Package: ggalign (via r-universe)

March 8, 2025

Title A 'ggplot2' Extension for Consistent Axis Alignment

Version 0.1.0

Description A 'ggplot2' extension offers various tools for organizing and arranging plots. It is designed to consistently align a specific axis across multiple 'ggplot' objects, making it especially useful for plots requiring data order manipulation. A typical use case includes organizing combinations like a dendrogram and a heatmap.

License MIT + file LICENSE

URL https://github.com/Yunuuuu/ggalign, https://yunuuuu.github.io/ggalign/

BugReports https://github.com/Yunuuuu/ggalign/issues

Depends ggplot2 (>= 3.3.0)

Imports cli, grDevices, grid, gtable, lifecycle, methods, rlang (>= 1.1.0), stats, utils, vctrs (>= 0.5.0)

Suggests ggrastr, gridGraphics, magick, patchwork, ragg, knitr, rmarkdown, scales, testthat (>= 3.0.0), vdiffr

ByteCompile true

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

Language en-GB

Collate 'active.R' 'plot-.R' 'align-.R' 'align-hclust.R' 'align-dendrogram.R' 'align-group.R' 'align-kmeans.R' 'align-order.R' 'align-phylo.R' 'align-reorder.R' 'alignpatch-.R' 'alignpatch-align_plots.R' 'alignpatch-alignpatches.R' 'alignpatch-area.R' 'alignpatch-build.R' 'alignpatch-free-align.R' 'alignpatch-free-border.R' 'alignpatch-free-guide.R' 'alignpatch-free-lab.R' 'alignpatch-free-space.R' 'alignpatch-free-vp.R' 'alignpatch-ggplot2.R'

'alignpatch-guides.R' 'alignpatch-inset.R'
'alignpatch-patchwork.R' 'alignpatch-title.R'
'alignpatch-wrap.R' 'cross-.R' 'cross-link.R' 'cross-mark.R'
'cross-none.R' 'dendrogram.R' 'fortify-data-frame.R'
'fortify-matrix.R' 'geom-draw.R' 'geom-draw2.R' 'geom-pie.R'
'geom-subrect.R' 'geom-tile3d.R' 'ggalign-package.R'
'ggalign.R' 'ggcross.R' 'ggfree.R' 'ggmark.R' 'ggplot-helper.R'
'ggplot-theme.R' 'ggplot-utils.R' 'import-standalone-assert.R'
'import-standalone-obj-type.R' 'import-standalone-purrr.R'
'import-standalone-tibble.R' 'layer-order.R' 'layout-.R'
'layout-align.R' 'layout-chain-.R' 'layout-chain-operator.R'
'layout-circle-.R' 'layout-circle-build.R'
'layout-circle-switch.R' 'layout-quad-.R' 'layout-heatmap-.R'
'layout-heatmap-build.R' 'layout-heatmap-oncoplot.R'
'layout-quad-operator.R' 'layout-operator.R'
'layout-quad-add.R' 'layout-quad-build.R'
'layout-quad-switch.R' 'layout-quad-upset.R' 'layout-stack-.R'
'layout-stack-build.R' 'layout-stack-composer.R'
'layout-stack-cross.R' 'layout-stack-switch.R' 'link.R'
'maftools.R' 'mark.R' 'object-name.R' 'pair-links.R'
'raster-magick.R' 'rasterise.R' 'scheme-.R' 'scheme-align.R'
'scheme-data.R' 'scheme-theme.R' 'tune.R' 'utils-assert.R'
'utils-grid.R' 'utils-rd.R' 'utils.R' 'with_quad.R' 'zzz.R'

NeedsCompilation no

Author Yun Peng [aut, cre] (<https://orcid.org/0000-0003-2801-3332>),
Shixiang Wang [aut] (<https://orcid.org/0000-0001-9855-7357>),
Guangchuang Yu [aut] (<https://orcid.org/0000-0002-6485-8781>)

Maintainer Yun Peng <yunyunp96@163.com>

Repository CRAN

Date/Publication 2025-02-06 12:10:03 UTC

Contents

.link_draw *Define the links to connect a pair of observations*

Description

A base version of link_draw(), optimized for performance. This function serves as the foundation for building other link_* functions that manage the drawing of links between pairs of observations.

Usage

```
.link_draw(.draw, ...)
```

Arguments

.draw	A function used to draw the links. The function must return a grob() object. If the function does not return a valid grob, no drawing will occur. The input data for the function should be a list, where each item is a data frame containing the coordinates of the pair of observations.
...	<dyn-dots> A list of formulas, where each side of the formula should be an integer or character index of the original data, or a range_link() object defining the linked observations. Use NULL to indicate no link on that side. You can also combine these by wrapping them into a single list(). If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with I() to use the ordering from the layout. You can also use waiver() to inherit values from the other group.

See Also

link_draw()

.mark_draw *Define the links to connect the marked observations*

Description

A base version of mark_draw, designed for performance optimization. This function is used to build other mark_* functions that manage the drawing of links between marked observations.

Usage

```
.mark_draw(.draw, ...)
```

Arguments

.draw	A function used to draw the links. The function must return a grob() object. If the function does not return a valid grob, nothing will be drawn. The input data for the function contains a list, where each item is a list of two data frames: one for the panel side coordinates ("panel") and one for the marked observations coordinates ("link").
...	<dyn-dots> A list of formulas, where each side of the formula should be an integer or character index of the original data, or a range_link() object defining the linked observations. Use NULL to indicate no link on that side. You can also combine these by wrapping them into a single list(). If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with I() to use the ordering from the layout. You can also use waiver() to inherit values from the other group.

See Also

mark_draw()

---

active *Plot Adding Context Settings*

Description

[Experimental]

These settings control the behavior of the plot when added to a layout, as well as the arrangement of individual plot areas within the layout.

Usage

```
active(order = waiver(), use = waiver(), name = waiver())
```

Arguments

order	An integer specifying the order of the plot area within the layout.
use	A logical (TRUE/FALSE) indicating whether to set the active context to the current plot when added to a layout. If TRUE, any subsequent ggplot elements will be applied to this plot.
name	A string specifying the plot's name, useful for switching active contexts through the what argument in functions like quad_anno()/stack_switch().

Details

By default, the active context is set only for functions that add plot areas. This allows other ggplot2 elements-such as geoms, stats, scales, or themes- to be seamlessly added to the current plot area.

The default ordering of the plot areas is from top to bottom or from left to right, depending on the layout orientation. However, users can customize this order using the order argument.

`align_dendro` *Plot dendrogram tree*

Description

Plot dendrogram tree

Usage

```
align_dendro(
  mapping = aes(),
  ...,
  distance = "euclidean",
  method = "complete",
  use_missing = "pairwise.complete.obs",
  reorder_dendrogram = FALSE,
  merge_dendrogram = FALSE,
  reorder_group = FALSE,
  k = NULL,
  h = NULL,
  cutree = NULL,
  plot_dendrogram = TRUE,
  plot_cut_height = NULL,
  root = NULL,
  center = FALSE,
  type = "rectangle",
  size = NULL,
  data = NULL,
  no_axes = NULL,
  active = NULL
)
```

Arguments

`mapping`	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
`...`	<dyn-dots> Additional arguments passed to `geom_segment()`.
`distance`	A string of distance measure to be used. This must be one of `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"` or `"minkowski"`. Correlation coefficient can be also used, including `"pearson"`, `"spearman"` or `"kendall"`. In this way, `1 - cor` will be used as the distance. In addition, you can also provide a `dist` object directly or a function return a `dist` object. Use `NULL`, if you don't want to calculate the distance.
`method`	A string of the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"` (= UPGMA), `"mcquitty"` (= WPGMA), `"median"` (= WPGMC)

or "centroid" (= UPGMC). You can also provide a function which accepts the calculated distance (or the input matrix if distance is NULL) and returns a hclust object. Alternative, you can supply an object which can be coerced to hclust.

use_missing An optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs". Only used when distance is a correlation coefficient string.

reorder_dendrogram
A single boolean value indicating whether to reorder the dendrogram based on the means. Alternatively, you can provide a custom function that accepts an hclust object and the data used to generate the tree, returning either an hclust or dendrogram object. Default is FALSE.

merge_dendrogram
A single boolean value, indicates whether we should merge multiple dendrograms, only used when previous groups have been established. Default: FALSE.

reorder_group A single boolean value, indicates whether we should do Hierarchical Clustering between groups, only used when previous groups have been established. Default: FALSE.

k An integer scalar indicates the desired number of groups.

h A numeric scalar indicates heights where the tree should be cut.

cutree A function used to cut the hclust tree. It should accept four arguments: the hclust tree object, distance (only applicable when method is a string or a function for performing hierarchical clustering), k (the number of clusters), and h (the height at which to cut the tree). By default, cutree() is used.

plot_dendrogram
A boolean value indicates whether plot the dendrogram tree.

plot_cut_height
A boolean value indicates whether plot the cut height.

root A length one string or numeric indicates the root branch.

center A boolean value. if TRUE, nodes are plotted centered with respect to all leaves/tips in the branch. Otherwise (default), plot them in the middle of the direct child nodes.

type A string indicates the plot type, "rectangle" or "triangle".

size The relative size of the plot, can be specified as a unit(). Note that for circle_layout(), all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.

data A matrix-like object. By default, it inherits from the layout matrix.

no_axes **[Experimental]** Logical; if TRUE, removes axes elements for the alignment axis using theme_no_axes(). By default, will use the option- "ggalign.align_no_axes".

active A active() object that defines the context settings when added to a layout.

ggplot2 specification

`align_dendro` initializes a ggplot `data` and `mapping`.

The internal `ggplot` object will always use a default mapping of `aes(x = .data$x, y = .data$y)`.

The default ggplot data is the `node` coordinates with `edge` data attached in `ggalign` attribute, in addition, a `geom_segment` layer with a data frame of the `edge` coordinates will be added when `plot_dendrogram = TRUE`.

See `fortify_data_frame.dendrogram()` for details.

Discrete Axis Alignment

It is important to note that we consider rows as observations, meaning `vec_size(data)`/`NROW(data)` must match the number of observations along the axis used for alignment (x-axis for a vertical stack layout, y-axis for a horizontal stack layout).

Examples

```
# align_dendro will always add a plot area
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_top() +
    align_dendro()
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_top() +
    align_dendro(k = 3L)
```

---

align_group *Group and align observations based on a group vector*

Description

[Stable]

Splits observations into groups, with slice ordering based on group levels.

Usage

```
align_group(group, active = NULL)
```

Arguments

`group`	A character define the groups of the observations.
`active`	A `active()` object that defines the context settings when added to a layout.

Examples

```
set.seed(1L)
small_mat <- matrix(rnorm(81), nrow = 9)
ggheatmap(small_mat) +
    anno_top() +
    align_group(sample(letters[1:4], ncol(small_mat), replace = TRUE))
```

`align_hclust` *Reorder or Group observations based on hierarchical clustering*

Description

[Stable]

This function aligns observations within the layout according to a hierarchical clustering tree, enabling reordering or grouping of elements based on clustering results.

Usage

```
align_hclust(
  distance = "euclidean",
  method = "complete",
  use_missing = "pairwise.complete.obs",
  reorder_dendrogram = FALSE,
  reorder_group = FALSE,
  k = NULL,
  h = NULL,
  cutree = NULL,
  data = NULL,
  active = NULL
)
```

Arguments

`distance`	A string of distance measure to be used. This must be one of `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"` or `"minkowski"`. Correlation coefficient can be also used, including `"pearson"`, `"spearman"` or `"kendall"`. In this way, `1 - cor` will be used as the distance. In addition, you can also provide a `dist` object directly or a function return a `dist` object. Use `NULL`, if you don't want to calculate the distance.
`method`	A string of the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of `"ward.D"`, `"ward.D2"`, `"single"`, `"complete"`, `"average"` (= UPGMA), `"mcquitty"` (= WPGMA), `"median"` (= WPGMC) or `"centroid"` (= UPGMC). You can also provide a function which accepts the calculated distance (or the input matrix if `distance` is `NULL`) and returns a `hclust` object. Alternative, you can supply an object which can be coerced to `hclust`.

use_missing — An optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs". Only used when distance is a correlation coefficient string.

reorder_dendrogram — A single boolean value indicating whether to reorder the dendrogram based on the means. Alternatively, you can provide a custom function that accepts an hclust object and the data used to generate the tree, returning either an hclust or dendrogram object. Default is FALSE.

reorder_group — A single boolean value, indicates whether we should do Hierarchical Clustering between groups, only used when previous groups have been established. Default: FALSE.

k — An integer scalar indicates the desired number of groups.

h — A numeric scalar indicates heights where the tree should be cut.

cutree — A function used to cut the hclust tree. It should accept four arguments: the hclust tree object, distance (only applicable when method is a string or a function for performing hierarchical clustering), k (the number of clusters), and h (the height at which to cut the tree). By default, cutree() is used.

data — A matrix-like object. By default, it inherits from the layout matrix.

active — A active() object that defines the context settings when added to a layout.

Discrete Axis Alignment

It is important to note that we consider rows as observations, meaning vec_size(data)/NROW(data) must match the number of observations along the axis used for alignment (x-axis for a vertical stack layout, y-axis for a horizontal stack layout).

See Also

hclust2()

Examples

```
# align_hclust won't add a dendrogram
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_top() +
    align_hclust(k = 3L)
```

---

align_kmeans *Split observations by k-means clustering groups.*

Description

[Stable]

Aligns and groups observations based on k-means clustering, enabling observation splits by cluster groups.

Usage

```
align_kmeans(..., data = NULL, active = NULL)
```

Arguments

...	Arguments passed on to stats::kmeans iter.max the maximum number of iterations allowed. nstart if centers is a number, how many random sets should be chosen? algorithm character: may be abbreviated. Note that "Lloyd" and "Forgy" are alternative names for one algorithm. trace logical or integer number, currently only used in the default method ("Hartigan-Wong"): if positive (or true), tracing information on the progress of the algorithm is produced. Higher values may produce more tracing information.
data	A numeric matrix to be used by k-means. By default, it will inherit from the layout matrix.
active	A active() object that defines the context settings when added to a layout.

Discrete Axis Alignment

It is important to note that we consider rows as observations, meaning vec_size(data)/NROW(data) must match the number of observations along the axis used for alignment (x-axis for a vertical stack layout, y-axis for a horizontal stack layout).

Examples

```
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_top() +
    align_kmeans(3L)
```

---

align_order *Order observations based on weights*

Description

[Stable]

Ordering observations based on summary weights or a specified ordering character or integer index.

Usage

```
align_order(
  weights = rowMeans,
  ...,
  reverse = FALSE,
  strict = TRUE,
  data = NULL,
  active = NULL
)
```

Arguments

weights	A summary function which accepts a data and returns the weights for each observations. Alternatively, you can provide an ordering index as either an integer or a character. Since characters have been designated as character indices, if you wish to specify a function name as a string, you must enclose it with I().
...	<dyn-dots> Additional arguments passed to function provided in weights argument.
reverse	A boolean value. Should the sort order be in reverse?
strict	A boolean value indicates whether the order should be strict. If previous groups has been established, and strict is FALSE, this will reorder the observations in each group.
data	A matrix, data frame, or atomic vector used as the input for the weights function. Alternatively, you can specify a function (including purrr-like lambda syntax) that will be applied to the layout matrix, transforming it as necessary for weight calculations. By default, it will inherit from the layout matrix.
active	A active() object that defines the context settings when added to a layout.

Discrete Axis Alignment

It is important to note that we consider rows as observations, meaning vec_size(data)/NROW(data) must match the number of observations along the axis used for alignment (x-axis for a vertical stack layout, y-axis for a horizontal stack layout).

Examples

```
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_left() +
    align_order(I("rowMeans"))
```

---

align_phylo *Plot Phylogenetics tree*

Description

Plot Phylogenetics tree

Usage

```
align_phylo(
  phylo,
  ...,
  ladderize = NULL,
  type = "rectangle",
  center = FALSE,
  tree_type = NULL,
```

```
  no_axes = NULL,
  active = NULL,
  size = NULL
)
```

Arguments

phylo	A phylo object.
...	<dyn-dots> Additional arguments passed to geom_segment().
ladderize	A single string of "left" or "right", indicating whether to ladderize the tree. Ladderizing arranges the tree so that the smallest clade is positioned on the "right" or the "left". By default, NULL means the tree will not be ladderized.
type	A string indicates the plot type, "rectangle" or "triangle".
center	A boolean value. if TRUE, nodes are plotted centered with respect to all leaves/tips in the branch. Otherwise (default), plot them in the middle of the direct child nodes.
tree_type	A single string, one of "phylogram" or "cladogram", indicating the type of tree. • phylogram: Represents a phylogenetic tree where branch lengths indicate evolutionary distance or time. • cladogram: Represents a tree where branch lengths are not used, or the branches do not reflect evolutionary time. Usually, you don't need to modify this.
no_axes	**[Experimental]** Logical; if TRUE, removes axes elements for the alignment axis using theme_no_axes(). By default, will use the option- "ggalign.align_no_axes".
active	A active() object that defines the context settings when added to a layout.
size	The relative size of the plot, can be specified as a unit(). Note that for circle_layout(), all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.

---

align_plots *Arrange multiple plots into a grid*

Description

Arrange multiple plots into a grid

Usage

```
align_plots(
  ...,
  ncol = NULL,
  nrow = NULL,
  byrow = TRUE,
```

```
  widths = NA,
  heights = NA,
  design = NULL,
  guides = waiver(),
  theme = NULL
)
```

Arguments

...	<dyn-dots> A list of plots, ususally the ggplot object. Use NULL to indicate an empty spacer.
ncol, nrow	The dimensions of the grid to create - if both are NULL it will use the same logic as facet_wrap() to set the dimensions
byrow	If FALSE the plots will be filled in in column-major order.
widths, heights	The relative widths and heights of each column and row in the grid. Will get repeated to match the dimensions of the grid. The special value of NA will behave as 1null unit unless a fixed aspect plot is inserted in which case it will allow the dimension to expand or contract to match the aspect ratio of the content.
design	Specification of the location of areas in the layout. Can either be specified as a text string or by concatenating calls to area() together.
guides	A string with one or more of "t", "l", "b", and "r" indicating which side of guide legends should be collected. Defaults to waiver(), which inherits from the parent layout. If there is no parent layout, or if NULL is provided, no guides will be collected.
theme	A theme() object used to customize various elements of the plot, including guides, title, subtitle, caption, margins, panel.border, and background. By default, the theme will inherit from the parent layout.

Value

An alignpatches object.

See Also

- layout_design()
- layout_title()
- layout_annotation()

Examples

```
# directly copied from patchwork
p1 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) +
    geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) +
    geom_bar(aes(gear)) +
    facet_wrap(~cyl)
```

```
p4 <- ggplot(mtcars) +
    geom_bar(aes(carb))
p5 <- ggplot(mtcars) +
    geom_violin(aes(cyl, mpg, group = cyl))

# Either add the plots as single arguments
align_plots(p1, p2, p3, p4, p5)

# Or use bang-bang-bang to add a list
align_plots(!!!list(p1, p2, p3), p4, p5)

# Match plots to areas by name
design <- "#BB
           AA#"
align_plots(B = p1, A = p2, design = design)

# Compare to not using named plot arguments
align_plots(p1, p2, design = design)
```

`align_reorder` *Reorders layout observations based on specific statistics.*

Description

Reorders layout observations based on specific statistics.

Usage

```
align_reorder(
  stat,
  ...,
  reverse = FALSE,
  strict = TRUE,
  data = NULL,
  active = NULL
)
```

Arguments

`stat`	A statistical function which accepts a data and returns the statistic, which we'll call `order2()` to extract the ordering information.
`...`	<dyn-dots> Additional arguments passed to function provided in `stat` argument.
`reverse`	A boolean value. Should the sort order be in reverse?
`strict`	A boolean value indicates whether the order should be strict. If previous groups has been established, and strict is `FALSE`, this will reorder the observations in each group.

data	A matrix, data frame, or atomic vector used as the input for the stat function. Alternatively, you can specify a function (including purrr-like lambda syntax) that will be applied to the layout matrix, transforming it as necessary for statistic calculations. By default, it will inherit from the layout matrix.
active	A active() object that defines the context settings when added to a layout.

