

Package ‘ggTimeSeries’

January 23, 2022

Type Package

Title Time Series Visualisations Using the Grammar of Graphics

Version 1.0.2

Date 2022-01-17

Maintainer Aditya Kothari <mail.thecomeonman@gmail.com>

Description Provides additional display mediums for time series visualisations.

URL <https://github.com/thecomeonman/ggTimeSeries>

BugReports <https://github.com/thecomeonman/ggTimeSeries/issues>

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.4.0), ggplot2

Imports data.table, stats, utils

RoxygenNote 6.1.0

Suggests knitr, rmarkdown, ggthemes, prettydoc

VignetteBuilder knitr, rmarkdown

NeedsCompilation no

Author Aditya Kothari [cre, aut],
Ather Energy [cph],
Jesse Vent [ctb]

Repository CRAN

Date/Publication 2022-01-23 16:22:42 UTC

R topics documented:

dtClimate	2
ggplot_calendar_heatmap	3
ggplot_horizon	4
ggplot_waterfall	6

Marimekkofy	7
StatCalendarMonthBreaks	7
StatHorizon	8
StatMarimekko	8
StatOccurrence	8
StatSteamgraph	9
StatWaterfall	9
stat_calendar_heatmap	9
stat_horizon	11
stat_marimekko	12
stat_occurrence	13
stat_steamgraph	15
stat_waterfall	16
Index	18

dtClimate

Climate data.

Description

The climate data used in the blogpost.

Usage

```
data(dtClimate)
```

Format

An object of class `data.table` (inherits from `data.frame`) with 23628 rows and 5 columns.

Source

<http://doi.org/10.7289/V5D21VHZ> Downloaded from ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/by_year/

Menne, M.J., I. Durre, B. Korzeniewski, S. McNeal, K. Thomas, X. Yin, S. Anthony, R. Ray, R.S. Vose, B.E. Gleason, and T.G. Houston, 2012: Global Historical Climatology Network - Daily, Version 3.12

`ggplot_calendar_heatmap`*Plots a calendar heatmap*

Description

A calendar heatmap provides context for weeks, and day of week which makes it a better way to visualise daily data than line charts. Largely uses Codoremifa's code from stackoverflow.com/questions/22815688/calendar-time-series-with-r.

Usage

```
ggplot_calendar_heatmap(dtDateValue, cDateColumnName = "",
  cValueColumnName = "", vcGroupingColumnNames = "Year",
  dayBorderSize = 0.25, dayBorderColour = "black",
  monthBorderSize = 2, monthBorderColour = "black",
  monthBorderLineEnd = "round")
```

Arguments

<code>dtDateValue</code>	Data set which may include other columns apart from date and values.
<code>cDateColumnName</code>	Column name of the dates.
<code>cValueColumnName</code>	Column name of the data.
<code>vcGroupingColumnNames</code>	The set of columns which together define the group for the chart to operate within. If you plan to facet your plot, you should specify the same column names to this argument. The function will automatically add the variable for the year to the facet.
<code>dayBorderSize</code>	Size of the border around each day
<code>dayBorderColour</code>	Colour of the border around each day
<code>monthBorderSize</code>	Size of the border around each month
<code>monthBorderColour</code>	Colour of the border around each month
<code>monthBorderLineEnd</code>	Line end for the border around each month

Value

Returns a ggplot friendly object which means the user can use ggplot scales to modify the look, add more geoms, etc.

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the output object: `+ xlab(NULL) + ylab(NULL) + scale_fill_continuous(low = 'green', high = 'red') + theme(axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), # strip.text = element_blank(), # useful if only one year of data plot.background = element_blank(), panel.border = element_blank(), panel.background = element_blank(), panel.grid = element_blank(), panel.border = element_blank())`

Also See

[stat_calendar_heatmap](#), a flexible but less polished alternative.

