

# Package ‘ggpubr’

September 4, 2019

**Type** Package

**Title** 'ggplot2' Based Publication Ready Plots

**Version** 0.2.3

**Date** 2019-09-03

**Description** The 'ggplot2' package is excellent and flexible for elegant data visualization in R. However the default generated plots requires 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.

**License** GPL-2

**LazyData** TRUE

**Encoding** UTF-8

**Depends** R (>= 3.1.0), ggplot2, magrittr

**Imports** ggrepel, grid, ggsci, stats, utils, tidyr, purrr, dplyr (>= 0.7.1), cowplot, ggsignif, scales, gridExtra, glue, polynom, rlang

**Suggests** grDevices, knitr, RColorBrewer, gtable

**URL** <https://rpkgs.datanovia.com/ggpubr/>

**BugReports** <https://github.com/kassambara/ggpubr/issues>

**RoxygenNote** 6.1.1

**Collate** 'utilities\_color.R' 'utilities\_base.R' 'desc\_statby.R'  
'utilities.R' 'add\_summary.R' 'annotate\_figure.R' 'as\_ggplot.R'  
'axis\_scale.R' 'background\_image.R' 'bgcolor.R' 'border.R'  
'compare\_means.R' 'diff\_express.R' 'facet.R' 'font.R'  
'gene\_citation.R' 'geom\_bracket.R' 'geom\_exec.R' 'get\_legend.R'  
'get\_palette.R' 'ggadd.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' 'ggmapplot.R'

'ggpaired.R' 'ggparagraph.R' 'ggpubr\_args.R' 'ggqqplot.R'  
 'utilities\_label.R' 'stat\_cor.R' 'stat\_stars.R' 'ggscatter.R'  
 'ggscatterhist.R' 'ggstripchart.R' 'ggtext.R' 'ggtexttable.R'  
 'ggviolin.R' 'gradient\_color.R' 'grids.R' 'reexports.R'  
 'rotate.R' 'rotate\_axis\_text.R' 'remove.R' 'set\_palette.R'  
 'show\_line\_types.R' 'show\_point\_shapes.R'  
 'stat\_central\_tendency.R' 'stat\_compare\_means.R' 'stat\_mean.R'  
 'stat\_overlay\_normal\_density.R' 'stat\_pvalue\_manual.R'  
 'stat\_regline\_equation.R' 'text\_grob.R' 'theme\_pubr.R'  
 'theme\_transparent.R'

**NeedsCompilation** no

**Author** Alboukadel Kassambara [aut, cre]

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

**Repository** CRAN

**Date/Publication** 2019-09-03 22:00:02 UTC

## R topics documented:

add_summary . . . . .	3
annotate_figure . . . . .	5
as_ggplot . . . . .	7
axis_scale . . . . .	7
background_image . . . . .	8
bgcolor . . . . .	9
border . . . . .	10
compare_means . . . . .	10
desc_statby . . . . .	13
diff_express . . . . .	14
facet . . . . .	15
font . . . . .	16
gene_citation . . . . .	17
geom_exec . . . . .	18
get_legend . . . . .	19
get_palette . . . . .	20
ggadd . . . . .	21
ggarrange . . . . .	23
ggballoonplot . . . . .	25
ggbarplot . . . . .	27
ggboxplot . . . . .	32
ggdensity . . . . .	36
ggdonutchart . . . . .	38
ggdotchart . . . . .	40
ggdotplot . . . . .	44
ggecdf . . . . .	47
ggerrorplot . . . . .	49
ggexport . . . . .	52

gghistogram . . . . .	53
ggline . . . . .	56
ggmaplot . . . . .	61
ggpaired . . . . .	63
ggpar . . . . .	65
ggparagraph . . . . .	68
ggpie . . . . .	69
ggpubr_args . . . . .	71
ggqqplot . . . . .	73
ggscatter . . . . .	75
ggscatterhist . . . . .	79
ggstripchart . . . . .	81
ggtext . . . . .	85
ggtexttable . . . . .	87
ggviolin . . . . .	90
gradient_color . . . . .	94
grids . . . . .	95
rotate . . . . .	96
rotate_axis_text . . . . .	96
rremove . . . . .	97
set_palette . . . . .	98
show_line_types . . . . .	99
show_point_shapes . . . . .	100
stat_bracket . . . . .	100
stat_central_tendency . . . . .	103
stat_chull . . . . .	105
stat_compare_means . . . . .	106
stat_conf_ellipse . . . . .	109
stat_cor . . . . .	111
stat_mean . . . . .	113
stat_overlay_normal_density . . . . .	114
stat_pvalue_manual . . . . .	115
stat_regline_equation . . . . .	118
stat_stars . . . . .	120
text_grob . . . . .	121
theme_pubr . . . . .	123
theme_transparent . . . . .	124

**Index****125**

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

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

**Arguments**

<code>p</code>	a <code>ggplot</code> on which you want to add summary statistics.
<code>fun</code>	a function that is given the complete data and should return a data frame with variables <code>ymin</code> , <code>y</code> , and <code>ymax</code> . Allowed values are one of: "mean", "mean_se", "mean_sd", "mean_ci", "mean_range", "median", "median_iqr", "median_mad", "median_range".
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "linerrange", "crossbar")</code> . Default value is "pointrange".
<code>color</code>	point or outline color.
<code>fill</code>	fill color. Used only when <code>error.plot = "crossbar"</code> .
<code>group</code>	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.
<code>width</code>	numeric value between 0 and 1 specifying bar or box width. Example <code>width = 0.8</code> . Used only when <code>error.plot</code> is one of <code>c("crossbar", "errorbar")</code> .
<code>shape</code>	point shape. Allowed values can be displayed using the function <code>show_point_shapes()</code> .
<code>size</code>	numeric value in [0-1] specifying point and line size.
<code>linetype</code>	line type.
<code>show.legend</code>	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
<code>ci</code>	the percent range of the confidence interval (default is 0.95).
<code>data</code>	a <code>data.frame</code> to be displayed. If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot</code> .

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_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 median_iqr
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)
```

**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".

**Author(s)**

Alboukadel Kassambara <alboukadel.kassambara@gmail.com>

**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 = "I'm done, thanks :-)",
  fig.lab = "Figure 1", fig.lab.face = "bold"
)

```

---

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

---

**Description**

Transform the output of `arrangeGrob()` and `grid.arrange()` to a 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)
```

---

`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	<i>Add Background Image to ggplot2</i>
------------------	--

---

**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)**

Alboukadel Kassambara <alboukadel.kassambara@gmail.com>



## 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	<i>Set ggplot Panel Border Line</i>
--------	-------------------------------------

---

### 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 <a href="#">show_line_types</a> .

### Examples

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

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

# Add border
p + border()
```

---

compare_means	<i>Comparison of Means</i>
---------------	----------------------------

---

### Description

Performs one or multiple mean comparisons.

### Usage

```
compare_means(formula, data, method = "wilcox.test", paired = FALSE,
  group.by = NULL, ref.group = NULL, symnum.args = list(),
  p.adjust.method = "holm", ...)
```

**Arguments**

formula	<p>a formula of the form <math>x \sim \text{group}</math> where <math>x</math> is a numeric variable giving the data values and <math>\text{group}</math> is a factor with one or multiple levels giving the corresponding groups. For example, <code>formula = TP53 ~ cancer_group</code>.</p> <p>It's also possible to perform the test for multiple response variables at the same time. For example, <code>formula = c(TP53, PTEN) ~ cancer_group</code>.</p>
data	a data.frame containing the variables in the formula.
method	<p>the type of test. Default is <code>wilcox.test</code>. Allowed values include:</p> <ul style="list-style-type: none"> <li>• <code>t.test</code> (parametric) and <code>wilcox.test</code> (non-parametric). Perform comparison between two groups of samples. If the grouping variable contains more than two levels, then a pairwise comparison is performed.</li> <li>• <code>anova</code> (parametric) and <code>kruskal.test</code> (non-parametric). Perform one-way ANOVA test comparing multiple groups.</li> </ul>
paired	a logical indicating whether you want a paired test. Used only in <code>t.test</code> and in <code>wilcox.test</code> .
group.by	a character vector containing the name of grouping variables.
ref.group	<p>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).</p> <p><code>ref.group</code> can be also <code>".all."</code>. In this case, each of the grouping variable levels is compared to all (i.e. basemean).</p>
symnum.args	<p>a list of arguments to pass to the function <code>symnum</code> for symbolic number coding of p-values. For example, <code>symnum.args &lt;- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, 1), symbol = c("****", "***", "**", "*", "ns"))</code>.</p> <p>In other words, we use the following convention for symbols indicating statistical significance:</p> <ul style="list-style-type: none"> <li>• ns: <math>p &gt; 0.05</math></li> <li>• *: <math>p \leq 0.05</math></li> <li>• **: <math>p \leq 0.01</math></li> <li>• ***: <math>p \leq 0.001</math></li> <li>• ****: <math>p \leq 0.0001</math></li> </ul>
p.adjust.method	<p>method for adjusting p values (see <code>p.adjust</code>). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Allowed values include <code>"holm"</code>, <code>"hochberg"</code>, <code>"hommel"</code>, <code>"bonferroni"</code>, <code>"BH"</code>, <code>"BY"</code>, <code>"fdr"</code>, <code>"none"</code>. If you don't want to adjust the p value (not recommended), use <code>p.adjust.method = "none"</code>.</p> <p>Note that, when the formula contains multiple variables, the p-value adjustment is done independently for each variable.</p>
...	Other arguments to be passed to the test function.

