Package ‘rbokeh’

October 14, 2022

Title R Interface for Bokeh
Version 0.5.2
Description A native R plotting library that provides a flexible declarative interface for creating inter-active web-based graphics, backed by the Bokeh visualization library <https://bokeh.pydata.org/>.
BugReports https://github.com/bokeh/rbokeh/issues
License MIT + file LICENSE
LazyData true
NeedsCompilation no
Imports htmlwidgets (>= 0.5), maps, methods, jsonlite, digest, hexbin,
lazyeval, pryr, magrittr, scales, gistr
Suggests testthat, data.table, lattice, lintr, roxygen2 (>= 5.0.0),
latticeExtra, knitr, rmarkdown, markdown
Enhances shiny (>= 0.12)
RoxygenNote 7.1.1
Encoding UTF-8
VignetteBuilder knitr
Author Ryan Hafen [aut, cre],
    Kenton Russell [ctb],
    Jonathan Owen [ctb],
    Barret Schloerke [ctb],
    Saptasrhi Guha [ctb],
    Continuum Analytics, Inc. [aut, cph] (Bokeh library in htmlwidgets/lib,
    https://bokeh.pydata.org)
Maintainer Ryan Hafen <rhafen@gmail.com>
Repository CRAN
Date/Publication 2021-08-04 00:40:02 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbokeh-package</td>
<td>3</td>
</tr>
<tr>
<td>bk_default_theme</td>
<td>4</td>
</tr>
<tr>
<td>bokeh_render_json</td>
<td>4</td>
</tr>
<tr>
<td>b_eval</td>
<td>5</td>
</tr>
<tr>
<td>catjitter</td>
<td>5</td>
</tr>
<tr>
<td>console_callback</td>
<td>6</td>
</tr>
<tr>
<td>custom_callback</td>
<td>6</td>
</tr>
<tr>
<td>data_name_list</td>
<td>7</td>
</tr>
<tr>
<td>debug_callback</td>
<td>7</td>
</tr>
<tr>
<td>elements</td>
<td>8</td>
</tr>
<tr>
<td>figure</td>
<td>9</td>
</tr>
<tr>
<td>figure_data</td>
<td>12</td>
</tr>
<tr>
<td>flightfreq</td>
<td>12</td>
</tr>
<tr>
<td>get_object_refs</td>
<td>13</td>
</tr>
<tr>
<td>gmap</td>
<td>13</td>
</tr>
<tr>
<td>gmap_style</td>
<td>15</td>
</tr>
<tr>
<td>grid_plot</td>
<td>16</td>
</tr>
<tr>
<td>ly_abline</td>
<td>18</td>
</tr>
<tr>
<td>ly_annular_wedge</td>
<td>21</td>
</tr>
<tr>
<td>ly_annulus</td>
<td>23</td>
</tr>
<tr>
<td>ly_arc</td>
<td>26</td>
</tr>
<tr>
<td>ly_bar</td>
<td>28</td>
</tr>
<tr>
<td>ly bezier</td>
<td>31</td>
</tr>
<tr>
<td>ly_boxplot</td>
<td>33</td>
</tr>
<tr>
<td>ly_contour</td>
<td>35</td>
</tr>
<tr>
<td>ly_crect</td>
<td>37</td>
</tr>
<tr>
<td>ly_curve</td>
<td>40</td>
</tr>
<tr>
<td>ly_density</td>
<td>42</td>
</tr>
<tr>
<td>ly_hexbin</td>
<td>43</td>
</tr>
<tr>
<td>ly_hist</td>
<td>45</td>
</tr>
<tr>
<td>ly_image</td>
<td>47</td>
</tr>
<tr>
<td>ly_image_url</td>
<td>48</td>
</tr>
<tr>
<td>ly_lines</td>
<td>49</td>
</tr>
<tr>
<td>ly_map</td>
<td>51</td>
</tr>
<tr>
<td>ly_multi_line</td>
<td>53</td>
</tr>
<tr>
<td>ly_oval</td>
<td>54</td>
</tr>
<tr>
<td>ly_patch</td>
<td>57</td>
</tr>
<tr>
<td>ly_points</td>
<td>59</td>
</tr>
<tr>
<td>ly_polygons</td>
<td>62</td>
</tr>
<tr>
<td>ly_quadratic</td>
<td>64</td>
</tr>
<tr>
<td>ly_quantile</td>
<td>66</td>
</tr>
<tr>
<td>ly_ray</td>
<td>68</td>
</tr>
<tr>
<td>ly_rect</td>
<td>70</td>
</tr>
<tr>
<td>ly_segments</td>
<td>72</td>
</tr>
<tr>
<td>ly_text</td>
<td>74</td>
</tr>
<tr>
<td>ly_wedge</td>
<td>76</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
</tr>
<tr>
<td>nyctaxihex</td>
<td>79</td>
</tr>
<tr>
<td>pal_color</td>
<td>80</td>
</tr>
<tr>
<td>phantom_install</td>
<td>80</td>
</tr>
<tr>
<td>point_types</td>
<td>81</td>
</tr>
<tr>
<td>print_model_json</td>
<td>81</td>
</tr>
<tr>
<td>rbokeh2html</td>
<td>82</td>
</tr>
<tr>
<td>rbokehOutput</td>
<td>83</td>
</tr>
<tr>
<td>renderRbokeh</td>
<td>84</td>
</tr>
<tr>
<td>set_palette</td>
<td>85</td>
</tr>
<tr>
<td>set_theme</td>
<td>87</td>
</tr>
<tr>
<td>shiny_callback</td>
<td>88</td>
</tr>
<tr>
<td>sub_names</td>
<td>88</td>
</tr>
<tr>
<td>theme_axis</td>
<td>89</td>
</tr>
<tr>
<td>theme_grid</td>
<td>93</td>
</tr>
<tr>
<td>theme_legend</td>
<td>95</td>
</tr>
<tr>
<td>theme_plot</td>
<td>97</td>
</tr>
<tr>
<td>theme_title</td>
<td>99</td>
</tr>
<tr>
<td>tool_box_select</td>
<td>100</td>
</tr>
<tr>
<td>tool_box_zoom</td>
<td>102</td>
</tr>
<tr>
<td>tool_crosshair</td>
<td>103</td>
</tr>
<tr>
<td>tool_hover</td>
<td>103</td>
</tr>
<tr>
<td>tool_lasso_select</td>
<td>104</td>
</tr>
<tr>
<td>tool_pan</td>
<td>105</td>
</tr>
<tr>
<td>tool_reset</td>
<td>106</td>
</tr>
<tr>
<td>tool_resize</td>
<td>107</td>
</tr>
<tr>
<td>tool_save</td>
<td>108</td>
</tr>
<tr>
<td>tool_selection</td>
<td>108</td>
</tr>
<tr>
<td>tool_tap</td>
<td>109</td>
</tr>
<tr>
<td>tool_wheel_zoom</td>
<td>110</td>
</tr>
<tr>
<td>widget2gist</td>
<td>110</td>
</tr>
<tr>
<td>widget2png</td>
<td>112</td>
</tr>
<tr>
<td>x_axis</td>
<td>112</td>
</tr>
<tr>
<td>x_range</td>
<td>115</td>
</tr>
<tr>
<td>y_axis</td>
<td>116</td>
</tr>
<tr>
<td>y_range</td>
<td>118</td>
</tr>
<tr>
<td>%&gt;%</td>
<td>119</td>
</tr>
</tbody>
</table>

Index 120

---

**rbokeh-package**  
**rbokeh**: *R interface for Bokeh*

---

**Description**

Details

For full documentation on the package, visit https://hafen.github.io/rbokeh/

---

### bk_default_theme

**Themes**

**Usage**

bk_default_theme()

bk_ggplot_theme()

---

### bokeh_render_json

**Plot a Bokeh JSON specification**

**Description**

Take a path to a Bokeh JSON plot specification file and render it in the browser.

**Usage**

bokeh_render_json(json_file)

**Arguments**

json_file path to json file

**Note**

This is mainly useful for development / debugging purposes for reading in json created from another platform like Python, or to be used with tweaking json output from print_model_json.

**See Also**

print_model_json
**b_eval**

*Eval lazy symbol*

---

**Description**

Evaluate the argument from the env it came from, or from within the data. The arg supplied to the returned function must be lazy.

**Usage**

```r
b_eval(data)
```

**Arguments**

- `data` : data set to be used for evaluation. May be NULL

**Value**

a function that takes in one lazy argument to be evaluated

---

**catjitter**

*Add a small amount of (rbokeh-compatible) noise to a character vector*

---

**Description**

Add a small amount of (rbokeh-compatible) noise to a character vector

**Usage**

```r
catjitter(x, factor = 0.5)
```

**Arguments**

- `x` : numeric vector to which jitter should be added
- `factor` : a factor between 0 and 1 that

**Examples**

```r
figure(data = lattice::singer) %>%
  ly_points(catjitter(voice.part), jitter(height), color = "black") %>%
  ly_boxplot(voice.part, height, with_outliers = FALSE)
```
### console_callback

**Specify a console callback**

**Description**

This registers a callback that simply prints the callback objects in the javascript console of your web browser. A probably more useful callback is the `debug_callback` which will place you inside a debugger in your web browser allowing you to inspect the callback objects.

**Usage**

```r
console_callback()
```

**Examples**

```r
test <- df %>%
    ly_points(1:10) %>%
    x_range(callback = console_callback()) %>%
    y_range(callback = console_callback())
```

### custom_callback

**Specify a custom callback**

**Description**

This registers a callback that allows you to specify your own custom callback javascript code. A probably more useful callback to use in conjunction with this for working on the javascript code is the `debug_callback` which will place you inside a debugger in your web browser allowing you to inspect the callback objects.

**Usage**

```r
custom_callback(code, lnames = NULL, args = NULL)
```

**Arguments**

- `code` 
  a string of javascript callback code
- `lnames` 
  vector of layer names to be made available inside the callback in addition to the default callback objects (see details)
- `args` 
  named list of additional references to objects to be addressable in the callback
Details

If we add a layer and provide it, for example the `lname` "points", then if we refer to it using the `lnames` parameter to the callback, several objects will be made available inside the callback for you to access, given the names "points_data", "points_glyph", "points_glyph_rend", "points_hov_glyph", "points_ns_glyph", all pointers to different objects associated with the "points" layer that your callback can manipulate.

Examples

```r
# hover over the blue points and make the orange points move
figure(title = "hover a blue point") %>%
  ly_points(1:10, lname = "blue", lgroup = "g1") %>%
  ly_points(2:12, lname = "orange", lgroup = "g1") %>%
  tool_hover(custom_callback(
    code = "debugger;if(cb_data.index['Var1d'].indices.length > 0)
      orange_data.get('data').x[cb_data.index['Var1d'].indices] += 0.1
      orange_data.trigger('change'), "orange", "blue")
```

data_name_list

List of all types of data name structures that could appear

Description

List of all types of data name structures that could appear

Usage

data_name_list()

describe_callback

Specify a "debug" callback

Description

This registers a callback that simply places you inside a debugger in your web browser allowing you to inspect the callback objects.

