

# Package: thisplot (via r-universe)

June 9, 2026

**Type** Package

**Title** Utility Functions for Plotting

**Version** 0.4.0

**Date** 2026-05-24

**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, ggforce, geomtextpath, ggrepel, ggnewscale, ggplot2, grDevices, grid, gtable, igraph, methods, patchwork, proxyC, reshape2, stats, thisutils (>= 0.4.3), utils

**Suggests** circlize, ComplexHeatmap, dplyr, forcats, ggalluvial, ggupset, ggVennDiagram, htmltools, Matrix, png, purrr, ragg, scales, stringr, testthat (>= 3.0.0), tidyr, venny

**Config/Needs/website** mengxu98/thistemplate

**Config/testthat/edition** 3

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**Language** en-US

**LazyData** true

**Config/pak/sysreqs** libfontconfig1-dev libfreetype6-dev libfribidi-dev libglpk-dev libharfbuzz-dev libicu-dev libxml2-dev

**Repository** <https://mengxu98.r-universe.dev>

**Date/Publication** 2026-06-09 02:13:14 UTC

**RemoteUrl** <https://github.com/mengxu98/thisplot>

**RemoteRef** HEAD

**RemoteSha** fda4b75496da03c1b20fd78f59c9d8579e2961c8

## Contents

thisplot-package . . . . .	3
add_grob . . . . .	4
adjcolors . . . . .	4
adjustlayout . . . . .	5
annotate_quadrants . . . . .	6
annotation_block_fill_graphics . . . . .	7
annotation_block_graphics . . . . .	8
annotation_graphics . . . . .	8
as_grob . . . . .	9
as_gtable . . . . .	9
Blend2Color . . . . .	10
blendcolors . . . . .	10
BlendRGBList . . . . .	11
build_heatmap_annotation . . . . .	12
build_patchwork . . . . .	13
chinese_colors . . . . .	14
ChineseColors . . . . .	14
clip_symmetric_range . . . . .	16
cluster_within_group2 . . . . .	16
col2hex . . . . .	17
compute_velocity_on_grid . . . . .	18
DimDataPlot . . . . .	20
drop_data . . . . .	23
extractgrobs . . . . .	24
geom_alluvial . . . . .	25
geom_alluvial_label . . . . .	26
geom_sankey . . . . .	27
geom_sankey_bump . . . . .	29
geom_sankey_label . . . . .	30
get_chinese_palettes . . . . .	31
get_colors . . . . .	31
get_legend . . . . .	32
get_vars . . . . .	33
GraphPlot . . . . .	33
grid_draw . . . . .	38
head.colors . . . . .	38
heatmap_fixsize . . . . .	39
heatmap_rendersize . . . . .	40
jitter_highlighted_points . . . . .	41
make_long . . . . .	42
mestimate . . . . .	42

normalize_drawable . . . . .	43
palette_colors . . . . .	43
palette_list . . . . .	45
panel_fix . . . . .	46
patchwork_grob . . . . .	48
print.ChineseColors . . . . .	48
print.colors . . . . .	49
print.thisplot_logo . . . . .	49
RGBA2RGB . . . . .	50
segements_df . . . . .	50
show_palettes . . . . .	51
simple_colors . . . . .	53
slim_data . . . . .	54
standardise . . . . .	55
StatPlot . . . . .	55
theme_blank . . . . .	60
theme_sankey . . . . .	61
theme_this . . . . .	61
thisplot_logo . . . . .	62
to_percent . . . . .	63
visual_colors . . . . .	63

**Index** **65**

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

<code>gtable</code>	A gtable object.
<code>grob</code>	A grob or gtable object to add.
<code>position</code>	The position to add the grob. One of "top", "bottom", "left", "right", or "none".
<code>space</code>	The space to allocate for the grob. If NULL, will be calculated automatically.
<code>clip</code>	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

<code>colors</code>	Color vectors.
<code>alpha</code>	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.