Details

[Experimental]

The align_reorder() function differs from align_order() in that the weights argument in align_order() must return atomic weights for each observation. In contrast, the stat argument in align_reorder() can return more complex structures, such as hclust or dendrogram, among others.

Typically, you can achieve the functionality of align_reorder() using align_order() by manually extracting the ordering information from the statistic.

Discrete Axis Alignment

It is important to note that we consider rows as observations, meaning vec_size(data)/NROW(data) must match the number of observations along the axis used for alignment (x-axis for a vertical stack layout, y-axis for a horizontal stack layout).

See Also

order2()

Examples

```
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_left() +
    align_reorder(hclust2)
```

---

area *Define the plotting areas in* align_plots

Description

This is a small helper used to specify a single area in a rectangular grid that should contain a plot. Objects constructed with area() can be concatenated together with c() in order to specify multiple areas.

Usage

```
area(t, l, b = t, r = l)
```

Arguments

t, b	The top and bottom bounds of the area in the grid
l, r	The left and right bounds of the area int the grid

Details

The grid that the areas are specified in reference to enumerate rows from top to bottom, and coloumns from left to right. This means that `t` and `l` should always be less or equal to `b` and `r` respectively. Instead of specifying area placement with a combination of `area()` calls, it is possible to instead pass in a single string

```
areas <- c(area(1, 1, 2, 1),
           area(2, 3, 3, 3))
```

is equivalent to

```
areas < -"A##
          A#B
          ##B"
```

Value

A `ggalign_area` object.

Examples

```
p1 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) +
    geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) +
    geom_bar(aes(gear)) +
    facet_wrap(~cyl)

layout <- c(
    area(1, 1),
    area(1, 3, 3),
    area(3, 1, 3, 2)
)

# Show the layout to make sure it looks as it should
plot(layout)

# Apply it to a alignpatches
align_plots(p1, p2, p3, design = layout)
```

`circle_layout` *Arrange plots in a circular layout*

Description

[Experimental]

If `limits` is provided, a continuous variable will be required and aligned in the direction specified (`circle_continuous`). Otherwise, a discrete variable will be required and aligned (`circle_discrete`).

Usage

```
circle_layout(
  data = NULL,
  ...,
  radial = NULL,
  direction = "outward",
  limits = waiver(),
  theme = NULL
)

circle_discrete(
  data = NULL,
  ...,
  radial = NULL,
  direction = "outward",
  theme = NULL
)

circle_continuous(
  data = NULL,
  ...,
  radial = NULL,
  direction = "outward",
  limits = NULL,
  theme = NULL
)
```

Arguments

`data`	Default dataset to use for the layout. If not specified, it must be supplied in each plot added to the layout: • If `limits` is not provided, `fortify_matrix()` will be used to get a matrix. • If `limits` is specified, `fortify_data_frame()` will be used to get a data frame.
`...`	Additional arguments passed to `fortify_data_frame()` or `fortify_matrix()`.

radial — A coord_radial() object that defines the global parameters for coord_radial across all plots in the layout. The parameters start, end, direction, and expand will be inherited and applied uniformly to all plots within the layout. The parameters theta and r.axis.inside will always be ignored and will be set to "x" and TRUE, respectively, for all plots.

direction — A single string of "inward" or "outward", indicating the direction in which the plot is added.

- outward: The plot is added from the inner to the outer.
- inward: The plot is added from the outer to the inner.

limits — A continuous_limits() object specifying the left/lower limit and the right/upper limit of the scale. Used to align the continuous axis.

theme — A theme() object used to customize various elements of the layout, including guides, title, subtitle, caption, margins, panel.border, and background. By default, the theme will inherit from the parent layout. It also controls the panel spacing for all plots in the layout.

Value

A CircleLayout object.

Examples

```
set.seed(123)

small_mat <- matrix(rnorm(56), nrow = 7)
rownames(small_mat) <- paste0("row", seq_len(nrow(small_mat)))
colnames(small_mat) <- paste0("column", seq_len(ncol(small_mat)))

# circle_layout
# same for circle_discrete()
circle_layout(small_mat) +
    ggalign() +
    geom_tile(aes(y = .column_index, fill = value)) +
    scale_fill_viridis_c() +
    align_dendro(aes(color = branch), k = 3L) +
    scale_color_brewer(palette = "Dark2")

# same for circle_continuous()
circle_layout(mpg, limits = continuous_limits(c(3, 5))) +
    ggalign(mapping = aes(displ, hwy, colour = class)) +
    geom_point(size = 2) +
    ggalign(mapping = aes(displ, hwy, colour = class)) +
    geom_point(size = 2) &
    scale_color_brewer(palette = "Dark2") &
    theme_bw()

# circle_discrete()
# direction outward
circle_discrete(small_mat) +
    align_dendro(aes(color = branch), k = 3L) +
```

```
    scale_color_brewer(palette = "Dark2") +
    ggalign() +
    geom_tile(aes(y = .column_index, fill = value)) +
    scale_fill_viridis_c()

# direction inward
circle_discrete(small_mat, direction = "inward") +
    ggalign() +
    geom_tile(aes(y = .column_index, fill = value)) +
    scale_fill_viridis_c() +
    align_dendro(aes(color = branch), k = 3L) +
    scale_color_brewer(palette = "Dark2")

# circle_continuous()
circle_continuous(mpg, limits = continuous_limits(c(3, 5))) +
    ggalign(mapping = aes(displ, hwy, colour = class)) +
    geom_point(size = 2) +
    ggalign(mapping = aes(displ, hwy, colour = class)) +
    geom_point(size = 2) &
    scale_color_brewer(palette = "Dark2") &
    theme_bw()
```

`circle_switch` *Determine the active context of circle layout*

Description

[Stable]

Usage

```
circle_switch(radial = waiver(), direction = NULL, what = waiver(), ...)
```

Arguments

`radial`	A `coord_radial()` object that defines the global parameters for `coord_radial` across all plots in the layout. The parameters `start`, `end`, `direction`, and `expand` will be inherited and applied uniformly to all plots within the layout. The parameters `theta` and `r.axis.inside` will always be ignored and will be set to `"x"` and `TRUE`, respectively, for all plots.
`direction`	A single string of `"inward"` or `"outward"`, indicating the direction in which the plot is added. • `outward`: The plot is added from the inner to the outer. • `inward`: The plot is added from the outer to the inner.
`what`	What should get activated for the `circle_layout()`? A single number or string of the plot elements in the layout. If `NULL`, will remove any active context.
`...`	These dots are for future extensions and must be empty.

Value

A `circle_switch` object which can be added to `circle_layout()`.

Examples

```
set.seed(123)
small_mat <- matrix(rnorm(56), nrow = 7)
rownames(small_mat) <- paste0("row", seq_len(nrow(small_mat)))
colnames(small_mat) <- paste0("column", seq_len(ncol(small_mat)))
circle_discrete(small_mat) +
    ggalign() +
    geom_tile(aes(y = .column_index, fill = value)) +
    scale_fill_viridis_c() +
    align_dendro(aes(color = branch), k = 3L) +
    scale_color_brewer(palette = "Dark2")
```

---

continuous_limits *Set continuous limits for the layout*

Description

To align continuous axes, it is important to keep the limits consistent across all plots in the layout. You can set the limits by passing a function directly to the `limits` or `xlim`/`ylim` argument, using `...` only. Alternatively, you can add a `continuous_limits()` object to the layout. For the `quad_layout()` function, you must specify `x`/`y` arguments. For other layouts, you should pass the limits using `...` directly.

Usage

```
continuous_limits(..., x = waiver(), y = waiver())
```

Arguments

`...`	A list of two numeric values, specifying the left/lower limit and the right/upper limit of the scale.
`x`, `y`	Same as `...`, but specifically for `quad_layout()`.

`cross_link` *Add a plot to connect selected observations*

Description

Add a plot to connect selected observations

Usage

```
cross_link(
  link,
  data = waiver(),
  ...,
  on_top = TRUE,
  obs_size = 1,
  inherit_index = NULL,
  inherit_panel = NULL,
  inherit_nobs = NULL,
  size = NULL,
  active = NULL
)
```

Arguments

`link`	A `link_draw()` object that defines how to draw the links, such as `link_line()`.
`data`	The dataset to use for the layout. By default, `fortify_matrix()` will convert the data to a matrix. This argument allows you to change the layout data. If not specified, the original data will be used.
`...`	<dyn-dots> Additional arguments passed to `fortify_matrix()`.
`on_top`	A boolean value indicating whether to draw the link on top of the plot panel (`TRUE`) or below (`FALSE`).
`obs_size`	A single numeric value that indicates the size of a single observation, ranging from `(0, 1]`.
`inherit_index`	A boolean value indicating whether to inherit the ordering index. If `TRUE`, will match the layout ordering index with the data names.
`inherit_panel`	A boolean value indicating whether to inherit the panel group. If `TRUE`, will match the layout panel with the data names.
`inherit_nobs`	A boolean value indicating whether to inherit the number of observations (nobs). If `TRUE`, the `data` input must be compatible with the layout data.
`size`	The relative size of the plot, can be specified as a `unit()`. Note that for `circle_layout()`, all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.
`active`	A `active()` object that defines the context settings when added to a layout.

ggplot2 Specification

The `cross_link` function initializes a `ggplot` object but does not initialize any data. Using `scheme_data()` to change the internal data if needed.

`cross_mark` *Add a plot to annotate observations*

Description

Add a plot to annotate observations

Usage

```
cross_mark(
  mark,
  data = waiver(),
  ...,
  obs_size = 1,
  inherit_index = NULL,
  inherit_panel = NULL,
  inherit_nobs = NULL,
  size = NULL,
  active = NULL
)
```

Arguments

`mark`	A `mark_draw()` object to define how to draw the links. Like `mark_line()`, `mark_tetragon()`. Note the names of the pair links will be used to define the panel names so must be unique.
`data`	The dataset to use for the layout. By default, `fortify_matrix()` will convert the data to a matrix. This argument allows you to change the layout data. If not specified, the original data will be used.
`...`	<dyn-dots> Additional arguments passed to `fortify_matrix()`.
`obs_size`	A single numeric value that indicates the size of a single observation, ranging from `(0, 1]`.
`inherit_index`	A boolean value indicating whether to inherit the ordering index. If `TRUE`, will match the layout ordering index with the data names.
`inherit_panel`	A boolean value indicating whether to inherit the panel group. If `TRUE`, will match the layout panel with the data names.
`inherit_nobs`	A boolean value indicating whether to inherit the number of observations (nobs). If `TRUE`, the `data` input must be compatible with the layout data.
`size`	The relative size of the plot, can be specified as a `unit()`. Note that for `circle_layout()`, all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.
`active`	A `active()` object that defines the context settings when added to a layout.

ggplot2 Specification

The `cross_mark` function initializes a `ggplot` object. The underlying data contains following columns:

- `.panel`: the panel for the aligned axis. It means `x-axis` for vertical stack layout (including top and bottom annotation), `y-axis` for horizontal stack layout (including left and right annotation).
- `.names` (`vec_names()`) and `.index` (`vec_size()`/`NROW()`): a character names (only applicable when names exists) and an integer of index of the original data.
- `.hand`: A factor with levels `c("left", "right")` for horizontal stack layouts, or `c("top", "bottom")` for vertical stack layouts, indicating the position of the linked observations.

You can use `scheme_data()` to modify the internal data if needed.

`cross_none` *Reset layout ordering and panel group*

Description

Reset layout ordering and panel group

Usage

```
cross_none(
  data = waiver(),
  ...,
  inherit_index = NULL,
  inherit_panel = NULL,
  inherit_nobs = NULL
)
```

Arguments

`data`	The dataset to use for the layout. By default, `fortify_matrix()` will convert the data to a matrix. This argument allows you to change the layout data. If not specified, the original data will be used.
`...`	<dyn-dots> Additional arguments passed to `fortify_matrix()`.
`inherit_index`	A boolean value indicating whether to inherit the ordering index. If `TRUE`, will match the layout ordering index with the data names.
`inherit_panel`	A boolean value indicating whether to inherit the panel group. If `TRUE`, will match the layout panel with the data names.
`inherit_nobs`	A boolean value indicating whether to inherit the number of observations (nobs). If `TRUE`, the `data` input must be compatible with the layout data.

draw_key_draw *Key glyphs for legends*

Description

Each geom has an associated function that draws the key when the geom needs to be displayed in a legend. These functions are called `draw_key_*()`, where `*` stands for the name of the respective key glyph. The key glyphs can be customized for individual geoms by providing a geom with the `key_glyph` argument. The `draw_key_draw` function provides this interface for custom key glyphs used with `geom_draw()`.

Usage

```
draw_key_draw(data, params, size)
```

Arguments

`data`	A single row data frame containing the scaled aesthetics to display in this key
`params`	A list of additional parameters supplied to the geom.
`size`	Width and height of key in mm.

Value

A grid grob.

Examples

```
p <- ggplot(economics, aes(date, psavert, color = "savings rate"))
# key glyphs can be specified by their name
p + geom_line(key_glyph = "timeseries")

# key glyphs can be specified via their drawing function
p + geom_line(key_glyph = draw_key_rect)
```

draw_key_draw2 *Key glyphs for legends*

Description

Each geom has an associated function that draws the key when the geom needs to be displayed in a legend. These functions are called `draw_key_*()`, where `*` stands for the name of the respective key glyph. The key glyphs can be customized for individual geoms by providing a geom with the `key_glyph` argument. The `draw_key_draw2` function provides this interface for custom key glyphs used with `geom_draw2()`.

Usage

```
draw_key_draw2(data, params, size)
```

Arguments

data	A single row data frame containing the scaled aesthetics to display in this key
params	A list of additional parameters supplied to the geom.
size	Width and height of key in mm.

Value

A grid grob.

Examples

```
p <- ggplot(economics, aes(date, psavert, color = "savings rate"))
# key glyphs can be specified by their name
p + geom_line(key_glyph = "timeseries")

# key glyphs can be specified via their drawing function
p + geom_line(key_glyph = draw_key_rect)
```

---

element_curve *Theme curve elements*

Description

Draw curve.

Usage

```
element_curve(
  colour = NULL,
  linewidth = NULL,
  linetype = NULL,
  lineend = NULL,
  color = NULL,
  curvature = NULL,
  angle = NULL,
  ncp = NULL,
  shape = NULL,
  arrow = NULL,
  arrow.fill = NULL,
  inherit.blank = FALSE
)
```

Arguments

colour, color	Line/border colour. Color is an alias for colour.
linewidth	Line/border size in mm.
linetype	Line type. An integer (0:8), a name (blank, solid, dashed, dotted, dotdash, longdash, twodash), or a string with an even number (up to eight) of hexadecimal digits which give the lengths in consecutive positions in the string.
lineend	Line end Line end style (round, butt, square)
curvature	A numeric value giving the amount of curvature. Negative values produce left-hand curves, positive values produce right-hand curves, and zero produces a straight line.
angle	A numeric value between 0 and 180, giving an amount to skew the control points of the curve. Values less than 90 skew the curve towards the start point and values greater than 90 skew the curve towards the end point.
ncp	The number of control points used to draw the curve. More control points creates a smoother curve.
shape	A numeric vector of values between -1 and 1, which control the shape of the curve relative to its control points. See `grid.xspline` for more details.
arrow	A list describing arrow heads to place at either end of the curve, as produced by the `arrow` function.
arrow.fill	Fill colour for arrows.
inherit.blank	Should this element inherit the existence of an `element_blank` among its parents? If `TRUE` the existence of a blank element among its parents will cause this element to be blank as well. If `FALSE` any blank parent element will be ignored when calculating final element state.

Value

A `element_curve` object

`element_polygon` *Theme Polygon elements*

Description

Draw polygon.

Usage

```
element_polygon(
  fill = NULL,
  colour = NULL,
  linewidth = NULL,
  linetype = NULL,
```

```
  alpha = NULL,
  lineend = NULL,
  linejoin = NULL,
  linemitre = NULL,
  color = NULL,
  inherit.blank = FALSE
)
```

Arguments

`fill`	Fill colour.
`colour`, `color`	Line/border colour. Color is an alias for colour.
`linewidth`	Line/border size in mm.
`linetype`	Line type. An integer (0:8), a name (blank, solid, dashed, dotted, dotdash, longdash, twodash), or a string with an even number (up to eight) of hexadecimal digits which give the lengths in consecutive positions in the string.
`alpha`	A transparency value between 0 (transparent) and 1 (opaque), parallel to `fill`.
`lineend`	Line end Line end style (round, butt, square)
`linejoin`	Line join style (round, mitre, bevel).
`linemitre`	Line mitre limit (number greater than 1).
`inherit.blank`	Should this element inherit the existence of an `element_blank` among its parents? If `TRUE` the existence of a blank element among its parents will cause this element to be blank as well. If `FALSE` any blank parent element will be ignored when calculating final element state.

Value

A `element_polygon` object

See Also

`element_rect`

---

element_vec *Apply a function to the fields of an element object*

Description

For an `element` object, some fields are vectorized, while others are not. This function allows you to apply a function to the vectorized fields.

The following helper functions are available:

- `element_rep`: Applies `rep()`.
- `element_rep_len`: Applies `rep_len()`.

- element_vec_recycle: Applies vec_recycle().
- element_vec_rep: Applies vec_rep().
- element_vec_rep_each: Applies vec_rep_each().
- element_vec_slice: Applies vec_slice().

Usage

```
element_vec(.el, .fn, ...)

element_rep(.el, ...)

element_rep_len(.el, length.out, ...)

element_vec_recycle(.el, size, ...)

element_vec_rep(.el, times, ...)

element_vec_rep_each(.el, times, ...)

element_vec_slice(.el, i, ...)
```

Arguments

.el	An element object.
.fn	The function to be applied to the vectorized fields of the element object.
...	Additional arguments passed on to fn.
length.out	Non-negative integer. The desired length of the output vector. Other inputs will be coerced to a double vector and the first element taken. Ignored if NA or invalid.
size	Desired output size.
times	For vec_rep(), a single integer for the number of times to repeat the entire vector. For vec_rep_each(), an integer vector of the number of times to repeat each element of x. times will be recycled to the size of x.
i	An integer, character or logical vector specifying the locations or names of the observations to get/set. Specify TRUE to index all elements (as in x[]), or NULL, FALSE or integer() to index none (as in x[NULL]).

---

fortify_data_frame *Build a data frame*

Description

[Stable]

This function converts various objects to a data frame.

Usage

```
fortify_data_frame(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

data	An object to be converted to a data frame.
...	Arguments passed to methods.
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

Value

A data frame.

See Also

- fortify_data_frame.default()
- fortify_data_frame.character()/fortify_data_frame.numeric()
- fortify_data_frame.matrix()
- fortify_data_frame.dendrogram()
- fortify_data_frame.phylo()

---

fortify_data_frame.character

Build a data frame

Description

[Stable]

This function converts various objects to a data frame.

Usage

```
## S3 method for class 'character'
fortify_data_frame(data, ..., data_arg = caller_arg(data), call = NULL)

## S3 method for class 'numeric'
fortify_data_frame(data, ..., data_arg = caller_arg(data), call = NULL)

## S3 method for class 'logical'
fortify_data_frame(data, ..., data_arg = caller_arg(data), call = NULL)

## S3 method for class 'complex'
fortify_data_frame(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

data	An object to be converted to a data frame.
...	These dots are for future extensions and must be empty.
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

Value

A data frame with following columns:

- .names: the names for the vector (only applicable if names exist).
- value: the actual value of the vector.

See Also

Other fortify_data_frame methods: fortify_data_frame.default(), fortify_data_frame.dendrogram(), fortify_data_frame.matrix(), fortify_data_frame.phylo()

---

fortify_data_frame.default

Build a data frame

Description

[Stable]

This function converts various objects to a data frame.