Usage

describe_callback(lnames = NULL, args = NULL)
**Arguments**

- **lnames**: vector of layer names to be made available inside the callback in addition to the default callback objects (see `custom_callback` for details)
- **args**: named list of additional references to objects to be addressable in the callback

**Examples**

```r
figure() %>%
  ly_points(1:10, lname = "points") %>%
  tool_tap(debug_callback("points"), "points")
```

**elements**

"Periodic Table" dataset

**Description**

Data for periodic table of the elements

**Usage**

elements

**Examples**

```r
# prepare data
elements <- subset(elements, !is.na(group))
elements$group <- as.character(elements$group)
elements$period <- as.character(elements$period)

# add colors for groups
metals <- c("alkali metal", "alkaline earth metal", "halogen", "metal", "metalloid", "noble gas", "nonmetal", "transition metal")
colors <- c("#a6cee3", "#1f78b4", "#f5b70b", "#b2df8a", "#33a02c", "#bbbb88", "#baa2a6", "#e08e79")
elements$color <- colors[match(elements$metal, metals)]
elements$type <- elements$metal

# make coordinates for labels
elements$symx <- paste(elements$group, ":0.1", sep = "")
elements$numbery <- paste(elements$period, ":0.8", sep = "")
elements$massy <- paste(elements$period, ":0.15", sep = "")
elements$namey <- paste(elements$period, ":0.3", sep = "")

# create figure
p <- figure(title = "Periodic Table", tools = ",
  ylim = as.character(c(7:1)), xlim = as.character(1:18),
```
figure

Initialize a Bokeh figure

Description

Initialize a Bokeh figure

Usage

```r
figure(
data = NULL,
width = NULL,
height = NULL,
title = NULL,
xlab = NULL,
ylab = NULL,
xlim = NULL,
ylim = NULL,
padding_factor = 0.07,
xgrid = TRUE,
ygrid = FALSE, xlab = "", ylab = "",
height = 600, width = 1200)
```

```r
# plot rectangles
ly_crect(group, period, data = elements, 0.9, 0.9,
  fill_color = color, line_color = color, fill_alpha = 0.6,
  hover = list(name, atomic.number, type, atomic.mass,
               electronic.configuration))

# add symbol text
ly_text(symx, period, text = symbol, data = elements,
  font_style = "bold", font_size = "15pt",
  align = "left", baseline = "middle")

# add atomic number text
ly_text(symx, numbery, text = atomic.number, data = elements,
  font_size = "9pt", align = "left", baseline = "middle")

# add name text
ly_text(symx, namey, text = name, data = elements,
  font_size = "6pt", align = "left", baseline = "middle")

# add atomic mass text
ly_text(symx, massy, text = atomic.mass, data = elements,
  font_size = "6pt", align = "left", baseline = "middle")
```
```r
ygrid = TRUE,
xaxes = "below",
yaxes = "left",
legend_location = "top_right",
tools = c("pan", "wheel_zoom", "box_zoom", "reset", "save", "help"),
theme = getOption("bokeh_theme"),
toolbar_location = "above",
h_symmetry = TRUE,
v_symmetry = FALSE,
logo = NULL,
lod_factor = 10,
lod_interval = 300,
lod_threshold = NULL,
lod_timeout = 500,
webgl = FALSE,
...
}

Arguments

data data to be supplied to all layers, if the layer doesn’t supply a data value
width figure width in pixels
height figure height in pixels
title a title to display above the plot. - "title" is also the prefix for a set of Text Properties, so you can set the font for the title with the parameter text_font.
xlab label for x axis
ylab label for y axis
xlim the extent of the plotting area in the x-dimension (will be computed automatically if not specified).
ylim the extent of the plotting area in the y-dimension (will be computed automatically if not specified).
padding_factor if limits are not specified, by what factor should the extents of the data be padded
xgrid whether to draw x axis grid lines
ygrid whether to draw y axis grid lines
xaxes where to put x axis, or FALSE if no x axis ticks / labels
yaxes where to put y axis, or FALSE if no y axis ticks / labels
legend_location ("top_right", "top_left", "bottom_left", "bottom_right") the location where the legend should draw itself, or NULL to omit the legend
tools character vector of interactivity tools options (acceptable values are: "pan", "wheel_zoom", "box_zoom", "reset", "croshair", "box_select", "lasso_select", "reset", "save", "help"). Additionally, tool functions can be called on a figure to specify more control - see the "See Also" section below for a list of tool functions. If NULL, the toolbar will not be drawn. If "" the toolbar will be drawn but no tools will be added by default.
```r
figure

theme

an rbokeh theme to use

toolbar_location

(‘above’, ‘below’, ‘left’, ‘right’) Where the toolbar will be located. If set to
NULL, no toolbar will be attached to the plot.

h_symmetry

(logical) Whether the total horizontal padding on both sides of the plot will be
made equal (the left or right padding amount, whichever is larger).

v_symmetry

(logical) Whether the total vertical padding on both sides of the plot will be
made equal (the top or bottom padding amount, whichever is larger).

logo

(‘normal’, ‘grey’) What version of the Bokeh logo to display on the toolbar. If
set to NULL, no logo will be displayed.

lod_factor

(integer) Decimation factor to use when applying level-of-detail decimation (see
"Controlling level of detail").

lod_interval

(integer) Interval (in ms) during which an interactive tool event will enable level-
of-detail downsampling (see "Controlling level of detail").

lod_threshold

(integer) A number of data points, above which level-of-detail downsampling
may be performed by glyph renderers. Set to NULL to disable any level-of-detail
downsampling (see "Controlling level of detail").

lod_timeout

(integer) Timeout (in ms) for checking whether interactive tool events are still
occurring. Once level-of-detail mode is enabled, a check is made every lod_timeout
ms. If no interactive tool events have happened, level-of-detail mode is disabled
(see "Controlling level of detail").

webgl

(logical) should webgl be used for rendering?

... parameters can be specified here that are available in `theme_plot`.

Controlling level of detail

Although the HTML canvas can comfortably display tens or even hundreds of thousands of glyphs,
doing so can have adverse affects on interactive performance. In order to accommodate large-ish
(but not enormous) data sizes, Bokeh plots offer "Level of Detail" (LOD) capability in the client.

The basic idea is that during interactive operations (e.g., panning or zooming), the plot only draws
some small fraction data points. This hopefully allows the general sense of the interaction to be
preserved mid-flight, while maintaining interactive performance. See the lod_ parameters for in-
formation on how to control this.

See Also

Layers to add to a figure: `ly_abline`; `ly_annular_wedge`; `ly_annulus`; `ly_arc`; `ly_bezier`;
`ly_boxplot`; `ly_contour`; `ly_crect`; `ly_curve`; `ly_density`; `ly_hist`; `ly_image_url`;
`ly_image`; `ly_lines`; `ly_map`; `ly_multi_line`; `ly_oval`; `ly_patch`; `ly_points`; `ly_polygons`; `ly_quadratic`;
`ly_quantile`; `ly_ray`; `ly_segments`; `ly_text`; `ly_wedge` Tools to add to a figure: `tool_box_select`;
`tool_box_zoom`; `tool_crosshair`; `tool_lasso_select`; `tool_reset`; `tool_resize`; `tool_save`;
`tool_wheel_zoom` Other figure types: `grid_plot`; `gmap`

Examples

```r
figure() %>% ly_points(1:10)
```
**figure_data**

*Retrieve rbokeh figure data*

**Description**

Retrieve rbokeh figure data

**Usage**

```r
figure_data(fig)
```

**Arguments**

- `fig` rbokeh figure

---

**flightfreq**

*Flight frequency dataset*

**Description**

Daily counts of domestic flights in the U.S. from 1999 to mid-2008

**Usage**

```r
flightfreq
```

**Examples**

```r
p <- figure(width = 1000) %>%
  ly_points(date, Freq, data = flightfreq,
    hover = list(date, Freq, dow), size = 5) %>%
  ly_abline(v = as.Date("2001-09-11"))
p
```
get_object_refs

Description
Get object ids and types from a figure

Usage
get_object.refs(fig)

Arguments
fig
a figure object

gmap

Description
Initialize a Bokeh Google Map plot

Usage
gmap(
  lat = 0,
  lng = 0,
  zoom = 0,
  api_key = NULL,
  map_type = "hybrid",
  map_style = NULL,
  width = 480,
  height = 480,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  xlim = NULL,
  ylim = NULL,
  padding_factor = 0.07,
  xgrid = FALSE,
  ygrid = FALSE,
  xaxes = FALSE,
  yaxes = FALSE,
  tools = c("pan", "wheel_zoom", "save"),
  theme = getOption("bokeh_theme")
)
Arguments

lat    latitude where the map should be centered
lng    longitude where the map should be centered
zoom  initial zoom level to use when displaying the map
api_key  Google Maps API key
map_type  map type to use for the plot - one of "hybrid", "satellite", "roadmap", "terrain"
map_style  a json string of a Google Maps style - see gmap_style
width    figure width in pixels
height   figure width in pixels
title   a title to display above the plot. - "title" is also the prefix for a set of Text Properties, so you can set the font for the title with the parameter text_font.
xlab    label for x axis
ylab    label for y axis
xlim    the extent of the plotting area in the x-dimension (will be computed automatically if not specified).
ylim    the extent of the plotting area in the y-dimension (will be computed automatically if not specified).
padding_factor  if limits are not specified, by what factor should the extents of the data be padded
xgrid   whether to draw x axis grid lines
ygrid   whether to draw y axis grid lines
xaxes   where to put x axis, or FALSE if no x axis ticks / labels
yaxes   where to put y axis, or FALSE if no y axis ticks / labels
tools   character vector of interactivity tools options (acceptable values are: "pan", "wheel_zoom", "box_zoom", "resize", "crosshair", "box_select", "lasso_select", "reset", "save", "help"). Additionally, tool functions can be called on a figure to specify more control - see the "See Also" section below for a list of tool functions. If NULL, the toolbar will not be drawn. If "" the toolbar will be drawn but no tools will be added by default.
theme    an rbokeh theme to use

Note

This can be used in the same way as figure, adding layers on top of the Google Map. There is an open issue documenting points appearing to sometimes be a few pixels off from their intended location. Google has its own terms of service for using Google Maps API and any use of rbokeh with Google Maps must be within Google’s Terms of Service

See Also

  gmap_style
Examples

```r
# custom map style
gmap(lat = 40.74, lng = -73.95, zoom = 11,
    width = 600, height = 600,
    map_style = gmap_style("blue_water"))
## Not run:
gmap(title = "NYC taxi pickups January 2013",
    lat = 40.74, lng = -73.95, zoom = 11,
    map_type = "roadmap", width = 1000, height = 800) %>%
  ly_hexbin(nyctaxihex, alpha = 0.5,
           palette = "Spectral10", trans = log, inv = exp)
## End(Not run)
```

---

gmap_style

Get a Google Map Style

Description

Get a Google Map Style

Usage

```r
gmap_style(name)
```

Arguments

- `name` - name of map style to retrieve (see details)

Details

This function provides Google Maps themes that can be passed to the `map_style` argument of `gmap`. Currently the most popular styles from https://snazzymaps.com are available. You can also visit this site or others to specify a custom `map_style`. Available styles are: "subtle_grayscale", "shades_of_grey", "blue_water", "pale_dawn", "blue_essence", "apple_mapsesque", "midnight_commander", "light_monochrome", "paper", "retro", "flat_map", "cool_grey".