---

annotate\_quadrants     *Annotate plot quadrants with percentages*

---

## Description

Annotate plot quadrants with percentages

## Usage

```
annotate_quadrants(  
  plot,  
  x,  
  y,  
  cutoffs,  
  line_color = "grey30",  
  line_type = "solid",  
  line_width = 0.5,  
  label_size = 3,  
  group = NULL  
)
```

## Arguments

plot	A 'ggplot' object.
x	Name of the x variable in 'plot\$data'.
y	Name of the y variable in 'plot\$data'.
cutoffs	Cut points for x and y axes. Can be a numeric vector of length 1 or 2, or a list with the x and y cut points.
line_color	Quadrant line color.
line_type	Quadrant line type.
line_width	Quadrant line width.
label_size	Label size.
group	Optional grouping column used to compute percentages within each group.

## Value

A 'ggplot' object.

## Examples

```
df <- data.frame(  
  x = c(0.1, 0.2, 0.8, 0.9),  
  y = c(0.1, 0.8, 0.2, 0.9)  
)  
p <- ggplot2::ggplot(df, ggplot2::aes(x = x, y = y)) +
```

```
ggplot2::geom_point()
p
annotate_quadrants(p, x = "x", y = "y", cutoffs = 0.5)
```

---

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 = TRUE
)
```

## Arguments

levels	Character/factor levels used as keys for fill mapping.
palette	Palette name. All available palette names can be queried with <a href="#">show_palettes</a> .
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	<i>Convert a plot object to a grob</i>
---------	--

---

**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	<i>Convert a plot object to a gtable</i>
-----------	--

---

**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	Whether to draw a rectangle border around the block.
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.

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$Chinese,
    color_sets$ChineseSet8,
    color_sets$ChineseSet16,
    color_sets$ChineseSet32
  )
)

# Use ChineseColors class
cc <- ChineseColors()
cc$visual_colors(
  loc_range = c(1, 180),
  num_per_row = 30,
  title = "Chinese Traditional Colors",
  name_type = "chinese"
)
```

---

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(
  loc_range = c(1, 180),
  num_per_row = 30,
  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"
)
```

---

clip\_symmetric\_range *Clip values to a symmetric range*

---

### Description

Clip values to a symmetric range

### Usage

```
clip_symmetric_range(data, value_col = "avg_log2FC")
```

### Arguments

data            A data frame.  
value\_col       Name of the numeric column to clip.

### Value

A list containing the clipped data frame and clipping limits.

### Examples

```
df <- data.frame(avg_log2FC = c(-10, -2, 0, 2, 10))
clip_symmetric_range(df)
```

---

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.

---

 compute\_velocity\_on\_grid

*Compute velocity on grid*


---

## Description

Compute velocity on grid

## Usage

```
compute_velocity_on_grid(
    x_emb,
    v_emb,
    density = 1,
    smooth = 0.5,
    n_neighbors = ceiling(n_obs/50),
    min_mass = 1,
    scale = 1,
    adjust_for_stream = FALSE,
    cutoff_perc = 5
)
```

## Arguments

x_emb	A matrix of dimension n_obs x n_dim specifying the embedding coordinates of the cells.
v_emb	A matrix of dimension n_obs x n_dim specifying the velocity vectors of the cells.
density	A numeric value specifying the density of the grid points along each dimension. Default is 1.
smooth	A numeric value specifying the smoothing factor for the velocity vectors. Default is 0.5.
n_neighbors	A numeric value specifying the number of nearest neighbors for each grid point. Default is ceiling(n_obs / 50).
min_mass	A numeric value specifying the minimum mass required for a grid point to be considered. Default is 1.
scale	A numeric value specifying the scaling factor for the velocity vectors. Default is 1.
adjust_for_stream	Whether to adjust the velocity vectors for streamlines. Default is FALSE.
cutoff_perc	A numeric value specifying the percentile cutoff for removing low-density grid points. Default is 5.

**Value**

A list with two components: `x_grid`, the grid coordinates, and `v_grid`, the smoothed velocity vectors on the grid.

**References**

[https://github.com/theislab/scvelo/blob/master/scvelo/plotting/velocity\\_embedding\\_grid.py](https://github.com/theislab/scvelo/blob/master/scvelo/plotting/velocity_embedding_grid.py)

**Examples**

