Package ‘plotly’

July 29, 2017

Title Create Interactive Web Graphics via ‘plotly.js’
Version 4.7.1
License MIT + file LICENSE
Description Easily translate ‘ggplot2’ graphs to an interactive web-based version and/or create custom web-based visualizations directly from R. Once uploaded to a ‘plotly’ account, ‘plotly’ graphs (and the data behind them) can be viewed and modified in a web browser.
BugReports https://github.com/ropensci/plotly/issues
Depends R (>= 3.2.0), ggplot2 (>= 2.2.1)
Imports tools, scales, httr, jsonlite, magrittr, digest, viridisLite, base64enc, htmltools, htmlwidgets (>= 0.9), tidyr, hexbin, RColorBrewer, dplyr, tibble, lazyeval (>= 0.2.0), crosstalk, purrr, data.table
Suggests MASS, maps, ggthemes, GGally, testthat, knitr, devtools, shiny (>= 0.14), curl, rmarkdown, Rserve, RSclient, Cairo, broom, webshot, listviewer, dendextend, sf, RSelenium, png, IRdisplay
LazyData true
RoxygenNote 6.0.1
NeedsCompilation no
Author Carson Sievert [aut, cre],
            Chris Parmer [aut],
            Toby Hocking [aut],
            Scott Chamberlain [aut],
            Karthik Ram [aut],
            Marianne Corvellec [aut],
            Pedro Despouy [aut],
            Plotly Technologies Inc. [cph]
Maintainer Carson Sievert <cpsievert1@gmail.com>
Repository CRAN
Date/Publication 2017-07-29 05:16:25 UTC
R topics documented:

add_annotations .................................................. 3
add_data .......................................................... 4
add_fun ............................................................ 4
add_trace .......................................................... 5
animation_opts ...................................................... 8
api_create .......................................................... 10
as.widget .......................................................... 13
as_widget .......................................................... 13
attrs_selected ....................................................... 14
bbox ................................................................. 14
colorbar ............................................................. 15
config ................................................................. 16
embed_notebook ...................................................... 17
event_data .......................................................... 17
export ................................................................. 18
geom2trace .......................................................... 19
get_figure ............................................................ 20
get_l ................................................................. 20
get_x ................................................................. 20
get_y ................................................................. 21
gg2list ............................................................... 21
ggplotly .............................................................. 22
group2NA ............................................................ 23
hide_colorbar ......................................................... 25
hide_guides .......................................................... 25
hide_legend .......................................................... 26
highlight ............................................................ 26
hobbs ................................................................. 28
knit_print.api_grid .................................................. 29
knit_print.api_grid_local ......................................... 29
knit_print.api_plot ................................................. 30
last_plot ............................................................ 30
layout ............................................................... 31
mic ................................................................. 31
offline ............................................................... 32
plotly-shiny ........................................................ 32
plotlyProxy .......................................................... 33
plotly_build ........................................................ 34
plotly_data .......................................................... 34
plotly_empty ........................................................ 37
plotly_example ..................................................... 37
plotly_IMAGE ....................................................... 38
plotly_json .......................................................... 38
plotly_POST ........................................................ 39
plotly_dendro ....................................................... 40
plot_geo ............................................................. 41
### add_annotations

Add annotation(s) to a plot

#### Description

Add an annotation(s) to a plot

#### Usage

```r
add_annotations(p, text = NULL, ..., data = NULL, inherit = TRUE)
```

#### Arguments

- `p` : a plotly object
- `text` : annotation text (required).
- `...` : these arguments are documented at [https://github.com/plotly/plotly.js/blob/master/src/components/annotations/attributes.js](https://github.com/plotly/plotly.js/blob/master/src/components/annotations/attributes.js)
- `data` : a data frame.
- `inherit` : inherit attributes from `plot_ly()`?

#### Author(s)

Carson Sievert
Examples

# single annotation
plot_ly(mtcars, x = ~wt, y = ~mpg) %>%
  slice(which.max(mpg)) %>%
  add_annotations(text = "Good mileage")

# multiple annotations
plot_ly(mtcars, x = ~wt, y = ~mpg) %>%
  filter(gear == 5) %>
  add_annotations("five cylinder", ax = 40)

---

add_data

**Add data to a plotly visualization**

**Description**

Add data to a plotly visualization

**Usage**

```
add_data(p, data = NULL)
```

**Arguments**

- `p` a plotly visualization
- `data` a data frame.

**Examples**

```
plot_ly() %>% add_data(economics) %>% add_trace(x = ~date, y = ~pce)
```

---

add_fun

**Apply function to plot, without modifying data**

**Description**

Useful when you need two or more layers that apply a summary statistic to the original data.

**Usage**

```
add_fun(p, fun, ...)
```
**Arguments**

- **p**: a plotly object.
- **fun**: a function. Should take a plotly object as input and return a modified plotly object.
- **...**: arguments passed to fun.

**Examples**

```r
# Example with txhousing dataset
txhousing %>%
  group_by(city) %>%
  plot_ly(x = ~date, y = ~median) %>%
  add_lines(alpha = 0.2, name = "Texan Cities") %>%
  add_fun(function(plot) {
    plot %>% filter(city == "Houston") %>%
    add_lines(name = "Houston")
  }) %>%
  add_fun(function(plot) {
    plot %>% filter(city == "San Antonio") %>%
    add_lines(name = "San Antonio")
  })
```

```r
# Example with mtcars dataset
plot_ly(mtcars, x = ~wt, y = ~mpg) %>%
  add_markers() %>%
  add_fun(function(p) {
    p %>% slice(which.max(mpg)) %>%
    add_annotations("Good mileage")
  }) %>%
  add_fun(function(p) {
    p %>% slice(which.min(mpg)) %>%
    add_annotations(text = "Bad mileage")
  })
```

**Description**

Add trace(s) to a plotly visualization

**Usage**

- `add_trace(p, ..., data = NULL, inherit = TRUE)`
- `add_markers(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)`
- `add_text(p, x = NULL, y = NULL, z = NULL, text = NULL, ..., data = NULL, inherit = TRUE)`
add_trace

add_paths(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)

add_lines(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)

add_segments(p, x = NULL, y = NULL, xend = NULL, yend = NULL, ..., data = NULL, inherit = TRUE)

add_polygons(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)

add_ribbons(p, x = NULL, ymin = NULL, ymax = NULL, ..., data = NULL, inherit = TRUE)

add_area(p, r = NULL, t = NULL, ..., data = NULL, inherit = TRUE)

add_pie(p, values = NULL, labels = NULL, ..., data = NULL, inherit = TRUE)

add_bars(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)

add_histogram(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)

add_histogram2d(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)

add_histogram2dcontour(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)

add_heatmap(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)

add_contour(p, z = NULL, ..., data = NULL, inherit = TRUE)

add_boxplot(p, x = NULL, y = NULL, ..., data = NULL, inherit = TRUE)

add_surface(p, z = NULL, ..., data = NULL, inherit = TRUE)

add_mesh(p, x = NULL, y = NULL, z = NULL, ..., data = NULL, inherit = TRUE)

add_scattergeo(p, ...)

add_choropleth(p, z = NULL, ..., data = NULL, inherit = TRUE)

Arguments

\(p\) a plotly object
These arguments are documented at https://plot.ly/r/reference/ Note that acceptable arguments depend on the value of type.

data A data frame (optional) or crosstalk::SharedData object.
inherit inherit attributes from plot_ly()?
x the x variable.
y the y variable.
z a numeric matrix
text textual labels.
xend "final" x position (in this context, x represents "start")
yend "final" y position (in this context, y represents "start")
ymin a variable used to define the lower boundary of a polygon.
ymax a variable used to define the upper boundary of a polygon.
r For polar chart only. Sets the radial coordinates.
t For polar chart only. Sets the radial coordinates.
values the value to associated with each slice of the pie.
labels the labels (categories) corresponding to values.

