Package 'linevis'

July 22, 2025

Title Interactive Time Series Visualizations

Version 1.0.0

Description Create interactive time series visualizations.
'linevis' includes an extensive API to manipulate time series after creation, and supports getting data out of the visualization. Based on the 'timevis' package and the 'vis.js' Timeline 'JavaScript' library <https://visjs.github.io/vis-timeline/docs/graph2d/>.

URL https://gitlab.com/thomaschln/linevis

BugReports https://gitlab.com/thomaschln/linevis/-/issues

Depends R (>= 3.1.0)

Imports crosstalk, htmlwidgets (>= 0.6), jsonlite, magrittr, methods, rmarkdown, shiny

Suggests knitr, testthat (>= 0.9.1)

VignetteBuilder knitr

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Thomas Charlon [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7497-0470>),
Dean Attali [aut, cph],
Almende B.V. [aut, cph] (vis.js Timeline library,
https://visjs.github.io/vis-timeline/docs/graph2d/)

Maintainer Thomas Charlon <charlon@protonmail.com>

Repository CRAN

Date/Publication 2025-01-10 20:50:02 UTC

Contents

`addCustomTime` *Add a new vertical bar at a time point that can be dragged by the user*

Description

Add a new vertical bar at a time point that can be dragged by the user

Usage

```
addCustomTime(id, time, itemId)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`time`	The date/time to add
`itemId`	The id of the custom time bar

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis() %>%
  addCustomTime(Sys.Date() - 1, "yesterday")

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Add time bar 24 hours ago")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis()
    )
    observeEvent(input$btn, {
      addCustomTime("graph2d", Sys.Date() - 1, "yesterday")
    })
  }
)
}
```

`centerItem` *Move the window such that given item or items are centered*

Description

Move the window such that given item or items are centered

Usage

```
centerItem(id, itemId, options)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`itemId`	A vector (or single value) of the item ids to center
`options`	Named list of options controlling mainly the animation. Most common option is `"animation" = TRUE/FALSE`. For a full list of options, see the "focus" method in the official graph2d documentation

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis(data.frame(
          start = c(Sys.Date() - 1, Sys.Date(), Sys.Date() + 1),
          content = c("Item 1", "Item 2", "Item 3"))
) %>%
  centerItem(1)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Center around item 1")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis(
        data.frame(
          start = c(Sys.Date() - 1, Sys.Date(), Sys.Date() + 1),
          content = c("Item 1", "Item 2", "Item 3"))
      )
    )
    observeEvent(input$btn, {
      centerItem("graph2d", 1)
    })
  }
)
}
```

`centerTime` *Move the window such that the given time is centered*

Description

Move the window such that the given time is centered

Usage

```
centerTime(id, time, options)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`time`	The date/time to center around
`options`	Named list of options controlling the animation. Most common option is `"animation"` = `TRUE/FALSE`. For a full list of options, see the "moveTo" method in the official graph2d documentation

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis() %>%
  centerTime(Sys.Date() - 1)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Center around 24 hours ago")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis()
    )
    observeEvent(input$btn, {
      centerTime("graph2d", Sys.Date() - 1)
    })
  }
)
}
```

`fitWindow` *Adjust the visible window such that it fits all items*

Description

Adjust the visible window such that it fits all items

Usage

```
fitWindow(id, options)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`options`	Named list of options controlling the animation. Most common option is `"animation" = TRUE/FALSE`. For a full list of options, see the "fit" method in the official graph2d documentation

Value

None, side-effect is Javascript call

Examples

```
if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Fit all items")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis(data.frame(
        start = c(Sys.Date(), Sys.Date() - 1), content = c("1", "2")
      ))
    )
    observeEvent(input$btn, {
      fitWindow("graph2d", list(animation = FALSE))
    })
  }
)
}
```

`linevis` *Create a graph2d visualization*

Description

`linevis` lets you create rich and fully interactive graph2d visualizations. graph2ds can be included in Shiny apps or R markdown documents. `linevis` Includes an extensive API to manipulate a graph2d after creation, and supports getting data out of the visualization into R. Based on the 'visjs' graph2d JavaScript library.

To see the full details on what the graph2d can support, please read the official documentation of visjs graph2d.

Usage

