ellipses and group mean points
ggscatter(df,
  x = "wt", y = "mpg",
  color = "cyl", shape = "cyl",
  mean.point = TRUE, ellipse = TRUE
) +
  stat_stars(aes(color = cyl))
```

stat_welch_anova_test *Add Welch One-Way ANOVA Test P-values to a GGPlot*

Description

Add Welch one-way ANOVA test p-values to a ggplot, such as box plots, dot plots and stripcharts.

Usage

```
stat_welch_anova_test(
  mapping = NULL,
  data = NULL,
  group.by = NULL,
  label = "{method}, p = {p.format}",
  label.x.npc = "left",
  label.y.npc = "top",
  label.x = NULL,
  label.y = NULL,
  step.increase = 0.1,
  p.adjust.method = "holm",
  significance = list(),
  p.format.style = "default",
  p.digits = NULL,
  p.leading.zero = NULL,
  p.min.threshold = NULL,
  p.decimal.mark = NULL,
  geom = "text",
  position = "identity",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  parse = FALSE,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.

A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

group.by (optional) character vector specifying the grouping variable; it should be used only for grouped plots. Possible values are :

- "x.var": Group by the x-axis variable and perform the test between legend groups. In other words, the p-value is compute between legend groups at each x position
- "legend.var": Group by the legend variable and perform the test between x-axis groups. In other words, the test is performed between the x-groups for each legend level.

label the column containing the label (e.g.: label = "p" or label = "p.adj"), where p is the p-value. Can be also an expression that can be formatted by the glue() package. For example, when specifying label = "t-test, p = {p}", the expression {p} will be replaced by its value. Values inserted through a glue expression are taken from the raw data column and are **not** rounded by p.digits; to round within a glue expression, wrap the value, e.g. label = "p = \{signif(p, 3)\}" (or {format_p_value(p)} for the p-value house style).

label.x.npc, label.y.npc

can be numeric or character vector of the same length as the number of groups and/or panels. If too short they will be recycled.

- If numeric, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".
- If character, allowed values include: i) one of c('right', 'left', 'center', 'centre', 'middle') for x-axis; ii) and one of c('bottom', 'top', 'center', 'centre', 'middle') for y-axis.

label.x, label.y

numeric Coordinates (in data units) to be used for absolute positioning of the label. If too short they will be recycled.

step.increase numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.

p.adjust.method

method for adjusting p values (see p.adjust). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".

significance a list of arguments specifying the significance cutpoints and symbols. For example, significance <- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns")).

In other words, we use the following convention for symbols indicating statistical significance:

- ns: p > 0.05
- *: p <= 0.05
- **: p <= 0.01

- ***: p <= 0.001
- ****: p <= 0.0001

Note: significance is kept for backward compatibility. For functions that also expose signif.cutoffs and related parameters, significance takes precedence when provided.

p.format.style character string specifying the p-value formatting style applied to recognized numeric p-value label columns (see p.digits). One of: "default" (backward compatible, uses scientific notation), "apa" (APA style, no leading zero), "nejm" (NEJM style), "lancet" (Lancet style), "ama" (AMA style), "graphpad" (GraphPad style), or "scientific" (scientific notation for GWAS). See list_p_format_styles for details. Default is "default", which leaves the rendered p-value labels unchanged.

p.digits integer indicating the number of digits used to format recognized **numeric** p-value label columns (label is one of "p", "p.adj", "p.value", "p.val", "pval", "padj"). Default is 3. Set to NULL to print the raw value without rounding (this also disables p.format.style and the related formatting arguments). All other labels are left unchanged: other numeric columns (e.g. "statistic", "n", effect sizes), significance symbols ("p.signif", "p.adj.signif"), already-formatted strings, and glue expressions. A p-named column whose values fall outside [0, 1] is also left as-is. Uses the same formatting engine (format_p_value()) as stat_anova_test() for consistency across layers. The style of this formatting is further controlled by p.format.style, p.leading.zero, p.min.threshold and p.decimal.mark (same arguments as stat_compare_means()).

p.leading.zero logical indicating whether to include the leading zero before the decimal point (e.g., "0.05" vs ".05"). If provided, overrides the style default.