Usage

```
## Default S3 method:
fortify_data_frame(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

data	An object to be converted to a data frame.
...	Additional arguments passed to fortify().
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

Details

By default, it calls `fortify()` to build the data frame.

Value

A data frame.

See Also

Other fortify_data_frame methods: `fortify_data_frame.character()`, `fortify_data_frame.dendrogram()`, `fortify_data_frame.matrix()`, `fortify_data_frame.phylo()`

`fortify_data_frame.dendrogram`
Build a data frame

Description

[Stable]

This function converts various objects to a data frame.

Usage

```
## S3 method for class 'dendrogram'
fortify_data_frame(
  data,
  ...,
  priority = "right",
  center = FALSE,
  type = "rectangle",
  leaf_pos = NULL,
  leaf_braches = NULL,
  reorder_branches = TRUE,
  branch_gap = NULL,
  root = NULL,
  double = TRUE,
  data_arg = caller_arg(data),
  call = NULL
)

## S3 method for class 'hclust'
fortify_data_frame(data, ...)
```

Arguments

data	A hclust or a dendrogram object.
...	Additional arguments passed to dendrogram method.
priority	A string of "left" or "right". if we draw from right to left, the left will override the right, so we take the "left" as the priority. If we draw from left to right, the right will override the left, so we take the "right" as priority. This is used by align_dendro() to provide support of facet operation in ggplot2.
center	A boolean value. if TRUE, nodes are plotted centered with respect to all leaves/tips in the branch. Otherwise (default), plot them in the middle of the direct child nodes.
type	A string indicates the plot type, "rectangle" or "triangle".
leaf_pos	The x-coordinates of the leaf node. Must be the same length of the number of observations in tree.
leaf_braches	Branches of the leaf node. Must be the same length of the number of observations in tree. Usually come from cutree.
reorder_branches	A single boolean value, indicates whether reorder the provided leaf_braches based on the actual index.
branch_gap	A single numeric value indicates the gap between different branches.
root	A length one string or numeric indicates the root branch.
double	A single logical value indicating whether horizontal lines should be doubled when segments span multiple branches. If TRUE, the horizontal lines will be repeated for each branch that the segment spans. If FALSE, only one horizontal line will be drawn. This is used by align_dendro() to provide support of facet operation in ggplot2.
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

Value

A data frame with the node coordinates:

- .panel: Similar with panel column, but always give the correct branch for usage of the ggplot facet.
- .index: the original index in the tree for the the node
- label: node label text
- x and y: x-axis and y-axis coordinates for the node
- branch: which branch the node is. You can use this column to color different groups.
- panel: which panel the node is, if we split the plot into panel using facet_grid, this column will show which panel the node is from. Note: some nodes may fall outside panel (between two panels), so there are possible NA values in this column.
- leaf: A logical value indicates whether the node is a leaf.

ggalign attributes

edge: A `data frame` for edge coordinates:

- `.panel`: Similar with `panel` column, but always give the correct branch for usage of the ggplot facet.
- `x` and `y`: x-axis and y-axis coordinates for the start node of the edge.
- `xend` and `yend`: the x-axis and y-axis coordinates of the terminal node for edge.
- `branch`: which branch the edge is. You can use this column to color different groups.
- `panel1` and `panel2`: The panel1 and panel2 columns have the same functionality as `panel`, but they are specifically for the edge data and correspond to both nodes of each edge.

See Also

Other fortify_data_frame methods: `fortify_data_frame.character()`, `fortify_data_frame.default()`, `fortify_data_frame.matrix()`, `fortify_data_frame.phylo()`

Examples

```
fortify_data_frame(hclust(dist(USArrests), "ave"))
```

---

fortify_data_frame.matrix

Build a data frame

Description

[Stable]

This function converts various objects to a data frame.

Usage

```
## S3 method for class 'matrix'
fortify_data_frame(data, ..., data_arg = caller_arg(data), call = NULL)

## S3 method for class 'DelayedMatrix'
fortify_data_frame(data, ...)

## S3 method for class 'Matrix'
fortify_data_frame(data, ...)
```

Arguments

`data`	A matrix-like object.
`...`	These dots are for future extensions and must be empty.
`data_arg`	The argument name for `data`. Developers can use it to improve messages. Not used by the user.
`call`	The execution environment where `data` and other arguments for the method are collected, e.g., `caller_env()`. Developers can use it to improve messages. Not used by the user.

Value

Matrix will be transformed into a long-form data frame, where each row represents a unique combination of matrix indices and their corresponding values. The resulting data frame will contain the following columns:

- `.row_names` and `.row_index`: the row names (only applicable when names exist) and an integer representing the row index of the original matrix.
- `.column_names` and `.column_index`: the column names (only applicable when names exist) and column index of the original matrix.
- `value`: the actual value.

See Also

Other fortify_data_frame methods: `fortify_data_frame.character()`, `fortify_data_frame.default()`, `fortify_data_frame.dendrogram()`, `fortify_data_frame.phylo()`

`fortify_data_frame.phylo` *Build a data frame*

Description

[Stable]

This function converts various objects to a data frame.

Usage

```
## S3 method for class 'phylo'
fortify_data_frame(
  data,
  ...,
  type = "rectangle",
  center = FALSE,
  tree_type = NULL,
  tip_pos = NULL,
  data_arg = caller_arg(data),
  call = NULL
)
```

Arguments

data	A hclust or a dendrogram object.
...	These dots are for future extensions and must be empty.
type	A string indicates the plot type, "rectangle" or "triangle".
center	A boolean value. if TRUE, nodes are plotted centered with respect to all leaves/tips in the branch. Otherwise (default), plot them in the middle of the direct child nodes.
tree_type	A single string, one of "phylogram" or "cladogram", indicating the type of tree. • phylogram: Represents a phylogenetic tree where branch lengths indicate evolutionary distance or time. • cladogram: Represents a tree where branch lengths are not used, or the branches do not reflect evolutionary time. Usually, you don't need to modify this.
tip_pos	The x-coordinates of the tip. Must be the same length of the number of tips in tree.
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

Value

A data frame with the node coordinates:

- .index: the original index in the tree for the the tip/node.
- label: the tip/node label text.
- x and y: x-axis and y-axis coordinates for the tip/node.
- tip: A logical value indicates whether current node is a tip.

ggalign attributes

edge: A data frame for edge coordinates:

- x and y: x-axis and y-axis coordinates for the start node of the edge.
- xend and yend: the x-axis and y-axis coordinates of the terminal node for edge.

See Also

Other fortify_data_frame methods: fortify_data_frame.character(), fortify_data_frame.default(), fortify_data_frame.dendrogram(), fortify_data_frame.matrix()

`fortify_matrix` *Build a Matrix*

Description

[Stable]

This function converts various objects into a matrix format. By default, it calls `as.matrix()` to build a matrix.

Usage

```
fortify_matrix(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

`data`	An object to be converted into a matrix.
`...`	Additional arguments passed to methods.
`data_arg`	The argument name for `data`. Developers can use it to improve messages. Not used by the user.
`call`	The execution environment where `data` and other arguments for the method are collected, e.g., `caller_env()`. Developers can use it to improve messages. Not used by the user.

Value

A matrix.

See Also

- `fortify_matrix.default()`
- `fortify_matrix.MAF()`
- `fortify_matrix.GISTIC()`
- `fortify_matrix.list_upset()`
- `fortify_matrix.matrix_upset()`

fortify_matrix.default *Build a Matrix*

Description

By default, it calls `as.matrix()` to build a matrix.

Usage

```
## Default S3 method:
fortify_matrix(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

`data`	An object to be converted into a matrix.
`...`	These dots are for future extensions and must be empty.
`data_arg`	The argument name for `data`. Developers can use it to improve messages. Not used by the user.
`call`	The execution environment where `data` and other arguments for the method are collected, e.g., `caller_env()`. Developers can use it to improve messages. Not used by the user.

Value

A matrix.

See Also

Other fortify_matrix methods: `fortify_matrix.GISTIC()`, `fortify_matrix.MAF()`, `fortify_matrix.list_upset()`, `fortify_matrix.matrix()`, `fortify_matrix.matrix_upset()`, `fortify_matrix.phylo()`

fortify_matrix.GISTIC *Build a matrix from a maftools object*

Description

Build a matrix from a maftools object

Usage

```
## S3 method for class 'GISTIC'
fortify_matrix(
  data,
  ...,
  n_top = NULL,
  bands = NULL,
  ignored_bands = NULL,
  sample_anno = NULL,
  remove_empty_samples = TRUE,
  data_arg = caller_arg(data),
  call = NULL
)
```

Arguments

data	A GISTIC object.
...	These dots are for future extensions and must be empty.
n_top	A single number indicates how many top bands to be drawn.
bands	An atomic character defines the bands to draw.
ignored_bands	An atomic character defines the bands to be ignored.
sample_anno	A data frame of sample clinical features to be added.
remove_empty_samples	A single boolean value indicating whether to drop samples without any genomic alterations.
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

ggalign attributes

- sample_anno: sample clinical informations provided in sample_anno.
- sample_summary: sample copy number summary informations. See data@cnv.summary for details.
- cytoband_summary: cytoband summary informations. See data@cytoband.summary for details.
- gene_summary: gene summary informations. See data@gene.summary for details.
- summary: A data frame of summary information. See data@summary for details.

See Also

Other fortify_matrix methods: fortify_matrix.MAF(), fortify_matrix.default(), fortify_matrix.list_upset(), fortify_matrix.matrix(), fortify_matrix.matrix_upset(), fortify_matrix.phylo()

`fortify_matrix.list_upset` *Build a Matrix for UpSet plot*

Description

[Experimental]

This function converts a list into a matrix format suitable for creating an UpSet plot. It always returns a matrix for a `horizontal` UpSet plot.

Usage

```
## S3 method for class 'list_upset'
fortify_matrix(
  data,
  mode = "distinct",
  ...,
  data_arg = caller_arg(data),
  call = NULL
)
```

Arguments

`data`	A list of sets.
`mode`	A string of `"distinct"`, `"intersect"`, or `"union"` indicates the mode to define the set intersections. Check https://jokergoo.github.io/ComplexHeatmap-reference/book/upset-plot.html#upset-mode for details.
`...`	These dots are for future extensions and must be empty.
`data_arg`	The argument name for `data`. Developers can use it to improve messages. Not used by the user.
`call`	The execution environment where `data` and other arguments for the method are collected, e.g., `caller_env()`. Developers can use it to improve messages. Not used by the user.

ggalign attributes

- `intersection_sizes`: An integer vector indicating the size of each intersection.
- `set_sizes`: An integer vector indicating the size of each set.

See Also

`tune.list()`

Other fortify_matrix methods: `fortify_matrix.GISTIC()`, `fortify_matrix.MAF()`, `fortify_matrix.default()`, `fortify_matrix.matrix()`, `fortify_matrix.matrix_upset()`, `fortify_matrix.phylo()`

`fortify_matrix.MAF` *Build a matrix from a maftools object*

Description

Convert `MAF` object to a matrix:

- `fortify_matrix.MAF`: Extract genomic alterations for genes.
- `fortify_matrix.MAF_pathways`: Extract genomic alterations for pathways. `tune.MAF()` helps convert `MAF` object to a `MAF_pathways` object.

Usage

```
## S3 method for class 'MAF'
fortify_matrix(
  data,
  ...,
  genes = NULL,
  n_top = NULL,
  remove_empty_genes = TRUE,
  remove_empty_samples = TRUE,
  collapse_vars = TRUE,
  use_syn = TRUE,
  missing_genes = "error",
  data_arg = caller_arg(data),
  call = NULL
)

## S3 method for class 'MAF_pathways'
fortify_matrix(
  data,
  ...,
  pathdb = "smgbp",
  remove_empty_pathways = TRUE,
  remove_empty_samples = TRUE,
  data_arg = caller_arg(data),
  call = NULL
)
```

Arguments

`data`	A `MAF` object.
`...`	These dots are for future extensions and must be empty.
`genes`	An atomic character defines the genes to draw.
`n_top`	A single number indicates how many top genes to be drawn.

remove_empty_genes
: A single boolean value indicats whether to drop genes without any genomic alterations.

remove_empty_samples
: A single boolean value indicats whether to drop samples without any genomic alterations.

collapse_vars
: A single boolean value indicating whether to collapse multiple alterations in the same sample and gene into a single value "Multi_Hit". Alternatively, you can provide a single string indicates the collapsed values.

use_syn
: A single boolean value indicates whether to include synonymous variants when Classifies SNPs into transitions and transversions.

missing_genes
: A string, either "error" or "remove", specifying the action for handling missing genes.

data_arg
: The argument name for data. Developers can use it to improve messages. Not used by the user.

call
: The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

pathdb
: A string of "smgbp" or "sigpw", or a named list of genes to define the pathways.

remove_empty_pathways
: A single boolean value indicats whether to drop pathways without any genomic alterations.

ggalign attributes

For fortify_matrix.MAF:

- gene_summary: gene summary informations. See maftools::getGeneSummary() for details.
- sample_summary: sample summary informations. See maftools::getSampleSummary() for details.
- sample_anno: sample clinical informations. See maftools::getClinicalData() for details.
- n_genes: Total of genes.
- n_samples: Total of samples.
- titv: A list of data.frames with Transitions and Transversions summary. See maftools::titv() for details.

For fortify_matrix.MAF_pathways:

- gene_list: the pathway contents.
- pathway_summary: pathway summary informations. See maftools::pathways() for details.
- sample_summary: sample summary informations. See maftools::getSampleSummary() for details.
- sample_anno: sample clinical informations. See maftools::getClinicalData() for details.

See Also

Other fortify_matrix methods: fortify_matrix.GISTIC(), fortify_matrix.default(), fortify_matrix.list_upset(
fortify_matrix.matrix(), fortify_matrix.matrix_upset(), fortify_matrix.phylo()

`fortify_matrix.matrix` *Build a matrix from a matrix*

Description

Build a matrix from a matrix

Usage

```
## S3 method for class 'matrix'
fortify_matrix(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

data	A matrix object.
...	These dots are for future extensions and must be empty.
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

shape

- upset: fortify_matrix.matrix_upset

See Also

Other fortify_matrix methods: fortify_matrix.GISTIC(), fortify_matrix.MAF(), fortify_matrix.default(), fortify_matrix.list_upset(), fortify_matrix.matrix_upset(), fortify_matrix.phylo()

`fortify_matrix.matrix_upset` *Build a Matrix for UpSet plot*

Description

Converts a matrix suitable for creating an UpSet plot. `tune.matrix()` helps convert `matrix` object to a `matrix_upset` object.

Usage

```
## S3 method for class 'matrix_upset'
fortify_matrix(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

`data`	A matrix where each row represents an element, and each column defines a set. The values in the matrix indicate whether the element is part of the set. Any non-missing value signifies that the element exists in the set.
`...`	Arguments passed on to `fortify_matrix.list_upset` `mode` A string of `"distinct"`, `"intersect"`, or `"union"` indicates the mode to define the set intersections. Check https://jokergoo.github.io/ComplexHeatmap-reference/book/upset-plot.html#upset-mode for details.
`data_arg`	The argument name for `data`. Developers can use it to improve messages. Not used by the user.
`call`	The execution environment where `data` and other arguments for the method are collected, e.g., `caller_env()`. Developers can use it to improve messages. Not used by the user.

ggalign attributes

- `intersection_sizes`: An integer vector indicating the size of each intersection.
- `set_sizes`: An integer vector indicating the size of each set.

See Also

`tune.matrix()`

Other fortify_matrix methods: `fortify_matrix.GISTIC()`, `fortify_matrix.MAF()`, `fortify_matrix.default()`, `fortify_matrix.list_upset()`, `fortify_matrix.matrix()`, `fortify_matrix.phylo()`

fortify_matrix.phylo *Build a matrix from* phylo *object*

Description

This method allows a phylo object to be directly input into stack_discrete() or circle_discrete(). This makes it possible to add align_phylo() to the stack independently, as align_phylo() requires the layout to have labels.

Usage

```
## S3 method for class 'phylo'
fortify_matrix(data, ..., data_arg = caller_arg(data), call = NULL)
```

Arguments

data	A phylo object.
...	These dots are for future extensions and must be empty.
data_arg	The argument name for data. Developers can use it to improve messages. Not used by the user.
call	The execution environment where data and other arguments for the method are collected, e.g., caller_env(). Developers can use it to improve messages. Not used by the user.

Value

A one-column matrix where the tip labels are the values, and the row names will also be the tip labels.

See Also

Other fortify_matrix methods: fortify_matrix.GISTIC(), fortify_matrix.MAF(), fortify_matrix.default(), fortify_matrix.list_upset(), fortify_matrix.matrix(), fortify_matrix.matrix_upset()

free_align *Free from alignment*

Description

align_plots will try to align plot panels, and every elements of the plot, following functions romove these restrictions:

- free_align: if we want to compose plots without alignment of some panel axes (panel won't be aligned). we can wrap the plot with free_align.

- `free_border`: If we want to compose plots without alignment of the panel borders (but still align the panels themselves), we can wrap the plot with `free_border`.
- `free_lab`: If we want to compose plots without alignment of the axis title, we can wrap the plot with `free_lab`.
- `free_space`: Removing the ggplot element sizes when aligning.
- `free_vp`: Customize the viewport when aligning.
- `free_guide`: If we want to override the behaviour of the overall guides behaviour, we can wrap the plot with `free_guide`.

Usage

```
free_align(plot, axes = "tlbr")

free_border(plot, borders = "tlbr")

free_guide(plot, guides = "tlbr")

free_lab(plot, labs = "tlbr")

free_space(plot, spaces = "tlbr")

free_vp(plot, x = 0.5, y = 0.5, width = NA, height = NA, ...)
```

Arguments

`plot`	A ggplot or alignpatches object.
`axes`	Which axes shouldn't be aligned? A string containing one or more of `"t"`, `"l"`, `"b"`, and `"r"`.
`borders`	Which border shouldn't be aligned? A string containing one or more of `"t"`, `"l"`, `"b"`, and `"r"`.
`guides`	A string containing one or more of `"t"`, `"l"`, `"b"`, and `"r"` indicates which side of guide legends should be collected for the plot. If `NULL`, no guide legends will be collected.
`labs`	Which axis labs to be free? A string containing one or more of `"t"`, `"l"`, `"b"`, and `"r"`.
`spaces`	Which border spaces should be removed? A string containing one or more of `"t"`, `"l"`, `"b"`, and `"r"`.
`x`	A numeric vector or unit object specifying x-location.
`y`	A numeric vector or unit object specifying y-location.
`width`	A numeric vector or unit object specifying width.
`height`	A numeric vector or unit object specifying height.
`...`	Arguments passed on to `grid::viewport`
	`default.units` A string indicating the default units to use if `x`, `y`, `width`, or `height` are only given as numeric vectors.

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

`gp` An object of class `"gpar"`, typically the output from a call to the function `gpar`. This is basically a list of graphical parameter settings.

`clip` One of `"on"`, `"inherit"`, or `"off"`, indicating whether to clip to the extent of this viewport, inherit the clipping region from the parent viewport, or turn clipping off altogether. For back-compatibility, a logical value of `TRUE` corresponds to `"on"` and `FALSE` corresponds to `"inherit"`.
May also be a grob (or a gTree) that describes a clipping path or the result of a call to `as.path`.

`mask` One of `"none"` (or `FALSE`) or `"inherit"` (or `TRUE`) or a grob (or a gTree) or the result of call to `as.mask`. This specifies that the viewport should have no mask, or it should inherit the mask of its parent, or it should have its own mask, as described by the grob.

`xscale` A numeric vector of length two indicating the minimum and maximum on the x-scale. The limits may not be identical.

`yscale` A numeric vector of length two indicating the minimum and maximum on the y-scale. The limits may not be identical.

`angle` A numeric value indicating the angle of rotation of the viewport. Positive values indicate the amount of rotation, in degrees, anticlockwise from the positive x-axis.

`layout` A Grid layout object which splits the viewport into subregions.

`layout.pos.row` A numeric vector giving the rows occupied by this viewport in its parent's layout.

`layout.pos.col` A numeric vector giving the columns occupied by this viewport in its parent's layout.

`name` A character value to uniquely identify the viewport once it has been pushed onto the viewport tree.

Value

- `free_align`: A modified version of `plot` with a `free_align` class.
- `free_border`: A modified version of `plot` with a `free_border` class.
- `free_guide`: A modified version of `plot` with a `free_guide` class.
- `free_lab`: A modified version of `plot` with a `free_lab` class.
- `free_space`: A modified version of `plot` with a `free_space` class.
- `free_vp`: A modified version of `plot` with a `free_vp` class.

Examples

```
# directly copied from `patchwork`
# Sometimes you have a plot that defies good composition alginment, e.g. due
# to long axis labels
p1 <- ggplot(mtcars) +
    geom_bar(aes(y = factor(gear), fill = factor(gear))) +
    scale_y_discrete(
        "",
        labels = c(
            "3 gears are often enough",
            "But, you know, 4 is a nice number",
            "I would def go with 5 gears in a modern car"
        )
    )

# When combined with other plots it ends up looking bad
p2 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp))

align_plots(p1, p2, ncol = 1L)

# We can fix this be using `free_align`
align_plots(free_align(p1), p2, ncol = 1L)

# If we still want the panels to be aligned to the right, we can choose to
# free only the left side
align_plots(free_align(p1, axes = "l"), p2, ncol = 1L)

# We could use `free_lab` to fix the layout in a different way
align_plots(p1, free_lab(p2), ncol = 1L)

# `free_border` is similar with `free_lab`, they have a distinction in terms
# of placement on either the top or bottom side of the panel. Specifically,
# the top side contains the `title` and `subtitle`, while the bottom side
# contains the `caption`. free_lab() does not attach these elements in the
# panel area.
p3 <- ggplot(mtcars) +
    geom_point(aes(hp, wt, colour = mpg)) +
    ggtitle("Plot 3")
p_axis_top <- ggplot(mtcars) +
    geom_point(aes(mpg, disp)) +
    ggtitle("Plot axis in top") +
    scale_x_continuous(position = "top")
align_plots(p_axis_top, free_lab(p3))
align_plots(p_axis_top, free_border(p3))

# Another issue is that long labels can occupy much spaces
align_plots(NULL, p1, p2, p2)

# This can be fixed with `free_space`
align_plots(NULL, free_space(p1, "l"), p2, p2)
```

geom_draw *Layer with Grid or Function*

Description

Draw a ggplot2 layer using a grob or a function.

Usage

```
geom_draw(
  draw,
  mapping = NULL,
  data = NULL,
  type = "group",
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

Arguments

draw	Either a grob object or a function (can be purrr-style) that accepts at least three arguments (data, panel_params and coord) and returns a grob. When draw is a function, it is used as the draw_group/draw_panel function in a Geom ggproto object. You should always call coord$transform(data, panel_params) inside the function draw to obtain transformed data in the plot scales.
mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
type	A single string of "group" or "panel", "group" draws geoms with draw_group, which displays multiple observations as one geometric object, and "panel" draws geoms with draw_panel, displaying individual graphical objects for each observation (row). Default: "group".