Author(s)
Carson Sievert

References
https://plot.ly/r/reference/

See Also
plot_ly()

Examples

```r
p <- plot_ly(economics, x = ~date, y = ~unempsmed)
p
p %>% add_markers()
p %>% add_lines()
p %>% add_text(text = "")

# attributes declared in plot_ly() carry over to downstream traces,
# but can be overwritten
plot_ly(economics, x = ~date, y = ~unempsmed, color = I("red")) %>%
  add_lines() %>%
  add_markers(color = ~pop) %>%
  layout(showlegend = FALSE)
txhousing %>%
```

group_by(city) %>%
plot_ly(x = ~date, y = ~median) %>%
add_lines(fill = "black")

ggplot2::map_data("world", "canada") %>%
group_by(group) %>%
plot_ly(x = ~long, y = ~lat) %>%
add_polygons(hoverinfo = "none") %>%
add_markers(text = ~paste(name, "\n"<br />", pop), hoverinfo = "text",
data = maps::canada.cities) %>%
layout(showlegend = FALSE)

plot_ly(economics, x = ~date) %>%
add_ribbons(ymin = ~pce - 1e3, ymax = ~pce + 1e3)
p <- plot_ly(plotly::wind, r = ~r, t = ~t) %>%
add_area(color = "nms")
layout(p, radialaxis = list(ticksuffix = ""), orientation = 270)
ds <- data.frame(
  labels = c("A", "B", "C"),
  values = c(10, 40, 60)
)

plot_ly(ds, labels = ~labels, values = ~values) %>%
add_pie() %>%
layout(title = "Basic Pie Chart using Plotly")
library(dplyr)
mtcars %>%
count(vs) %>%
plot_ly(x = ~vs, y = ~n) %>%
add_bars()

plot_ly(x = ~rnorm(100)) %>%
add_histogram()
plot_ly(x = ~LETTERS, y = ~LETTERS) %>%
add_histogram2d()
z <- as.matrix(table(LETTERS, LETTERS))
plot_ly(x = ~LETTERS, y = ~LETTERS, z = ~z) %>%
add_histogram2d()
plot_ly(MASS::geyser, x = ~waiting, y = ~duration) %>%
add_histogram2dcontour()
plot_ly(z = ~volcano) %>%
add_heatmap()
plot_ly(z = ~volcano) %>%
add_contour()
plot_ly(mtcars, x = ~factor(vs), y = ~mpg) %>%
add_boxplot()
plot_ly(z = ~volcano) %>%
add_surface()
plot_ly(x = c(0, 0, 1), y = c(0, 1, 0), z = c(0, 0, 0)) %>%
add_mesh()

---

### animation_opts

**Animation configuration options**

#### Description

Animations can be created by either using the `frame` argument in `plot_ly()` or the (unofficial) `frame` `ggplot2` aesthetic in `ggplotly()`. By default, animations populate a play button and slider component for controlling the state of the animation (to pause an animation, click on a relevant
location on the slider bar). Both the play button and slider component transition between frames according rules specified by `animation_opts()`.

**Usage**

```r
animation_opts(p, frame = 500, transition = frame, easing = "linear",
               redraw = TRUE, mode = "immediate")

animation_slider(p, hide = FALSE, ...)

animation_button(p, ...)
```

**Arguments**

- `p` a plotly object.
- `frame` The amount of time between frames (in milliseconds). Note that this amount should include the transition.
- `transition` The duration of the smooth transition between frames (in milliseconds).
- `easing` The type of transition easing. See the list of options here https://github.com/plotly/plotly.js/blob/master/src/plots/animation_attributes.js
- `redraw` Trigger a redraw of the plot at completion of the transition? A redraw may significantly impact performance, but may be necessary to update graphical elements that can’t be transitioned.
- `mode` Describes how a new animate call interacts with currently-running animations. If immediate, current animations are interrupted and the new animation is started. If next, the current frame is allowed to complete, after which the new animation is started. If afterall all existing frames are animated to completion before the new animation is started.
- `hide` remove the animation slider?

... for `animation_slider`, attributes are passed to a special layout.sliders object tied to the animation frames. The definition of these attributes may be found here https://github.com/plotly/plotly.js/blob/master/src/components/sliders/attributes.js. For `animation_button`, arguments are passed to a special layout.updatemenus button object tied to the animation https://github.com/plotly/plotly.js/blob/master/src/components/updatemenus/attributes.js.

**Author(s)**

Carson Sievert

**Examples**

```r
df <- data.frame(
  x = c(1, 2, 2, 1, 1, 2),
  y = c(1, 2, 2, 1, 1, 2),
  z = c(1, 1, 2, 2, 3, 3)
)```
## api_create

*Tools for working with plotly's REST API (v2)*

### Description

Convenience functions for working with version 2 of plotly's REST API. Upload R objects to a plotly account via `api_create()` and download plotly objects via `api_download_plot()`/`api_download_grid()`. For anything else, use `api()`.

### Usage

```r
api_create(x = last_plot(), filename = NULL, fileopt = c("overwrite", "new"), sharing = c("public", "private", "secret"), ...)
```

#### # S3 method for class 'plotly'

```r
api_create(x = last_plot(), filename = NULL, fileopt = "overwrite", sharing = "public", ...)
```

#### # S3 method for class 'ggplot'

```r
api_create(x = last_plot(), filename = NULL, fileopt = "overwrite", sharing = "public", ...)
```

#### # S3 method for class 'data.frame'

```r
api_create(x = last_plot(), filename = NULL, fileopt = "overwrite", sharing = "public", ...)
```
api_create

```r
api_create(x, filename = NULL, fileopt = "overwrite",
sharing = "public", ...)
```

```r
api_download_plot(id, username)
```

```r
api_download_grid(id, username)
```

```r
api(endpoint = "/", verb = "GET", body = NULL, ...)
```

**Arguments**

- **x**: An R object to be hosted on plotly's web platform. Can be a plotly/ggplot2 object or a data.frame.
- **filename**: character vector naming file(s). If `x` is a plot, can be a vector of length 2 naming both the plot AND the underlying grid.
- **fileopt**: character string describing whether to "overwrite" existing files or ensure "new" file(s) are always created.
- **sharing**: If 'public', anyone can view this graph. It will appear in your profile and can appear in search engines. You do not need to be logged in to Plotly to view this chart. If 'private', only you can view this plot. It will not appear in the Plotly feed, your profile, or search engines. You must be logged in to Plotly to view this graph. You can privately share this graph with other Plotly users in your online Plotly account and they will need to be logged in to view this plot. If 'secret', anyone with this secret link can view this chart. It will not appear in the Plotly feed, your profile, or search engines. If it is embedded inside a webpage or an IPython notebook, anybody who is viewing that page will be able to view the graph. You do not need to be logged in to view this plot.
- **...**: For `api()`, these arguments are passed onto `httr::verb()` and `httr::verb()`, these arguments are included in the body of the HTTP request.
- **id**: a filename id.
- **username**: a plotly username.
- **endpoint**: the endpoint (i.e., location) for the request. To see a list of all available endpoints, call `api()`. Any relevant query parameters should be included here (see examples).
- **verb**: name of the HTTP verb to use (as in, `httr::VERB()`).
- **body**: body of the HTTP request (as in, `httr::VERB()`). If this value is not already converted to JSON (via `jsonlite::toJSON()`), it uses the internal `to_json()` to ensure values are "automatically unboxed" (i.e., vec).

**Author(s)**

Carson Sievert

**References**

https://api.plot.ly/v2
See Also

```
signup()
```

Examples

```
# Not run:

# api_create() makes it easy to upload ggplot2/plotly objects
# and/or data frames to your plotly account
# A data frame creates a plotly "grid". Printing one will take you
# to the it's web address so you can start creating!
(m <- api_create(mtcars))

# A plotly/ggplot2 object create a plotly "plot".
p <- plot_ly(mtcars, x = ~factor(vs))
(r <- api_create(p))

# api_create() returns metadata about the remote "file". Here is
# one way you could use that metadata to download a plot for local use:
fileID <- strsplit(r$file$fid, ":")[[1]]
layout(
  api_download_plot(fileID[2], fileID[1]),
  title = sprintf("Local version of <a href='%s'>this</a> plot", r$file$web_url)
)

# The api() function provides a low-level interface for performing
# any action at any endpoint! It always returns a list.
# list all the endpoints
api()

# search the entire platform!
# see https://api.plot.ly/v2/search
api("search?q=overdose")
api("search?q=plottype:pie trump fake")

# these examples will require a user account
usr <- Sys.getenv("plotly_username", NA)
if (!is.na(usr)) {
  # your account info https://api.plot.ly/v2/#users
  api(sprintf("users/%s", usr))
  # your folders/files https://api.plot.ly/v2/folders#user
  api(sprintf("folders/home?user=%s", usr))
}