```
linevis(
  data,
  groups,
  zoomFactor = 0.5,
  fit = TRUE,
  log_scale = FALSE,
  options = list(),
  width = NULL,
```

```
  height = NULL,
  elementId = NULL,
  loadDependencies = TRUE,
  timezone = NULL
)
```

Arguments

`data`	A dataframe (or SharedData object for {crosstalk} support) containing the graph2d items. Each item on the graph2d is represented by a row in the dataframe. `x` and `y` are the only two required columns. See the **Data format** section below for more details. For a full list of all supported columns, see the Data Format section in the official visjs graph2d documentation.
`groups`	A dataframe containing the groups data (optional). See the **Groups** section below for more details. buttons on the widget.
`zoomFactor`	How much to zoom when zooming out. A zoom factor of 0.5 means that when zooming out the graph2d will show 50% more content. For example, if the graph2d currently shows 20 days, then after zooming out with a `zoomFactor` of 0.5, the graph2d will show 30 days, and zooming out again will show 45 days. Similarly, zooming out from 20 days with a `zoomFactor` of 1 will results in showing 40 days.
`fit`	If `TRUE`, then fit all the data on the graph2d when the graph2d initializes. Otherwise, the graph2d will be set to show the current date.
`log_scale`	If `TRUE`, use log scaling on vertical axis.
`options`	A named list containing any extra configuration options to customize the graph2d. All available options can be found in the official graph2d documentation. Note that any options that define a JavaScript function must be wrapped in a call to `htmlwidgets::JS()`. See the examples section below to see example usage. If using {crosstalk}, it's recommended to use 'list(multiselect = TRUE)'.
`width`	Fixed width for graph2d (in css units). Ignored when used in a Shiny app – use the `width` parameter in `linevisOutput`. It is not recommended to use this parameter because the widget knows how to adjust its width automatically.
`height`	Fixed height for graph2d (in css units). It is recommended to not use this parameter since the widget knows how to adjust its height automatically.
`elementId`	Use an explicit element ID for the widget (rather than an automatically generated one). Ignored when used in a Shiny app.
`loadDependencies`	Whether to load JQuery and bootstrap dependencies (you should only set to `FALSE` if you manually include them)
`timezone`	By default, the linevis widget displays times in the local time of the browser rendering it. You can set linevis to display times in another time zone by providing a number between -15 to 15 to specify the number of hours offset from UTC. For example, use '0' to display in UTC, and use '-4' to display in a timezone that is 4 hours behind UTC.

Value

A graph2d visualization `htmlwidgets` object

Data format

The `data` parameter supplies the input dataframe that describes the items in the graph2d. The following is a subset of the variables supported in the items dataframe. **The full list of supported variables can be found in the official visjs documentation**.

- `id` - A unique ID for the item. If not provided, then the row names will be used as IDs.
- `title` - Add a title for the item, displayed when hovering the mouse over the item. The title can only contain plain text.
- `group` - The ID of a group. When a `group` is provided, all items with the same group are placed on one line. A vertical axis is displayed showing the group names. See more details in the **Groups** section below.
- `className` - A className can be used to give items an individual CSS style.
- `style` - A CSS text string to apply custom styling for an individual item, for example `color: red;`.

Groups

The `groups` parameter must be provided if the data items have groups (i.e. if any of the items have a `group` variable). The following is a subset of the variables supported in the groups dataframe. **The full list of supported variables can be found in the official visjs documentation**.

- `id` - (required) An ID for the group. The group will display all items having a `group` variable which matches this ID.
- `content` - (required) The contents of the group. This can be plain text or HTML code.
- `className` - A className can be used to give groups an individual CSS style.
- `style` - A CSS text string to apply custom styling for an individual group label, for example `color: red;`.

`id` and `content` are the only required variables for each group, while the rest of the variables are optional. If you include a variable that is only used for some rows, you can use `NA` for the rows where it's not used. The groups data of a graph2d can either be set by supplying the `groups` argument to `linevis()`, or by calling the `setGroups` function.

Getting data out of a graph2d in Shiny

When a graph2d widget is created in a Shiny app, there are four pieces of information that are always accessible as Shiny inputs. These inputs have special names based on the graph2d's ID. Suppose that a graph2d is created with an `outputId` of **"mytime"**, then the following four input variables will be available:

- `input$mytime_data` - will return a data.frame containing the data of the items in the graph2d. The input is updated every time an item is modified, added, or removed.
- `input$mytime_ids` - will return the IDs (a vector) of all the items in the graph2d. The input is updated every time an item is added or removed from the graph2d.

- `input$mytime_window` - will return a 2-element vector containing the minimum and maximum dates currently visible in the graph2d. The input is updated every time the viewable window of dates is updated (by zooming or moving the window).
- `input$mytime_visible` - will return a list of IDs of items currently visible in the graph2d.

All four inputs will return a value upon initialization of the graph2d and every time the corresponding value is updated.

Extending linevis

If you need to perform any actions on the graph2d object that are not supported by this package's API, you may be able to do so by manipulating the graph2d's JavaScript object directly. The graph2d object is available via `document.getElementById("id").widget.graph2d` (replace `id` with the graph2d's ID).

This graph2d object is the direct widget that `vis.js` creates, and you can see the visjs documentation to see what actions you can perform on that object.

Customizing the linevis look and style using CSS

To change the styling of individual items or group labels, use the `className` and `style` columns in the `data` or `groups` dataframes.

When running a Shiny app, you can use CSS files to apply custom styling to other components of the linevis widget.

Examples

