Package 'plotly'

April 10, 2019

Title Create Interactive Web Graphics via 'plotly.js'

Version 4.9.0

License MIT + file LICENSE

Description

Create interactive web graphics from 'ggplot2' graphs and/or a custom interface to the (MIT-licensed) JavaScript library 'plotly.js' inspired by the grammar of graphics.

URL https://plotly-r.com, https://github.com/ropensci/plotly#readme,

https://plot.ly/r

BugReports https://github.com/ropensci/plotly/issues

Depends R (>= 3.2.0), ggplot2 (>= 3.0.0)

Imports tools, scales, httr, jsonlite (>= 1.6), magrittr, digest, viridisLite, base64enc, htmltools (>= 0.3.6), htmlwidgets (>= 1.3), tidyr, hexbin, RColorBrewer, dplyr, tibble, lazyeval (>= 0.2.0), rlang, crosstalk, purrr, data.table, promises

Suggests MASS, maps, ggthemes, GGally, testthat, knitr, devtools, shiny (>= 1.1.0), shinytest (>= 1.3.0), curl, rmarkdown, vdiffr, Cairo, broom, webshot, listviewer, dendextend, sf, maptools, rgeos, png, IRdisplay, processx, plotlyGeoAssets, forcats

LazyData true

RoxygenNote 6.1.1

Encoding UTF-8

NeedsCompilation no

Author Carson Sievert [aut, cre] (<https://orcid.org/0000-0002-4958-2844>),
Chris Parmer [aut],
Toby Hocking [aut],
Scott Chamberlain [aut],
Karthik Ram [aut],
Marianne Corvellec [aut] (<https://orcid.org/0000-0002-1994-3581>),
Pedro Despouy [aut],
Plotly Technologies Inc. [cph]

R **topics documented:**

---

add_annotations   *Add an annotation(s) to a plot*

Description

Add an annotation(s) to a plot

Usage

```
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

data	a data frame.
inherit	inherit attributes from [`plot_ly()`](#)?

Author(s)

Carson Sievert

---

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.

---

add_trace *Add trace(s) to a plotly visualization*

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_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_sf(p, ..., x = ~x, y = ~y, data = NULL, inherit = TRUE)

add_table(p, ..., rownames = TRUE, 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
...	Arguments (i.e., attributes) passed along to the trace type. See [schema()](#) for a list of acceptable attributes for a given trace type (by going to traces -> type -> attributes). Note that attributes provided at this level may override other arguments (e.g. plot_ly(x = 1:10, y = 1:10, color = I("red"), marker = list(color = "blue"))).
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")
rownames	whether or not to display the rownames of data.
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://plotly-r.com/overview.html

https://plot.ly/r

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

See Also

plot_ly()

Examples

```
# the `plot_ly()` function initiates an object, and if no trace type
# is specified, it sets a sensible default
p <- plot_ly(economics, x = ~date, y = ~uempmed)
p

# some `add_*()` functions are a specific case of a trace type
# for example, `add_markers()` is a scatter trace with mode of markers
add_markers(p)

# scatter trace with mode of text
add_text(p, text = "%")

# scatter trace with mode of lines
add_paths(p)

# like `add_paths()`, but ensures points are connected according to `x`
add_lines(p)

# if you prefer to work with plotly.js more directly, can always
# use `add_trace()` and specify the type yourself
add_trace(p, type = "scatter", mode = "markers+lines")

# mappings provided to `plot_ly()` are "global", but can be overwritten
plot_ly(economics, x = ~date, y = ~uempmed, color = I("red"), showlegend = FALSE) %>%
  add_lines() %>%
  add_markers(color = ~pop)

# a number of `add_*()` functions are special cases of the scatter trace
plot_ly(economics, x = ~date) %>%
  add_ribbons(ymin = ~pce - 1e3, ymax = ~pce + 1e3)

# use `group_by()` (or `group2NA()`) to apply visual mapping
# once per group (e.g. one line per group)
txhousing %>%
  group_by(city) %>%
```

```
  plot_ly(x = ~date, y = ~median) %>%
  add_lines(color = I("black"))