`stat` The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used the override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A `Stat` ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as `"count"`.
- For more information and other ways to specify the stat, see the layer stat documentation.

`position` A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `...`. Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

Details

If you want to combine the functionality of multiple geoms, it can typically be achieved by preparing the data for each geom inside the draw_*() call and sending it off to the different geoms, collecting the output in a grid::gList (a list of grobs) for draw_group() or a grid::gTree (a grob containing multiple child grobs) for draw_panel().

See Also

https://ggplot2.tidyverse.org/reference/ggplot2-ggproto.html

Examples

```
text <- grid::textGrob(
    "ggdraw",
    x = c(0, 0, 0.5, 1, 1),
    y = c(0, 1, 0.5, 0, 1),
    hjust = c(0, 0, 0.5, 1, 1),
    vjust = c(0, 1, 0.5, 0, 1)
)
ggplot(data.frame(x = 1, y = 2)) +
    geom_draw(text)
```

---

geom_draw2 *Layer with customized draw function*

Description

Layer with customized draw function

Usage

```
geom_draw2(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Additional arguments passed on to draw aesthetic.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. borders().

Details

geom_draw2 depends on the new aesthetics draw, which should always be provided with scale_draw_manual(), in which, we can provide a list of functions that define how each value should be drawn. Any ggplot2 aesthetics can be used as the arguments.

Aesthetics

geom_draw2() understands the following aesthetics (required aesthetics are in bold):

- draw
- alpha
- colour
- fill
- group
- linetype
- linewidth
- shape
- size
- stroke
- x
- y

Learn more about setting these aesthetics in vignette("ggplot2-specs", package = "ggplot2").

Examples

```
library(grid)
ggplot(data.frame(value = letters[seq_len(5)], y = seq_len(5))) +
    geom_draw2(aes(x = 1, y = y, draw = value, fill = value)) +
    scale_draw_manual(values = list(
        a = function(x, y, width, height, fill) {
            rectGrob(x, y,
                width = width, height = height,
                gp = gpar(fill = fill),
                default.units = "native"
            )
        },
        b = function(x, y, width, height, fill) {
            rectGrob(x, y,
                width = width, height = height,
                gp = gpar(fill = fill),
                default.units = "native"
            )
        },
        c = function(x, y, width, height, fill) {
            rectGrob(x, y,
                width = width, height = height,
```

```
                gp = gpar(fill = fill),
                default.units = "native"
            )
        },
        d = function(x, y, width, height, shape) {
            gList(
                pointsGrob(x, y, pch = shape),
                # To ensure the rectangle color is shown in the legends, you
                # must explicitly provide a color argument and include it in
                # the `gpar()` of the graphical object
                rectGrob(x, y, width, height,
                    gp = gpar(col = "black", fill = NA)
                )
            )
        },
        e = function(xmin, xmax, ymin, ymax) {
            segmentsGrob(
                xmin, ymin,
                xmax, ymax,
                gp = gpar(lwd = 2)
            )
        }
    )) +
    scale_fill_brewer(palette = "Dark2") +
    theme_void()
```

---

geom_pie *Pie charts*

Description

Pie charts

Usage

```
geom_pie(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  clockwise = TRUE,
  steps = 100,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
```

```
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is

technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`clockwise` A single boolean value indicates clockwise or not.

`steps` An integer indicating the number of steps to generate the pie chart radian. Increasing this value results in a smoother pie circular.

`lineend` Line end style (round, butt, square).

`linejoin` Line join style (round, mitre, bevel).

`linemitre` Line mitre limit (number greater than 1).

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

new aesthetics

- `angle`: the pie circle angle.
- `angle0`: the initial pie circle angle.
- `radius`: the circle radius.

Aesthetics

`geom_pie()` understands the following aesthetics (required aesthetics are in bold):

- `x`
- `y`
- `angle`
- `alpha`
- `angle0`
- `colour`

- fill
- group
- linetype
- linewidth
- radius

Learn more about setting these aesthetics in `vignette("ggplot2-specs", package = "ggplot2")`.

Examples

```
ggplot(data.frame(x = 1:10, y = 1:10, value = 1:10 / sum(1:10))) +
    geom_pie(aes(x, y, angle = value * 360))
```

---

geom_rect3d *Add z-aesthetic for geom_tile*

Description

Add z-aesthetic for geom_tile

Usage

```
geom_rect3d(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

geom_tile3d(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  lineend = "butt",
  linejoin = "round",
  linemitre = 10,
  na.rm = FALSE,
```

```
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following: • The result of calling a position function, such as position_jitter(). This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the position_ prefix. For example, to use position_jitter(), give the position as "jitter". • For more information and other ways to specify the position, see the layer position documentation.
...	Other arguments passed on to layer()'s params argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can *not* be passed through .... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, colour = "red" or linewidth = 3. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the

params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the stat part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The key_glyph argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

lineend — Line end style (round, butt, square).

linejoin — Line join style (round, mitre, bevel).

linemitre — Line mitre limit (number greater than 1).

na.rm — If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.

show.legend — logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

new aesthetics

- z: the third dimention (in the z direction).
- theta: Angle between x-axis and z-axis.

Aesthetics

`geom_rect3d()` understands the following aesthetics (required aesthetics are in bold):

- xmin
- xmax
- ymin
- ymax
- z
- alpha
- colour
- fill
- group

- linetype
- linewidth

Learn more about setting these aesthetics in `vignette("ggplot2-specs", package = "ggplot2")`.

`geom_tile3d()` understands the following aesthetics (required aesthetics are in bold):

- x
- y
- z
- alpha
- colour
- fill
- group
- height
- linetype
- linewidth
- width

Learn more about setting these aesthetics in `vignette("ggplot2-specs", package = "ggplot2")`.

`geom_subrect` *Subdivide Rectangles*

Description

These geoms subdivide rectangles with shared borders into a grid. Both geoms achieve the same result but differ in how the rectangles are parameterized:

- `geom_subrect()`: Defines rectangles using their four corners (`xmin`, `xmax`, `ymin`, `ymax`).
- `geom_subtile()`: Defines rectangles using the center (`x`, `y`) and dimensions (`width`, `height`).

Usage

```
geom_subrect(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  byrow = TRUE,
  direction = NULL,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
```

```
  show.legend = NA,
  inherit.aes = TRUE
)

geom_subtile(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  byrow = TRUE,
  direction = NULL,
  lineend = "butt",
  linejoin = "mitre",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

mapping	Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).
stat	The statistical transformation to use on the data for this layer. When using a geom_*() function to construct a layer, the stat argument can be used the override the default coupling between geoms and stats. The stat argument accepts the following: • A Stat ggproto subclass, for example StatCount. • A string naming the stat. To give the stat as a string, strip the function name of the stat_ prefix. For example, to use stat_count(), give the stat as "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
- A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as `"jitter"`.
- For more information and other ways to specify the position, see the layer position documentation.

`...` Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through `....` Unknown arguments that are not part of the 4 categories below are ignored.

- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
- When constructing a layer using a `stat_*()` function, the `...` argument can be used to pass on parameters to the `geom` part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*()` function, the `...` argument can be used to pass on parameters to the `stat` part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `....` This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

`byrow` A single boolean value indicates whether we should arrange the divided rectangles in the row-major order.

`direction` A string specifying the arrangement direction:

- `"h"`(`horizontal`): Creates a single row (one-row layout).
- `"v"`(`vertical`): Creates a single column (one-column layout).
- `NULL`: Automatically determines the layout dimensions using logic similar to `facet_wrap()`.

`lineend` Line end style (round, butt, square).

`linejoin` Line join style (round, mitre, bevel).

`na.rm` If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

`show.legend` logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

`inherit.aes` If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `borders()`.

Aesthetics

`geom_subrect()` understands the following aesthetics (required aesthetics are in bold):

- xmin
- xmax
- ymin
- ymax
- alpha
- colour
- fill
- group
- linetype
- linewidth

Learn more about setting these aesthetics in `vignette("ggplot2-specs", package = "ggplot2")`.

`geom_subtile()` understands the following aesthetics (required aesthetics are in bold):

- x
- y
- alpha
- colour
- fill
- group
- height
- linetype
- linewidth
- width

Learn more about setting these aesthetics in `vignette("ggplot2-specs", package = "ggplot2")`.

Examples

```
# arranges by row
ggplot(data.frame(value = letters[seq_len(5)])) +
    geom_subtile(aes(x = 1, y = 1, fill = value))

# arranges by column
ggplot(data.frame(value = letters[seq_len(9)])) +
    geom_subtile(aes(x = 1, y = 1, fill = value), byrow = FALSE)

# one-row
ggplot(data.frame(value = letters[seq_len(4)])) +
    geom_subtile(aes(x = 1, y = 1, fill = value), direction = "h")

# one-column
ggplot(data.frame(value = letters[seq_len(4)])) +
    geom_subtile(aes(x = 1, y = 1, fill = value), direction = "v")
```

ggalign *Add ggplot by Aligning discrete or continuous variable*

Description

[Stable]

`ggalign()` is similar to `ggplot` in that it initializes a `ggplot` data and `mapping`. `ggalign()` allowing you to provide data in various formats, including matrices, data frames, or simple vectors. By default, it will inherit from the layout. If a function, it will apply with the layout matrix. `ggalign()` focuses on integrating plots into a layout by aligning the axes.

Usage

```
ggalign(
  data = waiver(),
  mapping = aes(),
  ...,
  size = NULL,
  no_axes = NULL,
  active = NULL
)
```

Arguments

`data`	The following options can be used: • `NULL`: No data is set. • `waiver()`: Inherits the data from the layout matrix. • A `function` (including purrr-like lambda syntax): Applied to the layout matrix to transform the data before use. To transform the final plot data, please use `scheme_data()`. • A `matrix`, `data.frame`, or atomic vector.
`mapping`	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
`...`	<dyn-dots> Additional arguments passed to `fortify_data_frame()`.
`size`	The relative size of the plot, can be specified as a `unit()`. Note that for `circle_layout()`, all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.
`no_axes`	**[Experimental]** Logical; if `TRUE`, removes axes elements for the alignment axis using `theme_no_axes()`. By default, will use the option- `"ggalign.align_no_axes"`.
`active`	A `active()` object that defines the context settings when added to a layout.

ggplot2 specification

ggalign initializes a ggplot object. The underlying data is created using fortify_data_frame(). Please refer to it for more details.

When aligning discrete variables, ggalign() always applies a default mapping for the axis of the data index in the layout. Specifically:

- aes(y = .data$.y) is used for the horizontal stack_layout() (including left and right annotations).
- aes(x = .data$.x) is used for the vertical stack_layout() (including top and bottom annotations) and circle_layout().

The following columns will be added to the data frame to align discrete variables:

- .panel: The panel for the aligned axis. Refers to the x-axis for vertical stack_layout() (including top and bottom annotations), and the y-axis for horizontal stack_layout() (including left and right annotations).
- .names (vec_names()) and .index (vec_size()/NROW()): Character names (if available) and the integer index of the original data.
- .x/.y and .discrete_x/.discrete_y: Integer indices for x/y coordinates, and a factor of the data labels (only applicable when names exist).

It is recommended to use .x/.y, or .discrete_x/.discrete_y as the x/y mapping.

If the data inherits from quad_layout()/ggheatmap(), additional columns will be added:

- .extra_panel: Provides the panel information for the column (left or right annotation) or row (top or bottom annotation).
- .extra_index: The index information for the column (left or right annotation) or row (top or bottom annotation).

Discrete Axis Alignment

It is important to note that we consider rows as observations, meaning vec_size(data)/NROW(data) must match the number of observations along the axis used for alignment (x-axis for a vertical stack layout, y-axis for a horizontal stack layout).

Examples

```
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_top() +
    ggalign() +
    geom_point(aes(y = value))

ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_top(size = 0.5) +
    align_dendro(k = 3L) +
    ggalign(data = NULL, size = 0.2) +
    geom_tile(aes(y = 1L, fill = .panel))
```

ggalignGrob *Generate a plot grob.*

Description

Generate a plot grob.

Usage

```
ggalignGrob(x)
```

Arguments

x	An object to be converted into a grob.

Value

A grob() object.

Examples

```
ggalignGrob(ggplot())
```

ggalign_attr *Get Data from the Attribute Attached by ggalign*

Description

ggalign_attr provides access to supplementary information stored as attributes during the layout rendering process. These attributes, commonly attached during data transformation by functions like fortify_matrix() or fortify_data_frame(), can include essential details such as filtered or supplementary data that inform downstream operations.

An additional attribute, which stores the factor levels, can be accessed with ggalign_lvls.

Usage

```
ggalign_attr(x, field = NULL, check = TRUE)

ggalign_lvls(x)
```

Arguments

x	Data used, typically inherited from the layout quad_layout()/ggheatmap() or stack_layout() object.
field	A string specifying the particular data to retrieve from the attached attribute. If NULL, the entire attached attribute list will be returned.
check	A boolean indicating whether to check if the field exists. If TRUE, an error will be raised if the specified field does not exist.

Details

Attributes attached to the data are especially useful when the input data is transformed in ways that limit access to the complete dataset. For example, `fortify_matrix.MAF()` might filter mutation data while adding attributes that retain important context, such as the total number of observations, for detailed or aggregated analyses. Additionally, it stores the levels of `Variant_Classification` for further usage.

Value

- `ggalign_attr`: The specified data from the attached supplementary data or `NULL` if it is unavailable.
- `ggalign_lvls`: The attached supplementary levels or `NULL` if it is unavailable.

`ggalign_data_set` *Attach supplementary data and levels for ggalign*

Description

Attach supplementary data and levels for ggalign

Usage

```
ggalign_data_set(.data, ..., .lvls = NULL)
```

Arguments

`.data`	Input data for the layout.
`...`	A list of data to be attached.
`.lvls`	A character vector representing the attached levels.

Note

Used by developers in `fortify_matrix()`, `fortify_data_frame()`, and other related methods.

See Also

`ggalign_attr()`/`ggalign_lvls()`

ggalign_stat *Get the statistics from the layout*

Description

Get the statistics from the layout

Usage

```
ggalign_stat(x, ...)

## S3 method for class 'QuadLayout'
ggalign_stat(x, position, ...)

## S3 method for class 'StackLayout'
ggalign_stat(x, what, ...)
```

Arguments

x	A quad_layout()/ggheatmap() or stack_layout() object.
...	Arguments passed to methods.
position	A string of "top", "left", "bottom", or "right".
what	A single number or string of the plot elements in the stack layout.

Value

The statistics

ggcross *Connect two layout crosswise*

Description

ggcross resets the layout ordering index of a stack_cross(). This allows you to add other align_* objects to define a new layout ordering index. Any objects added after ggcross will use this updated layout ordering index. This feature is particularly useful for creating tanglegram visualizations. ggcross() is an alias of ggcross().

Usage

```
ggcross(mapping = aes(), size = NULL, no_axes = NULL, active = NULL)
```

Arguments

mapping	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
size	The relative size of the plot, can be specified as a unit(). Note that for circle_layout(), all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.
no_axes	**[Experimental]** Logical; if TRUE, removes axes elements for the alignment axis using theme_no_axes(). By default, will use the option- "ggalign.align_no_axes".
active	A active() object that defines the context settings when added to a layout.

ggplot2 specification

ggcross() initializes a ggplot data and mapping.

ggcross() always applies a default mapping for the axis of the data index in the layout. This mapping is aes(y = .data$.y) for horizontal stack layout (including left and right annotation) and aes(x = .data$.x) for vertical stack layout (including top and bottom annotation).

The data in the underlying ggplot object will contain following columns:

- .panel: The panel for the aligned axis. Refers to the x-axis for vertical stack_layout() (including top and bottom annotations), and the y-axis for horizontal stack_layout() (including left and right annotations).
- .names (vec_names()) and .index (vec_size()/NROW()): Character names (if available) and the integer index of the original data.
- .hand: a factor indicates the index groups.
- .x/.y and .discrete_x/.discrete_y: Integer indices for x/y coordinates, and a factor of the data labels (only applicable when names exist).

It is recommended to use .x/.y, or .discrete_x/.discrete_y as the x/y mapping.

---

ggfree *Add ggplot to layout without alignment*

Description

[Experimental]

The ggfree() function allows you to incorporate a ggplot object into your layout. Unlike ggalign(), which aligns every axis value precisely, ggfree() focuses on integrating plots into the layout without enforcing strict axis alignment.

Usage

```
ggfree(data = waiver(), ..., size = NULL, active = NULL)

## Default S3 method:
ggfree(data = waiver(), mapping = aes(), ..., size = NULL, active = NULL)
```

Arguments

data	The following options can be used: • NULL: No data is set. • waiver(): Inherits the data from the layout matrix. • A function (including purrr-like lambda syntax): Applied to the layout matrix to transform the data before use. To transform the final plot data, please use scheme_data(). • A matrix, data.frame, or atomic vector.
...	<dyn-dots> Additional arguments passed to fortify_data_frame().
size	The relative size of the plot, can be specified as a unit(). Note that for circle_layout(), all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.
active	A active() object that defines the context settings when added to a layout.
mapping	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.

ggplot2 specification

ggalign initializes a ggplot object. The underlying data is created using fortify_data_frame(). Please refer to this method for more details.

When used in quad_layout()/ggheatmap(), if the data is inherited from the quad_layout() and the other direction aligns discrete variables, following columns will be added:

- .extra_panel: Provides the panel information for the column (left or right annotation) or row (top or bottom annotation).
- .extra_index: The index information for the column (left or right annotation) or row (top or bottom annotation).

