Package 'thisplot'

March 7, 2026

Type Package

Title Utility Functions for Plotting

Version 0.3.6

Date 2026-03-07

Maintainer Meng Xu <mengxu98@qq.com>

Description Provides utility functions for plotting. Includes functions for color manipulation, plot customization, panel size control, data optimization for plots, and layout adjustments.

License MIT + file LICENSE

URL https://mengxu98.github.io/thisplot/

BugReports https://github.com/mengxu98/thisplot/issues

Depends R (>= 4.1.0)

Imports cli, dendextend, geomtextpath, ggrepel, ggplot2, gtable, igraph, methods, patchwork, stats, thisutils (>= 0.4.3), utils

Suggests circlize, ComplexHeatmap, dplyr, ggupset, ggVennDiagram, grDevices, grid, htmltools, Matrix, png, proxyC, purrr, ragg, scales, stringr, tidyr

Config/Needs/website mengxu98/thistemplate

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

Language en-US

LazyData true

NeedsCompilation no

Author Meng Xu [aut, cre] (ORCID: <https://orcid.org/0000-0002-8300-1054>)

Repository CRAN

Date/Publication 2026-03-07 15:40:02 UTC

Contents

`thisplot-package` *Utility Functions for Data Visualization and Plotting*

Description

Provides utility functions for data visualization and plotting in R. Includes functions for color manipulation, plot customization, panel size control, data optimization for plots, and layout adjustments. Designed to enhance workflows with ggplot2, patchwork, and ComplexHeatmap.

Author(s)

Meng Xu (Maintainer), <mengxu98@qq.com>

Source

https://mengxu98.github.io/thisplot/

See Also

Useful links:

- https://mengxu98.github.io/thisplot/
- Report bugs at https://github.com/mengxu98/thisplot/issues

add_grob *Add a grob to a gtable*

Description

Add a grob to a gtable at a specified position (top, bottom, left, or right).

Usage

```
add_grob(
  gtable,
  grob,
  position = c("top", "bottom", "left", "right", "none"),
  space = NULL,
  clip = "on"
)
```

Arguments

gtable	A gtable object.
grob	A grob or gtable object to add.
position	The position to add the grob. One of "top", "bottom", "left", "right", or "none".
space	The space to allocate for the grob. If NULL, will be calculated automatically.
clip	The clipping mode. Default is "on".

Value

A gtable object with the grob added.

adjcolors *Convert a color with specified alpha level*

Description

Convert a color with specified alpha level

Usage

```
adjcolors(colors, alpha)
```

Arguments

colors	Color vectors.
alpha	Alpha level in [0,1].

Value

A character vector of hexadecimal color codes with the specified alpha level.

Examples

```
colors <- c("red", "blue", "green")
adjcolors(colors, 0.5)
ggplot2::alpha(colors, 0.5)

show_palettes(
  list(
    "raw" = colors,
    "adjcolors" = adjcolors(colors, 0.5),
    "ggplot2::alpha" = ggplot2::alpha(colors, 0.5)
  )
)
```

`adjustlayout` *Adjust graph layout to avoid node overlaps*

Description

Adjust the layout of a graph to prevent node overlaps by considering node widths and heights.

Usage

```
adjustlayout(graph, layout, width, height = 2, scale = 100, iter = 100)
```

Arguments

`graph`	An igraph graph object.
`layout`	A matrix with two columns representing the initial layout coordinates.
`width`	A numeric vector of node widths.
`height`	The height constraint for nodes. Default is `2`.
`scale`	The scaling factor for the layout. Default is `100`.
`iter`	The number of iterations for the adjustment algorithm. Default is `100`.

Value

A matrix with adjusted layout coordinates.

`annotation_block_fill_graphics` *Build block fill panel function*

Description

Build block fill panel function

Usage

```
annotation_block_fill_graphics(
  levels,
  palette = NULL,
  palcolor = NULL,
  fill_values = NULL,
  border = FALSE
)
```

Arguments

`levels`	Character/factor levels used as keys for fill mapping.
`palette`	Palette name. All available palette names can be queried with show_palettes.
`palcolor`	Custom colors used to create a color palette.
`fill_values`	Optional named fill vector. Names should match `levels`.
`border`	Whether to draw block border.

Value

A function with signature `(index, levels)` suitable for `panel_fun` in ComplexHeatmap::anno_block.

Examples

```
library(ggplot2)
lv <- c("A", "B", "C")
panel_fun <- annotation_block_fill_graphics(
  levels = lv,
  fill_values = c(A = "#1b9e77", B = "#d95f02", C = "#7570b3")
)
ComplexHeatmap::anno_block(panel_fun = panel_fun)
```

annotation_block_graphics *Build block panel function*

Description

Build block panel function

Usage

```
annotation_block_graphics(subplot, name, border = TRUE)
```

Arguments

subplot	A single drawable subplot object.
name	A name assigned to the generated grob.
border	Whether to draw a rectangle border around the block.

Value

A function with signature `(index, levels)` suitable for `panel_fun` in ComplexHeatmap::anno_block.

Examples

```
library(ggplot2)
p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
  ggplot2::geom_point()
panel_fun <- annotation_block_graphics(
  subplot = p, name = "demo-block"
)
ComplexHeatmap::anno_block(panel_fun = panel_fun)
```

annotation_graphics *Build graphics callback list for anno_customize*

Description

Build graphics callback list for anno_customize

Usage

```
annotation_graphics(subplots, prefix)
```

Arguments

subplots	A named list of subplot objects.
prefix	Prefix used to generate stable grob names.

Value

A named list of callback functions accepted by ComplexHeatmap::anno_customize.

`as_grob` *Convert a plot object to a grob*

Description

Convert various plot objects (gList, patchwork, ggplot) to a grob object.

Usage

```
as_grob(plot, ...)
```

Arguments

`plot`	A plot object (gList, patchwork, or ggplot).
`...`	Additional arguments passed to other functions.

Value

A grob object.

`as_gtable` *Convert a plot object to a gtable*

Description

Convert various plot objects (gtable, grob, patchwork, ggplot) to a gtable object.

