Package ‘tidyplots’

July 2, 2025

Title Tidy Plots for Scientific Papers

Version 0.3.1

Description The goal of 'tidyplots' is to streamline the creation of publication-ready plots for scientific papers. It allows to gradually add, remove and adjust plot components using a consistent and intuitive syntax.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports cli, dplyr, forcats, ggbeeswarm, ggplot2 (>= 3.5.0), ggpubr, ggrastr, ggrepel, glue, Hmisc, htmltools, lifecycle, patchwork (>= 1.2.0), purrr, rlang, scales, stringr, tidyr, tidyselect

Depends R (>= 4.1.0)

LazyData true

URL `https://github.com/jbengler/tidyplots`, `https://jbengler.github.io/tidyplots/`

BugReports `https://github.com/jbengler/tidyplots/issues`

Suggests knitr, rmarkdown, testthat (>= 3.0.0), vdiffr

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Jan Broder Engler [aut, cre, cph] (ORCID: <`https://orcid.org/0000-0002-3169-2076`>)

Maintainer Jan Broder Engler <`broder.engler@gmail.com`>

Repository CRAN

Date/Publication 2025-07-02 20:30:02 UTC

Contents

`add` *Add ggplot2 code to a tidyplot*

Description

Add ggplot2 code to a tidyplot

Usage

```
add()
```

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add(ggplot2::geom_point())
```

`add_annotation_text` *Add annotation*

Description

Add annotation

Usage

```
add_annotation_text(plot, text, x, y, fontsize = 7, ...)

add_annotation_rectangle(
  plot,
  xmin,
  xmax,
  ymin,
  ymax,
  fill = "#000000",
  color = NA,
  alpha = 0.1,
  ...
)

add_annotation_line(plot, x, xend, y, yend, color = "#000000", ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`text`	String for annotation text.
`x`, `xmin`, `xmax`, `xend`, `y`, `ymin`, `ymax`, `yend`	Coordinates for the annotation.
`fontsize`	Font size in points. Defaults to `7`.
`...`	Arguments passed on to `ggplot2::annotate()`.
`fill`	A hex color for the fill color. For example, `"#FFFFFF"` for white.
`color`	A hex color for the stroke color. For example, `"#FFFFFF"` for white.
`alpha`	A `number` between `0` and `1` for the opacity of an object. A value of `0` is completely transparent, `1` is completely opaque.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_boxplot() |>
  add_annotation_text("Look here!", x = 2, y = 25)

eu_countries |>
  tidyplot(x = area, y = population) |>
  add_data_points() |>
  add_annotation_rectangle(xmin = 2.5e5, xmax = Inf, ymin = 42, ymax = Inf)

eu_countries |>
  tidyplot(x = area, y = population) |>
  add_data_points() |>
  add_annotation_rectangle(xmin = 2.5e5, xmax = 6e5, ymin = 42, ymax = 90,
                           color = "#E69F00", fill = NA)

eu_countries |>
  tidyplot(x = area, y = population) |>
  add_data_points() |>
  add_annotation_line(x = 0, xend = Inf, y = 0, yend = Inf)
```

`add_areastack_absolute` *Add area stack*

Description

Add area stack

Usage

```
add_areastack_absolute(
  plot,
  linewidth = 0.25,
  alpha = 0.4,
  reverse = FALSE,
  replace_na = FALSE,
  ...
)

add_areastack_relative(
  plot,
  linewidth = 0.25,
  alpha = 0.4,
  reverse = FALSE,
  replace_na = FALSE,
```

```
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`alpha`	A `number` between `0` and `1` for the opacity of an object. A value of `0` is completely transparent, `1` is completely opaque.
`reverse`	Whether the order should be reversed or not. Defaults to `FALSE`, meaning not reversed.
`replace_na`	Whether to replace `count = NA` with `count = 0`.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
# for a `count` provide `x` and `color`
# `count` of the data points in each `energy_type` category
energy |>
  tidyplot(x = year, color = energy_type) |>
  add_areastack_absolute()

energy |>
  tidyplot(x = year, color = energy_type) |>
  add_areastack_relative()

# for a `sum` provide `x`, `y` and `color`
# `sum` of `energy` in each `energy_type` category
energy |>
  tidyplot(x = year, y = energy, color = energy_type) |>
  add_areastack_absolute()

energy |>
  tidyplot(x = year, y = energy, color = energy_type) |>
  add_areastack_relative()

# Flip x and y-axis
energy |>
  tidyplot(x = energy, y = year, color = energy_type) |>
  add_areastack_absolute(orientation = "y")

energy |>
  tidyplot(x = energy, y = year, color = energy_type) |>
  add_areastack_relative(orientation = "y")
```

`add_barstack_absolute` *Add bar stack*

Description

Add bar stack

Usage

```
add_barstack_absolute(plot, width = 0.8, reverse = FALSE, ...)

add_barstack_relative(plot, width = 0.8, reverse = FALSE, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`width`	Width of the bar.
`reverse`	Whether the order should be reversed or not. Defaults to `FALSE`, meaning not reversed.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
# for a `count` only provide `color`
# `count` of the data points in each `energy_type` category
energy |>
  tidyplot(color = energy_type) |>
  add_barstack_absolute()

energy |>
  tidyplot(color = energy_type) |>
  add_barstack_relative()

# for a `sum` provide `color` and `y`
# `sum` of `energy` in each `energy_type` category
energy |>
  tidyplot(y = energy, color = energy_type) |>
  add_barstack_absolute()

energy |>
  tidyplot(y = energy, color = energy_type) |>
  add_barstack_relative()

# Include variable on second axis
```

```
energy |>
  tidyplot(x = year, y = energy, color = energy_type) |>
  add_barstack_absolute()

energy |>
  tidyplot(x = year, y = energy, color = energy_type) |>
  add_barstack_relative()

# Flip x and y-axis
energy |>
  tidyplot(x = energy, y = year, color = energy_type) |>
  add_barstack_absolute(orientation = "y")

energy |>
  tidyplot(x = energy, y = year, color = energy_type) |>
  add_barstack_relative(orientation = "y")
```

`add_boxplot` *Add boxplot*

Description

Add boxplot

Usage

```
add_boxplot(
  plot,
  dodge_width = NULL,
  alpha = 0.3,
  saturation = 1,
  show_whiskers = TRUE,
  show_outliers = TRUE,
  box_width = 0.6,
  whiskers_width = 0.8,
  outlier.size = 0.5,
  coef = 1.5,
  outlier.shape = 19,
  outlier.alpha = 1,
  linewidth = 0.25,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.

`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`alpha`	A `number` between `0` and `1` for the opacity of an object. A value of `0` is completely transparent, `1` is completely opaque.
`saturation`	A `number` between `0` and `1` for the color saturation of an object. A value of `0` is completely desaturated (white), `1` is the original color.
`show_whiskers`	Whether to show boxplot whiskers. Defaults to `TRUE`.
`show_outliers`	Whether to show outliers. Defaults to `TRUE`.
`box_width`	Width of the boxplot. Defaults to `0.6`.
`whiskers_width`	Width of the whiskers. Defaults to `0.8`.
`outlier.size`	Size of the outliers. Defaults to `0.5`.
`coef`	Length of the whiskers as multiple of IQR. Defaults to 1.5.
`outlier.shape`	Shape of the outliers. Defaults to `19`.
`outlier.alpha`	Opacity of the outliers. Defaults to `1`.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_boxplot()

# Changing arguments:
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_boxplot(show_whiskers = FALSE)

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_boxplot(show_outliers = FALSE)

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_boxplot(box_width = 0.2)

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_boxplot(whiskers_width = 0.2)
```

`add_count_bar` *Add count*

Description

Add count

Usage

```
add_count_bar(
  plot,
  dodge_width = NULL,
  width = 0.6,
  saturation = 1,
  preserve = "total",
  ...
)

add_count_dash(
  plot,
  dodge_width = NULL,
  width = 0.6,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_count_dot(plot, dodge_width = NULL, size = 2, preserve = "total", ...)

add_count_value(
  plot,
  dodge_width = NULL,
  accuracy = 0.1,
  scale_cut = NULL,
  fontsize = 7,
  extra_padding = 0.15,
  vjust = NULL,
  hjust = NULL,
  preserve = "total",
  ...
)

add_count_line(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
```

```
  preserve = "total",
  ...
)

add_count_area(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`width`	Width of the bar.
`saturation`	A `number` between `0` and `1` for the color saturation of an object. A value of `0` is completely desaturated (white), `1` is the original color.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`...`	Arguments passed on to the `geom` function.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`size`	A `number` representing the size of the plot symbol. Typical values range between `1` and `3`.
`accuracy`	A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.
`scale_cut`	Scale cut function to be applied. See `scales::cut_short_scale()` and friends.
`fontsize`	Font size in points. Defaults to `7`.
`extra_padding`	Extra padding to create space for the value label.
`vjust`	Vertical position adjustment of the value label.
`hjust`	Horizontal position adjustment of the value label.
`group`	Variable in the dataset to be used for grouping.

Value

A `tidyplot` object.

Examples

```
dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_bar()

dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_dash()

dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_dot()

dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_value()

dinosaurs |>
  tidyplot(x = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_line()

dinosaurs |>
  tidyplot(x = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_area()

# Combination
dinosaurs |>
  tidyplot(x = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_bar(alpha = 0.4) |>
  add_count_dash() |>
  add_count_dot() |>
  add_count_value() |>
  add_count_line()

# Changing arguments: alpha
# Makes objects transparent
dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  theme_minimal_y() |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_bar(alpha = 0.4)

# Changing arguments: saturation
# Reduces fill color saturation without making the object transparent
dinosaurs |>
```

```
  tidyplot(x = time_lived, color = time_lived) |>
  theme_minimal_y() |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_bar(saturation = 0.3)

# Changing arguments: accuracy
dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_value(accuracy = 1)

# Changing arguments: fontsize
dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_value(fontsize = 10)

# Changing arguments: color
dinosaurs |>
  tidyplot(x = time_lived, color = time_lived) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_count_value(color = "black")
```

`add_curve_fit` *Add curve fit*

Description

Add curve fit

Usage

```
add_curve_fit(
  plot,
  dodge_width = NULL,
  method = "loess",
  linewidth = 0.25,
  alpha = 0.4,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.

`method`	Smoothing method (function) to use, accepts either `NULL` or a character vector, e.g. `"lm"`, `"glm"`, `"gam"`, `"loess"` or a function, e.g. `MASS::rlm` or `mgcv::gam`, `stats::lm`, or `stats::loess`. `"auto"` is also accepted for backwards compatibility. It is equivalent to `NULL`. For `method = NULL` the smoothing method is chosen based on the size of the largest group (across all panels). `stats::loess()` is used for less than 1,000 observations; otherwise `mgcv::gam()` is used with `formula = y ~ s(x, bs = "cs")` with `method = "REML"`. Somewhat anecdotally, `loess` gives a better appearance, but is $O(N^2)$ in memory, so does not work for larger datasets. If you have fewer than 1,000 observations but want to use the same `gam()` model that `method = NULL` would use, then set `method = "gam", formula = y ~ s(x, bs = "cs")`.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`alpha`	A `number` between `0` and `1` for the opacity of an object. A value of `0` is completely transparent, `1` is completely opaque.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`...`	Arguments passed on to `ggplot2::geom_smooth()`.

Value

A `tidyplot` object.

Examples

```
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_curve_fit()

# Changing arguments
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_curve_fit(linewidth = 1)

time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_curve_fit(alpha = 0.8)

# Remove confidence interval
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_curve_fit(se = FALSE)
```

`add_data_labels` *Add data labels*

Description

Add data labels

Usage

```
add_data_labels(
  plot,
  label,
  data = all_rows(),
  fontsize = 7,
  dodge_width = NULL,
  jitter_width = 0,
  jitter_height = 0,
  preserve = "total",
  background = FALSE,
  background_color = "#FFFFFF",
  background_alpha = 0.6,
  label_position = c("below", "above", "left", "right", "center"),
  ...
)

add_data_labels_repel(
  plot,
  label,
  data = all_rows(),
  fontsize = 7,
  dodge_width = NULL,
  jitter_width = 0,
  jitter_height = 0,
  preserve = "total",
  segment.size = 0.2,
  box.padding = 0.2,
  max.overlaps = Inf,
  background = FALSE,
  background_color = "#FFFFFF",
  background_alpha = 0.6,
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`label`	Variable in the dataset to be used for the text label.