See Also

- `gmap`
Examples

```r
# custom map style
gmap(lat = 40.74, lng = -73.95, zoom = 11,
     width = 600, height = 600,
     map_style = gmap_style("blue_water"))

## Not run:
gmap(title = "NYC taxi pickups January 2013",
     lat = 40.74, lng = -73.95, zoom = 11,
     map_type = "roadmap", width = 1000, height = 800) %>%
     ly_hexbin(nyctaxihex, alpha = 0.5,
     palette = "Spectral10", trans = log, inv = exp)

## End(Not run)
```

---

**grid_plot**

Create a Bokeh grid plot from a list of Bokeh figures

### Description

Create a Bokeh grid plot from a list of Bokeh figures

### Usage

```r
ggrid_plot(
  figs,
  width = NULL,
  height = NULL,
  nrow = 1,
  ncol = 1,
  byrow = TRUE,
  xlim = NULL,
  ylim = NULL,
  logo = NULL,
  same_axes = FALSE,
  simplify_axes = TRUE,
  y_margin = NULL,
  x_margin = NULL,
  link_data = FALSE
)
```

### Arguments

- **figs**: list of Bokeh figures - see details for what is acceptable
- **width**: width of the entire grid plot in pixels - if NULL, the sum of the grid widths of columns will be used - if not NULL, the widths of the plots will be proportionately shrunk to meet the specified width
grid_plot

height  height of the entire grid plot in pixels - if NULL, the sum of the grid heights of rows will be used - if not NULL, the heights of the plots will be proportionately shrunk to meet the specified height

nrow  number of rows in the grid

ncol  number of columns in the grid

byrow  populate the grid by row according to the order of figure elements supplied in params

xlim  the extent of the plotting area in the x-dimension to be applied to every panel (original individual panel limits will be honored if not specified).

ylim  the extent of the plotting area in the y-dimension to be applied to every panel (original individual panel limits will be honored if not specified).

logo  ('normal', 'grey') What version of the Bokeh logo to display on the toolbar. If set to NULL, no logo will be displayed.

same_axes  logical or vector of two logicals specifying whether the x and/or y axis limits should be the same for each plot in the grid

simplify_axes  logical or vector of logicals specifying whether to simply the x and/or y axes (only show the axes along the bottom and left sides of the grid) - only valid if same_axes is TRUE for the axis

x_margin, y_margin  specify the margin space in pixels to be left for axes when using simplify_axes=TRUE

link_data  logical - should an attempt be made to join the data sources of each plot for linked brushing? (see details)

Details

The figs parameter can either be a list of figures or a list of lists of figures. If the latter, the list structure will determine the layout, with each super-list of figures defining a single row of the grid. If the former, the parameters nrow and ncol and byrow are used to determine the layout. The grid is from top to bottom left to right.

If link_data is TRUE, then an effort will be made to link all data sources that are common among the different figures in the plot. Note that at this point, only data sources that are specified in the data argument to the different layer functions are checked.

Examples

```r
idx <- split(1:150, iris$Species)
figs <- lapply(idx, function(x) {
  figure(width = 300, height = 300) %>%
    ly_points(Sepal.Length, Sepal.Width, data = iris[x, ],
              hover = list(Sepal.Length, Sepal.Width))
})

# 1 row, 3 columns
grid_plot(figs)
# specify xlim and ylim to be applied to all panels
grid_plot(figs, xlim = c(4, 8), ylim = c(1.5, 4.5))
```
# unnamed list will remove labels
grid_plot(unname(figs))
# 2 rows, 2 columns
grid_plot(figs, nrow = 2)
# x and y axis with same (and linked) limits
grid_plot(figs, same_axes = TRUE)
# x axis with same (and linked) limits
grid_plot(figs, same_axes = c(TRUE, FALSE), nrow = 2)
# x axis with same (and linked) limits and custom xlim
grid_plot(figs, same_axes = c(TRUE, FALSE), xlim = c(5, 7), nrow = 2)
# send lists instead of specifying nrow and ncol
grid_plot(list(
  c(list(figs[[1]])), list(figs[[3]])),
  c(list(NULL), list(figs[[2]]))
))
# a null entry will be skipped in the grid
figs2 <- figs
figs2[[1]] <- list(NULL)
grid_plot(figs2, nrow = 2)
# with themes
grid_plot(figs) %>%
  theme_title(text_color = "red") %>%
  theme_plot(background_fill_color = "#E6E6E6",
    outline_line_color = "white") %>%
  theme_grid(c("x", "y"), grid_line_color = "white",
    minor_grid_line_color = "white",
    minor_grid_line_alpha = 0.4) %>%
  theme_axis(c("x", "y"), axis_line_color = "white",
    major_label_text_color = "#7F7F7F",
    major_tick_line_color = "#7F7F7F",
    minor_tick_line_alpha = 0, num_minor_ticks = 2)
# themes again
grid_plot(figs) %>%
  set_theme(bk_ggplot_theme)

# link data across plots in the grid (try box_select tool)
# (data sources must be the same)
tools <- c("pan", "wheel_zoom", "box_zoom", "box_select", "reset")
p1 <- figure(tools = tools, width = 500, height = 500) %>%
  ly_points(Sepal.Length, Sepal.Width, data = iris, color = Species)
p2 <- figure(tools = tools, width = 500, height = 500) %>%
  ly_points(Petal.Length, Petal.Width, data = iris, color = Species)
grid_plot(list(p1, p2), same_axes = TRUE, link_data = TRUE)

---

**ly_abline**

*Add an "abline" layer to a Bokeh figure*

**Description**

Draws one or more straight lines.
**Usage**

```r
ly_abline(
    fig,
    a = NULL,
    b = NULL,
    v = NULL,
    h = NULL,
    coef = NULL,
    color = "black",
    alpha = NULL,
    width = 1,
    type = 1,
    legend = NULL,
    lname = NULL,
    lgroup = NULL,
    ...
)
```

**Arguments**

- **fig**: figure to modify
- **a, b**: the intercept and slope of the line(s) to draw
- **v**: the x value(s) for vertical lines
- **h**: the y value(s) for horizontal lines
- **coef**: a vector of length two giving the intercept and slope
- **color**: color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
- **alpha**: transparency value for the line between 0 (transparent) and 1 (opaque)
- **width**: stroke width in units of pixels
- **type**: an integer between 1 and 6 matching the lty property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- **legend**: either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
- **lname**: layer name
- **lgroup**: layer group
- **...**: additional parameters for fine control over line properties (see "Additional parameters" below)

**Mapped plot attributes and legends**

When specifying an input data frame for a layer through the `data` argument, columns of `data` can be used to specify various plot attributes such as `color`, etc. For example, with `ly_points(..., data = iris, color = Species)`, the `Species` variable is used to determine how to color the points.
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

**Additional parameters**

- **line_join**: how path segments should be joined together 'miter' 'round' 'bevel'
- **line_cap**: how path segments should be terminated 'butt' 'round' 'square'
- **line_dash**: an integer between 1 and 6 matching the lty property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- **line_dash_offset**: the distance in pixels into the line_dash that the pattern should start from

**See Also**


**Examples**

```r
z <- lm(dist ~ speed, data = cars)
p <- figure() %>%
  ly_points(cars, hover = cars) %>%
  ly_lines(lowess(cars), legend = "lowess") %>%
  ly_abline(z, type = 2, legend = "lm", width = 2)

p

# abline with mixed axes for h and v
figure() %>%
  ly_points(1:26, letters) %>%
  ly_abline(h = "j") %>%
  ly_abline(v = 10)

# multiple hv lines
figure() %>%
  ly_points(1:10) %>%
  ly_abline(v = 1:10) %>%
  ly_abline(h = 1:10)

# multiple ab lines
figure() %>%
  ly_points(0:10) %>%
  ly_abline(0, seq(0, 1, by = 0.1))
```
ly_annular_wedge

Add an "annular_wedge" layer to a Bokeh figure

Description

Add an "annular_wedge" layer to a Bokeh figure

Usage

ly_annular_wedge(
    fig,
    x,
    y = NULL,
    data = figure_data(fig),
    inner_radius = 0.1,
    outer_radius = 0.3,
    start_angle = 0,
    end_angle = 2 * pi,
    direction = "anticlock",
    color = NULL,
    alpha = 1,
    hover = NULL,
    url = NULL,
    legend = NULL,
    lname = NULL,
    lgroup = NULL,
    ...
)

Arguments

fig  figure to modify
x    values or field name of center x coordinates
y    values or field name of center y coordinates
data  an optional data frame, providing the source for inputs x, y, and other glyph
      properties
inner_radius  values or field name of inner radii
outer_radius  values or field name of outer radii
start_angle  the angles to start the annular wedges, in radians, as measured from the horizontal
end_angle  the angles to end the annular wedges, in radians, as measured from the horizontal
direction  direction to turn between starting and ending angles ("anticlock", "clock")
color  color for the glyph - a hex code (with no alpha) or any of the 147 named CSS
       colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see
       "Handling color" below
alpha
the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if
glyph has both fill and color properties, see "Handling alpha" below

hover
a data frame of variables to be displayed when hovering over the glyph or a
vector of variable names that can be found and extracted from the data argument

url
a string of URLs or a single string that references a variable name (via @var_name)
that can be found and extracted from the data argument

legend
either a logical specifying not to plot a legend for this layer (FALSE) or a string
indicating the name of the legend entry for this layer (note that when mapping
plot attributes to variables in data, a legend is automatically created and does
not need to be specified - see "Mapped plot attributes and legends" below)

1name
layer name

1group
layer group

...additional parameters for fine control over fill and line properties (see "Addi-
tional parameters" below)

Handling color
The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.
- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has has line and fill properties, this will be the color of the line and
  the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes
  can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen
from the theme.

Handling alpha
The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.
- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has has line and fill properties, this will be the alpha of the line and
  the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with fill_alpha and line_alpha and will
  override alpha.