p.min.threshold

numeric specifying the minimum p-value to display exactly. Values below this threshold are shown as "< threshold" (e.g. p.min.threshold = 0.001 renders very small p-values as "< 0.001"). If provided, overrides the style default.

p.decimal.mark character string to use as the decimal mark. If NULL, uses getOption("OutDec").

geom The geometric object to use to display the data for this layer. When using a stat_*() function to construct a layer, the geom argument can be used to override the default coupling between stats and geoms. The geom argument accepts the following:

- A Geom ggproto subclass, for example GeomPoint.
- A string naming the geom. To give the geom as a string, strip the function name of the geom_ prefix. For example, to use geom_point(), give the geom as "point".
- For more information and other ways to specify the geom, see the layer geom documentation.

position A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position.

- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`na.rm` If FALSE (the default), removes missing values with a warning. If TRUE silently removes missing values.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`inherit.aes` If `FALSE` (the default for most ggpubr functions), overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification. Set to `TRUE` to inherit aesthetics from the parent ggplot layer.

`parse` If TRUE, the labels will be parsed into expressions and displayed as described in `?plotmath`.

`...` other arguments passed to the function `geom_bracket()` or `geom_text()`

Computed variables

- statistic: the value of the test statistic (F-value)
- DFn: Degrees of Freedom in the numerator (i.e. DF effect)
- DFd: Degrees of Freedom in the denominator (i.e., DF error)
- p: p-value.
- p.adj: Adjusted p-values.
- p.signif: P-value significance.
- p.adj.signif: Adjusted p-value significance.
- p.format: Formatted p-value.
- p.format.signif: Formatted p-value with significance symbols.
- p.adj.format: Formatted adjusted p-value.
- n: number of samples.

Examples

```
# Data preparation
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Transform `dose` into factor variable
df <- ToothGrowth
df$dose <- as.factor(df$dose)
# Add a random grouping variable
set.seed(123)
df$group <- sample(factor(rep(c("grp1", "grp2", "grp3"), 20)))
df$len <- ifelse(df$group == "grp2", df$len + 2, df$len)
```

```
df$len <- ifelse(df$group == "grp3", df$len + 7, df$len)
head(df, 3)


# Basic boxplot
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a basic boxplot
# Add 5% and 10% space to the plot bottom and the top, respectively
bxp <- ggboxplot(df, x = "dose", y = "len") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))

# Add the p-value to the boxplot
bxp + stat_welch_anova_test()

# Change the label position
# Using coordinates in data units
bxp + stat_welch_anova_test(label.x = "1", label.y = 10, hjust = 0)

# Format the p-value differently
custom_p_format <- function(p) {
  rstatix::p_format(p, accuracy = 0.0001, digits = 3, leading.zero = FALSE)
}
bxp + stat_welch_anova_test(
  label = "Welch Anova, italic(p) = {custom_p_format(p)}{p.signif}"
)

# Show a detailed label in italic
bxp + stat_welch_anova_test(label = "as_detailed_italic")


# Faceted plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
# Create a ggplot facet
bxp <- ggboxplot(df, x = "dose", y = "len", facet.by = "supp") +
  scale_y_continuous(expand = expansion(mult = c(0.05, 0.1)))
# Add p-values
bxp + stat_welch_anova_test()


# Grouped plots
# %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bxp2 <- ggboxplot(df, x = "group", y = "len", color = "dose", palette = "npg")

# For each x-position, computes tests between legend groups
bxp2 + stat_welch_anova_test(aes(group = dose), label = "p = {p.format}{p.signif}")

#  For each legend group, computes tests between x variable groups
bxp2 + stat_welch_anova_test(aes(group = dose, color = dose), group.by = "legend.var")
```

---

text_grob *Create a Text Graphical object*

Description

Create easily a customized text grob (graphical object). Wrapper around `textGrob`.