`data`	The data to be displayed in this layer. There are three options: • If `all_rows()` (the default) the complete dataset is displayed. • A `function` to subset the plot data. See `filter_rows()` and friends. • A `data.frame` to override the plot data.
`fontsize`	Font size in points. Defaults to `7`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`jitter_width`	Amount of random noise to be added to the horizontal position of the of the data points. This can be useful to deal with overplotting. Typical values range between `0` and `1`.
`jitter_height`	Amount of random noise to be added to the vertical position of the of the data points. This can be useful to deal with overplotting. Typical values range between `0` and `1`.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`background`	Whether to include semitransparent background box behind the labels to improve legibility. Defaults to `FALSE`.
`background_color`	Hex color of the background box. Defaults to `"#FFFFFF"` for white.
`background_alpha`	Opacity of the background box. Defaults to `0.6`.
`label_position`	Position of the label in relation to the data point. Can be one of `c("below", "above", "left", "right", "center")`.
`...`	Arguments passed on to the `geom` function.
`segment.size`	Thickness of the line connecting the label with the data point. Defaults to `0.2`.
`box.padding`	Amount of padding around bounding box, as unit or number. Defaults to 0.25. (Default unit is lines, but other units can be specified by passing `unit(x, "units")`).
`max.overlaps`	Exclude text labels when they overlap too many other things. For each text label, we count how many other text labels or other data points it overlaps, and exclude the text label if it has too many overlaps. Defaults to 10.

Details

- `add_data_labels_repel()` uses `ggrepel::geom_text_repel()`. Check there and in ggrepel examples for additional arguments.
- `add_data_labels()` and `add_data_labels_repel()` support data subsetting. See Advanced plotting.

Value

A `tidyplot` object.

Examples

```
# Create plot and increase padding to make more space for labels
p <-
  animals |>
  dplyr::slice_head(n = 5) |>
  tidyplot(x = weight, y = speed) |>
  theme_ggplot2() |>
  add_data_points() |>
  adjust_padding(all = 0.3)

# Default label position is `below` the data point
p |> add_data_labels(label = animal)

# Alternative label positions
p |> add_data_labels(label = animal, label_position = "above")

p |> add_data_labels(label = animal, label_position = "right")

p |> add_data_labels(label = animal, label_position = "left")

# Include white background box
p |> add_data_labels(label = animal, background = TRUE)

p |> add_data_labels(label = animal, background = TRUE,
  background_color = "pink")

# Black labels
p |> add_data_labels(label = animal, color = "black")

# Use repelling data labels
p |> add_data_labels_repel(label = animal, color = "black")

p |> add_data_labels_repel(label = animal, color = "black",
  background = TRUE)

p |> add_data_labels_repel(label = animal, color = "black",
  background = TRUE, min.segment.length = 0)
```

`add_data_points` *Add data points*

Description

Add data points

Usage

```
add_data_points(
```

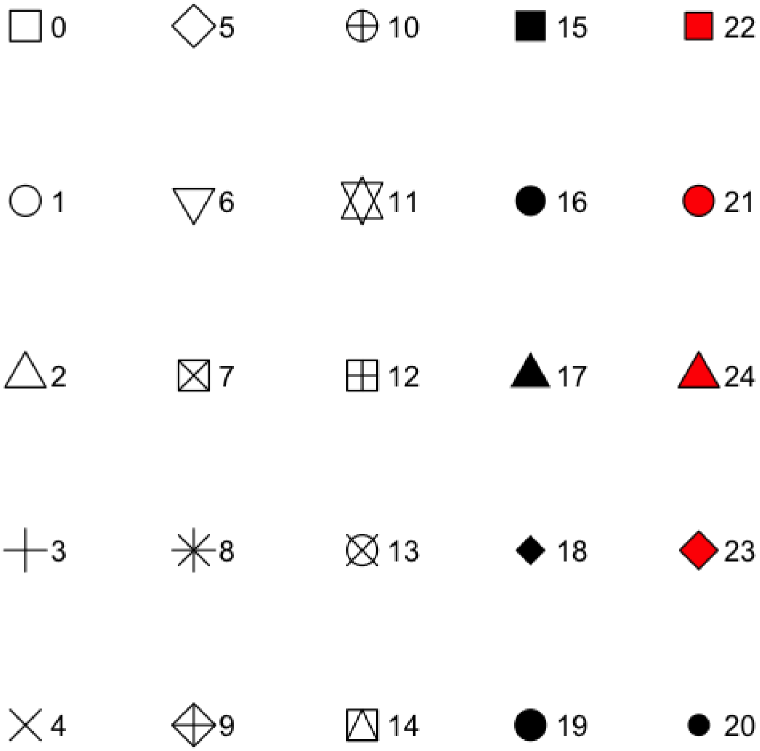
```
  plot,
  data = all_rows(),
  shape = 19,
  size = 1,
  white_border = FALSE,
  dodge_width = NULL,
  preserve = "total",
  rasterize = FALSE,
  rasterize_dpi = 300,
  ...
)

add_data_points_jitter(
  plot,
  data = all_rows(),
  shape = 19,
  size = 1,
  white_border = FALSE,
  dodge_width = NULL,
  jitter_width = 0.2,
  jitter_height = 0,
  preserve = "total",
  rasterize = FALSE,
  rasterize_dpi = 300,
  ...
)

add_data_points_beeswarm(
  plot,
  data = all_rows(),
  shape = 19,
  size = 1,
  white_border = FALSE,
  cex = 3,
  corral = "wrap",
  corral.width = 0.5,
  dodge_width = NULL,
  preserve = "total",
  rasterize = FALSE,
  rasterize_dpi = 300,
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`data`	The data to be displayed in this layer. There are three options: • If `all_rows()` (the default) the complete dataset is displayed.

	- A `function` to subset the plot data. See `filter_rows()` and friends. - A `data.frame` to override the plot data.
`shape`	An `integer` between `0` and `24`, representing the shape of the plot symbol. 
`size`	A `number` representing the size of the plot symbol. Typical values range between `1` and `3`.
`white_border`	Whether to include a white border around data points. Defaults to `FALSE`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`rasterize`	If `FALSE` (the default) the layer will be constructed of vector shapes. If `TRUE` the layer will be rasterized to a pixel image. This can be useful when plotting many individual objects (1,000 or more) compromises the performance of the generated PDF file.
`rasterize_dpi`	The resolution in dots per inch (dpi) used for rastering the layer if `rasterize` is `TRUE`. The default is `300` dpi.
`...`	Arguments passed on to the `geom` function.
`jitter_width`	Amount of random noise to be added to the horizontal position of the of the data points. This can be useful to deal with overplotting. Typical values range between `0` and `1`.

`jitter_height`	Amount of random noise to be added to the vertical position of the of the data points. This can be useful to deal with overplotting. Typical values range between `0` and `1`.
`cex`	Scaling for adjusting point spacing (see `beeswarm::swarmx()`). Values between 1 (default) and 3 tend to work best.
`corral`	`string`. Method used to adjust points that would be placed to wide horizontally, default is `"none"`. See details below.
`corral.width`	`numeric`. Width of the corral, default is `0.9`.

Details

- `add_data_points_beeswarm()` is based on `ggbeeswarm::geom_beeswarm()`. Check there for additional arguments.
- `add_data_points()` and friends support rasterization. See examples and Advanced plotting.
- `add_data_points()` and friends support data subsetting. See examples and Advanced plotting.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_jitter()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm()

# Changing arguments
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_jitter(jitter_width = 1)

animals |>
  tidyplot(x = weight, y = size) |>
  add_data_points(white_border = TRUE)

animals |>
  tidyplot(x = weight, y = size) |>
  add_data_points(alpha = 0.4)

# Rasterization
animals |>
```

```
  tidyplot(x = weight, y = size) |>
  add_data_points(rasterize = TRUE, rasterize_dpi = 50)

# Data subsetting
animals |>
  tidyplot(x = weight, y = size) |>
  add_data_points() |>
  add_data_points(data = filter_rows(size > 300), color = "red")
```

`add_ellipse` *Add ellipse*

Description

Add ellipse

Usage

```
add_ellipse(plot, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`...`	Arguments passed on to `ggplot2::stat_ellipse()`.

Value

A `tidyplot` object.

Examples

```
pca |>
  tidyplot(x = pc1, y = pc2, color = group) |>
  add_data_points() |>
  add_ellipse()

pca |>
  tidyplot(x = pc1, y = pc2, color = group) |>
  add_data_points() |>
  add_ellipse(level = 0.75)

pca |>
  tidyplot(x = pc1, y = pc2, color = group) |>
  add_data_points() |>
  add_ellipse(type = "norm")
```

`add_heatmap` *Add heatmap*

Description

Add heatmap

Usage

```
add_heatmap(
  plot,
  scale = c("none", "row", "column"),
  rotate_labels = 90,
  rasterize = FALSE,
  rasterize_dpi = 300,
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`scale`	Whether to compute row z scores for `"row"` or `"column"`. Defaults to `"none"`.
`rotate_labels`	Degree to rotate the x-axis labels. Defaults to `90`.
`rasterize`	If `FALSE` (the default) the layer will be constructed of vector shapes. If `TRUE` the layer will be rasterized to a pixel image. This can be useful when plotting many individual objects (1,000 or more) compromises the performance of the generated PDF file.
`rasterize_dpi`	The resolution in dots per inch (dpi) used for rastering the layer if `rasterize` is `TRUE`. The default is `300` dpi.
`...`	Arguments passed on to the `geom` function.