Mapped plot attributes and legends
When specifying an input data frame for a layer through the data argument, columns of data can
be used to specify various plot attributes such as color, etc. For example, with ly_points(...,
data = iris, color = Species), the Species variable is used to determine how to color the points.
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be
mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to
each interval. The mapping from the values of the variable to the actual plot attributes is determined
based on the theme.
ly_annulus

**Additional parameters**

- **fill_color**: color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo'
- **fill_alpha**: transparency value between 0 (transparent) and 1 (opaque)
- **line_color**: color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo'
- **line_width**: stroke width in units of pixels
- **line_alpha**: transparency value between 0 (transparent) and 1 (opaque)
- **line_join**: how path segments should be joined together 'miter' 'round' 'bevel'
- **line_cap**: how path segments should be terminated 'butt' 'round' 'square'
- **line_dash**: array of integer pixel distances that describe the on-off pattern of dashing to use
- **line_dash_offset**: the distance in pixels into the line_dash that the pattern should start from

**See Also**

Other layer functions: `ly_abline()`, `ly_annulus()`, `ly_arc()`, `ly_bar()`, `ly_bezier()`, `ly_boxplot()`, `ly_contour()`, `ly_crect()`, `ly_curve()`, `ly_density()`, `ly_hist()`, `ly_image_url()`, `ly_image()`, `ly_lines()`, `ly_map()`, `ly_multi_line()`, `ly_oval()`, `ly_patch()`, `ly_points()`, `ly_polygons()`, `ly_quadratic()`, `ly_quantile()`, `ly_ray()`, `ly_rect()`, `ly_segments()`, `ly_text()`, `ly_wedge()`

**Examples**

```r
rescale <- function(x)
  (x - min(x)) / diff(range(x))
figure() %>%
  ly_annular_wedge(Sepal.Length, Sepal.Width, data = iris,
  end_angle = rescale(Petal.Length) * 2 * pi, color = Species,
  inner_radius = 0.1, outer_radius = 0.15, alpha = 0.5,
  hover = Species)
```

```r
ly_annulus

**Add an "annulus" layer to a Bokeh figure**

**Description**

Add an "annulus" layer to a Bokeh figure

**Usage**

```r
ly_annulus(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  inner_radius = 0.1,
  outer_radius = 0.2,
)```
color = NULL,
alpha = 1,
hover = NULL,
url = NULL,
legend = NULL,
lname = NULL,
lgroup = NULL,
...
)

Arguments

fig figure to modify
x values or field name of center x coordinates
y values or field name of center y coordinates
data an optional data frame, providing the source for inputs x, y, and other glyph properties
inner_radius values or field name of inner radii
outer_radius values or field name of outer radii
color color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below
alpha the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below
hover a data frame of variables to be displayed when hovering over the glyph or a vector of variable names that can be found and extracted from the data argument
url a string of URLs or a single string that references a variable name (via @var_name) that can be found and extracted from the data argument
legend either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
lname layer name
lgroup layer group
... additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
• If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

• When using a glyph that only has line properties, this will be the alpha of the line.
• When using a glyph that has has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
• Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

fill_color color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
fill_alpha transparency value between 0 (transparent) and 1 (opaque)
line_color color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
line_width stroke width in units of pixels
line_alpha transparency value between 0 (transparent) and 1 (opaque)
line_join how path segments should be joined together 'miter' 'round' 'bevel'
line_cap how path segments should be terminated 'butt' 'round' 'square'
line_dash array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset the distance in pixels into the line_dash that the pattern should start from

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_arc(), ly_bar(), ly_bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(), ly_image_url(), ly_image(), ly_lines(), ly_map(), ly_multi_line(), ly_oval(), ly_patch(), ly_points(), ly_polygons(), ly_quadratic(), ly_quantile(), ly_ray(), ly_rect(), ly_segments(), ly_text(), ly_wedge()
Examples

```r
rescale <- function(x)
  (x - min(x)) / diff(range(x))
figure() %>%
  ly_annulus(Sepal.Length, Sepal.Width, data = iris,
  color = Species, hover = Species, alpha = 0.5,
  outer_radius = rescale(Petal.Length) * 0.3,
  inner_radius = rescale(Petal.Length) * 0.1)
```

---

**ly_arc**

*Add an "arc" layer to a Bokeh figure*

---

**Description**

Add an "arc" layer to a Bokeh figure

**Usage**

```r
ly_arc(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  color = NULL,
  alpha = 1,
  width = 2,
  type = 1,
  radius = 0.2,
  start_angle = 0,
  end_angle = 2 * pi,
  direction = "anticlock",
  legend = NULL,
  lname = NULL,
  lgroup = NULL,
  ...
)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fig</code></td>
<td>figure to modify</td>
</tr>
<tr>
<td><code>x</code></td>
<td>values or field name of center x coordinates</td>
</tr>
<tr>
<td><code>y</code></td>
<td>values or field name of center y coordinates</td>
</tr>
<tr>
<td><code>data</code></td>
<td>an optional data frame, providing the source for inputs x, y, and other glyph properties</td>
</tr>
<tr>
<td><code>color</code></td>
<td>color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'</td>
</tr>
</tbody>
</table>
alpha transparency value for the line between 0 (transparent) and 1 (opaque)
width stroke width in units of pixels
type an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
radius values or field name of arc radii
start_angle values or field name of starting angles
end_angle values or field name of ending angles
direction direction to turn between starting and ending angles ("anticlock", "clock")
legend either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
lname layer name
lgroup layer group
... additional parameters for fine control over line properties (see "Additional parameters" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

line_join how path segments should be joined together 'miter' 'round' 'bevel'
line_cap how path segments should be terminated 'butt' 'round' 'square'
line_dash an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset the distance in pixels into the line_dash that the pattern should start from

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_bar(), ly_bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(), ly_image_url(), ly_image(), ly_lines(), ly_map(), ly_multi_line(), ly_oval(), ly_patch(), ly_points(), ly_polygons(), ly_quadratic(), ly_quantile(), ly_ray(), ly_rect(), ly_segments(), ly_text(), ly_wedge()
**Examples**

```r
rescale <- function(x)
  (x - min(x)) / diff(range(x))
figure() %>%
  ly_arc(Sepal.Length, Sepal.Width, data = iris,
         end_angle = rescale(Petal.Length) * 2 * pi, color = Species,
         alpha = 0.5)
```

---

**ly_bar**

`ly_bar` *Add a "barchart" layer to a Bokeh figure*

---

**Description**

Draws a bar chart

**Usage**

```r
ly_bar(
  fig,
  x = NULL,
  y = NULL,
  data = figure_data(fig),
  color = NULL,
  alpha = 1,
  position = c("stack", "fill", "dodge"),
  width = 0.9,
  hover = FALSE,
  origin = NULL,
  breaks = NULL,
  right = FALSE,
  binwidth = NULL,
  lname = NULL,
  lgroup = NULL,
  legend = NULL,
  ...
)
```

**Arguments**

- `fig` figure to modify
- `x` values or field name for x variable, or if NULL, x-axis will be counts of y
- `y` values or field name for y variable, or if NULL, y-axis will be counts of x
- `data` an optional data frame, providing the source for inputs x, y, and color properties
- `color` color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below
alpha the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if
glyph has both fill and color properties, see "Handling alpha" below
position either "stack", "fill", or "dodge" (see details)
width with of each bar, a value between 0 (no width) and 1 (full width)
hover logical - should a hover tool be added to show the value of each bar?
origin, breaks, right, binwidth parameters to be used for binning x when it is continuous (not yet implemented)
lname layer name
lgroup layer group
legend either a logical specifying not to plot a legend for this layer (FALSE) or a string
indicating the name of the legend entry for this layer (note that when mapping
plot attributes to variables in data, a legend is automatically created and does
not need to be specified - see "Mapped plot attributes and legends" below)
...
additional parameters for fine control over fill and line properties (see "Addi-
tional parameters" below)

Details
This function expects one of either x or y to be categorical and the other to be numeric or NULL. The numeric variable is summed for each categorical variable and bars are plotted. If no numeric variable is supplied, the unique values of the categorical variable will be tabulated. Within each categorical variable, if color maps to another grouping variable then the bars are split up. In this case, there are three ways to display the bars with the position argument. The default, "stack" will stack the bars. The "fill" choice will show the relative proportion for each group within each categorical variable level, stacking the bars. The "dodge" choice will plot the bars for each level of the categorical variable side by side.

Handling color
The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

• When using a glyph that only has line properties, this will be the color of the line.
• When using a glyph that has line and fill properties, this will be the color of the line and
  the fill, with the alpha level of the fill reduced by 50%.
• If full control over fill and line color is desired, the fill_color and line_color attributes
  can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha
The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

• When using a glyph that only has line properties, this will be the alpha of the line.
• When using a glyph that has has line and fill properties, this will be the alpha of the line and
  the alpha of the fill will be set to 50% of this value.
• Individual fill and line alpha can be specified with fill_alpha and line_alpha and will
  override alpha.
Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with `ly_points(..., data = iris, color = Species)`, the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

- `fill_color` - color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
- `fill_alpha` - transparency value between 0 (transparent) and 1 (opaque)
- `line_color` - color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
- `line_width` - stroke width in units of pixels
- `line_alpha` - transparency value between 0 (transparent) and 1 (opaque)
- `line_join` - how path segments should be joined together 'miter' 'round' 'bevel'
- `line_cap` - how path segments should be terminated 'butt' 'round' 'square'
- `line_dash` - array of integer pixel distances that describe the on-off pattern of dashing to use
- `line_dash_offset` - the distance in pixels into the line_dash that the pattern should start from

See Also


Examples

```r
# count of variety
figure() %>%
  ly_bar(variety, data = lattice::barley) %>%
  theme_axis("x", major_label_orientation = 90)

# total yield per variety
figure() %>%
  ly_bar(variety, yield, data = lattice::barley, hover = TRUE) %>%
  theme_axis("x", major_label_orientation = 90)

# swap axes and add hover
figure() %>%
  ly_bar(yield, variety, data = lattice::barley, hover = TRUE)

# stack by year
```
ly_bezier

Figure() %>%
ly_bar(variety, yield, color = year, data = lattice::barley, hover = TRUE) %>%
theme_axis("x", major_label_orientation = 90)

# proportional bars
figure() %>%
ly_bar(variety, yield, color = year,
data = lattice::barley, position = "fill", width = 1) %>%
theme_axis("x", major_label_orientation = 90) %>%
set_palette(discrete_color = pal_color(c("red", "blue")))

# swap axes and use different palette
figure() %>%
ly_bar(yield, variety, color = year,
data = lattice::barley, position = "fill") %>%
set_palette(discrete_color = pal_color(c("red", "blue")))

# side by side bars
figure() %>%
ly_bar(variety, yield, color = year,
data = lattice::barley, position = "dodge") %>%
theme_axis("x", major_label_orientation = 90)

# use a different theme
figure() %>%
ly_bar(variety, yield, color = year,
data = lattice::barley, position = "dodge") %>%
theme_axis("x", major_label_orientation = 90)

---

**ly_bezier**

*Add a "bezier" layer to a Bokeh figure*

**Description**

Draws Bezier curves with the given starting, ending, and control points.

**Usage**

```r
ly_bezier(
  fig,
x0,
y0,
x1,
y1,
cx0,
cy0,
cx1,
cy1,
```

---
data = figure_data(fig),
color = "black",
alpha = 1,
width = 1,
type = 1,
legend = NULL,
lname = NULL,
lgroup = NULL,
...
}

Arguments

fig  figure to modify
x0   values or field name of starting x coordinates
y0   values or field name of starting y coordinates
x1   values or field name of ending x coordinates
y1   values or field name of ending y coordinates
cx0  values or field name of first control point x coordinates
cy0  values or field name of first control point y coordinates
cx1  values or field name of second control point x coordinates
cy1  values or field name of second control point y coordinates
data  an optional data frame, providing the source for start, end, and control point inputs, as well as other glyph properties
color color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
alpha transparency value for the line between 0 (transparent) and 1 (opaque)
width stroke width in units of pixels
type an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
legend either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
lname layer name
lgroup layer group
... additional parameters for fine control over line properties (see "Additional parameters" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points.
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

**Additional parameters**

- `line_join`: how path segments should be joined together 'miter' 'round' 'bevel'
- `line_cap`: how path segments should be terminated 'butt' 'round' 'square'
- `line_dash`: an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that define a dash pattern
- `line_dash_offset`: the distance in pixels into the line_dash that the pattern should start from

---

**See Also**


---

### ly_boxplot

Add a "boxplot" layer to a Bokeh figure

---

**Description**

Add a "boxplot" layer to a Bokeh figure

**Usage**