```
x_emb <- matrix(
  c(
    0, 0,
    1, 0,
    0, 1,
    1, 1
  ),
  ncol = 2,
  byrow = TRUE
)
v_emb <- matrix(
  c(
    1, 0,
    1, 0,
    0, 1,
    0, 1
  ),
  ncol = 2,
  byrow = TRUE
)

velocity_grid <- compute_velocity_on_grid(
  x_emb = x_emb,
  v_emb = v_emb,
  density = 0.1,
  n_neighbors = 2,
  adjust_for_stream = TRUE
)

names(velocity_grid)
dim(velocity_grid$x_grid)
dim(velocity_grid$v_grid)
head(velocity_grid$x_grid)
head(velocity_grid$v_grid)

grid_df <- expand.grid(
  x = velocity_grid$x_grid[1, ],
  y = velocity_grid$x_grid[2, ]
)

plot_df <- data.frame(
```

```

x = grid_df$x,
y = grid_df$y,
xend = grid_df$x + c(velocity_grid$u_grid[1, , ]) * 0.2,
yend = grid_df$y + c(velocity_grid$v_grid[2, , ]) * 0.2
)

ggplot2::ggplot(plot_df) +
  ggplot2::geom_segment(
    ggplot2::aes(x = x, y = y, xend = xend, yend = yend),
    arrow = grid::arrow(length = grid::unit(0.12, "inches")),
    na.rm = TRUE
  ) +
  ggplot2::coord_equal()

```

---

DimDataPlot

*Data Dimensional Plot*


---

## Description

Visualize observations on precomputed two-dimensional coordinates.

## Usage

```

DimDataPlot(
  data,
  x,
  y,
  group.by = NULL,
  split.by = NULL,
  shape.by = NULL,
  show_na = FALSE,
  palette = "Chinese",
  palcolor = NULL,
  NA_color = "grey80",
  pt.size = 2,
  pt.alpha = 1,
  shape.values = NULL,
  add_mark = FALSE,
  mark_type = c("hull", "ellipse", "rect", "circle"),
  mark_expand = grid::unit(3, "mm"),
  mark_alpha = 0.1,
  mark_linetype = 1,
  label = FALSE,
  label.size = 4,
  label.fg = "black",
  label.bg = "white",
  label.bg.r = 0.1,
  label_repel = FALSE,

```