```
## Not run:

#----------------------- Most basic -----------------
linevis()

#----------------------- Minimal data -----------------
df_data = data.frame(x = c('2014-06-11',
                           '2014-06-12',
                           '2014-06-13',
                           '2014-06-14',
                           '2014-06-15',
                           '2014-06-16'),
                     y = c(0,
                           1,
                           30000,
                           10,
                           150,
                           30000,
                           20,
                           20))

linevis(df_data)
```

```
#---------------------- Using groups -----------------
df_data = rbind(df_data, data.frame(x = c('2014-06-09', '2014-06-18'),
                                    y = c(20, 20)))
df_data$group = c(rep(0, 6), 1, 1)

df_grp = data.frame(id = 0:1, content = c('ESR', 'threshold'),
                    className = c('grp1', 'grp2'))

linevis(df_data, df_grp)

#---------------------- Getting data out of the graph2d into Shiny -----------------
if (interactive()) {
library(shiny)

ui <- fluidPage(
  linevisOutput("appts"),
  div("Visible window:", textOutput("window", inline = TRUE)),
  tableOutput("table")
)

server <- function(input, output) {
  output$appts <- renderLinevis(
    linevis(df_data)
  )

  output$window <- renderText(
    paste(input$appts_window[1], "to", input$appts_window[2])
  )

  output$table <- renderTable(
    input$appts_data
  )
}
shinyApp(ui, server)
}

## End(Not run)
```

`linevis-shiny` *Shiny bindings for linevis*

Description

Output and render functions for using linevis within Shiny applications and interactive Rmd documents.

Usage

```
linevisOutput(outputId, width = "100%", height = "auto")
```

```
renderLinevis(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended. height will probably not have an effect; instead, use the height parameter in linevis.
expr	An expression that generates a linevis
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

Value

Htmlwidgets render and output objects

See Also

linevis.