Examples

```
{
library(data.table)
library(ggplot2)
set.seed(1)
dtData = data.table(
  DateCol = seq(
    as.Date("1/01/2014", "%d/%m/%Y"),
    as.Date("31/12/2015", "%d/%m/%Y"),
    "days"
  ),
  ValueCol = runif(730)
)
# you could also try categorical data with
# ValueCol = sample(c('a','b','c'), 730, replace = T)
p1 = ggplot_calendar_heatmap(
  dtData,
  'DateCol',
  'ValueCol'
)
p1
# add new geoms
p1 +
geom_text(label = '!!!') +
scale_colour_continuous(low = 'red', high = 'green')
}
```

Description

A horizon plot breaks the Y dimension down using colours. This is useful when visualising y values spanning a vast range and / or trying to highlight outliers without losing context of the rest of the data. Horizon plots are best viewed in an aspect ratio of very low vertical length.

Usage

```
ggplot_horizon(dtData, cXColumnName, cYColumnName, bandwidth = NULL,
              vcGroupingColumnNames = NULL)
```

Arguments

dtData Data set which may include other columns apart from date and values.

cXColumnName Column name of dates.

cYColumnName Column name of values.

bandwidth the width of one band of Y values. easier to differentiate between the bands.

vcGroupingColumnNames The set of columns which together define the group for the chart to operate within If you plan to facet your plot, you should specify the same column names to this argument.

Value

Returns a ggplot friendly object which means the user can use ggplot scales, etc. to modify the look.

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the example output object: `+ xlab(NULL) + ylab(NULL) + scale_fill_continuous(low = 'green', high = 'red') + theme(axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), # strip.text = element_blank(), # useful if only one year of data plot.background = element_blank(), panel.border = element_blank(), panel.background = element_blank(), panel.grid = element_blank(), panel.border = element_blank()) + coord_fixed(0.5 * diff(range(dfData$x)) / diff(range(dfData$y)))`

Also See

[stat_horizon](#), a less polished but more flexible alternative.

Examples

```
{
  library(ggplot2)
  set.seed(1)
  dfData = data.frame(x = 1:1000, y = cumsum(rnorm(1000)))
  p1 = ggplot_horizon(dfData, 'x', 'y')
  p1
  # add new geoms or colours
  p1 +
  geom_text(label = '!!!!') +
  scale_colour_continuous(low = 'red', high = 'green')
}
```

ggplot_waterfall *Plots a water fall plot*

Description

A waterfall plot highlights the change in the time series rather than the value of the time series itself.

Usage

```
ggplot_waterfall(dtData, cXColumnName, cYColumnName, nArrowSize = 0.25,  
  vcGroupingColumnNames = NULL)
```

Arguments

dtData	Data set which may include other columns apart from the columns mapped to x and y .
cXColumnName	Column name of the x mapping.
cYColumnName	Column name of the y mapping.
nArrowSize	the size of the arrow head on the plot in cm
vcGroupingColumnNames	The set of columns which together define the group for the chart to operate between. If you plan to facet your plot, you should specify the same column names to this argument.

Value

Returns a ggplot friendly object which means the user can use ggplot scales to modify the look, add more geoms, etc.

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the output object: `+ xlab(NULL) + ylab(NULL) + scale_fill_continuous(low = 'green', high = 'red') + theme(axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), # strip.text = element_blank(), # useful if only one year of data plot.background = element_blank(), panel.background = element_blank(), panel.border = element_blank(), panel.grid = element_blank(), panel.border = element_blank())`

Also See

[stat_waterfall](#), a flexible but less polished alternative.

Examples

```
{
  library(ggplot2)
  set.seed(1)
  dfData = data.frame(x = 1:100, y = cumsum(rnorm(100)))
  ggplot_waterfall(
    dtData = dfData,
    'x',
    'y'
  )}
```

Marimekkofy

Marimekkofy

Description

Marimekkofy

Usage

```
Marimekkofy(data, xbucket = "xbucket", ybucket = "ybucket",
  weight = NULL)
```

Arguments

data	dataframe
xbucket	x value
ybucket	y value
weight	weight value

Value

df

StatCalendarMonthBreaks

Transforms data for the month breaks of the calendar heatmap

Description

Transforms data for the month breaks of the calendar heatmap

Usage

```
StatCalendarMonthBreaks
```

Format

An object of class StatCalendarMonthBreaks (inherits from Stat, ggproto, gg) of length 3.