```

    label_repulsion = 20,
    label_point_size = 1,
    label_point_color = "black",
    label_segment_color = "black",
    add_origin = FALSE,
    origin.color = "grey30",
    origin.linetype = "dashed",
    origin.linewidth = 0.4,
    aspect.ratio = 1,
    title = NULL,
    subtitle = NULL,
    xlab = NULL,
    ylab = NULL,
    legend.position = "right",
    legend.direction = "vertical",
    legend.title = NULL,
    theme_use = "theme_this",
    theme_args = list(),
    seed = 11
  )

```

### Arguments

<code>data</code>	A data frame containing coordinates and optional grouping columns.
<code>x</code>	Name of the column used as the x-axis coordinate.
<code>y</code>	Name of the column used as the y-axis coordinate.
<code>group.by</code>	Name of the column used to color observations. Default is <code>NULL</code> .
<code>split.by</code>	Name of the column used to split the plot into facets. Default is <code>NULL</code> .
<code>shape.by</code>	Name of the column used to map point shapes. Default is <code>NULL</code> .
<code>show_na</code>	Whether to keep missing values in <code>group.by</code> , <code>split.by</code> , or <code>shape.by</code> as an explicit "NA" level. Default is <code>FALSE</code> .
<code>palette</code>	Color palette name. Available palettes can be found in <a href="#">show_palettes</a> . Default is "Chinese".
<code>palcolor</code>	Custom colors used to create a color palette. Default is <code>NULL</code> .
<code>NA_color</code>	Color used for missing values.
<code>pt.size</code>	Point size.
<code>pt.alpha</code>	Point transparency.
<code>shape.values</code>	Point shapes used when <code>shape.by</code> is set.
<code>add_mark</code>	Whether to add marks around groups. Default is <code>FALSE</code> .
<code>mark_type</code>	Type of mark to add around groups. One of "hull", "ellipse", "rect", or "circle". Default is "hull". "ellipse" uses <code>ggplot2::stat_ellipse</code> with <code>type = "norm"</code> and <code>level = 0.95</code> ; the other mark types use ggforce marks.
<code>mark_expand</code>	Expansion of the mark around each group. This is used for "hull", "rect", and "circle" marks. Default is <code>grid::unit(3, "mm")</code> .

<code>mark_alpha</code>	Transparency of the mark. Default is 0.1.
<code>mark_linetype</code>	Line type of the mark border. Default is 1.
<code>label</code>	Whether to label group centers.
<code>label.size</code>	Size of labels.
<code>label.fg</code>	Foreground color of labels.
<code>label.bg</code>	Background color of labels.
<code>label.bg.r</code>	Background ratio of labels.
<code>label_repel</code>	Whether labels repel from their group centers.
<code>label_repulsion</code>	Repulsion force for labels.
<code>label_point_size</code>	Size of center points for repelled labels.
<code>label_point_color</code>	Color of center points for repelled labels.
<code>label_segment_color</code>	Color of label segments.
<code>add_origin</code>	Whether to add dashed $x = 0$ and $y = 0$ reference lines.
<code>origin.color</code>	Color of origin reference lines.
<code>origin.linetype</code>	Line type of origin reference lines.
<code>origin.linewidth</code>	Line width of origin reference lines.
<code>aspect.ratio</code>	Aspect ratio passed to <code>theme()</code> .
<code>title</code>	Plot title.
<code>subtitle</code>	Plot subtitle.
<code>xlab</code>	Label for the x axis.
<code>ylab</code>	Label for the y axis.
<code>legend.position</code>	Position of the legend.
<code>legend.direction</code>	Direction of the legend.
<code>legend.title</code>	Title of the group legend.
<code>theme_use</code>	Theme function applied to the plot.
<code>theme_args</code>	A named list of arguments passed to <code>theme_use</code> .
<code>seed</code>	Random seed.

**Value**

A ggplot object.

**Examples**

```
set.seed(1)
plot_data <- data.frame(
  x = c(rnorm(40, -1), rnorm(40, 1)),
  y = c(rnorm(40, 0), rnorm(40, 1)),
  group = rep(c("A", "B"), each = 40)
)
DimDataPlot(
  plot_data,
  x = "x",
  y = "y",
  group.by = "group",
  add_mark = TRUE
)

DimDataPlot(
  plot_data,
  x = "x",
  y = "y",
  group.by = "group",
  add_mark = TRUE,
  add_origin = TRUE,
  mark_type = "ellipse"
)

DimDataPlot(
  plot_data,
  x = "x",
  y = "y",
  group.by = "group",
  add_mark = TRUE,
  mark_type = "circle",
  mark_alpha = 0.3,
  mark_expand = grid::unit(1, "mm"),
  mark_linetype = 2
)
```

---

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	<i>geom_alluvial</i>
---------------	----------------------

---

**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.

<code>width</code>	Width of nodes.
<code>smooth</code>	How much smooth should the curve have? More means steeper curve.
<code>inherit.aes</code>	Should the geom inherit aesthetics.
<code>...</code>	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'.

<code>width</code>	Width of nodes.
<code>inherit.aes</code>	Should the geom inherit aesthetics.
<code>...</code>	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**

<code>prefix</code>	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	<i>Get used vars in a ggplot object</i>
----------	---

---

**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)
```

---

GraphPlot	<i>Graph Plot</i>
-----------	-------------------

---

**Description**

A function to plot a graph with nodes and edges.

