

# Package ‘vistime’

March 3, 2019

**Title** Pretty Timeline Creation

**Version** 0.8.0

**Date** 2019-03-01

**Maintainer** Sandro Raabe <shosaco\_nospam@hotmail.com>

**Description** Create interactive timelines or Gantt charts that are usable in the 'RStudio' viewer pane, in 'R Markdown' documents and in 'Shiny' apps. Hover the mouse pointer over a point or task to show details or drag a rectangle to zoom in. Timelines and their components can afterwards be manipulated using 'plotly\_build()', which transforms the plot into a mutable list.

**License** GPL-3 | file LICENSE

**URL** <https://shosaco.github.io/vistime/>

**BugReports** <https://github.com/shosaco/vistime/issues>

**Depends** R (>= 3.2.0), plotly (>= 4.0.0)

**Imports** RColorBrewer (>= 0.2.2)

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**Suggests** ggplot2, knitr, rmarkdown, devtools, testthat, covr, purrr

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Sandro Raabe [aut, cre]

**Repository** CRAN

**Date/Publication** 2019-03-03 21:20:03 UTC

## R topics documented:

fix_columns . . . . .	2
plot_events . . . . .	3
plot_glued . . . . .	3

plot_ranges . . . . .	4
set_colors . . . . .	5
set_subplots . . . . .	5
set_y_values . . . . .	6
validate_input . . . . .	7
vistime . . . . .	8

<b>Index</b>	<b>11</b>
--------------	-----------

---

fix_columns	<i>Standardize column names</i>
-------------	---------------------------------

---

## Description

Standardize column names

## Usage

```
fix_columns(data, events, start, end, groups, tooltips)
```

## Arguments

data	input data frame
events	event column name
start	name of start column
end	name of end column
groups	name of group column
tooltips	column name of tooltips

## Value

the data frame prepared for plotting

## Examples

```
## Not run:
fix_columns(data.frame(
  event = 1:4,
  start = c("2019-01-01", "2019-01-10"),
  end = c("2019-01-01", "2019-01-10"),
  events = "event", start = "start", end = "end",
  groups = "group", tooltips = "tooltip"
))

## End(Not run)
```

---

plot_events	<i>Plot the events of a data frame</i>
-------------	--

---

**Description**

Plot the events of a data frame

**Usage**

```
plot_events(data_orig, show_labels, background_lines)
```

**Arguments**

data_orig	the data frame to be plotted (ranges + events)
show_labels	boolean, show labels on events or not
background_lines	number of grey background lines to draw

**Value**

a list containing the plots for the groups in data

**Examples**

```
## Not run:
plot_events(data.frame(
  event = 1:2, start = as.POSIXct(c("2019-01-01", "2019-01-10")),
  end = as.POSIXct(c("2019-01-01", "2019-01-10")),
  group = "", tooltip = "", col = "green", fontcol = "black",
  subplot = 1, y = 1:2, labelPos = "center", label = 1:2
),
show_labels = TRUE, background_lines = 11
)

## End(Not run)
```

---

plot_glued	<i>PLot ranges and events together as one plot</i>
------------	--

---

**Description**

PLot ranges and events together as one plot

**Usage**

```
plot_glued(data, title, ranges, events)
```

**Arguments**

data	the data frame to plot
title	title of the plot, can be NULL
ranges	ranges as generated by plot_ranges
events	events as generated by plot_ranges

**Value**

plotly plot

---

plot_ranges	<i>Plot the ranges of a data frame</i>
-------------	--

---

**Description**

Plot the ranges of a data frame

**Usage**

```
plot_ranges(data_orig, linewidth, show_labels, background_lines)
```

**Arguments**

data_orig	the data frame to be plotted (ranges + events)
linewidth	the width in pixel for the range lines
show_labels	boolean, show labels on events or not
background_lines	number of grey background lines to draw

**Value**

a list containing the plots for the groups in data

**Examples**

```
## Not run:
plot_ranges(data.frame(
  event = 1:2, start = as.POSIXct(c("2019-01-01", "2019-01-10")),
  end = as.POSIXct(c("2019-01-10", "2019-01-25")),
  group = "", tooltip = "", col = "green", fontcol = "black",
  subplot = 1, y = 1:2, labelPos = "center", label = 1:2
),
linewidth = 10, show_labels = TRUE, background_lines = 11
)

## End(Not run)
```

---

set_colors	<i>Set column col for event colors and column fontcol for label colors</i>
------------	--

---

**Description**

Set column col for event colors and column fontcol for label colors

**Usage**

```
set_colors(data, eventcolor_column, fontcolor_column)
```

**Arguments**

data	the data frame containing event data
eventcolor_column	name of the event color column
fontcolor_column	name of the lable color column

**Value**

same data frame as input, but with columns col and fontcol filled with color codes or names.