**Value**

return 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.signif`: the significance level.
- `method`: the statistical test used to compare groups.

## 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)

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

# pairwise comparisons
#::::::::::::::::::::::::::::::::::::::::::
# As dose contains more than 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")
```

---

desc_statby	<i>Descriptive statistics by groups</i>
-------------	---

---

### Description

Computes descriptive statistics by groups for a measure variable.

### Usage

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

### Arguments

<code>data</code>	a data frame.
<code>measure.var</code>	the name of a column containing the variable to be summarized.
<code>grps</code>	a character vector containing grouping variables; e.g.: <code>grps = c("grp1", "grp2")</code>
<code>ci</code>	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 mean
- **se**: standard error of the mean
- **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 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())
```

---

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, 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

<code>p</code>	a <code>ggplot</code>
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>nrow, ncol</code>	Number of rows and columns in the panel. Used only when the data is faceted by one grouping variable.
<code>scales</code>	should axis scales of panels be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y").
<code>short.panel.labs</code>	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.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>panel.labs.background</code>	a list to customize the background of panel labels. Should contain the combination of the following elements: <ul style="list-style-type: none"> <li>• <code>color, linetype, size</code>: background line color, type and size</li> <li>• <code>fill</code>: background fill color.</li> </ul> For example, <code>panel.labs.background = list(color = "blue", fill = "pink", linetype = "dashed", size = 0.5)</code> .
<code>panel.labs.font</code>	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 <code>element_text()</code> .

### 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")
```

---

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",
             "BMP2", "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
        )
```

---

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 carrying 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**

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

**Examples**

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

## End(Not run)
```

---

get\_legend

*Extract Legends from a ggplot object*

---

**Description**

Extract the legend labels from a ggplot object.

**Usage**

```
get_legend(p)
```

**Arguments**

**p** an object of class ggplot or a list of ggplots. If p is a list, only the first legend is returned.

**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", "uc-scgb", "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))

```

---

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,
  binwidth = NULL, dotsize = size, linetype = 1, show.legend = NA,
  error.plot = "pointrange", ci = 0.95, data = NULL,
  position = position_dodge(0.8), p_geom = "")

```

**Arguments**

<code>p</code>	a <code>ggplot</code>
<code>add</code>	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_mad", "median_range".
<code>color</code>	point or outline color.
<code>fill</code>	fill color. Used only when <code>error.plot = "crossbar"</code> .
<code>group</code>	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.
<code>width</code>	numeric value between 0 and 1 specifying bar or box width. Example <code>width = 0.8</code> . Used only when <code>error.plot</code> is one of <code>c("crossbar", "errorbar")</code> .
<code>shape</code>	point shape. Allowed values can be displayed using the function <code>show_point_shapes()</code> .
<code>size</code>	numeric value in [0-1] specifying point and line size.
<code>alpha</code>	numeric value specifying fill color transparency. Value should be in [0, 1], where 0 is full transparency and 1 is no transparency.
<code>jitter</code>	a numeric value specifying the amount of jittering. Used only when <code>add</code> contains "jitter".
<code>binwidth</code>	numeric value specifying bin width. use value between 0 and 1 when you have a strong dense dotplot. For example <code>binwidth = 0.2</code> . Used only when <code>add</code> contains "dotplot".
<code>dotsize</code>	as <code>size</code> but applied only to dotplot.
<code>linetype</code>	line type.
<code>show.legend</code>	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.
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "linerrange", "crossbar")</code> . Default value is "pointrange".
<code>ci</code>	the percent range of the confidence interval (default is 0.95).
<code>data</code>	a <code>data.frame</code> to be displayed. If NULL, the default, the data is inherited from the plot data as specified in the call to <code>ggplot</code> .
<code>position</code>	position adjustment, either as a string, or the result of a call to a position adjustment function. Used to adjust position for multiple groups.
<code>p_geom</code>	the geometry of the main plot. Ex: <code>p_geom = "geom_line"</code> . If NULL, the geometry is extracted from <code>p</code> . Used only by <code>ggline()</code> .

**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")
```

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, legend = NULL, common.legend = FALSE)
```

**Arguments**

<code>...</code>	list of plots to be arranged into the grid. The plots can be either ggplot2 plot objects or arbitrary gtables.
<code>plotlist</code>	(optional) list of plots to display.
<code>ncol</code>	(optional) number of columns in the plot grid.
<code>nrow</code>	(optional) number of rows in the plot grid.
<code>labels</code>	(optional) list of labels to be added to the plots. You can also set <code>labels="AUTO"</code> to auto-generate upper-case labels or <code>labels="auto"</code> to auto-generate lower-case labels.
<code>label.x</code>	(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.)
<code>label.y</code>	(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.)
<code>hjust</code>	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.
<code>vjust</code>	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.

<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> .
<code>align</code>	(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".
<code>widths</code>	(optional) numerical vector of relative columns widths. For example, in a two-column grid, <code>widths = c(2, 1)</code> would make the first column twice as wide as the second column.
<code>heights</code>	same as <code>widths</code> but for column heights.
<code>legend</code>	character specifying legend position. Allowed values are one of <code>c("top", "bottom", "left", "right", "none")</code> . To remove the legend use <code>legend = "none"</code> .
<code>common.legend</code>	logical value. Default is FALSE. If TRUE, a common unique legend will be created for arranged plots.

**Value**

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

**Author(s)**

Alboukadel Kassambara <[alboukadel.kassambara@gmail.com](mailto:alboukadel.kassambara@gmail.com)>

**See Also**

[annotate\\_figure\(\)](#)

**Examples**

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

# Create some plots
# ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Box plot
bpx <- 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(bpx, dp, dens, ncol = 2, nrow = 2)
# Use a common legend for multiple plots
ggarrange(bpx, dp, common.legend = TRUE)
```



ggballoonplot

*Ballon 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: <ul style="list-style-type: none"> <li>• <b>a standard contingency table</b> 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.</li> <li>• <b>a stretched contingency table</b>: 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()</li> </ul>
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.



```

# 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 = FALSE)

# 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(), ggtheme = theme_pubr(),
...)

```

## Arguments

<code>data</code>	a data frame
<code>x, y</code>	x and y variables for drawing.
<code>combine</code>	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.
<code>merge</code>	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.
<code>color, fill</code>	outline and fill colors.
<code>palette</code>	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".
<code>size</code>	Numeric value (e.g.: size = 1). change the size of points and outlines.
<code>width</code>	numeric value between 0 and 1 specifying box width.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	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")).
<code>short.panel.labs</code>	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.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.

<code>order</code>	character vector specifying the order of items.
<code>add</code>	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 <code>?desc_statby</code> for more details.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "linrange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linrange", "lower_linrange")</code> . Default value is "pointrange" or "errorbar". Used only when <code>add != "none"</code> and <code>add</code> contains one "mean_*" or "med_*" where "*" = sd, se, ....
<code>label</code>	specify whether to add labels on the bar plot. Allowed values are: <ul style="list-style-type: none"> <li>• <b>logical value</b>: If TRUE, y values is added as labels on the bar plot</li> <li>• <b>character vector</b>: Used as text labels; must be the same length as y.</li> </ul>
<code>lab.col, lab.size</code>	text color and size for labels.
<code>lab.pos</code>	character specifying the position for labels. Allowed values are "out" (for outside) or "in" (for inside). Ignored when <code>lab.vjust != NULL</code> .
<code>lab.vjust</code>	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).
<code>lab.hjust</code>	numeric, horizontal justification of labels.
<code>lab.nb.digits</code>	integer indicating the number of decimal places (round) to be used.
<code>sort.val</code>	a string specifying whether the value should be sorted. Allowed values are "none" (no sorting), "asc" (for ascending) or "desc" (for descending).
<code>sort.by.groups</code>	logical value. If TRUE the data are sorted by groups. Used only when <code>sort.val != "none"</code> .
<code>top</code>	a numeric value specifying the number of top elements to be shown.
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to be passed to <code>ggpar()</code> .

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

# Basic plot with label outside
# ++++++
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,
  width = 0.7, notch = FALSE, 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, 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".



<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide ylab.
<code>bxp.errorbar</code>	logical value. If TRUE, shows error bars of box plots.
<code>bxp.errorbar.width</code>	numeric value specifying the width of box plot error bars. Default is 0.4.
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>linetype</code>	line types.
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>width</code>	numeric value between 0 and 1 specifying box width.
<code>notch</code>	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.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>add</code>	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 <code>?desc_statby</code> for more details.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "linrange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linrange", "lower_linrange")</code> . Default value is "pointrange" or "errorbar". Used only when <code>add != "none"</code> and <code>add</code> contains one "mean_*" or "med_*" where "*" = sd, se, ....
<code>label</code>	the name of the column containing point labels. Can be also a character vector with <code>length = nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .

<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components:           <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variables values, use this: <code>label.select = list(criteria = "`y` &gt; 2 &amp; `y` &lt; 5 &amp; `x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
...	other arguments to be passed to <code>geom_boxplot</code> , <code>ggpar</code> and <code>facet</code> .

## 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,
  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**

<code>data</code>	a data frame
<code>x</code>	variable to be drawn.
<code>y</code>	one of <code>"..density.."</code> or <code>"..count.."</code> .
<code>combine</code>	logical value. Default is <code>FALSE</code> . Used only when <code>y</code> is a vector containing multiple variables to plot. If <code>TRUE</code> , create a multi-panel plot by combining the plot of <code>y</code> variables.
<code>merge</code>	logical or character value. Default is <code>FALSE</code> . Used only when <code>y</code> is a vector containing multiple variables to plot. If <code>TRUE</code> , merge multiple <code>y</code> variables in the same plotting area. Allowed values include also <code>"asis"</code> ( <code>TRUE</code> ) and <code>"flip"</code> . If <code>merge = "flip"</code> , then <code>y</code> variables are used as <code>x</code> tick labels and the <code>x</code> variable is used as grouping variable.
<code>color, fill</code>	density line color and fill color.
<code>palette</code>	the color palette to be used for coloring or filling by groups. Allowed values include <code>"grey"</code> for grey color palettes; brewer palettes e.g. <code>"RdBu"</code> , <code>"Blues"</code> , ...; or custom color palette e.g. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: <code>"npg"</code> , <code>"aaas"</code> , <code>"lancet"</code> , <code>"jco"</code> , <code>"ucscgb"</code> , <code>"uchicago"</code> , <code>"simpsons"</code> and <code>"rickandmorty"</code> .
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>linetype</code>	line type. See <a href="#">show_line_types</a> .
<code>alpha</code>	numeric value specifying fill color transparency. Value should be in <code>[0, 1]</code> , where 0 is full transparency and 1 is no transparency.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying <code>x</code> axis labels. Use <code>xlab = FALSE</code> to hide <code>xlab</code> .

<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide <code>ylab</code> .
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>add</code>	allowed values are one of "mean" or "median" (for adding mean or median line, respectively).
<code>add.params</code>	parameters (color, size, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>rug</code>	logical value. If TRUE, add marginal rug.
<code>label</code>	the name of the column containing point labels. Can be also a character vector with <code>length = nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .
<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variabes values, use this: <code>label.select = list(criteria = "`y` &gt; 2 &amp; `y` &lt; 5 &amp; `x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to <code>geom_density</code> and <code>ggpar</code> .

## 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, "bold", "black"), font.family = "",
  color = "black", fill = "white", palette = NULL, size = NULL,
  ggtheme = theme_pubr(), ...)
```

**Arguments**

<code>data</code>	a data frame
<code>x</code>	variable containing values for drawing.
<code>label</code>	variable specifying the label of each slice.
<code>lab.pos</code>	character specifying the position for labels. Allowed values are "out" (for outside) or "in" (for inside).
<code>lab.adjust</code>	numeric value, used to adjust label position when <code>lab.pos = "in"</code> . Increase or decrease this value to see the effect.
<code>lab.font</code>	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 <code>lab.font= c(4, "bold", "red")</code> .
<code>font.family</code>	character vector specifying font family.
<code>color, fill</code>	outline and fill colors.
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmarty".
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to be passed to <code>ggpar()</code> .

**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"))

```



**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"),
  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**

<code>data</code>	a data frame
<code>x, y</code>	x and y variables for drawing.
<code>group</code>	an optional column name indicating how the elements of x are grouped.
<code>combine</code>	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.
<code>color, size</code>	points color and size.
<code>palette</code>	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 "rickandmarty".
<code>shape</code>	point shape. See <a href="#">show_point_shapes</a> .
<code>dot.size</code>	numeric value specifying the dot size.
<code>sorting</code>	a character vector for sorting into ascending or descending order. Allowed values are one of "descending" and "ascending". Partial match are allowed (e.g. sorting = "desc" or "asc"). Default is "descending".
<code>add</code>	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 <code>?desc_statby</code> for more details.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>x.text.col</code>	logical. If TRUE (default), x axis texts are colored by groups.

<code>rotate</code>	logical value. If TRUE, rotate the graph by setting the plot orientation to horizontal.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide ylab.
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>label</code>	the name of the column containing point labels.
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .
<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variables values, use this: <code>label.select = list(criteria = "\y` &gt; 2 &amp; \y` &lt; 5 &amp; \x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to <code>geom_point</code> and <code>ggpar</code> .

## 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", size = 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

<code>data</code>	a data frame
<code>x</code>	character string containing the name of x variable.
<code>y</code>	character vector containing one or more variables to plot
<code>combine</code>	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.
<code>merge</code>	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.
<code>color</code>	outline color.
<code>fill</code>	fill color.
<code>palette</code>	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".

<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide <code>xlab</code> .
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide <code>ylab</code> .
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>binwidth</code>	numeric value specifying bin width. use value between 0 and 1 when you have a strong dense dotplot. For example <code>binwidth = 0.2</code> .
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>add</code>	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 <code>?desc_statby</code> for more details.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "lin- erange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linerange", "lower_linerange")</code> . Default value is "pointrange" or "errorbar". Used only when <code>add != "none"</code> and <code>add</code> contains one "mean_*" or "med_*" where "*" = sd, se, ....
<code>label</code>	the name of the column containing point labels. Can be also a character vector with <code>length = nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .
<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> </ul> </li> </ul>

	– 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(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ....
...	other arguments to be passed to <a href="#">geom_dotplot</a> , <a href="#">ggpar</a> and <a href="#">facet</a> .

## 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"))

```

---

ggedf

*Empirical cumulative density function*


---

## Description

Empirical Cumulative Density Function (ECDF).

## Usage

```

ggedf(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.

<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
<code>size</code>	line and point size.
<code>linetype</code>	line type. See <a href="#">show_line_types</a> .
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide ylab.
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
...	other arguments to be passed to <a href="#">stat_ecdf</a> and <a href="#">ggpar</a> .

## 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
ggedf(wdata, x = "weight")

# Change colors and linetype by groups ("sex")
# Use custom palette
ggedf(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", 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_mad", "median_range"; see <a href="#">desc_statby</a> for more details.
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.

<code>merge</code>	logical or character value. Default is FALSE. Used only when <code>y</code> is a vector containing multiple variables to plot. If TRUE, merge multiple <code>y</code> variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If <code>merge = "flip"</code> , then <code>y</code> variables are used as <code>x</code> tick labels and the <code>x</code> variable is used as grouping variable.
<code>color, fill</code>	outline and fill colors.
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmarty".
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>width</code>	numeric value between 0 and 1 specifying box width.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying <code>x</code> axis labels. Use <code>xlab = FALSE</code> to hide <code>xlab</code> .
<code>ylab</code>	character vector specifying <code>y</code> axis labels. Use <code>ylab = FALSE</code> to hide <code>ylab</code> .
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>add</code>	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 <code>?desc_statby</code> for more details.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "linrange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linrange", "lower_linrange")</code> . Default value is "pointrange" or "errorbar". Used only when <code>add != "none"</code> and <code>add</code> contains one "mean_*" or "med_*" where "*" = sd, se, ....
<code>ci</code>	the percent range of the confidence interval (default is 0.95).
<code>position</code>	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()`. 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 <a href="#">ggarrange</a> .
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)**

Alboukadel Kassambara <alboukadel.kassambara@gmail.com>

**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,
  color = "black", fill = NA, palette = NULL, size = NULL,
  linetype = "solid", 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

<code>data</code>	a data frame
<code>x</code>	variable to be drawn.
<code>y</code>	one of <code>"..density.."</code> or <code>"..count.."</code> .
<code>combine</code>	logical value. Default is <code>FALSE</code> . Used only when <code>y</code> is a vector containing multiple variables to plot. If <code>TRUE</code> , create a multi-panel plot by combining the plot of <code>y</code> variables.
<code>merge</code>	logical or character value. Default is <code>FALSE</code> . Used only when <code>y</code> is a vector containing multiple variables to plot. If <code>TRUE</code> , merge multiple <code>y</code> variables in the same plotting area. Allowed values include also <code>"asis"</code> ( <code>TRUE</code> ) and <code>"flip"</code> . If <code>merge = "flip"</code> , then <code>y</code> variables are used as <code>x</code> tick labels and the <code>x</code> variable is used as grouping variable.
<code>color, fill</code>	histogram line color and fill color.
<code>palette</code>	the color palette to be used for coloring or filling by groups. Allowed values include <code>"grey"</code> for grey color palettes; brewer palettes e.g. <code>"RdBu"</code> , <code>"Blues"</code> , ...; or custom color palette e.g. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: <code>"npg"</code> , <code>"aaas"</code> , <code>"lancet"</code> , <code>"jco"</code> , <code>"ucscgb"</code> , <code>"uchicago"</code> , <code>"simpsons"</code> and <code>"rickandmorty"</code> .
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>linetype</code>	line type. See <a href="#">show_line_types</a> .
<code>alpha</code>	numeric value specifying fill color transparency. Value should be in <code>[0, 1]</code> , where 0 is full transparency and 1 is no transparency.
<code>bins</code>	Number of bins. Defaults to 30.
<code>binwidth</code>	numeric value specifying bin width. use value between 0 and 1 when you have a strong dense dotplot. For example <code>binwidth = 0.2</code> .
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying <code>x</code> axis labels. Use <code>xlab = FALSE</code> to hide <code>xlab</code> .
<code>ylab</code>	character vector specifying <code>y</code> axis labels. Use <code>ylab = FALSE</code> to hide <code>ylab</code> .
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the <code>"sex"</code> variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	logical value. Default is <code>TRUE</code> . If <code>TRUE</code> , create short labels for panels by omitting variable names; in other words panels will be labelled only by variable grouping levels.
<code>add</code>	allowed values are one of <code>"mean"</code> or <code>"median"</code> (for adding mean or median line, respectively).