Usage

```
as_gtable(plot, ...)
```

Arguments

`plot`	A plot object (gtable, grob, patchwork, or ggplot).
`...`	Additional arguments passed to other functions.

Value

A gtable object.

Blend2Color *Blend two colors using a specified mode*

Description

Blend two colors with alpha channels using one of several blending modes: blend, average, screen, or multiply.

Usage

```
Blend2Color(C1, C2, mode = "blend")
```

Arguments

C1	A list containing the first color RGB values and alpha channel.
C2	A list containing the second color RGB values and alpha channel.
mode	The blending mode to use. One of "blend", "average", "screen", or "multiply". Default is "blend".

Value

A list containing the blended RGB values and alpha channel.

blendcolors *Blends a list of colors using the specified blend mode*

Description

Blends a list of colors using the specified blend mode

Usage

```
blendcolors(colors, mode = c("blend", "average", "screen", "multiply"))
```

Arguments

colors	Color vectors.
mode	Blend mode. One of "blend", "average", "screen", or "multiply".

Value

A character vector of hexadecimal color codes representing the blended color.

Examples

```
blend <- c(
  "red",
  "green",
  blendcolors(c("red", "green"),
    mode = "blend"
  )
)
average <- c(
  "red",
  "green",
  blendcolors(c("red", "green"),
    mode = "average"
  )
)
screen <- c(
  "red",
  "green",
  blendcolors(c("red", "green"),
    mode = "screen"
  )
)
multiply <- c(
  "red",
  "green",
  blendcolors(c("red", "green"),
    mode = "multiply"
  )
)
show_palettes(
  list(
    "blend" = blend,
    "average" = average,
    "screen" = screen,
    "multiply" = multiply
  )
)
```

`BlendRGBList` *Blend a list of colors*

Description

Blend multiple colors with alpha channels into a single color using a specified blending mode.

Usage

```
BlendRGBList(Clist, mode = "blend", RGB_BackGround = c(1, 1, 1))
```

Arguments

Clist	A list of colors, where each color is a list containing RGB values and alpha channel.
mode	The blending mode to use. One of "blend", "average", "screen", or "multiply". Default is "blend".
RGB_BackGround	The background RGB color to composite with. Default is c(1, 1, 1) (white).

Value

A numeric vector of RGB values.

---

build_heatmap_annotation

Build HeatmapAnnotation with safe parameter merge

Description

Build HeatmapAnnotation with safe parameter merge

Usage

```
build_heatmap_annotation(
  annotations,
  which,
  show_annotation_name = TRUE,
  annotation_name_side = NULL,
  border = NULL,
  params = NULL
)
```

Arguments

annotations	Named list of annotation components (for example, anno_simple, anno_block, anno_customize).
which	Annotation direction ("row" or "column").
show_annotation_name	Whether to show annotation names.
annotation_name_side	Side for annotation names.
border	Border flag passed to ComplexHeatmap::HeatmapAnnotation.
params	Additional user parameters; duplicated keys are ignored if already set explicitly.

Value

A ComplexHeatmap::HeatmapAnnotation object.

Examples

```
anno <- list(
  group = ComplexHeatmap::anno_simple(
    x = c("A", "B", "A"),
    col = c(A = "#1b9e77", B = "#d95f02"),
    which = "column"
  )
)
ha <- build_heatmap_annotation(
  annotations = anno,
  which = "column",
  show_annotation_name = TRUE,
  annotation_name_side = "left",
  params = list(gap = grid::unit(1, "mm"))
)
ha
```

---

build_patchwork *Build a patchwork gtable*

Description

Build a gtable from a patchwork object by arranging multiple plots according to the layout specification.

Usage

```
build_patchwork(
  x,
  guides = "auto",
  table_rows = 18,
  table_cols = 15,
  panel_row = 10,
  panel_col = 8
)
```

Arguments

x	A patchwork object.
guides	How to handle guides. Default is "auto".
table_rows	The number of rows in the table grid. Default is 18.
table_cols	The number of columns in the table grid. Default is 15.
panel_row	The row index for panels. Default is 10.
panel_col	The column index for panels. Default is 8.

Value

A gtable object.

ChineseColors *Chinese traditional colors system*

Description

A color system based on Chinese traditional colors with 1058 colors.

Usage

```
ChineseColors()
```

Value

A ChineseColors object. Detailed information can be found in print.ChineseColors().

See Also

chinese_colors for the dataset of Chinese traditional colors. get_chinese_palettes for getting Chinese color palettes. visual_colors for visualizing any color vector. get_colors for searching colors in dataset and palettes.

Examples

```
cc <- ChineseColors()
cc

# Get a color by pinyin
get_colors("pinlan")

# By number
get_colors(44)

# By hex code
get_colors("#2B73AF")

# Multiple colors
get_colors("pinlan", "piao")
get_colors(91:100)

# Chinese names
cc$visual_colors(
  title = "Chinese Traditional Colors",
  name_type = "chinese"
)

# pinyin as names
cc$visual_colors(
  loc_range = c(1, 120),
  title = "Chinese Traditional Colors",
  name_type = "pinyin"
```

```
)

# rgb as names
cc$visual_colors(
  loc_range = c(1, 120),
  title = "Colors with RGB values",
  name_type = "rgb"
)

# hex as names
cc$visual_colors(
  loc_range = c(1, 120),
  title = "Colors with hex codes",
  name_type = "hex"
)
```

`chinese_colors` *Chinese traditional colors dataset*

Description

A dataset containing optimized traditional Chinese colors. These colors are extracted from those books:

- Chinese Traditional Colors - Color Aesthetics in the Forbidden City (ISBN: 9787521716054)
- Chinese Beautiful Colors - The Most Chinese Culture Vol.3 (ISBN: 9781672897198)
- Chinese Colors (ISBN: 9787558016479)
- Chinese Journal of Chromatography (ISSN 1000-8713)
- Chinese Color Atlas

Thanks to the author of the blog for providing the data.