StatHorizon	<i>Transforms data for a horizon plot</i>
-------------	---

Description

Transforms data for a horizon plot

Usage

StatHorizon

Format

An object of class StatHorizon (inherits from Stat, ggproto, gg) of length 4.

StatMarimekko	<i>Transforms data for the tiles of the heatmap</i>
---------------	---

Description

Transforms data for the tiles of the heatmap

Usage

StatMarimekko

Format

An object of class StatMarimekko (inherits from Stat, ggproto, gg) of length 4.

StatOccurrence	<i>Transforms data for an occurrence plot</i>
----------------	---

Description

Transforms data for an occurrence plot

Usage

StatOccurrence

Format

An object of class StatOccurrence (inherits from Stat, ggproto, gg) of length 3.

StatSteamgraph	<i>Transforms data for a steam graph</i>
----------------	--

Description

Transforms data for a steam graph

Usage

```
StatSteamgraph
```

Format

An object of class StatSteamgraph (inherits from Stat, ggproto, gg) of length 4.

StatWaterfall	<i>Transforms data for a horizon plot</i>
---------------	---

Description

Transforms data for a horizon plot

Usage

```
StatWaterfall
```

Format

An object of class StatWaterfall (inherits from Stat, ggproto, gg) of length 3.

stat_calendar_heatmap	<i>Plots a calendar heatmap</i>
-----------------------	---------------------------------

Description

A calendar heatmap provides context for weeks, and day of week and is a better way to visualise daily data than line charts.

Usage

```
stat_calendar_heatmap(mapping = NULL, data = NULL, show.legend = NA,  
  inherit.aes = TRUE, na.rm = T, bandwidth = NULL, ...)
```

Arguments

mapping	mapping
data	df
show.legend	logical
inherit.aes	logical
na.rm	logical
bandwidth	bandwidth
...	more functions

Aesthetics

date, fill.

Data Tips

[strptime](#) can help extract the value of the year, week of year, and day of week from the date column. You might need to extract the year to facet multiple years as demonstrated in the example. This stat uses the following transformation to obtain the x and y coordinate to be used in the heatmap -
`data$x = 1 + as.integer(strptime(data$date, "%W"))` `data$y = as.integer(strptime(data$date, "%w"))`
`data$y[data$y == 0L] = 7` `data$y = 8 - data$y`

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the output object: `+ xlab(NULL) + ylab(NULL) + scale_fill_continuous(low = 'green', high = 'red') + theme(axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), panel.background = element_blank(), panel.border = element_blank())`

Also See

[ggplot_calendar_heatmap](#), a polished but less flexible alternative.

Examples

```
{
library(ggplot2)
DailyData = data.frame(
  DateCol = seq(
    as.Date("1/01/2014", "%d/%m/%Y"),
    as.Date("31/12/2015", "%d/%m/%Y"),
    "days"
  ),
  ValueCol = runif(730)
)
DailyData$Year = strptime(DailyData$DateCol, "%Y")
ggplot(
  DailyData,
```

```

aes(
  date = DateCol,
  fill = ValueCol
)
) +
  stat_calendar_heatmap() +
  facet_wrap(~Year, ncol = 1)}

```

stat_horizon

Plot a time series as a horizon plot

Description

Plot a time series as a horizon plot

Usage

```

stat_horizon(mapping = NULL, data = NULL, show.legend = NA,
  inherit.aes = TRUE, na.rm = T, bandwidth = NULL, ...)

```

Arguments

mapping	mapping
data	dataframe
show.legend	logical
inherit.aes	logical
na.rm	logical
bandwidth	bandwith
...	other functions

A horizon plot breaks the Y dimension down using colours. This is useful when visualising y values spanning a vast range and / or trying to highlight outliers without losing context of the rest of the data. Horizon plots are best viewed in an aspect ratio of very low vertical length.

Aesthetics

x, y, fill. Fill argument is overridden internally but is required for ggplot to assign a colour / fill scale.

Other parameters

bandwidth, to dictate the span of a band.

