Package 'ggpointdensity'

May 8, 2026

Type Package

Title A Cross Between a 2D Density Plot and a Scatter Plot

Version 0.2.1

Description A cross between a 2D density plot and a scatter plot, implemented as a 'ggplot2' geom. Points in the scatter plot are colored by the number of neighboring points. This is useful to visualize the 2D-distribution of points in case of overplotting.

URL https://github.com/LKremer/ggpointdensity

BugReports https://github.com/LKremer/ggpointdensity/issues

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Depends R (>= 3.2)

Imports ggplot2, MASS

Suggests viridis, dplyr, testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Lukas P. M. Kremer [aut, cre] (ORCID: <https://orcid.org/0000-0003-3170-6295>), Simon Anders [ctb] (ORCID: <https://orcid.org/0000-0003-4868-1805>)

Maintainer Lukas P. M. Kremer <L-Kremer@web.de>

Repository CRAN

Date/Publication 2025-11-18 12:50:02 UTC

Contents

count_neighbors *Count Neighbors within a Radius*

Description

This function counts the number of neighboring points within a specified radius for each point in a given set of coordinates using a C implementation.

Usage

```
count_neighbors(x, y, r2, xy)
```

Arguments

x	A numeric vector of x-coordinates of the points.
y	A numeric vector of y-coordinates of the points.
r2	A numeric value representing the squared radius within which to search for neighboring points.
xy	A numeric value representing the aspect ratio (usually the ratio of the y-scale to the x-scale).

Value

A numeric vector where each element represents the count of neighboring points within the specified radius for each point.

geom_pointdensity *A cross between a scatter plot and a 2D density plot*

Description

`geom_pointdensity()` visualizes overlapping data points on a 2D coordinate system. It combines the benefits of `geom_point()`, `geom_density2d()`, and `geom_bin2d()` by coloring individual points based on the density of neighboring points. This approach highlights the overall data distribution while preserving the visibility of individual outliers, making it ideal for data exploration.

Usage

```
geom_pointdensity(
  mapping = NULL,
  data = NULL,
  stat = "pointdensity",
  position = "identity",
  ...,
```

```
  method = c("auto", "kde2d", "neighbors"),
  method.args = list(),
  adjust = 1,
  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 to 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.

`method` Density estimation method. Options are `"auto"`, `"neighbors"`, or `"kde2d"`.

- `"auto"` (default): Selects the appropriate method based on the number of points. `"neighbors"` is faster for small datasets, while `"kde2d"` is more efficient for large datasets.
- `"neighbors"`: Determines an appropriate radius and counts the number of points within this radius for each point.
- `"kde2d"`: Uses 2D kernel density estimation via `MASS::kde2d()`. Additional arguments can be provided through `method.args`.

`method.args` List of additional arguments passed on to the density estimation function defined by `method` (e.g. `MASS::kde2d()`).

`adjust` Multiplicative bandwidth adjustment for density estimation. A value less than 1 (e.g., `adjust = 0.1`) yields a smoother density estimate, while a value greater than 1 (e.g., `adjust = 5`) increases the level of visible 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. `annotation_borders()`.

Aesthetics

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

- x
- y
- alpha → NA
- colour → via theme()
- fill → via theme()
- group → inferred
- shape → via theme()
- size → via theme()
- stroke → via theme()

The fill aesthetic only applies to shapes 21-25.

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

Author(s)

Lukas PM Kremer & Simon Anders

See Also

You can find examples and demo plots at https://github.com/LKremer/ggpointdensity

Examples

```
library(ggpointdensity)
library(ggplot2)
library(dplyr)

# generate some toy data
dat <- bind_rows(
  tibble(x = rnorm(3500, sd = 1),
         y = rnorm(3500, sd = 10),
         group = "foo"),
  tibble(x = rnorm(1500, mean = 1, sd = .5),
         y = rnorm(1500, mean = 7, sd = 5),
         group = "bar"))

# plot it with geom_pointdensity()
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity()

# adjust the smoothing bandwidth,
# i.e. the radius around the points
# in which neighbors are counted
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity(adjust = .1)

ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity(adjust = 4)

ggplot(data = dat, mapping = aes(x = x, y = y)) +
```

```
  geom_pointdensity(adjust = 4) +
  scale_colour_continuous(low = "red", high = "black")

# I recommend the viridis package
# for a more useful color scale
library(viridis)
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis()