Examples

```
data(chinese_colors)
color_sets <- attr(chinese_colors, "color_sets")
show_palettes(
  list(
    color_sets$ChineseSet8,
    color_sets$ChineseSet16,
    color_sets$ChineseSet32
  )
)

# Use ChineseColors class
cc <- ChineseColors()
cc$visual_colors(
  title = "Chinese Traditional Colors",
  name_type = "chinese"
)
```

cluster_within_group2 *Cluster within group*

Description

Cluster within group

Usage

```
cluster_within_group2(mat, factor)
```

Arguments

mat	A matrix of data
factor	A factor

Value

A dendrogram with ordered leaves

Examples

```
mat <- matrix(rnorm(100), 10, 10)
factor <- factor(rep(1:2, each = 5))
dend <- cluster_within_group2(mat, factor)
dend
plot(dend)
```

col2hex *Convert color names to hexadecimal format*

Description

Convert color names to hexadecimal RGB color codes.

Usage

```
col2hex(cname)
```

Arguments

cname	A character vector of color names.

Value

A character vector of hexadecimal color codes.

drop_data *Drop unused data in the plot*

Description

Drop unused data points from a ggplot or patchwork object while preserving the plot structure. This function keeps only a single row of data for each unique combination of used variables, which can significantly reduce the object size when the original data contains many rows that are not displayed in the plot (e.g., due to scale limits or filtering).

Usage

```
drop_data(p)

## S3 method for class 'ggplot'
drop_data(p)

## S3 method for class 'patchwork'
drop_data(p)

## Default S3 method:
drop_data(p)
```

Arguments

p	A ggplot object or a patchwork object.

Value

A ggplot or patchwork object with unused data points removed.

Examples

```
library(ggplot2)
library(patchwork)
p <- ggplot(
  data = mtcars,
  aes(x = mpg, y = wt, colour = cyl)
) +
  geom_point() +
  scale_x_continuous(limits = c(10, 30)) +
  scale_y_continuous(limits = c(1, 6))
object.size(p)

p_drop <- drop_data(p)
object.size(p_drop)

p / p_drop
```

extractgrobs *Extract grobs from a list*

Description

Extract grobs from a named list of grobs based on the specified x and y indices.

Usage

```
extractgrobs(vlnplots, x_nm, y_nm, x, y)
```

Arguments

vlnplots	A named list of grobs.
x_nm	A character vector of names for the x dimension.
y_nm	A character vector of names for the y dimension.
x	An integer index for the x dimension.
y	An integer index for the y dimension.

Value

The extracted grob(s).

geom_alluvial *geom_alluvial*

Description

Creates an alluvial plot which visualize flows between nodes. Each observation needs to have a ‘x‘ aesthetic as well as a ‘next_x‘ column which declares where that observation should flow. Also each observation should have a ‘node‘ and a ‘next_node‘ aesthetic which provide information about which group in the y-direction.

Usage

```
geom_alluvial(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  space = 0,
  width = 0.1,
  smooth = 8,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Provide you own mapping. Both x and y need to be numeric.
data	Provide you own data.
position	Change position.
na.rm	Remove missing values.
show.legend	Show legend in plot.
space	Space between nodes in the y-direction.
width	Width of nodes.
smooth	How much smooth should the curve have? More means steeper curve.
inherit.aes	Should the geom inherit aesthetics.
...	Other arguments to be passed to the geom.

Value

A ggplot layer.

---

geom_alluvial_label *geom_alluvial_label*

Description

Creates centered labels or text in nodes of your alluvial plot. Needs to have the exact same aesthetics as the call to ‘geom_alluvial‘ to work.

Usage

```
geom_alluvial_text(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  space = 0,
  width = 0.1,
  inherit.aes = TRUE,
  ...
)

geom_alluvial_label(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
```

```
  space = 0,
  width = 0.1,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Provide you own mapping. Both x and y need to be numeric.
data	Provide you own data.
position	Change position.
na.rm	Remove missing values.
show.legend	Show legend in plot.
space	Space between nodes in the y-direction.
width	Width of nodes.
inherit.aes	Should the geom inherit aesthetics.
...	Other arguments to be passed to the geom.

Details

Other important arguments are; ‘space‘ which provides the space between nodes in the y-direction; ‘shift‘ which shifts nodes in the y-direction.

Value

A ggplot layer.

---

geom_sankey *geom_sankey*

Description

Creates a sankey plot which visualize flows between nodes. Each observation needs to have a ‘x‘ aesthetic as well as a ‘next_x‘ column which declares where that observation should flow. Also each observation should have a ‘node‘ and a ‘next_node‘ aesthetic which provide information about which group in the y-direction. By default each row of the data frame is counted to calculate the size of flows. A manual flow value can be added with the ‘value‘ aesthetic.

Usage

```
geom_sankey(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  space = NULL,
  type = "sankey",
  width = 0.1,
  smooth = 8,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Provide you own mapping. Both x and y need to be numeric.
data	Provide you own data.
position	Change position.
na.rm	Remove missing values.
show.legend	Show legend in plot.
space	Space between nodes in the y-direction.
type	Either ‘sankey‘ or ‘alluvial‘.
width	Width of nodes.
smooth	How much smooth should the curve have? More means steeper curve.
inherit.aes	Should the geom inherit aesthetics.
...	Other arguments to be passed to the geom.

Value

A ggplot layer.

Aesthetics

geom_sankey understand the following aesthetics (required aesthetics are in bold):

- **x0** - **y0** - **a** - **b** - **angle** - m1 - m2 - color - fill - size - linetype - alpha - lineend

Examples

```
dat <- data.frame(
  Group = c("A", "A", "B", "B", "C"),
  Type = c("X", "Y", "X", "Y", "X")
)
long <- make_long(dat, Group, Type)
```

```
ggplot2::ggplot(
  long,
  ggplot2::aes(
    x = x,
    next_x = next_x,
    node = node,
    next_node = next_node,
    fill = node
  )
) +
  geom_sankey() +
  theme_sankey()
```

---

geom_sankey_bump *geom_sankey_bump*

Description

Creates a sankey bump plot which visualize flows between nodes. Each observation needs to have a ‘x‘ aesthetic as well as a ‘next_x‘ column which declares where that observation should flow. Also, each observation should have a ‘node‘ and a ‘next_node‘ aesthetic which provide information about which group in the y-direction.

Usage

```
geom_sankey_bump(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  smooth = 8,
  type = "sankey",
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Provide you own mapping. Both x and y need to be numeric.
data	Provide you own data.
position	Change position.
na.rm	Remove missing values.
show.legend	Show legend in plot.
smooth	How much smooth should the curve have? More means steeper curve.
type	Either ‘sankey‘ or ‘alluvial‘.
inherit.aes	Should the geom inherit aesthetics.
...	Other arguments to be passed to the geom.