---

set_subplots	<i>Find correct subplot for ranges and events</i>
--------------	---

---

**Description**

For technical reasons (plotly), ranges and events need different subplots. We are still trying to keep events and ranges that are in the same group together. We determine the subplot as follows: Groups appear in order of appearance in the data, events and ranges in the same group are plotted directly below each other

**Usage**

```
set_subplots(data)
```

**Arguments**

data	the data frame with ranges and events
------	---------------------------------------

**Value**

data with additional numeric column "subplot", reordered by subplot column.

**Examples**

```
## Not run:
set_subplots(data.frame(
  event = 1:4, start = c("2019-01-01", "2019-01-10"),
  end = c("2019-01-01", "2019-01-10"), group = ""
), stringsAsFactors = F)
set_subplots(data.frame(
  event = 1:4, start = c("2019-01-01", "2019-01-10"),
  end = c("2019-01-01", "2019-01-10"), group = 1:2
), stringsAsFactors = F)
set_subplots(data.frame(
  event = 1:3, start = c("2019-01-01", "2019-01-10", "2019-01-01"),
  end = c("2019-01-10", "2019-01-20", "2019-01-10"),
  group = c(1, 2, 1), stringsAsFactors = F
))

## End(Not run)
```

---

set\_y\_values

*Heuristic to distribute events and ranges in y space*


---

**Description**

Instead of naive "always increment by 1" approach, we are using a more sophisticated method to use plot space efficiently

**Usage**

```
set_y_values(data)
```

**Arguments**

**data** the data frame with data to be distributed, has to have start, end and subplot column

**Value**

the data frame enriched with numeric y column

**Examples**

```
## Not run:
set_y_values(data.frame(
  event = 1:4, start = c("2019-01-01", "2019-01-10"),
  end = c("2019-01-01", "2019-01-10"), subplot = 1
), stringsAsFactors = F)

## End(Not run)
```

---

validate_input	<i>Validate input data</i>
----------------	----------------------------

---

**Description**

Validate input data

**Usage**

```
validate_input(data, start, end, events, groups, linewidth, title,  
              showLabels, show_labels, lineInterval, background_lines)
```

**Arguments**

data	the data
start	start dates
end	end dates
events	event column name
groups	group column name
linewidth	width of range lines
title	plot title
showLabels	deprecated, replaced by show_labels
show_labels	boolean
lineInterval	deprecated, replaced by background_lines
background_lines	interval of grey background lines

**Value**

the data frame with possibly new or renamed columns, or an error

**Examples**

```
## Not run:  
validate_input(data.frame(event = 1:2, start = c("2019-01-01", "2019-01-10")),  
              events = "event", start = "start", end = "end", groups = "group",  
              linewidth = NULL, title = NULL, showLabels = NULL, show_labels = TRUE,  
              lineInterval = NULL, background_lines = 11  
              )  
## End(Not run)
```

vistime

*Create a Timeline***Description**

Visualize interactive timelines offline.

Provide a data frame with event data to create a visual timeline plot. Simplest drawable dataframe can have columns 'event' and 'start'.

**Usage**

```
vistime(data, events = "event", start = "start", end = "end",
        groups = "group", colors = "color", fontcolors = "fontcolor",
        tooltips = "tooltip", linewidth = NULL, title = NULL,
        showLabels = NULL, show_labels = TRUE, lineInterval = NULL,
        background_lines = 11)
```

**Arguments**

data	(required) data.frame that contains the data to be visualised
events	(optional) the column name in data that contains event names. Default: <i>event</i> .
start	(optional, character) the column name in data that contains start dates. Default: <i>start</i> .
end	(optional, character) the column name in data that contains end dates. Default: <i>end</i> .
groups	(optional, character) the column name in data to be used for grouping. Default: <i>group</i> .
colors	(optional, character) the column name in data that contains colors for events. Default: <i>color</i> , if not present, colors are chosen via RColorBrewer.
fontcolors	(optional, character) the column name in data that contains the font color for event labels. Default: <i>fontcolor</i> , if not present, color will be black.
tooltips	(optional, character) the column name in data that contains the mouseover tooltips for the events. Default: <i>tooltip</i> , if not present, then tooltips are build from event name and date.
linewidth	(optional, numeric) the linewidth (in pixel) for the events (typically used for large amount of parallel events). Default: heuristic value.
title	(optional, character) the title to be shown on top of the timeline. Default: NULL.
showLabels	deprecated and replaced by argument <i>show_labels</i> .
show_labels	(optional, boolean) choose whether or not event labels shall be visible. Default: TRUE.
lineInterval	deprecated, use argument <i>background_lines</i> instead.
background_lines	(optional, integer) the number of vertical lines to draw in the background to demonstrate structure (default: 10). Less means more memory-efficient plot.