<code>add.params</code>	parameters (color, size, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>rug</code>	logical value. If TRUE, add marginal rug.
<code>add_density</code>	logical value. If TRUE, add density curves.
<code>label</code>	the name of the column containing point labels. Can be also a character vector with length = <code>nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .
<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variables values, use this: <code>label.select = list(criteria = "`y` &gt; 2 &amp; `y` &lt; 5 &amp; `x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function. Allowed values include "identity", "stack", "dodge".
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
...	other arguments to be passed to <code>geom_histogram</code> and <code>ggpar</code> .

## 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)

```

---

**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 = 0.5,

```



```

shape = 19, point.size = size, 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,
ggtheme = theme_pubr(), ...)

```

## Arguments

<code>data</code>	a data frame
<code>x, y</code>	x and y variables for drawing.
<code>group</code>	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).
<code>numeric.x.axis</code>	logical. If TRUE, x axis will be treated as numeric. Default is FALSE.
<code>combine</code>	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.
<code>merge</code>	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.
<code>color</code>	line colors.
<code>palette</code>	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 "rickandmarty".
<code>linetype</code>	line type.
<code>plot_type</code>	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".
<code>size</code>	Numeric value (e.g.: size = 1). change the size of points and outlines.
<code>shape</code>	point shapes.
<code>point.size</code>	point size.
<code>point.color</code>	point color.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use xlab = FALSE to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use ylab = FALSE to hide ylab.
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.

<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>add</code>	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 <code>?desc_statby</code> for more details.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "linrange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linrange", "lower_linrange")</code> . Default value is "pointrange" or "errorbar". Used only when <code>add != "none"</code> and <code>add</code> contains one "mean_*" or "med_*" where "*" = sd, se, ....
<code>label</code>	the name of the column containing point labels. Can be also a character vector with length = <code>nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .
<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variabes values, use this: <code>label.select = list(criteria = "\y` &gt; 2 &amp; \y` &lt; 5 &amp; \x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>show.line.label</code>	logical value. If TRUE, shows line labels.

```

ggtheme      function, ggplot2 theme name. Default value is theme_pubr(). 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 log<sub>2</sub> fold changes (on the y-axis) versus the mean expression signal (on the x-axis).

**Usage**

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

**Arguments**

<code>data</code>	an object of class <code>DESeqResults</code> , <code>get_diff</code> , <code>DE_Results</code> , matrix or data frame containing the columns <code>baseMean</code> , <code>log2FoldChange</code> , and <code>padj</code> . Rows are genes. <ul style="list-style-type: none"> <li><code>baseMean</code>: the mean expression of genes in the two groups.</li> <li><code>log2FoldChange</code>: the log<sub>2</sub> fold changes of group 2 compared to group 1</li> <li><code>padj</code>: the adjusted p-value of the used statistical test.</li> </ul>
<code>fdr</code>	Accepted false discovery rate for considering genes as differentially expressed.
<code>fc</code>	the fold change threshold. Only genes with a fold change $\geq fc$ and $padj \leq fdr$ are considered as significantly differentially expressed.
<code>genenames</code>	a character vector of length <code>nrow(data)</code> specifying gene names corresponding to each row. Used for point labels.
<code>detection_call</code>	a numeric vector with length = <code>nrow(data)</code> , specifying if the genes is expressed (value = 1) or not (value = 0). For example <code>detection_call = c(1, 1, 0, 1, 0, 1)</code> . Default is <code>NULL</code> . If <code>detection_call</code> column is available in data, it will be used.
<code>size</code>	points size.
<code>font.label</code>	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 <code>font.label = c(14, "bold", "red")</code> .
<code>label.rectangle</code>	logical value. If <code>TRUE</code> , add rectangle underneath the text, making it easier to read.

<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
<code>top</code>	the number of top genes to be shown on the plot. Use <code>top = 0</code> to hide to gene labels.
<code>select.top.method</code>	methods to be used for selecting top genes. Allowed values include "padj" and "fc" for selecting by adjusted p values or fold changes, respectively.
<code>main</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide ylab.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to <a href="#">ggpar</a> .

### Value

returns 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())
```

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",
  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(), ...)
```

**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.
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.

<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>label</code>	the name of the column containing point labels. Can be also a character vector with <code>length = nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .
<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variables values, use this: <code>label.select = list(criteria = "`y` &gt; 2 &amp; `y` &lt; 5 &amp; `x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to be passed to <code>ggpar()</code> .

## 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.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

<code>p</code>	an object of class <code>ggplot</code> or a list of <code>ggplots</code>
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from <code>ggsci</code> R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty". Can be also a numeric vector of length( <code>groups</code> ); in this case a basic color palette is created using the function <a href="#">palette</a> .
<code>gradient.cols</code>	vector of colors to use for n-colour gradient. Allowed values include brewer and <code>ggsci</code> color palettes.
<code>main, title</code>	plot main title.
<code>submain, subtitle</code>	plot subtitle.
<code>caption</code>	plot caption.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide <code>xlab</code> .
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide <code>ylab</code> .

<code>font.main</code> , <code>font.submain</code> , <code>font.caption</code> , <code>font.x</code> , <code>font.y</code>	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 <code>font.x = c(14, "bold", "red")</code> . Use <code>font.x = 14</code> , to change only font size; or use <code>font.x = "bold"</code> , to change only font face.
<code>font.title</code> , <code>font.subtitle</code>	alias of <code>font.submain</code> and <code>font.submain</code> , respectively.
<code>font.family</code>	character vector specifying font family.
<code>xlim</code> , <code>ylim</code>	a numeric vector of length 2, specifying x and y axis limits (minimum and maximum), respectively. e.g.: <code>ylim = c(0, 50)</code> .
<code>xscale</code> , <code>yscale</code>	x and y axis scale, respectively. Allowed values are one of <code>c("none", "log2", "log10", "sqrt")</code> ; e.g.: <code>yscale="log2"</code> .
<code>format.scale</code>	logical value. If TRUE, axis tick mark labels will be formatted when <code>xscale = "log2"</code> or <code>yscale = "log10"</code> .
<code>legend</code>	character specifying legend position. Allowed values are one of <code>c("top", "bottom", "left", "right", "none")</code> . To remove the legend use <code>legend = "none"</code> . Legend position can be also specified using a numeric vector <code>c(x, y)</code> ; see details section.
<code>legend.title</code>	legend title, e.g.: <code>legend.title = "Species"</code> . Can be also a list, <code>legend.title = list(color = "Species", linetype = "Species", shape = "Species")</code> .
<code>font.legend</code>	legend text font style; e.g.: <code>font.legend = c(10, "plain", "black")</code> .
<code>ticks</code>	logical value. Default is TRUE. If FALSE, hide axis tick marks.
<code>tickslab</code>	logical value. Default is TRUE. If FALSE, hide axis tick labels.
<code>font.tickslab</code> , <code>font.xtickslab</code> , <code>font.ytickslab</code>	Font style (size, face, color) for tick labels, e.g.: <code>c(14, "bold", "red")</code> .
<code>x.text.angle</code> , <code>y.text.angle</code>	Numeric value specifying the rotation angle of x and y axis tick labels, respectively. Default value is NULL. For vertical x axis texts use <code>x.text.angle = 90</code> .
<code>xtickslab.rt</code> , <code>ytickslab.rt</code>	Same as <code>x.text.angle</code> and <code>y.text.angle</code> , respectively. Will be deprecated in the near future.
<code>xticks.by</code> , <code>yticks.by</code>	numeric value controlling x and y axis breaks, respectively. For example, if <code>yticks.by = 5</code> , a tick mark is shown on every 5. Default value is NULL.
<code>rotate</code>	logical value. If TRUE, rotate the graph by setting the plot orientation to horizontal.
<code>orientation</code>	change the orientation of the plot. Allowed values are one of <code>c("vertical", "horizontal", "reverse")</code> . Partial match is allowed.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
...	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 of 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: <code>lineheight = 2</code> .
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.
...	other arguments passed to <code>grob</code> .

**Author(s)**

Alboukadel Kassambara <alboukadel.kassambara@gmail.com>

**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, "bold", "black"), font.family = "",
      color = "black", fill = "white", palette = NULL, size = NULL,
      ggtheme = theme_pubr(), ...)
```

**Arguments**

<code>data</code>	a data frame
<code>x</code>	variable containing values for drawing.
<code>label</code>	variable specifying the label of each slice.
<code>lab.pos</code>	character specifying the position for labels. Allowed values are "out" (for outside) or "in" (for inside).
<code>lab.adjust</code>	numeric value, used to adjust label position when <code>lab.pos = "in"</code> . Increase or decrease this value to see the effect.
<code>lab.font</code>	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 <code>lab.font= c(4, "bold", "red")</code> .
<code>font.family</code>	character vector specifying font family.
<code>color, fill</code>	outline and fill colors.
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to be passed to <code>ggpar()</code> .