```r
ly_boxplot(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  width = 0.9,
  coef = 1.5,
  color = "blue",
  alpha = 1,
  outlier_glyph = 1,
  outlier_size = 10,
  lname = NULL,
  lgroup = NULL,
  ...
)
```
Arguments

fig  
figure to modify

x  
either a numeric vector or a factor

y  
either a numeric vector or a factor

data  
an optional data frame, providing the source for x and y

width  
with of each box, a value between 0 (no width) and 1 (full width)

coef  
see boxplot.stats

color  
color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below

alpha  
the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below

outlier_glyph  
the glyph used to plot the outliers. If set to NA, no outlier points are plotted. Run point_types() for possible values.

outlier_size  
the size of the glyph used to plot outliers. If set to NA, no outlier points are plotted.

lname  
layer name

lgroup  
layer group

...  
additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.

Additional parameters
Add a "contour" layer to a Bokeh figure

Description

Computes and draws contour lines.

Usage

ly_contour(fig, z, x = seq(0, 1, length.out = nrow(z)), y = seq(0, 1, length.out = ncol(z)), nlevels = 10, levels = pretty(range(z, na.rm = TRUE), nlevels), color = "black", alpha = 1, width = 1, type = 1, lname = NULL, lgroup = NULL, ...)

Examples

```r
lyBoxplot(voice.part, height, data = lattice::singer)
```
Arguments

- **fig**: figure to modify
- **z**: a matrix containing the values to compute contour lines for
- **x, y**: locations of grid lines at which the values in `image` are measured (see `contourLines`)
- **nlevels, levels**: parameters sent to `contourLines`
- **color**: color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
- **alpha**: transparency value for the line between 0 (transparent) and 1 (opaque)
- **width**: stroke width in units of pixels
- **type**: an integer between 1 and 6 matching the `lty` property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- **lname**: layer name
- **lgroup**: layer group
- **...**: additional parameters for fine control over line properties (see "Additional parameters" below)

Additional parameters

- **line_join**: how path segments should be joined together 'miter' 'round' 'bevel'
- **line_cap**: how path segments should be terminated 'butt' 'round' 'square'
- **line_dash**: an integer between 1 and 6 matching the `lty` property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- **line_dash_offset**: the distance in pixels into the `line_dash` that the pattern should start from

See Also


Examples

```r
p <- figure(xlim = c(0, 1), ylim = c(0, 1), title = "Volcano") %>%
  ly_image(volcano) %>%
  ly_contour(volcano)
p
```
ly_crect

Add a "crect" (centered rectangle) layer to a Bokeh figure

Description

Add a "crect" (centered rectangle) layer to a Bokeh figure

Usage

```r
ly_crect(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  width = 1,
  height = 1,
  angle = 0,
  dilate = FALSE,
  color = NULL,
  alpha = 1,
  hover = NULL,
  url = NULL,
  legend = NULL,
  lname = NULL,
  lgroup = NULL,
  ...
)
```

Arguments

- `fig` figure to modify
- `x` values or field name of center x coordinates
- `y` values or field name of center y coordinates
- `data` an optional data frame, providing the source for inputs xleft, ybottom, xright, ytop, and other glyph properties
- `width` values or field name of widths
- `height` values or field name of heights
- `angle` values or field name of rotation angles
- `dilate` logical - whether to dilate pixel distance computations when drawing
- `color` color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below
- `alpha` the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below
hover is a data frame of variables to be displayed when hovering over the glyph or a
vector of variable names that can be found and extracted from the data argument
url is a string of URLs or a single string that references a variable name (via @var_name)
    that can be found and extracted from the data argument
legend is either a logical specifying not to plot a legend for this layer (FALSE) or a string
    indicating the name of the legend entry for this layer (note that when mapping
    plot attributes to variables in data, a legend is automatically created and does
    not need to be specified - see "Mapped plot attributes and legends" below)
lname is layer name
lgroup is layer group
... is additional parameters for fine control over fill and line properties (see "Addi-
tional parameters" below)

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has has line and fill properties, this will be the color of the line and
  the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes
  can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen
from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has has line and fill properties, this will be the alpha of the line and
  the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with fill_alpha and line_alpha and will
  override alpha.

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can
be used to specify various plot attributes such as color, etc. For example, with ly_points(...,
data = iris, color = Species), the Species variable is used to determine how to color the points.
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be
mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to
each interval. The mapping from the values of the variable to the actual plot attributes is determined
based on the theme.

Additional parameters
fill_color  color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g `'green'`
fill_alpha  transparency value between 0 (transparent) and 1 (opaque)
line_color  color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g `'green'`
line_width  stroke width in units of pixels
line_alpha  transparency value between 0 (transparent) and 1 (opaque)
line_join  how path segments should be joined together 'miter' 'round' 'bevel'
line_cap  how path segments should be terminated 'butt' 'round' 'square'
line_dash  array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset  the distance in pixels into the line_dash that the pattern should start from

See Also

Other layer functions: `ly_abline()`, `ly_annular_wedge()`, `ly_annulus()`, `ly_arc()`, `ly_bar()`,
`ly_bezier()`, `ly_boxplot()`, `ly_contour()`, `ly_curve()`, `ly_density()`, `ly_hist()`, `ly_image_url()`,
`ly_image()`, `ly_lines()`, `ly_map()`, `ly_multiline()`, `ly oval()`, `ly_patch()`, `ly_points()`,
`ly_polygons()`, `ly_quadratic()`, `ly_quantile()`, `ly Ray()`, `ly_rect()`, `ly_segments()`, `ly_text()`,
`ly_wedge()`

Examples

```r
# prepare data
elements <- subset(elements, !is.na(group))
elements$group <- as.character(elements$group)
elements$period <- as.character(elements$period)

# add colors for groups
metals <- c("alkali metal", "alkaline earth metal", "halogen",
"metal", "metalloid", "noble gas", "nonmetal", "transition metal")
colors <- c("#a6cee3", "#1f78b4", "#fb8f08", "#b2df8a", "#33a02c",
"#bab086", "#b3a2a6", "#e08e79")
elements$color <- colors[match(elements$metal, metals)]
elements$type <- elements$metal

# make coordinates for labels
elements$symx <- paste(elements$group, ":0.1", sep = "")
elements$numbery <- paste(elements$period, ":0.8", sep = "")
elements$massy <- paste(elements$period, ":0.15", sep = "")
elements$namey <- paste(elements$period, ":0.3", sep = "")

# create figure
p <- figure(title = "Periodic Table", tools = ":",
ylim = as.character(c(7:1)), xlim = as.character(1:18),
xgrid = FALSE, ygrid = FALSE, xlab = "", ylab = "",
height = 600, width = 1200) %>%

# plot rectangles
ly_crect(group, period, data = elements, 0.9, 0.9,
fill_color = color, line_color = color, fill_alpha = 0.6,
```
ly_curve

Add a "curve" layer to a Bokeh figure

Description

Draws a curve corresponding to a function over the interval [from, to].

Usage

ly_curve(
    fig,
    expr,
    from = NULL,
    to = NULL,
    n = 101,
    color = "black",
    alpha = 1,
    width = 1,
    type = 1,
    legend = NULL,
    lname = NULL,
    lgroup = NULL,
    ...
)
Arguments

fig
expr, from, to, n

Arguments to modify
parameters sent to curve

color
color to use to stroke lines with - a hex code (with no alpha) or any of the 147
named CSS colors, e.g. 'green', 'indigo'

alpha
transparency value for the line between 0 (transparent) and 1 (opaque)

width
stroke width in units of pixels

type
an integer between 1 and 6 matching the lty property in par or an array of
integer pixel distances that describe the on-off pattern of dashing to use

legend
either a logical specifying not to plot a legend for this layer (FALSE) or a string
indicating the name of the legend entry for this layer (note that when mapping
plot attributes to variables in data, a legend is automatically created and does
not need to be specified - see "Mapped plot attributes and legends" below)

lname
layer name

lgroup
layer group

... additional parameters for fine control over line properties (see "Additional pa-
rameters" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can
be used to specify various plot attributes such as color, etc. For example, with
ly_points(...,
data = iris, color = Species)
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be
mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to
each interval. The mapping from the values of the variable to the actual plot attributes is determined
based on the theme.

Additional parameters

line_join how path segments should be joined together 'miter' 'round' 'bevel'
line_cap how path segments should be terminated 'but' 'round' 'square'

line_dash an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset the distance in pixels into the line_dash that the pattern should start from

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_arc(), ly_bar(),
ly_bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_density(), ly_hist(), ly_image_url(),
ly_image(), ly_lines(), ly_map(), ly_multi_line(), ly_oval(), ly_patch(), ly_points(),
ly_polygons(), ly_quadratic(), ly_quantile(), ly_ray(), ly_rect(), ly_segments(), ly_text(),
ly_wedge()
Examples

chippy <- function(x) sin(cos(x)*exp(-x/2))
figure(width = 800) %>%
  ly_curve(chippy, -8, 7, n = 2001)

ly_density

Add a "density" layer to a Bokeh figure

Description

Draws a kernel density estimate

Usage

ly_density(
  fig, x, data = figure_data(fig), bw = "nrd0",
  adjust = 1, kernel = c("gaussian", "epanechnikov", "rectangular", "triangular", "biweight",
                      "cosine", "optcosine"), weights = NULL, window = kernel,
  n = 512, cut = 3, na.rm = FALSE, color = "black",
  alpha = 1, width = 1, type = 1, legend = NULL, lname = NULL,
  lgroup = NULL, ...
)

Arguments

fig figure to modify
x, bw, adjust, kernel, weights, window, n, cut, na.rm
  parameters passed to density
data an optional data frame, providing the source for x
color color to use to stroke lines with - a hex code (with no alpha) or any of the 147
  named CSS colors, e.g 'green', 'indigo'
ly_hexbin

Add a "hexbin" layer to a Bokeh figure

Description

Add a "hexbin" layer to a Bokeh figure

Parameters

- **alpha**: transparency value for the line between 0 (transparent) and 1 (opaque)
- **width**: stroke width in units of pixels
- **type**: an integer between 1 and 6 matching the lty property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- **legend**: text to display in the legend entry for the density line
- **lname**: layer name
- **lgroup**: layer group
- ... additional parameters for fine control over line properties (see "Additional parameters" below)

Additional parameters

- **line_join**: how path segments should be joined together 'miter' 'round' 'bevel'
- **line_cap**: how path segments should be terminated 'butt' 'round' 'square'
- **line_dash**: an integer between 1 and 6 matching the lty property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- **line_dash_offset**: the distance in pixels into the line_dash that the pattern should start from

See Also


Examples

```r
h <- figure(width = 600, height = 400) %>%
  ly_hist(eruptions, data = faithful, breaks = 40, freq = FALSE) %>%
  ly_density(eruptions, data = faithful)
h
```
ly_hexbin

Usage

ly_hexbin(
    fig,
    x,
    y = NULL,
    data = figure_data(fig),
    xbins = 30,
    shape = 1,
    xbnds = NULL,
    ybnds = NULL,
    style = "colorscale",
    trans = NULL,
    inv = NULL,
    lname = NULL,
    palette = "RdYlGn11",
    line = FALSE,
    alpha = 1,
    hover = TRUE
)

Arguments

fig  figure to modify
x    values or field name of center x coordinates to be binned
y    values or field name of center y coordinates to be binned
data  an optional data frame, providing the source for x and y
xbins, shape, xbnds, ybnds
      parameters passed to hexbin
style  type of plotting for hexbins (see grid.hexagons) - "colorramp" and "lattice" are currently supported
trans, inv
      transformation and inverse transformation function for the bin counts
lname  layer name
palette  name of color palette to use for color ramp (see here for acceptable values)
line  logical - should hexagons have an outline?
alpha  the alpha transparency of the hexagons between 0 (transparent) and 1 (opaque)
hover  logical - should a hover tool be added to show the count in each hexagon?

Examples

figure() %>% ly_hexbin(rnorm(10000), rnorm(10000))
**ly_hist**

Add a "hist" layer to a Bokeh figure

---

### Description

Draws a histogram

### Usage