# Retrieve a specific file https://api.plot.ly/v2/files#retrieve
```
as.widget

Convert a plotly object to an htmlwidget object

Description
This function was deprecated in 4.0.0, as plotly objects are now htmlwidget objects, so there is no need to convert them.

Usage
as.widget(x, ...)

Arguments
x a plotly object.
... other options passed onto htmlwidgets::createWidget

as_widget

Convert a list to a plotly htmlwidget object

Description
Convert a list to a plotly htmlwidget object

Usage
as_widget(x, ...)
Arguments

- `x` a plotly object.
- `...` other options passed onto `htmlwidgets::createWidget`

Examples

```r
trace <- list(x = 1, y = 1)
obj <- list(data = list(trace), layout = list(title = "my plot"))
as_widget(obj)
```

attrs_selected

Specify attributes of selection traces

Description

By default the name of the selection trace derives from the selected values.

Usage

```r
attrs_selected(opacity = 1, ...)
```

Arguments

- `opacity` a number between 0 and 1 specifying the overall opacity of the selected trace
- `...` other trace attributes attached to the selection trace.

Author(s)

Carson Sievert

bbox

Estimate bounding box of a rotated string

Description

Estimate bounding box of a rotated string

Usage

```r
bbox(txt = "foo", angle = 0, size = 12)
```
**colorbar**

**Arguments**

- **txt**: a character string of length 1
- **angle**: sets the angle of the tick labels with respect to the horizontal (e.g., tickangle of -90 draws the tick labels vertically)
- **size**: vertical size of a character

**References**

https://www.dropbox.com/s/nc6968prgw8ne4w/bbox.pdf?dl=0

---

**colorbar**

*Modify the colorbar*

---

**Description**

Modify the colorbar

**Usage**

```r
colorbar(p, ..., limits = NULL, which = 1)
```

**Arguments**

- **p**: a plotly object
- **limits**: numeric vector of length 2. Set the extent of the colorbar scale.
- **which**: colorbar to modify? Should only be relevant for subplots with multiple colorbars.

**Author(s)**

Carson Sievert

**Examples**

```r
p <- plot_ly(mtcars, x = wt, y = mpg, color = cyl)

# pass any colorbar attribute --
# https://plot.ly/r/reference/#scatter-marker-colorbar
colorbar(p, len = 0.5)

# Expand the limits of the colorbar
colorbar(p, limits = c(0, 20))
# values outside the colorbar limits are considered "missing"
colorbar(p, limits = c(5, 6))
```
# also works on colorbars generated via a z value
corr <- cor(diamonds[vapply(diamonds, is.numeric, logical(1))])
plot_ly(x = rownames(corr), y = colnames(corr), z = corr) %>%
  add_heatmap() %>%
  colorbar(limits = c(-1, 1))

---

**config**

*Set the default configuration for plotly*

**Description**

Set the default configuration for plotly

**Usage**

```r
config(p, ..., collaborate = TRUE, cloud = FALSE)
```

**Arguments**

- `p` a plotly object
- `...` these arguments are documented at [https://github.com/plotly/plotly.js/blob/master/src/plot_api/plot_config.js](https://github.com/plotly/plotly.js/blob/master/src/plot_api/plot_config.js)
- `collaborate` include the collaborate mode bar button (unique to the R pkg)?
- `cloud` include the send data to cloud button?

**Author(s)**

Carson Sievert

**Examples**

```r
config(plot_ly(), displaylogo = FALSE, collaborate = FALSE)
```
**embed_notebook**

Embed a plot as an iframe into a Jupyter Notebook

**Description**

Embed a plot as an iframe into a Jupyter Notebook

**Usage**

```r
embed_notebook(x, width = NULL, height = NULL, file = NULL)
```

**Arguments**

- `x`: a plotly object
- `width`: attribute of the iframe. If NULL, the width in plot_ly is used. If that is also NULL, '100%' is the default.
- `height`: attribute of the iframe. If NULL, the height in plot_ly is used. If that is also NULL, '400px' is the default.
- `file`: deprecated.

**Author(s)**

Carson Sievert

**event_data**

Access plotly user input event data in shiny

**Description**

This function must be called within a reactive shiny context.

**Usage**

```r
event_data(event = c("plotly_hover", "plotly_click", "plotly_selected", "plotly_relayout"), source = "A", session = shiny::getDefaultReactiveDomain())
```

**Arguments**

- `event`: The type of plotly event. Currently 'plotly_hover', 'plotly_click', 'plotly_selected', and 'plotly_relayout' are supported.
- `source`: a character string of length 1. Match the value of this string with the source argument in plot_ly() to retrieve the event data corresponding to a specific plot (shiny apps can have multiple plots).
- `session`: a shiny session object (the default should almost always be used).
Description

Export a plotly graph to a static file

Usage

```r
export(p = last_plot(), file = "plotly.png", selenium = NULL, ...)
```

Arguments

- **p**: a plotly or ggplot object.
- **file**: a filename. The file type is inferred from the file extension. Valid extensions include 'jpeg' | 'png' | 'webp' | 'svg' | 'pdf'
- **selenium**: used only when `p` is a WebGL plot or the output format is 'webp' or 'svg'. Should be an object of class "rsClientServer" returned by `RSelenium::rsDriver` (see examples).
- **...**: if `p` is non-WebGL and the output file format is jpeg/png/pdf arguments are passed along to `webshot::webshot()`. Otherwise, they are ignored.

Details

For SVG plots, a screenshot is taken via `webshot::webshot()`. Since phantomjs (and hence `webshot`) does not support WebGL, the RSelenium package is used for exporting WebGL plots.

Author(s)

Carson Sievert
Examples

```r
# The webshot package handles non-WebGL conversion to jpeg/png/pdf
## Not run:
export(plot_ly(economics, x = ~date, y = ~pce))
export(plot_ly(economics, x = ~date, y = ~pce), "plot.pdf")

# svg/webp output or WebGL conversion can be done via R Selenium
if (requireNamespace("RSelenium")) {
  rd <- RSelenium::rsDriver(browser = "chrome")
  export(
    plot_ly(economics, x = ~date, y = ~pce), "plot.svg", rd
  )
  export(
    plot_ly(economics, x = ~date, y = ~pce, z = ~pop), "yay.svg", rd
  )
}

# If you can't get a selenium server running, another option is to
# use Plotly.downloadImage() via htmlwidgets::onRender()...
# Downloading images won't work inside RStudio, but you can set the viewer
# option to NULL to prompt your default web browser
options(viewer = NULL)
plot_ly(economics, x = ~date, y = ~pce, z = ~pop) %>%
htmlwidgets::onRender(
  function(el, x) {
    var gd = document.getElementById(el.id);
    Plotly.downloadImage(gd, {format: 'png', width: 600, height: 400, filename: 'plot'});
  } }

## End(Not run)
```

geom2trace

Convert a "basic" geoms to a plotly.js trace.

Description

This function makes it possible to convert ggplot2 geoms that are not included with ggplot2 itself. Users shouldn’t need to use this function. It exists purely to allow other package authors to write their own conversion method(s).

Usage

```
geom2trace(data, params, p)
```

Arguments

- `data` the data returned by plotly::to_basic.
- `params` parameters for the geom, statistic, and 'constant' aesthetics
get_x

get_figure

Description

Deprecated: see api_download_plot().

Usage

get_figure(username, id)

Arguments

username corresponding username for the figure.
id of the Plotly figure.

get_1

Description

Exported for internal reasons. Not intended for general use.

Usage

get_1(g)

Arguments

g an sf geometry

get_x

Description

Exported for internal reasons. Not intended for general use.

Usage

get_x(g)

Arguments

g an sf geometry

p a ggplot2 object (the conversion may depend on scales, for instance).
**get_y**

*Obtain y coordinates of sf geometry/geometries*

**Description**

Exported for internal reasons. Not intended for general use.

**Usage**

```r
get_y(g)
```

**Arguments**

- `g`: An sf geometry.

---

**gg2list**

*Convert a ggplot to a list.*

**Description**

Convert a ggplot to a list.

**Usage**