## Not run:
# use `add_sf()` or `add_polygons()` to create geo-spatial maps
# http://blog.cpsievert.me/2018/03/30/visualizing-geo-spatial-data-with-sf-and-plotly/
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  plot_ly() %>% add_sf(data = nc)
}

# univariate summary statistics
plot_ly(mtcars, x = ~factor(vs), y = ~mpg) %>%
  add_boxplot()
plot_ly(mtcars, x = ~factor(vs), y = ~mpg) %>%
  add_trace(type = "violin")

# `add_histogram()` does binning for you...
mtcars %>%
  plot_ly(x = ~factor(vs)) %>%
  add_histogram()

# ...but you can 'pre-compute' bar heights in R
mtcars %>%
  dplyr::count(vs) %>%
  plot_ly(x = ~vs, y = ~n) %>%
  add_bars()

# the 2d analogy of add_histogram() is add_histogram2d()/add_histogram2dcontour()
library(MASS)
(p <- plot_ly(geyser, x = ~waiting, y = ~duration))
add_histogram2d(p)
add_histogram2dcontour(p)

# the 2d analogy of add_bars() is add_heatmap()/add_contour()
# (i.e., bin counts must be pre-specified)
den <- kde2d(geyser$waiting, geyser$duration)
p <- plot_ly(x = den$x, y = den$y, z = den$z)
add_heatmap(p)
add_contour(p)

# `add_table()` makes it easy to map a data frame to the table trace type
plot_ly(economics) %>%
  add_table()

# pie charts!
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")

data(wind)
plot_ly(wind, r = ~r, t = ~t) %>%
```

```
  add_area(color = ~nms) %>%
  layout(radialaxis = list(ticksuffix = "%"), orientation = 270)

# -----------------------------------------------------------
# 3D chart types
# -----------------------------------------------------------
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()

## End(Not run)
```

---

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

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

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

animation_button(p, ..., label)
```

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
label	a character string used for the animation button's label

Author(s)

Carson Sievert

Examples

```
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)
)
plot_ly(df) %>%
  add_markers(x = 1.5, y = 1.5) %>%
  add_markers(x = ~x, y = ~y, frame = ~z)

# it's a good idea to remove smooth transitions when there is
# no relationship between objects in each view
plot_ly(mtcars, x = ~wt, y = ~mpg, frame = ~cyl) %>%
  animation_opts(transition = 0)

# works the same way with ggplotly
if (interactive()) {
  p <- ggplot(txhousing, aes(month, median)) +
    geom_line(aes(group = year), alpha = 0.3) +
    geom_smooth() +
    geom_line(aes(frame = year, ids = month), color = "red") +
    facet_wrap(~ city)

  ggplotly(p, width = 1200, height = 900) %>%
    animation_opts(1000)
}


#' # for more, see https://cpsievert.github.io/plotly_book/key-frame-animations.html
```

---

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

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

## S3 method for class 'plotly'
api_create(x = last_plot(), filename = NULL,
  fileopt = "overwrite", sharing = "public", ...)

## S3 method for class 'ggplot'
api_create(x = last_plot(), filename = NULL,
  fileopt = "overwrite", sharing = "public", ...)

## S3 method for class 'data.frame'
api_create(x, filename = NULL,
  fileopt = "overwrite", sharing = "public", ...)

api_download_plot(id, username)

api_download_grid(id, username)

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

Arguments

x	An R object to 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(). For api_create(), 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
api("files/cpsievert:14681")

# change the filename https://api.plot.ly/v2/files#update
# (note: this won't work unless you have proper credentials to the relevant account)
api("files/cpsievert:14681", "PATCH", list(filename = "toy file"))

# Copy a file https://api.plot.ly/v2/files#lookup
api("files/cpsievert:14681/copy", "POST")

# Create a folder https://api.plot.ly/v2/folders#create
api("folders", "POST", list(path = "/starts/at/root/and/ends/here"))


## End(Not run)
```

---

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

```
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

```
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

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

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

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

Arguments

p	a plotly object
...	arguments are documented here https://plot.ly/r/reference/#scatter-marker-colorbar.
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

```
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

```
config(p, ..., cloud = FALSE, showSendToCloud = cloud, locale = NULL,
  mathjax = NULL)
```