Examples

```
if (interactive()) {
library(shiny)

#---------------------- Most basic example -----------------
shinyApp(
  ui = fluidPage(linevisOutput("graph2d")),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis()
    )
  }
)


#---------------------- More advanced example -----------------
df_data = data.frame(x = c('2014-06-11',
                           '2014-06-12',
                           '2014-06-13',
                           '2014-06-14',
                           '2014-06-15',
                           '2014-06-16'),
                     y = c(0,
                           1,
                           30000,
                           10,
                           150,
```

```
                            30000,
                            20,
                            20))

  ui <- fluidPage(
    linevisOutput("appts"),
    div("Visible window:", textOutput("window", inline = TRUE)),
    tableOutput("table")
  )

  server <- function(input, output) {
    output$appts <- renderLinevis(
      linevis(df_data)
    )

    output$window <- renderText(
      paste(input$appts_window[1], "to", input$appts_window[2])
    )

    output$table <- renderTable(
      input$appts_data
    )
  }
  shinyApp(ui, server)
  }
```

`removeCustomTime` *Remove a custom time previously added*

Description

Remove a custom time previously added

Usage

```
removeCustomTime(id, itemId)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`itemId`	The id of the custom time bar

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis() %>%
  addCustomTime(Sys.Date() - 1, "yesterday") %>%
  addCustomTime(Sys.Date() + 1, "tomorrow") %>%
  removeCustomTime("yesterday")

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn0", "Add custom time"),
    actionButton("btn", "Remove custom time bar")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis()
    )
    observeEvent(input$btn0, {
      addCustomTime("graph2d", Sys.Date() - 1, "yesterday")
    })
    observeEvent(input$btn, {
      removeCustomTime("graph2d", "yesterday")
    })
  }
)
}
```

`setCurrentTime` *Adjust the time of the current time bar*

Description

Adjust the time of the current time bar

Usage

```
setCurrentTime(id, time)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`time`	The new date/time

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis() %>%
  setCurrentTime(Sys.Date())

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Set current time to beginning of today")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis()
    )
    observeEvent(input$btn, {
      setCurrentTime("graph2d", Sys.Date())
    })
  }
)
}
```

`setCustomTime` *Adjust the time of a custom time bar*

Description

Adjust the time of a custom time bar

Usage

```
setCustomTime(id, time, itemId)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`time`	The new date/time
`itemId`	The id of the custom time bar

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis() %>%
  addCustomTime(Sys.Date(), "yesterday") %>%
  setCustomTime(Sys.Date() - 1, "yesterday")

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Set time bar 24 hours ago")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis() %>% addCustomTime(Sys.Date(), "yesterday")
    )
    observeEvent(input$btn, {
      setCustomTime("graph2d", Sys.Date() - 1, "yesterday")
    })
  }
)
}
```

`setGroups` *Set the groups of a graph2d*

Description

Set the groups of a graph2d

Usage

```
setGroups(id, data)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`data`	A dataframe containing the groups data to use.

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis(data = data.frame(
  start = c(Sys.Date(), Sys.Date(), Sys.Date() + 1, Sys.Date() + 2),
  content = c("one", "two", "three", "four"),
  group = c(1, 2, 1, 2)),
  groups = data.frame(id = 1:2, content = c("G1", "G2"))
) %>%
  setGroups(data.frame(id = 1:2, content = c("Group 1", "Group 2")))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Change group names")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis(data = data.frame(
                start = c(Sys.Date(), Sys.Date(), Sys.Date() + 1, Sys.Date() + 2),
                          content = c("one", "two", "three", "four"),
                          group = c(1, 2, 1, 2)),
              groups = data.frame(id = 1:2, content = c("G1", "G2")))

    )
    observeEvent(input$btn, {
      setGroups("graph2d",
                data.frame(id = 1:2, content = c("Group 1", "Group 2")))
    })
  }
)
}
```