```r
gg2list(p, width = NULL, height = NULL, tooltip = "all",
        dynamicTicks = FALSE, layerData = 1, originalData = TRUE,
        source = "A", ...)
```

**Arguments**

- `p`: ggplot2 plot.
- `width`: Width of the plot in pixels (optional, defaults to automatic sizing).
- `height`: Height of the plot in pixels (optional, defaults to automatic sizing).
- `tooltip`: A character vector specifying which aesthetic tooltips to show in the tooltip. The default, "all", means show all the aesthetic tooltips (including the unofficial "text" aesthetic).
- `dynamicTicks`: Accepts the following values: FALSE, TRUE, "x", or "y". Dynamic ticks are useful for updating ticks in response to zoom/pan/filter interactions; however, there is no guarantee they reproduce axis tick text as they would appear in the static ggplot2 image.
- `layerData`: Data from which layer should be returned?
- `originalData`: Should the "original" or "scaled" data be returned?
- `source`: A character string of length 1. Match the value of this string with the source argument in `event_data()` to retrieve the event data corresponding to a specific plot (shiny apps can have multiple plots).
- `...`: Currently not used.
Value

a 'built' plotly object (list with names "data" and "layout").

ggplotly

Convert ggplot2 to plotly

Description

This function converts a `ggplot2::ggplot()` object to a plotly object.

Usage

```r
ggplotly(p = ggplot2::last_plot(), width = NULL, height = NULL,
    tooltip = "all", dynamicTicks = FALSE, layerData = 1,
    originalData = TRUE, source = "A", ...)
```

Arguments

- `p` a ggplot object.
- `width` Width of the plot in pixels (optional, defaults to automatic sizing).
- `height` Height of the plot in pixels (optional, defaults to automatic sizing).
- `tooltip` a character vector specifying which aesthetic mappings to show in the tooltip. The default, "all", means show all the aesthetic mappings (including the unofficial "text" aesthetic). The order of variables here will also control the order they appear. For example, use `tooltip = c("y", "x", "colour")` if you want y first, x second, and colour last.
- `dynamicTicks` should plotly.js dynamically generate axis tick labels? Dynamic ticks are useful for updating ticks in response to zoom/pan interactions; however, they can not always reproduce labels as they would appear in the static ggplot2 image.
- `layerData` data from which layer should be returned?
- `originalData` should the "original" or "scaled" data be returned?
- `source` a character string of length 1. Match the value of this string with the source argument in `event_data()` to retrieve the event data corresponding to a specific plot (shiny apps can have multiple plots).
- `...` arguments passed onto methods.

Details

Conversion of relative sizes depends on the size of the current graphics device (if no device is open, width/height of a new (off-screen) device defaults to 640/480). In other words, height and width must be specified at runtime to ensure sizing is correct.

Author(s)

Carson Sievert
Separate groups with missing values

This function is used internally by plotly, but may also be useful to some power users. The details section explains when and why this function is useful.
Usage

```r
group2NA(data, groupNames = "group", nested = NULL, ordered = NULL, retrace.first = inherits(data, "GeomPolygon"))
```

Arguments

data = a data frame.
groupNames = character vector of grouping variable(s)
nested = other variables that group should be nested (i.e., ordered) within.
ordered = a variable to arrange by (within nested & groupNames). This is useful primarily for ordering by x
retrace.first = should the first row of each group be appended to the last row? This is useful for enclosing polygons with lines.

Details

If a group of scatter traces share the same non-positional characteristics (i.e., color, fill, etc), it is more efficient to draw them as a single trace with missing values that separate the groups (instead of multiple traces), In this case, one should also take care to make sure connectgaps is set to FALSE.

Value

A data frame with rows ordered by: nested, then groupNames, then ordered. As long as groupNames contains valid variable names, new rows will also be inserted to separate the groups.

Examples

```r
# note the insertion of new rows with missing values
group2NA(mtcars, "vs", "cyl")

# need to group lines by city somehow!
plot_ly(txhousing, x = ~date, y = ~median) %>% add_lines()

# instead of using group_by(), you could use group2NA()
tx <- group2NA(txhousing, "city")
plot_ly(tx, x = ~date, y = ~median) %>% add_lines()

# add_lines() will ensure paths are sorted by x, but this is equivalent
tx <- group2NA(txhousing, "city", ordered = "date")
plot_ly(tx, x = ~date, y = ~median) %>% add_paths()
```
**hide_colorbar**

Hide color bar(s)

**Usage**

```r
hide_colorbar(p)
```

**Arguments**

- `p` a plotly object.

**See Also**

- `hide_legend`
- `hide_colorbar`

**Examples**

```r
p <- plot_ly(mtcars, x = ~wt, y = ~cyl, color = ~cyl)
hide_colorbar(p)
```

---

**hide_guides**

Hide guides (legends and colorbars)

**Description**

Hide guides (legends and colorbars)

**Usage**

```r
hide_guides(p)
```

**Arguments**

- `p` a plotly object.

**See Also**

- `hide_legend`
- `hide_colorbar()`
highlight

---

**hide_legend**

*Hide legend*

**Description**

Hide legend

**Usage**

`hide_legend(p)`

**Arguments**

- `p` a plotly object.

**See Also**

`hide_colorbar()`

**Examples**

```r
p <- plot_ly(mtcars, x = ~wt, y = ~cyl, color = ~factor(cyl))
hide_legend(p)
```

---

**highlight**

*Query graphical elements in multiple linked views*

**Description**

This function sets a variety of options for brushing (i.e., highlighting) multiple plots. These options are primarily designed for linking multiple plotly graphs, and may not behave as expected when linking plotly to another htmlwidget package via crosstalk. In some cases, other htmlwidgets will respect these options, such as persistent selection in leaflet (see demo("highlight-leaflet", package = "plotly")).

**Usage**

```r
highlight(p, on = "plotly_click", off, persistent = getOption("persistent", FALSE), dynamic = FALSE, color = NULL, selectize = FALSE, defaultValues = NULL, opacityDim = getOption("opacityDim", 0.2), selected = attrs_selected(), ...)
```
Arguments

- **p** a plotly visualization.
- **on** turn on a selection on which event(s)? To disable on events altogether, use `NULL`. Currently the following are supported:
  - 'plotly_click'
  - 'plotly_hover'
  - 'plotly_selected': triggered through rectangular (layout.dragmode = 'select') or lasso (layout.dragmode = 'lasso') brush. Currently only works for scatter traces with mode 'markers'.
- **off** turn off a selection on which event(s)? To disable off events altogether, use `NULL`. Currently the following are supported:
  - 'plotly_doubleclick': triggered on a double mouse click while (layout.dragmode = 'zoom') or (layout.dragmode = 'pan')
  - 'plotly_deselect': triggered on a double mouse click while (layout.dragmode = 'select') or (layout.dragmode = 'lasso')
  - 'plotly_relayout': triggered whenever axes are rescaled (i.e., clicking the home button in the modebar) or whenever the height/width of the plot changes.

- **persistent** should selections persist (i.e., accumulate)?
- **dynamic** should a widget for changing selection colors be included?
- **color** character string of color(s) to use for highlighting selections. See `torgb()` for valid color specifications. If `NULL` (the default), the color of selected marks are not altered.
- **selectize** provide a selectize.js widget for selecting keys? Note that the label used for this widget derives from the groupName of the SharedData object.
- **defaultValues** a vector of values for setting a "default selection". These values should match the key attribute.
- **opacityDim** a number between 0 and 1 used to reduce the opacity of non-selected traces (by multiplying with the existing opacity).
- **selected** attributes of the selection, see `attrs_selected()`. ...
  - currently not supported.

Author(s)

Carson Sievert

References


See Also

`attrs_selected()`
Examples

# These examples are designed to show you how to highlight/brush a *single* view. For examples of multiple linked views, see `demo(package = "plotly")`

library(crosstalk)
d <- SharedData$new(txhousing, ~city)
p <- ggplot(d, aes(date, median, group = city)) + geom_line()
gg <- ggplotly(p, tooltip = "city")
highlight(gg, persistent = TRUE, dynamic = TRUE)

# supply custom colors to the brush
cols <- toRGB(RColorBrewer::brewer.pal(3, "Dark2"), 0.5)
highlight(
  gg, on = "plotly_hover", color = cols, persistent = TRUE, dynamic = TRUE
)

# Use attrs_selected() for complete control over the selection appearance
# note any relevant colors you specify here should override the color argument
s <- attrs_selected(
  showlegend = TRUE,
  mode = "lines+markers",
  marker = list(symbol = "x")
)

highlight(
  layout(gg, showlegend = TRUE),
  selected = s, persistent = TRUE
)

---

hobbs  

Hobbs data

Description

Description TBD.

Usage

hobbs

Format

A data frame with three variables: r, t, nms.
knit_print.api_grid

Embed a plotly grid as an iframe in a knitr doc

Description
Embed a plotly grid as an iframe in a knitr doc

Usage
knit_print.api_grid(x, options, ...)

Arguments
x a plotly figure object
options knitr options.
... placeholder.

References

knit_print.api_grid_local

Embed a plotly grid as an iframe in a knitr doc

Description
Embed a plotly grid as an iframe in a knitr doc

Usage
knit_print.api_grid_local(x, options, ...)

Arguments
x a plotly figure object
options knitr options.
... placeholder.

References
knit_print.api_plot  
*Embed a plotly figure as an iframe in a knitr doc*

**Description**

Embed a plotly figure as an iframe in a knitr doc

**Usage**

