Package ‘sugrrants’

March 12, 2024

Title Supporting Graphs for Analysing Time Series

Version 0.2.9

Description Provides 'ggplot2' graphics for analysing time series data. It aims to fit into the 'tidyverse' and grammar of graphics framework for handling temporal data.

License GPL (>= 3)

URL https://pkg.earo.me/sugrrants/

BugReports https://github.com/earowang/sugrrants/issues

Depends ggplot2 (>= 2.2.0), R (>= 3.1.3)

Imports dplyr (>= 0.8.0), grid, gtable, lubridate (>= 1.7.1), rlang (>= 0.2.0)

Suggests covr, knitr, plotly, readr, rmarkdown, testthat, tidyr, tsibble (>= 0.8.0), viridis

VignetteBuilder knitr

Encoding UTF-8

Language en-GB

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Earo Wang [aut, cre] (<https://orcid.org/0000-0001-6448-5260>), Di Cook [aut, ths] (<https://orcid.org/0000-0002-3813-7155>), Rob Hyndman [aut, ths] (<https://orcid.org/0000-0002-2140-5352>)

Maintainer Earo Wang <earo.wang@gmail.com>

Repository CRAN

Date/Publication 2024-03-12 05:20:03 UTC

R topics documented:

sugrrants-package 2
facet_calendar 2
frame_calendar 4
geom_acf 7
hourly_peds 8
stat_acf 9

Index **11**

`sugrrants-package` *sugrrants: supporting graphs for analysing time series*

Description

Provides 'ggplot2' graphics for analysing time series data. It aims to fit into the 'tidyverse' and grammar of graphics framework for handling temporal data.

Author(s)

Maintainer: Earo Wang `<earo.wang@gmail.com>` (ORCID)

Authors:

- Di Cook (ORCID) [thesis advisor]
- Rob Hyndman (ORCID) [thesis advisor]

See Also

Useful links:

- `https://pkg.earo.me/sugrrants/`
- Report bugs at `https://github.com/earowang/sugrrants/issues`

`facet_calendar` *Lay out panels in a calendar format*

Description

Lay out panels in a calendar format

Usage

```
facet_calendar(
  date,
  format = "%b %d",
  week_start = getOption("lubridate.week.start", 1),
  nrow = NULL,
  ncol = NULL,
  scales = "fixed",
  shrink = TRUE,
  dir = "h",
  labeller = "label_value",
  strip.position = "top"
)
```

Arguments

`date`	A variable that contains dates or an expression that generates dates will be mapped in the calendar.
`format`	A character string, such as `%Y-%b-%d` and `%a (%d)`, formatting the display of facet strips. See `?strptime` for details.
`week_start`	Day on which week starts following ISO conventions - 1 means Monday (default), 7 means Sunday. You can set `lubridate.week.start` option to control this parameter globally.
`nrow, ncol`	Number of rows and columns defined for "monthly" calendar layout. If `NULL`, it computes a sensible layout.
`scales`	Should scales be fixed (`"fixed"`, the default), free (`"free"`), or free in one dimension (`"free_x"`, `"free_y"`)?
`shrink`	If `TRUE`, will shrink scales to fit output of statistics, not raw data. If `FALSE`, will be range of raw data before statistical summary.
`dir`	Direction of calendar: "h" for horizontal (the default) or "v" for vertical.
`labeller`	A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with `vars(cyl, am)`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. You can use different labeling functions for different kind of labels, for example use `label_parsed()` for formatting facet labels. `label_value()` is used by default, check it for more details and pointers to other options.
`strip.position`	By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top", "bottom", "left", "right")`

Details

A monthly calendar is set up as a 5 by 7 layout matrix. Each month could extend over six weeks but in these months is to wrap the last few days up to the top row of the block.

See Also

frame_calendar for a compact calendar display, by quickly transforming the data.

Examples

```
fs <- hourly_peds %>%
  dplyr::filter(Date < as.Date("2016-05-01"))

fs %>%
  ggplot(aes(x = Time, y = Hourly_Counts)) +
  geom_line(aes(colour = Sensor_Name)) +
  facet_calendar(~ Date, nrow = 2) + # or ~ as.Date(Date_Time)
  theme(legend.position = "bottom")
```

`frame_calendar` *Rearrange a temporal data frame to a calendar-based data format using linear algebra*

Description

Temporal data of daily intervals or higher frequency levels can be organised into a calendar-based format, which is useful for visually presenting calendar-related activities or multiple seasonality (such as time of day, day of week, day of month). The function only returns a rearranged data frame, and ggplot2 takes care of the plotting afterwards. It allows more flexibility for users to visualise the data in various ways.

