Package 'PieGlyph'

August 22, 2025

Title Axis Invariant Scatter Pie Plots

Version 1.1.0

Description
Extends 'ggplot2' to help replace points in a scatter plot with pie-chart glyphs showing the relative proportions of different categories. The pie glyphs are independent of the axes and plot dimensions, to prevent distortions when the plot dimensions are changed.

License GPL (>= 3)

Encoding UTF-8

Imports ggplot2, dplyr, tidyr, rlang, ggiraph, ggforce, purrr, forcats, plyr, grid, scales, cli, utils

Suggests spelling, ranger, maps, cowplot, mapproj, knitr, rmarkdown, testthat (>= 3.0.0), vdiffr

RoxygenNote 7.3.2

VignetteBuilder knitr

URL https://rishvish.github.io/PieGlyph/, https://github.com/rishvish/PieGlyph

BugReports https://github.com/rishvish/PieGlyph/issues

Config/testthat/edition 3

Language en-US

NeedsCompilation no

Author Rishabh Vishwakarma [aut, cre] (ORCID: <https://orcid.org/0000-0002-4847-3494>), Caroline Brophy [aut], Catherine Hurley [aut]

Maintainer Rishabh Vishwakarma <vishwakr@tcd.ie>

Repository CRAN

Date/Publication 2025-08-22 18:50:11 UTC

Contents

`draw_key_pie` *Legend key for the pie glyphs*

Description

Controls the aesthetics of the legend entries for the pie glyphs

Usage

```
draw_key_pie(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

`draw_key`

geom_pie_glyph *Scatter plot with points replaced by axis-invariant pie-chart glyphs*

Description

This geom replaces the points in a scatter plot with pie-chart glyphs showing the relative proportions of different categories. The pie-chart glyphs are independent of the plot dimensions, so won't distort when the plot is scaled. The ideal dataset for this geom would contain columns with non-negative values showing the magnitude of the different categories to be shown in the pie glyphs (The proportions of the different categories within the pie glyph will be calculated automatically). The different categories can also be stacked together into a single column according to the rules of tidy-data (see vignette('tidy-data') or vignette('pivot') for more information).

Usage

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

Arguments

mapping	Set of aesthetic (see Aesthetics below) mappings to be created by aes() or 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 of the plot. The default, NULL, inherits the plot data specified in the ggplot() call. 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)).
slices	Each pie glyph in the plot shows the relative abundances of a set of categories; those categories are specified by this argument and should contain numeric and non-negative values. The names of the categories can be the names of individual columns (wide format) or can be stacked and contained in a single column (long

format using pivot_longer()). The categories can also be specified as the numeric indices of the columns.

values — This parameter is not needed if the data is in wide format. The default is NA assuming that the categories are in separate columns. However, if the pie categories are stacked in one column, this parameter describes the column for the values of the categories shown in the pie glyphs. The values should be numeric and non-negative and the proportions of the different slices within each observation will be calculated automatically.

stat — The statistical transformation to use on the data for this layer, either as a ggproto Geom subclass or as a string naming the stat stripped of the stat_ prefix (e.g. "count" rather than "stat_count")

position — Position adjustment, either as a string naming the adjustment (e.g. "jitter" to use position_jitter), or the result of a call to a position adjustment function. Use the latter if you need to change the settings of the adjustment.

na.rm — If all slices for an observation are NA or 0, the observation is dropped while if at least one slice is not NA, the other slices with value NA are assumed to be 0. This parameter indicates whether the user is notified about these changes. If FALSE, the default, user is given a warning. If TRUE, the problematic observations are silently removed/modified to 0, without notifying the user.

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.

inherit.aes — If FALSE, overrides the default aesthetics, rather than combining with them

... — Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or radius = 1. They may also be parameters to the paired geom/stat.

Value

A ggplot layer

Aesthetics

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

- **x** - variable to be shown on X-axis.
- **y** - variable to be shown on Y-axis.
- alpha - adjust opacity of the pie glyphs.
- radius - adjust the radius of the pie glyphs (in cm).
- colour - specify colour of the border of pie glyphs.
- linetype - specify style of pie glyph borders.
- linewidth - specify width of pie glyph borders (in mm).
- group - specify grouping structure for the observations (see grouping for more details).
- pie_group - manually specify a grouping variable for separating pie-glyphs with identical x and y coordinates (see vignette("unusual-situations") for more information).

Examples