```r
knit_print.api_plot(x, options, ...)
```

**Arguments**

- `x`  
  a plotly figure object
- `options`  
  knitr options.
- `...`  
  placeholder.

**References**


---

**last_plot**  
*Retrieve the last plot to be modified or created.*

**Description**

Retrieve the last plot to be modified or created.

**Usage**

```r
last_plot()
```

**See Also**

```r
ggplot2::last_plot()
```
layout

Modify the layout of a plotly visualization

Description

Modify the layout of a plotly visualization

Usage

layout(p, ..., data = NULL)

Arguments

p

A plotly object.

...  

Arguments to the layout object. For documentation, see https://plot.ly/r/reference/#Layout_and_layout_style_objects

data

A data frame to associate with this layout (optional). If not provided, arguments are evaluated using the data frame in plot_ly().

Author(s)

Carson Sievert

mic

Mic data

Description

Description TBD.

Usage

mic

Format

A data frame with three variables: r, t, nms.
Description

Deprecated in version 2.0 (offline plots are now the default)

Usage

offline(p, height, width, out_dir, open_browser)

Arguments

p
a plotly object

height
A valid CSS unit. (like "100%", "600px", "auto") or a number, which will be coerced to a string and have "px" appended.

width
A valid CSS unit. (like "100%", "600px", "auto") or a number, which will be coerced to a string and have "px" appended.

out_dir
a directory to place the visualization. If NULL, a temporary directory is used when the offline object is printed.

open_browser
open the visualization after creating it?

Value

a plotly object of class "offline"

Author(s)

Carson Sievert

plotly-shiny

Shiny bindings for plotly

Description

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

Usage

plotlyOutput(outputId, width = "100\%", height = "400px", inline = FALSE)

renderPlotly(expr, env = parent.frame(), quoted = FALSE)
**plotlyProxy**  

Modify a plotly object inside a shiny app

### Description

Modify a plotly object inside a shiny app

### Usage

```r
plotlyProxy(outputId, session = shiny::getDefaultReactiveDomain(),
             deferUntilFlush = TRUE)
```

```r
plotlyProxyInvoke(p, method, ...)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>outputId</code></td>
<td>single-element character vector indicating the output ID map to modify</td>
</tr>
<tr>
<td><code>session</code></td>
<td>the Shiny session object to which the map belongs; usually the default value</td>
</tr>
<tr>
<td><code>deferUntilFlush</code></td>
<td>indicates whether actions performed against this instance should be carried</td>
</tr>
<tr>
<td></td>
<td>out right away, or whether they should be held until after the next time all</td>
</tr>
<tr>
<td></td>
<td>of the outputs are updated.</td>
</tr>
<tr>
<td><code>p</code></td>
<td>a plotly proxy object (created with <code>plotlyProxy</code>)</td>
</tr>
<tr>
<td><code>method</code></td>
<td>a plotlyjs method to invoke. For a list of options, visit the <code>plotlyjs</code></td>
</tr>
<tr>
<td></td>
<td>function reference</td>
</tr>
<tr>
<td><code>...</code></td>
<td>unnamed arguments passed onto the plotlyjs method</td>
</tr>
</tbody>
</table>
Examples

```r
if (require("shiny") && interactive()) {
  plotly_example("shiny", "proxy_relayout")
  plotly_example("shiny", "proxy_mapbox")
}
```

---

**plotly_build**

*Build* (i.e., evaluate) a plotly object

**Description**

This generic function creates the list object sent to plotly.js for rendering. Using this function can be useful for overriding defaults provided by `ggplotly/plot_ly` or for debugging rendering errors.

**Usage**

`plotly_build(p, registerFrames = TRUE)`

**Arguments**

- `p`: a ggplot object, or a plotly object, or a list.
- `registerFrames`: should a frame trace attribute be interpreted as frames in an animation?

**Examples**

```r
p <- plot_ly(economics, x = ~date, y = ~pce)
# the unevaluated plotly object
str(p)
# the evaluated data
str(plotly_build(p)$x$data)
```

---

**plotly_data**

Obtain data associated with a plotly graph

**Description**

`plotly_data()` returns data associated with a plotly visualization (if there are multiple data frames, by default, it returns the most recent one).
plotly_data

Usage

plotly_data(p, id = p$x$cur_data)

## S3 method for class 'plotly'
groups(x)

## S3 method for class 'plotly'
ungroup(x, ...)

## S3 method for class 'plotly'
group_by_(.data, ..., .dots, add = FALSE)

## S3 method for class 'plotly'
summarise_(.data, ..., .dots)

## S3 method for class 'plotly'
mutate_(.data, ..., .dots)

## S3 method for class 'plotly'
do_(.data, ..., .dots)

## S3 method for class 'plotly'
arrange_(.data, ..., .dots)

## S3 method for class 'plotly'
select_(.data, ..., .dots)

## S3 method for class 'plotly'
filter_(.data, ..., .dots)

## S3 method for class 'plotly'
distinct_(.data, ..., .dots)

## S3 method for class 'plotly'
slice_(.data, ..., .dots)

## S3 method for class 'plotly'
rename_(.data, ..., .dots)

## S3 method for class 'plotly'
transmute_(.data, ..., .dots)

Arguments

p a plotly visualization

id a character string or number referencing an "attribute layer".

x a plotly visualization
... stuff passed onto the relevant method

```
data
.dots
```

a plotly visualization

```
add
```

Used to work around non-standard evaluation. See vignette("nse") for details

By default, when add = FALSE, group_by will override existing groups. To instead add to the existing groups, use add = TRUE

**Examples**

```r
# use group_by() to define groups of visual markings
p <- txhousing
  group_by(city)
  plot_ly(x = ~date, y = ~sales)
  
# plotly objects preserve data groupings
groups(p)
plotly_data(p)

# dplyr verbs operate on plotly objects as if they were data frames
p <- economics
  plot_ly(x = ~date, y = ~unemploy / pop)
  add_lines()
  mutate(rate = unemploy / pop)
  filter(rate == max(rate))
plotly_data(p)
add_markers()
layout(p, annotations = list(x = ~date, y = ~rate, text = "peak"))

# use group_by() + do() + subplot() for trellis displays
d <- group_by(mpg, drv)
plots <- do(d, p = plot_ly(.x = ~cty, name = ~drv))
subplot(plots[["p"]], nrow = 3, shareX = TRUE)

# arrange displays by their mean
means <- summarise(d, mn = mean(cty, na.rm = TRUE))
means
  dplyr::left_join(plots)
  arrange(mn)
  subplot(nrows = NROW(.), shareX = TRUE)

# more dplyr verbs applied to plotly objects
p <- mtcars
  plot_ly(x = ~wt, y = ~mpg, name = "scatter trace")
  add_markers()
  slice(1) %>% plotly_data()
  slice(1) %>% add_markers(name = "first observation")
  filter(cyl == 4) %>% plotly_data()
  filter(cyl == 4) %>% add_markers(name = "four cylinders")
```
plotly_empty

Create a complete empty plotly graph.

Description

Useful when used with subplot()

Usage

plotly_empty(...)  

Arguments

... arguments passed onto plot_ly()

plotly_example

Run a plotly example(s)

Description

Provides a unified interface for running demos, shiny apps, and Rmd documents which are bundled with the package.

Usage

plotly_example(type = c("demo", "shiny", "rmd"), name, ...)

Arguments

type      the type of example
name      the name of the example (valid names depend on type).
...       arguments passed onto the suitable method.