**Value**

vistime returns an object of class plotly and htmlwidget.

**Author(s)**

Sandro Raabe <shosaco\_nospam@hotmail.com>

**Examples**

```
# presidents and vice presidents
pres <- data.frame(
  Position = rep(c("President", "Vice"), each = 3),
  Name = c("Washington", rep(c("Adams", "Jefferson"), 2), "Burr"),
  start = c("1789-03-29", "1797-02-03", "1801-02-03"),
  end = c("1797-02-03", "1801-02-03", "1809-02-03"),
  color = c("#cbb69d", "#603913", "#c69c6e"),
  fontcolor = c("black", "white", "black")
)

vistime(pres, events = "Position", groups = "Name", title = "Presidents of the USA")
## Not run:
# more complex and colorful example
data <- read.csv(text = "event,group,start,end,color
Phase 1,Project,2018-12-22,2018-12-23,#c8e6c9
Phase 2,Project,2018-12-23,2018-12-29,#a5d6a7
Phase 3,Project,2018-12-29,2019-01-06,#fb8c00
Phase 4,Project,2019-01-06,2019-02-02,#DD4B39
Room 334,Team 1,2018-12-22,2018-12-28,#DEEBF7
Room 335,Team 1,2018-12-28,2019-01-05,#C6DBEF
Room 335,Team 1,2019-01-05,2019-01-23,#9ECAE1
Group 1,Team 2,2018-12-22,2018-12-28,#E5F5E0
Group 2,Team 2,2018-12-28,2019-01-23,#C7E9C0
3-200,category 1,2018-12-25,2018-12-25,#1565c0
3-330,category 1,2018-12-25,2018-12-25,#1565c0
3-223,category 1,2018-12-28,2018-12-28,#1565c0
3-225,category 1,2018-12-28,2018-12-28,#1565c0
3-226,category 1,2018-12-28,2018-12-28,#1565c0
3-226,category 1,2019-01-19,2019-01-19,#1565c0
3-330,category 1,2019-01-19,2019-01-19,#1565c0
1-217.0,category 2,2018-12-27,2018-12-27,#90caf9
3-399.7,moon rising,2019-01-13,2019-01-13,#f44336
8-831.0,sundowner drink,2019-01-17,2019-01-17,#8d6e63
9-984.1,birthday party,2018-12-22,2018-12-22,#90a4ae
F01.9,Meetings,2018-12-26,2018-12-26,#e8a735
Z71,Meetings,2019-01-12,2019-01-12,#e8a735
B95.7,Meetings,2019-01-15,2019-01-15,#e8a735
T82.7,Meetings,2019-01-15,2019-01-15,#e8a735")

vistime(data)

# ----- It is possible to change all attributes of the timeline using plotly_build(),
# ----- which generates a list which can be inspected using str
```

```
p <- vistime(data.frame(event = 1:4, start = c("2019-01-01", "2019-01-10")))
pp <- plotly_build(p) # transform into a list

# Example 1: change x axis font size:
pp$x$layout$xaxis$tickfont <- list(size = 28)
pp

# Example 2: change y axis font size (several y-axes, therefore we need a loop):
for (i in grep("yaxis*", names(pp$x$layout))) pp$x$layout[[i]]$tickfont <- list(size = 28)
pp

# Example 3: Changing events font size
for (i in 1:length(pp$x$data)) {
  if (pp$x$data[[i]]$mode == "text") pp$x$data[[i]]$textfont$size <- 28
}
pp

# or, using purrr:
text_idx <- which(purrr::map_chr(pp$x$data, "mode") == "text")
for(i in text_idx) pp$x$data[[i]]$textfont$size <- 28
pp

# Example 4: change marker size
# loop over pp$x$data, and change the marker size of all text elements to 50px
for (i in 1:length(pp$x$data)) {
  if (pp$x$data[[i]]$mode == "markers") pp$x$data[[i]]$marker$size <- 40
}
pp

# or, using purrr:
marker_idx <- which(purrr::map_chr(pp$x$data, "mode") == "markers")
for(i in marker_idx) pp$x$data[[i]]$marker$size <- 40
pp

## End(Not run)
```

# Index

- \*Topic **gantt**
  - vistime, 8
- \*Topic **plotly**
  - vistime, 8
- \*Topic **timeline**
  - vistime, 8
- \*Topic **vistime**
  - vistime, 8

fix\_columns, 2

plot\_events, 3  
plot\_glued, 3  
plot\_ranges, 4

set\_colors, 5  
set\_subplots, 5  
set\_y\_values, 6

validate\_input, 7  
vistime, 8  
vistime-package (vistime), 8