Details

- `add_heatmap()` supports rasterization. See examples and Advanced plotting.

Value

A `tidyplot` object.

Examples

```
climate |>
  tidyplot(x = month, y = year, color = max_temperature) |>
  add_heatmap()

# Calculate row-wise z score
```

```
climate |>
  tidyplot(x = month, y = year, color = max_temperature) |>
  add_heatmap(scale = "row")

# Calculate column-wise z score
climate |>
  tidyplot(x = month, y = year, color = max_temperature) |>
  add_heatmap(scale = "column")

# Rasterize heatmap
climate |>
  tidyplot(x = month, y = year, color = max_temperature) |>
  add_heatmap(rasterize = TRUE, rasterize_dpi = 20)
```

`add_histogram` *Add histogram*

Description

Add histogram

Usage

```
add_histogram(plot, binwidth = NULL, bins = NULL, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`binwidth`	The width of the bins. Can be specified as a numeric value or as a function that takes x after scale transformation as input and returns a single numeric value. When specifying a function along with a grouping structure, the function will be called once per group. The default is to use the number of bins in `bins`, covering the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data. The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.
`bins`	Number of bins. Overridden by `binwidth`. Defaults to 30.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
energy |>
  tidyplot(x = energy) |>
  add_histogram()

energy |>
  tidyplot(x = energy, color = energy_type) |>
  add_histogram()
```

`add_line` *Add line or area*

Description

`add_line()` and `add_area()` connect individual data points, which is rarely needed. In most cases, you are probably looking for `add_sum_line()`, `add_mean_line()`, `add_sum_area()` or `add_mean_area()`.

Usage

```
add_line(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_area(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  alpha = 0.4,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`group`	Variable in the dataset to be used for grouping.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.

`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`...`	Arguments passed on to the `geom` function.
`alpha`	A `number` between `0` and `1` for the opacity of an object. A value of `0` is completely transparent, `1` is completely opaque.

Value

A `tidyplot` object.

Examples

```
# Paired data points
study |>
  tidyplot(x = treatment, y = score, color = group) |>
  reorder_x_axis_labels("A", "C", "B", "D") |>
  add_data_points() |>
  add_line(group = participant, color = "grey")

study |>
  tidyplot(x = treatment, y = score) |>
  reorder_x_axis_labels("A", "C", "B", "D") |>
  add_data_points() |>
  add_area(group = participant)
```

`add_mean_bar` *Add mean*

Description

Add mean

Usage

```
add_mean_bar(
  plot,
  dodge_width = NULL,
  width = 0.6,
  saturation = 1,
  preserve = "total",
  ...
)

add_mean_dash(
  plot,
  dodge_width = NULL,
  width = 0.6,
```

```
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_mean_dot(plot, dodge_width = NULL, size = 2, preserve = "total", ...)

add_mean_value(
  plot,
  dodge_width = NULL,
  accuracy = 0.1,
  scale_cut = NULL,
  fontsize = 7,
  extra_padding = 0.15,
  vjust = NULL,
  hjust = NULL,
  preserve = "total",
  ...
)

add_mean_line(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_mean_area(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`width`	Width of the bar.
`saturation`	A `number` between `0` and `1` for the color saturation of an object. A value of `0` is completely desaturated (white), `1` is the original color.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?

`...`	Arguments passed on to the `geom` function.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`size`	A `number` representing the size of the plot symbol. Typical values range between `1` and `3`.
`accuracy`	A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.
`scale_cut`	Scale cut function to be applied. See `scales::cut_short_scale()` and friends.
`fontsize`	Font size in points. Defaults to `7`.
`extra_padding`	Extra padding to create space for the value label.
`vjust`	Vertical position adjustment of the value label.
`hjust`	Horizontal position adjustment of the value label.
`group`	Variable in the dataset to be used for grouping.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_dash()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_dot()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_value()

study |>
  tidyplot(x = treatment, y = score) |>
  add_mean_line()

study |>
  tidyplot(x = treatment, y = score) |>
  add_mean_area()

# Combination
study |>
  tidyplot(x = treatment, y = score) |>
```

```
    add_mean_bar(alpha = 0.4) |>
    add_mean_dash() |>
    add_mean_dot() |>
    add_mean_value() |>
    add_mean_line()

  # Changing arguments: alpha
  # Makes objects transparent
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    theme_minimal_y() |>
    add_mean_bar(alpha = 0.4)

  # Changing arguments: saturation
  # Reduces fill color saturation without making the object transparent
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    theme_minimal_y() |>
    add_mean_bar(saturation = 0.3)

  # Changing arguments: accuracy
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    add_mean_value(accuracy = 0.01)

  # Changing arguments: fontsize
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    add_mean_value(fontsize = 10)

  # Changing arguments: color
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    add_mean_value(color = "black")
```

`add_median_bar` *Add median*

Description

Add median

Usage

```
add_median_bar(
  plot,
  dodge_width = NULL,
  width = 0.6,
  saturation = 1,
```

```
  preserve = "total",
  ...
)

add_median_dash(
  plot,
  dodge_width = NULL,
  width = 0.6,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_median_dot(plot, dodge_width = NULL, size = 2, preserve = "total", ...)

add_median_value(
  plot,
  dodge_width = NULL,
  accuracy = 0.1,
  scale_cut = NULL,
  fontsize = 7,
  extra_padding = 0.15,
  vjust = NULL,
  hjust = NULL,
  preserve = "total",
  ...
)

add_median_line(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_median_area(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.

`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`width`	Width of the bar.
`saturation`	A `number` between `0` and `1` for the color saturation of an object. A value of `0` is completely desaturated (white), `1` is the original color.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`...`	Arguments passed on to the `geom` function.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`size`	A `number` representing the size of the plot symbol. Typical values range between `1` and `3`.
`accuracy`	A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.
`scale_cut`	Scale cut function to be applied. See `scales::cut_short_scale()` and friends.
`fontsize`	Font size in points. Defaults to `7`.
`extra_padding`	Extra padding to create space for the value label.
`vjust`	Vertical position adjustment of the value label.
`hjust`	Horizontal position adjustment of the value label.
`group`	Variable in the dataset to be used for grouping.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_median_bar()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_median_dash()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_median_dot()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_median_value()

study |>
  tidyplot(x = treatment, y = score) |>
```

```
  add_median_line()

study |>
  tidyplot(x = treatment, y = score) |>
  add_median_area()

# Combination
study |>
  tidyplot(x = treatment, y = score) |>
  add_median_bar(alpha = 0.4) |>
  add_median_dash() |>
  add_median_dot() |>
  add_median_value() |>
  add_median_line()

# Changing arguments: alpha
# Makes objects transparent
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  theme_minimal_y() |>
  add_median_bar(alpha = 0.4)

# Changing arguments: saturation
# Reduces fill color saturation without making the object transparent
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  theme_minimal_y() |>
  add_median_bar(saturation = 0.3)

# Changing arguments: accuracy
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_median_value(accuracy = 0.01)

# Changing arguments: fontsize
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_median_value(fontsize = 10)

# Changing arguments: color
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_median_value(color = "black")
```

`add_pie` *Add pie or donut chart*

Description

Add pie or donut chart

Usage

```
add_pie(plot, width = 1, reverse = FALSE, ...)

add_donut(plot, width = 1, reverse = FALSE, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`width`	Width of the donut ring.
`reverse`	Whether the order should be reversed or not. Defaults to `FALSE`, meaning not reversed.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
# for a `count` only provide `color`
# `count` of the data points in each `energy_type` category
energy |>
  tidyplot(color = energy_type) |>
  add_pie()

energy |>
  tidyplot(color = energy_type) |>
  add_donut()

energy |>
  tidyplot(color = energy_type) |>
  add_donut(width = 0.5)

# for a `sum` provide `color` and `y`
# `sum` of `energy` in each `energy_type` category
energy |>
  tidyplot(y = energy, color = energy_type) |>
  add_pie()

energy |>
  tidyplot(y = energy, color = energy_type) |>
  add_donut()

energy |>
  tidyplot(y = energy, color = energy_type) |>
  add_donut(width = 0.5)
```

`add_reference_lines` *Add reference lines*

Description

Add reference lines

Usage

```
add_reference_lines(
  plot,
  x = NULL,
  y = NULL,
  linetype = "dashed",
  linewidth = 0.25,
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`x`	Numeric values where the reference lines should meet the x-axis. For example, `x = 4` or `x = c(2,3,4)`.
`y`	Numeric values where the reference lines should meet the y-axis. For example, `y = 4` or `y = c(2,3,4)`.
`linetype`	Either an integer (0-6) or a name (0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash).
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
animals |>
  tidyplot(x = weight, y = speed) |>
   add_reference_lines(x = 4000, y = c(100, 200)) |>
   add_data_points()

animals |>
  tidyplot(x = weight, y = speed) |>
   add_reference_lines(x = 4000, y = c(100, 200), linetype = "dotdash") |>
   add_data_points()
```

`add_sem_errorbar` *Add error bar*

Description

- `add_sem_errorbar()` adds the standard error of mean.
- `add_range_errorbar()` adds the range from smallest to largest value.
- `add_sd_errorbar()` adds the standard deviation.
- `add_ci95_errorbar()` adds the 95% confidence interval.

Usage

```
add_sem_errorbar(
  plot,
  dodge_width = NULL,
  width = 0.4,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_range_errorbar(
  plot,
  dodge_width = NULL,
  width = 0.4,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_sd_errorbar(
  plot,
  dodge_width = NULL,
  width = 0.4,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_ci95_errorbar(
  plot,
  dodge_width = NULL,
  width = 0.4,
  linewidth = 0.25,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`width`	Width of the error bar.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
# Standard error of the mean
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sem_errorbar()

# Range from minimum to maximum value
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_range_errorbar()

# Standard deviation
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sd_errorbar()

# 95% confidence interval
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_ci95_errorbar()

# Changing arguments: error bar width
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sem_errorbar(width = 0.8)

# Changing arguments: error bar line width
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
```

```
add_sem_errorbar(linewidth = 1)
```

`add_sem_ribbon` *Add ribbon*

Description

- `add_sem_ribbon()` adds the standard error of mean.
- `add_range_ribbon()` adds the range from smallest to largest value.
- `add_sd_ribbon()` adds the standard deviation.
- `add_ci95_ribbon()` adds the 95% confidence interval.

Usage

```
add_sem_ribbon(plot, dodge_width = NULL, alpha = 0.4, color = NA, ...)

add_range_ribbon(plot, dodge_width = NULL, alpha = 0.4, color = NA, ...)

add_sd_ribbon(plot, dodge_width = NULL, alpha = 0.4, color = NA, ...)

add_ci95_ribbon(plot, dodge_width = NULL, alpha = 0.4, color = NA, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`alpha`	A `number` between `0` and `1` for the opacity of an object. A value of `0` is completely transparent, `1` is completely opaque.
`color`	A hex color for the stroke color. For example, `"#FFFFFF"` for white.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
# Standard error of the mean
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_mean_line() |>
  add_sem_ribbon()