```
## Load libraries
library(dplyr)
library(tidyr)
library(ggplot2)

## Simulate raw data
set.seed(123)
plot_data <- data.frame(response = rnorm(10, 100, 30),
                        system = as.factor(1:10),
                        group = sample(size = 10,
                                       x = c("G1", "G2", "G3"),
                                       replace = TRUE),
                        A = round(runif(10, 3, 9), 2),
                        B = round(runif(10, 1, 5), 2),
                        C = round(runif(10, 3, 7), 2),
                        D = round(runif(10, 1, 9), 2))

head(plot_data)

## Basic plot
ggplot(data = plot_data, aes(x = system, y = response))+
   geom_pie_glyph(slices = c("A", "B", "C", "D"),
                  data = plot_data)+
   theme_classic()


## Change pie radius using `radius` and border colour using `colour`
ggplot(data = plot_data, aes(x = system, y = response))+
       # Can also specify slices as column indices
       geom_pie_glyph(slices = 4:7, data = plot_data,
                      colour = "black", radius = 0.5)+
       theme_classic()


## Map radius to a variable
p <- ggplot(data = plot_data, aes(x = system, y = response))+
       geom_pie_glyph(aes(radius = group),
                      slices = c("A", "B", "C", "D"),
                      data = plot_data, colour = "black")+
                      theme_classic()
p


## Add custom labels
p <- p + labs(x = "System", y = "Response",
              fill = "Attributes", radius = "Group")
p


## Change slice colours
p + scale_fill_manual(values = c("#56B4E9", "#CC79A7",
```

```
                                    "#F0E442", "#D55E00"))

##### Stack the attributes in one column
# The attributes can also be stacked into one column to generate
# the plot. This variant of the function is useful for situations
# when the data is in tidy format. See vignette("tidy-data") and
# vignette("pivot") for more information.

plot_data_stacked <- plot_data %>%
                        pivot_longer(cols = c("A", "B", "C", "D"),
                                     names_to = "Attributes",
                                     values_to = "values")
head(plot_data_stacked, 8)


ggplot(data = plot_data_stacked, aes(x = system, y = response))+
  # Along with slices column, values column is also needed now
  geom_pie_glyph(slices = "Attributes", values = "values")+
  theme_classic()
```

`geom_pie_interactive` *Scatter plots with interactive pie-chart glyphs*

Description

This geom is based on geom_pie_glyph and replaces points in a scatter plot with interactive pie-chart glyphs to show the relative proportions of different categories. Like geom_pie_glyph, the pie-chart glyphs are independent of the axes, with the additional feature of being interactive and can be hovered over to show information about the raw counts of the different categories. The interactivity is added using the ggiraph framework and all features offered by ggiraph are supported.

Usage

```
geom_pie_interactive(...)
```

Arguments

`...`	arguments passed to geom_pie_glyph, in addition to all interactive_parameters offered by ggiraph.

Value

A ggplot layer with interactive parameters for creating ggiraph plots.

See Also

girafe(), geom_pie_glyph()

Examples

```
#' ## Load libraries
library(dplyr)
library(tidyr)
library(ggplot2)
library(ggiraph)

## Simulate raw data
set.seed(123)
plot_data <- data.frame(response = rnorm(10, 100, 30),
                        system = as.factor(1:10),
                        group = sample(size = 10,
                                       x = c("G1", "G2", "G3"),
                                       replace = TRUE),
                        A = round(runif(10, 3, 9), 2),
                        B = round(runif(10, 1, 5), 2),
                        C = round(runif(10, 3, 7), 2),
                        D = round(runif(10, 1, 9), 2))

head(plot_data)