`setItems` *Set the items of a graph2d*

Description

Set the items of a graph2d

Usage

```
setItems(id, data)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`data`	A dataframe containing the item data to use.

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis(data.frame(start = Sys.Date(), content = "Today")) %>%
  setItems(data.frame(start = Sys.Date() - 1, content = "yesterday"))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Change the data to yesterday")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis(data.frame(start = Sys.Date(), content = "Today"))
    )
    observeEvent(input$btn, {
      setItems("graph2d",
               data.frame(start = Sys.Date() - 1, content = "yesterday"))
    })
  }
)
}
```

---

setOptions *Update the configuration options of a graph2d*

Description

Update the configuration options of a graph2d

Usage

```
setOptions(id, options)
```

Arguments

id	graph2d id or a linevis object (the output from linevis())
options	A named list containing updated configuration options to use. See the options parameter of the linevis function to see more details.

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis(
  data.frame(start = Sys.Date(), content = "Today"),
  options = list(showCurrentTime = FALSE, orientation = "top")
) %>%
  setOptions(list(editable = TRUE, showCurrentTime = TRUE))

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Show current time and allow items to be editable")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis(
        data.frame(start = Sys.Date(), content = "Today"),
        options = list(showCurrentTime = FALSE, orientation = "top")
      )
    )
    observeEvent(input$btn, {
      setOptions("graph2d", list(editable = TRUE, showCurrentTime = TRUE))
    })
  }
)
}
```

`setWindow` *Set the current visible window*

Description

Set the current visible window

Usage

```
setWindow(id, start, end, options)
```

Arguments

`id`	graph2d id or a `linevis` object (the output from `linevis()`)
`start`	The start date/time to show in the graph2d
`end`	The end date/time to show in the graph2d
`options`	Named list of options controlling mainly the animation. Most common option is `animation = TRUE/FALSE`. For a full list of options, see the "setWindow" method in the official graph2d documentation

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis() %>%
  setWindow(Sys.Date() - 1, Sys.Date() + 1)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    actionButton("btn", "Set window to show between yesterday to tomorrow")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis()
    )
    observeEvent(input$btn, {
      setWindow("graph2d", Sys.Date() - 1, Sys.Date() + 1)
    })
  }
)
}
```

---

zoom *Zoom in/out the current visible window*

Description

Zoom in/out the current visible window

Usage

```
zoomIn(id, percent = 0.5, animation = TRUE)

zoomOut(id, percent = 0.5, animation = TRUE)
```

Arguments

id	graph2d id or a `linevis` object (the output from `linevis()`)
percent	The amount to zoom in or out. Must be a number between 0 and 1. A value of 0.5 means that after zooming out the graph2d will show 50% more content.
animation	Whether or not to animate the zoom.

Value

None, side-effect is Javascript call

Examples

```
## Not run:
linevis() %>%
  zoomIn()

linevis() %>%
  zoomOut(0.3)

## End(Not run)

if (interactive()) {
library(shiny)
shinyApp(
  ui = fluidPage(
    linevisOutput("graph2d"),
    sliderInput("zoom", "Zoom by", min = 0, max = 1, value = 0.5, step = 0.1),
    checkboxInput("animate", "Animate?", TRUE),
    actionButton("zoomIn", "Zoom IN"),
    actionButton("zoomOut", "Zoom OUT")
  ),
  server = function(input, output) {
    output$graph2d <- renderLinevis(
      linevis()
    )
    observeEvent(input$zoomIn, {
      zoomIn("graph2d", percent = input$zoom, animation = input$animate)
    })
    observeEvent(input$zoomOut, {
      zoomOut("graph2d", percent = input$zoom, animation = input$animate)
    })
  }
)
}
```

`%>%` *Pipe Pipe an object forward into a function or call expression. Magrittr imported function, see details and examples in the magrittr package.*

Description

Pipe

Pipe an object forward into a function or call expression. Magrittr imported function, see details and examples in the magrittr package.

Arguments

lhs	A value or the magrittr placeholder.
rhs	A function call using the magrittr semantics.

Value

Result of rhs applied to lhs, see details in magrittr package.

Index