Arguments

p	a plotly object
...	these arguments are documented at `https://github.com/plotly/plotly.js/blob/master/src/plot_api/plot_config.js`
cloud	deprecated. Use showSendToCloud instead.
showSendToCloud	
	include the send data to cloud button?
locale	locale to use. See here for more info.
mathjax	add MathJax rendering support. If "cdn", mathjax is loaded externally (meaning an internet connection is needed for TeX rendering). If "local", the PLOTLY_MATHJAX_PATH environment variable must be set to the location (a local file path) of MathJax. IMPORTANT: **plotly** uses SVG-based mathjax rendering which doesn't play nicely with HTML-based rendering (e.g., **rmarkdown** documents and **shiny** apps). To leverage both types of rendering, you must <iframe> your plotly graph(s) into the larger document (see here for an **rmarkdown** example and here for a **shiny** example).

Author(s)

Carson Sievert

Examples

```
# remove the plotly logo and collaborate button from modebar
config(plot_ly(), displaylogo = FALSE, collaborate = FALSE)

# enable mathjax
# see more examples at https://plot.ly/r/LaTeX/
plot_ly(x = c(1, 2, 3, 4), y = c(1, 4, 9, 16)) %>%
  layout(title = TeX("\\text{Some mathjax: }\\alpha+\\beta x")) %>%
  config(mathjax = "cdn")

# change the language used to render date axes and on-graph text
# (e.g., modebar buttons)
today <- Sys.Date()
x <- seq.Date(today, today + 360, by = "day")
p <- plot_ly(x = x, y = rnorm(length(x))) %>%
  add_lines()

# japanese
config(p, locale = "ja")
# german
config(p, locale = "de")
# spanish
config(p, locale = "es")
# chinese
config(p, locale = "zh-CN")
```

---

embed_notebook   *Embed a plot as an iframe into a Jupyter Notebook*

Description

Embed a plot as an iframe into a Jupyter Notebook

Usage

```
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

```
event_data(event = c("plotly_hover", "plotly_unhover", "plotly_click",
  "plotly_doubleclick", "plotly_selected", "plotly_selecting",
  "plotly_brushed", "plotly_brushing", "plotly_deselect",
  "plotly_relayout", "plotly_restyle", "plotly_legendclick",
  "plotly_legenddoubleclick", "plotly_clickannotation",
  "plotly_afterplot"), source = "A",
  session = shiny::getDefaultReactiveDomain(), priority = c("input",
  "event"))
```

Arguments

event	The type of plotly event. All supported events are listed in the function signature above (i.e., the usage section).
source	a character string of length 1. Match the value of this string with the source argument in plot_ly() (or ggplotly()) to respond to events emitted from that specific plot.
session	a shiny session object (the default should almost always be used).
priority	the priority of the corresponding shiny input value. If equal to "event", then event_data() always triggers re-execution, instead of re-executing only when the relevant shiny input value changes (the default).

Author(s)

Carson Sievert

References

- https://plotly-r.com/linking-views-with-shiny.html#shiny-plotly-inputs
- https://plot.ly/javascript/plotlyjs-function-reference/

See Also

event_register, event_unregister

Examples

```
## Not run:
plotly_example("shiny", "event_data")

## End(Not run)
```

---

event_register	*Register a shiny input value*

Description

Register a shiny input value

Usage

```
event_register(p, event = NULL)
```

Arguments

p	a plotly object.
event	The type of plotly event. All supported events are listed in the function signature above (i.e., the usage section).

Author(s)

Carson Sievert

See Also

[event_data](#)

---

event_unregister   *Un-register a shiny input value*

Description

Un-register a shiny input value

Usage

```
event_unregister(p, event = NULL)
```

Arguments

p	a plotly object.
event	The type of plotly event. All supported events are listed in the function signature above (i.e., the usage section).

Author(s)

Carson Sievert

See Also

[event_data](#)

---

export   *Export a plotly graph to a static file*

Description

This function is in the process of being deprecated (use [orca](#) instead).

Usage

```
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`.
...	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

---

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

---

get_figure   *Request a figure object*

Description

Deprecated: see api_download_plot().

Usage

```
get_figure(username, id)
```

Arguments

username   corresponding username for the figure.

id   of the Plotly figure.