# One of the interactive aesthetics is tooltip. It is set that by default
# it shows the value and percentage of each slice in the pie-chart.
# Hover over any pie-chart in the plot to see this
plot_obj1 <- ggplot(data = plot_data, aes(x = system, y = response)) +
               geom_pie_interactive(slices = c("A", "B", "C", "D"),
                                    data = plot_data)+
               theme_classic()
x1 <- girafe(ggobj = plot_obj1)
if(interactive()) print(x1)

# The user can also set their own custom tooltip which could either by
# a column in the data or a custom string
plot_obj2 <- ggplot(data = plot_data, aes(x = system, y = response)) +
               # Setting the group as a tooltip
               geom_pie_interactive(slices = c("A", "B", "C", "D"),
                                    data = plot_data,
                                    aes(tooltip = paste0("Group: ", group)))+
               theme_classic()
x2 <- girafe(ggobj = plot_obj2)
if(interactive()) print(x2)

# It is also possible to add an identifier to highlight all elements within
# a group when one element of a group is hovered over by using data_id
plot_obj3 <- ggplot(data = plot_data, aes(x = system, y = response)) +
               # Setting the group as a tooltip
               geom_pie_interactive(slices = c("A", "B", "C", "D"),
                                    data = plot_data, colour = "black",
                                    aes(data_id = group))+
               theme_classic()
x3 <- girafe(ggobj = plot_obj3)
# Hover over one pie-glyph to highlight all observations within the same group
```

```
if(interactive()) print(x3)

# All other aesthetics and attributes of geom_pie_glyph can be used as well
```

---

pieGrob *Create pie-chart glyph*

Description

This function creates a pie-chart glyph. The proportions of the different slices are calculated automatically using the numbers in the values parameter.

Usage

```
pieGrob(
  x = 0.5,
  y = 0.5,
  values,
  radius = 1,
  radius_unit = "cm",
  edges = 360,
  col = "black",
  fill = NA,
  lwd = 1,
  lty = 1,
  alpha = 1,
  default.units = "npc"
)
```

Arguments

x	A number or unit object specifying x-location of pie chart.
y	A number or unit object specifying y-location of pie chart.
values	A numeric vector specifying the values of the different slices of the pie chart.
radius	A number specifying the radius of the pie-chart.
radius_unit	Character string specifying the unit for the radius of the pie-chart.
edges	Number of edges which make up the circumference of the pie-chart (Increase for higher resolution).
col	Character specifying the colour of the border between the pie slices.
fill	A character vector specifying the colour of the individual slices.
lwd	Line width of the pie borders.
lty	Linetype of the pie borders.
alpha	Number between 0 and 1 specifying the opacity of the pie-charts.
default.units	Change the default units for the position and radius of the pie-glyphs.

Value

A grob object

Examples

```
library(grid)
grid.newpage()
p1 <- pieGrob(x = 0.2, y = 0.2,
              values = c(.7, .1, .1, .1), radius = 1,
              fill = c("purple", "red", "green", "orange"))
grid.draw(p1)

## Change unit of radius using `radius_unit` and slice colours using `fill`
## Note `values` don't need to proportions. They can be anything and
## proportions would be calculated
grid.newpage()
p2 <- pieGrob(x = 0.5, y = 0.75,
              values = c(1, 2, 3, 4, 5), radius = 1,
              radius_unit = "in",
              fill = c("purple", "yellow", "green", "orange", "blue"))
grid.draw(p2)

## Change border attributes using `col`, `lwd`, and `lty`
grid.newpage()
p3 <- pieGrob(x = 0.5, y= 0.5,
              values = c(10, 40, 50), radius = 20,
              radius_unit = "mm",
              col = "red", lwd = 5, lty = 3,
              fill = c("purple", "yellow", "blue"))
grid.draw(p3)

## Use `alpha` to change opacity of pies
grid.newpage()
p4 <- pieGrob(x = 0.25, y = 0.75,
              values = c(50), radius = 25,
              radius_unit = "mm", edges = 36000,
              col = "navy", lwd = 4, lty = "33",
              fill = c("purple4"), alpha = 0.5)
grid.draw(p4)