```
ly_hist(
    fig,
    x,
    data = figure_data(fig),
    breaks = "Sturges",
    freq = TRUE,
    include.lowest = TRUE,
    right = TRUE,
    color = NULL,
    alpha = 1,
    lname = NULL,
    lgroup = NULL,
    ...
)
```

### Arguments

- **fig**: figure to modify
- **x**: either a vector to be passed to `hist` or an object of class "histogram"
- **data**: an optional data frame, providing the source for x
- **breaks**, **freq**, **include.lowest**, **right**: parameters passed to `hist`
- **color**: color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below
- **alpha**: the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below
- **lname**: layer name
- **lgroup**: layer group
- **...**: additional parameters for fine control over fill and line properties (see "Additional parameters" below)
Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the `fill_color` and `line_color` attributes can be specified explicitly and will override `color`.

When `color` is `NULL` and `fill_color` or `line_color` are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with `fill_alpha` and `line_alpha` and will override alpha.

Additional parameters

- `fill_color` color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
- `fill_alpha` transparency value between 0 (transparent) and 1 (opaque)
- `line_color` color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
- `line_width` stroke width in units of pixels
- `line_alpha` transparency value between 0 (transparent) and 1 (opaque)
- `line_join` how path segments should be joined together 'miter' 'round' 'bevel'
- `line_cap` how path segments should be terminated 'butt' 'round' 'square'
- `line_dash` array of integer pixel distances that describe the on-off pattern of dashing to use
- `line_dash_offset` the distance in pixels into the `line_dash` that the pattern should start from

See Also


Examples

```r
h <- figure(width = 600, height = 400) %>%
  ly_hist(eruptions, data = faithful, breaks = 40, freq = FALSE) %>%
```
ly_image

ly_density(eruptions, data = faithful)

Add an "image" layer to a Bokeh figure

Description

Draws a grid of rectangles with colors corresponding to the values in z

Usage

ly_image(
  fig, 
  z, 
  rows, 
  byrow = TRUE, 
  x = 0, 
  y = 0, 
  dw = 1, 
  dh = 1, 
  palette = "Spectral10", 
  dilate = FALSE, 
  lname = NULL, 
  lgroup = NULL
)

Arguments

fig      figure to modify
z        matrix or vector of image values
rows     if z is a vector, how many rows should be used in treating it as a matrix
byrow    if z is a vector, should it be turned into a matrix by row
x        lower left x coordinates
y        lower left y coordinates
dw       image width distances
dh       image height distances
palette   name of color palette to use for color ramp (see here for acceptable values)
dilate   logical - whether to dilate pixel distance computations when drawing
lname    layer name
lgroup   layer group
See Also


Examples

```r
p <- figure(xlim = c(0, 1), ylim = c(0, 1), title = "Volcano") %>%
  ly_image(volcano) %>%
  ly_contour(volcano)
```

---

**ly_image_url**  
Add an "image_url" layer to a Bokeh figure

Description

Renders raster images from URLs at provided coordinates

Usage

```r
ly_image_url(
  fig,
  x = 0,
  y = 0,
  data = figure_data(fig),
  w = 10,
  h = 10,
  image_url,
  dilate = TRUE,
  anchor = "top_left",
  angle = 0,
  lname = NULL,
  lgroup = NULL
)
```

Arguments

- `fig`  
  figure to modify
- `x`  
  x coordinates
- `y`  
  y coordinates
- `data`  
  an optional data frame, providing the source for inputs x, y, and other properties
ly_lines

w, h  values or field names of width and height of image
image_url values or field name of image URLs
dilate logical - whether to dilate pixel distance computations when drawing
anchor where the image is anchored to with respect to x and y
angle values or field name of the angle to rotate the image, in radians
lname layer name
lgroup layer group

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_arc(), ly_bar(),
ly_bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(),
ly_image(), ly_lines(), ly_map(), ly_multi_line(), ly_oval(), ly_patch(), ly_points(),
ly_polygons(), ly_quadratic(), ly_quantile(), ly_ray(), ly_rect(), ly_segments(), ly_text(),
ly_wedge()

Examples

         "http://developer.r-project.org/Logo/Rlogo-4.png")

ss <- seq(0, 2*pi, length = 13)[-1]
ws <- runif(12, 2.5, 5) * rep(c(1, 0.8), 6)

imgdat <- data.frame(
  x = sin(ss) * 10, y = cos(ss) * 10,
  w = ws, h = ws * rep(c(1, 0.76), 6),
  url = rep(url, 6)
)

p <- figure(xlab = "x", ylab = "y") %>%
  ly_image_url(x, y, w = w, h = h, image_url = url, data = imgdat,
               anchor = "center") %>%
  ly_lines(sin(c(ss, ss[1])) * 10, cos(c(ss, ss[1])) * 10,
            width = 15, alpha = 0.1)
p

ly_lines  Add a "lines" layer to a Bokeh figure Draws lines with the given coordinates.

Description

Add a "lines" layer to a Bokeh figure. Draws lines with the given coordinates.
Usage

```r
ly_lines(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  group = NULL,
  color = "black",
  type = 1,
  width = 1,
  alpha = 1,
  legend = NULL,
  lname = NULL,
  lgroup = NULL,
  ...
)
```

Arguments

- **fig** figure to modify
- **x** values or field name of line x coordinates
- **y** values or field name of line y coordinates
- **data** an optional data frame, providing the source for inputs x, y, and other glyph properties
- **group** values or field name of a grouping variable to break lines up by
- **color** color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
- **type** an integer between 1 and 6 matching the lty property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- **width** stroke width in units of pixels
- **alpha** transparency value for the line between 0 (transparent) and 1 (opaque)
- **legend** either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in `data`, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
- **lname** layer name
- **lgroup** layer group
- **...** additional parameters for fine control over line properties (see "Additional parameters" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the `data` argument, columns of `data` can be used to specify various plot attributes such as color, etc. For example, with `ly_points(..., data = iris, color = Species)`, the `Species` variable is used to determine how to color the points.
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

### Additional parameters

- `line_join`: how path segments should be joined together 'miter' 'round' 'bevel'
- `line_cap`: how path segments should be terminated 'butt' 'round' 'square'
- `line_dash`: an integer between 1 and 6 matching the lty property in `par` or an array of integer pixel distances that describe the on-off pattern of dashing to use
- `line_dash_offset`: the distance in pixels into the line_dash that the pattern should start from

### See Also


### Examples

```r
z <- lm(dist ~ speed, data = cars)
p <- figure() %>%
  ly_points(cars, hover = cars) %>%
  ly_lines(lowess(cars), legend = "lowess") %>%
  ly_abline(z, type = 2, legend = "lm", width = 2)
p
```

---

### ly_map

Add a "map" layer to a Bokeh figure

### Description

Draws lines and polygons as specified by a map database

### Usage

```r
ly_map(
  fig,
  database = "world",
  regions = ".",
  color = NULL,
)```
alpha = 1, 
lname = NULL, 
lgroup = NULL, 
...
)

Arguments

fig figure to modify
database, regions parameters passed to map
color color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below
alpha the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below
lname layer name
lgroup layer group
... additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

• When using a glyph that only has line properties, this will be the color of the line.
• When using a glyph that has has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
• If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

• When using a glyph that only has line properties, this will be the alpha of the line.
• When using a glyph that has has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
• Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.
ly_multi_line

Additional parameters

fill_color  color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green'
fill_alpha  transparency value between 0 (transparent) and 1 (opaque)
line_color  color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green'
line_width  stroke width in units of pixels
line_alpha  transparency value between 0 (transparent) and 1 (opaque)
line_join  how path segments should be joined together 'miter' 'round' 'bevel'
line_cap  how path segments should be terminated 'butt' 'round' 'square'
line_dash  array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset  the distance in pixels into the line_dash that the pattern should start from

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_arc(), ly_bar(), ly_bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(), ly_image_url(), ly_image(), ly_lines(), ly_multi_line(), ly_oval(), ly_patch(), ly_points(), ly_polygons(), ly_quadratic(), ly_quantile(), ly_ray(), ly_rect(), ly_segments(), ly_text(), ly_wedge()

ly_multi_line  Add a "multi_line" layer to a Bokeh figure

Description

Draws multiple lines with the given lists of coordinates.

Usage

ly_multi_line(
    fig,
    xs,
    ys,
    color = "black",
    alpha = 1,
    width = 1,
    type = 1,
    lname = NULL,
    lgroup = NULL,
    ...
Arguments

fig figure to modify
xs list of vectors of x coordinates
ys list of vectors of y coordinates
color color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
alpha transparency value for the line between 0 (transparent) and 1 (opaque)
width stroke width in units of pixels
type an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
lname layer name
lgroup layer group
... additional parameters for fine control over line properties (see "Additional parameters" below)

Additional parameters

line_join how path segments should be joined together 'miter' 'round' 'bevel'
line_cap how path segments should be terminated 'butt' 'round' 'square'
line_dash an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset the distance in pixels into the line_dash that the pattern should start from

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_arc(), ly_bar(), ly bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(), ly_image_url(), ly_image(), ly_lines(), ly_map(), lyoval(), ly_patch(), ly_points(), ly_polygons(), ly_quadratic(), ly_quantile(), ly_ray(), ly_rect(), ly_segments(), ly_text(), ly wedge()
**ly_oval**

### Usage

```r
ly_oval(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  width = 0.1,
  height = 0.1,
  angle = 0,
  color = NULL,
  alpha = 1,
  legend = NULL,
  lname = NULL,
  lgroup = NULL,
  ...
)
```

### Arguments

- **fig**: figure to modify
- **x**: values or field name of center x coordinates
- **y**: values or field name of center y coordinates
- **data**: an optional data frame, providing the source for inputs x, y, and other glyph properties
- **width**: values or field name of widths
- **height**: values or field name of heights
- **angle**: values or field name of rotation angles
- **color**: color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below
- **alpha**: the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below
- **legend**: either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
- **lname**: layer name
- **lgroup**: layer group
- **...**: additional parameters for fine control over fill and line properties (see "Additional parameters" below)

### Handling color

The `color` parameter is a high-level plot attribute that provides default behavior for coloring glyphs.
• When using a glyph that only has line properties, this will be the color of the line.
• When using a glyph that has line and fill properties, this will be the color of the line and
  the fill, with the alpha level of the fill reduced by 50%.
• If full control over fill and line color is desired, the fill_color and line_color attributes
  can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen
from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

• When using a glyph that only has line properties, this will be the alpha of the line.
• When using a glyph that has line and fill properties, this will be the alpha of the line and
  the alpha of the fill will be set to 50% of this value.
• Individual fill and line alpha can be specified with fill_alpha and line_alpha and will
  override alpha.

