Package: gghexsize (via r-universe)

June 2, 2026

Title Make Hexagonal Heatmaps with Varying Hexagon Sizes

Version 0.1.0.9000

Description Create hexagonal heatmaps with 'ggplot2', using the 'size' aesthetic to variably size each hexagon.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports cli, farver, ggplot2, grid, hexbin, rlang, scales, vctrs

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

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

Repository https://hrryt.r-universe.dev

Date/Publication 2025-06-07 00:36:59 UTC

RemoteUrl https://github.com/hrryt/gghexsize

RemoteRef HEAD

RemoteSha 6f1cc0f217fc8e409a6f1492e64e0ddb0278a61a

Contents

`draw_key_hextile` *Hexagon key glyph 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 (see `ggplot2::layer()` or examples below.)

Usage

```
draw_key_hextile(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.

See Also

ggplot2::draw_key, `geom_hextile()`.

Examples

```
library(ggplot2)

d <- ggplot(diamonds, aes(carat, price, linewidth = after_stat(count))) +
  scale_linewidth(trans = "log10")
d + geom_hex(colour = "black")

# key glyphs can be specified by their name
d + geom_hex(colour = "black", key_glyph = "hextile")

# key glyphs can be specified via their drawing function
d + geom_hex(colour = "black", key_glyph = draw_key_hextile)
```

geom_hextile *Hexagonal heatmap of 2d bin summaries sized by bin counts*

Description

Divides the plane into regular hexagons, counts the number of cases in each hexagon, and then (by default) maps the number of cases to the hexagon size and fill. If a `z` aesthetic is provided, the hexagon fill is instead mapped to the summary of `z` with `fun`. `z2` and `z3` are made available in the case that multiple summary statistics are required. Hexagon bins avoid the visual artefacts sometimes generated by the very regular alignment of `ggplot2::geom_bin_2d()`.

Usage

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

stat_summary_hextile(
  mapping = NULL,
  data = NULL,
  geom = "hextile",
  position = "identity",
  ...,
  bins = 30,
  binwidth = NULL,
  drop = TRUE,
  fun = "mean",
  fun.args = list(),
  fun2 = "mean",
  fun2.args = list(),
  fun3 = "mean",
  fun3.args = list(),
  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)`).

`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().
geom, stat	geom,stat Override the default connection between geom_hextile() and stat_summary_hextile(). For more information about overriding these connections, see how the stat and geom arguments work.
bins	numeric vector giving number of bins in both vertical and horizontal directions. Set to 30 by default.
binwidth	Numeric vector giving bin width in both vertical and horizontal directions. Overrides bins if both set.
drop	drop if the output of fun is NA.
fun, fun2, fun3	function for summary.
fun.args, fun2.args, fun3.args	A list of extra arguments to pass to fun

Value

A ggplot2::layer()]

Aesthetics

geom_hextile() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x
- y
- alpha
- colour
- fill
- group
- linetype
- linetype
- size

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

stat_summary_hextile() understands the following aesthetics. Required aesthetics are displayed in bold and defaults are displayed for optional aesthetics:

- x: horizontal position
- y: vertical position
- z, z2, z3: value passed to each summary function

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(x)`, `after_stat(y)`
  location.
- `after_stat(count)`
  number of points in bin.
- `after_stat(density)`
  density of points in bin, scaled to integrate to 1.
- `after_stat(ncount)`
  count, scaled to maximum of 1.
- `after_stat(ndensity)`
  density, scaled to maximum of 1.
- `after_stat(value)`
  number of points in bin, or if `z` is supplied, value of summary statistic from `z`.
- `after_stat(value2)`
  if `z2` is supplied, value of summary statistic from `z2`.
- `after_stat(value3)`
  if `z3` is supplied, value of summary statistic from `z3`.

Controlling binning parameters for the x and y directions

The arguments `bins` and `binwidth` can be set separately for the x and y directions. When given as a scalar, one value applies to both directions. When given as a vector of length two, the first is applied to the x direction and the second to the y direction. Alternatively, these can be a named list containing x and y elements, for example `list(x = 10, y = 20)`.

See Also

`scale_size_tile()`, `draw_key_hextile()`, `ggplot2::stat_summary_hex()`, `ggplot2::stat_bin_hex()`, `ggplot2::geom_hex()`.

Examples

```
library(ggplot2)

d <- ggplot(diamonds, aes(carat, depth, z = price))

# fill: median price in bin
# size: number of points in bin
d +
  geom_hextile(fun = "median") +
  scale_size_tile(limits = c(0, 100))

# fill: mean price in bin
# size: sum of prices in bin
d +
  geom_hextile(aes(z2 = price, size = after_stat(value2)), fun2 = "sum") +
```

```
  scale_size_tile(limits = c(0, 1e5))

# fill: mean price in bin
# size: density, scaled to maximum of 1, weighted by price
d +
  geom_hextile(aes(weight = price, size = after_stat(ndensity))) +
  scale_size_tile(limits = c(0, 0.1))

# fill: number of points in bin
# size: number of points in bin
ggplot(diamonds, aes(carat, depth)) +
  geom_hextile() +
  scale_size_tile(limits = c(0, 100))
```

`scale_size_tile` *Scales for area or radius of bin tiles*

Description

Replacements for `ggplot2::scale_size_area()` and `ggplot2::scale_size_binned_area()` with convenient defaults for `geom_hextile()`.

Usage

```
scale_size_tile(
  name = ggplot2::waiver(),
  ...,
  max_size = 1,
  oob = scales::squish
)

scale_size_binned_tile(
  name = ggplot2::waiver(),
  ...,
  max_size = 1,
  oob = scales::squish
)
```

Arguments

`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.
`...`	Arguments passed on to `ggplot2::continuous_scale()` or `ggplot2::binned_scale()`.
`max_size`	Size of largest points.
`oob`	One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation.
- The default (`scales::squish()`) squishes out of bounds values into range.
- `scales::censor()` for replacing out of bounds values with `NA`.
- `scales::squish_infinite()` for squishing infinite values into range.

Details

These are convenience functions with the following changed defaults:

- `max_size` = 1 rather than 6,
- `oob` = `scales::squish`.

In practice, this makes it easy to set a hard upper limit on a scale, above which sizes are clamped to 1.

Value

A `ggplot2::Scale`.

Examples

```
library(ggplot2)

d <- ggplot(diamonds, aes(carat, depth, z = price)) +
  geom_hextile()
d + scale_size_tile(limits = c(NA, 100))

d + scale_size_binned_tile(transform = "log10")
```

Index