Author(s)

Carson Sievert
plotly_IMAGE  
Create a static image

Description
The images endpoint turns a plot (which may be given in multiple forms) into an image of the desired format.

Usage
plotly_IMAGE(x, width = 1000, height = 500, format = "png", scale = 1, out_file, ...)

Arguments
- x: either a plotly object or a list.
- width: Image width in pixels
- height: Image height in pixels
- format: The desired image format 'png', 'jpeg', 'svg', 'pdf', 'eps', or 'webp'
- scale: Both png and jpeg formats will be scaled beyond the specified width and height by this number.
- out_file: A filename for writing the image to a file.
- ... arguments passed onto httr::POST

Examples
```r
## Not run:
p <- plot_ly(x = 1:10)
png <- plotly_IMAGE(p, out_file = "plotly-test-image.png")
jpeg <- plotly_IMAGE(p, format = "jpeg", out_file = "plotly-test-image.jpeg")
svg <- plotly_IMAGE(p, format = "svg", out_file = "plotly-test-image.svg")
pdf <- plotly_IMAGE(p, format = "pdf", out_file = "plotly-test-image.pdf")
## End(Not run)
```

plotly_json  
Inspect JSON sent to plotly.js

Description
This function is useful for obtaining/viewing/debugging JSON sent to plotly.js.
**Usage**

plotly_json(p = last_plot(), jsonedit = interactive(), ...)

**Arguments**

- `p` a plotly or ggplot object.
- `jsonedit` use `listviewer::jsonedit` to view the JSON?
- `...` other options passed onto `listviewer::jsonedit`

**Examples**

```r
plotly_json(plot_ly())
plotly_json(plot_ly(), FALSE)
```

---

**plotly_POST**

*Create/Modify plotly graphs*

**Description**

Deprecated: see `api_create()`.

**Usage**

plotly_POST(x = last_plot(), filename = NULL, fileopt = "overwrite", sharing = c("public", "private", "secret"), ...)

**Arguments**

- `x` either a ggplot object, a plotly object, or a list.
- `filename` character string describing the name of the plot in your plotly account. Use `/` to specify directories. If a directory path does not exist it will be created. If this argument is not specified and the title of the plot exists, that will be used for the filename.
- `fileopt` character string describing whether to create a "new" plotly, "overwrite" an existing plotly, "append" data to existing plotly, or "extend" it.
- `sharing` If 'public', anyone can view this graph. It will appear in your profile and can appear in search engines. You do not need to be logged in to Plotly to view this chart. If 'private', only you can view this plot. It will not appear in the Plotly feed, your profile, or search engines. You must be logged in to Plotly to view this graph. You can privately share this graph with other Plotly users in your online Plotly account and they will need to be logged in to view this plot. If 'secret', anyone with this secret link can view this chart. It will not appear in the Plotly feed, your profile, or search engines. If it is embedded inside a webpage or an IPython notebook, anybody who is viewing that page will be able to view the graph. You do not need to be logged in to view this plot.
- `...` not used
plot_dendro

Description

This function takes advantage of nested key selections to implement an interactive dendrogram. Selecting a node selects all the labels (i.e. leafs) under that node.

Usage

plot_dendro(d, set = "A", xmin = -50, height = 500, width = 500, ...)

Arguments

d a dendrogram object
set defines a crosstalk group
xmin minimum of the range of the x-scale
height height
width width
... arguments supplied to subplot()

Author(s)

Carson Sievert

See Also

plot_ly(), plot_mapbox(), ggplotly()

Examples

hc <- hclust(dist(USArrests), "ave")
dend1 <- as.dendrogram(hc)
plot_dendro(dend1, height = 600) %>%
  hide_legend() %>%
  highlight(persistent = TRUE, dynamic = TRUE)
**plot_geo**  
*Initiate a plotly-geo object*

**Description**

Use this function instead of `plot_ly()` to initialize a plotly-geo object. This enforces the entire plot so use the scattergeo trace type, and enables higher level geometries like `add_polygons()` to work.

**Usage**

```r
plot_geo(data = data.frame(), ...)
```

**Arguments**

- `data`  
  A data frame (optional).
- `...`  
  Arguments passed along to `plot_ly()`.

**Author(s)**

Carson Sievert

**See Also**

`plot_ly()`, `plot_mapbox()`, `ggplotly()`

**Examples**

```r
map_data("world", "canada") %>%
  group_by(group) %>%
  plot_geo(x = ~long, y = ~lat) %>%
  add_markers(size = I(1))
```

---

**plot_ly**  
*Initiate a plotly visualization*

**Description**

Transform data into a plotly visualization.

**Usage**

```r
plot_ly(data = data.frame(), ..., type = NULL, color, colors = NULL,
  alpha = 1, symbol, symbols = NULL, size, sizes = c(10, 100), linetype,
  linetypes = NULL, split, frame, width = NULL, height = NULL,
  source = "A")
```
Arguments

- **data**: A data frame (optional) or crosstalk::SharedData object.
- **type**: A character string describing the type of trace.
- **color**: A formula containing a name or expression. Values are scaled and mapped to color codes based on the value of `colors` and `alpha`. To avoid scaling, wrap with `I()`, and provide value(s) that can be converted to rgb color codes by `grDevices::col2rgb()`.
- **colors**: Either a colorbrewer2.org palette name (e.g. "YlOrRd" or "Blues"), or a vector of colors to interpolate in hexadecimal "#RRGGBB" format, or a color interpolation function like `colorRamp()`. A number between 0 and 1 specifying the alpha channel applied to color.
- **symbol**: A formula containing a name or expression. Values are scaled and mapped to symbols based on the value of `symbols`. To avoid scaling, wrap with `I()`, and provide valid `pch` values and/or valid plotly symbol(s) as a string.
- **symbols**: A character vector of symbol types. Either valid `pch` or plotly symbol codes may be supplied.
- **size**: A formula containing a name or expression yielding a numeric vector. Values are scaled according to the range specified in `sizes`.
- **sizes**: A numeric vector of length 2 used to scale sizes to pixels.
- **linetype**: A formula containing a name or expression. Values are scaled and mapped to linetypes based on the value of `linetypes`. To avoid scaling, wrap with `I()`. A character vector of line types. Either valid `par` (lty) or plotly dash codes may be supplied.
- **linetypes**: A character vector of line types. Either valid `par` (lty) or plotly dash codes may be supplied.
- **split**: A formula containing a name or expression. Similar to `group_by()`, but ensures at least one trace for each unique value. This replaces the functionality of the (now deprecated) `group` argument.
- **frame**: A formula containing a name or expression. The resulting value is used to split data into frames, and then animated.
- **width**: Width in pixels (optional, defaults to automatic sizing).
- **height**: Height in pixels (optional, defaults to automatic sizing).
- **source**: A character string of length 1. Match the value of this string with the source argument in `event_data()` to retrieve the event data corresponding to a specific plot (shiny apps can have multiple plots).

Details

There are a number of "visual properties" that aren’t included in the official Reference section (see below).

Author(s)

Carson Sievert
See Also

- For initializing a plotly-geo object: `plot_geo()`.
- For initializing a plotly-mapbox object: `plot_mapbox()`.
- For translating a ggplot2 object to a plotly object: `ggplotly()`.
- For modifying any plotly object: `layout()`, `add_trace()`, `style()`.

Examples

```r
## Not run:

# `plot_ly()` tries to create a sensible plot based on the information you
give it. If you don't provide a trace type, `plot_ly()` will infer one.
plot_ly(economics, x = ~pop)  
plot_ly(economics, x = ~date, y = ~pop)  
# `plot_ly()` doesn't require data frame(s), which allows one to take
# advantage of trace type(s) designed specifically for numeric matrices
plot_ly(z = ~volcano)  
plot_ly(z = ~volcano, type = "surface")

# `plotly` has a functional interface: every `plotly` function takes a `plotly`
# object as its first input argument and returns a modified `plotly` object
add_lines(plot_ly(economics, x = ~date, y = ~unemploy/pop))

# To make code more readable, `plotly` imports the pipe operator from `magrittr`
economics %>% plot_ly(x = ~date, y = ~unemploy/pop) %>% add_lines()

# Attributes defined via `plot_ly()` set 'global' attributes that
# are carried onto subsequent traces, but those may be over-written
plot_ly(economics, x = ~date, color = I("black")) %>%
  add_lines(y = ~uempmed) %>%
  add_lines(y = ~psavert, color = I("red"))

# Attributes are documented in the figure reference -> https://plot.ly/r/reference
# You might notice `plot_ly()` has named arguments that aren't in this figure
# reference. These arguments make it easier to map abstract data values to
# visual attributes.
p <- plot_ly(iris, x = ~Sepal.Width, y = ~Sepal.Length)
add_markers(p, color = ~Petal.Length, size = ~Petal.Length)
add_markers(p, color = ~Species)
add_markers(p, color = ~Species, colors = "Set1")
add_markers(p, symbol = ~Species)
add_paths(p, linetype = ~Species)

## End(Not run)
```
initiates a plotly-mapbox object

Description

Use this function instead of `plot_ly()` to initialize a plotly-mapbox object. This enforces the entire plot so use the scattermapbox trace type, and enables higher level geometries like `add_polygons()` to work.