Usage

```
frame_calendar(
  data,
  x,
  y,
  date,
  calendar = "monthly",
  dir = "h",
  week_start = getOption("lubridate.week.start", 1),
  nrow = NULL,
  ncol = NULL,
  polar = FALSE,
  scale = "fixed",
  width = 0.95,
  height = 0.95,
  margin = NULL,
  ...
)

prettify(plot, label = c("label", "text"), locale, abbr = TRUE, ...)
```

Arguments

`data`	A data frame or a grouped data frame including a `Date` variable.
`x`	A bare (or unquoted) variable mapping to x axis, for example time of day. If integer 1 is specified, it simply returns calendar grids on x without transformation.
`y`	A bare (or unquoted) variable or more mapping to y axis. More than one variable need putting to `vars()`. If integer 1 is specified, it returns calendar grids on y without transformation.
`date`	A `Date` variable mapping to dates in the calendar.
`calendar`	Type of calendar. (1) "monthly" calendar (the default) organises the `data` to a common format comprised of day of week in the column and week of month in the row. A monthly calendar is set up as a 5 by 7 layout matrix. Each month could extend over six weeks but in these months is to wrap the last few days up to the top row of the block. (2) "weekly" calendar consists of day of week and week of year. (3) "daily" calendar refers to day of month and month of year.
`dir`	Direction of calendar: "h" for horizontal (the default) or "v" for vertical.
`week_start`	Day on which week starts following ISO conventions - 1 means Monday (default), 7 means Sunday. You can set `lubridate.week.start` option to control this parameter globally.
`nrow, ncol`	Number of rows and columns defined for "monthly" calendar layout. If `NULL`, it computes a sensible layout.
`polar`	FALSE (the default) for Cartesian or TRUE for polar coordinates.
`scale`	"fixed" (the default) for fixed scale. "free" for scaling conditional on each daily cell, "free_wday" for scaling on weekdays, "free_mday" for scaling on day of month.
`width, height`	Numerics between 0 and 1 to specify the width/height for each glyph.
`margin`	Numerics of length two between 0 and 1 to specify the horizontal and vertical margins between month panels.
`...`	Extra arguments passed to `geom_label()` and `geom_text()`
`plot`	A "ggplot" object or "plotly".
`label`	If "label" is specified, it will add month/week text on the `ggplot` object, which is actually passed to `geom_label()`. If "text" is specified, it will add weekday/day of month text on the `ggplot` object, which is actually passed to `geom_text()`. By default, both "label" and "text" are used. If "text2" is specified for the "monthly" calendar only, it will add day of month to the `ggplot` object.
`locale`	ISO 639 language code. The default is "en" (i.e. US English). For other languages support, package **readr** needs to be installed. See readr::locale for more details.
`abbr`	Logical to specify if the abbreviated version of label should be used.

Details

The calendar-based graphic can be considered as small multiples of sub-series arranged into many daily cells. For every multiple (or facet), it requires the `x` variable mapped to be time of day and y to value. New x and y are computed and named with a . prefixed to variable according to x and y respectively, and get ready for ggplot2 aesthetic mappings. In conjunction with group_by(), it allows the grouped variable to have their individual scales. For more details, see vignette("frame-calendar", package = "sugrrants")

Value

A data frame or a dplyr::tibble with newly added columns of .x, .y. .x and .y together give new coordinates computed for different types of calendars. date groups the same dates in a chronological order, which is useful for geom_line or geom_path. The basic use is ggplot(aes(x = .x, y = .y, group = date)) + geom_
The variable names .x and .y reflect the actual x and y with a prefix ..

See Also

facet_calendar for a fully-fledged faceting calendar with formal labels and axes.