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can
be used to specify various plot attributes such as color, etc. For example, with ly_points(...,
data = iris, color = Species), the Species variable is used to determine how to color the points.
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be
mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to
each interval. The mapping from the values of the variable to the actual plot attributes is determined
based on the theme.

Additional parameters

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill_color</td>
<td>color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'</td>
</tr>
<tr>
<td>fill_alpha</td>
<td>transparency value between 0 (transparent) and 1 (opaque)</td>
</tr>
<tr>
<td>line_color</td>
<td>color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'</td>
</tr>
<tr>
<td>line_width</td>
<td>stroke width in units of pixels</td>
</tr>
<tr>
<td>line_alpha</td>
<td>transparency value between 0 (transparent) and 1 (opaque)</td>
</tr>
<tr>
<td>line_join</td>
<td>how path segments should be joined together 'miter' 'round' 'bevel'</td>
</tr>
<tr>
<td>line_cap</td>
<td>how path segments should be terminated 'butt' 'round' 'square'</td>
</tr>
<tr>
<td>line_dash</td>
<td>array of integer pixel distances that describe the on-off pattern of dashing to use</td>
</tr>
<tr>
<td>line_dash_offset</td>
<td>the distance in pixels into the line_dash that the pattern should start from</td>
</tr>
</tbody>
</table>

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_arc(), ly_bar(),
ly_bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(),
ly_image_url(), ly_image(), ly_lines(), ly_map(), ly_multi_line(), ly_patch(), ly_points(),
**ly_patch**

Add a "patch" layer to a Bokeh figure

**Description**

Add a "patch" layer to a Bokeh figure

**Usage**

```
ly_patch(
    fig,
    x,
    y,
    data = figure_data(fig),
    color = NULL,
    alpha = 1,
    hover = NULL,
    url = NULL,
    legend = NULL,
    lname = NULL,
    lgroup = NULL,
    ...
)
```

**Arguments**

- `fig`  
  figure to modify

- `x`  
  values or field name of patch x coordinates

- `y`  
  values or field name of patch y coordinates

- `data`  
  an optional data frame, providing the source for inputs x, y, and other glyph properties

- `color`  
  color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below

- `alpha`  
  the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below

- `hover`  
  a data frame of variables to be displayed when hovering over the glyph or a vector of variable names that can be found and extracted from the data argument

- `url`  
  a string of URLs or a single string that references a variable name (via @var_name) that can be found and extracted from the data argument
ly_patch

legend
    either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)

lname
    layer name

lgroup
    layer group

... additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

fill_color
    color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo'

fill_alpha
    transparency value between 0 (transparent) and 1 (opaque)

line_color
    color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo'

line_width
    stroke width in units of pixels
ly_points

Add a "points" layer to a Bokeh figure

Description

Add a "points" layer to a Bokeh figure

Usage

ly_points(
    fig,
    x,
    y = NULL,
    data = figure_data(fig),
    glyph = 21,
    color = NULL,
    alpha = 1,
    size = 10,
    hover = NULL,
    url = NULL,
    legend = NULL,
    lname = NULL,
    lgroup = NULL,
    ...  
)
Arguments

fig  
figure to modify

x  
values or field name of center x coordinates

y  
values or field name of center y coordinates

data  
an optional data frame, providing the source for inputs x, y, and other glyph properties

glyph  
value(s) or field name of the glyph to use (see point_types)

color  
color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below

alpha  
the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below

size  
size of the glyph in screen units

hover  
a data frame of variables to be displayed when hovering over the glyph or a vector of variable names that can be found and extracted from the data argument

url  
a string of URLs or a single string that references a variable name (via @var_name) that can be found and extracted from the data argument

legend  
either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)

lname  
layer name

lgroup  
layer group

...  
additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
• When using a glyph that has has line and fill properties, this will be the alpha of the line and
the alpha of the fill will be set to 50% of this value.
• Individual fill and line alpha can be specified with fill_alpha and line_alpha and will
override alpha.

Mapped plot attributes and legends
When specifying an input data frame for a layer through the data argument, columns of data can
be used to specify various plot attributes such as color, etc. For example, with `ly_points(...,
data = iris, color = Species)`, the Species variable is used to determine how to color the points.
Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be
mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to
each interval. The mapping from the values of the variable to the actual plot attributes is determined
based on the theme.

Additional parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| fill_color        | color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g.
| fill_alpha        | transparency value between 0 (transparent) and 1 (opaque)                     |
| line_color        | color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g.
| line_width        | stroke width in units of pixels                                             |
| line_alpha        | transparency value between 0 (transparent) and 1 (opaque)                     |
| line_join         | how path segments should be joined together 'miter' 'round' 'bevel'           |
| line_cap          | how path segments should be terminated 'butt' 'round' 'square'               |
| line_dash         | array of integer pixel distances that describe the on-off pattern of dashing to use |
| line_dash_offset  | the distance in pixels into the line_dash that the pattern should start from |

See Also
Other layer functions: `ly_abline()`, `ly_annular_wedge()`, `ly_annulus()`, `ly_arc()`, `ly_bar()`,
`ly_bezier()`, `ly_boxplot()`, `ly_contour()`, `ly_crect()`, `ly_curve()`, `ly_density()`, `ly_hist()`,
`ly_image_url()`, `ly_image()`, `ly_lines()`, `ly_map()`, `ly_multi_line()`, `ly_oval()`, `ly_patch()`,
`ly_polygons()`, `ly_quadratic()`, `ly_quantile()`, `ly_ray()`, `ly_rect()`, `ly_segments()`, `ly_text()`,
`ly_wedge()`

Examples

```r
figure() %>%
  ly_points(Sepal.Length, Sepal.Width, data = iris,
            color = Species, glyph = Species,
            hover = list(Sepal.Length, Sepal.Width))

# custom hover
mtcars$model <- row.names(mtcars)
figure() %>%
  ly_points(disp, mpg, data = mtcars, color = cyl,
```
hover = "This <strong>@model</strong><br>has @hp horsepower!"

z <- lm(dist ~ speed, data = cars)
p <- figure() %>%
  ly_points(cars, hover = cars) %>%
  ly_lines(lowess(cars), legend = "lowess") %>%
  ly_abline(z, type = 2, legend = "lm", width = 2)
p

ly_polygons

Add a "polygons" layer to a Bokeh figure

Description

Add a "polygons" layer to a Bokeh figure

Usage

ly_polygons(
  fig,
  xs,
  ys,
  group = NULL,
  data = figure_data(fig),
  color = NULL,
  alpha = 1,
  hover = NULL,
  url = NULL,
  lname = NULL,
  lgroup = NULL,
  ...
)

Arguments

fig figure to modify
xs vector or list of values or field name of polygon x coordinates - see details
ys vector or list of values or field name of polygon y coordinates - see details
group vector or field name of grouping variable - see details
data an optional data frame, providing the source for inputs xs, ys, group, and other
glyph properties
color color for the glyph - a hex code (with no alpha) or any of the 147 named CSS
colors, e.g ‘green’, ‘indigo’ - for glyphs with both fill and line properties, see
"Handling color" below
ly_polygons

alpha  the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below
hover  a data frame of variables to be displayed when hovering over the glyph or a vector of variable names that can be found and extracted from the data argument
url    a string of URLs or a single string that references a variable name (via @var_name) that can be found and extracted from the data argument
lname  layer name
lgroup layer group
...    additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Details

gxs and ys can be a list of vectors, each element for one polygon to be drawn, or can be vectors with the group argument specifying how to break them up into individual polygons.

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.

Additional parameters

color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
fill_alpha transparency value between 0 (transparent) and 1 (opaque)
color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
line_width stroke width in units of pixels
transparency value between 0 (transparent) and 1 (opaque)
how path segments should be joined together 'miter' 'round' 'bevel'
ly_quadratic

Add a "quadratic" layer to a Bokeh figure

Description

Draws quadratic curves with the given starting, ending, and control points.

Usage

ly_quadratic(
    fig,
    x0,
    y0,
    x1,
    y1,
    cx,
    cy,
    data = figure_data(fig),
    color = "black",
    alpha = 1,
    width = 1,
    type = 1,
    legend = NULL,
    lname = NULL,
    lgroup = NULL,
    ...
)

Arguments

fig figure to modify
x0 values or field name of starting x coordinates
ly_quadratic

y0: values or field name of starting y coordinates
x1: values or field name of ending x coordinates
y1: values or field name of ending y coordinates
cx: values or field name of control point x coordinates
cy: values or field name of control point y coordinates
data: an optional data frame, providing the source for start, end, and control point inputs, as well as other glyph properties
color: color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
alpha: transparency value for the line between 0 (transparent) and 1 (opaque)
width: stroke width in units of pixels
type: an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
legend: either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
lname: layer name
lgroup: layer group
...
additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

fill_color: color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green'
fill_alpha: transparency value between 0 (transparent) and 1 (opaque)
line_color: color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green'
line_width: stroke width in units of pixels
line_alpha: transparency value between 0 (transparent) and 1 (opaque)
line_join: how path segments should be joined together 'miter' 'round' 'bevel'
line_cap: how path segments should be terminated 'butt' 'round' 'square'
line_dash: array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset: the distance in pixels into the line_dash that the pattern should start from
ly_quantile

Add a "quantile" layer to a Bokeh figure

Description

Draws quantiles

Usage

ly_quantile(
fig,
  x,
  group = NULL,
  data = figure_data(fig),
  probs = NULL,
  distn = stats::qunif,
  ncutoff = 200,
  color = NULL,
  alpha = 1,
  legend = TRUE,
  lname = NULL,
  lgroup = NULL,
  ...
)

Arguments

fig            figure to modify
x              numeric vector or field name of variable to compute sample quantiles for
group          values or field name of a grouping variable to break quantile computations up by
data            an optional data frame, providing the source for x
probs           numeric vector of probabilities with values in [0,1] at which to compute quantiles - if NULL, every point of x is a quantile
distn           quantile function to use on the x-axis (e.g. qnorm) - default is qunif.

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_arc(), ly_bar(),
ly bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(),
ly_image_url(), ly_image(), ly_lines(), ly_map(), ly_multi_line(), ly_oval(), ly_patch(),
ly_points(), ly_polygons(), ly_quantile(), ly_ray(), ly_rect(), ly_segments(), ly_text(),
ly_wedge()
ncutoff if the length of x exceeds this value and probs is not specified, compute quantiles at ncutoff points

color color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below

alpha the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below

legend either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)

lname layer name

lgroup layer group

... additional parameters for fine control over fill and line properties (see "Additional parameters" below)

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

• When using a glyph that only has line properties, this will be the color of the line.
• When using a glyph that has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
• If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

• When using a glyph that only has line properties, this will be the alpha of the line.
• When using a glyph that has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
• Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.
**ly_ray**

**Additional parameters**

- **fill_color**: color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo'
- **fill_alpha**: transparency value between 0 (transparent) and 1 (opaque)
- **line_color**: color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g. 'green', 'indigo'
- **line_width**: stroke width in units of pixels
- **line_alpha**: transparency value between 0 (transparent) and 1 (opaque)
- **line_join**: how path segments should be joined together 'miter' 'round' 'bevel'
- **line_cap**: how path segments should be terminated 'butt' 'round' 'square'
- **line_dash**: array of integer pixel distances that describe the on-off pattern of dashing to use
- **line_dash_offset**: the distance in pixels into the line_dash that the pattern should start from

**See Also**


**Examples**

```r
figure(legend_location = "top_left") %>%
  ly_quantile(Sepal.Length, group = Species, data = iris)
```

**Description**

Draws line segments starting at the given coordinate and extending the given length at the given angle.