Examples

```
ggheatmap(matrix(rnorm(56), nrow = 7)) +
    anno_top() +
    align_dendro() +
    ggfree(mtcars, aes(wt, mpg)) +
    geom_point()
```

---

ggmark *Add a plot to annotate selected observations*

Description

Add a plot to annotate selected observations

Usage

```
ggmark(
  mark,
  data = waiver(),
  mapping = aes(),
  ...,
  group1 = NULL,
  group2 = NULL,
  obs_size = 1,
  size = NULL,
  active = NULL
)
```

Arguments

mark	A mark_draw() object to define how to draw the links. Like mark_line(), mark_tetragon(). Note the names of the pair links will be used to define the panel names so must be unique.
data	The following options can be used: • NULL: No data is set. • waiver(): Inherits the data from the layout matrix. • A function (including purrr-like lambda syntax): Applied to the layout matrix to transform the data before use. To transform the final plot data, please use scheme_data(). • A matrix, data.frame, or atomic vector.
mapping	Default list of aesthetic mappings to use for plot. If not specified, must be supplied in each layer added to the plot.
...	<dyn-dots> Additional arguments passed to fortify_data_frame().
group1, group2	A single boolean value indicating whether to use the panel group information from the layout as the paired groups. By default, if no specific observations are selected in mark, ggmark() will automatically connect all observations and group them according to the layout's defined groups.
obs_size	A single numeric value that indicates the size of a single observation, ranging from (0, 1].
size	The relative size of the plot, can be specified as a unit(). Note that for circle_layout(), all size values will be interpreted as relative sizes, as this layout type adjusts based on the available space in the circular arrangement.
active	A active() object that defines the context settings when added to a layout.

ggplot2 specification

ggmark initializes a ggplot object. The underlying data is created using fortify_data_frame(). Please refer to it for more details.

In addition, the following columns will be added to the data frame:

- `.panel`: the panel for the aligned axis. It means `x-axis` for vertical stack layout (including top and bottom annotation), `y-axis` for horizontal stack layout (including left and right annotation).
- `.names` (`vec_names()`) and `.index` (`vec_size()`/`NROW()`): a character names (only applicable when names exists) and an integer of index of the original data.
- `.hand`: A factor with levels `c("left", "right")` for horizontal stack layouts, or `c("top", "bottom")` for vertical stack layouts, indicating the position of the linked observations.

Examples

```
set.seed(123)
small_mat <- matrix(rnorm(56), nrow = 7)
rownames(small_mat) <- paste0("row", seq_len(nrow(small_mat)))
colnames(small_mat) <- paste0("column", seq_len(ncol(small_mat)))

# mark_line
ggheatmap(small_mat) +
    theme(axis.text.x = element_text(hjust = 0, angle = -60)) +
    anno_right() +
    align_kmeans(3L) +
    ggmark(mark_line(I(1:3) ~ NULL)) +
    geom_boxplot(aes(.names, value)) +
    theme(plot.margin = margin(l = 0.1, t = 0.1, unit = "npc"))

# mark_tetragon
ggheatmap(small_mat) +
    theme(axis.text.x = element_text(hjust = 0, angle = -60)) +
    anno_right() +
    align_kmeans(3L) +
    ggmark(mark_tetragon(I(1:3) ~ NULL)) +
    geom_boxplot(aes(.names, value)) +
    theme(plot.margin = margin(l = 0.1, t = 0.1, unit = "npc"))
```

---

ggoncoplot *Create an OncoPrint*

Description

[Stable]

The `ggoncoplot()` function generates `oncoPrint` visualizations that display genetic alterations in a matrix format. This function is especially useful for visualizing complex genomic data, such as mutations, copy number variations, and other genomic alterations in cancer research.

Usage

```
ggoncoplot(
  data = NULL,
  mapping = aes(),
```

```
  ...,
  map_width = NULL,
  map_height = NULL,
  reorder_row = reorder_column,
  reorder_column = TRUE,
  width = NA,
  height = NA,
  filling = waiver(),
  theme = NULL,
  active = NULL
)

## Default S3 method:
ggoncoplot(
  data = NULL,
  mapping = aes(),
  ...,
  map_width = NULL,
  map_height = NULL,
  reorder_row = reorder_column,
  reorder_column = TRUE,
  width = NA,
  height = NA,
  filling = waiver(),
  theme = NULL,
  active = NULL
)
```

Arguments

data	A character matrix which encodes the alterations, you can use ";", ":", ",", or "|" to separate multiple alterations.
mapping	Default list of aesthetic mappings to use for main plot in the layout. If not specified, must be supplied in each layer added to the main plot.
...	Additional arguments passed to fortify_matrix().
map_width, map_height	A named numeric value defines the width/height of each alterations.
reorder_row	A boolean value indicating whether to reorder the rows based on the frequency of alterations. You can set this to FALSE, then add align_order(~rowSums(!is.na(.x)), reverse = TRUE) to achieve the same result. You may also need to set strit = FALSE in align_order() if there are already groups.
reorder_column	A boolean value indicating whether to reorder the columns based on the characteristics of the alterations. You can set this to FALSE, then add align_reorder(memo_order) to achieve the same result. You may also need to set strit = FALSE in align_reorder() if there are already groups.
width, height	The relative width/height of the main plot, can be a unit object.
filling	Same as ggheatmap(), but only "tile" can be used.

theme — A theme() object used to customize various elements of the layout, including guides, title, subtitle, caption, margins, panel.border, and background. By default, the theme will inherit from the parent layout. It also controls the panel spacing for all plots in the layout.

active — A active() object that defines the context settings when added to a layout.

Details

ggoncoplot() is a wrapper around the ggheatmap() function, designed to simplify the creation of OncoPrint-style visualizations. The function automatically processes the input character matrix by splitting the encoded alterations (delimited by ";", ":", ",", or "|") into individual genomic events and unnesting the columns for visualization.

Value

A HeatmapLayout object.

Examples

```
# A simple example from `ComplexHeatmap`
mat <- read.table(textConnection(
    "s1,s2,s3
g1,snv;indel,snv,indel
g2,,snv;indel,snv
g3,snv,,indel;snv"
), row.names = 1, header = TRUE, sep = ",", stringsAsFactors = FALSE)

ggoncoplot(mat, map_width = c(snv = 0.5), map_height = c(indel = 0.9)) +
    # Note that guide legends from `geom_tile` and `geom_bar` are different.
    # Although they appear similar, the internal mechanisms won't collapse
    # the guide legends. Therefore, we remove the guide legends from
    # `geom_tile`.
    guides(fill = "none") +
    anno_top(size = 0.5) +
    ggalign() +
    geom_bar(aes(fill = value), data = function(x) {
        subset(x, !is.na(value))
    }) +
    anno_right(size = 0.5) +
    ggalign() +
    geom_bar(aes(fill = value), orientation = "y", data = function(x) {
        subset(x, !is.na(value))
    }) &
    scale_fill_brewer(palette = "Dark2", na.translate = FALSE)
```

ggupset *Create an UpSet plot*

Description

[Experimental]

ggupset is a specialized version of quad_discrete(), which simplifies the creation of Upset plot.

Usage

```
ggupset(
  data = NULL,
  mapping = aes(),
  ...,
  direction = "h",
  point = NULL,
  line = NULL,
  rect = NULL,
  width = NA,
  height = NA,
  theme = NULL,
  active = NULL
)
```

Arguments

data	Data used to create the UpSet plot. fortify_matrix() will be used to convert the data to a matrix. Currently, only fortify_matrix.list_upset and fortify_matrix.matrix_upset are suitable for creating an UpSet plot.
mapping	Default list of aesthetic mappings to use for main plot in the layout. If not specified, must be supplied in each layer added to the main plot.
...	Additional arguments passed to fortify_matrix().
direction	A string indicating the direction of the UpSet plot, "h"(horizontal) or "v"(vertical). In a vertical UpSet plot, the columns of the matrix correspond to the sets, and the rows correspond to the intersections. By default, the horizontal UpSet plot is used, where the rows of the matrix correspond to the sets and the columns correspond to the intersections.
point	A list of parameters passed to geom_point().
line	A list of parameters passed to geom_line().
rect	A list of parameters passed to geom_rect().
width, height	The relative width/height of the main plot, can be a unit object.
theme	A theme() object used to customize various elements of the layout, including guides, title, subtitle, caption, margins, panel.border, and background. By default, the theme will inherit from the parent layout. It also controls the panel spacing for all plots in the layout.
active	A active() object that defines the context settings when added to a layout.

ggplot2 specification

The data input will be converted to a matrix using `fortify_matrix()`, and the data in the underlying main plot will contain the following columns:

- `.panel_x` and `.panel_y`: the column and row panel groups.
- `.x` and `.y`: an integer index of `x` and `y` coordinates
- `.discrete_x` and `.discrete_y`: a factor of the data labels (only applicable when `.row_names` and `.column_names` exists).
- `.row_names` and `.column_names`: A character of the row and column names of the original matrix (only applicable when names exist).
- `.row_index` and `.column_index`: the row and column index of the original matrix.
- `value`: the actual matrix value.

Examples

```
set.seed(123)
lt <- list(
    a = sample(letters, 5),
    b = sample(letters, 10),
    c = sample(letters, 15)
)
ggupset(tune(lt)) +
    scale_fill_manual(values = c("#F0F0F0", "white"), guide = "none") +
    scale_color_manual(values = c("grey", "black"), guide = "none") +
    anno_top() +
    ggalign(data = function(d) ggalign_attr(d, "intersection_sizes")) +
    ggplot2::geom_bar(aes(y = .data$value), stat = "identity") +
    anno_right() +
    ggalign(data = function(d) ggalign_attr(d, "set_sizes")) +
    ggplot2::geom_bar(aes(x = .data$value),
        stat = "identity",
        orientation = "y"
    )
```

---

ggwrap *Wrap Arbitrary Graphics to ggplot*

Description

The `ggwrap()` function allows non-ggplot2 elements to be converted into a compliant representation for use with `align_plots()`. This is useful for adding any graphics that can be converted into a `grob` with the `patch()` method.

Usage

```
ggwrap(plot, ..., align = "panel", on_top = FALSE, clip = TRUE, vp = NULL)
```

Arguments

plot	Any graphic that can be converted into a grob using patch().
...	Additional arguments passed to the patch() method.
align	A string specifying the area to place the plot: "full" for the full area, "plot" for the full plotting area (including the axis label), or "panel" for only the actual area where data is drawn.
on_top	A single boolean value indicates whether the graphic plot should be put front-most. Note: the graphic plot will always put above the background.
clip	A single boolean value indicating whether the grob should be clipped if they expand outside their designated area.
vp	A viewport object, you can use this to define the plot area.

Value

A wrapped_plot object that can be directly placed into align_plots().

See Also

- patch.grob() / patch.gList()
- patch.ggplot()
- patch.patch_ggplot()
- patch.patchwork()
- patch.patch()
- patch.trellis()
- patch.formula() / patch.function()
- patch.recordedplot()
- patch.Heatmap()
- patch.HeatmapList()
- patch.HeatmapAnnotation()
- patch.pheatmap()

Examples

```
library(grid)
ggwrap(rectGrob(gp = gpar(fill = "goldenrod")), align = "full") +
    inset(rectGrob(gp = gpar(fill = "steelblue")), align = "panel") +
    inset(textGrob("Here are some text", gp = gpar(color = "black")),
        align = "panel"
    )
p1 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp)) +
    ggtitle("Plot 1")
align_plots(p1, ggwrap(
    ~ plot(mtcars$mpg, mtcars$disp),
    mar = c(0, 2, 0, 0), bg = NA
))
```

hclust2 *Generate Tree Structures with Hierarchical Clustering*

Description

Generate Tree Structures with Hierarchical Clustering

Usage

```
hclust2(
  matrix,
  distance = "euclidean",
  method = "complete",
  use_missing = "pairwise.complete.obs"
)
```

Arguments

matrix	A numeric matrix, or data frame.
distance	A string of distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski". Correlation coefficient can be also used, including "pearson", "spearman" or "kendall". In this way, 1 - cor will be used as the distance. In addition, you can also provide a dist object directly or a function return a dist object. Use NULL, if you don't want to calculate the distance.
method	A string of the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward.D", "ward.D2", "single", "complete", "average" (= UPGMA), "mcquitty" (= WPGMA), "median" (= WPGMC) or "centroid" (= UPGMC). You can also provide a function which accepts the calculated distance (or the input matrix if distance is NULL) and returns a hclust object. Alternative, you can supply an object which can be coerced to hclust.
use_missing	An optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs". Only used when distance is a correlation coefficient string.

Value

A hclust object.

See Also

- cor()
- dist()
- hclust()

Examples

```
hclust2(dist(USArrests), method = "ward.D")
```

`heatmap_layout` *Create a heatmap*

Description

[Stable]

`heatmap_layout` is a specialized version of `quad_discrete()`, which simplifies the creation of heatmap plots by integrating essential elements for a standard heatmap layout, ensuring that the appropriate data mapping and visualization layers are automatically applied. `ggheatmap` is an alias for `heatmap_layout`.

Usage

```
heatmap_layout(
  data = NULL,
  mapping = aes(),
  ...,
  width = NA,
  height = NA,
  filling = waiver(),
  theme = NULL,
  active = NULL
)
```

Arguments

`data`	Default dataset to use for the layout. If not specified, it must be supplied in each plot added to the layout. By default, it will try to inherit from parent layout. `fortify_matrix()` will be used to convert data to a matrix.
`mapping`	Default list of aesthetic mappings to use for main plot in the layout. If not specified, must be supplied in each layer added to the main plot.
`...`	Additional arguments passed to `fortify_matrix()`.
`width, height`	The relative width/height of the main plot, can be a `unit` object.
`filling`	A single string of `"raster"` or `"tile"` to indicate the filling style. By default, `waiver()` is used, which means that if the input matrix has more than 20,000 cells (`nrow * ncol > 20000`), `geom_raster()` will be used for performance efficiency; for smaller matrices, `geom_tile()` will be used. To customize the filling style, set this to `NULL`. For backward compatibility, a single boolean value is acceptable: `TRUE` means `waiver()`, and `FALSE` means `NULL`. By default, the classic heatmap color scheme `scale_fill_gradient2(low = "blue", high = "red")` is utilized for continuous values.

	You can use the options "ggalign.heatmap_continuous_fill" or "ggalign.heatmap_discrete_fil to modify the default heatmap body filling color scale. See scale_fill_continuous() or scale_fill_discrete() for details on option settings.
theme	A theme() object used to customize various elements of the layout, including guides, title, subtitle, caption, margins, panel.border, and background. By default, the theme will inherit from the parent layout. It also controls the panel spacing for all plots in the layout.
active	A active() object that defines the context settings when added to a layout.

Value

A HeatmapLayout object.

ggplot2 specification

The data input will be converted to a matrix using fortify_matrix(), and the data in the underlying main plot will contain the following columns:

- .panel_x and .panel_y: the column and row panel groups.
- .x and .y: an integer index of x and y coordinates
- .discrete_x and .discrete_y: a factor of the data labels (only applicable when .row_names and .column_names exists).
- .row_names and .column_names: A character of the row and column names of the original matrix (only applicable when names exist).
- .row_index and .column_index: the row and column index of the original matrix.
- value: the actual matrix value.

Examples

```
ggheatmap(1:10)
ggheatmap(letters)
ggheatmap(matrix(rnorm(81), nrow = 9L))
```

---

inset *Create a ggplot inset*

Description

Create a ggplot inset

Usage

```
inset(plot, ..., align = "panel", on_top = TRUE, clip = TRUE, vp = NULL)
```

Arguments

plot	Any graphic that can be converted into a grob using patch().
...	Additional arguments passed to the patch() method.
align	A string specifying the area to place the plot: "full" for the full area, "plot" for the full plotting area (including the axis label), or "panel" for only the actual area where data is drawn.
on_top	A single boolean value indicates whether the graphic plot should be put front-most. Note: the graphic plot will always put above the background.
clip	A single boolean value indicating whether the grob should be clipped if they expand outside their designated area.
vp	A viewport object, you can use this to define the plot area.

Value

A patch_inset object, which can be added in ggplot.

See Also

- patch.grob() / patch.gList()
- patch.ggplot()
- patch.patch_ggplot()
- patch.patchwork()
- patch.patch()
- patch.trellis()
- patch.formula() / patch.function()
- patch.recordedplot()
- patch.Heatmap()
- patch.HeatmapList()
- patch.HeatmapAnnotation()
- patch.pheatmap()

Examples

```
library(grid)
p1 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) +
    geom_boxplot(aes(gear, disp, group = gear))
p1 + inset(p2, vp = viewport(0.6, 0.6,
    just = c(0, 0), width = 0.4, height = 0.4
))
```

is_layout *Reports whether* x *is layout object*

Description

Reports whether x is layout object

Usage

```
is_layout(x)

is_quad_layout(x)

is_stack_layout(x)

is_stack_cross(x)

is_circle_layout(x)

is_heatmap_layout(x)

is_ggheatmap(x)
```

Arguments

x	An object to test.

Value

A single boolean value.

Examples

```
is_layout(ggheatmap(1:10))

# for quad_layout()
is_quad_layout(quad_alignb(1:10))
is_quad_layout(quad_alignh(1:10))
is_quad_layout(quad_alignv(1:10))
is_quad_layout(quad_free(mtcars))

# for stack_layout()
is_stack_layout(stack_discrete("h", 1:10))
is_stack_layout(stack_continuous("h", 1:10))

# for heatmap_layout()
is_heatmap_layout(ggheatmap(1:10))
is_ggheatmap(ggheatmap(1:10))
```

layer_order *Change the layer adding order*

Description

This function allows you to change the order in which layers are added to a ggplot.

Usage

```
layer_order(layer, order = 0)
```

Arguments

layer	A layer geometry object to be added.
order	An integer indicating the position at which the layer should be added. If <= 0, the layer will be added at the beginning. If greater than the number of plot layers, it will be added at the end.

Value

A layer_order object.

Examples

```
ggplot(faithfuld, aes(waiting, eruptions)) +
    geom_raster(aes(fill = density)) +
    geom_point(color = "red", size = 1)
ggplot(faithfuld, aes(waiting, eruptions)) +
    geom_raster(aes(fill = density)) +
    layer_order(geom_point(color = "red", size = 1))
```

layout-operator *Layout operator*

Description

[Experimental]

- +: Adds elements to the active plot in the active layout.
- &: Applies elements to all plots in the layout.
- -: Adds elements to multiple plots in the layout.

Arguments

e1	A quad_layout()/ggheatmap() or stack_layout() object.
e2	An object to be added to the plot.

Details

The + operator is straightforward and should be used as needed.

In order to reduce code repetition ggalign provides two operators for adding ggplot elements (geoms, themes, facets, etc.) to multiple/all plots in quad_layout()/ggheatmap() or stack_layout() object: - and &.

Value

A modified Layout object.

Examples

```
set.seed(123)
small_mat <- matrix(rnorm(56), nrow = 7)
ggheatmap(small_mat) +
    anno_top() +
    ggalign() +
    geom_point(aes(y = value))

# `&` operator apply it to all plots
ggheatmap(small_mat) +
    anno_top() +
    align_dendro() &
    theme(panel.border = element_rect(
        colour = "red", fill = NA, linewidth = unit(2, "mm")
    ))

# If the active layout is the annotation stack, the `-` operator will only
# add the elements to all plots in the active annotation stack:
ggheatmap(small_mat) +
    anno_left(size = 0.2) +
    align_dendro(aes(color = branch), k = 3L) +
    align_dendro(aes(color = branch), k = 3L) -
    # Modify the the color scales of all plots in the left annotation
    scale_color_brewer(palette = "Dark2")

# If the active layout is the `stack_layout()` itself, `-`
# applies the elements to all plots in the layout except the nested
# `ggheatmap()`/`quad_layout()`.
stack_alignv(small_mat) +
    align_dendro() +
    ggtitle("I'm from the parent stack") +
    ggheatmap() +
    # remove any active context
    stack_active() +
    align_dendro() +
    ggtitle("I'm from the parent stack") -
    # Modify the the color scales of all plots in the stack layout except the
    # heatmap layout
    scale_color_brewer(palette = "Dark2") -
    # set the background of all plots in the stack layout except the heatmap
```

```
# layout
theme(plot.background = element_rect(fill = "red"))
```

`layout_annotation` *Modify components of the layout*

Description

- modify the theme of the layout

Usage

```
layout_annotation(theme = waiver(), ...)
```

Arguments

`theme`	A `theme()` object used to customize various elements of the plot, including `guides`, `title`, `subtitle`, `caption`, `margins`, `panel.border`, and `background`. By default, the theme will inherit from the parent `layout`.
`...`	These dots are for future extensions and must be empty.

Details

- `guides`, `panel.border`, and `background` will always be used even for the nested `alignpatches` object.
- `title`, `subtitle`, `caption`, and `margins` will be added for the top-level `alignpatches` object only.