Details

Other important arguments are; ‘space‘ which provides the space between nodes in the y-direction; ‘shift‘ which shifts nodes in the y-direction.

Value

A ggplot layer.

`geom_sankey_label` *geom_sankey_label*

Description

Creates centered labels or text in nodes of your sankey plot. Needs to have the exact same aesthetics as the call to ‘geom_sankey‘ to work.

Usage

```
geom_sankey_label(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  space = NULL,
  type = "sankey",
  width = 0.1,
  inherit.aes = TRUE,
  ...
)

geom_sankey_text(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  space = NULL,
  type = "sankey",
  width = 0.1,
  inherit.aes = TRUE,
  ...
)
```

Arguments

mapping	Provide you own mapping. Both x and y need to be numeric.
data	Provide you own data.
position	Change position.
na.rm	Remove missing values.
show.legend	Show legend in plot.
space	Space between nodes in the y-direction.
type	Either ‘sankey‘ or ‘alluvial‘.
width	Width of nodes.
inherit.aes	Should the geom inherit aesthetics.
...	Other arguments to be passed to the geom.

Value

A ggplot layer.

---

get_chinese_palettes *Get Chinese color palettes*

Description

Get Chinese color palettes

Usage

```
get_chinese_palettes(prefix = "Chinese")
```

Arguments

prefix	The prefix of the palette names. Default is ‘"Chinese_"‘.

Value

A list of Chinese color palettes.

Examples

```
show_palettes(get_chinese_palettes())
```

`get_colors` *Get colors from Chinese colors dataset or palettes*

Description

Search for colors in the Chinese colors dataset and all available palettes. This function can search by palette names, color names (pinyin or Chinese), numbers, or hex codes. It automatically searches in all palettes and reports which palette(s) contain the found colors.

Usage

```
get_colors(..., palettes = NULL)
```

Arguments

`...`	One or more search values. Can be palette names, color names (pinyin or Chinese), numbers, or hex codes. If NULL, using all Chinese colors.
`palettes`	Optional. A named list of palettes to search in. If `NULL` (default), searches in all available palettes.

Value

A data frame with class `colors` containing matching color information. The result is automatically printed using `print.colors()`.

See Also

chinese_colors for the dataset of Chinese traditional colors. get_chinese_palettes for getting Chinese color palettes. ChineseColors for the ChineseColors object.

Examples

```
get_colors("Paired")

get_colors("#FF7F00")

get_colors("pinlan")
get_colors(44)
get_colors("#2B73AF")

get_colors("cyan", palettes = "ChineseSet64")
```

get_legend *Extract legend from a plot*

Description

Extract the legend grob from a plot object.

Usage

```
get_legend(plot)
```

Arguments

plot	A plot object.

Value

The legend grob.

get_vars *Get used vars in a ggplot object*

Description

Get used vars in a ggplot object

Usage

```
get_vars(p, reverse = FALSE, verbose = TRUE)
```

Arguments

p	A ggplot object.
reverse	Whether to return unused vars. Default is FALSE.
verbose	Whether to print the message. Default is TRUE.

Value

A character vector of variable names. If reverse is FALSE, returns used variables; if TRUE, returns unused variables.

Examples

```
library(ggplot2)
p <- ggplot(
  data = mtcars,
  aes(x = mpg, y = wt, colour = cyl)
) +
  geom_point()
get_vars(p)
get_vars(p, reverse = TRUE)
```

`grid_draw` *Draw grobs at specified positions*

Description

Draw a list of grobs at specified positions with given widths and heights.

Usage

```
grid_draw(groblist, x, y, width, height)
```

Arguments

`groblist`	A grob or a list of grobs to draw.
`x`	A numeric vector of x positions for each grob.
`y`	A numeric vector of y positions for each grob.
`width`	A numeric vector of widths for each grob.
`height`	A numeric vector of heights for each grob.

Value

No return value, called for side effects (drawing grobs).

`head.colors` *Return the first part of a colors object*

Description

Returns the first part of a colors object, similar to `head()` for data frames.

Usage

```
## S3 method for class 'colors'
head(x, n = 6L, ...)
```

Arguments

x	A colors object (data frame with color information).
n	Number of rows to return. Default is 6.
...	Additional arguments passed to head().

Value

A colors object with the first n rows.

Examples

```
head(get_colors())

head(get_colors(), n = 10)
```

---

heatmap_fixsize *Compute fixed heatmap device size*

Description

Compute fixed heatmap device size

Usage

```
heatmap_fixsize(
  width,
  width_sum,
  height,
  height_sum,
  units,
  ht_list,
  legend_list
)
```

Arguments

width	Optional user-provided target width. If NULL, width is estimated from the drawn heatmap layout.
width_sum	Numeric baseline width used when heatmap body size is in npc units.
height	Optional user-provided target height. If NULL, height is estimated from the drawn heatmap layout.
height_sum	Numeric baseline height used when heatmap body size is in npc units.
units	Output unit string passed to grid conversion helpers, such as "in", "cm", or "mm".
ht_list	A ComplexHeatmap::Heatmap or ComplexHeatmap::HeatmapList object.
legend_list	A list of ComplexHeatmap::Legend objects.

Value

A list with two converted units: `ht_width` and `ht_height`.