## Use `edges` to increase resolution of pie-charts
grid.newpage()
p5 <- pieGrob(x = 0.8, y = 0.2,
              values = c(.7, .1, .1, .1), radius = 1,
              fill = c("purple", "red", "green", "orange"),
              edges = 10000)
grid.draw(p5)
```

`scale_radius_discrete` *Scales for the pie glyph radius*

Description

`scale_radius_*()` is useful for adjusting the radius of the pie glyphs.

Usage

```
scale_radius_discrete(..., range = c(0.25, 0.6), unit = "cm")

scale_radius_manual(..., values, unit = "cm", breaks = waiver(), na.value = NA)

scale_radius_continuous(..., range = c(0.25, 0.6), unit = "cm")

scale_radius(..., range = c(0.25, 0.6), unit = "cm")
```

Arguments

`...`	Arguments passed on to `continuous_scale` `minor_breaks` One of: • `NULL` for no minor breaks • `waiver()` for the default breaks (none for discrete, one minor break between each major break for continuous) • A numeric vector of positions • A function that given the limits returns a vector of minor breaks. Also accepts rlang lambda function notation. When the function has two arguments, it will be given the limits and major break positions. `oob` One of: • Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang lambda function notation. • The default (`scales::censor()`) replaces out of bounds values with `NA`. • `scales::squish()` for squishing out of bounds values into range. • `scales::squish_infinite()` for squishing infinite values into range. `na.value` Missing values will be replaced with this value. `call` The `call` used to construct the scale for reporting messages. `super` The super class to use for the constructed scale
`range`	a numeric vector of length 2 that specifies the minimum and maximum size of the plotting symbol after transformation.
`unit`	Unit for the radius of the pie glyphs. Default is "cm", but other units like "in", "mm", etc. can be used.
`values`	a set of aesthetic values to map data values to. The values will be matched in order (usually alphabetical) with the limits of the scale, or with `breaks` if provided. If this is a named vector, then the values will be matched based on the names instead. Data values that don't match will be given `na.value`.
`breaks`	One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object

- A numeric vector of positions
- A function that takes the limits as input and returns breaks as output (e.g., a function returned by `scales::extended_breaks()`). Note that for position scales, limits are provided after scale expansion. Also accepts rlang lambda function notation.

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

Value

A ggplot scale object adjusting the radii of the pie glyphs

Examples

```
## Load libraries
library(dplyr)
library(tidyr)
library(ggplot2)

## Simulate raw data
set.seed(789)
plot_data <- data.frame(y = rnorm(10, 100, 30),
                        x = 1:10,
                        group = sample(size = 10,
                                       x = c(1, 2, 3),
                                       replace = TRUE),
                        A = round(runif(10, 3, 9), 2),
                        B = round(runif(10, 1, 5), 2),
                        C = round(runif(10, 3, 7), 2),
                        D = round(runif(10, 1, 9), 2))

head(plot_data)

## Create plot
p <- ggplot(data = plot_data)+
    geom_pie_glyph(aes(x = x, y = y, radius = group),
                   slices = c('A', 'B', 'C', 'D'))+
    labs(y = 'Response', x = 'System',
         fill = 'Attributes')+
    theme_classic()

p + scale_radius_continuous(range = c(0.2, 0.5))

q <- ggplot(data = plot_data)+
    geom_pie_glyph(aes(x = x, y = y,
                       radius = as.factor(group)),
                   slices = c('A', 'B', 'C', 'D'))+
    labs(y = 'Response', x = 'System',
         fill = 'Attributes', radius = 'Group')+
    theme_classic()

q + scale_radius_discrete(range = c(0.05, 0.2), unit = 'in',
                          name = 'Group')
```

```
q + scale_radius_manual(values = c(2, 6, 4), unit = 'mm',
                        labels = paste0('G', 1:3), name = 'G')
```

`upgradeUnit.unit.list` *Upgrading units for a list*

Description

Upgrading units for a list

Usage

```
## S3 method for class 'unit.list'
upgradeUnit(x)
```

Arguments

`x`	A `unit.list` object.

Value

A combined unit object.

Index