# Range from minimum to maximum value
```

```
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_mean_line() |>
  add_range_ribbon()

# Standard deviation
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_mean_line() |>
  add_sd_ribbon()

# 95% confidence interval
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_mean_line() |>
  add_ci95_ribbon()

# Changing arguments: alpha
time_course |>
  tidyplot(x = day, y = score, color = treatment) |>
  add_mean_line() |>
  add_sem_ribbon(alpha = 0.7)
```

`add_sum_bar` *Add sum*

Description

Add sum

Usage

```
add_sum_bar(
  plot,
  dodge_width = NULL,
  width = 0.6,
  saturation = 1,
  preserve = "total",
  ...
)

add_sum_dash(
  plot,
  dodge_width = NULL,
  width = 0.6,
  linewidth = 0.25,
  preserve = "total",
  ...
```

```
)

add_sum_dot(plot, dodge_width = NULL, size = 2, preserve = "total", ...)

add_sum_value(
  plot,
  dodge_width = NULL,
  accuracy = 0.1,
  scale_cut = NULL,
  fontsize = 7,
  extra_padding = 0.15,
  vjust = NULL,
  hjust = NULL,
  preserve = "total",
  ...
)

add_sum_line(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)

add_sum_area(
  plot,
  group,
  dodge_width = NULL,
  linewidth = 0.25,
  preserve = "total",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`width`	Width of the bar.
`saturation`	A `number` between `0` and `1` for the color saturation of an object. A value of `0` is completely desaturated (white), `1` is the original color.
`preserve`	Should dodging preserve the `"total"` width of all elements at a position, or the width of a `"single"` element?
`...`	Arguments passed on to the `geom` function.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.

`size`	A `number` representing the size of the plot symbol. Typical values range between `1` and `3`.
`accuracy`	A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.
`scale_cut`	Scale cut function to be applied. See `scales::cut_short_scale()` and friends.
`fontsize`	Font size in points. Defaults to `7`.
`extra_padding`	Extra padding to create space for the value label.
`vjust`	Vertical position adjustment of the value label.
`hjust`	Horizontal position adjustment of the value label.
`group`	Variable in the dataset to be used for grouping.

Value

A `tidyplot` object.

Examples

```
spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_bar()

spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_dash()

spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_dot()

spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_value()

spendings |>
  tidyplot(x = category, y = amount) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_line()

spendings |>
  tidyplot(x = category, y = amount) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_area()
```

```
# Combination
spendings |>
  tidyplot(x = category, y = amount) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_median_bar(alpha = 0.4) |>
  add_median_dash() |>
  add_median_dot() |>
  add_median_value() |>
  add_median_line()

# Changing arguments: alpha
# Makes objects transparent
spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  theme_minimal_y() |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_bar(alpha = 0.4)

# Changing arguments: saturation
# Reduces fill color saturation without making the object transparent
spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  theme_minimal_y() |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_bar(saturation = 0.3)

# Changing arguments: accuracy
spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_value(accuracy = 1)

# Changing arguments: fontsize
spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_value(fontsize = 10)

# Changing arguments: color
spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_value(color = "black")

# Changing arguments: extra_padding
spendings |>
  tidyplot(x = category, y = amount, color = category) |>
  adjust_x_axis(rotate_labels = TRUE) |>
  add_sum_value(extra_padding = 0.5)
```

`add_test_pvalue` *Add statistical test*

Description

Add statistical test

Usage

```
add_test_pvalue(
  plot,
  padding_top = 0.15,
  method = "t_test",
  p.adjust.method = "none",
  ref.group = NULL,
  comparisons = NULL,
  paired_by = NULL,
  label = "{format_p_value(p.adj, 0.0001)}",
  label.size = 7/ggplot2::.pt,
  step.increase = 0.15,
  vjust = -0.25,
  bracket.nudge.y = 0.1,
  hide.ns = FALSE,
  p.adjust.by = "panel",
 symnum.args = list(cutpoints = c(0, 0.001, 0.01, 0.05, Inf), symbols = c("***", "**",
    "*", "ns")),
  hide_info = FALSE,
  ...
)

add_test_asterisks(
  plot,
  padding_top = 0.1,
  method = "t_test",
  p.adjust.method = "none",
  ref.group = NULL,
  comparisons = NULL,
  paired_by = NULL,
  label = "p.adj.signif",
  label.size = 10/ggplot2::.pt,
  step.increase = 0.2,
  vjust = 0.3,
  bracket.nudge.y = 0.15,
  hide.ns = TRUE,
  p.adjust.by = "panel",
 symnum.args = list(cutpoints = c(0, 0.001, 0.01, 0.05, Inf), symbols = c("***", "**",
    "*", "ns")),
```

```
  hide_info = FALSE,
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`padding_top`	Extra padding above the data points to accommodate the statistical comparisons.
`method`	a character string indicating which method to be used for pairwise comparisons. Default is `"t_test"`. Allowed methods include pairwise comparisons methods implemented in the `rstatix` R package. These methods are: `"wilcox_test"`, `"t_test"`, `"sign_test"`, `"dunn_test"`, `"emmeans_test"`, `"tukey_hsd"`, `"games_howell_test"`.
`p.adjust.method`	method for adjusting p values (see `p.adjust`). Has impact only in a situation, where multiple pairwise tests are performed; or when there are multiple grouping variables. Ignored when the specified method is `"tukey_hsd"` or `"games_howell_test"` because they come with internal p adjustment method. Allowed values include "holm", "hochberg", "hommel", "bonferroni", "BH", "BY", "fdr", "none". If you don't want to adjust the p value (not recommended), use p.adjust.method = "none".
`ref.group`	a character string or a numeric value specifying the reference group. If specified, for a given grouping variable, each of the group levels will be compared to the reference group (i.e. control group). `ref.group` can be also `"all"`. In this case, each of the grouping variable levels is compared to all (i.e. basemean). Allowed values can be: • **numeric value**: specifying the rank of the reference group. For example, use `ref.group = 1` when the first group is the reference; use `ref.group = 2` when the second group is the reference, and so on. This works for all situations, including i) when comparisons are performed between x-axis groups and ii) when comparisons are performed between legend groups. • **character value**: For example, you can use `ref.group = "ctrl"` instead of using the numeric rank value of the "ctrl" group. • **"all"**: In this case, each of the grouping variable levels is compared to all (i.e. basemean).
`comparisons`	A list of length-2 vectors. The entries in the vector are 2 integers that correspond to the index of the groups of interest, to be compared.
`paired_by`	Variable to be used for paired analysis.
`label`	character string specifying label. Can be: • the column containing the label (e.g.: `label = "p"` or `label = "p.adj"`), where `p` is the p-value. Other possible values are `"p.signif"`, `"p.adj.signif"`, `"p.format"`, `"p.adj.format"`. • an expression that can be formatted by the `glue()` package. For example, when specifying `label = "Wilcoxon, p = \{p\}"`, the expression {p} will be replaced by its value.

- a combination of plotmath expressions and glue expressions. You may want some of the statistical parameter in italic; for example:`label = "Wilcoxon,italic(p)= {p}"`

.

`label.size`	change the size of the label text
`step.increase`	numeric vector with the increase in fraction of total height for every additional comparison to minimize overlap.
`vjust`	move the text up or down relative to the bracket.
`bracket.nudge.y`	Vertical adjustment to nudge brackets by (in fraction of the total height). Useful to move up or move down the bracket. If positive value, brackets will be moved up; if negative value, brackets are moved down.
`hide.ns`	can be logical value (`TRUE` or `FALSE`) or a character vector (`"p.adj"` or `"p"`).
`p.adjust.by`	possible value is one of `c("group", "panel")`. Default is `"group"`: for a grouped data, if pairwise test is performed, then the p-values are adjusted for each group level independently. P-values are adjusted by panel when `p.adjust.by = "panel"`.
`symnum.args`	a list of arguments to pass to the function symnum for symbolic number coding of p-values. For example, `symnum.args <- list(cutpoints = c(0, 0.0001, 0.001, 0.01, 0.05, Inf), symbols = c("****", "***", "**", "*", "ns"))`. In other words, we use the following convention for symbols indicating statistical significance: • `ns`: p > 0.05 • `*`: p <= 0.05 • `**`: p <= 0.01 • `***`: p <= 0.001 • `****`: p <= 0.0001
`hide_info`	Whether to hide details about the statistical testing as caption. Defaults to `FALSE`.
`...`	Arguments passed on to `ggpubr::geom_pwc()`.

Details

- `add_test_pvalue()` and `add_test_asterisks()` use `ggpubr::geom_pwc()`. Check there for additional arguments.
- Known limitation: `add_test_pvalue()` and `add_test_asterisks()` expect a discrete variable on the x-axis and a continuous variable on the y-axis. To produce horizontal plots, use `flip_plot()`.

Value

A `tidyplot` object.

Examples

```
# Add p value
study |>
  tidyplot(x = dose, y = score, color = group) |>
  add_mean_dash() |>
  add_sem_errorbar() |>
  add_data_points() |>
  add_test_pvalue()

# Add asterisks
study |>
  tidyplot(x = dose, y = score, color = group) |>
  add_mean_dash() |>
  add_sem_errorbar() |>
  add_data_points() |>
  add_test_asterisks()

# Change stat method
study |>
  tidyplot(x = dose, y = score, color = group) |>
  add_mean_dash() |>
  add_sem_errorbar() |>
  add_data_points() |>
  add_test_pvalue(method = "wilcoxon")

# Change p.adjust method
study |>
  tidyplot(x = dose, y = score, color = group) |>
  add_mean_dash() |>
  add_sem_errorbar() |>
  add_data_points() |>
  add_test_pvalue(p.adjust.method = "bonferroni")

# Define reference group to test against
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_dash() |>
  add_sem_errorbar() |>
  add_data_points() |>
  add_test_asterisks(ref.group = 1)

# Define selected comparisons
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_dash() |>
  add_sem_errorbar() |>
  add_data_points() |>
  add_test_pvalue(comparisons = list(c(1,3),c(2,4)))

# Paired analysis
x <- c(2.3, 4.5, 6.3, 3.4, 7.8, 6.7)
df <- data.frame(
```

```
    x = c(x, x + c(0.8, 0.75)),
    group = paste0("g", rep(c(1, 2), each = 6)),
    batch = paste0("b", c(1:6, 1:6)),
    shuffle = paste0("c", c(1:6, 6:1))
  )

  df |>
    tidyplot(group, x, color = group) |>
    add_boxplot() |>
    add_data_points() |>
    add_test_pvalue(paired_by = shuffle) |>
    add_line(group = shuffle, color = "black")

  df |>
    tidyplot(group, x, color = group) |>
    add_boxplot() |>
    add_data_points() |>
    add_test_pvalue(paired_by = batch) |>
    add_line(group = batch, color = "black")

  # hide non-significant p values
  gene_expression |>
    # filter to one gene
    dplyr::filter(external_gene_name == "Apol6") |>
    # start plotting
    tidyplot(x = condition, y = expression, color = sample_type) |>
    add_mean_dash() |>
    add_sem_errorbar() |>
    add_data_points() |>
    add_test_pvalue(hide.ns = TRUE)

  # Flip plot
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    add_mean_dash() |>
    add_sem_errorbar() |>
    add_data_points() |>
    add_test_asterisks(comparisons = list(c(1,4),c(2,3))) |>
    flip_plot()

  # Adjust top padding for statistical comparisons
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    add_mean_dash() |>
    add_sem_errorbar() |>
    add_data_points() |>
    add_test_pvalue(padding_top = 0.08)

  # Hide stats information
  study |>
    tidyplot(x = dose, y = score, color = group) |>
    add_mean_dash() |>
    add_sem_errorbar() |>
```

```
  add_data_points() |>
  add_test_pvalue(hide_info = TRUE)
```

`add_title` *Add plot title or caption*

Description

Add plot title or caption

Usage

```
add_title(plot, title = ggplot2::waiver())

add_caption(plot, caption = ggplot2::waiver())
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`title`	Title of the plot.
`caption`	Caption of the plot.