Examples

```
mat <- matrix(rnorm(100), nrow = 10)
ht <- ComplexHeatmap::Heatmap(mat, name = "expr")
lgd <- list(ComplexHeatmap::Legend(title = "expr", at = c(-2, 0, 2)))
out <- heatmap_fixsize(
  width = NULL,
  width_sum = 6,
  height = NULL,
  height_sum = 4,
  units = "in",
  ht_list = ht,
  legend_list = lgd
)
out
```

---

heatmap_rendersize *Estimate heatmap render size*

Description

Estimate heatmap render size

Usage

```
heatmap_rendersize(
  width,
  height,
  units,
  ha_top_list,
  ha_left,
  ha_right,
  ht_list,
  legend_list,
  flip
)
```

Arguments

`width`	Numeric vector of heatmap body widths. Interpretation depends on `flip`.
`height`	Numeric vector of heatmap body heights. Interpretation depends on `flip`.
`units`	Unit string for numeric-to-grid conversion (for example, `"in"`, `"cm"`, `"mm"`).
`ha_top_list`	A list of top annotations (typically `ComplexHeatmap::HeatmapAnnotation` objects).

ha_left	Optional left annotation.
ha_right	Optional right annotation.
ht_list	A ComplexHeatmap::Heatmap or ComplexHeatmap::HeatmapList object.
legend_list	A list containing legend objects; NULL entries are ignored.
flip	Logical; when TRUE, width/height interpretation follows the flipped layout branch.

Value

A list with numeric width_sum and height_sum in units.

Examples

```
mat <- matrix(rnorm(100), nrow = 10)
ht <- ComplexHeatmap::Heatmap(mat, name = "expr")
size <- heatmap_rendersize(
  width = c(4),
  height = c(3),
  units = "in",
  ha_top_list = list(),
  ha_left = NULL,
  ha_right = NULL,
  ht_list = ht,
  legend_list = list(),
  flip = FALSE
)
size
```

---

make_long *Make a long data frame for sankey plot*

Description

Prepares a 'wide' data frame into a format that 'geom_sankey' or 'geom_alluvial' understands. Useful to show flows between dimensions in dataset.

Usage

```
make_long(.df, ..., value = NULL)
```

Arguments

.df	a data frame
...	unquoted columnnames of df that you want to include in the plot.
value	if each row have a weight this weight could be kept by providing column name of weight.

Value

a longer data frame

mestimate *Estimate the fuzzifier parameter m*

Description

Estimate the fuzzifier parameter m for fuzzy clustering based on the data dimensions.

Usage

```
mestimate(data)
```

Arguments

data	A matrix or data frame.

Value

The estimated fuzzifier parameter m.

normalize_drawable *Normalize drawable objects to grobs*

Description

Normalize drawable objects to grobs

Usage

```
normalize_drawable(obj)
```

Arguments

obj	A drawable object, such as a ggplot, patchwork, or grid grob.

Value

A grid grob object. If the input is not drawable, returns a null grob.

Examples

```
library(ggplot2)
p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
  ggplot2::geom_point()
g <- normalize_drawable(p)
p + g
```

palette_colors *Color palettes collected*

Description

This function creates a color palette for a given vector of values.

Usage

```
palette_colors(
  x,
  n = 100,
  palette = "Paired",
  palcolor = NULL,
  type = c("auto", "discrete", "continuous"),
  matched = FALSE,
  reverse = FALSE,
  NA_keep = FALSE,
  NA_color = "grey80"
)
```

Arguments

x	A vector of character/factor or numeric values. If missing, numeric values 1:n will be used as x.
n	The number of colors to return for numeric values.
palette	Palette name. All available palette names can be queried with show_palettes.
palcolor	Custom colors used to create a color palette.
type	Type of x. Can be one of "auto", "discrete" or "continuous". The default is "auto", which automatically detects if x is a numeric value.
matched	Whether to return a color vector of the same length as x. Default is FALSE.
reverse	Whether to invert the colors. Default is FALSE.
NA_keep	Whether to keep the color assignment to NA in x. Default is FALSE.
NA_color	Color assigned to NA if NA_keep is TRUE. Default is "grey80".

Value

A character vector of color codes (hexadecimal format) corresponding to the input values x. The length and structure depend on the matched parameter.

See Also

show_palettes, palette_list

Examples

```
x <- c(1:3, NA, 3:5)
(pal1 <- palette_colors(
  x,
  palette = "Spectral"
))
(pal2 <- palette_colors(
  x,
  palcolor = c("red", "white", "blue")
))
(pal3 <- palette_colors(
  x,
  palette = "Spectral",
  n = 10
))
(pal4 <- palette_colors(
  x,
  palette = "Spectral",
  n = 10,
  reverse = TRUE
))
(pal5 <- palette_colors(
  x,
  palette = "Spectral",
  matched = TRUE
))
(pal6 <- palette_colors(
  x,
  palette = "Spectral",
  matched = TRUE,
  NA_keep = TRUE
))
show_palettes(
  list(pal1, pal2, pal3, pal4, pal5, pal6)
)

# Use Chinese color palettes
palette_colors(
  x = letters[1:5],
  palette = "ChineseRed",
  type = "discrete"
)
palette_colors(
  x = letters[1:5],
  palette = "Chinese",
  type = "discrete"
)

all_palettes <- show_palettes(return_palettes = TRUE)
names(all_palettes)
```

palette_list *A list of palettes for use in data visualization*

Description

A list of palettes for use in data visualization

panel_fix *Set the panel width/height of a plot to a fixed value*

Description

The ggplot object, when stored, can only specify the height and width of the entire plot, not the panel. The latter is obviously more important to control the final result of a plot. This function can set the panel width/height of plot to a fixed value and rasterize it.