**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")

# 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"))

```

**Description**

ggpubr General Arguments Description

**Arguments**

<code>data</code>	a data frame
<code>x</code>	character string containing the name of x variable.
<code>y</code>	character vector containing one or more variables to plot
<code>combine</code>	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.
<code>merge</code>	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.
<code>color</code>	outline color.
<code>fill</code>	fill color.
<code>palette</code>	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 "rickandmarty".
<code>linetype</code>	line types.
<code>size</code>	Numeric value (e.g.: size = 1). change the size of points and outlines.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>add</code>	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.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: add.params = list(color = "red").
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of c("pointrange", "linrange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linrange", "lower_linrange"). Default value is "pointrange" or "errorbar". Used only when add != "none" and add contains one "mean_*" or "med_*" where "*" = sd, se, ....
<code>font.label</code>	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(). Allowed values include ggplot2 official themes: theme_gray(), theme_bw(), theme_minimal(), theme_classic(), theme_void(), ....

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.

<code>add</code>	character vector. Allowed values are one of "none" and "qqline" (for adding qqline).
<code>add.params</code>	parameters (color, size, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>conf.int</code>	logical value. If TRUE, confidence interval is added.
<code>conf.int.level</code>	the confidence level. Default value is 0.95.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide ylab.
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to <a href="#">ggpar</a> .

## 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,
  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, 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.

<code>merge</code>	logical or character value. Default is FALSE. Used only when <code>y</code> is a vector containing multiple variables to plot. If TRUE, merge multiple <code>y</code> variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If <code>merge = "flip"</code> , then <code>y</code> variables are used as <code>x</code> tick labels and the <code>x</code> variable is used as grouping variable.
<code>color, fill</code>	point colors.
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
<code>shape</code>	point shape. See <a href="#">show_point_shapes</a> .
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>point</code>	logical value. If TRUE, show points.
<code>rug</code>	logical value. If TRUE, add marginal rug.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying <code>x</code> axis labels. Use <code>xlab = FALSE</code> to hide <code>xlab</code> .
<code>ylab</code>	character vector specifying <code>y</code> axis labels. Use <code>ylab = FALSE</code> to hide <code>ylab</code> .
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>add</code>	allowed values are one of "none", "reg.line" (for adding linear regression line) or "loess" (for adding local regression fitting).
<code>add.params</code>	parameters (color, size, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>conf.int</code>	logical value. If TRUE, adds confidence interval.
<code>conf.int.level</code>	Level controlling confidence region. Default is 95%. Used only when <code>add != "none"</code> and <code>conf.int = TRUE</code> .
<code>fullrange</code>	should the fit span the full range of the plot, or just the data. Used only when <code>add != "none"</code> .
<code>ellipse</code>	logical value. If TRUE, draws ellipses around points.
<code>ellipse.level</code>	the size of the concentration ellipse in normal probability.
<code>ellipse.type</code>	Character specifying frame type. Possible values are "convex", "confidence" or types supported by <a href="#">stat_ellipse()</a> including one of <code>c("t", "norm", "euclid")</code> for plotting concentration ellipses.

- "convex": plot convex hull of a set o points.
- "confidence": plot confidence ellipses around group mean points as `coord.ellipse()` [in FactoMineR].
- "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.

<code>ellipse.alpha</code>	Alpha for ellipse specifying the transparency level of fill color. Use <code>alpha = 0</code> for no fill color.
<code>ellipse.border.remove</code>	logical value. If TRUE, remove ellipse border lines.
<code>mean.point</code>	logical value. If TRUE, group mean points are added to the plot.
<code>mean.point.size</code>	numeric value specifying the size of mean points.
<code>star.plot</code>	logical value. If TRUE, a star plot is generated.
<code>star.plot.lty</code> , <code>star.plot.lwd</code>	line type and line width (size) for star plot, respectively.
<code>label</code>	the name of the column containing point labels. Can be also a character vector with <code>length = nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = c(14, "bold", "red")</code> . To specify only the size and the style, use <code>font.label = c(14, "plain")</code> .
<code>font.family</code>	character vector specifying font family.
<code>label.select</code>	character vector specifying some labels to show.
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>cor.coef</code>	logical value. If TRUE, correlation coefficient with the p-value will be added to the plot.
<code>cor.coeff.args</code>	a list of arguments to pass to the function <code>stat_cor</code> for customizing the displayed correlation coefficients. For example: <code>cor.coeff.args = list(method = "pearson", label.x.npc = "right", label.y.npc = "top")</code> .
<code>cor.method</code>	method for computing correlation coefficient. Allowed values are one of "pearson", "kendall", or "spearman".
<code>cor.coef.coord</code>	numeric vector, of length 2, specifying the x and y coordinates of the correlation coefficient. Default values are NULL.
<code>cor.coef.size</code>	correlation coefficient text font size.
<code>ggp</code>	a ggplot. If not NULL, points are added to an existing plot.
<code>show.legend.text</code>	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()`. 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.coef.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(), ...)

```

## Arguments

data	a data frame
x	x and y variables for drawing.

<code>y</code>	x and y variables for drawing.
<code>group</code>	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.
<code>color</code>	point colors.
<code>fill</code>	point colors.
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
<code>shape</code>	point shape. See <a href="#">show_point_shapes</a> .
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>linetype</code>	line type ("solid", "dashed", ...)
<code>bins</code>	Number of histogram bins. Defaults to 30. Pick a better value that fit to your data.
<code>margin.plot</code>	the type of the marginal plot. Default is "hist".
<code>margin.params</code>	parameters to be applied to the marginal plots.
<code>margin.ggtheme</code>	the theme of the marginal plot. Default is <code>theme_void()</code> .
<code>margin.space</code>	logical value. If TRUE, adds space between the main plot and the marginal plot.
<code>main.plot.size</code>	the width of the main plot. Default is 2.
<code>margin.plot.size</code>	the width of the marginal plot. Default is 1.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide ylab.
<code>legend</code>	specify the legend position. Allowed values include: "top", "bottom", "left", "right".
<code>ggtheme</code>	the theme to be used for the scatter plot. Default is <code>theme_pubr()</code> .
<code>...</code>	other arguments passed to the function <code>ggscatter()</code> .

## 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()
)

```

---

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), 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.

<code>fill</code>	fill color.
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying x axis labels. Use <code>xlab = FALSE</code> to hide xlab.
<code>ylab</code>	character vector specifying y axis labels. Use <code>ylab = FALSE</code> to hide ylab.
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>shape</code>	point shape
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>add</code>	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 <code>?desc_statby</code> for more details.
<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "linrange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linrange", "lower_linrange")</code> . Default value is "pointrange" or "errorbar". Used only when <code>add != "none"</code> and <code>add</code> contains one "mean_*" or "med_*" where "*" = sd, se, ....
<code>label</code>	the name of the column containing point labels. Can be also a character vector with <code>length = nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .

<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components:           <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variables values, use this: <code>label.select = list(criteria = "`y` &gt; 2 &amp; `y` &lt; 5 &amp; `x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>jitter</code>	the amount of jitter.
<code>position</code>	position adjustment, either as a string, or the result of a call to a position adjustment function. Used to adjust position for multiple groups.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
...	other arguments to be passed to <code>geom_jitter</code> , <code>ggpar</code> and <code>facet</code> .

## 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") )

```

---

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, 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: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– 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).</li> <li>– criteria: to filter, for example, by x and y variables values, use this: label.select = list(criteria = "`y` &gt; 2 &amp; `y` &lt; 5 &amp; `x` %in% c('A', 'B')").</li> </ul> </li> </ul>
repel	a logical value, whether to use ggrepel to avoid overplotting text labels or not.

<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>grouping.vars</code>	grouping variables to sort the data by, when the user wants to display the top n up/down labels.
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>ggp</code>	a ggplot. If not NULL, points are added to an existing plot.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to <a href="#">ggpar</a> .