---

gg2list   *Convert a ggplot to a list.*

Description

Convert a ggplot to a list.

Usage

```
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

```
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. For examples on how to specify the output container's `height/width` in a shiny app, see `plotly_example("shiny", "ggplotly_sizing")`.

Author(s)

Carson Sievert

References

https://plot.ly/ggplot2

See Also

plot_ly()

Examples

```
## Not run:
# simple example
ggiris <- qplot(Petal.Width, Sepal.Length, data = iris, color = Species)
ggplotly(ggiris)

data(canada.cities, package = "maps")
viz <- ggplot(canada.cities, aes(long, lat)) +
  borders(regions = "canada") +
  coord_equal() +
  geom_point(aes(text = name, size = pop), colour = "red", alpha = 1/2)
ggplotly(viz, tooltip = c("text", "size"))

# linked scatterplot brushing
d <- highlight_key(mtcars)
qplot(data = d, x = mpg, y = wt) %>%
  subplot(qplot(data = d, x = mpg, y = vs)) %>%
  layout(title = "Click and drag to select points") %>%
  highlight("plotly_selected")


# more brushing (i.e. highlighting) examples
demo("crosstalk-highlight-ggplotly", package = "plotly")

# client-side linked brushing in a scatterplot matrix
highlight_key(iris) %>%
  GGally::ggpairs(aes(colour = Species), columns = 1:4) %>%
  ggplotly(tooltip = c("x", "y", "colour")) %>%
  highlight("plotly_selected")

## End(Not run)
```

---

group2NA *Separate groups with missing values*

Description

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

```
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

```
# 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)*

Description

Hide color bar(s)

Usage

```
hide_colorbar(p)
```

Arguments

p   a plotly object.

See Also

[hide_legend()](#)

Examples

```
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

```
hide_guides(p)
```

Arguments

p   a plotly object.

See Also

[hide_legend()](#), [hide_colorbar()](#)

---

hide_legend   *Hide legend*

Description

Hide legend

Usage

```
hide_legend(p)
```

Arguments

p   a plotly object.

See Also

[hide_colorbar()](#)

Examples

```
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

```
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(), debounce = 0, ...)
```

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.

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)? We often refer to the default (FALSE) as a 'transient' selection mode; which is recommended, because one may switch from 'transient' to 'persistent' selection by holding the shift key.
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()](#).
debounce	amount of time to wait before firing an event (in milliseconds). The default of 0 means do not debounce at all. Debouncing is mainly useful when on = "plotly_hover" to avoid firing too many events when users clickly move the mouse over relevant graphical marks.
...	currently not supported.

Author(s)

Carson Sievert

References

<https://plotly-r.com/client-side-linking.html>

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")`

d <- highlight_key(txhousing, ~city)
p <- ggplot(d, aes(date, median, group = city)) + geom_line()
gg <- ggplotly(p, tooltip = "city")
highlight(gg, dynamic = TRUE)

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

---

highlight_key	*Highlight/query data based on primary key*

Description

This function simply creates an object of class [crosstalk::SharedData](#). The reason it exists is to make it easier to teach others how to leverage it's functionality in plotly. It also makes it more discoverable if one is already aware of [highlight](#).

Usage

```
highlight_key(...)
```

Arguments

...   arguments passed to crosstalk::SharedData$new()

Value

An object of class [crosstalk::SharedData](#)

Author(s)

Carson Sievert

See Also

[highlight](#)

---

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

https://github.com/yihui/knitr/blob/master/vignettes/knit_print.Rmd

---

```
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

https://github.com/yihui/knitr/blob/master/vignettes/knit_print.Rmd

---

```
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

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

Arguments

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

References

https://github.com/yihui/knitr/blob/master/vignettes/knit_print.Rmd

---

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

Description

Retrieve the last plot to be modified or created.

Usage

```
last_plot()
```

See Also

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.

---

offline   *Plotly Offline*

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

---

orca   *Static image exporting*

Description

Export plotly objects to static images (e.g., pdf, png, jpeg, svg, etc) via the orca command-line utility.