Details

- `add_title()` and `add_caption()` support plotmath expressions to include special characters. See examples and Advanced plotting.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points_beeswarm() |>
  add_title("This is my title")

study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points_beeswarm() |>
  add_caption("This is the fine print in the caption")

# Plotmath expression
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points_beeswarm() |>
  add_title("$H[2]*O~and~E==m*c^{2}$")
```

`add_violin` *Add violin plot*

Description

Add violin plot

Usage

```
add_violin(
  plot,
  dodge_width = NULL,
  alpha = 0.3,
  saturation = 1,
  trim = FALSE,
  linewidth = 0.25,
  scale = "width",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`dodge_width`	For adjusting the distance between grouped objects. Defaults to `0.8` for plots with at least one discrete axis and `0` for plots with two continuous axes.
`alpha`	A `number` between `0` and `1` for the opacity of an object. A value of `0` is completely transparent, `1` is completely opaque.
`saturation`	A `number` between `0` and `1` for the color saturation of an object. A value of `0` is completely desaturated (white), `1` is the original color.
`trim`	If `TRUE` (default), trim the tails of the violins to the range of the data. If `FALSE`, don't trim the tails.
`linewidth`	Thickness of the line in points (pt). Typical values range between `0.25` and `1`.
`scale`	if "area" (default), all violins have the same area (before trimming the tails). If "count", areas are scaled proportionally to the number of observations. If "width", all violins have the same maximum width.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_violin()

# Changing arguments:
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_violin(saturation = 0.6)

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_violin(draw_quantiles = c(0.25, 0.5, 0.75))

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_violin(trim = TRUE)

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_violin(linewidth = 1)
```

`adjust_colors` *Adjust colors*

Description

Adjust colors

Usage

```
adjust_colors(
  plot,
  new_colors = NULL,
  saturation = 1,
  labels = tidyplot_parse_labels(),
  downsample = c("evenly", "first", "last", "middle"),
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`new_colors`	A character vector of new hex colors to use. Can be a named character vector of hex colors to assign certain data labels to specific colors.
`saturation`	A `number` between `0` and `1` for the color saturation of an object. A value of `0` is completely desaturated (white), `1` is the original color.

`labels`	One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches. • `NULL` for no labels • `waiver()` for the default labels computed by the transformation object • A character vector giving labels (must be same length as `breaks`) • An expression vector (must be the same length as breaks). See ?plotmath for details. • A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.
`downsample`	If too many colors are provided, whether to downsample `evenly`, or use the `first`, the `last` or the `middle` colors of the color vector. Defaults to `evenly`.
`...`	Arguments passed on to the ggplot2 `scale` function.

Value

A `tidyplot` object.

See Also

`colors_discrete_friendly()`, `colors_continuous_viridis()`, `colors_diverging_blue2brown()`, and `new_color_scheme()`

Examples

```
# Plot without adjustments
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Provide hex colors
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_colors(new_colors = c("#644296","#F08533","#3B78B0", "#D1352C"))

# Provide discrete color scheme
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_colors(new_colors = colors_discrete_seaside)

# Provide named vector
study |>
```

```
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_colors(new_colors = c(
    "A" = "pink",
    "B" = "purple",
    "C" = "grey",
    "D" = "blue"))

# Provide continuous color scheme
climate |>
  tidyplot(x = month, y = year, color = max_temperature) |>
  add_heatmap() |>
  adjust_colors(new_colors = colors_continuous_turbo)
```

`adjust_font` *Adjust font*

Description

Adjust font

Usage

```
adjust_font(plot, fontsize = 7, family = NULL, face = NULL, color = "black")
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`fontsize`	Font size in points. Defaults to 7.
`family`	The typeface to use. The validity of this value will depend on the graphics device being used for rendering the plot. See the systemfonts vignette for guidance on the best way to access fonts installed on your computer. The values `"sans"`, `"serif"`, and `"mono"` should always be valid and will select the default typeface for the respective styles. However, what is considered default is dependant on the graphics device and the operating system.
`face`	Font face ("plain", "italic", "bold", "bold.italic")
`color`	A hex color for the stroke color. For example, `"#FFFFFF"` for white.

Value

A `tidyplot` object.

Examples

```
# Plot without adjustments
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Increase font size
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_font(fontsize = 16)

# Change font family
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_font(family = "mono")

# Change font face
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_font(face = "bold")
```

`adjust_legend_title` *Adjust legend*

Description

Adjust legend

Usage

```
adjust_legend_title(
  plot,
  title = ggplot2::waiver(),
  fontsize = NULL,
  family = NULL,
  face = NULL,
  color = "black",
```

```
  ...
)

adjust_legend_position(plot, position = "right")
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`title`	Legend title.
`fontsize`	Font size in points. Defaults to `7`.
`family`	The typeface to use. The validity of this value will depend on the graphics device being used for rendering the plot. See the systemfonts vignette for guidance on the best way to access fonts installed on your computer. The values `"sans"`, `"serif"`, and `"mono"` should always be valid and will select the default typeface for the respective styles. However, what is considered default is dependant on the graphics device and the operating system.
`face`	Font face ("plain", "italic", "bold", "bold.italic")
`color`	A hex color for the stroke color. For example, `"#FFFFFF"` for white.
`...`	Arguments passed on to `ggplot2::element_text()`.
`position`	The position of the legend. Can be one of `c("right", "left", "bottom", "top", "none")`. Defaults to `"right"`.

Details

- The `title` argument of `adjust_legend_title()` supports plotmath expressions to include special characters. See examples and Advanced plotting.

Value

A `tidyplot` object.

Examples

```
# Plot without adjustments
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# New title
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_legend_title("My new legend title")

# New title with plotmath expression
```

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_legend_title("$E==m*c^{2}$")

# Alternative legend positions
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_legend_position("left")

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_legend_position("top")

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_legend_position("bottom")

# `position = "none"` hides the legend
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_legend_position("none")
```

`adjust_padding` *Adjust plot area padding*

Description

Adjust plot area padding

Usage

```
adjust_padding(
  plot,
  top = NA,
```

```
  right = NA,
  bottom = NA,
  left = NA,
  all = NA,
  force_continuous = FALSE,
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`top`	Extra space between the data points and the top. Defaults to `NA`, which does not change the padding.
`right`	Extra space between the data points and the right. Defaults to `NA`, which does not change the padding.
`bottom`	Extra space between the data points and the bottom. Defaults to `NA`, which does not change the padding.
`left`	Extra space between the data points and the left. Defaults to `NA`, which does not change the padding.
`all`	Extra space around the data pointst. Overwrites `top`, `right`, `bottom`, `left` if set. Defaults to `NA`, which does not change the padding.
`force_continuous`	Whether to force the axis to be continuous. Defaults to `FALSE`.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
# Plot without adjustments
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_padding()

# Increase plot area padding
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_padding(all = 0.2)

animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_padding(top = 0.8)

animals |>
```

```
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_padding(bottom = 0.8)

animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_padding(right = 0.8)

animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_padding(left = 0.8)
```

`adjust_size` *Adjust plot area size*

Description

Adjust plot area size

Usage

```
adjust_size(plot, width = NULL, height = NULL, unit = NULL)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`width`	Width of the plot area.
`height`	Height of the plot area.
`unit`	Unit of the plot area width and height.

Value

A `tidyplot` object.

Examples

```
# Plot without adjustments
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm(shape = 1) |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Resize to 15 x 15 mm
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
```

```
  add_data_points_beeswarm(shape = 1) |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_size(width = 15, height = 15)

# Resize to 4 x 4 cm
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm(shape = 1) |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_size(width = 4, height = 4, unit = "cm")

# Remove absolute dimensions and take all available space.
# This is the ggplot2 default.
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm(shape = 1) |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  adjust_size(width = NA, height = NA)
```

`adjust_theme_details` *Adjust theme details*

Description

This function is a wrapper around `ggplot2::theme()`. To use the required theme helper functions `ggplot2::element_blank()`, `ggplot2::element_rect()`, `ggplot2::element_line()`, and `ggplot2::element_text()` you need to either load the ggplot2 package via `library(ggplot2)` or use the `ggplot2::` prefix as shown above.