Examples

```
library(dplyr, warn.conflicts = FALSE)
# compute the calendar layout for the data frame
calendar_df <- hourly_peds %>%
  filter(Sensor_ID == 13, Year == 2016) %>%
  frame_calendar(x = Time, y = Hourly_Counts, date = Date, nrow = 4)

# ggplot
p1 <- calendar_df %>%
  ggplot(aes(x = .Time, y = .Hourly_Counts, group = Date)) +
  geom_line()
prettify(p1, size = 3, label.padding = unit(0.15, "lines"))

# use in conjunction with group_by()
grped_calendar <- hourly_peds %>%
  filter(Year == "2017", Month == "March") %>%
  group_by(Sensor_Name) %>%
  frame_calendar(x = Time, y = Hourly_Counts, date = Date, week_start = 7)

p2 <- grped_calendar %>%
  ggplot(aes(x = .Time, y = .Hourly_Counts, group = Date)) +
  geom_line() +
  facet_wrap(~ Sensor_Name, nrow = 2)
prettify(p2)
## Not run:
# allow for different languages
# below gives simplied Chinese labels with STKaiti font family,
# assuming this font installed in user's local system
prettify(p2, locale = "zh", family = "STKaiti")

# plotly example
if (!requireNamespace("plotly", quietly = TRUE)) {
  stop("Please install the 'plotly' package to run these following examples.")
}
library(plotly)
```

```
pp <- calendar_df %>%
  group_by(Date) %>%
  plot_ly(x = ~ .Time, y = ~ .Hourly_Counts) %>%
  add_lines(text = ~ paste("Count: ", Hourly_Counts, "<br> Time: ", Time))
prettify(pp)

## End(Not run)
```

geom_acf — *Autocorrelation for temporal data*

Description

Since the data input is `data.frame`, it's better to sort the date-times from early to recent and make implicit missing values explicit before using `geom_acf`.

Usage

```
geom_acf(
  mapping = NULL,
  data = NULL,
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  lag.max = NULL,
  type = "correlation",
  level = 0.95,
  ...
)
```

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)`).

`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`	Logical. If `TRUE`, missing values are removed. default is the "correlation" and other options are "covariance" and "partial".
`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.
`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()`.
`lag.max`	An integer indicating the maximum lag at which to calculate the acf.
`type`	A character string giving the type of the acf to be computed. The
`level`	A numeric defining the confidence level. If `NULL`, no significant line to be drawn.
`...`	Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Examples

```
library(dplyr)
fstaff <- hourly_peds %>%
  filter(Sensor_ID == 13)

# use ggplot2
fstaff %>%
  ggplot(aes(x = ..lag.., y = Hourly_Counts)) +
  geom_acf()
```

`hourly_peds` *Pedestrian counts in Melbourne city*

Description

A dataset containing the pedestrian counts at hourly intervals from 2016-01-01 to 2017-04-20 at 7 sensors in the city of Melbourne. The variables are as follows:

Usage

```
hourly_peds
```

Format

A tibble with 78755 rows and 9 variables:

Date_Time Date time when the pedestrian counts are recorded

Year Year associated with Date_Time

Month Month associated with Date_Time

Mdate Day of month associated with Date_Time

Day Weekday associated with Date_Time

Time Hour associated with Date_Time

Sensor_ID Sensor identifiers

Sensor_Name Sensor names

Hourly_Counts Hourly pedestrian counts

Examples

```
hourly_peds
```

`stat_acf` *Autocorrelation for temporal data*

Description

Since the data input is `data.frame`, it's better to sort the date-times from early to recent and make implicit missing values explicit before using `stat_acf`.

Usage

```
stat_acf(
  mapping = NULL,
  data = NULL,
  geom = "bar",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  lag.max = NULL,
  type = "correlation",
  level = 0.95,
  ...
)
```

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, either as a `ggproto` `Geom` subclass or as a string naming the geom stripped of the `geom_` prefix (e.g. `"point"` rather than `"geom_point"`)
`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`	Logical. If `TRUE`, missing values are 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.
`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()`.
`lag.max`	An integer indicating the maximum lag at which to calculate the acf.
`type`	A character string giving the type of the acf to be computed. The default is the "correlation" and other options are "covariance" and "partial".
`level`	A numeric defining the confidence level. If `NULL`, no significant line to be drawn.
`...`	Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

Examples

```
library(dplyr)
fstaff <- hourly_peds %>%
  filter(Sensor_ID == 13)

# use ggplot2
fstaff %>%
  ggplot(aes(x = ..lag.., y = Hourly_Counts)) +
  stat_acf(geom = "bar")
```

Index

∗ **datasets**
  facet_calendar, 2
  geom_acf, 7
  hourly_peds, 8

aes(), *7*, *10*

borders(), *8*, *10*

facet_calendar, 2, *6*
FacetCalendar *(*facet_calendar*)*, 2
fortify(), *7*, *10*
frame_calendar, *4*, 4

geom_acf, 7
GeomAcf *(*geom_acf*)*, 7
ggplot(), *7*, *10*

hourly_peds, 8

label_parsed(), *3*
label_value(), *3*
labeller(), *3*
layer(), *8*, *10*

prettify *(*frame_calendar*)*, 4

readr::locale, *5*

stat_acf, 9
sugrrants-package, 2