Examples

```
p1 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) +
    geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) +
    geom_bar(aes(gear)) +
    facet_wrap(~cyl)
align_plots(
    p1 + theme(plot.background = element_blank()),
    p2 + theme(plot.background = element_blank()),
    p3 + theme(plot.background = element_blank())
) +
    layout_annotation(
        theme = theme(plot.background = element_rect(fill = "red"))
    )
```

layout_design *Define the grid to compose plots in*

Description

To control how different plots are laid out, you need to add a layout design specification. If you are nesting grids, the layout is scoped to the current nesting level.

Usage

```
layout_design(
  ncol = waiver(),
  nrow = waiver(),
  byrow = waiver(),
  widths = waiver(),
  heights = waiver(),
  design = waiver(),
  guides = NA
)
```

Arguments

ncol, nrow	The dimensions of the grid to create - if both are NULL it will use the same logic as facet_wrap() to set the dimensions
byrow	If FALSE the plots will be filled in in column-major order.
widths, heights	The relative widths and heights of each column and row in the grid. Will get repeated to match the dimensions of the grid. The special value of NA will behave as 1null unit unless a fixed aspect plot is inserted in which case it will allow the dimension to expand or contract to match the aspect ratio of the content.
design	Specification of the location of areas in the layout. Can either be specified as a text string or by concatenating calls to area() together.
guides	A string with one or more of "t", "l", "b", and "r" indicating which side of guide legends should be collected. Defaults to waiver(), which inherits from the parent layout. If there is no parent layout, or if NULL is provided, no guides will be collected.

Value

A layout_design object.

Examples

```
p1 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) +
    geom_boxplot(aes(gear, disp, group = gear))
```

```
p3 <- ggplot(mtcars) +
    geom_bar(aes(gear)) +
    facet_wrap(~cyl)
align_plots(p1, p2, p3) +
    layout_design(nrow = 1L)
align_plots(p1, p2, p3) +
    layout_design(ncol = 1L)
```

`layout_title` *Annotate the whole layout*

Description

Annotate the whole layout

Usage

```
layout_title(title = waiver(), subtitle = waiver(), caption = waiver())
```

Arguments

`title`	The text for the title.
`subtitle`	The text for the subtitle for the plot which will be displayed below the title.
`caption`	The text for the caption which will be displayed in the bottom-right of the plot by default.

Value

A `layout_title` object.

Examples

```
p1 <- ggplot(mtcars) +
    geom_point(aes(mpg, disp))
p2 <- ggplot(mtcars) +
    geom_boxplot(aes(gear, disp, group = gear))
p3 <- ggplot(mtcars) +
    geom_bar(aes(gear)) +
    facet_wrap(~cyl)
align_plots(p1, p2, p3) +
    layout_title(title = "I'm title")
```

link_draw *Define the links to connect a pair of observations*

Description

This function allows users to define links between a pair of observations, facilitating the visualization of connections between related data points.

Usage

```
link_draw(.draw, ...)
```

Arguments

`.draw`	A function used to draw the links. The function must return a `grob()` object. If the function does not return a valid `grob`, no drawing will occur. The input data for the function should include a data frame with the coordinates of the pair of observations to be linked.
`...`	<dyn-dots> A list of formulas, where each side of the formula should be an `integer` or `character` index of the original data, or a `range_link()` object defining the linked observations. Use `NULL` to indicate no link on that side. You can also combine these by wrapping them into a single `list()`. If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with `I()` to use the ordering from the layout. You can also use `waiver()` to inherit values from the other group.

See Also

- `link_line()`
- `.link_draw()`

link_line *Link the paired observations with a line*

Description

Link the paired observations with a line

Usage

```
link_line(..., .element = NULL)
```

Arguments

...	<dyn-dots> A list of formulas, where each side of the formula should be an integer or character index of the original data, or a range_link() object defining the linked observations. Use NULL to indicate no link on that side. You can also combine these by wrapping them into a single list(). If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with I() to use the ordering from the layout. You can also use waiver() to inherit values from the other group.
.element	A element_line() object. Vectorized fields will be recycled to match the total number of groups, or you can wrap the element with I() to recycle to match the drawing groups. The drawing groups typically correspond to the product of the number of observations from both sides, as each pair of observations will be linked with a single line.

---

link_tetragon *Link the paired observations with a quadrilateral*

Description

Link the paired observations with a quadrilateral

Usage

```
link_tetragon(..., .element = NULL)
```

Arguments

...	<dyn-dots> A list of formulas, where each side of the formula should be an integer or character index of the original data, or a range_link() object defining the linked observations. Use NULL to indicate no link on that side. You can also combine these by wrapping them into a single list(). If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with I() to use the ordering from the layout. You can also use waiver() to inherit values from the other group.
.element	A element_polygon() object. Vectorized fields will be recycled to match the total number of groups, or you can wrap the element with I() to recycle to match the drawing groups. The drawing groups are usually the same as the defined groups, but they will differ when the defined group of observations is separated and cannot be linked with a single quadrilateral. In such cases, the number of drawing groups will be larger than the number of defined groups.

mark_draw *Define the links to connect the marked observations*

Description

This function allows users to define links between marked observations and plot panel (e.g., for creating visual connections for related data), which could help explain the observations.

Usage

```
mark_draw(.draw, ...)
```

Arguments

`.draw`	A function used to draw the links. The function must return a `grob()` object. If the function does not return a valid `grob`, nothing will be drawn. The input data for the function must contain two arguments: a data frame for the panel side coordinates and a data frame for the marked observation coordinates.
`...`	<dyn-dots> A list of formulas, where each side of the formula should be an `integer` or `character` index of the original data, or a `range_link()` object defining the linked observations. Use `NULL` to indicate no link on that side. You can also combine these by wrapping them into a single `list()`. If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with `I()` to use the ordering from the layout. You can also use `waiver()` to inherit values from the other group.

See Also

- `mark_line()`
- `mark_tetragon()`
- `mark_triangle()`
- `.mark_draw()`

mark_line *Link the observations and the panel with a line*

Description

Link the observations and the panel with a line

Usage

```
mark_line(..., .element = NULL)
```

Arguments

...	<dyn-dots> A list of formulas, where each side of the formula should be an `integer` or `character` index of the original data, or a `range_link()` object defining the linked observations. Use `NULL` to indicate no link on that side. You can also combine these by wrapping them into a single `list()`. If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with `I()` to use the ordering from the layout. You can also use `waiver()` to inherit values from the other group.
.element	A `element_line()` object. Vectorized fields will be recycled to match the total number of groups, or you can wrap the element with `I()` to recycle to match the drawing groups. The drawing groups typically correspond to the number of observations, as each observation will be linked with the plot panel.

---

mark_tetragon *Link the observations and the panel with a quadrilateral*

Description

Link the observations and the panel with a quadrilateral

Usage

```
mark_tetragon(..., .element = NULL)
```

Arguments

...	<dyn-dots> A list of formulas, where each side of the formula should be an `integer` or `character` index of the original data, or a `range_link()` object defining the linked observations. Use `NULL` to indicate no link on that side. You can also combine these by wrapping them into a single `list()`. If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with `I()` to use the ordering from the layout. You can also use `waiver()` to inherit values from the other group.
.element	A `element_polygon()` object. Vectorized fields will be recycled to match the total number of groups, or you can wrap the element with `I()` to recycle to match the drawing groups. The drawing groups are usually the same as the defined groups, but they will differ when the defined group of observations is separated and cannot be linked with a single quadrilateral. In such cases, the number of drawing groups will be larger than the number of defined groups.

mark_triangle *Link the observations and the panel with a triangle*

Description

Link the observations and the panel with a triangle

Usage

```
mark_triangle(..., orientation = "plot", .element = NULL)
```

Arguments

`...`	<dyn-dots> A list of formulas, where each side of the formula should be an `integer` or `character` index of the original data, or a `range_link()` object defining the linked observations. Use `NULL` to indicate no link on that side. You can also combine these by wrapping them into a single `list()`. If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with `I()` to use the ordering from the layout. You can also use `waiver()` to inherit values from the other group.
`orientation`	A single string, either `"plot"` or `"observation"`, indicating the base of the triangle.
`.element`	An `element_polygon()` object. Vectorized fields will be recycled to match the total number of groups, or you can wrap the element with `I()` to recycle to match the drawing groups. • When `orientation` is `"plot"`, the drawing groups typically correspond to the number of observations. • When `orientation` is `"observation"`, the drawing groups usually match the defined groups, but will differ if the defined group of observations is separated and cannot be linked with a single triangle. In this case, the number of drawing groups will be larger than the number of defined groups.

memo_order *Sort matrix for better visualization*

Description

Helper function used to order the Oncoplot samples. Typically, you would use this in combination with `align_reorder()`, e.g., `align_reorder(memo_order)`.

Usage

```
memo_order(x)
```

Arguments

x	A matrix, where NA values will be treated as empty.

Value

A vector of ordering weights.

---

new_tune *Change the shape of the input object*

Description

- new_tune: Creates a new object by wrapping it in a scalar list with the specified attributes and class.
- tune_data: Retrieves the original input data.

Usage

```
new_tune(x, ..., class = character())

tune_data(x)
```

Arguments

x	An R object.
...	Additional attributes passed to structure().
class	A character vector specifying the class name to be added.

---

no_expansion *Remove scale expansion*

Description

Remove scale expansion

Usage

```
no_expansion(borders = "tlbr")
```

Arguments

borders	Which border should be removed? A string containing one or more of "t", "l", "b", "r", "x", and "y".

Value

An object which can be added to ggplot.

order2 *Ordering Permutation*

Description

order2 returns a permutation which rearranges its first argument into ascending order.

Usage

```
order2(x)

## S3 method for class 'hclust'
order2(x)

## S3 method for class 'dendrogram'
order2(x)

## S3 method for class 'ser_permutation_vector'
order2(x)

## S3 method for class 'ser_permutation'
order2(x)

## S3 method for class 'phylo'
order2(x)

## S3 method for class 'memo_weights'
order2(x)
```

Arguments

x	Any objects can be extracting ordering.

Value

An integer vector unless any of the inputs has 2^31 or more elements, when it is a double vector.

Examples

```
order2(hclust2(matrix(rnorm(100L), nrow = 10L)))
```

pair_links *Helper function to create pairs of observation groups*

Description

ggmark() and cross_link() allow users to add links between observations. These functions help define the linked observations. The selected pairs will either be linked together, or each group in the pair will be linked separately to the same plot area.

- pair_links: Helper function to create pairs of observation groups.
- range_link: Helper function to create a range of observations.

Usage

```
pair_links(..., .handle_missing = "error", .reorder = NULL)

range_link(point1, point2)
```

Arguments

...	<dyn-dots> A list of formulas, where each side of the formula should be an integer or character index of the original data, or a range_link() object defining the linked observations. Use NULL to indicate no link on that side. You can also combine these by wrapping them into a single list(). If only the left-hand side of the formula exists, you can input it directly. For integer indices, wrap them with I() to use the ordering from the layout. You can also use waiver() to inherit values from the other group.
.handle_missing	A string of "error" or "remove" indicates the action for handling missing observations.
.reorder	A string of "hand1" or "hand2" indicating whether to reorder the input links to follow the specified layout ordering.
point1, point2	A single integer or character index, defining the lower and higher bounds of the range. For integer indices, wrap them with I() to indicate the ordered index by the layout.

Examples

```
x <- pair_links(
    # group on the left hand only
    c("a", "b"),
    # normally, integer index will be interpreted as the index of the
    # origianl data
    1:2,
    # wrapped with `I()` indicate` the integer index is ordering of the
    # layout
    I(1:2),
```

```
    range_link(1, 6),
    range_link("a", "b"),
    # group on the right hand only
    ~ 1:2,
    ~ c("a", "b"),
    ~ range_link(1, 6),
    # group on the both side
    range_link(1, 6) ~ c("a", "b"),
    # waiver() indicates the right hand is the same of the left hand
    range_link(1, 6) ~ waiver(),
    # the same for the left hand
    waiver() ~ 1:2,
    ~NULL # an empty link
)
x

# we can modify it as usual list
x[[1]] <- NULL # remove the first link
x$a <- ~LETTERS
x

# modify with a list
x[1:2] <- list(~ c("a", "b"), ~ range_link("a", "b"))
x
```

`patch.alignpatches` *Convert Object into a Grob*

Description

The `patch()` function is used by `ggwrap()` and `inset()` to convert objects into a `grob`.

Usage

```
## S3 method for class 'alignpatches'
patch(x, ...)
```

Arguments

`x`	An object to be converted into a `grob`.
`...`	Not used currently.

Value

A `grob` object.

See Also

alignpatches

Other patch methods: patch.Heatmap(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

---

patch.formula *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'formula'
patch(x, ..., device = NULL, name = NULL)

## S3 method for class '`function`'
patch(x, ..., device = NULL, name = NULL)
```

Arguments

x	An object to be converted into a grob.
...	Graphical Parameters passed on to par().
device	A function that opens a graphics device for grid.echo() to work on. By default this is an off-screen, in-memory device based on the pdf device. This default device may not be satisfactory when using custom fonts.
name	A character identifier.

Value

A grob object.

See Also

plot()

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.ggplot(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

patch.ggplot *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'ggplot'
patch(x, ...)
```

Arguments

x	An object to be converted into a grob.
...	Not used currently.

Value

A grob object.

See Also

ggplot

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

patch.grob *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'grob'
patch(x, ...)

## S3 method for class 'gList'
patch(x, ...)
```

Arguments

x	An object to be converted into a grob.
...	Not used currently.

Value

A grob object.

See Also

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.ggplot(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

---

patch.Heatmap *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'Heatmap'
patch(x, ..., device = NULL)

## S3 method for class 'HeatmapList'
patch(x, ..., device = NULL)

## S3 method for class 'HeatmapAnnotation'
patch(x, ..., device = NULL)
```

Arguments

x	An object to be converted into a grob.
...	Additional arguments passed to draw().
device	A function that opens a graphics device for temporary rendering. By default this is an off-screen, in-memory device based on the pdf device, but this default device may not be satisfactory when using custom fonts.

Value

A grob object.

See Also

- Heatmap()
- HeatmapAnnotation()

Other patch methods: patch.alignpatches(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

---

patch.patch *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

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

Arguments

x	An object to be converted into a grob.
...	Not used currently.

Value

A grob object.

See Also

patch

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

patch.patchwork *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'patchwork'
patch(x, ...)
```

Arguments

x	An object to be converted into a grob.
...	Not used currently.

Value

A grob object.

See Also

patchwork

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

patch.patch_ggplot *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'patch_ggplot'
patch(x, ...)
```

Arguments

x	An object to be converted into a grob.
...	Not used currently.

Value

A grob object.

See Also

- patch_titles()
- inset()
- ggwrap()

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch(), patch.patchwork(), patch.pheatmap(), patch.recordedplot(), patch.trellis()

---

patch.pheatmap *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'pheatmap'
patch(x, ...)
```

Arguments

x	An object to be converted into a grob.
...	Not used currently.

Value

A grob object.

See Also

pheatmap()

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.recordedplot(), patch.trellis()

patch.recordedplot *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'recordedplot'
patch(x, ..., device = NULL)
```

Arguments

x	An object to be converted into a grob.
...	Not used currently.
device	A function that opens a graphics device for grid.echo() to work on. By default this is an off-screen, in-memory device based on the pdf device. This default device may not be satisfactory when using custom fonts.

Value

A grob object.

See Also

recordPlot()

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.trellis()

patch.trellis *Convert Object into a Grob*

Description

The patch() function is used by ggwrap() and inset() to convert objects into a grob.

Usage

```
## S3 method for class 'trellis'
patch(x, ..., device = NULL)
```

Arguments

x	An object to be converted into a grob.
...	Arguments passed on to grid::grid.grabExpr
	warn An integer specifying the amount of warnings to emit. 0 means no warnings, 1 means warn when it is certain that the grab will not faithfully represent the original scene. 2 means warn if there's any possibility that the grab will not faithfully represent the original scene.
	wrap A logical indicating how the output should be captured. If TRUE, each non-grob element on the display list is captured by wrapping it in a grob.
	wrap.grobs A logical indicating whether, if we are wrapping elements (wrap=TRUE), we should wrap grobs (or just wrap viewports).
	width,height Size of the device used for temporary rendering.
device	A function that opens a graphics device for temporary rendering. By default this is an off-screen, in-memory device based on the pdf device, but this default device may not be satisfactory when using custom fonts.

Value

A grob object.

See Also

trellis

Other patch methods: patch.Heatmap(), patch.alignpatches(), patch.formula(), patch.ggplot(), patch.grob(), patch.patch(), patch.patch_ggplot(), patch.patchwork(), patch.pheatmap(), patch.recordedplot()

---

patch_titles *Add patch titles to plot borders*

Description

This function extends ggplot2's title functionality, allowing you to add titles to each border of the plot: top, left, bottom, and right.

Usage

```
patch_titles(
  top = waiver(),
  left = waiver(),
  bottom = waiver(),
  right = waiver()
)
```

Arguments

top, left, bottom, right
: A string specifying the title to be added to the top, left, bottom, and right border of the plot.

Details

The appearance and alignment of these patch titles can be customized using theme():

- plot.patch_title/plot.patch_title.*: Controls the text appearance of patch titles. By default, plot.patch_title inherit from plot.title, and settings for each border will inherit from plot.patch_title, with the exception of the angle property, which is not inherited.
- plot.patch_title.position/plot.patch_title.position.*: Determines the alignment of the patch titles. By default, plot.patch_title.position inherit from plot.title.position, and settings for each border will inherit from plot.patch_title. The value "panel" aligns the patch titles with the plot panels. Setting this to "plot" aligns the patch title with the entire plot (excluding margins and plot tags).

Value

A labels object to be added to ggplot.

Examples

```
ggplot(mtcars) +
    geom_point(aes(mpg, disp)) +
    patch_titles(
        top = "I'm top patch title",
        left = "I'm left patch title",
        bottom = "I'm bottom patch title",
        right = "I'm right patch title"
    )
```

---

quad_active *Determine the Active Context of Quad-Layout*

Description

[Stable]

- quad_active: Sets the active context to the quad_layout()/ggheatmap() itself.
- quad_anno: Sets the active context to the specified annotation stack based on the position argument.
- anno_top: A special case of quad_anno with position = "top".
- anno_left: A special case of quad_anno with position = "left".
- anno_bottom: A special case of quad_anno with position = "bottom".
- anno_right: A special case of quad_anno with position = "right".

Usage

```
quad_active(width = NULL, height = NULL)

quad_anno(
  position,
  size = NULL,
  free_guides = waiver(),
  initialize = NULL,
  what = waiver()
)

anno_top(
  size = NULL,
  free_guides = waiver(),
  initialize = NULL,
  what = waiver()
)

anno_left(
  size = NULL,
  free_guides = waiver(),
  initialize = NULL,
  what = waiver()
)

anno_bottom(
  size = NULL,
  free_guides = waiver(),
  initialize = NULL,
  what = waiver()
)

anno_right(
  size = NULL,
  free_guides = waiver(),
  initialize = NULL,
  what = waiver()
)
```

Arguments

width, height	The relative width/height of the main plot, can be a unit object.
position	A string of "top", "left", "bottom", or "right" indicates which annotation stack should be activated.
size	A numeric value or an unit object to set the total height/width of the annotation stack. • If position is "top" or "bottom", size sets the total height of the annotation.

- If position is "left" or "right", size sets the total width of the annotation.

free_guides — Override the guides collection behavior specified in the quad_layout()/ggheatmap() for the annotation stack.

initialize — A boolean indicating whether the annotation stack should be initialized if it is not already. By default, the annotation stack layout will attempt to initialize when the data is compatible. If set to TRUE, and the data in quad_layout()/ggheatmap() is incompatible with the annotation stack, no data will be used in the stack.

what — What should get activated in the annotation stack? A single number or string of the plot elements in the layout. If NULL, will remove any active context.

Details

By default, quad_anno() attempts to initialize the annotation stack layout using data from quad_layout()/ggheatmap(). However, in situations where you want to use different data for the annotation stack, you can set initialize = FALSE and then provide a custom stack_layout().

Value

An object that can be added to quad_layout()/ggheatmap().