Usage

```
orca(p, file = "plot.png", format = tools::file_ext(file),
  scale = NULL, width = NULL, height = NULL, mathjax = FALSE,
  parallel_limit = NULL, verbose = FALSE, debug = FALSE,
  safe = FALSE, more_args = NULL, ...)

orca_serve(port = 5151, mathjax = FALSE, safe = FALSE,
  request_limit = NULL, keep_alive = TRUE, window_max_number = NULL,
  quiet = FALSE, debug = FALSE, more_args = NULL, ...)
```

Arguments

p	a plotly object.
file	output filename.
format	the output format (png, jpeg, webp, svg, pdf, eps).
scale	Sets the image scale. Applies to all output images.
width	Sets the image width. If not set, defaults to `layout.width` value. Applies to all output images.
height	Sets the image height. If not set, defaults to `layout.height` value. Applies to all output images.
mathjax	whether or not to include MathJax (required to render TeX). If TRUE, the PLOTLY_MATHJAX_PATH environment variable must be set and point to the location of MathJax (this variable is also used to render TeX in interactive graphs, see config).
parallel_limit	Sets the limit of parallel tasks run.
verbose	Turn on verbose logging on stdout.
debug	Starts app in debug mode and turn on verbose logs on stdout.
safe	Turns on safe mode: where figures likely to make browser window hang during image generating are skipped.
more_args	additional arguments to pass along to system command. This is useful for specifying display and/or electron options, such as `--enable-webgl` or `--disable-gpu`.
...	for `orca()`, additional arguments passed along to `processx::run`. For `orca_serve()`, additional arguments passed along to `processx::process`.
port	Sets the server's port number.
request_limit	Sets a request limit that makes orca exit when reached.

`keep_alive`	Turn on keep alive mode where orca will (try to) relaunch server if process unexpectedly exits.
`window_max_number`	
	Sets maximum number of browser windows the server can keep open at a given time.
`quiet`	Suppress all logging info.

Details

The `orca()` function is designed for exporting one plotly graph whereas `orca_serve()` is meant for exporting many graphs at once. The former starts and stops an external (nodejs) process everytime it is called whereas the latter starts up a process when called, then returns an `export()` method for exporting graphs as well as a `close()` method for stopping the external (background) process.

Methods

The `orca_serve()` function returns an object with two methods:

`export(p, file = "plot.png", format = tools::file_ext(file), scale = NULL, width = NULL, height = N`
> Export a static image of a plotly graph. Arguments found here are the same as those found in `orca()`

`close()` Close down the orca server and kill the underlying node process.

Fields

The `orca_serve()` function returns an object with two fields:

`port` The port number that the server is listening to.

`process` An R6 class for controlling and querying the underlying node process.

Author(s)

Carson Sievert

Examples

```
## Not run:
# NOTE: in a headless environment, you may need to set `more_args="--enable-webgl"`
# to export webgl correctly
p <- plot_ly(z = ~volcano) %>% add_surface()
orca(p, "surface-plot.svg")

#' # launch the server
server <- orca_serve()

# export as many graphs as you'd like
server$export(qplot(1:10), "test1.pdf")
server$export(plot_ly(x = 1:10, y = 1:10), "test2.pdf")
```

```
# the underlying process is exposed as a field, so you
# have full control over the external process
server$process$is_alive()

# convenience method for closing down the server
server$close()

# remove the exported files from disk
unlink("test1.pdf")
unlink("test2.pdf")

## End(Not run)
```

---

partial_bundle   *Use a partial bundle of plotly.js*

Description

Leveraging plotly.js' partial bundles can lead to smaller file sizes and faster rendering. The full list
of available bundles, and the trace types that they support, are available here

Usage

```
partial_bundle(p, type = "auto", local = TRUE, minified = TRUE)
```

Arguments

p	a plotly object.
type	name of the (partial) bundle. The default, 'auto', attempts to find the smallest single bundle that can render p. If no single partial bundle can render p, then the full bundle is used.
local	whether or not to download the partial bundle so that it can be viewed later without an internet connection.
minified	whether or not to use a minified js file (non-minified file can be useful for debugging plotly.js)

Details

WARNING: use this function with caution when rendering multiple plotly graphs on a single website. That's because, if multiple plotly.js bundles are used, the most recent bundle will override the
other bundles. See the examples section for an example.