### 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).

## 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)
```

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

## Arguments

<code>x</code>	a data.frame or matrix.
<code>rows</code>	optional vector to specify row names

<code>cols</code>	optional vector to specify column names
<code>vp</code>	optional viewport
<code>theme</code>	a list, as returned by the function <code>ttheme()</code> , defining the parameters of the table theme. Allowed values include one of <code>ttheme()</code> and <code>ttheme_clean()</code> .
<code>...</code>	extra parameters for text justification, e.g.: <code>hjust</code> and <code>x</code> . Default is "centre" for the body and header, and "right" for the row names. Left justification: <code>hjust = 0, x = 0.1</code> . Right justification: <code>hjust = 1, x = 0.9</code> .
<code>base_style</code>	character string the table style/theme. The available themes are illustrated in the <a href="#">ggtexttable-theme.pdf</a> file. Allowed values include one of <code>c("default", "blank", "classic", "minimal")</code> . Note that, <code>l = "light"</code> ; <code>m = "medium"</code> .
<code>base_size</code>	default font size
<code>base_colour</code>	default font colour
<code>padding</code>	length-2 unit vector specifying the horizontal and vertical padding of text within each cell
<code>colnames.style</code>	a list, as returned by the function <code>colnames_style()</code> , defining the style of the table column names. Considered only when <code>base_size = "default"</code> .
<code>rownames.style</code>	a list, as returned by the function <code>rownames_style()</code> , defining the style of the table row names. Considered only when <code>base_size = "default"</code> .
<code>tbody.style</code>	a list, as returned by the function <code>tbody_style()</code> , defining the style of the table body. Considered only when <code>base_size = "default"</code> .
<code>color, face, size</code>	text font color, face and size, respectively. Allowed values for face include <code>c("plain", "bold", "italic", "bold.italic")</code> .
<code>fill</code>	background color.
<code>linewidth, linecolor</code>	line width and color, respectively.
<code>parse</code>	logical, default behaviour for parsing text as plotmath
<code>tab</code>	an object of class <code>ggtexttable</code> .
<code>row, column</code>	an integer specifying the row and the column numbers for the cell of interest.

**Value**

an object of class `ggplot`.

**Examples**

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

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

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



```

# 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

```

```

# 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,
         size = NULL, width = 1, 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(), ...)

```

## Arguments

<code>data</code>	a data frame
<code>x</code>	character string containing the name of x variable.
<code>y</code>	character vector containing one or more variables to plot

<code>combine</code>	logical value. Default is FALSE. Used only when <code>y</code> is a vector containing multiple variables to plot. If TRUE, create a multi-panel plot by combining the plot of <code>y</code> variables.
<code>merge</code>	logical or character value. Default is FALSE. Used only when <code>y</code> is a vector containing multiple variables to plot. If TRUE, merge multiple <code>y</code> variables in the same plotting area. Allowed values include also "asis" (TRUE) and "flip". If <code>merge = "flip"</code> , then <code>y</code> variables are used as <code>x</code> tick labels and the <code>x</code> variable is used as grouping variable.
<code>color</code>	outline color.
<code>fill</code>	fill color.
<code>palette</code>	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. <code>c("blue", "red")</code> ; and scientific journal palettes from ggsci R package, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".
<code>alpha</code>	color transparency. Values should be between 0 and 1.
<code>title</code>	plot main title.
<code>xlab</code>	character vector specifying <code>x</code> axis labels. Use <code>xlab = FALSE</code> to hide <code>xlab</code> .
<code>ylab</code>	character vector specifying <code>y</code> axis labels. Use <code>ylab = FALSE</code> to hide <code>ylab</code> .
<code>facet.by</code>	character vector, of length 1 or 2, specifying grouping variables for faceting the plot into multiple panels. Should be in the data.
<code>panel.labs</code>	a list of one or two character vectors to modify facet panel labels. For example, <code>panel.labs = list(sex = c("Male", "Female"))</code> specifies the labels for the "sex" variable. For two grouping variables, you can use for example <code>panel.labs = list(sex = c("Male", "Female"), rx = c("Obs", "Lev", "Lev2"))</code> .
<code>short.panel.labs</code>	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.
<code>linetype</code>	line types.
<code>trim</code>	If TRUE (default), trim the tails of the violins to the range of the data. If FALSE, don't trim the tails.
<code>size</code>	Numeric value (e.g.: <code>size = 1</code> ). change the size of points and outlines.
<code>width</code>	violin width.
<code>draw_quantiles</code>	If not (NULL) (default), draw horizontal lines at the given quantiles of the density estimate.
<code>select</code>	character vector specifying which items to display.
<code>remove</code>	character vector specifying which items to remove from the plot.
<code>order</code>	character vector specifying the order of items.
<code>add</code>	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 <code>?desc_statby</code> for more details.

<code>add.params</code>	parameters (color, shape, size, fill, linetype) for the argument 'add'; e.g.: <code>add.params = list(color = "red")</code> .
<code>error.plot</code>	plot type used to visualize error. Allowed values are one of <code>c("pointrange", "lin- erange", "crossbar", "errorbar", "upper_errorbar", "lower_errorbar", "upper_pointrange", "lower_pointrange", "upper_linrange", "lower_linrange")</code> . Default value is <code>"pointrange"</code> or <code>"errorbar"</code> . Used only when <code>add != "none"</code> and <code>add</code> contains one <code>"mean_*</code> " or <code>"med_*</code> " where <code>"*" = sd, se, ....</code>
<code>label</code>	the name of the column containing point labels. Can be also a character vector with <code>length = nrow(data)</code> .
<code>font.label</code>	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 <code>font.label = list(size = 14, face = "bold", color = "red")</code> . To specify only the size and the style, use <code>font.label = list(size = 14, face = "plain")</code> .
<code>label.select</code>	can be of two formats: <ul style="list-style-type: none"> <li>• a character vector specifying some labels to show.</li> <li>• a list containing one or the combination of the following components: <ul style="list-style-type: none"> <li>– <code>top.up</code> and <code>top.down</code>: to display the labels of the top up/down points. For example, <code>label.select = list(top.up = 10, top.down = 4)</code>.</li> <li>– <code>criteria</code>: to filter, for example, by x and y variabes values, use this: <code>label.select = list(criteria = "`y` &gt; 2 &amp; `y` &lt; 5 &amp; `x` %in% c('A', 'B')")</code>.</li> </ul> </li> </ul>
<code>repel</code>	a logical value, whether to use <code>ggrepel</code> to avoid overplotting text labels or not.
<code>label.rectangle</code>	logical value. If TRUE, add rectangle underneath the text, making it easier to read.
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjust- ment function.
<code>ggtheme</code>	function, ggplot2 theme name. Default value is <code>theme_pubr()</code> . Allowed values include ggplot2 official themes: <code>theme_gray()</code> , <code>theme_bw()</code> , <code>theme_minimal()</code> , <code>theme_classic()</code> , <code>theme_void()</code> , ....
<code>...</code>	other arguments to be passed to <code>geom_violin</code> , <code>ggpar</code> and <code>facet</code> .

## 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",
  draw_quantiles = 0.5)

# 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 <code>c("xy", "x", "y")</code> .
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). See <a href="#">show_line_types</a> .

## Examples

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

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

```
p
# Add border
p + grids(linetype = "dashed")
```

---

rotate	<i>Rotate a ggplot Horizontally</i>
--------	-------------------------------------

---

### 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	<i>Rotate Axes Text</i>
------------------	-------------------------

---

### 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 <code>element_text()</code> .

**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: <ul style="list-style-type: none"> <li>• "grid" for both x and y grids</li> <li>• "x.grid" for x axis grids</li> <li>• "y.grid" for y axis grids</li> <li>• "axis" for both x and y axes</li> <li>• "x.axis" for x axis</li> <li>• "y.axis" for y axis</li> <li>• "xlab", or "x.title" for x axis label</li> <li>• "ylab", or "y.title" for y axis label</li> </ul>
--------	--

- "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: <ul style="list-style-type: none"> <li>• <b>Grey color palettes:</b> "grey" or "gray";</li> <li>• <b>RColorBrewer palettes</b>, see <a href="#">brewer.pal</a> and details section. Examples of palette names include: "RdBu", "Blues", "Dark2", "Set2", ...;</li> <li>• <b>Custom color palettes.</b> For example, palette = c("#00AFBB", "#E7B800", "#FC4E07");</li> <li>• <b>ggsci scientific journal palettes</b>, e.g.: "npg", "aaas", "lancet", "jco", "ucscgb", "uchicago", "simpsons" and "rickandmorty".</li> </ul>
...	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\_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, y.position = NULL, xmin = NULL, xmax = NULL,
  step.increase = 0, step.group.by = NULL, tip.length = 0.03,
  size = 0.3, 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, y.position = NULL, xmin = NULL,
  xmax = NULL, step.increase = 0, step.group.by = NULL,
  tip.length = 0.03, size = 0.3, label.size = 3.88, family = "",
  vjust = 0, ...)
```

**Arguments**

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If <code>FALSE</code> (the default), removes missing values with a warning. If <code>TRUE</code> silently removes missing values.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , 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. <code>borders()</code> .
label	character vector with alternative label, if not null test is ignored
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.