See Also

quad_switch()

---

quad_layout *Arrange plots in the quad-side of a main plot*

Description

[Stable]

This function arranges plots around the quad-sides of a main plot, aligning both horizontal and vertical axes, and can handle either discrete or continuous variables.

- If xlim is provided, a continuous variable will be required and aligned in the vertical direction. Otherwise, a discrete variable will be required and aligned.
- If ylim is provided, a continuous variable will be required and aligned in the horizontal direction. Otherwise, a discrete variable will be required and aligned.

The quad_discrete is a special case where both xlim and ylim are not provided.

The quad_continuous is a special case where both xlim and ylim are provided.

For historical reasons, the following aliases are available:

- quad_alignh: Align discrete variables in the horizontal direction and continuous variables in vertical direction.

- quad_alignv: Align discrete variables in the vertical direction and continuous variables in horizontal direction.
- quad_alignb is an alias for quad_discrete.
- quad_free is an alias for quad_continuous.

Usage

```
quad_layout(
  data = waiver(),
  mapping = aes(),
  xlim = waiver(),
  ylim = waiver(),
  ...,
  theme = NULL,
  active = NULL,
  width = NA,
  height = NA
)

quad_alignh(..., ylim = waiver())

quad_alignv(..., xlim = waiver())

quad_discrete(
  data = waiver(),
  mapping = aes(),
  ...,
  theme = NULL,
  active = NULL,
  width = NA,
  height = NA
)

quad_continuous(
  data = waiver(),
  mapping = aes(),
  xlim = NULL,
  ylim = NULL,
  ...,
  theme = NULL,
  active = NULL,
  width = NA,
  height = NA
)
```

Arguments

data	Default dataset to use for the layout. If not specified, it must be supplied in each plot added to the layout. By default, this will attempt to inherit from the parent

	layout. If both `xlim` and `ylim` are provided, a `data frame` is required, and `fortify_data_frame()` will be used to convert the data to a data frame. When inherited by an annotation stack, no transposition will be applied. Otherwise, a `matrix` is required, and `fortify_matrix()` will be used to convert the data to a matrix. When inherited by the column annotation stack, the data will be transposed.
`mapping`	Default list of aesthetic mappings to use for main plot in the layout. If not specified, must be supplied in each layer added to the main plot.
`xlim`, `ylim`	A `continuous_limits()` object specifying the left/lower limit and the right/upper limit of the scale. Used to align the continuous axis.
`...`	Additional arguments passed to `fortify_data_frame()` or `fortify_matrix()`.
`theme`	A `theme()` object used to customize various elements of the layout, including `guides`, `title`, `subtitle`, `caption`, `margins`, `panel.border`, and `background`. By default, the theme will inherit from the parent `layout`. It also controls the panel spacing for all plots in the layout.
`active`	A `active()` object that defines the context settings when added to a layout.
`width`, `height`	The relative width/height of the main plot, can be a `unit` object.

Value

A `QuadLayout` object.

ggplot2 specification

If either `xlim` or `ylim` is not provided, the data input will be converted to a matrix using `fortify_matrix()`, and the data in the underlying main plot will contain the following columns:

- `.panel_x` and `.panel_y`: the column and row panel groups.
- `.x` and `.y`: an integer index of `x` and `y` coordinates
- `.discrete_x` and `.discrete_y`: a factor of the data labels (only applicable when `.row_names` and `.column_names` exists).
- `.row_names` and `.column_names`: A character of the row and column names of the original matrix (only applicable when names exist).
- `.row_index` and `.column_index`: the row and column index of the original matrix.
- `value`: the actual matrix value.

Otherwise, the data input will be used for the main plot.

quad_switch *Determine the Active Context of Quad-Layout*

Description

[Stable]

quad_switch() integrates quad_active() and quad_anno() into one function for ease of use. This function allows you to quickly change the active context of the quad_layout() and its annotations.

hmanno is an alias for quad_switch, with additional arguments for backward compatibility

Usage

```
quad_switch(
  position = NULL,
  size = NULL,
  width = NULL,
  height = NULL,
  free_guides = waiver(),
  initialize = NULL,
  what = waiver()
)

hmanno(
  position = NULL,
  size = NULL,
  width = NULL,
  height = NULL,
  free_guides = waiver(),
  initialize = NULL,
  what = waiver()
)
```

Arguments

position	A string of "top", "left", "bottom", or "right" indicates which annotation stack should be activated. If NULL, it sets the active context to the quad_layout()/ggheatmap() itself.
size	A numeric value or an unit object to set the total height/width of the annotation stack. • If position is "top" or "bottom", size sets the total height of the annotation. • If position is "left" or "right", size sets the total width of the annotation.
width, height	The relative width/height of the main plot, can be a unit object.

free_guides	Override the guides collection behavior specified in the quad_layout()/ggheatmap() for the annotation stack.
initialize	A boolean indicating whether the annotation stack should be initialized if it is not already. By default, the annotation stack layout will attempt to initialize when the data is compatible. If set to TRUE, and the data in quad_layout()/ggheatmap() is incompatible with the annotation stack, no data will be used in the stack.
what	What should get activated in the annotation stack? A single number or string of the plot elements in the layout. If NULL, will remove any active context.

Value

An object that can be added to quad_layout()/ggheatmap().

See Also

quad_active()/quad_anno()

Examples

```
ggheatmap(matrix(rnorm(81), nrow = 9)) +
    anno_top() +
    align_dendro()
```

---

raster_magick *Rasterize the input object*

Description

The function rasterizes input graphical objects (e.g., grob, layer, ggplot) and optionally processes the resulting raster using magick, a powerful image manipulation library. This allows for advanced graphical transformations directly within the plotting pipeline.

Usage

```
raster_magick(x, magick = NULL, ..., res = NULL, interpolate = FALSE)
```

Arguments

x	An object to rasterize, can be a grob(), layer(), ggplot(), or a list of such objects.
magick	A function (purrr-style formula is accepted) that takes an image_read() object as input and returns an object compatible with as.raster(). You can use any of the image_*() functions from the **magick** package to process the raster image.
...	Not used currently.
res	An integer sets the desired resolution in pixels.
interpolate	A logical value indicating whether to linearly interpolate the image (the alternative is to use nearest-neighbour interpolation, which gives a more blocky result).

Value

An object with the same class of the input.

Examples

```
# data generated code was copied from `ComplexHeatmap`
set.seed(123)
small_mat <- matrix(rnorm(56), nrow = 7)
rownames(small_mat) <- paste0("row", seq_len(nrow(small_mat)))
colnames(small_mat) <- paste0("column", seq_len(ncol(small_mat)))
ggheatmap(small_mat, aes(.x, .y), filling = NULL) +
    raster_magick(geom_tile(aes(fill = value)), res = 20)

ggheatmap(small_mat, aes(.x, .y), filling = NULL) +
    # Use `magick::filter_types()` to check available `filter` arguments
    raster_magick(geom_tile(aes(fill = value)),
        magick = function(image) {
            magick::image_resize(image,
                geometry = "50%x", filter = "Lanczos"
            )
        }
    )
```

---

read_example *Read Example Data*

Description

This function reads example data from the file. If no file is specified, it returns a list of available example files.

Usage

```
read_example(file = NULL)
```

Arguments

file	A string representing the name of the example file to be read. If NULL, the function will return a list of available example file names.

Value

If file is NULL, returns a character vector of available example file names. Otherwise, returns the contents of the specified example file, read as an R object.

Examples

```
read_example()
```

`scale_draw_manual` *Scale for* `draw` *aesthetic*

Description

Draw a ggplot2 layer using a grob or a function.

Usage

```
scale_draw_manual(
  ...,
  values,
  aesthetics = "draw",
  breaks = waiver(),
  na.value = NA
)
```

Arguments

`...` Arguments passed on to `ggplot2::discrete_scale`

`name` The name of the scale. Used as the axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.

`labels` One of:

- `NULL` for no labels
- `waiver()` for the default labels computed by the transformation object
- A character vector giving labels (must be same length as `breaks`)
- An expression vector (must be the same length as breaks). See ?plotmath for details.
- A function that takes the breaks as input and returns labels as output. Also accepts rlang lambda function notation.

`limits` One of:

- `NULL` to use the default scale values
- A character vector that defines possible values of the scale and their order
- A function that accepts the existing (automatic) values and returns new ones. Also accepts rlang lambda function notation.

`na.translate` Unlike continuous scales, discrete scales can easily show missing values, and do so by default. If you want to remove missing values from a discrete scale, specify `na.translate = FALSE`.

`drop` Should unused factor levels be omitted from the scale? The default, `TRUE`, uses the levels that appear in the data; `FALSE` includes the levels in the factor. Please note that to display every level in a legend, the layer should use `show.legend = TRUE`.

guide A function used to create a guide or its name. See guides() for more information.

call The call used to construct the scale for reporting messages.

super The super class to use for the constructed scale

values A list of functions (including purrr-like lambda syntax) that define how each cell's grob (graphical object) should be drawn.

aesthetics Character string or vector of character strings listing the name(s) of the aesthetic(s) that this scale works with. This can be useful, for example, to apply colour settings to the colour and fill aesthetics at the same time, via aesthetics = c("colour", "fill").

breaks One of:

- NULL for no breaks
- waiver() for the default breaks (the scale limits)
- A character vector of breaks
- A function that takes the limits as input and returns breaks as output

na.value The aesthetic value to use for missing (NA) values

Details

If you want to combine the functionality of multiple geoms, it can typically be achieved by preparing the data for each geom inside the draw_*() call and sending it off to the different geoms, collecting the output in a grid::gList (a list of grobs) for draw_group() or a grid::gTree (a grob containing multiple child grobs) for draw_panel().

See Also

https://ggplot2.tidyverse.org/reference/ggplot2-ggproto.html

Examples

```
text <- grid::textGrob(
    "ggdraw",
    x = c(0, 0, 0.5, 1, 1),
    y = c(0, 1, 0.5, 0, 1),
    hjust = c(0, 0, 0.5, 1, 1),
    vjust = c(0, 1, 0.5, 0, 1)
)
ggplot(data.frame(x = 1, y = 2)) +
    geom_draw(text)
```

scheme_align *Align Specifications in the Layout*

Description

[Experimental]

The scheme_align() function defines the align Specifications for plots.

Usage

```
scheme_align(guides = NA, free_spaces = NA, free_labs = NA)
```

Arguments

guides	A string with one or more of "t", "l", "b", and "r" indicating which side of guide legends should be collected. Defaults to waiver(), which inherits from the parent layout. If no parent layout, all guides will be collected. If NULL, no guides will be collected.
free_spaces	A string with one or more of "t", "l", "b", and "r" indicating which border spaces should be removed. Defaults to waiver(), which inherits from the parent layout. If no parent, the default is NULL, meaning no spaces are removed. Usually you want to apply this with the whole layout, instead of individual plots.
free_labs	A string with one or more of "t", "l", "b", and "r" indicating which axis titles should be free from alignment. Defaults to waiver(), which inherits from the parent layout. If no parent layout, no axis titles will be aligned. If NULL, all axis titles will be aligned.

Value

A scheme_align object.

Examples

```
set.seed(123)
mat <- matrix(rnorm(72), nrow = 8)
# used in the layout, define the default action for all plots in the layout
ggheatmap(mat) -
    scheme_align(guides = NULL) +
    anno_right() +
    align_dendro(aes(color = branch), k = 3)

# You can also add it for a single plot
ggheatmap(mat) -
    # for all plots in the layout, we default won't collect any guide legends
    scheme_align(guides = NULL) +
    # for the heatmap body, we collect guide legends in the right
    # note, the guide legends will be collected to the right side of the
```

```
    # layout which will overlap the legends in the right annotation
    scheme_align(guides = "r") +
    anno_right() +
    align_dendro(aes(color = branch), k = 3)

# to avoid overlapping, we can also collect the guide legends in the
# right annotation
ggheatmap(mat) -
    scheme_align(guides = NULL) +
    scheme_align(guides = "r") +
    anno_right() +
    align_dendro(aes(color = branch), k = 3) +
    scheme_align(guides = "r")
```

`scheme_data` *Plot data Specifications*

Description

[Experimental]

Transforms the plot data. Many functions in this package require a specific data format to align observations, `scheme_data()` helps reformat data frames as needed.

Usage

```
scheme_data(data, inherit = FALSE)
```

Arguments

`data`	A function to transform the plot data before rendering. Acceptable values include: • `NULL`: No action taken. • `waiver()`: Inherits from the parent layout. • A `function` or purrr-style `formula`: Used to transform the plot data, which should accept a data frame and return a data frame. You can apply this after the parent layout `scheme_data` function, using the `inherit` argument. Use this hook to modify the data for all `geoms` after the layout is created (for matrix data, it has been melted to a long format data frame) but before rendering by `ggplot2`. The returned data must be a data frame for ggplot.
`inherit`	A single boolean value indicates whether to apply the parent `scheme_data` first and then apply the specified `scheme_data` for the plot. Defaults to `FALSE`.

Details

Defaults will attempt to inherit from the parent layout if the actual data is inherited from the parent layout, with one exception: `align_dendro()`, which will not inherit the `scheme_data` by default.

scheme_theme *Plot default theme*

Description

[Experimental]

scheme_theme() serves as the default theme and will always be overridden by any theme() settings applied directly to the plot. The default theme (scheme_theme()) is applied first, followed by any specific theme() settings, even if theme() is added before scheme_theme().

Usage

```
scheme_theme(
  ...,
  line,
  rect,
  text,
  title,
  aspect.ratio,
  axis.title,
  axis.title.x,
  axis.title.x.top,
  axis.title.x.bottom,
  axis.title.y,
  axis.title.y.left,
  axis.title.y.right,
  axis.text,
  axis.text.x,
  axis.text.x.top,
  axis.text.x.bottom,
  axis.text.y,
  axis.text.y.left,
  axis.text.y.right,
  axis.text.theta,
  axis.text.r,
  axis.ticks,
  axis.ticks.x,
  axis.ticks.x.top,
  axis.ticks.x.bottom,
  axis.ticks.y,
  axis.ticks.y.left,
  axis.ticks.y.right,
  axis.ticks.theta,
  axis.ticks.r,
  axis.minor.ticks.x.top,
  axis.minor.ticks.x.bottom,
```

```
    axis.minor.ticks.y.left,
    axis.minor.ticks.y.right,
    axis.minor.ticks.theta,
    axis.minor.ticks.r,
    axis.ticks.length,
    axis.ticks.length.x,
    axis.ticks.length.x.top,
    axis.ticks.length.x.bottom,
    axis.ticks.length.y,
    axis.ticks.length.y.left,
    axis.ticks.length.y.right,
    axis.ticks.length.theta,
    axis.ticks.length.r,
    axis.minor.ticks.length,
    axis.minor.ticks.length.x,
    axis.minor.ticks.length.x.top,
    axis.minor.ticks.length.x.bottom,
    axis.minor.ticks.length.y,
    axis.minor.ticks.length.y.left,
    axis.minor.ticks.length.y.right,
    axis.minor.ticks.length.theta,
    axis.minor.ticks.length.r,
    axis.line,
    axis.line.x,
    axis.line.x.top,
    axis.line.x.bottom,
    axis.line.y,
    axis.line.y.left,
    axis.line.y.right,
    axis.line.theta,
    axis.line.r,
    legend.background,
    legend.margin,
    legend.spacing,
    legend.spacing.x,
    legend.spacing.y,
    legend.key,
    legend.key.size,
    legend.key.height,
    legend.key.width,
    legend.key.spacing,
    legend.key.spacing.x,
    legend.key.spacing.y,
    legend.frame,
    legend.ticks,
    legend.ticks.length,
    legend.axis.line,
    legend.text,
```

```
legend.text.position,
legend.title,
legend.title.position,
legend.position,
legend.position.inside,
legend.direction,
legend.byrow,
legend.justification,
legend.justification.top,
legend.justification.bottom,
legend.justification.left,
legend.justification.right,
legend.justification.inside,
legend.location,
legend.box,
legend.box.just,
legend.box.margin,
legend.box.background,
legend.box.spacing,
panel.background,
panel.border,
panel.spacing,
panel.spacing.x,
panel.spacing.y,
panel.grid,
panel.grid.major,
panel.grid.minor,
panel.grid.major.x,
panel.grid.major.y,
panel.grid.minor.x,
panel.grid.minor.y,
panel.ontop,
plot.background,
plot.title,
plot.title.position,
plot.subtitle,
plot.caption,
plot.caption.position,
plot.tag,
plot.tag.position,
plot.tag.location,
plot.margin,
strip.background,
strip.background.x,
strip.background.y,
strip.clip,
strip.placement,
strip.text,
```

```
  strip.text.x,
  strip.text.x.bottom,
  strip.text.x.top,
  strip.text.y,
  strip.text.y.left,
  strip.text.y.right,
  strip.switch.pad.grid,
  strip.switch.pad.wrap,
  complete = FALSE,
  validate = TRUE
)
```

Arguments

`...`	A `theme()` object or additional element specifications not part of base ggplot2. In general, these should also be defined in the `element tree` argument. `Splicing` a list is also supported.
`line`	all line elements (`element_line()`)
`rect`	all rectangular elements (`element_rect()`)
`text`	all text elements (`element_text()`)
`title`	all title elements: plot, axes, legends (`element_text()`; inherits from `text`)
`aspect.ratio`	aspect ratio of the panel
`axis.title`, `axis.title.x`, `axis.title.y`, `axis.title.x.top`, `axis.title.x.bottom`, `axis.title.y.left`, `axis.title.y.right`	labels of axes (`element_text()`). Specify all axes' labels (`axis.title`), labels by plane (using `axis.title.x` or `axis.title.y`), or individually for each axis (using `axis.title.x.bottom`, `axis.title.x.top`, `axis.title.y.left`, `axis.title.y.right`). `axis.title.*.*` inherits from `axis.title.*` which inherits from `axis.title`, which in turn inherits from `text`
`axis.text`, `axis.text.x`, `axis.text.y`, `axis.text.x.top`, `axis.text.x.bottom`, `axis.text.y.left`, `axis.text.y.right`, `axis.text.theta`, `axis.text.r`	tick labels along axes (`element_text()`). Specify all axis tick labels (`axis.text`), tick labels by plane (using `axis.text.x` or `axis.text.y`), or individually for each axis (using `axis.text.x.bottom`, `axis.text.x.top`, `axis.text.y.left`, `axis.text.y.right`). `axis.text.*.*` inherits from `axis.text.*` which inherits from `axis.text`, which in turn inherits from `text`
`axis.ticks`, `axis.ticks.x`, `axis.ticks.x.top`, `axis.ticks.x.bottom`, `axis.ticks.y`, `axis.ticks.y.left`, `axis.ticks.y.right`, `axis.ticks.theta`, `axis.ticks.r`	tick marks along axes (`element_line()`). Specify all tick marks (`axis.ticks`), ticks by plane (using `axis.ticks.x` or `axis.ticks.y`), or individually for each axis (using `axis.ticks.x.bottom`, `axis.ticks.x.top`, `axis.ticks.y.left`, `axis.ticks.y.right`). `axis.ticks.*.*` inherits from `axis.ticks.*` which inherits from `axis.ticks`, which in turn inherits from `line`

axis.minor.ticks.x.top, axis.minor.ticks.x.bottom, axis.minor.ticks.y.left, axis.minor.ticks.y.right, axis.minor.ticks.theta, axis.minor.ticks.r
: minor tick marks along axes (element_line()). axis.minor.ticks.*.* inherit from the corresponding major ticks axis.ticks.*.*.

axis.ticks.length, axis.ticks.length.x, axis.ticks.length.x.top, axis.ticks.length.x.bottom, axis.ticks.length.y, axis.ticks.length.y.left, axis.ticks.length.y.right, axis.ticks.length.theta, axis.ticks.length.r
: length of tick marks (unit)

axis.minor.ticks.length, axis.minor.ticks.length.x, axis.minor.ticks.length.x.top, axis.minor.ticks.length.x.bottom, axis.minor.ticks.length.y, axis.minor.ticks.length.y.left, axis.minor.ticks.length.y.right, axis.minor.ticks.length.theta, axis.minor.ticks.length.r
: length of minor tick marks (unit), or relative to axis.ticks.length when provided with rel().

axis.line, axis.line.x, axis.line.x.top, axis.line.x.bottom, axis.line.y, axis.line.y.left, axis.line.y.right, axis.line.theta, axis.line.r
: lines along axes (element_line()). Specify lines along all axes (axis.line), lines for each plane (using axis.line.x or axis.line.y), or individually for each axis (using axis.line.x.bottom, axis.line.x.top, axis.line.y.left, axis.line.y.right). axis.line.*.* inherits from axis.line.* which inherits from axis.line, which in turn inherits from line

legend.background
: background of legend (element_rect(); inherits from rect)

legend.margin
: the margin around each legend (margin())

legend.spacing, legend.spacing.x, legend.spacing.y
: the spacing between legends (unit). legend.spacing.x & legend.spacing.y inherit from legend.spacing or can be specified separately

legend.key
: background underneath legend keys (element_rect(); inherits from rect)

legend.key.size, legend.key.height, legend.key.width
: size of legend keys (unit); key background height & width inherit from legend.key.size or can be specified separately

legend.key.spacing, legend.key.spacing.x, legend.key.spacing.y
: spacing between legend keys given as a unit. Spacing in the horizontal (x) and vertical (y) direction inherit from legend.key.spacing or can be specified separately.

legend.frame
: frame drawn around the bar (element_rect()).

legend.ticks
: tick marks shown along bars or axes (element_line())

legend.ticks.length
: length of tick marks in legend (unit)

legend.axis.line
: lines along axes in legends (element_line())

legend.text
: legend item labels (element_text(); inherits from text)

`legend.text.position`
: placement of legend text relative to legend keys or bars ("top", "right", "bottom" or "left"). The legend text placement might be incompatible with the legend's direction for some guides.