Usage

```
text_grob(
  label,
  just = "centre",
  hjust = NULL,
  vjust = NULL,
  rot = 0,
  color = "black",
  face = "plain",
  size = NULL,
  lineheight = NULL,
  family = NULL,
  ...
)
```

Arguments

`label`	A character or expression vector. Other objects are coerced by `as.graphicsAnnot`.
`just`	The justification of the text relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: `"left"`, `"right"`, `"centre"`, `"center"`, `"bottom"`, and `"top"`. For numeric values, 0 means left (bottom) alignment and 1 means right (top) alignment.
`hjust`	A numeric vector specifying horizontal justification. If specified, overrides the `just` setting.
`vjust`	A numeric vector specifying vertical justification. If specified, overrides the `just` setting.
`rot`	The angle to rotate the text.
`color`	text font color.
`face`	font face. Allowed values include one of `"plain"`, `"bold"`,`"italic"`, `"bold.italic"`.
`size`	font size (e.g.: size = 12)
`lineheight`	line height (e.g.: lineheight = 2).
`family`	font family.
`...`	other arguments passed to textGrob.

Value

a text grob.

Examples

```
text <- paste("iris data set gives the measurements in cm",
  "of the variables sepal length and width",
  "and petal length and width, respectively,",
  "for 50 flowers from each of 3 species of iris.",
  "The species are Iris setosa, versicolor, and virginica.",
  sep = "\n"
)

# Create a text grob
tgrob <- text_grob(text, face = "italic", color = "steelblue")
# Draw the text
as_ggplot(tgrob)
```

---

theme_pubr *Publication ready theme*

Description

- **theme_pubr()**: Create a publication ready theme
- **theme_pubclean()**: a clean theme without axis lines, to direct more attention to the data.
- **labs_pubr()**: Format only plot labels to a publication ready style
- **theme_classic2()**: Create a classic theme with axis lines.
- **clean_theme()**: Remove axis lines, ticks, texts and titles.
- **clean_table_theme()**: Clean the theme of a table, such as those created by `ggsummarytable()`

.

Usage

```
theme_pubr(
  base_size = 12,
  base_family = "",
  border = FALSE,
  margin = TRUE,
  legend = c("top", "bottom", "left", "right", "none"),
  x.text.angle = 0
)

theme_pubclean(base_size = 12, base_family = "", flip = FALSE)

labs_pubr(base_size = 14, base_family = "")

theme_classic2(base_size = 12, base_family = "")

clean_theme()

clean_table_theme()
```

Arguments

base_size	base font size
base_family	base font family
border	logical value. Default is FALSE. If TRUE, add panel border.
margin	logical value. Default is TRUE. If FALSE, reduce plot margin.
legend	character specifying legend position. Allowed values are one of c("top", "bottom", "left", "right", "none"). Default is "top" side position. to remove the legend use legend = "none". Legend position can be also specified using a numeric vector c(x, y). In this case it is possible to position the legend inside the plotting area. x and y are the coordinates of the legend box. Their values should be between 0 and 1. c(0,0) corresponds to the "bottom left" and c(1,1) corresponds to the "top right" position. For instance use legend = c(0.8, 0.2).
x.text.angle	Rotation angle of x axis tick labels. Default value is 0. Use 90 for vertical text.
flip	logical. If TRUE, grid lines are added to y axis instead of x axis.

Details

`theme_pubr()` sets `strip.clip = "off"` so that the facet strip background border is drawn at its full width (with the ggplot2 default `strip.clip = "on"` the border is clipped to the strip area and renders thinner than requested). As a side effect, a facet label that is wider than its panel will overflow the strip instead of being truncated; if that is a problem for very long labels, restore clipping with `+ theme(strip.clip = "on")`.

Examples

```
p <- ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point(aes(color = gear))

# Default plot
p

# Use theme_pubr()
p + theme_pubr()

# Format labels
p + labs_pubr()
```

---

theme_transparent *Create a ggplot with Transparent Background*

Description

Create a ggplot with transparent background.

