Package 'ggtreebar'

May 16, 2025

Title Make Treemap Bar Charts with 'ggplot2'

Version 0.1.0

Description Provides 'ggplot2' geoms analogous to 'geom_col()' and 'geom_bar()' that allow for treemaps using 'treemapify' nested within each bar segment. Also provides geometries for subgroup bordering and text annotation.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports grid, ggplot2, treemapify, ggfittext, dplyr, cli

URL https://github.com/hrryt/ggtreebar, https://hrryt.github.io/ggtreebar/

BugReports https://github.com/hrryt/ggtreebar/issues

Depends R (>= 4.1.0)

NeedsCompilation no

Author Harry Thompson [aut, cre, cph]

Maintainer Harry Thompson <harry@mayesfield.uk>

Repository CRAN

Date/Publication 2025-05-16 10:00:01 UTC

Contents

geom_treebar *Treemap Bar Charts*

Description

ggplot2 geoms analogous to ggplot2::geom_bar() and ggplot2::geom_col() that allow for treemaps like with treemapify::geom_treemap() nested within each bar segment.

Usage

```
geom_treebar(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  fixed = NULL,
  layout = "squarified",
  start = "bottomleft",
  ...
)

geom_treecol(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "stack",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  fixed = NULL,
  layout = "squarified",
  start = "bottomleft",
  ...
)
```

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` Override the default connection between `geom_treebar()` and `stat_count()`.

`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.

`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()`.

`fixed` Deprecated. Use `layout = "fixed"` instead. Will be removed in later versions.

`layout` The layout algorithm, one of either 'squarified' (the default), 'scol', 'srow' or 'fixed'. See Details for full details on the different layout algorithms.

`start` The corner in which to start placing the tiles. One of 'bottomleft' (the default), 'topleft', 'topright' or 'bottomright'.

`...` 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.

Details

`data` is split by all aesthetics except for the `subgroup` aesthetics.

A treemap is then drawn using `treemapify::treemapify()` from each section of the `data`, inheriting its aesthetics, and using the `subgroup` aesthetics to determine hierarchy.

Value

A `ggplot2::layer()`.

Aesthetics

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

- `x`
- `y`
- `alpha`
- `colour`
- `fill`
- `linetype`
- `linewidth`
- `subgroup`
- `subgroup2`
- `subgroup3`

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

- `x`
- `y`
- `alpha`
- `colour`
- `fill`
- `linetype`
- `linewidth`
- `subgroup`
- `subgroup2`

- subgroup3

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

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

- x ***or*** y
- group
- weight

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

Computed variables

These are calculated by the 'stat' part of layers and can be accessed with delayed evaluation.

- after_stat(count)
  number of points in bin.
- after_stat(prop)
  groupwise proportion

See Also

geom_treebar_subgroup_border(), geom_treebar_subgroup_text().

Examples

```
library(ggplot2)
ggplot(diamonds, aes(clarity, fill = cut, subgroup = color)) +
  geom_treebar()
ggplot(diamonds, aes(y = cut, fill = color, subgroup = clarity)) +
  geom_treebar(position = "dodge")
```

---

geom_treebar_subgroup_border

Subgroup Borders for Treemap Bar Charts

Description

Add borders to subgroups of a treemap bar chart generated by geom_treebar() or geom_treecol().

Usage

```
geom_treebar_subgroup_border(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
  na.rm = FALSE,
```

```
  show.legend = NA,
  inherit.aes = TRUE,
  fixed = NULL,
  layout = "squarified",
  start = "bottomleft",
  level = "subgroup",
  ...
)

geom_treecol_subgroup_border(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "stack",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  fixed = NULL,
  layout = "squarified",
  start = "bottomleft",
  level = "subgroup",
  ...
)

geom_treebar_subgroup2_border(...)

geom_treecol_subgroup2_border(...)

geom_treebar_subgroup3_border(...)

geom_treecol_subgroup3_border(...)
```

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 over-

ride 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.

`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()`.

`fixed` Deprecated. Use `layout = 'fixed'` instead. Will be removed in later versions.