`legend.title`
: title of legend (`element_text()`; inherits from `title`)

`legend.title.position`
: placement of legend title relative to the main legend ("top", "right", "bottom" or "left").

`legend.position`
: the default position of legends ("none", "left", "right", "bottom", "top", "inside")

`legend.position.inside`
: A numeric vector of length two setting the placement of legends that have the `"inside"` position.

`legend.direction`
: layout of items in legends ("horizontal" or "vertical")

`legend.byrow`
: whether the legend-matrix is filled by columns (`FALSE`, the default) or by rows (`TRUE`).

`legend.justification`
: anchor point for positioning legend inside plot ("center" or two-element numeric vector) or the justification according to the plot area when positioned outside the plot

`legend.justification.top`, `legend.justification.bottom`, `legend.justification.left`, `legend.justification.right`, `legend.justification.inside`
: Same as `legend.justification` but specified per `legend.position` option.

`legend.location`
: Relative placement of legends outside the plot as a string. Can be `"panel"` (default) to align legends to the panels or `"plot"` to align legends to the plot as a whole.

`legend.box`
: arrangement of multiple legends ("horizontal" or "vertical")

`legend.box.just`
: justification of each legend within the overall bounding box, when there are multiple legends ("top", "bottom", "left", or "right")

`legend.box.margin`
: margins around the full legend area, as specified using `margin()`

`legend.box.background`
: background of legend area (`element_rect()`; inherits from `rect`)

`legend.box.spacing`
: The spacing between the plotting area and the legend box (`unit`)

`panel.background`
: background of plotting area, drawn underneath plot (`element_rect()`; inherits from `rect`)

`panel.border`
: border around plotting area, drawn on top of plot so that it covers tick marks and grid lines. This should be used with `fill = NA` (`element_rect()`; inherits from `rect`)

panel.spacing, panel.spacing.x, panel.spacing.y
: spacing between facet panels (unit). panel.spacing.x & panel.spacing.y inherit from panel.spacing or can be specified separately.

panel.grid, panel.grid.major, panel.grid.minor, panel.grid.major.x, panel.grid.major.y, panel.grid.minor.x, panel.grid.minor.y
: grid lines (element_line()). Specify major grid lines, or minor grid lines separately (using panel.grid.major or panel.grid.minor) or individually for each axis (using panel.grid.major.x, panel.grid.minor.x, panel.grid.major.y, panel.grid.minor.y). Y axis grid lines are horizontal and x axis grid lines are vertical. panel.grid.*.* inherits from panel.grid.* which inherits from panel.grid, which in turn inherits from line

panel.ontop
: option to place the panel (background, gridlines) over the data layers (logical). Usually used with a transparent or blank panel.background.

plot.background
: background of the entire plot (element_rect(); inherits from rect)

plot.title
: plot title (text appearance) (element_text(); inherits from title) left-aligned by default

plot.title.position, plot.caption.position
: Alignment of the plot title/subtitle and caption. The setting for plot.title.position applies to both the title and the subtitle. A value of "panel" (the default) means that titles and/or caption are aligned to the plot panels. A value of "plot" means that titles and/or caption are aligned to the entire plot (minus any space for margins and plot tag).

plot.subtitle
: plot subtitle (text appearance) (element_text(); inherits from title) left-aligned by default

plot.caption
: caption below the plot (text appearance) (element_text(); inherits from title) right-aligned by default

plot.tag
: upper-left label to identify a plot (text appearance) (element_text(); inherits from title) left-aligned by default

plot.tag.position
: The position of the tag as a string ("topleft", "top", "topright", "left", "right", "bottomleft", "bottom", "bottomright") or a coordinate. If a coordinate, can be a numeric vector of length 2 to set the x,y-coordinate relative to the whole plot. The coordinate option is unavailable for plot.tag.location = "margin".

plot.tag.location
: The placement of the tag as a string, one of "panel", "plot" or "margin". Respectively, these will place the tag inside the panel space, anywhere in the plot as a whole, or in the margin around the panel space.

plot.margin
: margin around entire plot (unit with the sizes of the top, right, bottom, and left margins)

strip.background, strip.background.x, strip.background.y
: background of facet labels (element_rect(); inherits from rect). Horizontal facet background (strip.background.x) & vertical facet background (strip.background.y) inherit from strip.background or can be specified separately

strip.clip
: should strip background edges and strip labels be clipped to the extend of the strip background? Options are "on" to clip, "off" to disable clipping or "inherit" (default) to take the clipping setting from the parent viewport.

`strip.placement`
: placement of strip with respect to axes, either "inside" or "outside". Only important when axes and strips are on the same side of the plot.

`strip.text`, `strip.text.x`, `strip.text.y`, `strip.text.x.top`, `strip.text.x.bottom`, `strip.text.y.left`, `strip.text.y.right`
: facet labels (`element_text()`; inherits from `text`). Horizontal facet labels (`strip.text.x`) & vertical facet labels (`strip.text.y`) inherit from `strip.text` or can be specified separately. Facet strips have dedicated position-dependent theme elements (`strip.text.x.top`, `strip.text.x.bottom`, `strip.text.y.left`, `strip.text.y.right`) that inherit from `strip.text.x` and `strip.text.y`, respectively. As a consequence, some theme stylings need to be applied to the position-dependent elements rather than to the parent elements

`strip.switch.pad.grid`
: space between strips and axes when strips are switched (`unit`)

`strip.switch.pad.wrap`
: space between strips and axes when strips are switched (`unit`)

`complete`
: set this to `TRUE` if this is a complete theme, such as the one returned by `theme_grey()`. Complete themes behave differently when added to a ggplot object. Also, when setting `complete = TRUE` all elements will be set to inherit from blank elements.

`validate`
: `TRUE` to run `validate_element()`, `FALSE` to bypass checks.

Theme inheritance

Theme elements inherit properties from other theme elements hierarchically. For example, `axis.title.x.bottom` inherits from `axis.title.x` which inherits from `axis.title`, which in turn inherits from `text`. All text elements inherit directly or indirectly from `text`; all lines inherit from `line`, and all rectangular objects inherit from `rect`. This means that you can modify the appearance of multiple elements by setting a single high-level component.

Learn more about setting these aesthetics in `vignette("ggplot2-specs")`.

See Also

`+.gg()` and %+replace%, `element_blank()`, `element_line()`, `element_rect()`, and `element_text()` for details of the specific theme elements.

The modifying theme components and theme elements sections of the online ggplot2 book.

Examples

```
set.seed(123)
small_mat <- matrix(rnorm(56), nrow = 8)
ggheatmap(small_mat) +
    scheme_theme(plot.background = element_rect(fill = "red"))

# `scheme_theme()` serves as the default theme and will always be
# overridden by any `theme()` settings applied directly to the plot
ggheatmap(small_mat) +
    theme(plot.background = element_rect(fill = "blue")) +
    scheme_theme(plot.background = element_rect(fill = "red"))
```

stack_cross *Arrange plots crosswise horizontally or vertically*

Description

[Experimental]

The stack_cross function is derived from stack_discrete() and allows for different layout ordering indices within a single layout.

Two aliases are provided for convenience:

- stack_crossv: A special case of stack_cross that sets direction = "v" for vertical alignment.
- stack_crossh: A special case of stack_cross that sets direction = "h" for horizontal alignment.

Usage

```
stack_cross(direction, data = NULL, ..., theme = NULL, sizes = NA)

stack_crossv(data = NULL, ...)

stack_crossh(data = NULL, ...)
```

Arguments

direction	A string indicating the direction of the stack layout, either "h"(horizontal) or "v"(vertical).
data	Default dataset to use for the layout. If not specified, it must be supplied in each plot added to the layout, fortify_matrix() will be used to convert the data to a matrix.
...	Additional arguments passed to fortify_matrix().
theme	A theme() object used to customize various elements of the layout, including guides, title, subtitle, caption, margins, panel.border, and background. By default, the theme will inherit from the parent layout. It also controls the panel spacing for all plots in the layout.
sizes	A numeric value or a unit object. When used for the quad_layout() annotation, it must be of length 1. When used in the stack_layout() with a nested quad_layout(), it should be of length 3, specifying the relative heights (for direction = "h") or widths (for direction = "v") to be applied to the layout.

See Also

ggcross()

stack_layout *Arrange plots horizontally or vertically*

Description

[Stable]

If `limits` is provided, a continuous variable will be required and aligned in the direction specified (`stack_continuous`). Otherwise, a discrete variable will be required and aligned (`stack_discrete`).

Several aliases are provided for convenience:

- `stack_vertical`: A special case of `stack_layout` that sets `direction = "v"`.
- `stack_horizontal`: A special case of `stack_layout` that sets `direction = "h"`.
- `stack_discretev`: A special case of `stack_discrete` that sets `direction = "v"`.
- `stack_discreteh`: A special case of `stack_discrete` that sets `direction = "h"`.
- `stack_continuousv()`: A special case of `stack_free` that sets `direction = "v"`.
- `stack_continuoush()`: A special case of `stack_free` that sets `direction = "h"`.

For historical reasons, the following aliases are available:

- `stack_align` is an alias for `stack_discrete`.
- `stack_alignv` is an alias for `stack_discretev`.
- `stack_alignh` is an alias for `stack_discreteh`.
- `stack_free` is an alias for `stack_continuous`.
- `stack_freev` is an alias for `stack_continuousv`.
- `stack_freeh` is an alias for `stack_continuoush`.

Usage

```
stack_layout(
  direction,
  data = NULL,
  ...,
  theme = NULL,
  sizes = NA,
  limits = waiver()
)

stack_horizontal(data = NULL, ..., limits = waiver())

stack_vertical(data = NULL, ..., limits = waiver())

stack_discrete(direction, data = NULL, ..., theme = NULL, sizes = NA)

stack_discretev(data = NULL, ...)
```

```
stack_discreteh(data = NULL, ...)

stack_continuous(
  direction,
  data = NULL,
  ...,
  limits = NULL,
  theme = NULL,
  sizes = NA
)

stack_continuousv(data = NULL, ...)

stack_continuoush(data = NULL, ...)
```

Arguments

direction	A string indicating the direction of the stack layout, either "h"(horizontal) or "v"(vertical).
data	Default dataset to use for the layout. If not specified, it must be supplied in each plot added to the layout: • If limits is not provided, fortify_matrix() will be used to get a matrix. • If limits is specified, fortify_data_frame() will be used to get a data frame.
...	Additional arguments passed to fortify_data_frame() or fortify_matrix().
theme	A theme() object used to customize various elements of the layout, including guides, title, subtitle, caption, margins, panel.border, and background. By default, the theme will inherit from the parent layout. It also controls the panel spacing for all plots in the layout.
sizes	A numeric value or a unit object. When used for the quad_layout() annotation, it must be of length 1. When used in the stack_layout() with a nested quad_layout(), it should be of length 3, specifying the relative heights (for direction = "h") or widths (for direction = "v") to be applied to the layout.
limits	A continuous_limits() object specifying the left/lower limit and the right/upper limit of the scale. Used to align the continuous axis.

Value

A StackLayout object.

Examples

```
set.seed(123)
small_mat <- matrix(rnorm(56), nrow = 7L)

stack_horizontal(small_mat) + align_dendro()
```

```
# this is the same with:
stack_discrete("h", small_mat) + align_dendro()

stack_discreteh(small_mat) + align_dendro()

# For vertical layout:
stack_vertical(small_mat) + align_dendro()
```

---

stack_switch *Determine the active context of stack layout*

Description

[Stable]

`stack_active` is an alias for `stack_switch()`, which sets `what = NULL` by default.

Usage

```
stack_switch(sizes = NULL, what = waiver(), ...)

stack_active(sizes = NULL, ...)
```

Arguments

`sizes`	A numeric value or a `unit` object. When used for the `quad_layout()` annotation, it must be of length `1`. When used in the `stack_layout()` with a nested `quad_layout()`, it should be of length `3`, specifying the relative heights (for `direction = "h"`) or widths (for `direction = "v"`) to be applied to the layout.
`what`	What should get activated for the stack layout? A single number or string of the plot elements in the layout. If `NULL`, will remove any active context, this is useful when the active context is a `quad_layout()` object, where any `align_*()` will be added to the `quad_layout()`. By removing the active context, we can add `align_*()` into the `stack_layout()`.
`...`	These dots are for future extensions and must be empty.

Value

A `stack_switch` object which can be added to `stack_layout()`.

Examples

```
stack_discrete("h", matrix(1:9, nrow = 3L)) +
    ggheatmap() +
    # ggheamtap will set the active context, directing following addition
    # into the heatmap plot area. To remove the heatmap active context,
    # we can use `stack_active()` which will direct subsequent addition into
    # the stack
```

```
  stack_active() +
  # here we add a dendrogram to the stack.
  align_dendro()
```

`theme_no_axes` *Remove axis elements*

Description

Remove axis elements

Usage

```
theme_no_axes(
  axes = "xy",
  text = TRUE,
  ticks = TRUE,
  title = TRUE,
  line = FALSE
)
```

Arguments

`axes`	Which axes elements should be removed? A string containing one or more of `"t"`, `"l"`, `"b"`, `"r"`, `"x"`, and `"y"`.
`text`	If `TRUE`, will remove the axis labels.
`ticks`	If `TRUE`, will remove the axis ticks.
`title`	If `TRUE`, will remove the axis title.
`line`	If `TRUE`, will remove the axis line.

Value

A `theme()` object.

Examples

```
p <- ggplot() +
    geom_point(aes(x = wt, y = qsec), data = mtcars)
p + theme_no_axes()
p + theme_no_axes("b")
p + theme_no_axes("l")
```

tune *Change the shape of the input object*

Description

Change the shape of the input object

Usage

```
tune(data, shape = NULL)
```

Arguments

data	An R object.
shape	Usually NULL or a string, specifying the new shape for the object. Refer to the detailed method for allowed values.

Details

In most cases, fortify_matrix() or fortify_data_frame() provide full support for transforming objects. However, some objects may require two completely different approaches to be fortified. The tune function acts as a helper to create a new class tailored for these objects.

See Also

- tune.MAF()
- tune.list()
- tune.matrix()

tune.list *Convert the shape of a list for fortify method*

Description

Convert the shape of a list for fortify method

Usage

```
## S3 method for class 'list'
tune(data, shape = NULL)
```

Arguments

data	A list
shape	Not used currently.

See Also

fortify_matrix.list_upset()

Other tune methods: tune.MAF(), tune.matrix()

---

tune.MAF *Convert the shape of a MAF for fortify method*

Description

Convert the shape of a MAF for fortify method

Usage

```
## S3 method for class 'MAF'
tune(data, shape = NULL)
```

Arguments

data	A MAF object.
shape	Not used currently.

See Also

fortify_matrix.MAF_pathways()

Other tune methods: tune.list(), tune.matrix()

---

tune.matrix *Convert the shape of a matrix for fortify method*

Description

Convert the shape of a matrix for fortify method

Usage

```
## S3 method for class 'matrix'
tune(data, shape = NULL)
```

Arguments

data	A matrix.
shape	Not used currently.

See Also

- fortify_matrix.matrix()
- fortify_matrix.matrix_upset()

Other tune methods: tune.MAF(), tune.list()

---

with_quad *Modify operated Context in* quad_layout()

Description

[Experimental]

The with_quad() function modifies the application context of elements in ggheatmap()/quad_layout(). It controls how objects like themes, scales, or other plot modifications apply to specific annotation stacks or the main plot without altering the currently active layout or plot.

Usage

```
with_quad(x, position = waiver(), main = NULL)
```

Arguments

x	An object which can be added to the ggplot, including **schemes**. See scheme_align(), scheme_data(), and scheme_theme()
position	A string specifying one or more positions- "t", "l", "b", and "r"- to indicate the annotation stack context for x. If NULL, will change the operated context to the quad_layout() itself. For default behaivours, see details section.
main	A single boolean value indicating whether x should apply to the main plot, used only when position is not NULL. By default, if position is waiver() and the active context of quad_layout() is an annotation stack or the active context of stack_layout() is itself, main will be set to TRUE; otherwise, it defaults to FALSE.

Details

Default Behavior when adding object wrapped with with_quad():

For quad_layout() object:

- When ggheatmap()/quad_layout() has no active annotation stack, objects added via + or - operate normally without with_quad().
- When the active annotation stack is set, with_quad() ensures the applied object also modifies:
  - The main plot (by default).
  - Opposite annotation stacks when using -.

For stack_layout() object:

- When the active layout is the `stack_layout()` itself:
  - `-` operator will apply changes to all plots along the `stack_layout()`, which means if the stack layout is in `horizontal`, `-` operator will also add the element to the `left` and `right` annotation, if the stack layout is in `vertical`, `-` operator will also add element to the `top` and `bottom` annotation.
  - `+` operator won't do anything special.
- When the active layout is the nested `ggheatmap()`/`quad_layout()`, the `+/-` operator applies the elements to this nested layout, following the same principles as for `ggheatmap()`/`quad_layout()`.

Value

The original object with an added attribute that sets the specified context.

Examples

```
set.seed(123)
small_mat <- matrix(rnorm(56), nrow = 7)

# By wrapping object with `with_quad()`, the `+` operator will apply the
# object not only to the active plot in the annotation stack, but also to
# the main plot unless specified by `main` argument otherwise.
ggheatmap(small_mat) +
    # initialize the left annotation
    anno_left(size = 0.2) +
    align_dendro() +
    # apply the object not only to the active plot in the annotation stack,
    # but also to the main plot
    with_quad(theme(plot.background = element_rect(fill = "red")))

# the `-` operator will apply changes not only to the active annotation
# stack but also to the opposite one (i.e., bottom if top is active, and
# vice versa). The same principle applies to the left and right annotation.
ggheatmap(small_mat) +
    anno_left(size = 0.2) +
    align_dendro(aes(color = branch), k = 3L) +
    # Change the active layout to the left annotation
    anno_top(size = 0.2) +
    align_dendro(aes(color = branch), k = 3L) +
    anno_bottom(size = 0.2) +
    align_dendro(aes(color = branch), k = 3L) -
    # Modify the color scale of all plots in the bottom and the opposite
    # annotation, in this way, the `main` argument by default would be `TRUE`
    with_quad(scale_color_brewer(palette = "Dark2", name = "Top and bottom"))

# When the `position` argument is manually set, the
# default value of the `main` argument will be `FALSE`.
ggheatmap(small_mat) +
    anno_left(size = 0.2) +
    align_dendro(aes(color = branch), k = 3L) +
    anno_top(size = 0.2) +
    align_dendro(aes(color = branch), k = 3L) +
```

```
anno_bottom(size = 0.2) +
align_dendro(aes(color = branch), k = 3L) -
# Modify the background of all plots in the left and top annotation
with_quad(theme(plot.background = element_rect(fill = "red")), "tl")
```

Index