Package ‘PieGlyph’
February 9, 2023

Title  Axis Invariant Scatter Pie Plots
Version 0.1.0
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, forcats, pplyr, grid, scales, cli, utils
Suggests ranger, maps, mapproj, knitr, tidyverse, rmarkdown
RoxygenNote 7.2.3
VignetteBuilder knitr

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

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

NeedsCompilation no

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

Maintainer Rishabh Vishwakarma <vishwakr@tcd.ie>
Repository CRAN

Date/Publication 2023-02-09 13:50:09 UTC

R topics documented:
draw_key_pie ......................................................... 2
gem_pie_glyph ...................................................... 2
pieGrob ............................................................... 6
scale_radius_discrete ............................................. 7

Index 10
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**

```r
gem_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.

- **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.
If all slices for an observation are NA, the observation is dropped while if at least one slice is not NA, the other slices 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, observations are silently removed/modified to 0, without notifying the user.

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.

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.

A ggplot layer

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)

## Load libraries
library(tidyverse)
library(PieGlyph)

## Simulate raw data
set.seed(123)
plot_data <- data.frame(response = rnorm(10, 100, 30),
    system = 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),

    x = c('G1', 'G2', 'G3'),
    y = c(10, 20, 30))

head(plot_data)
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 and border 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()

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

## 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()

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

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

Value
A grob object

Examples

```r
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
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
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)
```

scale_radius_discrete  Scales for the pie glyph radius

Description

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

Usage

```r
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 (one minor break between each major break)
- A numeric vector of positions
- A function that given the limits returns a vector of minor breaks. Also accepts rlang `lambda` function notation.

**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.

**expand** For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the `expand` argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

**position** For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.

**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()`). 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**

```r
## Load libraries
library(tidyverse)
library(PieGlyph)

## 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")
```
Index

aes(), 3
aes_(), 3

continuous_scale, 8
draw_key, 2
draw_key_pie, 2

expansion(), 8
fortify(), 3

geom_pie_glyph, 2
ggplot(), 3
grouping, 4

lambda, 8

pieGrob, 6
pivot_longer(), 3

scale_radius(scale_radius_discrete), 7
scale_radius_continuous
(scale_radius_discrete), 7
scale_radius_discrete, 7
scale_radius_manual
(scale_radius_discrete), 7
scales::censor(), 8
scales::extended_breaks(), 8
scales::squish(), 8
scales::squish_infinite(), 8

transformation object, 8