Usage

```
panel_fix(
  x = NULL,
  panel_index = NULL,
  respect = NULL,
  width = NULL,
  height = NULL,
  margin = 1,
  padding = 0,
  units = "in",
  raster = FALSE,
  dpi = 300,
  return_grob = FALSE,
  bg_color = "white",
  save = NULL,
  verbose = FALSE,
  ...
)

panel_fix_overall(
  x,
  panel_index = NULL,
  respect = NULL,
  width = NULL,
  height = NULL,
  margin = 1,
  units = "in",
  raster = FALSE,
```

```
  dpi = 300,
  return_grob = FALSE,
  bg_color = "white",
  save = NULL,
  verbose = TRUE
)
```

Arguments

x	A ggplot object, a grob object, or a combined plot made by patchwork or cowplot package.
panel_index	Specify the panel to be fixed. If NULL, will fix all panels.
respect	Whether row heights and column widths should respect each other.
width	The desired width of the fixed panels.
height	The desired height of the fixed panels.
margin	The margin to add around each panel, in inches. Default is 1.
padding	The padding to add around each panel, in inches. Default is 0.
units	The units in which height, width and margin are given. Can be "mm", "cm", "in", etc. See grid::unit.
raster	Whether to rasterize the panel.
dpi	Plot resolution.
return_grob	Whether to return a grob object instead of a wrapped patchwork object. Default is FALSE.
bg_color	The background color of the plot.
save	NULL or the file name used to save the plot.
verbose	Whether to print the message. Default is TRUE.
...	Additional arguments passed to other functions.

Value

If return_grob is TRUE, returns a gtable object. Otherwise, returns a patchwork object with fixed panel sizes. The returned object has a size attribute containing width, height, and units.

Examples

```
library(ggplot2)
p <- ggplot(
  data = mtcars, aes(x = mpg, y = wt, colour = cyl)
) +
  geom_point() +
  facet_wrap(~gear, nrow = 2)
# fix the size of panel
panel_fix(
  p,
  width = 5,
  height = 3,
```

```
  units = "cm"
)
# rasterize the panel
panel_fix(
  p,
  width = 5,
  height = 3,
  units = "cm",
  raster = TRUE,
  dpi = 90
)

# `panel_fix` will build and render the plot when input a ggplot object.
# so after `panel_fix`, the size of the object will be changed.
object.size(p)
object.size(
  panel_fix(
    p,
    width = 5,
    height = 3,
    units = "cm"
  )
)
```

---

patchwork_grob *Convert a patchwork object to a grob*

Description

Convert a patchwork object to a gtable grob by processing annotations and building the patchwork layout.

Usage

```
patchwork_grob(x, ...)
```

Arguments

x	A patchwork object.
...	Additional arguments passed to other functions.

Value

A gtable object.

print.ChineseColors *Print* ChineseColors *object*

Description

Print ChineseColors object

Usage

```
## S3 method for class 'ChineseColors'
print(x, ...)
```

Arguments

x	A ChineseColors object.
...	Additional arguments.

Value

Details of the ChineseColors object.

print.colors *Print colors object*

Description

Print colors object

Usage

```
## S3 method for class 'colors'
print(x, ...)
```

Arguments

x	A colors object (data frame with color information).
...	Additional arguments passed to print.

Value

Details of the colors objec.

print.thisplot_logo *Print logo*

Description

Print logo

Usage

```
## S3 method for class 'thisplot_logo'
print(x, ...)
```

Arguments

x	Input information.
...	Other parameters.

Value

Print the ASCII logo

RGBA2RGB *Convert RGBA color to RGB with background*

Description

Convert an RGBA (Red, Green, Blue, Alpha) color to RGB by compositing it with a background color based on the alpha channel.

Usage

```
RGBA2RGB(RGBA, BackGround = c(1, 1, 1))
```

Arguments

RGBA	A list containing RGB values and alpha channel.
BackGround	The background RGB color to composite with. Default is c(1, 1, 1) (white).

Value

A numeric vector of RGB values.

segements_df *Shorten and offset the segment*

Description

This function takes a data frame representing segments in a plot and shortens and offsets them based on the provided arguments.

Usage

```
segements_df(data, shorten_start, shorten_end, offset)
```

Arguments

data	A data frame containing the segments. It should have columns 'x', 'y', 'xend', and 'yend' representing the start and end points of each segment.
shorten_start	The amount to shorten the start of each segment by.
shorten_end	The amount to shorten the end of each segment by.
offset	The amount to offset each segment by.

Value

The modified data frame with the shortened and offset segments.

Examples

```
library(ggplot2)
temp_nodes <- data.frame(
  "x" = c(10, 40),
  "y" = c(10, 30)
)
data <- data.frame(
  "x" = c(10, 40),
  "y" = c(10, 30),
  "xend" = c(40, 10),
  "yend" = c(30, 10)
)

ggplot(temp_nodes, aes(x = x, y = y)) +
  geom_point(size = 12) +
  xlim(0, 50) +
  ylim(0, 50) +
  geom_segment(
    data = data,
    aes(x = x, xend = xend, y = y, yend = yend)
  )

ggplot(temp_nodes, aes(x = x, y = y)) +
```

```
  geom_point(size = 12) +
  xlim(0, 50) +
  ylim(0, 50) +
  geom_segment(
    data = segements_df(
      data,
      shorten_start = 2,
      shorten_end = 3,
      offset = 1
    ),
    aes(x = x, xend = xend, y = y, yend = yend)
  )
```

`show_palettes` *Show the color palettes*

Description

This function displays color palettes using ggplot2.

Usage

```
show_palettes(
  palettes = NULL,
  type = c("discrete", "continuous"),
  index = NULL,
  palette_names = NULL,
  return_names = TRUE,
  return_palettes = FALSE
)
```

Arguments

`palettes`	A list of color palettes. Default is `NULL`.
`type`	The type of palettes to include. Default is `"discrete"`.
`index`	The indices of the palettes to include. Default is `NULL`.
`palette_names`	The names of the palettes to include. Default is `NULL`.
`return_names`	Whether to return the names of the selected palettes. Default is `TRUE`.
`return_palettes`	Whether to return the colors of selected palettes. Default is `FALSE`.

Value

If `return_palettes` is `TRUE`, returns a list of color palettes. If `return_names` is `TRUE` (default), returns a character vector of palette names. Otherwise, returns `NULL` (called for side effects to display the plot).