Usage

```r
plot_mapbox(data = data.frame(), ...)
```

Arguments

- `data`: A data frame (optional).
- `...`: arguments passed along to `plot_ly()`. They should be valid scattermapbox attributes - [https://plot.ly/r/reference/#scattermapbox](https://plot.ly/r/reference/#scattermapbox). Note that x/y can also be used in place of lat/lon.

Author(s)

Carson Sievert

See Also

`plot_ly()`, `plot_geo()`, `ggplotly()`

Examples

```r
## Not run:

map_data("world", "canada") %>%
  group_by(group) %>%
  plot_mapbox(x = ~long, y = ~lat) %>%
  add_polygons() %>%
  layout(
    mapbox = list(
      center = list(lat = ~median(lat), lon = ~median(long))
    )
  )

## End(Not run)
```
**print.api**

Print method for a 'generic' API response

**Description**

Print method for a 'generic' API response

**Usage**

```r
## S3 method for class 'api'
print(x, ...)
```

**Arguments**

- `x` a list.
- `...` additional arguments (currently ignored)

---

**print.api_grid**

Print a plotly grid object

**Description**

Print a plotly grid object

**Usage**

```r
## S3 method for class 'api_grid'
print(x, ...)
```

**Arguments**

- `x` a plotly grid object
- `...` additional arguments (currently ignored)
Description

Print a plotly grid object

Usage

## S3 method for class 'api_grid_local'

print(x, ...)

Arguments

- **x**: a plotly grid object
- **...**: additional arguments (currently ignored)

Description

Print a plot on plotly’s platform

Usage

## S3 method for class 'api_plot'

print(x, ...)

Arguments

- **x**: a plotly figure object
- **...**: additional arguments (currently ignored)
rangeslider

Add a range slider to the x-axis

Description

Add a range slider to the x-axis

Usage

rangeslider(p, start = NULL, end = NULL, ...)

Arguments

p  
plotly object.
start  
a start date/value.
end  
an end date/value.
...  
these arguments are documented here [https://plot.ly/r/reference/#layout-xaxis-rangeslider](https://plot.ly/r/reference/#layout-xaxis-rangeslider)

Author(s)

Carson Sievert

Examples

```r
plot_ly(x = time(USAccDeaths), y = USAccDeaths) %>%
  add_lines() %>%
  rangeslider()

d <- tibble::tibble(
  time = seq(as.Date("2016-01-01"), as.Date("2016-08-31"), by = "days"),
  y = rnorm(seq_along(time))
)
plot_ly(d, x = ~time, y = ~y) %>%
  add_lines() %>%
  rangeslider(d$time[5], d$time[50])
```
raster2uri \hspace{1cm} \textit{Convert a raster object to a data URI}

\section*{Description}

Convenient embedding images via \texttt{layout()} \url{https://plot.ly/r/reference/#layout-images}.

\section*{Usage}

\begin{verbatim}
  raster2uri(r, ...)
\end{verbatim}

\section*{Arguments}

\begin{verbatim}
  r \hspace{0.5cm} \text{an object coercable to a raster object via \texttt{as.raster()}}
  ...
  \text{arguments passed onto \texttt{as.raster()}.}
\end{verbatim}

\section*{Author(s)}

Carson Sievert

\section*{Examples}

\begin{verbatim}
  # a red gradient (from \texttt{?as.raster})
  r <- as.raster(matrix(hcl(0, 80, seq(50, 80, 10)), nrow = 4, ncol = 5))
  plot(r)

  # embed the raster as an image
  plot_ly(x = 1, y = 1) %>%
    layout(
      images = list(list(
        source = raster2uri(r),
        xref = "paper",
        yref = "paper",
        x = 0, y = 0,
        sizex = 0.5, sizey = 0.5,
        xanchor = "left", yanchor = "bottom"
      )
    )
  )
\end{verbatim}
Description

By default, plotly.js' TypedArray polyfill is included as a dependency, so printing "just works" in any context. Many users won’t need this polyfill, so this function may be used to remove it and thus reduce the size of the page.

Usage

`remove_typedarray_polyfill(p)`

Arguments

- `p`: a plotly object

Details

The polyfill seems to be only relevant for those rendering plots via phantomjs and RStudio on some Windows platforms.

Examples

```r
defline{Not run:
pl <- plot_ly()
p2 <- remove_typedarray_polyfill(pl)
t1 <- tempfile(fileext = ".html")
htmlwidgets::saveWidget(pl, t1)
file.info(t1)$size
htmlwidgets::saveWidget(p2, t1)
file.info(t1)$size
}
```
showRGB

Usage

```r
showRGB = interactive(), ...)
```

Arguments

- `jsonedit` use `listviewer::jsonedit` to view the JSON?
- `...` other options passed onto `listviewer::jsonedit`

Examples

```r
s <- schema()

# retrieve acceptable 'layout.mapbox.style' values
if (!is.na(Sys.getenv('MAPBOX_TOKEN', NA))) {
  styles <- s$layout$layoutAttributes$mapbox$style$values
  subplot(
    plot_mapbox() %>% layout(mapbox = list(style = styles[3])),
    plot_mapbox() %>% layout(mapbox = list(style = styles[5]))
  )
}
```

---

showRGB View colors already formatted by `toRGB()`

Description

Useful for viewing colors after they've been converted to plotly.js' color format – "rgba(255, 255, 255, 1)"

Usage

```r
showRGB(x, ...)
```

Arguments

- `x` character string specifying color(s).
- `...` arguments passed along to `scales::show_col`.

Author(s)

Carson Sievert

Examples

```r
showRGB(toRGB(colors()), labels = FALSE)
```
Description

A sign up interface to plotly through the R Console.

Usage

```r
signup(username, email, save = TRUE)
```

Arguments

- `username` Desired username.
- `email` Desired email.
- `save` If request is successful, should the username & API key be automatically stored as an environment variable in a .Rprofile?

Value

- `api_key` key to use with the api
- `tmp_pw` temporary password to access your plotly account

References

https://plot.ly/rest/

Examples

```r
## Not run:
# You need a plotly username and API key to communicate with the plotly API.

# If you don't already have an API key, you can obtain one with a valid
# username and email via signup().
s <- signup('anna.lyst', 'anna.lyst@plot.ly')

# If you already have a username and API key, please create the following
# environment variables:
Sys.setenv("plotly_username" = "me")
Sys.setenv("plotly_api_key" = "mykey")
# You can also change the default domain if you have a plotly server.
Sys.setenv("plotly_domain" = "http://mydomain.com")

# If you want to automatically load these environment variables when you
# start R, you can put them inside your ~/.Rprofile
# (see help(.Rprofile) for more details)
```

## End(Not run)
### style

**Modify trace(s)**

Modify trace(s) of an existing plotly visualization. Useful when used in conjunction with `get_figure()`.

**Usage**

```r
style(p, ..., traces = NULL)
```

**Arguments**

- `p`: A plotly visualization.
- `...`: Visual properties.
- `traces`: numeric vector. Which traces should be modified? By default, attributes placed in `...` will be applied to every trace.