Author(s)

Carson Sievert

Examples

```
# ----------------------------------------------------------------------
# This function is always safe to use when rendering a single
# plotly graph. In this case, we get a 3x file reduction.
# ----------------------------------------------------------------------

library(plotly)
p <- plot_ly(x = 1:10, y = 1:10) %>% add_markers()
save_widget <- function(p, f) {
  owd <- setwd(dirname(f))
  on.exit(setwd(owd))
  htmlwidgets::saveWidget(p, f)
  mb <- round(file.info(f)$size / 1e6, 3)
  message("File is: ", mb," MB")
}
f1 <- tempfile(fileext = ".html")
f2 <- tempfile(fileext = ".html")
save_widget(p, f1)
save_widget(partial_bundle(p), f2)


# ----------------------------------------------------------------------
# But, since plotly.js bundles override one another,
# be careful when putting multiple graphs in a larger document!
# Note how the surface (part of the gl3d bundle) renders, but the
# heatmap (part of the cartesian bundle) doesn't...
# ----------------------------------------------------------------------

## Not run:
library(htmltools)
p1 <- plot_ly(z = ~volcano) %>%
  add_heatmap() %>%
  partial_bundle()
p2 <- plot_ly(z = ~volcano) %>%
  add_surface() %>%
  partial_bundle()
browsable(tagList(p1, p2))

## End(Not run)
```

| 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)
```

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. Note that, for height, using "auto" or "100 height is computed with HTML/CSS.
inline	use an inline (span()) or block container (div()) for the output
expr	An expression that generates a plotly
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.

---

plotlyProxy   *Modify a plotly object inside a shiny app*

Description

Modify a plotly object inside a shiny app

Usage

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

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

Arguments

outputId	single-element character vector indicating the output ID map to modify (if invoked from a Shiny module, the namespace will be added automatically)
session	the Shiny session object to which the map belongs; usually the default value will suffice.
deferUntilFlush	
	indicates whether actions performed against this instance should be carried out right away, or whether they should be held until after the next time all of the outputs are updated.
p	a plotly proxy object (created with plotlyProxy)
method	a plotlyjs method to invoke. For a list of options, visit the plotlyjs function reference
...	unnamed arguments passed onto the plotly.js method

Examples

```
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

```
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).

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	a plotly visualization
.dots	Used to work around non-standard evaluation. See vignette("nse") for details
add	By default, when add = FALSE, group_by will override existing groups. To instead add to the existing groups, use add = TRUE

Examples

```
# use group_by() to define groups of visual markings
p <- txhousing %>%
  group_by(city) %>%
  plot_ly(x = ~date, y = ~sales)
p

# 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(p)
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"]], nrows = 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()
p %>% slice(1) %>% plotly_data()
p %>% slice(1) %>% add_markers(name = "first observation")
p %>% filter(cyl == 4) %>% plotly_data()
p %>% 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, edit = TRUE,
  ...)
```

Arguments

type	the type of example
name	the name of the example (valid names depend on type).
edit	whether to open the relevant source files using [file.edit](#). Only relevant if type is "shiny" or "rmd".
...	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

```
## 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(), pretty = TRUE,
  ...)
```

Arguments

p	a plotly or ggplot object.
jsonedit	use listviewer::jsonedit to view the JSON?
pretty	adds indentation whitespace to JSON output. Can be TRUE/FALSE or a number specifying the number of spaces to indent. See jsonlite::prettify.
...	other options passed onto listviewer::jsonedit

Examples

```
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

See Also

[plot_ly()](), [signup()]()

---

plot_dendro   *Plot an interactive dendrogram*

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

```
plot_geo(data = data.frame(), ..., offline = FALSE)
```

Arguments

data	A data frame (optional).
...	arguments passed along to `plot_ly()`.
offline	whether or not to include geo assets so that the map can be viewed with or without an internet connection. The plotlyGeoAssets package is required for this functionality.