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the output object: `+ xlab(NULL) + ylab(NULL) + scale_fill_continuous(low = 'green', high = 'red') + theme(axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), # strip.text = element_blank(), # useful if only one year of data plot.background = element_blank(), panel.border = element_blank(), panel.background = element_blank(), panel.grid = element_blank(), panel.border = element_blank())`

Also See

[ggplot_horizon](#), a more polished but less flexible alternative.

Examples

```
{
  library(ggplot2)
  ggplot(data.frame(x = 1:89, y = as.numeric(unlist(austres))), aes(x = x, y = y, fill = y)) +
    stat_horizon() +
    scale_fill_continuous(low = 'white', high = 'red')

  set.seed(10)
  ggplot(data.frame(x = 1:1000, y = cumsum(rnorm(1000))), aes(x = x, y = y, fill = y)) +
    stat_horizon() +
    scale_fill_continuous(low = 'white', high = 'red')}
```

 stat_marimekko

Plot two categorical variables as marimekko

Description

Plot two categorical variables as marimekko

Usage

```
stat_marimekko(mapping = NULL, data = NULL, show.legend = NA,
  inherit.aes = TRUE, na.rm = T, xlabeyposition = NULL, ...)
```

Arguments

mapping	mapping
data	data
show.legend	logical
inherit.aes	logical
na.rm	logical
xlabeyposition	position

... other functions A marimekko plot, or a mosaic plot, visualises the co-occurrence of two categorical / ordinal variables. In a time series, it could be used to visualise the transitions from one state to another by considering each state to be a category and plotting current category vs. next category.

Aesthetics

xbucket, ybucket, fill. Fill argument needs to be assigned to ybucket., or some other column which is a one to one mapping of ybucket.

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the output object: `+ xlab(NULL) + ylab(NULL) + scale_fill_continuous(low = 'green', high = 'red') + theme(axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), # strip.text = element_blank(), # useful if only one year of data plot.background = element_blank(), panel.border = element_blank(), panel.background = element_blank(), panel.grid = element_blank(), panel.border = element_blank())`

Examples

```
{
library(ggplot2)
ggplot(
  data.frame(
    x1 = round(3 * runif(10000), 0),
    y1 = pmax(pmin(round(3 * rnorm(10000), 0), 3), -3),
    weight = 1:10000
  )
) +
  stat_marimekko(
    aes(
      xbucket = x1,
      ybucket = y1,
      fill = factor(y1),
      weight = weight
    ),
    xlabeleyposition = 1.1,
    color = 'black'
  )
})
```

stat_occurrence

Plots a time series as a dot plot

Description

Plots a time series as a dot plot

Usage

```
stat_occurrence(mapping = NULL, data = NULL, show.legend = NA,
  inherit.aes = TRUE, na.rm = T, bandwidth = NULL, ...)
```

Arguments

mapping	mapping
data	df
show.legend	logical
inherit.aes	logical
na.rm	logical
bandwidth	bandwidth
...	more functions

For rare events, it's convenient to have the count of events encoded in the chart itself. A bar chart requires the user to perceive the y axis which this does not.

Aesthetics

x, y

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the output object: `+ xlab(NULL) + ylab(NULL) + scale_fill_continuous(low = 'green', high = 'red') + theme(axis.text.y = element_blank(), axis.ticks.y = element_blank(), legend.position = 'none', strip.background = element_blank(), panel.background = element_blank(), panel.border = element_blank(), panel.grid = element_blank(), panel.border = element_blank()) + coord_fixed(ylim = c(0, 1 + max(dfData$y)))` `coord_fixed` can provide a balance to the aspect ratio of the chart.