See Also

palette_colors, palette_list

Examples

```
show_palettes(
  palettes = list(
    c("red", "blue", "green"),
    c("yellow", "purple", "orange")
  )
)
all_palettes <- show_palettes(return_palettes = TRUE)
names(all_palettes)
all_palettes[["simspec"]]
show_palettes(index = 1:10)
show_palettes(
  type = "discrete",
  index = 1:10
)
show_palettes(
  type = "continuous",
  index = 1:10
)
show_palettes(
  palette_names = c(
    "Paired", "nejm", "simspec", "Spectral", "jet", "Chinese"
  ),
  return_palettes = TRUE
)
# Include Chinese palettes via prefix
show_palettes(
  palette_names = c("ChineseRed", "ChineseBlue"),
  return_palettes = TRUE
)
```

---

simple_colors *Simple random color selection*

Description

Simple random color selection

Usage

```
simple_colors(n = 10, palette = NULL)
```

Arguments

n	The number of colors to return. Default is 10.
palette	The name of the palette to use. Default is NULL, colors will be selected from ChineseColors. Otherwise, colors will be selected from the specified palette. Available palette names can be queried with show_palettes.

Value

A character vector of hexadecimal color codes.

Examples

```
simple_colors()

show_palettes(simple_colors(n = 5))

# Get colors from a specific palette
simple_colors(n = 10, palette = "Paired")
simple_colors(n = 10, palette = "ChineseBlue")
simple_colors(n = 10, palette = "Spectral")
```

---

slim_data *Slim unused data in the plot*

Description

Remove unused columns from the data in a ggplot or patchwork object. This function keeps only the columns that are actually used in the plot (e.g., in mappings, aesthetics, or facets), which can significantly reduce the object size when the original data contains many unused columns.

Usage

```
slim_data(p)

## S3 method for class 'ggplot'
slim_data(p)

## S3 method for class 'patchwork'
slim_data(p)
```

Arguments

p	A ggplot object or a patchwork object.

Value

A ggplot or patchwork object with unused data columns removed.

Examples

```
library(ggplot2)
p <- ggplot(
  data = mtcars,
  aes(x = mpg, y = wt, colour = cyl)
) +
  geom_point()
object.size(p)
colnames(p$data)

p_slim <- slim_data(p)
object.size(p_slim)
colnames(p_slim$data)
```

`standardise` *Standardize data by rows*

Description

Standardize each row of a data matrix by subtracting the mean and dividing by the standard deviation.

Usage

```
standardise(data)
```

Arguments

`data`	A matrix or data frame to standardize.

Value

The standardized data with the same structure as input.

`StatPlot` *Statistic Plot*

Description

Visualizes data using various plot types such as bar plots, rose plots, ring plots, pie charts, trend plots, area plots, dot plots, sankey plots, chord plots, venn diagrams, and upset plots.

Usage

```
StatPlot(
  meta.data,
  stat.by,
  group.by = NULL,
  split.by = NULL,
  bg.by = NULL,
  flip = FALSE,
  NA_color = "grey",
  NA_stat = TRUE,
  keep_empty = FALSE,
  individual = FALSE,
  stat_level = NULL,
 plot_type = c("bar", "rose", "ring", "pie", "trend", "area", "dot", "sankey", "chord",
    "venn", "upset"),
  stat_type = c("percent", "count"),
  position = c("stack", "dodge"),
  palette = "Chinese",
  palcolor = NULL,
  alpha = 1,
  bg_palette = "Chinese",
  bg_palcolor = NULL,
  bg_alpha = 0.2,
  label = FALSE,
  label.size = 3.5,
  label.fg = "black",
  label.bg = "white",
  label.bg.r = 0.1,
  aspect.ratio = NULL,
  title = NULL,
  subtitle = NULL,
  xlab = NULL,
  ylab = NULL,
  legend.position = "right",
  legend.direction = "vertical",
  theme_use = "theme_this",
  theme_args = list(),
  combine = TRUE,
  nrow = NULL,
  ncol = NULL,
  byrow = TRUE,
  force = FALSE,
  seed = 11
)
```

Arguments

meta.data	The data frame containing the data to be plotted.

stat.by	The column name(s) in meta.data specifying the variable(s) to be plotted.
group.by	The column name(s) in meta.data specifying the grouping variable(s). Default is NULL.
split.by	The column name in meta.data specifying the variable to split plots by. Default is NULL.
bg.by	The column name in meta.data specifying the background variable for bar plots.
flip	Whether to flip the plot. Default is FALSE.
NA_color	The color to use for missing values.
NA_stat	Whether to include missing values in the plot. Default is TRUE.
keep_empty	Whether to keep empty groups in the plot. Default is FALSE.
individual	Whether to plot individual groups separately. Default is FALSE.
stat_level	The level(s) of the variable(s) specified in stat.by to include in the plot. Default is NULL.
plot_type	The type of plot to create. Can be one of "bar", "rose", "ring", "pie", "trend", "area", "dot", "sankey", "chord", "venn", or "upset".
stat_type	The type of statistic to compute for the plot. Can be one of "percent" or "count".
position	The position adjustment for the plot. Can be one of "stack" or "dodge".
palette	The name of the color palette to use. Default is "Chinese".
palcolor	Custom colors to use instead of palette. Default is NULL.
alpha	The transparency level for the plot.
bg_palette	The name of the background color palette to use for bar plots.
bg_palcolor	The color to use in the background color palette.
bg_alpha	The transparency level for the background color in bar plots.
label	Whether to add labels on the plot. Default is FALSE.
label.size	The size of the labels.
label.fg	The foreground color of the labels.
label.bg	The background color of the labels.
label.bg.r	The radius of the rounded corners of the label background.
aspect.ratio	Aspect ratio of the panel. Default is NULL.
title	The title of the plot. Default is NULL.
subtitle	The subtitle of the plot. Default is NULL.
xlab	The label for the x-axis. Default is NULL.
ylab	The label for the y-axis. Default is NULL.
legend.position	The position of the legend. Can be one of "none", "left", "right", "bottom", "top", or a two-element numeric vector. Default is "right".

legend.direction
: The direction of the legend. Can be one of "vertical" or "horizontal". Default is "vertical".

theme_use
: The theme to use for the plot. Default is "theme_this".

theme_args
: Additional arguments to pass to the theme function. Default is list().

combine
: Whether to combine multiple plots into one. Default is TRUE.

nrow
: Number of rows when combining plots. Default is NULL.

ncol
: Number of columns when combining plots. Default is NULL.

byrow
: Whether to fill plots by row when combining. Default is TRUE.

force
: Whether to force plotting even when variables have more than 100 levels. Default is FALSE.

seed
: Random seed for reproducibility. Default is 11.