Author(s)

Carson Sievert

See Also

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

Examples

```
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

This function maps R objects to plotly.js, an (MIT licensed) web-based interactive charting library. It provides abstractions for doing common things (e.g. mapping data values to fill colors (via color) or creating animations (via frame)) and sets some different defaults to make the interface feel more 'R-like' (i.e., closer to plot() and ggplot2::qplot()).

Usage

```
plot_ly(data = data.frame(), ..., type = NULL, name, color,
  colors = NULL, alpha = NULL, stroke, strokes = NULL,
  alpha_stroke = 1, size, sizes = c(10, 100), span, spans = c(1, 20),
  symbol, symbols = NULL, linetype, linetypes = NULL, split, frame,
  width = NULL, height = NULL, source = "A")
```

Arguments

data	A data frame (optional) or crosstalk::SharedData object.
...	Arguments (i.e., attributes) passed along to the trace type. See schema() for a list of acceptable attributes for a given trace type (by going to traces -> type -> attributes). Note that attributes provided at this level may override other arguments (e.g. plot_ly(x = 1:10, y = 1:10, color = I("red"), marker = list(color = "blue"))).
type	A character string specifying the trace type (e.g. "scatter", "bar", "box", etc). If specified, it *always* creates a trace, otherwise
name	Values mapped to the trace's name attribute. Since a trace can only have one name, this argument acts very much like split in that it creates one trace for every unique value.
color	Values mapped to relevant 'fill-color' attribute(s) (e.g. fillcolor, marker.color, textfont.color, etc.). The mapping from data values to color codes may be controlled using colors and alpha, or avoided altogether via [I()](I()) (e.g., color = I("red")). Any color understood by grDevices::col2rgb() may be used in this way.
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().
alpha	A number between 0 and 1 specifying the alpha channel applied to color. Defaults to 0.5 when mapping to fillcolor and 1 otherwise.
stroke	Similar to color, but values are mapped to relevant 'stroke-color' attribute(s) (e.g., marker.line.color and line.color for filled polygons). If not specified, stroke inherits from color.
strokes	Similar to colors, but controls the stroke mapping.
alpha_stroke	Similar to alpha, but applied to stroke.

size	(Numeric) values mapped to relevant 'fill-size' attribute(s) (e.g., marker.size, textfont.size, and error_x.width). The mapping from data values to symbols may be controlled using `sizes`, or avoided altogether via `I()` (e.g., size = I(30)).
sizes	A numeric vector of length 2 used to scale `size` to pixels.
span	(Numeric) values mapped to relevant 'stroke-size' attribute(s) (e.g., marker.line.width, line.width for filled polygons, and error_x.thickness) The mapping from data values to symbols may be controlled using `spans`, or avoided altogether via `I()` (e.g., span = I(30)).
spans	A numeric vector of length 2 used to scale `span` to pixels.
symbol	(Discrete) values mapped to marker.symbol. The mapping from data values to symbols may be controlled using `symbols`, or avoided altogether via `I()` (e.g., symbol = I("pentagon")). Any pch value or symbol name may be used in this way.
symbols	A character vector of pch values or symbol names.
linetype	(Discrete) values mapped to line.dash. The mapping from data values to symbols may be controlled using `linetypes`, or avoided altogether via `I()` (e.g., linetype = I("dash")). Any lty (see par) value or dash name may be used in this way.
linetypes	A character vector of lty values or dash names
split	(Discrete) values used to create multiple traces (one trace per value).
frame	(Discrete) values used to create animation frames.
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

Unless `type` is specified, this function just initiates a plotly object with 'global' attributes that are passed onto downstream uses of add_trace() (or similar). A formula must always be used when referencing column name(s) in data (e.g. `plot_ly(mtcars, x = ~wt)`). Formulas are optional when supplying values directly, but they do help inform default axis/scale titles (e.g., `plot_ly(x = mtcars$wt)` vs `plot_ly(x = ~mtcars$wt)`)

Author(s)

Carson Sievert

References

https://plotly-r.com/overview.html

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()`
- For linked brushing: `highlight()`
- For arranging multiple plots: `subplot()`, `crosstalk::bscols()`
- For inspecting plotly objects: `plotly_json()`
- For quick, accurate, and searchable plotly.js reference: `schema()`

Examples

```
## 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 it's 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)
```

---

plot_mapbox *Initiate 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

```
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. 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

```
## Not run:

plot_mapbox(res_mn)
plot_mapbox(res_mn, color = ~INDRESNAME)

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

```
## 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

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

Arguments

x   a plotly grid object

...   additional arguments (currently ignored)