Examples

```
{
  library(data.table)
  library(ggplot2)
  set.seed(1)
  dfData = data.table(x = 1:100, y = floor(4 * abs(rnorm(100, 0, 0.4))))
  ggplot(dfData, aes(x = x, y = y)) +
    stat_occurrence() +
    coord_fixed()
}
```

stat_steamgraph	<i>Plot multiple time series as a steamgraph</i>
-----------------	--

Description

Plot multiple time series as a steamgraph

Usage

```
stat_steamgraph(mapping = NULL, data = NULL, show.legend = NA,
  inherit.aes = TRUE, na.rm = T, ...)
```

Arguments

mapping	mapping
data	data
show.legend	logical
inherit.aes	logical
na.rm	logical
...	other functions

Plots [geom_ribbon](#) for each time series and stacks them one on top of the other. It's a more aesthetically appealing version of a stacked area chart. The groups with the most variance are placed on the outside, and the groups with the least variance are placed on the inside.

Aesthetics

geom_steamgraph needs x, y, group, fill.

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the example output object: `+ xlab(NULL) + ylab(NULL) + theme(axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), # strip.text = element_blank(), # useful if only one year of data plot.background = element_blank(), panel.background = element_blank(), panel.border = element_blank(), panel.grid = element_blank(), panel.border = element_blank()) + coord_fixed(0.2 * diff(range(df$Time)) / diff(range(df$Signal)))`

Examples

```
{
  library(ggplot2)
  set.seed(10)
  df = data.frame(
    Time=1:1000,
    Signal=abs(c(cumsum(rnorm(1000, 0, 3))),
```

```

      cumsum(rnorm(1000, 0, 4)), cumsum(rnorm(1000, 0, 1)),
      cumsum(rnorm(1000, 0, 2))))),
  Variable = c(rep('a', 1000), rep('b', 1000), rep('c',
    1000), rep('d', 1000)),
  VariableLabel = c(rep('Class A', 1000), rep('Class B',
    1000), rep('Class C', 1000), rep('Class D', 1000))
)

ggplot(df, aes(x = Time, y = Signal, group = Variable, fill = VariableLabel)) +
  stat_steamgraph()}

```

stat_waterfall

Plot a time series as a waterfall plot

Description

Plot a time series as a waterfall plot

Usage

```

stat_waterfall(mapping = NULL, data = NULL, show.legend = NA,
  inherit.aes = TRUE, na.rm = T, bandwidth = NULL, ...)

```

Arguments

mapping	mapping
data	data
show.legend	legend
inherit.aes	logical
na.rm	logical
bandwidth	bandwidth
...	more functions

A waterfall plot highlights the change in the time series rather than the value of the time series itself.

Aesthetics

stat_waterfall needs x, y

Cosmetic Tips

The minimalist look can be achieved by appending the following chunk of code to the output object:

```

+ xlab(NULL) + ylab(NULL) + theme( axis.text = element_blank(), axis.ticks = element_blank(), legend.position = 'none', strip.background = element_blank(), plot.background = element_blank(), panel.background = element_blank(), panel.border = element_blank(), panel.grid = element_blank(), panel.border = element_blank() )

```


Also See

[ggplot_waterfall](#), a flexible but less polished alternative.

Examples

```
{
library(ggplot2)
set.seed(1)
dfData = data.frame(x = 1:20, y = cumsum(rnorm(20)))
ggplot(dfData, aes(x =x, y = y) )+
  stat_waterfall()}
```

Index

* datasets

- dtClimate, [2](#)
- StatCalendarMonthBreaks, [7](#)
- StatHorizon, [8](#)
- StatMarimekko, [8](#)
- StatOccurrence, [8](#)
- StatSteamgraph, [9](#)
- StatWaterfall, [9](#)

coord_fixed, [14](#)

dtClimate, [2](#)

geom_ribbon, [15](#)

ggplot_calendar_heatmap, [3](#), [10](#)

ggplot_horizon, [4](#), [12](#)

ggplot_waterfall, [6](#), [17](#)

Marimekkofy, [7](#)

stat_calendar_heatmap, [4](#), [9](#)

stat_horizon, [5](#), [11](#)

stat_marimekko, [12](#)

stat_occurrence, [13](#)

stat_steamgraph, [15](#)

stat_waterfall, [6](#), [16](#)

StatCalendarMonthBreaks, [7](#)

StatHorizon, [8](#)

StatMarimekko, [8](#)

StatOccurrence, [8](#)

StatSteamgraph, [9](#)

StatWaterfall, [9](#)

strftime, [10](#)