**Usage**

```
GraphPlot(  
  node,  
  edge,  
  transition = NULL,  
  node_coord = c("x", "y"),  
  node_group = NULL,  
  node_palette = "Chinese",  
  node_palcolor = NULL,  
  node_size = 4,  
  node_alpha = 1,  
  node_highlight = NULL,  
  node_highlight_color = "red",  
  label = FALSE,  
  label.size = 3.5,  
  label.fg = "white",  
  label.bg = "black",  
  label.bg.r = 0.1,  
  label_insitu = FALSE,  
  label_repel = FALSE,  
  label_repulsion = 20,  
  label_point_size = 1,  
  label_point_color = "black",  
  label_segment_color = "black",  
  edge_threshold = 0.01,  
  use_triangular = c("upper", "lower", "both"),  
  edge_line = c("straight", "curved"),  
  edge_line_curvature = 0.3,  
  edge_line_angle = 90,  
  edge_color = "grey30",  
  edge_size = c(0.2, 1),  
  edge_alpha = 0.5,  
  edge_shorten = 0,  
  edge_offset = 0,  
  edge_highlight = NULL,  
  edge_highlight_color = "red",  
  transition_threshold = 0.01,  
  transition_line = c("straight", "curved"),  
  transition_line_curvature = 0.3,  
  transition_line_angle = 90,  
  transition_color = "black",  
  transition_size = c(0.2, 1),  
  transition_alpha = 1,  
  transition_arrow_type = "closed",  
  transition_arrow_angle = 20,  
  transition_arrow_length = grid::unit(0.02, "npc"),  
  transition_shorten = 0.05,  
  transition_offset = 0,
```

```

transition_highlight = NULL,
transition_highlight_color = "red",
aspect.ratio = 1,
title = NULL,
subtitle = NULL,
xlab = NULL,
ylab = NULL,
legend.position = "right",
legend.direction = "vertical",
theme_use = "theme_this",
theme_args = list(),
return_layer = FALSE
)

```

### Arguments

node	A data frame representing the nodes of the graph.
edge	A matrix representing the edges of the graph.
transition	A matrix representing the transitions between nodes.
node_coord	A character vector specifying the names of the columns in node that represent the x and y coordinates.
node_group	A character vector specifying the name of the column in node that represents the grouping of the nodes.
node_palette	A character vector specifying the name of the color palette for node groups.
node_palcolor	A character vector specifying the names of the colors for each node group.
node_size	A numeric value or column name of node specifying the size of the nodes.
node_alpha	A numeric value or column name of node specifying the transparency of the nodes.
node_highlight	A character vector specifying the names of nodes to highlight.
node_highlight_color	A character vector specifying the color for highlighting nodes.
label	Whether to show labels for the nodes.
label.size	The size of the labels.
label.fg	A character vector specifying the foreground color of the labels.
label.bg	A character vector specifying the background color of the labels.
label.bg.r	The background color transparency of the labels.
label.insitu	Whether to display the node group labels in situ or as numeric values.
label.repel	Whether to use force-directed label repulsion.
label.repulsion	The repulsion force for labels.
label.point.size	The size of the label points.

label_point_color	A character vector specifying the color of the label points.
label_segment_color	A character vector specifying the color for the label segments.
edge_threshold	The threshold for removing edges.
use_triangular	A character vector specifying which part of the edge matrix to use (upper, lower, both).
edge_line	A character vector specifying the type of line for edges (straight, curved).
edge_line_curvature	The curvature of curved edges.
edge_line_angle	The angle of curved edges.
edge_color	A character vector specifying the color of the edges.
edge_size	A numeric vector specifying the range of edge sizes.
edge_alpha	The transparency of the edges.
edge_shorten	The length of the edge shorten.
edge_offset	The length of the edge offset.
edge_highlight	A character vector specifying the names of edges to highlight.
edge_highlight_color	A character vector specifying the color for highlighting edges.
transition_threshold	The threshold for removing transitions.
transition_line	A character vector specifying the type of line for transitions (straight, curved).
transition_line_curvature	The curvature of curved transitions.
transition_line_angle	The angle of curved transitions.
transition_color	A character vector specifying the color of the transitions.
transition_size	A numeric vector specifying the range of transition sizes.
transition_alpha	The transparency of the transitions.
transition_arrow_type	A character vector specifying the type of arrow for transitions (closed, open).
transition_arrow_angle	The angle of the transition arrow.
transition_arrow_length	The length of the transition arrow.
transition_shorten	The length of the transition shorten.