<code>step.group.by</code>	a variable name for grouping brackets before adding <code>step.increase</code> . Useful to group bracket by facet panel.
<code>tip.length</code>	numeric vector with the fraction of total height that the bar goes down to indicate the precise column
<code>size</code>	change the width of the lines of the bracket
<code>label.size</code>	change the size of the label text
<code>family</code>	change the font used for the text
<code>vjust</code>	move the text up or down relative to the bracket
<code>...</code>	other arguments passed on to <code>layer</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>color = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired <code>geom/stat</code> .
<code>stat</code>	The statistical transformation to use on the data for this layer, as a string.

### 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)
  )

# 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)
  )

```

---

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 continuous variable.

For 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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	The geometric object to use display the data

position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
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.
inherit.aes	If FALSE, 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. <code>borders()</code> .
type	the type of central tendency measure to be used. Possible values include: "mean", "median", "mode".
...	other arguments to pass to <code>geom_line</code> .

### 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	<i>Plot convex hull of a set of points</i>
------------	--

---

**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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If <code>FALSE</code> , the default, missing values are removed with a warning. If <code>TRUE</code> , missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , 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. <code>borders()</code> .
...	Other arguments passed on to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired geom/stat.

**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 blots, dot plots and stripcharts.

**Usage**

```
stat_compare_means(mapping = NULL, data = NULL, method = NULL,
  paired = FALSE, 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, tip.length = 0.03,
  bracket.size = 0.3, step.increase = 0, symnum.args = list(),
  geom = "text", position = "identity", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, ...)
```

**Arguments**

mapping	Set of aesthetic mappings created by <a href="#">aes()</a> or <a href="#">aes_()</a> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a> . A data frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> 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. <code>~ head(.x, 10)</code> ).
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 <code>t.test</code> and in <code>wilcox.test</code> .
method.args	a list of additional arguments used for the test method. For example one might use <code>method.args = list(alternative = "greater")</code> 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 <code>".all."</code> . 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.
hide.ns	logical value. If TRUE, hide ns symbol when displaying significance levels.
label.sep	a character string to separate the terms. Default is <code>" "</code> , to separate the correlation coefficient and the p.value.
label	character string specifying label type. Allowed values include <code>"p.signif"</code> (shows the significance levels), <code>"p.format"</code> (shows the formatted 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. <ul style="list-style-type: none"> <li>• If numeric, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".</li> <li>• If character, allowed values include: i) one of <code>c('right', 'left', 'center', 'centre', 'middle')</code> for x-axis; ii) and one of <code>c('bottom', 'top', 'center', 'centre', 'middle')</code> for y-axis.</li> </ul>
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.
tip.length	numeric vector with the fraction of total height that the bar goes 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 <code>tip.length = c(0.01, 0.03)</code> . If too short they will be recycled.
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 <code>symnum</code> for symbolic number coding of p-values. For example, <code>symnum.args &lt;- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, 1), symbol = c("****", "***", "**", "*", "ns"))</code> . In other words, we use the following convention for symbols indicating statistical significance: <ul style="list-style-type: none"> <li>• ns: <math>p &gt; 0.05</math></li> <li>• *: <math>p \leq 0.05</math></li> <li>• **: <math>p \leq 0.01</math></li> </ul>

	<ul style="list-style-type: none"> <li>• ***: <math>p \leq 0.001</math></li> <li>• ****: <math>p \leq 0.0001</math></li> </ul>
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
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.
inherit.aes	If FALSE, 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. <code>borders()</code> .
...	other arguments to pass to <code>geom_text</code> or <code>geom_label</code> .

## 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)

# 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 = ..p.signif..),
  method = "t.test", ref.group = "0.5")

# Multiple grouping variables
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
# Box plot faceted 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 = ", ..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 <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes</code> = 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 <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> 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)`).

<code>geom</code>	The geometric object to use display the data
<code>position</code>	Position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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.
<code>inherit.aes</code>	If FALSE, 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. <code>borders()</code> .
<code>level</code>	confidence level used to construct the ellipses. By default, 0.95.
<code>npoint</code>	number of points used to draw the ellipses.
<code>bary</code>	logical value. If TRUE, the coordinates of the ellipse around the barycentre of individuals are calculated.
<code>...</code>	Other arguments passed on to <code>layer()</code> . These are often aesthetics, used to set an aesthetic to a fixed value, like <code>colour = "red"</code> or <code>size = 3</code> . They may also be parameters to the paired <code>geom/stat</code> .

**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	<i>Add Correlation Coefficients with P-values to a Scatter Plot</i>
----------	---

---

### 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",
  label.sep = ", ", label.x.npc = "left", label.y.npc = "top",
  label.x = NULL, label.y = NULL, output.type = "expression",
  geom = "text", position = "identity", na.rm = FALSE,
  show.legend = NA, inherit.aes = TRUE, ...)
```

### Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
method	a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default), "kendall", or "spearman".
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	<p>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.</p> <ul style="list-style-type: none"> <li>• If numeric, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".</li> <li>• If character, allowed values include: i) one of <code>c('right', 'left', 'center', 'centre', 'middle')</code> for x-axis; ii) and one of <code>c('bottom', 'top', 'center', 'centre', 'middle')</code> for y-axis.</li> </ul> <p>If too short they will be recycled.</p>
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.

output.type	character One of "expression", "latex" or "text".
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
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.
inherit.aes	If FALSE, 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. <code>borders()</code> .
...	other arguments to pass to <code>geom_text</code> or <code>geom_label</code> .

### See Also

[ggscatter](#)

### 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)

# Use R2 instead of R
ggscatter(df, x = "wt", y = "mpg", add = "reg.line") +
  stat_cor(
    aes(label = paste(..rr.label.., ..p.label.., sep = "~`,"~")),
    label.x = 3
  )

# 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_mean	<i>Draw group mean points</i>
-----------	-------------------------------

---

**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 <a href="#">aes()</a> or <a href="#">aes_()</a> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If <code>FALSE</code> (the default), removes missing values with a warning. If <code>TRUE</code> silently removes missing values.
show.legend	logical. Should this layer be included in the legends? <code>NA</code> , the default, includes if any aesthetics are mapped. <code>FALSE</code> never includes, and <code>TRUE</code> always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If <code>FALSE</code> , 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. <a href="#">borders()</a> .
...	other arguments to pass to <a href="#">geom_point</a> .

**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 <a href="#">aes()</a> or <a href="#">aes_()</a> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
na.rm	If <code>FALSE</code> (the default), removes missing values with a warning. If <code>TRUE</code> 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.
inherit.aes	If FALSE, 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. <code>borders()</code> .
...	other arguments to pass to <code>geom_line</code> .

**See Also**

[ggdensity](#)

**Examples**

```
# Simple density 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 Manually P-values to a ggplot*

---

**Description**

Add manually p-values to a ggplot, such as box blots, dot plots and stripcharts.

**Usage**

```
stat_pvalue_manual(data, label = NULL, y.position = "y.position",
  xmin = "group1", xmax = "group2", x = NULL, size = 3.88,
  label.size = size, bracket.size = 0.3, color = "black",
  linetype = 1, tip.length = 0.03, remove.bracket = FALSE,
  step.increase = 0, step.group.by = NULL, hide.ns = FALSE,
  vjust = 0, position = "identity", ...)
```

**Arguments**

<code>data</code>	a data frame containing statistical test results. The expected default format should contain the following columns: <code>group1   group2   p   y.position  </code> etc. <code>group1</code> and <code>group2</code> are the groups that have been compared. <code>p</code> is the resulting p-value. <code>y.position</code> is the y coordinates of the p-values in the plot.
<code>label</code>	the column containing the label (e.g.: <code>label = "p"</code> or <code>label = "p.adj"</code> ), where <code>p</code> is the p-value. Can be also an expression that can be formatted by the <a href="#">glue()</a> package. For example, when specifying <code>label = "t-test, p = {p}"</code> , the expression <code>{p}</code> will be replaced by its value.
<code>y.position</code>	column containing the coordinates (in data units) to be used for absolute positioning of the label. Default value is <code>"y.position"</code> . Can be also a numeric vector.
<code>xmin</code>	column containing the position of the left sides of the brackets. Default value is <code>"group1"</code> .
<code>xmax</code>	(optional) column containing the position of the right sides of the brackets. Default value is <code>"group2"</code> . If <code>NULL</code> , the p-values are plotted as a simple text.
<code>x</code>	x position of the p-value. Should be used only when you want plot the p-value as text (without brackets).
<code>size, label.size</code>	size of label text.
<code>bracket.size</code>	Width of the lines of the bracket.
<code>color</code>	text and line color. Can be variable name in the data for coloring by groups.
<code>linetype</code>	linetype. Can be variable name in the data for changing linetype by groups.
<code>tip.length</code>	numeric vector with the fraction of total height that the bar goes down to indicate the precise column. Default is 0.03.
<code>remove.bracket</code>	logical, if <code>TRUE</code> , brackets are removed from the plot. Considered only in the situation, where comparisons are performed against reference group or against <code>"all"</code> .
<code>step.increase</code>	numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.
<code>step.group.by</code>	a variable name for grouping brackets before adding <code>step.increase</code> . Useful to group bracket by facet panel.
<code>hide.ns</code>	logical value. If <code>TRUE</code> , hide <code>ns</code> symbol when displaying significance levels. Filter is done by checking the column <code>p.adj.signif</code> , <code>p.signif</code> , <code>p.adj</code> and <code>p</code> .
<code>vjust</code>	move the text up or down relative to the bracket. Can be also a column name available in the data.
<code>position</code>	position adjustment, either as a string, or the result of a call to a position adjustment function.
<code>...</code>	other arguments passed to the function <code>geom_bracket()</code> or <code>geom_text()</code>

**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 manually p-values 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\_regline\_equation *Add Regression Line Equation and R-Square to a GGLOT.*

---

### 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, output.type = "expression", geom = "text",
  position = "identity", na.rm = FALSE, show.legend = NA,
  inherit.aes = TRUE, ...)
```

### Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> or <code>aes_()</code> . If specified and <code>inherit.aes = TRUE</code> (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 <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
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. <ul style="list-style-type: none"><li>• If numeric, value should be between 0 and 1. Coordinates to be used for positioning the label, expressed in "normalized parent coordinates".</li><li>• If character, allowed values include: i) one of <code>c('right', 'left', 'center', 'centre', 'middle')</code> for x-axis; ii) and one of <code>c('bottom', 'top', 'center', 'centre', 'middle')</code> for y-axis.</li></ul> 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.
output.type	character One of "expression", "latex" or "text".
geom	The geometric object to use display the data

position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
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.
inherit.aes	If FALSE, 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. <code>borders()</code> .
...	other arguments to pass to <code>geom_text</code> or <code>geom_label</code> .

### 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

the source code of the function `stat_regline_equation()` is inspired from the code of the function `stat_poly_eq()` (in `ggpmisc` package).

### 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)

# Grouped 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()` or `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	<p>The data to be displayed in this layer. There are three options:</p> <p>If NULL, the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code>.</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See <code>fortify()</code> for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
geom	The geometric object to use display the data
position	Position adjustment, either as a string, or the result of a call to a position adjustment function.
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.
inherit.aes	If FALSE, 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. <code>borders()</code> .
...	other arguments to pass to <code>geom_segment</code> .

**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))
```

---

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 <a href="#">expression</a> vector. Other objects are coerced by <a href="#">as.graphicsAnnot</a> .
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 alignment and 1 means right 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 <a href="#">textGrob</a> .

**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.

## 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()
```

## 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.

## 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	<i>Create a ggplot with Transparent Background</i>
-------------------	--

---

## 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`, 3  
`aes()`, 101, 103, 105, 106, 109, 111, 113, 114, 118, 120  
`aes_()`, 101, 103, 105, 106, 109, 111, 113, 114, 118, 120  
`annotate_figure`, 5, 24  
`anova`, 11  
`arrangeGrob`, 7  
`as.graphicsAnnot`, 122  
`as_ggplot`, 7  
`axis_scale`, 7  
  
`background_image`, 8  
`bgcolor`, 9  
`border`, 9, 10  
`borders()`, 101, 104, 105, 108, 110, 112, 113, 115, 119, 121  
`brewer.pal`, 20, 99  
  
`change_palette` (`set_palette`), 98  
`clean_theme` (`theme_pubr`), 123  
`colnames_style` (`ggtexttable`), 87  
`color_palette` (`set_palette`), 98  
`compare_means`, 10, 108  
`coord.ellipse`, 77  
`coord_fixed`, 77  
`coord_flip`, 96  
  
`desc_statby`, 13, 49  
`diff_express`, 14  
`drawDetails.splitText` (`ggparagraph`), 68  
  
`element_text`, 16, 17, 97  
`expression`, 122  
  
`facet`, 15, 34, 46, 83, 92  
`facet_wrap`, 15  
`fill_palette` (`set_palette`), 98  
`font`, 16  
`fortify()`, 101, 103, 105, 106, 109, 111, 113, 114, 118, 121  
  
`gene_citation`, 17  
`geom_boxplot`, 34  
`geom_bracket` (`stat_bracket`), 100  
`geom_density`, 37  
`geom_dotplot`, 46  
`geom_exec`, 18  
`geom_histogram`, 55  
`geom_jitter`, 83  
`geom_label`, 108, 112, 119  
`geom_line`, 104, 115  
`geom_point`, 42, 78, 113  
`geom_segment`, 121  
`geom_text`, 108, 112, 119  
`geom_violin`, 92  
`get_legend`, 19  
`get_palette`, 20, 99  
`ggadd`, 21  
`ggarrange`, 5, 6, 23, 52  
`ggballoonplot`, 25  
`ggbarplot`, 27, 59  
`ggboxplot`, 32, 46, 83  
`ggdensity`, 36, 55, 104, 115  
`ggdonutchart`, 38  
`ggdotchart`, 40  
`ggdotplot`, 34, 44, 83  
`ggecdf`, 47  
`ggerrorplot`, 49  
`ggexport`, 52  
`gghistogram`, 38, 53  
`ggline`, 22, 30, 51, 56, 70, 100  
`ggmaplot`, 61  
`ggpaired`, 63  
`ggpar`, 26, 30, 34, 37–39, 42, 43, 46, 48, 51, 55, 59, 62, 64, 65, 70, 74, 78, 83, 86, 92, 93, 100, 105  
`ggparagraph`, 68  
`ggpie`, 39, 69  
`ggplot`, 4, 22  
`ggplot()`, 101, 103, 105, 106, 109, 111, 113,

[114, 118, 121](#)  
 ggpubr\_args, [71](#)  
 ggqqplot, [73](#)  
 ggscatter, [75, 80, 105, 112, 113, 119, 121](#)  
 ggscatterhist, [79](#)  
 ggstripchart, [34, 46, 81](#)  
 ggtext, [85](#)  
 ggtexttable, [87](#)  
 ggviolin, [34, 46, 83, 90](#)  
 glue, [116](#)  
 gradient\_color, [94](#)  
 gradient\_fill(gradient\_color), [94](#)  
 grid.arrange, [5, 7](#)  
 grids, [95](#)  
 grob, [69](#)  
  
 kruskal.test, [11](#)  
  
 labs\_pubr(theme\_pubr), [123](#)  
 layer, [102](#)  
 layer(), [105, 110](#)  
 lm, [118](#)  
  
 mean\_ci(add\_summary), [3](#)  
 mean\_range(add\_summary), [3](#)  
 mean\_sd(add\_summary), [3](#)  
 mean\_se, [5](#)  
 mean\_se\_(add\_summary), [3](#)  
 median\_iqr(add\_summary), [3](#)  
 median\_mad(add\_summary), [3](#)  
 median\_range(add\_summary), [3](#)  
  
 p.adjust, [11](#)  
 palette, [65, 94](#)  
 plot\_grid, [5, 23](#)  
  
 rotate, [96](#)  
 rotate\_axis\_text, [96](#)  
 rotate\_x\_text(rotate\_axis\_text), [96](#)  
 rotate\_y\_text(rotate\_axis\_text), [96](#)  
 rownames\_style(ggtexttable), [87](#)  
 rremove, [97](#)  
  
 set\_palette, [94, 98](#)  
 show\_line\_types, [10, 36, 48, 54, 95, 99](#)  
 show\_point\_shapes, [4, 22, 41, 76, 80, 100](#)  
 stat\_bracket, [100](#)  
 stat\_central\_tendency, [103](#)  
 stat\_chull, [105, 113](#)  
 stat\_compare\_means, [106, 116](#)  
  
 stat\_conf\_ellipse, [78, 109, 110, 113](#)  
 stat\_cor, [77, 78, 111](#)  
 stat\_ecdf, [48](#)  
 stat\_ellipse, [76](#)  
 stat\_mean, [113](#)  
 stat\_overlay\_normal\_density, [114](#)  
 stat\_pvalue\_manual, [115](#)  
 stat\_regline\_equation, [118](#)  
 stat\_stars, [78, 120](#)  
 symnum, [11, 107](#)  
  
 t.test, [11, 107](#)  
 table\_cell\_bg(ggtexttable), [87](#)  
 table\_cell\_font(ggtexttable), [87](#)  
 tbody\_style(ggtexttable), [87](#)  
 text\_grob, [121](#)  
 textGrob, [121, 122](#)  
 theme\_classic2(theme\_pubr), [123](#)  
 theme\_cleveland(ggdotchart), [40](#)  
 theme\_pubclean(theme\_pubr), [123](#)  
 theme\_pubr, [80, 123, 124](#)  
 theme\_transparent, [124](#)  
 theme\_void, [80](#)  
 ttheme(ggtexttable), [87](#)  
  
 wilcox.test, [11, 107](#)  
  
 xscale(axis\_scale), [7](#)  
  
 yscale(axis\_scale), [7](#)