Usage

```
theme_transparent(base_size = 12, base_family = "")
```

Arguments

base_size	base font size
base_family	base font family

See Also

theme_pubr

Examples

```
# Create a scatter plot
sp <- ggscatter(iris,
  x = "Sepal.Length", y = "Sepal.Width",
  color = "Species", palette = "jco",
  size = 3, alpha = 0.6
)
sp

# Transparent theme
sp + theme_transparent()
```

Index

add_summary, 4
aes(), *151*, *158*, *162*, *164*, *167*, *172*, *175*, *180*, *185*, *190*, *191*, *199*, *206*, *210*, *212*
annotate_figure, 7, *44*
Anova, *151*
anova, *16*
arrangeGrob, *9*
as.graphicsAnnot, *217*
as_ggplot, 9
as_npc, 9, *33*, *144*
as_npcx *(*as_npc*)*, 9
as_npcy *(*as_npc*)*, 9
axis_scale, 11

background_image, 12
bgcolor, 13
border, *13*, 13
brewer.pal, *35*, *148*

change_palette *(*set_palette*)*, 147
clean_lock_files, 14
clean_table_theme *(*theme_pubr*)*, 218
clean_theme *(*theme_pubr*)*, 218
colnames_style *(*ggtexttable*)*, 127
color_palette *(*set_palette*)*, 147
compare_means, 15, *167*, *168*, *170*
coord_fixed, *111*
coord_flip, *145*, *160*, *196*
create_aes, 19
create_p_label, 20

desc_statby, 21, *77*
diff_express, 22
drawDetails.splitText *(*ggparagraph*)*, 100

element_text, *25*, *26*, *146*
expression, *217*

facet, 23, *57*, *72*, *119*, *123*, *140*
facet_wrap, *23*
fill_palette *(*set_palette*)*, 147
font, 25
format_p_value, 27, *182*, *187*, *195*, *214*
fortify(), *151*, *158*, *162*, *164*, *167*, *172*, *175*, *180*, *185*, *190*, *192*, *199*, *207*, *210*, *212*

gene_citation, 29
gene_expression, 30
geom_boxplot, *57*
geom_bracket *(*stat_bracket*)*, 156
geom_density, *62*
geom_dotplot, *72*
geom_exec, 30
geom_histogram, *83*
geom_jitter, *119*
geom_label, *170*, *177*, *208*
geom_line, *163*, *192*
geom_point, *68*, *112*, *190*
geom_pwc, *39*, *168*
geom_pwc *(*stat_pwc*)*, 197
geom_segment, *211*
geom_text, *154*, *170*, *177*, *208*
geom_violin, *138*, *140*
get_breaks, 31
get_coord, *10*, 32, *144*
get_legend, 34, *44*
get_p_format_style, *28*, 36
get_palette, 35, *148*
get_summary_stats, *123*
ggadd, 37
ggadjust_pvalue, 39, *204*
ggarrange, *7*, *8*, 42, *80*, *123*, *176*
ggballoonplot, 45
ggbarplot, 48, *88*
ggboxplot, 54, *72*, *120*
ggdensity, 59, *84*, *163*, *192*
ggdonutchart, 63
ggdotchart, 65
ggdotplot, *58*, 69, *120*
ggecdf, 73

ggerrorplot, 76
ggexport, 79
gghistogram, *62*, 81
ggline, *39*, *51*, *78*, 85, *103*, *149*
ggmaplot, 90
ggpaired, 93
ggpar, *46*, *51*, *57, 58*, *62*, *64*, *68*, *72*, *75*, *78*, *83, 84*, *88*, *92*, *96*, 96, *103*, *108*, *112*, *119, 120*, *123*, *126, 127*, *140*, *149*, *165*
ggparagraph, 100
ggpie, *64*, 102
ggplot, *6*, *39*
ggplot(), *151*, *158*, *162*, *164*, *167*, *172*, *175*, *180*, *185*, *190*, *192*, *199*, *207*, *210*, *212*
ggpubr_args, 104
ggpubr_options, 106
ggqqplot, 106
ggscatter, 109, *116*, *165*, *178*, *191*, *209*, *211*
ggscatterhist, 114
ggstripchart, *58*, *72*, 117
ggsummarystats *(*ggsummarytable*)*, 121
ggsummarytable, 121, *218*
ggtext, 125
ggtexttable, 127
ggviolin, *58*, *72*, *120*, 136
glue, *40*, *152*, *180*, *182*, *185*, *187*, *194, 195*, *200*, *213, 214*
gradient_color, 142
gradient_fill *(*gradient_color*)*, 142
grid.arrange, *7*, *9*
grids, 143