**Author(s)**

Carson Sievert

**See Also**

`api_download_plot()`

**Examples**

```r
p <- qplot(data = mtcars, wt, mpg, geom = c("point", "smooth"))
# keep the hover info for points, but remove it for the line/ribbon
style(p, hoverinfo = "none", traces = c(2, 3))
```

---

### subplot

**View multiple plots in a single view**

**Description**

View multiple plots in a single view

**Usage**

```r
subplot(..., nrows = 1, widths = NULL, heights = NULL, margin = 0.02,
        shareX = FALSE, shareY = FALSE, titleX = shareX, titleY = shareY,
        which_layout = "merge")
```
Arguments

... One of the following
  • any number of plotly/ggplot2 objects.
  • a list of plotly/ggplot2 objects.
  • a tibble with one list-column of plotly/ggplot2 objects.

nrows number of rows for laying out plots in a grid-like structure. Only used if no
domain is already specified.

widths relative width of each column on a 0-1 scale. By default all columns have an
equal relative width.

heights relative height of each row on a 0-1 scale. By default all rows have an equal
relative height.

margin either a single value or four values (all between 0 and 1). If four values are
provided, the first is used as the left margin, the second is used as the right
margin, the third is used as the top margin, and the fourth is used as the bottom
margin. If a single value is provided, it will be used as all four margins.

shareX should the x-axis be shared amongst the subplots?

shareY should the y-axis be shared amongst the subplots?

titleX should x-axis titles be retained?

titleY should y-axis titles be retained?

which_layout adopt the layout of which plot? If the default value of "merge" is used, layout
options found later in the sequence of plots will override options found earlier
in the sequence. This argument also accepts a numeric vector specifying which
plots to consider when merging.

Value

A plotly object

Author(s)

Carson Sievert

Examples

# pass any number of plotly objects to subplot()
p1 <- plot_ly(economics, x ~date, y ~uempmed)
p2 <- plot_ly(economics, x ~date, y ~unemploy)
subplot(p1, p2, p1, p2, nrows = 2, margin = 0.05)

# anchor multiple traces on the same legend entry
p1 <- add_lines(p1, color = 1("black"), name = "1st", legendgroup = "1st")
p2 <- add_lines(p2, color = 1("red"), name = "2nd", legendgroup = "2nd")

subplot(
  p1, style(p1, showlegend = FALSE),
```r
toRGB

p2, style(p2, showlegend = FALSE),
nrows = 2, margin = 0.05
)

# or pass a list
economics_long %>%
split(.$variable) %>%
lapply(function(d) plot_ly(d, x = ~date, y = ~value)) %>%
subplot(nrows = NROW(.), shareX = TRUE)

# or pass a tibble with a list-column of plotly objects
economics_long %>%
group_by(variable) %>%
do(p = plot_ly(. , x = ~date, y = ~value)) %>%
subplot(nrows = NROW(.), shareX = TRUE)

# learn more at https://cpsievert.github.io/plotly_book/subplot.html
```

---

toRGB

**Convert R colours to RGBA hexadecimal colour values**

**Description**

Convert R colours to RGBA hexadecimal colour values

**Usage**

```r
toRGB(x, alpha = 1)
```

**Arguments**

- **x**: see the col argument in col2rgb for valid specifications
- **alpha**: alpha channel on 0-1 scale

**Value**

hexadecimal colour value (if is.na(x), return "transparent" for compatibility with Plotly)

**See Also**

```r
showRGB()
```
Examples

toRGB("steelblue")
# [1] "rgba(70,130,180,1)"

m <- list(
  color = toRGB("red"),
  line = list(
    color = toRGB("black"),
    width = 19
  )
)

plot_ly(x = 1, y = 1, marker = m)

---

toWebGL  

Convert trace types to WebGL

Description

Convert trace types to WebGL.

Usage

toWebGL(p)

Arguments

p  
  a plotly or ggplot object.

Examples

# currently no bargl trace type
toWebGL(qplot(1:10))
toWebGL(qplot(1:10, 1:10))
to_basic

Convert a geom to a "basic" geom.

Description

This function makes it possible to convert ggplot2 geoms that are not included with ggplot2 itself. Users shouldn’t need to use this function. It exists purely to allow other package authors to write their own conversion method(s).

Usage

to_basic(data, prestats_data, layout, params, p, ...)

Arguments

data the data returned by ggplot2::ggplot_build().
prestats_data the data before statistics are computed.
layout the panel layout.
params parameters for the geom, statistic, and 'constant' aesthetics
p a ggplot2 object (the conversion may depend on scales, for instance).
... currently ignored

wind

Wind data

Description

Description TBD.

Usage

wind

Format

A data frame with three variables: r, t, nms.
Index

*Topic **datasets**
   - hobbs, 28
   - mic, 31
   - wind, 56

add_annotations, 3
add_area (add_trace), 5
add_bars (add_trace), 5
add_boxplot (add_trace), 5
add_choropleth (add_trace), 5
add_contour (add_trace), 5
add_data, 4
add_fun, 4
add_heatmap (add_trace), 5
add_histogram (add_trace), 5
add_histogram2d (add_trace), 5
add_histogram2dcontour (add_trace), 5
add_lines (add_trace), 5
add_markers (add_trace), 5
add_mesh (add_trace), 5
add_paths (add_trace), 5
add_pie (add_trace), 5
add_polygons (add_trace), 5
add_polygons(), 41, 44
add_ribbons (add_trace), 5
add_scattergeo (add_trace), 5
add_segments (add_trace), 5
add_surface (add_trace), 5
add_text (add_trace), 5
add_trace, 5
add_trace(), 43
animation (animation_opts), 8
animation_button (animation_opts), 8
animation_opts, 8
animation_opts(), 9
animation_slider (animation_opts), 8
api (api_create), 10
api_create, 10
api_create(), 39
api_download_grid (api_create), 10
api_download_plot (api_create), 10
api_download_plot(), 20, 52
arrange_plotly (plotly_data), 34
as.raster(), 48
as.widget, 13
as_widget, 13
attrs_selected, 14
attrs_selected(), 27
bbox, 14
colorbar, 15
cfg, 16
crosstalk::SharedData, 7, 42
data.frame, 11
distinct_plotly (plotly_data), 34
d_plotly (plotly_data), 34
embed_notebook, 17
event_data, 17
event_data(), 21, 22, 42
export, 18
filter_plotly (plotly_data), 34
geom2trace, 19
gf, 20
gf(), 52
gl, 20
gl(), 20
gl(), 21
gg2list, 21
ggplot2::ggplot(), 22
ggplot2::last_plot(), 30
ggplotly, 22
ggplotly(), 8, 40, 41, 43, 44
grDevices::col2rgb(), 42
group2NA, 23
group_by, 42
group_by_plotly (plotly_data), 34
groups.plotly(plotly_data), 34
hide_colorbar, 25
hide_colorbar(), 25, 26
hide_guides, 25
hide_legend, 26
hide_legend(), 25
highlight, 26
hobbs, 28
httr::VERB(), 11
I(), 42
jsonlite::toJSON(), 11
knit_print.api_grid, 29
knit_print.api_grid_local, 29
knit_print.api_plot, 30
last_plot, 30
layout, 31
layout(), 43, 48
mic, 31
mutate_.plotly(plotly_data), 34
offline, 32
par, 42
pch, 42
pch(), 42
plot_dendro, 40
plot_geo, 41
plot_geo(), 43, 44
plot_ly, 41
plot_ly(), 3, 7, 8, 17, 23, 31, 37, 40, 41, 44
plot_mapbox, 44
plot_mapbox(), 40, 41, 43
plotly-shiny, 32
plotly_build, 34
plotly_data, 34
plotly_empty, 37
plotly_example, 37
plotly_IMAGE, 38
plotly_json, 38
plotly_POST, 39
plotlyOutput(plotly-shiny), 32
plotlyProxy, 33
plotlyProxyInvoke(plotlyProxy), 33
print.api, 45
print.api_grid, 45
print.api_grid_local, 46
print.api_plot, 46
rangeslider, 47
raster2uri, 48
removeTypedarrayPolyfill, 49
rename_.plotly(plotly_data), 34
renderPlotly(plotly-shiny), 32
schema, 49
select_.plotly(plotly_data), 34
showRGB, 50
showRGB(), 54
signup, 51
signup(), 12, 40
slice_.plotly(plotly_data), 34
style, 52
style(), 43
subplot, 52
subplot(), 37, 40
summarise_.plotly(plotly_data), 34
to_basic, 56
toRGB, 54
toRGB(), 27
toWebGL, 55
transmute_.plotly(plotly_data), 34
ungroup.plotly(plotly_data), 34
wind, 56