Examples

```
set.seed(1)
meta_data <- data.frame(
  Type = factor(
    sample(c("A", "B", "C"),
      50,
      replace = TRUE,
      prob = c(0.5, 0.3, 0.2)
    )
  ),
  Group = factor(sample(c("X", "Y", "Z"), 50, replace = TRUE)),
  Batch = factor(sample(c("B1", "B2"), 50, replace = TRUE))
)
meta_data$Region <- factor(
  ifelse(meta_data$Group %in% c("X", "Y"), "R1", "R2"),
  levels = c("R1", "R2")
)

StatPlot(
  meta_data,
  stat.by = "Type",
  group.by = "Group",
  split.by = "Batch",
  plot_type = "bar",
  position = "dodge"
)

StatPlot(
  meta_data,
  stat.by = "Type",
  group.by = "Group",
  stat_type = "count",
  plot_type = "ring",
  position = "dodge"
)
```

```
StatPlot(
  meta_data,
  stat.by = "Type",
  group.by = "Group",
  stat_type = "count"
)

StatPlot(
  meta_data,
  stat.by = "Type",
  plot_type = "pie"
)

StatPlot(
  meta_data,
  stat.by = "Type",
  group.by = "Group",
  stat_type = "count",
  plot_type = "area"
)

StatPlot(
  meta_data,
  stat.by = "Type",
  group.by = "Group",
  plot_type = "dot"
)

StatPlot(
  meta_data,
  stat.by = "Type",
  group.by = "Group",
  stat_type = "count",
  plot_type = "trend"
)
```

`theme_blank` *Blank theme*

Description

This function creates a theme with all elements blank except for axis lines and labels. It can optionally add coordinate axes in the plot.

Usage

```
theme_blank(
  add_coord = TRUE,
  xlen_npc = 0.15,
```

```
  ylen_npc = 0.15,
  xlab = "",
  ylab = "",
  lab_size = 12,
  ...
)
```

Arguments

add_coord	Whether to add coordinate arrows. Default is TRUE.
xlen_npc	The length of the x-axis arrow in "npc".
ylen_npc	The length of the y-axis arrow in "npc".
xlab	The label of the x-axis.
ylab	The label of the y-axis.
lab_size	The size of the axis labels.
...	Arguments passed to the ggplot2::theme.

Value

A list containing ggplot2 theme objects and annotation objects. If add_coord is TRUE, returns a list with coordinate arrows; otherwise returns a list with theme only.

Examples

```
library(ggplot2)
p <- ggplot(mtcars, aes(x = wt, y = mpg, colour = factor(cyl))) +
  geom_point()
p + theme_blank()
p + theme_blank(xlab = "x-axis", ylab = "y-axis", lab_size = 16)
```

---

theme_sankey *Themes for sankey plot*

Description

Minimal themes for sankey, alluvial and sankey bump plots

Usage

```
theme_sankey(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)
```

```
theme_alluvial(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)

theme_sankey_bump(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)
```

Arguments

base_size	Base font size, given in pts.
base_family	Base font family.
base_line_size	Base size for line elements.
base_rect_size	Base size for rect elements.

---

theme_this *The default theme for scop plot function.*

Description

The default theme for scop plot function.

Usage

```
theme_this(aspect.ratio = NULL, base_size = 12, ...)
```

Arguments

aspect.ratio	Aspect ratio of the panel.
base_size	Base font size
...	Arguments passed to the ggplot2::theme.

Value

A ggplot2 theme object (class theme, gg).

Examples

```
library(ggplot2)
p <- ggplot(
  data = mtcars,
  aes(x = wt, y = mpg, colour = factor(cyl))
) +
  geom_point()
p + theme_this()
```

`thisplot_logo` *The logo of thisplot*

Description

The thisplot logo, using ASCII or Unicode characters Use cli::ansi_strip to get rid of the colors.

Usage

```
thisplot_logo(unicode = cli::is_utf8_output())
```

Arguments

`unicode`	Unicode symbols on UTF-8 platforms. Default is cli::is_utf8_output.

Value

A character vector with class `thisplot_logo`.

References

https://github.com/tidyverse/tidyverse/blob/main/R/logo.R

Examples

```
thisplot_logo()
```

`visual_colors` *Visualize colors in HTML widget*

Description

Display a grid of color swatches with optional names or color codes.

Usage

```
visual_colors(colors, names = NULL, num_per_row = 30, title = NULL)
```

Arguments

`colors`	A character vector of hex color codes.
`names`	Optional. A character vector of names for each color. Default is `NULL`, which means hex color codes will be displayed. You can pass any labels (e.g., RGB values, custom names) via this parameter.
`num_per_row`	Number of colors per row. Default is `30`.
`title`	Optional title for the visualization. Default is `NULL`.

Value

An HTML widget.

Examples

```
# Visualize a simple color palette
visual_colors(
  colors = c("#FF0000", "#00FF00", "#0000FF"),
  names = c("Red", "Green", "Blue")
)

visual_colors(
  colors = c("#FF0000", "#00FF00"),
  names = c("(255, 0, 0)", "(0, 255, 0)")
)

visual_colors(thisplot::palette_list$Paired)

# Use with ChineseColors
cc <- ChineseColors()
visual_colors(
  colors = cc$blue[1:60],
  title = "Chinese Blue Colors"
)
```

Index