Usage

```
adjust_theme_details(plot, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`...`	Arguments passed on to the `geom` function.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm() |>
  add_mean_bar(alpha = 0.4) |>
  adjust_theme_details(plot.background = ggplot2::element_rect(fill = "#FFEBFF"))
```

`adjust_title` *Adjust titles and caption*

Description

Adjust titles and caption

Usage

```
adjust_title(
  plot,
  title = ggplot2::waiver(),
  fontsize = NULL,
  family = NULL,
  face = NULL,
  color = "black",
  ...
)

adjust_x_axis_title(
  plot,
  title = ggplot2::waiver(),
  fontsize = NULL,
  family = NULL,
  face = NULL,
  color = "black",
  ...
)

adjust_y_axis_title(
  plot,
  title = ggplot2::waiver(),
  fontsize = NULL,
  family = NULL,
  face = NULL,
  color = "black",
  ...
)
```

```
adjust_caption(
  plot,
  caption = ggplot2::waiver(),
  fontsize = NULL,
  family = NULL,
  face = NULL,
  color = "black",
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`title`	Plot or axes title.
`fontsize`	Font size in points. Defaults to `7`.
`family`	The typeface to use. The validity of this value will depend on the graphics device being used for rendering the plot. See the systemfonts vignette for guidance on the best way to access fonts installed on your computer. The values `"sans"`, `"serif"`, and `"mono"` should always be valid and will select the default typeface for the respective styles. However, what is considered default is dependant on the graphics device and the operating system.
`face`	Font face ("plain", "italic", "bold", "bold.italic")
`color`	A hex color for the stroke color. For example, `"#FFFFFF"` for white.
`...`	Arguments passed on to `ggplot2::element_text()`.
`caption`	Plot caption.

Details

Adjust the plot title, axis titles and caption

- All functions support plotmath expressions to include special characters. See examples and Advanced plotting.

Value

A `tidyplot` object.

Examples

```
# Plot without adjustments
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Adjust description
study |>
```

```
    tidyplot(x = treatment, y = score, color = treatment) |>
    add_data_points() |>
    add_mean_bar(alpha = 0.4) |>
    add_sem_errorbar() |>
    adjust_title("This is my fantastic plot title") |>
    adjust_x_axis_title("Treatment group") |>
    adjust_y_axis_title("Disease score") |>
    adjust_legend_title("Legend title") |>
    adjust_caption("Here goes the caption")

  # Plotmath expressions
  study |>
    tidyplot(x = treatment, y = score, color = treatment) |>
    add_data_points() |>
    add_mean_bar(alpha = 0.4) |>
    add_sem_errorbar() |>
    adjust_title("$H[2]*O$") |>
    adjust_x_axis_title("$H[2]*O$") |>
    adjust_y_axis_title("$H[2]*O$") |>
    adjust_legend_title("$H[2]*O$") |>
    adjust_caption("$H[2]*O$")
```

`adjust_x_axis` *Adjust axes*

Description

Adjust axes

Usage

```
adjust_x_axis(
  plot,
  title = ggplot2::waiver(),
  breaks = ggplot2::waiver(),
  labels = ggplot2::waiver(),
  limits = NULL,
  padding = c(NA, NA),
  rotate_labels = FALSE,
  transform = "identity",
  cut_short_scale = FALSE,
  force_continuous = FALSE,
  ...
)

adjust_y_axis(
  plot,
  title = ggplot2::waiver(),
```

```
  breaks = ggplot2::waiver(),
  labels = ggplot2::waiver(),
  limits = NULL,
  padding = c(NA, NA),
  rotate_labels = FALSE,
  transform = "identity",
  cut_short_scale = FALSE,
  force_continuous = FALSE,
  ...
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`title`	Axis title.
`breaks`	One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object • A numeric vector of positions • A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.
`labels`	One of the options below. Please note that when `labels` is a vector, it is highly recommended to also set the `breaks` argument as a vector to protect against unintended mismatches. • `NULL` for no labels • `waiver()` for the default labels computed by the transformation object • A character vector giving labels (must be same length as `breaks`) • An expression vector (must be the same length as breaks). See ?plotmath for details. • A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.
`limits`	Axis limits. For example, with `limits = c(20, 90)` the axis starts at 20 and ends at 90.
`padding`	Extra space between the data points and the axes. Defaults to `c(NA, NA)`, which does not change the padding.
`rotate_labels`	Whether to rotate axis labels. If `TRUE` is set to 45 degrees. You can also provide custom degree values, for example, `rotate_labels = 90`. Defaults to `FALSE`.
`transform`	For continuous scales, the name of a transformation object or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "date", "exp", "hms", "identity", "log", "log10", "log1p", "log2", "logit", "modulus", "probability", "probit", "pseudo_log", "reciprocal", "reverse", "sqrt" and "time". A transformation object bundles together a transform, its inverse, and methods for generating breaks and labels. Transformation objects are defined in the scales

	package, and are called `transform_<name>`. If transformations require arguments, you can call them from the scales package, e.g. `scales::transform_boxcox(p = 2)`. You can create your own transformation with `scales::new_transform()`.
`cut_short_scale`	Whether to shorten axis labels using `K` for thousand, `M` for million, and so on. Defaults to `FALSE`.
`force_continuous`	Whether to force the axis to be continuous. Defaults to `FALSE`.
`...`	Arguments passed on to ggplot2 `scale` function.

Details

- The `title` argument of `adjust_x_axis()` and `adjust_y_axis()` supports plotmath expressions to include special characters. See examples and Advanced plotting.

Value

A `tidyplot` object.

Examples

```
# Plot without adjustments
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points()

# New titles
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_x_axis(title = "My new x-axis title") |>
  adjust_y_axis(title = "My new y-axis title")

# New titles with plotmath expressions
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_x_axis(title = "$H[2]*O$") |>
  adjust_y_axis(title = "$E==m*c^{2}$")

# Axes limits
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_x_axis(limits = c(-1000, 4000)) |>
  adjust_y_axis(limits = c(-200, 600))

# Rotate labels
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
```

```
  adjust_x_axis(rotate_labels = 90) |>
  adjust_y_axis(rotate_labels = 90)

# Increase plot area padding
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_x_axis(padding = c(0.2, 0.2)) |>
  adjust_y_axis(padding = c(0.2, 0.2))

# Scale transformation
animals |>
  tidyplot(x = weight, y = size, color = family) |>
  add_data_points() |>
  adjust_x_axis(transform = "log10") |>
  adjust_y_axis(transform = "log2")
```

`all_rows` *Subset data rows*

Description

Subset data rows

Usage

```
all_rows()

filter_rows(..., .by = NULL)

max_rows(order_by, n, by = NULL, with_ties = TRUE, na_rm = FALSE)

min_rows(order_by, n, by = NULL, with_ties = TRUE, na_rm = FALSE)

first_rows(n, by = NULL)

last_rows(n, by = NULL)

sample_rows(n, by = NULL)
```

Arguments

`...`	<`data-masking`> Expressions that return a logical value, and are defined in terms of the variables in `.data`. If multiple expressions are included, they are combined with the `&` operator. Only rows for which all conditions evaluate to `TRUE` are kept.

`.by, by`	**[Experimental]** <`tidy-select`> Optionally, a selection of columns to group by for just this operation, functioning as an alternative to `group_by()`. For details and examples, see ?dplyr_by.
`order_by`	<`data-masking`> Variable or function of variables to order by. To order by multiple variables, wrap them in a data frame or tibble.
`n`	The number of rows to select. If not are supplied, `n = 1` will be used. If `n` is greater than the number of rows in the group, the result will be silently truncated to the group size. A negative value of `n` will be subtracted from the group size. For example, `n = -2` with a group of 5 rows will select 5 - 2 = 3 rows.
`with_ties`	Should ties be kept together? The default, `TRUE`, may return more rows than you request. Use `FALSE` to ignore ties, and return the first `n` rows.
`na_rm`	Should missing values in `order_by` be removed from the result? If `FALSE`, `NA` values are sorted to the end (like in `dplyr::arrange()`), so they will only be included if there are insufficient non-missing values to reach `n`.

Value

A `function` to achieve the desired data subsetting.

Examples

```
# Highlight all animals
animals |>
 tidyplot(x = weight, y = size) |>
 add_data_points() |>
 add_data_points(data = all_rows(),
  color = "red", shape = 1, size = 3)

# Highlight 3 animals with the highest weight
animals |>
 tidyplot(x = weight, y = size) |>
 add_data_points() |>
 add_data_points(data = max_rows(weight, n = 3),
  color = "red", shape = 1, size = 3)

# Highlight 3 animals with the lowest weight
animals |>
 tidyplot(x = weight, y = size) |>
 add_data_points() |>
 add_data_points(data = min_rows(weight, n = 3),
  color = "red", shape = 1, size = 3)

# Highlight the first 3 animals in the dataset
animals |>
 tidyplot(x = weight, y = size) |>
 add_data_points() |>
 add_data_points(data = first_rows(n = 3),
  color = "red", shape = 1, size = 3)
```

```
# Highlight the last 3 animals in the dataset
animals |>
 tidyplot(x = weight, y = size) |>
 add_data_points() |>
 add_data_points(data = last_rows(n = 3),
  color = "red", shape = 1, size = 3)

# Highlight 3 random animals
animals |>
 tidyplot(x = weight, y = size) |>
 add_data_points() |>
 add_data_points(data = sample_rows(n = 3),
  color = "red", shape = 1, size = 3)
```

`animals` *Animals data*

Description

Animals data

Usage

```
animals
```

Format

A data frame.

Source

ChatGPT-3.5, *Caution: The accuracy of the data has not been verified.*

Examples

```
dplyr::glimpse(animals)
```

`climate` *Climate data*

Description

Climate data

Usage

```
climate
```

Format

A data frame.

Source

National Oceanic and Atmospheric Administration, Temperature data, weather station Hamburg Fuhlsbüttel, Germany

Examples

```
dplyr::glimpse(climate)
```

`colors_continuous_viridis` *Continuous color schemes*

Description

For more information about the use of color schemes in tidyplots, check out this article: Color schemes

Usage

```
colors_continuous_viridis

colors_continuous_magma

colors_continuous_inferno

colors_continuous_plasma

colors_continuous_cividis

colors_continuous_rocket
```

`colors_continuous_mako`

`colors_continuous_turbo`

`colors_continuous_bluepinkyellow`

Format

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 265.

An object of class `tidycolor` (inherits from `character`) of length 11.

Details

Color schemes can be conveniently previewed by using the print method of the `tidycolor` class. This will send a html preview to the RStudio Viewer pane.

`colors_continuous_viridis`

`colors_continuous_magma`

`colors_continuous_inferno`

`colors_continuous_plasma`

`colors_continuous_cividis`

`colors_continuous_rocket`

`colors_continuous_mako`

`colors_continuous_turbo`

`colors_continuous_bluepinkyellow`

`colors_discrete_friendly` *Discrete color schemes*

Description

For more information about the use of color schemes in tidyplots, check out this article: Color schemes

Usage

```
colors_discrete_friendly

colors_discrete_seaside

colors_discrete_apple

colors_discrete_friendly_long

colors_discrete_okabeito

colors_discrete_ibm

colors_discrete_metro

colors_discrete_candy

colors_discrete_alger

colors_discrete_rainbow
```

Format

An object of class `tidycolor` (inherits from `character`) of length 6.

An object of class `tidycolor` (inherits from `character`) of length 5.

An object of class `tidycolor` (inherits from `character`) of length 7.

An object of class `tidycolor` (inherits from `character`) of length 7.

An object of class `tidycolor` (inherits from `character`) of length 7.

An object of class `tidycolor` (inherits from `character`) of length 5.

An object of class `tidycolor` (inherits from `character`) of length 5.

An object of class `tidycolor` (inherits from `character`) of length 5.

An object of class `tidycolor` (inherits from `character`) of length 5.

An object of class `tidycolor` (inherits from `character`) of length 9.

Details

The signature theme of tidyplots `colors_discrete_friendly` was adapted from the Okabe & Ito color palette that was designed to work well for people with color vision deficiency.

Color schemes can be conveniently previewed by using the print method of the `tidycolor` class. This will send a html preview to the RStudio Viewer pane.

```
colors_discrete_friendly

colors_discrete_seaside

colors_discrete_apple

colors_discrete_friendly_long

colors_discrete_okabeito

colors_discrete_ibm

colors_discrete_metro

colors_discrete_candy

colors_discrete_alger

colors_discrete_rainbow
```

`colors_diverging_blue2red`

Diverging color schemes

Description

For more information about the use of color schemes in tidyplots, check out this article: Color schemes

Usage

```
colors_diverging_blue2red

colors_diverging_blue2brown

colors_diverging_BuRd

colors_diverging_BuYlRd

colors_diverging_spectral

colors_diverging_icefire
```

Format

An object of class `tidycolor` (inherits from `character`) of length 17.