`transition_offset`      The length of the transition offset.  
`transition_highlight`    A character vector specifying the names of transitions to highlight.  
`transition_highlight_color`    A character vector specifying the color for highlighting transitions.  
`aspect.ratio`      Aspect ratio passed to `theme()`.  
`title`              Plot title.  
`subtitle`          Plot subtitle.  
`xlab`              Label for the x axis.  
`ylab`              Label for the y axis.  
`legend.position`      Legend position passed to `theme()`.  
`legend.direction`      Legend direction passed to `theme()`.  
`theme_use`          Theme function applied to the plot.  
`theme_args`        A named list of arguments passed to `theme_use`.  
`return_layer`      Whether to return the plot layers as a list. Defaults is `FALSE`.

### Examples

```

node <- data.frame(
  x = c(0.2, 0.3, 0.5, 0.3),
  y = c(0.2, 0, 0.6, 0.7),
  group = c("A", "B", "C", "B"),
  row.names = c("n1", "n2", "n3", "n4")
)
edge <- matrix(
  c(
    0, 0.4, 0.2, 0.1,
    0.4, 0, 0.3, 0.2,
    0.2, 0.3, 0, 0.5,
    0.2, 0.2, 0.5, 0.3
  ),
  nrow = 4,
  byrow = TRUE,
  dimnames = list(rownames(node), rownames(node))
)
GraphPlot(
  node = node,
  edge = edge
)

GraphPlot(
  node = node,
  edge = edge,
  node_group = "group",
  node_highlight = "n1",

```

```

    label = TRUE,
    label_repel = TRUE
  )

```

---

grid_draw	<i>Draw grobs at specified positions</i>
-----------	--

---

### 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	<i>Return the first part of a colors object</i>
-------------	---

---

### 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 <a href="#">head()</a> .

**Value**

A colors object with the first n rows.

**Examples**

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

---

heatmap_fixsize	<i>Compute fixed heatmap device size</i>
-----------------	--

---

**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
```

---

`jitter_highlighted_points`

*Jitter highlighted points deterministically*

---

**Description**

Jitter highlighted points deterministically

**Usage**

```
jitter_highlighted_points(
  data,
  jitter_width = 0.2,
  jitter_height = 0.2,
  seed = 11
)
```

**Arguments**

<code>data</code>	A data frame containing ‘x’, ‘y’, and ‘border’ columns.
<code>jitter_width</code>	Width of the x jitter.
<code>jitter_height</code>	Height of the y jitter.
<code>seed</code>	Numeric value used to offset the deterministic sequence.

**Value**

A data frame with ‘x\_plot’ and ‘y\_plot’ columns.

**Examples**

```
df <- data.frame(
  x = c(0, 1),
  y = c(0, 1),
  border = c(FALSE, TRUE)
)
jitter_highlighted_points(df, jitter_width = 0.2, jitter_height = 0.2)
```

---

make_long	<i>Make a long data frame for sankey plot</i>
-----------	---

---

**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	<i>Estimate the fuzzifier parameter m</i>
-----------	---

---

**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 = "Chinese",
  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 <a href="#">show_palettes</a> .
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	<i>Set the panel width/height of a plot to a fixed value</i>
-----------	--

---

## 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 <a href="#">grid::unit</a> .
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 = segments_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 <a href="#">show_palettes</a> .

## 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	<i>Standardize data by rows</i>
-------------	---------------------------------

---

**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	<i>Statistic Plot</i>
----------	-----------------------

---

**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", "trend_alluvial", "area",
    "dot", "sankey", "chord", "venn", "upset"),
  venn_engine = c("ggVennDiagram", "venny"),
  venn_args = list(),
```

```

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(),
grid_major = TRUE,
grid_major_colour = "grey80",
grid_major_linetype = 2,
grid_major_linewidth = 0.3,
combine = TRUE,
nrow = NULL,
ncol = NULL,
byrow = TRUE,
force = FALSE,
seed = 11
)