key glyphs, *165*, *174*
kruskal.test, *16*

labs_pubr *(*theme_pubr*)*, 218
layer, *159*, *203*
layer geom, *153*, *162*, *164*, *169*, *173*, *177*, *182*, *187*, *190*, *192*, *208*, *210*, *214*
layer position, *154*, *158*, *163*, *165*, *170*, *173*, *177*, *182*, *187*, *190*, *192*, *203*, *208*, *211*, *215*
layer stat, *160*, *203*
layer(), *165*, *173, 174*
list_p_format_styles, *17*, *28*, *41*, 144, *153*, *169*, *181*, *186*, *195*, *202*, *214*
lm, *206*

mean_ci *(*add_summary*)*, 4
mean_range *(*add_summary*)*, 4
mean_sd *(*add_summary*)*, 4
mean_se, *6*
mean_se_ *(*add_summary*)*, 4
median_hilow_ *(*add_summary*)*, 4
median_iqr *(*add_summary*)*, 4
median_mad *(*add_summary*)*, 4
median_q1q3 *(*add_summary*)*, 4
median_range *(*add_summary*)*, 4

npc_to_data_coord, *10*, *33*, 144

p.adjust, *17*, *40*, *152*, *181*, *186*, *201*, *213*
palette, *97*, *142*
plot_grid, *7*, *42*, *44*
print.ggscatterhist *(*ggscatterhist*)*, 114
print.ggsummarystats *(*ggsummarytable*)*, 121
print.ggsummarystats_list *(*ggsummarytable*)*, 121

rotate, 145
rotate_axis_text, 145
rotate_x_text *(*rotate_axis_text*)*, 145
rotate_y_text *(*rotate_axis_text*)*, 145
rownames_style *(*ggtexttable*)*, 127
rremove, 146

scale_x_continuous, *31*
scale_y_continuous, *31*
set_palette, *142*, 147
show_line_types, *14*, *61*, *74*, *82*, *143*, 149
show_point_shapes, *5*, *38*, *67*, *110*, *115*, 149
stat_anova_test, 150, *182*, *187*, *195*, *214*
stat_bracket, 156
stat_central_tendency, 161
stat_chull, 164, *191*
stat_compare_means, 166, *182*, *187*, *195, 196*, *214*
stat_conf_ellipse, *112*, 172, *174*, *191*
stat_cor, *112*, 174
stat_ecdf, *75*
stat_ellipse, *111*
stat_friedman_test, 179
stat_kruskal_test, 184
stat_mean, 189
stat_overlay_normal_density, 191
stat_pvalue_manual, *168*, 193, *204*

stat_pwc, 197
stat_regline_equation, 206
stat_stars, *112*, 210
stat_welch_anova_test, 212
symnum, *16*, *40*, *168*, *201*

t.test, *16*, *167*
tab_add_border *(*ggtexttable*)*, 127
tab_add_footnote *(*ggtexttable*)*, 127
tab_add_hline *(*ggtexttable*)*, 127
tab_add_title *(*ggtexttable*)*, 127
tab_add_vline *(*ggtexttable*)*, 127
tab_cell_crossout *(*ggtexttable*)*, 127
tab_ncol *(*ggtexttable*)*, 127
tab_nrow *(*ggtexttable*)*, 127
table_cell_bg *(*ggtexttable*)*, 127
table_cell_font *(*ggtexttable*)*, 127
tbody_add_border *(*ggtexttable*)*, 127
tbody_style *(*ggtexttable*)*, 127
text_grob, *7*, 216
textGrob, *217*
thead_add_border *(*ggtexttable*)*, 127
theme_classic2 *(*theme_pubr*)*, 218
theme_cleveland *(*ggdotchart*)*, 65
theme_pubclean *(*theme_pubr*)*, 218
theme_pubr, *116*, 218, *220*
theme_transparent, 219
theme_void, *115*
ttheme *(*ggtexttable*)*, 127

wilcox.test, *16*, *167*

xscale *(*axis_scale*)*, 11

yscale *(*axis_scale*)*, 11