An object of class `tidycolor` (inherits from `character`) of length 17.

An object of class `tidycolor` (inherits from `character`) of length 11.

An object of class `tidycolor` (inherits from `character`) of length 11.

An object of class `tidycolor` (inherits from `character`) of length 96.

An object of class `tidycolor` (inherits from `character`) of length 96.

Details

Color schemes can be conveniently previewed by using the print method of the `tidycolor` class. This will send a html preview to the RStudio Viewer pane.

`colors_diverging_blue2red`

`colors_diverging_blue2brown`

`colors_diverging_BuRd`

`colors_diverging_BuYlRd`

`colors_diverging_spectral`

`colors_diverging_icefire`

`dinosaurs` *Dinosaurs data*

Description

Dinosaurs data

Usage

```
dinosaurs
```

Format

A data frame.

Source

ChatGPT-3.5, *Caution: The accuracy of the data has not been verified.*

Examples

```
dplyr::glimpse(dinosaurs)
```

`distributions` *Distributions data*

Description

Distributions data

Usage

```
distributions
```

Format

A data frame.

Source

tidyplots package

Examples

```
dplyr::glimpse(distributions)
```

`energy` *Energy data*

Description

Energy data

Usage

```
energy
```

Format

A data frame.

Source

Energy-Charts, Energy production data, Germany

Examples

```
dplyr::glimpse(energy)
```

`energy_week` *Energy week data*

Description

Energy week data

Usage

```
energy_week
```

Format

A data frame.

Source

Energy-Charts, Energy production data, Germany

Examples

```
dplyr::glimpse(energy_week)
```

`eu_countries` *EU countries data*

Description

EU countries data

Usage

```
eu_countries
```

Format

A data frame.

Source

ChatGPT-3.5, *Caution: The accuracy of the data has not been verified.*

Examples

```
dplyr::glimpse(eu_countries)
```

`flip_plot` *Flip x and y-axis*

Description

[Superseded]

This function is superseded because in many cases, `flip_plot()` can easily be replaced by swapping the `x` and `y` axis. Some plot components additionally require to set the `orientation` argument to `"y"`.

Usage

```
flip_plot(plot, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`...`	Arguments passed on to `ggplot2::coord_flip()`.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  flip_plot()

energy |>
  tidyplot(x = year, y = energy, color = energy_type) |>
  add_barstack_absolute() |>
  flip_plot()

# Better solutions without `flip_plot()`
study |>
  tidyplot(x = score, y = treatment, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

energy |>
  tidyplot(x = energy, y = year, color = energy_type) |>
  add_barstack_absolute(orientation = "y")
```

`format_p_value` *Format p values*

Description

Format p values

Usage

```
format_p_value(x, accuracy = 1e-04)
```

Arguments

`x`	A number to format.
`accuracy`	A number to round to. For example, use `0.01` to show 2 decimal places of precision. Defaults to `0.0001`, corresponding to 4 decimal places of precision.

Value

Formatted number as `character` string.

Examples

```
format_p_value(0.03445553)
format_p_value(0.0003445553)
format_p_value(0.00003445553)
```

`gene_expression` *RNA-Seq expression data*

Description

RNA-Seq expression data

Usage

```
gene_expression
```

Format

A data frame.

Source

Bassoon proteinopathy drives neurodegeneration in multiple sclerosis, Nature Neuroscience 2019

GSE104899, Gene Expression Omnibus

Examples

```
dplyr::glimpse(gene_expression)
```

`new_color_scheme` *New color scheme*

Description

For more information about the use of color schemes in tidyplots, check out this article: Color schemes

Usage

```
new_color_scheme(x, name = "Untitled color scheme", reverse = FALSE)
```

Arguments

`x`	Character vector of hex colors. For example x = `c("#FF00FF", "#00FFFF")`.
`name`	Name of the custom color scheme.
`reverse`	Whether the order should be reversed or not. Defaults to `FALSE`, meaning not reversed.

Value

A `tidyplot` object.

Examples

```
new_color_scheme(c("#ECA669","#E06681","#8087E2","#E2D269"))

new_color_scheme(c("#ECA669","#E06681","#8087E2","#E2D269"),
  name = "my_custom_color_scheme")
```

`pca` *Principle component analysis data*

Description

Principle component analysis data

Usage

```
pca
```

Format

A data frame.

Source

Bassoon proteinopathy drives neurodegeneration in multiple sclerosis, Nature Neuroscience 2019

Examples

```
dplyr::glimpse(pca)
```

`remove_legend` *Remove legend or legend title*

Description

Remove legend or legend title

Usage

```
remove_legend(plot)

remove_legend_title(plot)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.

Value

A `tidyplot` object.

Examples

```
# Before removing
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar()

# After removing
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_legend_title()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_legend()
```

`remove_padding` *Remove plot area padding*

Description

Remove plot area padding

Usage

```
remove_padding(plot, force_continuous = FALSE)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`force_continuous`	Whether to force the axis to be continuous. Defaults to `FALSE`.

Value

A `tidyplot` object.

Examples

```
# Before removing
animals |>
  tidyplot(x = weight, y = speed, color = family) |>
  add_data_points()

# After removing
animals |>
  tidyplot(x = weight, y = speed, color = family) |>
  add_data_points() |>
  remove_padding()
```

`remove_title` *Remove plot title or caption*

Description

Remove plot title or caption

Usage

```
remove_title(plot)

remove_caption(plot)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.

Value

A `tidyplot` object.

Examples

```
# Before removing
animals |>
  tidyplot(x = weight, y = speed, color = family) |>
  add_data_points() |>
  add_title("Name of the plot") |>
  add_caption("This is the caption")

# After removing
animals |>
  tidyplot(x = weight, y = speed, color = family) |>
  add_data_points() |>
  add_title("Name of the plot") |>
  add_caption("This is the caption") |>
  remove_title()

animals |>
  tidyplot(x = weight, y = speed, color = family) |>
  add_data_points() |>
  add_title("Name of the plot") |>
  add_caption("This is the caption") |>
  remove_caption()
```

`remove_x_axis` *Remove x-axis or parts of it*

Description

Remove x-axis or parts of it

Usage

```
remove_x_axis(plot)

remove_x_axis_line(plot)

remove_x_axis_ticks(plot)

remove_x_axis_labels(plot)

remove_x_axis_title(plot)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.

Value

A `tidyplot` object.

Examples

```
# Before removing
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar()

# After removing
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_x_axis_line()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_x_axis_ticks()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_x_axis_labels()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_x_axis_title()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_x_axis()
```

`remove_y_axis` *Remove y-axis or parts of it*

Description

Remove y-axis or parts of it

Usage

```
remove_y_axis(plot)

remove_y_axis_line(plot)

remove_y_axis_ticks(plot)

remove_y_axis_labels(plot)

remove_y_axis_title(plot)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.

Value

A `tidyplot` object.

Examples

```
# Before removing
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar()

# After removing
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_y_axis_line()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_y_axis_ticks()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_y_axis_labels()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_y_axis_title()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar() |>
  remove_y_axis()
```

`rename_x_axis_levels` *Rename axis or color levels*

Description

Rename axis or color levels

Usage

```
rename_x_axis_levels(plot, new_names)

rename_y_axis_levels(plot, new_names)

rename_color_levels(plot, new_names)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`new_names`	Named character vector in the format c("old1" = "new1", "old2" = "new2").

Value

A `tidyplot` object.

Examples

```
# Before adjustments
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Rename x-axis levels
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  rename_x_axis_levels(new_names = c(
    "A" = "This",
    "B" = "is",
    "C" = "totally",
    "D" = "new"))

# Before adjustments
study |>
```

```
    tidyplot(x = score, y = treatment) |>
    add_data_points() |>
    add_mean_bar(alpha = 0.4) |>
    add_sem_errorbar()

  # Rename y-axis levels
  study |>
    tidyplot(x = score, y = treatment) |>
    add_data_points() |>
    add_mean_bar(alpha = 0.4) |>
    add_sem_errorbar() |>
    rename_y_axis_levels(new_names = c(
      "A" = "This",
      "B" = "is",
      "C" = "totally",
      "D" = "new"))

  # Before adjustment
  study |>
    tidyplot(x = group, y = score, color = dose) |>
    add_data_points() |>
    add_mean_bar(alpha = 0.4) |>
    add_sem_errorbar()

  # Rename color levels
  study |>
    tidyplot(x = group, y = score, color = dose) |>
    add_data_points() |>
    add_mean_bar(alpha = 0.4) |>
    add_sem_errorbar() |>
    rename_color_levels(new_names = c(
      "high" = "Sky high",
      "low" = "Deep low"))
```

`reorder_x_axis_levels` *Reorder axis or color levels*

Description

Reorder axis or color levels

Usage

```
reorder_x_axis_levels(plot, ...)

reorder_y_axis_levels(plot, ...)

reorder_color_levels(plot, ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`...`	Arguments passed on to `forcats::fct_relevel()`.

Value

A `tidyplot` object.

Examples

```
# Before adjustments
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Reorder x-axis levels
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  reorder_x_axis_levels("D", "B", "A")

# Before adjustments
study |>
  tidyplot(x = score, y = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Reorder y-axis levels
study |>
  tidyplot(x = score, y = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  reorder_y_axis_levels("D", "B", "A")

# Before adjustment
study |>
  tidyplot(x = group, y = score, color = dose) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Reorder color levels
study |>
  tidyplot(x = group, y = score, color = dose) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
```

```
  add_sem_errorbar() |>
  reorder_color_levels("low")
```

`reverse_x_axis_levels` *Reverse axis or color levels*

Description

Reverse axis or color levels

Usage

```
reverse_x_axis_levels(plot)

reverse_y_axis_levels(plot)

reverse_color_levels(plot)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.

Value

A `tidyplot` object.

Examples

```
# Before adjustments
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Reverse x-axis levels
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  reverse_x_axis_levels()

# Before adjustments
study |>
  tidyplot(x = score, y = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()
```

```
# Reverse y-axis levels
study |>
  tidyplot(x = score, y = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  reverse_y_axis_levels()

# Before adjustment
study |>
  tidyplot(x = group, y = score, color = dose) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Reverse color levels
study |>
  tidyplot(x = group, y = score, color = dose) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  reverse_color_levels()
```

`save_plot` *Save plots to file*

Description

This function takes a plot or list of plots and writes them to a (multipage) file.

Usage

```
save_plot(
  plot = ggplot2::last_plot(),
  filename,
  width = NA,
  height = NA,
  units = c("mm", "cm", "in"),
  multiple_files = FALSE,
  view_plot = TRUE,
  bg = "transparent",
  ...
)
```

Arguments

`plot`	Plot to save, defaults to last plot displayed.