```

### Arguments

<code>meta.data</code>	The data frame containing the data to be plotted.
<code>stat.by</code>	The column name(s) in <code>meta.data</code> specifying the variable(s) to be plotted.
<code>group.by</code>	The column name(s) in <code>meta.data</code> specifying the grouping variable(s). Default is <code>NULL</code> .
<code>split.by</code>	The column name in <code>meta.data</code> specifying the variable to split plots by. Default is <code>NULL</code> .
<code>bg.by</code>	The column name in <code>meta.data</code> specifying the background variable for bar plots.
<code>flip</code>	Whether to flip the plot. Default is <code>FALSE</code> .
<code>NA_color</code>	The color to use for missing values.
<code>NA_stat</code>	Whether to include missing values in the plot. Default is <code>TRUE</code> .

<code>keep_empty</code>	Whether to keep empty groups in the plot. Default is FALSE.
<code>individual</code>	Whether to plot individual groups separately. Default is FALSE.
<code>stat_level</code>	The level(s) of the variable(s) specified in <code>stat.by</code> to include in the plot. Default is NULL.
<code>plot_type</code>	The type of plot to create. Can be one of "bar", "rose", "ring", "pie", "trend", "trend_alluvial", "area", "dot", "sankey", "chord", "venn", or "upset".
<code>venn_engine</code>	The engine used when <code>plot_type = "venn"</code> . Can be one of "ggVennDiagram" or "venny". Default is "ggVennDiagram". The "venny" engine supports 2 to 4 sets.
<code>venn_args</code>	A list of additional arguments passed to <code>venny::venny()</code> when <code>plot_type = "venn"</code> and <code>venn_engine = "venny"</code> . The data argument is generated internally and cannot be supplied here.
<code>stat_type</code>	The type of statistic to compute for the plot. Can be one of "percent" or "count".
<code>position</code>	The position adjustment for the plot. Can be one of "stack" or "dodge".
<code>palette</code>	The name of the color palette to use. Default is "Chinese".
<code>palcolor</code>	Custom colors to use instead of palette. Default is NULL.
<code>alpha</code>	The transparency level for the plot.
<code>bg_palette</code>	The name of the background color palette to use for bar plots.
<code>bg_palcolor</code>	The color to use in the background color palette.
<code>bg_alpha</code>	The transparency level for the background color in bar plots.
<code>label</code>	Whether to add labels on the plot. Default is FALSE.
<code>label.size</code>	The size of the labels.
<code>label.fg</code>	The foreground color of the labels.
<code>label.bg</code>	The background color of the labels.
<code>label.bg.r</code>	The radius of the rounded corners of the label background.
<code>aspect.ratio</code>	Aspect ratio of the panel. Default is NULL.
<code>title</code>	The title of the plot. Default is NULL.
<code>subtitle</code>	The subtitle of the plot. Default is NULL.
<code>xlab</code>	The label for the x-axis. Default is NULL.
<code>ylab</code>	The label for the y-axis. Default is NULL.
<code>legend.position</code>	The position of the legend. Can be one of "none", "left", "right", "bottom", "top", or a two-element numeric vector. Default is "right".
<code>legend.direction</code>	The direction of the legend. Can be one of "vertical" or "horizontal". Default is "vertical".
<code>theme_use</code>	The theme to use for the plot. Default is "theme_this".
<code>theme_args</code>	Additional arguments to pass to the theme function. Default is <code>list()</code> .

**grid\_major** Whether to show major panel grid lines. Default is TRUE.  
**grid\_major\_colour** Color of major panel grid lines.  
**grid\_major\_linetype** Linetype of major panel grid lines.  
**grid\_major\_linewidth** Line width of major panel grid lines.  
**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"
  )

  meta_data$A <- meta_data$Type == "A"
  meta_data$B <- meta_data$Group == "X"
  meta_data$C <- meta_data$Batch == "B1"
  StatPlot(
    meta_data,
    stat.by = c("A", "B", "C"),
    stat_level = TRUE,
    plot_type = "venn",
    venn_engine = "venny",
    venn_args = list(detail = TRUE)
  )
```

---

theme_blank	<i>Blank theme</i>
-------------	--------------------

---

### 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 <a href="#">ggplot2::theme</a> .

### 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	<i>Themes for sankey plot</i>
--------------	-------------------------------

---

### 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	<i>The default theme for scop plot function.</i>
------------	--

---

### 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()
```

---

to_percent	<i>Convert proportions to percentages</i>
------------	---

---

**Description**

Convert proportions to percentages

**Usage**

```
to_percent(x)
```

**Arguments**

x                    A numeric vector.