# Of course you can combine the geom with standard
# ggplot2 features such as facets...
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis() +
  facet_wrap(~ group)

# ... or point shape and size:
dat_subset <- sample_frac(dat, .1)  #' smaller data set
ggplot(data = dat_subset, mapping = aes(x = x, y = y)) +
  geom_pointdensity(size = 3, shape = 17) +
  scale_color_viridis()

# Zooming into the axis works as well, keep in mind
# that xlim() and ylim() affect the density since they
# remove data points.
# It may be better to use coord_cartesian() instead.
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis() +
  xlim(c(-1, 3)) + ylim(c(-5, 15))

ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis() +
  coord_cartesian(xlim = c(-1, 3), ylim = c(-5, 15))
```

`stat_pointdensity` *A cross between a scatter plot and a 2D density plot*

Description

`geom_pointdensity()` visualizes overlapping data points on a 2D coordinate system. It combines the benefits of `geom_point()`, `geom_density2d()`, and `geom_bin2d()` by coloring individual points based on the density of neighboring points. This approach highlights the overall data distribution while preserving the visibility of individual outliers, making it ideal for data exploration.

Usage

```
stat_pointdensity(
  mapping = NULL,
  data = NULL,
  geom = "point",
  position = "identity",
  ...,
  adjust = 1,
  na.rm = FALSE,
  method = c("auto", "kde2d", "neighbors"),
  method.args = list(),
  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, defaults to "point".
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.

adjust — Multiplicative bandwidth adjustment for density estimation. A value less than 1 (e.g., adjust = 0.1) yields a smoother density estimate, while a value greater than 1 (e.g., adjust = 5) increases the level of visible detail.

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

method — Density estimation method. Options are "auto", "neighbors", or "kde2d".

- "auto" (default): Selects the appropriate method based on the number of points. "neighbors" is faster for small datasets, while "kde2d" is more efficient for large datasets.
- "neighbors": Determines an appropriate radius and counts the number of points within this radius for each point.
- "kde2d": Uses 2D kernel density estimation via MASS::kde2d(). Additional arguments can be provided through method.args.

method.args — List of additional arguments passed on to the density estimation function defined by method (e.g. MASS::kde2d()).

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. annotation_borders().

Aesthetics

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

- x
- y
- alpha → NA
- colour → via theme()
- fill → via theme()
- group → inferred
- shape → via theme()
- size → via theme()
- stroke → via theme()

The fill aesthetic only applies to shapes 21-25.

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

Author(s)

Lukas PM Kremer & Simon Anders

See Also

You can find examples and demo plots at https://github.com/LKremer/ggpointdensity

Examples

```
library(ggpointdensity)
library(ggplot2)
library(dplyr)

# generate some toy data
dat <- bind_rows(
  tibble(x = rnorm(3500, sd = 1),
         y = rnorm(3500, sd = 10),
         group = "foo"),
  tibble(x = rnorm(1500, mean = 1, sd = .5),
         y = rnorm(1500, mean = 7, sd = 5),
         group = "bar"))

# plot it with geom_pointdensity()
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity()

# adjust the smoothing bandwidth,
# i.e. the radius around the points
# in which neighbors are counted
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity(adjust = .1)

ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity(adjust = 4)

ggplot(data = dat, mapping = aes(x = x, y = y)) +
```

```
  geom_pointdensity(adjust = 4) +
  scale_colour_continuous(low = "red", high = "black")

# I recommend the viridis package
# for a more useful color scale
library(viridis)
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis()

# Of course you can combine the geom with standard
# ggplot2 features such as facets...
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis() +
  facet_wrap(~ group)

# ... or point shape and size:
dat_subset <- sample_frac(dat, .1)  #' smaller data set
ggplot(data = dat_subset, mapping = aes(x = x, y = y)) +
  geom_pointdensity(size = 3, shape = 17) +
  scale_color_viridis()

# Zooming into the axis works as well, keep in mind
# that xlim() and ylim() affect the density since they
# remove data points.
# It may be better to use coord_cartesian() instead.
ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis() +
  xlim(c(-1, 3)) + ylim(c(-5, 15))

ggplot(data = dat, mapping = aes(x = x, y = y)) +
  geom_pointdensity() +
  scale_color_viridis() +
  coord_cartesian(xlim = c(-1, 3), ylim = c(-5, 15))
```

Index