`layout` The layout algorithm, one of either 'squarified' (the default), 'scol', 'srow' or 'fixed'. See Details for full details on the different layout algorithms.

`start` The corner in which to start placing the tiles. One of 'bottomleft' (the default), 'topleft', 'topright' or 'bottomright'.

`level` One of 'subgroup', 'subgroup2' or 'subgroup3', giving the subgrouping level for which to draw borders. It is recommended to use the aliases `geom_treemap_subgroup2_border()` and `geom_treemap_subgroup3_border()` instead of this argument.

`...` 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.

Details

These functions take the same aesthetic mappings as `geom_treebar()` and `geom_treecol()`, and are to be used in conjunction with them, ensuring that arguments like `position` match where supplied.

Value

A `ggplot2::layer()`.

See Also

`geom_treebar()`, `geom_treebar_subgroup_text()`.

Examples

```
library(ggplot2)
ggplot(diamonds, aes(y = clarity, fill = color, subgroup = color, subgroup2 = cut)) +
  geom_treebar(position = "dodge") +
  geom_treebar_subgroup_border(position = "dodge", linewidth = 2)
```

---

geom_treebar_subgroup_text

Subgroup Text Labels for Treemap Bar Charts

Description

Add text labels to subgroups of a treemap bar chart generated by `geom_treebar()` or `geom_treecol()`.

Usage

```
geom_treebar_subgroup_text(
  mapping = NULL,
  data = NULL,
  stat = "count",
  position = "stack",
```

```
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  padding.x = grid::unit(1, "mm"),
  padding.y = grid::unit(1, "mm"),
  place = "bottom",
  min.size = 4,
  grow = FALSE,
  reflow = FALSE,
  fixed = NULL,
  layout = "squarified",
  start = "bottomleft",
  level = "subgroup",
  ...
)

geom_treecol_subgroup_text(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "stack",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE,
  padding.x = grid::unit(1, "mm"),
  padding.y = grid::unit(1, "mm"),
  place = "bottom",
  min.size = 4,
  grow = FALSE,
  reflow = FALSE,
  fixed = NULL,
  layout = "squarified",
  start = "bottomleft",
  level = "subgroup",
  ...
)

geom_treebar_subgroup2_text(...)

geom_treecol_subgroup2_text(...)

geom_treebar_subgroup3_text(...)

geom_treecol_subgroup3_text(...)
```

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` Override the default connection between `geom_treebar()` and `stat_count()`.

`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.

`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()`.

`padding.x`, `padding.y`

`grid::unit()` object, giving horizontal or vertical padding between text and edge of tile. Defaults to 1 mm.

`place` Where inside the box to place the text. Default is `bottom`; other options are `topleft`, `top`, `topright`, etc.

`min.size` Minimum font size, in points. If provided, text that would need to be shrunk below this size to fit the box will not be drawn. Defaults to 4 pt.

`grow` If `TRUE`, text will be grown as well as shrunk to fill the box.

`reflow` If `TRUE`, text will be reflowed (wrapped) to better fit the box.

`fixed` Deprecated. Use `layout = "fixed"` instead. Will be removed in later versions.

`layout` The layout algorithm, one of either 'squarified' (the default), 'scol', 'srow' or 'fixed'. See Details for full details on the different layout algorithms.

`start` The corner in which to start placing the tiles. One of 'bottomleft' (the default), 'topleft', 'topright' or 'bottomright'.

`level` One of 'subgroup', 'subgroup2' or 'subgroup3', giving the subgrouping level for which to draw text labels. It is recommended to use the aliases `geom_treemap_subgroup2_text()` and `geom_treemap_subgroup3_text()` instead of this argument.

`...` 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.

Details

These functions take the same aesthetic mappings as `geom_treebar()` and `geom_treecol()`, and are to be used in conjunction with them, ensuring that arguments like `position` match where supplied.

Value

A `ggplot2::layer()`.

See Also

`geom_treebar()`, `geom_treebar_subgroup_border()`.

Examples

```
library(ggplot2)
ggplot(diamonds, aes(y = clarity, fill = cut, subgroup = color)) +
  geom_treebar(position = "dodge") +
  geom_treebar_subgroup_text(position = "dodge")
```

Index