---

print.api_grid_local   *Print a plotly grid object*

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)

---

print.api_plot   *Print a plot on plotly's platform*

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

Author(s)

Carson Sievert

Examples

```
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   *Convert a raster object to a data URI*

Description

Convenient embedding images via `layout()` https://plot.ly/r/reference/#layout-images.

Usage

```
raster2uri(r, ...)
```

Arguments

r   an object coercable to a raster object via `as.raster()`

...   arguments passed onto `as.raster()`.

Author(s)

Carson Sievert

Examples

```
# a red gradient (from ?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"
  ))
  )
```

---

remove_typedarray_polyfill

Remove TypedArray polyfill

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

```
## Not run:
p1 <- plot_ly()
p2 <- remove_typedarray_polyfill(p1)
t1 <- tempfile(fileext = ".html")
htmlwidgets::saveWidget(p1, t1)
file.info(t1)$size
htmlwidgets::saveWidget(p2, t1)
file.info(t1)$size

## End(Not run)
```

---

res_mn   *Minnesotan Indian Reservation Lands*

Description

Minnesotan Indian Reservation Lands

Usage

```
res_mn
```

Format

An sf data frame with 13 features and 5 fields

References

<https://www.dot.state.mn.us/maps/gdma/gis-data.html>

---

schema   *Acquire (and optionally display) plotly's plot schema*

Description

The schema contains valid attributes names, their value type, default values (if any), and min/max values (if applicable).

Usage

```
schema(jsonedit = interactive(), ...)
```

Arguments

jsonedit   use `listviewer::jsonedit` to view the JSON?

...   other options passed onto `listviewer::jsonedit`

Examples

```
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

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

Arguments

x   character string specifying color(s).

...   arguments passed along to `scales::show_col`.

Author(s)

Carson Sievert

Examples

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

---

signup *Create a new plotly account.*

Description

A sign up interface to plotly through the R Console.

Usage

```
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

```
## 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)*

Description

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

Usage

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

Arguments

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

Author(s)

Carson Sievert

See Also

api_download_plot()

Examples

```
# style() is especially useful in conjunction with ggplotly()
# It allows you to leverage the underlying plotly.js library to change
# the return result of ggplotly()
(p <- ggplotly(qplot(data = mtcars, wt, mpg, geom = c("point", "smooth"))))

# removes hoverinfo for the line/ribbon traces (use `plotly_json()` to verify!)
style(p, hoverinfo = "none", traces = c(2, 3))

# another example with plot_ly() instead of ggplotly()
marker <- list(
  color = "red",
  line = list(
    width = 20,
    color = "black"
 )
)
(p <- plot_ly(x = 1:10, y = 1:10, marker = marker))

# note how the entire (marker) object is replaced if a list is provided
style(p, marker = list(line = list(color = "blue")))

# similar to plotly.js, you can update a particular attribute like so
# https://github.com/plotly/plotly.js/issues/1866#issuecomment-314115744
style(p, marker.line.color = "blue")
# this clobbers the previously supplied marker.line.color
style(p, marker.line = list(width = 2.5), marker.size = 10)
```

---

subplot   *View multiple plots in a single view*

Description

View multiple plots in a single view

Usage

```
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 = I("black"), name = "1st", legendgroup = "1st")
 p2 <- add_lines(p2, color = I("red"), name = "2nd", legendgroup = "2nd")

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

```
   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
```

TeX   *Render TeX in a plotly graph using MathJax*

Description

This function makes it slightly easier to render TeX in a plotly graph – it ensures that MathJax is
included with the final result and also ensures the provided string is surrounded with $ (this is what
plotly.js uses to declare a string as TeX).

Usage

```
TeX(x)
```

Arguments

x   a character vector

See Also

config

Examples

```
plot_ly(x = c(1, 2, 3, 4), y = c(1, 4, 9, 16)) %>%
  layout(title = TeX("\\text{Some mathjax: }\\alpha+\\beta x")) %>%
  config(mathjax = "cdn")
```

---

toRGB   *Convert R colours to RGBA hexadecimal colour values*

Description

Convert R colours to RGBA hexadecimal colour values

Usage

```
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

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(ggplot() + geom_bar(aes(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