Package ‘gglinedensity’

May 10, 2025

Title Make DenseLines Heatmaps with 'ggplot2'

Version 0.1.0

Description Create a raster plot of line density using a 'ggplot2' statistic
implementing the DenseLines algorithm, which
``normalizes time series by the arc length to compute accurate densities''
(Moritz and Fisher, 2018) <doi:10.48550/arXiv.1808.06019>.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

SystemRequirements Cargo (Rust's package manager), rustc >= 1.65.0

Config/gglinedensity/MSRV 1.65.0

Imports cli, ggplot2, lifecycle, rlang, scales, vctrs

URL `https://github.com/hrryt/gglinedensity`

BugReports `https://github.com/hrryt/gglinedensity/issues`

NeedsCompilation yes

Author Harry Thompson [cre, aut, cph],
Dominik Moritz [aut] (Rust original,
<`https://github.com/domoritz/line-density-rust`>),
The authors of the dependency Rust crates [ctb] (see inst/AUTHORS file
for details),
Hiroaki Yutani [ctb] (ORCID: <`https://orcid.org/0000-0002-3385-7233`>,
Scripts where noted,
<`https://yutannihilation.github.io/string2path`>)

Maintainer Harry Thompson <`harry@mayesfield.uk`>

Repository CRAN

Date/Publication 2025-05-09 23:00:02 UTC

Contents

stat_line_density *Create a DenseLines Heatmap*

Description

A 'ggplot2' statistic implementing the DenseLines algorithm described by Moritz and Fisher (2018).

Usage

```
stat_line_density(
  mapping = NULL,
  data = NULL,
  geom = "raster",
  position = "identity",
  ...,
  bins = 30,
  binwidth = NULL,
  drop = TRUE,
  normalise = TRUE,
  orientation = NA,
  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)`).

`geom` The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.

- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as `"point"`.
- For more information and other ways to specify the geom, see the layer geom 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.

`bins`

Number of bins. Overridden by `binwidth`. Defaults to 30.

`binwidth`

The width of the bins. Can be specified as a numeric value or as a function that takes x after scale transformation as input and returns a single numeric value. When specifying a function along with a grouping structure, the function will be called once per group. The default is to use the number of bins in `bins`, covering the range of the data. You should always override this value, exploring multiple widths to find the best to illustrate the stories in your data.

The bin width of a date variable is the number of days in each time; the bin width of a time variable is the number of seconds.

`drop` if `TRUE` removes all cells with 0 counts.

`normalise` if `TRUE`, the default, density is normalised per group by the sum in each bin vertically, or horizontally if `orientation` is set to `"y"`.

`orientation` The orientation of the layer. The default (`NA`) automatically determines the orientation from the aesthetic mapping. In the rare event that this fails it can be given explicitly by setting `orientation` to either `"x"` or `"y"`. See the *Orientation* section for more detail.

`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. To include legend keys for all levels, even when no data exists, use `TRUE`. If `NA`, all levels are shown in legend, but unobserved levels are omitted.

`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()`.

Value

A `ggplot2::layer()`.

Aesthetics

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

- `x`
- `y`
- `group`

Computed variables

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

- `after_stat(density)`
  density estimate.

Orientation

This geom treats each axis differently and, thus, can thus have two orientations. Often the orientation is easy to deduce from a combination of the given mappings and the types of positional scales in use. Thus, ggplot2 will by default try to guess which orientation the layer should have. Under rare circumstances, the orientation is ambiguous and guessing may fail. In that case the orientation can be specified directly using the `orientation` parameter, which can be either `"x"` or `"y"`. The value gives the axis that the geom should run along, `"x"` being the default orientation you would expect for the geom.

References

Moritz, D. & Fisher, D. (2018). Visualizing a Million Time Series with the Density Line Chart. arXiv preprint arXiv:1409.0473. doi:10.48550/arxiv.1808.06019.

See Also

`ggplot2::stat_bin_2d()`, `ggplot2::geom_line()`, `ggplot2::geom_raster()`.

Examples

```
library(ggplot2)

p <- ggplot(txhousing, aes(date, median, group = city))

p +
  stat_line_density(na.rm = TRUE)

p +
  stat_line_density(
    # map density to colour rather than fill
    aes(colour = after_stat(density)),
    geom = "point", size = 5, na.rm = TRUE
  ) +
  stat_line_density(
    aes(
      # add a label where density > 7
      label = after_stat(ifelse(density > 7, round(density, 2), NA)),
      # label background fill
      fill = after_stat(density)
    ),
    geom = "label", na.rm = TRUE
  ) +
  scale_colour_viridis_c(trans = "log10") +
  scale_fill_viridis_c(trans = "log10")

p +
  stat_line_density(
    # convert to factor for a discrete scale
    aes(fill = after_stat(as.factor(density))),
    normalise = FALSE, drop = FALSE, na.rm = TRUE
  ) +
  geom_text( # equivalent to stat_line_density(geom = "text")
    aes(label = after_stat(ifelse(density > 20, density, NA)), fill = NULL),
    stat = "line_density", # or stat = StatLineDensity
    normalise = FALSE, na.rm = TRUE
  ) +
  scale_fill_ordinal(name = "count")

ggplot(txhousing, aes(median, date, group = city)) +
  stat_line_density(
    # scale the maximum density to 1
    aes(fill = after_stat(density / max(density))),
    bins = 50, orientation = "y", na.rm = TRUE
  ) +
  scale_fill_continuous(name = "density") +
  scale_y_reverse()
```

Index