**Value**

A numeric vector expressed as percentages.

**Examples**

```
to_percent(c(1, 1, 2))
to_percent(c(0, 0))
```

---

visual_colors	<i>Visualize colors in HTML widget</i>
---------------	--

---

**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,  
  label_mode = c("auto", "chinese", "pinyin", "rgb", "hex")  
)
```

**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.
label_mode	Label layout mode. Default is "auto", which detects the label type from names. Use "chinese", "pinyin", "rgb", or "hex" to force a layout optimized for that label style.

**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

- \* **data**
  - chinese\_colors, 14
  - palette\_list, 45
- add\_grob, 4
- adjcolors, 4
- adjustlayout, 5
- annotate\_quadrants, 6
- annotation\_block\_fill\_graphics, 7
- annotation\_block\_graphics, 8
- annotation\_graphics, 8
- as\_grob, 9
- as\_gtable, 9
  
- Blend2Color, 10
- blendcolors, 10
- BlendRGBList, 11
- build\_heatmap\_annotation, 12
- build\_patchwork, 13
  
- chinese\_colors, 14, 15, 32
- ChineseColors, 14, 32
- cli::ansi\_strip, 62
- cli::is\_utf8\_output, 62
- clip\_symmetric\_range, 16
- cluster\_within\_group2, 16
- col2hex, 17
- ComplexHeatmap::anno\_block, 7, 8
- ComplexHeatmap::anno\_customize, 9
- ComplexHeatmap::HeatmapAnnotation, 12
- compute\_velocity\_on\_grid, 18
  
- DimDataPlot, 20
- drop\_data, 23
  
- extractgrobs, 24
  
- geom\_alluvial, 25
- geom\_alluvial\_label, 26
- geom\_alluvial\_text
  - (geom\_alluvial\_label), 26
- geom\_sankey, 27
- geom\_sankey\_bump, 29
- geom\_sankey\_label, 30
- geom\_sankey\_text (geom\_sankey\_label), 30
- get\_chinese\_palettes, 15, 31, 32
- get\_colors, 15, 31
- get\_legend, 32
- get\_vars, 33
- ggplot2::stat\_ellipse, 21
- ggplot2::theme, 60, 62
- GraphPlot, 33
- grid::unit, 47
- grid\_draw, 38
  
- head(), 38
- head.colors, 38
- heatmap\_fixsize, 39
- heatmap\_rendersize, 40
  
- jitter\_highlighted\_points, 41
  
- make\_long, 42
- mestimate, 42
  
- normalize\_drawable, 43
  
- palette\_colors, 43, 52
- palette\_list, 44, 45, 52
- panel\_fix, 46
- panel\_fix\_overall (panel\_fix), 46
- patchwork\_grob, 48
- print.ChineseColors, 48
- print.ChineseColors(), 15
- print.colors, 49
- print.colors(), 32
- print.thisplot\_logo, 49
  
- RGBA2RGB, 50
- segments\_df, 50
- show\_palettes, 7, 21, 44, 51, 53

simple\_colors, [53](#)  
slim\_data, [54](#)  
standardise, [55](#)  
StatPlot, [55](#)

theme\_alluvial (theme\_sankey), [61](#)  
theme\_blank, [60](#)  
theme\_sankey, [61](#)  
theme\_sankey\_bump (theme\_sankey), [61](#)  
theme\_this, [61](#)  
thisplot (thisplot-package), [3](#)  
thisplot-package, [3](#)  
thisplot\_logo, [62](#)  
to\_percent, [63](#)

venny::venny(), [57](#)  
visual\_colors, [15](#), [63](#)