`filename`	File name to create on disk.
`width, height`	Dimensions of the graphic device to save the plot. Defaults to `NA`. In case of `NA`, the dimensions are inferred from the incoming `plot` object (see Details).
`units`	Units of length. Defaults to `"mm"`.
`multiple_files`	Whether to save multiple pages as individual files.
`view_plot`	Whether to view the plot on screen after saving.
`bg`	Background colour. If `NULL`, uses the `plot.background` fill value from the plot theme.
`...`	Other arguments passed on to the graphics device function, as specified by `device`.

Details

Handling of file dimensions. Output file dimensions are determined according the the following precedence.

1. The `width` and `height` arguments.
2. Dimensions inferred from the incoming `plot` object with absolute dimensions.
3. System default device dimensions.

Value

A `tidyplot` object.

Examples

```
# Save plot to file
study |>
  tidyplot(treatment, score) |>
  add_data_points() |>
  save_plot("single_plot.pdf")

# Save intermediate stages to file
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  add_data_points_beeswarm() |>
  save_plot("before.pdf") |>
  adjust_colors(colors_discrete_seaside) |>
  save_plot("after.pdf")



# Save multipage PDF file
gene_expression |>
  dplyr::slice_head(n = 160) |>
  tidyplot(group, expression, color = sample_type) |>
  add_data_points() |>
```

```
  adjust_size(width = 30, height = 25) |>
  split_plot(by = external_gene_name, nrow = 2, ncol = 2) |>
  save_plot("multipage_plot.pdf")

# Save multiple PDF files
gene_expression |>
  dplyr::slice_head(n = 160) |>
  tidyplot(group, expression, color = sample_type) |>
  add_data_points() |>
  adjust_size(width = 30, height = 25) |>
  split_plot(by = external_gene_name, nrow = 2, ncol = 2) |>
  save_plot("plot.pdf", multiple_files = TRUE)
```

`sort_x_axis_levels` *Sort axis or color levels*

Description

Sort axis or color levels

Usage

```
sort_x_axis_levels(plot, ..., .fun = NULL, .reverse = FALSE)

sort_y_axis_levels(plot, ..., .fun = NULL, .reverse = FALSE)

sort_color_levels(plot, ..., .fun = NULL, .reverse = FALSE)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`...`	Optional variables to use for sorting.
`.fun`	Override the function used for sorting. Is automatically determined from the plot.
`.reverse`	Whether the order should be reversed or not. Defaults to `FALSE`, meaning not reversed.

Value

A `tidyplot` object.

Examples

```
# Before adjustments
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Sort x-axis levels by score
study |>
  tidyplot(x = treatment, y = score) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  sort_x_axis_levels()

# Before adjustments
study |>
  tidyplot(x = score, y = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Sort y-axis levels by score
study |>
  tidyplot(x = score, y = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  sort_y_axis_levels()

# Before adjustment
study |>
  tidyplot(x = group, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar()

# Sort color levels by score
study |>
  tidyplot(x = group, y = score, color = treatment) |>
  add_data_points() |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  sort_color_levels()
```

`spendings` *Spending data*

Description

Spending data

Usage

```
spendings
```

Format

A data frame.

Source

tidyplots package

Examples

```
dplyr::glimpse(spendings)
```

`split_plot` *Split plot into multiple subplots*

Description

Split plot into multiple subplots

Usage

```
split_plot(
  plot,
  by,
  ncol = NULL,
  nrow = NULL,
  byrow = NULL,
  guides = "collect",
  tag_level = NULL,
  design = NULL
)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`by`	Variable that should be used for splitting.
`ncol`, `nrow`	The number of columns and rows per page.
`byrow`	Analogous to `byrow` in matrix(). If `FALSE` the plots will be filled in in column-major order

`guides`	A string specifying how guides should be treated in the layout. `'collect'` will collect guides below to the given nesting level, removing duplicates. `'keep'` will stop collection at this level and let guides be placed alongside their plot. `auto` will allow guides to be collected if a upper level tries, but place them alongside the plot if not. If you modify default guide "position" with theme(legend.position=...) while also collecting guides you must apply that change to the overall patchwork (see example).
`tag_level`	A string (`'keep'` or `'new'`) to indicate how auto-tagging should behave. See `plot_annotation()`.
`design`	Specification of the location of areas in the layout. Can either be specified as a text string or by concatenating calls to `area()` together. See the examples for further information on use.

Value

A `tidyplot` object.

Examples

```
# Before splitting
energy |>
  dplyr::filter(year %in% c(2005, 2010, 2015, 2020)) |>
  tidyplot(y = energy, color = energy_source) |>
  add_donut() |>
  adjust_size(width = 25, height = 25)

# Split by year
energy |>
  dplyr::filter(year %in% c(2005, 2010, 2015, 2020)) |>
  tidyplot(y = energy, color = energy_source) |>
  add_donut() |>
  adjust_size(width = 25, height = 25) |>
  split_plot(by = year)

# Change dimensions of subplots
energy |>
  dplyr::filter(year %in% c(2005, 2010, 2015, 2020)) |>
  tidyplot(y = energy, color = energy_source) |>
  add_donut() |>
  adjust_size(width = 15, height = 15) |>
  split_plot(by = year)

# Spread plots across multiple pages
energy |>
  dplyr::filter(year %in% c(2005, 2010, 2015, 2020)) |>
  tidyplot(y = energy, color = energy_source) |>
  add_donut() |>
  adjust_size(width = 25, height = 25) |>
  split_plot(by = year, ncol = 2, nrow = 1)
```

`study` *Study data*

Description

Study data

Usage

```
study
```

Format

A data frame.

Source

tidyplots package

Examples

```
dplyr::glimpse(study)
```

`theme_tidyplot` *Themes*

Description

Themes

Usage

```
theme_tidyplot(plot, fontsize = 7)

theme_ggplot2(plot, fontsize = 7)

theme_minimal_xy(plot, fontsize = 7)

theme_minimal_x(plot, fontsize = 7)

theme_minimal_y(plot, fontsize = 7)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`fontsize`	Font size in points. Defaults to 7.

Value

A `tidyplot` object.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sem_errorbar() |>
  add_mean_dash() |>
  theme_tidyplot()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sem_errorbar() |>
  add_mean_dash() |>
  theme_ggplot2()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sem_errorbar() |>
  add_mean_dash() |>
  theme_minimal_xy()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sem_errorbar() |>
  add_mean_dash() |>
  theme_minimal_x()

study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points() |>
  add_sem_errorbar() |>
  add_mean_dash() |>
  theme_minimal_y()
```

`tidyplot` *Create a new tidyplot*

Description

Create a new tidyplot

Usage

```
tidyplot(
  data,
  ...,
  width = NULL,
  height = NULL,
  unit = NULL,
  dodge_width = NULL,
  my_style = NULL
)
```

Arguments

`data`	A tidy `data.frame` to use for plotting.
`...`	Mappings for the `x` axis, `y` axis and `color`, see examples. Additional argument are passed to `ggplot2::aes()`.
`width`	Width of the plot area. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `50`.
`height`	Height of the plot area. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `50`.
`unit`	Unit of the plot area width and height. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `"mm"`.
`dodge_width`	For adjusting the distance between grouped objects. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `0.8` for plots with at least one discrete axis and to `0` for plots with two continuous axes.
`my_style`	Styling function to apply to the plot. The default (`NULL`) retrieves the setting from the tidyplots options, which default to no additional styling.

Examples

```
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_data_points_beeswarm()

study |>
  tidyplot(x = group, y = score, color = dose) |>
  add_mean_bar()

# Change plot area size
study |>
  tidyplot(x = treatment, y = score, color = treatment,
    width = 25, height = 25) |>
  add_data_points_beeswarm()

# Change dodge_width
study |>
  tidyplot(x = group, y = score, color = dose, dodge_width = 0.3) |>
  add_mean_bar()
```

`tidyplots_options` *Tidyplots options*

Description

Control the settings for formatting tidyplots globally.

Usage

```
tidyplots_options(
  width = NULL,
  height = NULL,
  unit = NULL,
  dodge_width = NULL,
  my_style = NULL
)
```

Arguments

`width`	Width of the plot area. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `50`.
`height`	Height of the plot area. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `50`.
`unit`	Unit of the plot area width and height. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `"mm"`.
`dodge_width`	For adjusting the distance between grouped objects. The default (`NULL`) retrieves the setting from the tidyplots options, which defaults to `0.8` for plots with at least one discrete axis and to `0` for plots with two continuous axes.
`my_style`	Styling function to apply to the plot. The default (`NULL`) retrieves the setting from the tidyplots options, which default to no additional styling.

Value

The old options invisibly

Examples

```
# Define custom style
my_style <- function(x) x |>
  adjust_colors(colors_discrete_candy) |>
  adjust_font(family = "mono")

# Set tidyplots options
tidyplots_options(
  width = 3,
  height = 4,
  unit = "cm",
```

```
    dodge_width = 1,
    my_style = my_style
    )

  # Plot
  study |>
    tidyplot(x = group, y = score, color = dose) |>
    add_mean_bar()

  # Reset tidyplots options
  tidyplots_options()

  # Same plot
  study |>
    tidyplot(x = group, y = score, color = dose) |>
    add_mean_bar()
```

`time_course` *Time course data*

Description

Time course data

Usage

```
time_course
```

Format

A data frame.

Source

tidyplots package

Examples

```
dplyr::glimpse(time_course)
```

`view_plot` *View plot on screen*

Description

View plot on screen

Usage

```
view_plot(plot, data = all_rows(), title = ggplot2::waiver(), ...)
```

Arguments

`plot`	A `tidyplot` generated with the function `tidyplot()`.
`data`	The data to be displayed in this layer. There are three options: • If `all_rows()` (the default) the complete dataset is displayed. • A `function` to subset the plot data. See `filter_rows()` and friends. • A `data.frame` to override the plot data.
`title`	Plot title.
`...`	Arguments passed on to `print()`.

Details

- `view_plot()` supports data subsetting. See examples and Advanced plotting.

Value

A `tidyplot` object.

Examples

```
# View intermediate stages on screen
study |>
  tidyplot(x = treatment, y = score, color = treatment) |>
  add_mean_bar(alpha = 0.4) |>
  add_sem_errorbar() |>
  add_data_points_beeswarm() |>
  view_plot(title = "Before changing color scheme") |>
  adjust_colors(colors_discrete_seaside) |>
  view_plot(title = "After changing color scheme")

# View data subsets on screen
gene_expression |>
  tidyplot(x = condition, y = expression, color = sample_type) |>
  add_mean_dash() |>
  add_sem_errorbar() |>
  add_data_points_beeswarm() |>
```

```
view_plot(data = filter_rows(external_gene_name == "Apol6"),
  title = "Apol6") |>
view_plot(data = filter_rows(external_gene_name == "Bsn"),
  title = "Bsn")
```

Index