**Usage**

```r
ly_ray(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  length = NULL,
  angle = 0,
  color = "black",
  type = 1,
  width = 1,
)```
ly_ray

alpha = NULL,
legend = NULL,
lname = NULL,
lgroup = NULL,
...
)

Arguments

fig figure to modify
x values or field name of center x coordinates
y values or field name of center y coordinates
data an optional data frame, providing the source for inputs x, y, and other glyph properties
length values or field name of ray lengths in screen units
angle values or field name of ray angles
color color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
type an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
width stroke width in units of pixels
alpha transparency value for the line between 0 (transparent) and 1 (opaque)
legend either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
lname layer name
lgroup layer group
... additional parameters for fine control over line properties (see "Additional parameters" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

line_join how path segments should be joined together 'miter' 'round' 'bevel'
line_cap how path segments should be terminated 'butt' 'round' 'square'
line_dash an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset the distance in pixels into the line_dash that the pattern should start from
See Also


---

```r
ly_rect(fig, xleft, ybottom, xright, ytop, data = figure_data(fig), color = NULL, alpha = 1, hover = NULL, url = NULL, legend = NULL, lname = NULL, lgroup = NULL, ...
)
```

Description

Add a "rect" layer to a Bokeh figure

Usage

```r
ly_rect(
  fig, xleft, ybottom, xright, ytop,
  data = figure_data(fig),
  color = NULL,
  alpha = 1,
  hover = NULL,
  url = NULL,
  legend = NULL,
  lname = NULL,
  lgroup = NULL,
  ...
)
```

Arguments

- `fig` figure to modify
- `xleft` values or field name of left edges
- `ybottom` values or field name of bottom edges
- `xright` values or field name of right edges
- `ytop` values or field name of top edges
- `data` an optional data frame, providing the source for inputs xleft, ybottom, xright, ytop, and other glyph properties
ly_rect

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see &quot;Handling color&quot; below</td>
</tr>
<tr>
<td>alpha</td>
<td>the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see &quot;Handling alpha&quot; below</td>
</tr>
<tr>
<td>hover</td>
<td>a data frame of variables to be displayed when hovering over the glyph or a vector of variable names that can be found and extracted from the data argument</td>
</tr>
<tr>
<td>url</td>
<td>a string of URLs or a single string that references a variable name (via @var_name) that can be found and extracted from the data argument</td>
</tr>
<tr>
<td>legend</td>
<td>either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see &quot;Mapped plot attributes and legends&quot; below)</td>
</tr>
<tr>
<td>lname</td>
<td>layer name</td>
</tr>
<tr>
<td>lgroup</td>
<td>layer group</td>
</tr>
<tr>
<td>...</td>
<td>additional parameters for fine control over fill and line properties (see &quot;Additional parameters&quot; below)</td>
</tr>
</tbody>
</table>

Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be
mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill_color</td>
<td>color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'</td>
</tr>
<tr>
<td>fill_alpha</td>
<td>transparency value between 0 (transparent) and 1 (opaque)</td>
</tr>
<tr>
<td>line_color</td>
<td>color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'</td>
</tr>
<tr>
<td>line_width</td>
<td>stroke width in units of pixels</td>
</tr>
<tr>
<td>line_alpha</td>
<td>transparency value between 0 (transparent) and 1 (opaque)</td>
</tr>
<tr>
<td>line_join</td>
<td>how path segments should be joined together 'miter' 'round' 'bevel'</td>
</tr>
<tr>
<td>line_cap</td>
<td>how path segments should be terminated 'butt' 'round' 'square'</td>
</tr>
<tr>
<td>line_dash</td>
<td>array of integer pixel distances that describe the on-off pattern of dashing to use</td>
</tr>
<tr>
<td>line_dash_offset</td>
<td>the distance in pixels into the line_dash that the pattern should start from</td>
</tr>
</tbody>
</table>

See Also

Other layer functions: ly_abline(), ly_annular_wedge(), ly_annulus(), ly_arc(), ly_bar(), ly_bezier(), ly_boxplot(), ly_contour(), ly_crect(), ly_curve(), ly_density(), ly_hist(), ly_image_url(), ly_image(), ly_lines(), ly_map(), ly_multi_line(), ly_oval(), ly_patch(), ly_points(), ly_polygons(), ly_quadratic(), ly_quantile(), ly_ray(), ly_segments(), ly_text(), ly_wedge()

---

ly_segments

Add a "segments" layer to a Bokeh figure

Description

Draws line segments with the given starting and ending coordinates.

Usage

```r
ly_segments(
  fig,
  x0, y0, x1, y1,
  data = figure_data(fig),
  color = "black",
  alpha = 1,
  width = 1,
  type = 1,
)```

ly_segments

legend = NULL,
lname = NULL,
lgroup = NULL,
...
)

Arguments

fig  figure to modify
x0   values or field name of starting x coordinates
y0   values or field name of starting y coordinates
x1   values or field name of ending x coordinates
y1   values or field name of ending y coordinates
data  an optional data frame, providing the source for inputs x, y, and other glyph properties
color color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo'
alpha transparency value for the line between 0 (transparent) and 1 (opaque)
width stroke width in units of pixels
type  an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
legend either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
lname  layer name
lgroup layer group
...  additional parameters for fine control over line properties (see "Additional parameters" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

line_join how path segments should be joined together 'miter' 'round' 'bevel'
line_cap how path segments should be terminated 'butt' 'round' 'square'
line_dash an integer between 1 and 6 matching the lty property in par or an array of integer pixel distances that describe the on-off pattern of dashing to use
line_dash_offset the distance in pixels into the line_dash that the pattern should start from
See Also

Other layer functions: `ly_abline()`, `ly_annular_wedge()`, `ly_annulus()`, `ly_arc()`, `ly_bar()`,
`ly_bezier()`, `ly_boxplot()`, `ly_contour()`, `ly_crect()`, `ly_curve()`, `ly_density()`, `ly_hist()`,
`ly_image_url()`, `ly_image()`, `ly_lines()`, `ly_map()`, `ly_multi_line()`, `ly_oval()`, `ly_patch()`,
`ly_points()`, `ly_polygons()`, `ly_quadratic()`, `ly_quantile()`, `ly_ray()`, `ly_rect()`, `ly_text()`,
`ly_wedge()`

---

**ly_text**

Add a "text" layer to a Bokeh figure

**Description**

Add a "text" layer to a Bokeh figure

**Usage**

```r
ly_text(
  fig,
  x, y = NULL,
  text = NULL,
  data = figure_data(fig),
  color = "black",
  alpha = 1,
  angle = 0,
  align = NULL,
  baseline = NULL,
  font = NULL,
  font_size = NULL,
  font_style = NULL,
  x_offset = NULL,
  y_offset = NULL,
  legend = NULL,
  lname = NULL,
  lgroup = NULL
)
```

**Arguments**

- `fig` figure to modify
- `x` x coordinates of text anchors
- `y` y coordinates of text anchors
- `text` text values to render
ly_text

an optional data frame, providing the source for inputs x, y, text, and other glyph properties

color
text color values for the text
alpha
text alpha values for the text
angle
angle to rotate the text in radians
align
text align values for the text ("left", "right", "center")
baseline
text baseline values for the text ("top", "middle", "bottom", "alphabetic", "hanging")
font
text font values for the text
font_size
text font size values for the text
font_style
text font style values for the text ("normal", "italic", "bold")
x_offset
offset values to apply to the x-coordinates
y_offset
offset values to apply to the y-coordinates
legend
whether a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with `ly_points(..., data = iris, color = Species)`, the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

See Also


Examples

```r
# prepare data
elements <- subset(elements, !is.na(group))
elements$group <- as.character(elements$group)
elements$period <- as.character(elements$period)
```
ly_wedge

Add a "wedge" layer to a Bokeh figure

Description

Add a "wedge" layer to a Bokeh figure.

# add colors for groups
metals <- c("alkali metal", "alkaline earth metal", "halogen", "metal", "metalloid", "noble gas", "nonmetal", "transition metal")
colors <- c("#a6cee3", "#1f78b4", "#ff7f0e", "#b2df8a", "#33a02c", "#bbbb88", "#bab5a6", "#e08e79")
elements$color <- colors[match(elements$metal, metals)]
elements$type <- elements$metal

# make coordinates for labels
elements$symx <- paste(elements$group, ":0.1", sep = "")
elements$numbery <- paste(elements$period, ":0.8", sep = "")
elements$massy <- paste(elements$period, ":0.15", sep = "")
elements$namey <- paste(elements$period, ":0.3", sep = "")

# create figure
p <- figure(title = "Periodic Table", tools = ",
ylim = as.character(c(7:1)), xlim = as.character(1:18),
xgrid = FALSE, ygrid = FALSE, xlab = "", ylab = "",
height = 600, width = 1200) %>%

# plot rectangles
ly_crect(group, period, data = elements, 0.9, 0.9,
fill_color = color, line_color = color, fill_alpha = 0.6,
hover = list(name, atomic.number, type, atomic.mass,
electronic.configuration)) %>%

# add symbol text
ly_text(symx, period, text = symbol, data = elements,
font_style = "bold", font_size = "15pt",
align = "left", baseline = "middle") %>%

# add atomic number text
ly_text(symx, numbery, text = atomic.number, data = elements,
font_size = "9pt", align = "left", baseline = "middle") %>%

# add name text
ly_text(symx, namey, text = name, data = elements,
font_size = "6pt", align = "left", baseline = "middle") %>%

# add atomic mass text
ly_text(symx, massy, text = atomic.mass, data = elements,
font_size = "6pt", align = "left", baseline = "middle")

p
ly_wedge

Usage

ly_wedge(
  fig,
  x,
  y = NULL,
  data = figure_data(fig),
  radius = 0.3,
  start_angle = 0,
  end_angle = 2 * pi,
  direction = "anticlock",
  color = NULL,
  alpha = 1,
  hover = NULL,
  url = NULL,
  legend = NULL,
  lname = NULL,
  lgroup = NULL,
  ...
)

Arguments

fig figure to modify
x values or field name of center x coordinates
y values or field name of center y coordinates
data an optional data frame, providing the source for inputs x, y, and other glyph properties
radius values or field name of wedge radii
start_angle the angles to start the wedges, in radians, as measured from the horizontal
direction the angles to end the wedges, in radians, as measured from the horizontal
direction direction to turn between starting and ending angles ("anticlock", "clock")
color color for the glyph - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green', 'indigo' - for glyphs with both fill and line properties, see "Handling color" below
alpha the alpha transparency of the glyph between 0 (transparent) and 1 (opaque) - if glyph has both fill and color properties, see "Handling alpha" below
hover a data frame of variables to be displayed when hovering over the glyph or a vector of variable names that can be found and extracted from the data argument
url a string of URLs or a single string that references a variable name (via @var_name) that can be found and extracted from the data argument
legend either a logical specifying not to plot a legend for this layer (FALSE) or a string indicating the name of the legend entry for this layer (note that when mapping plot attributes to variables in data, a legend is automatically created and does not need to be specified - see "Mapped plot attributes and legends" below)
Handling color

The color parameter is a high-level plot attribute that provides default behavior for coloring glyphs.

- When using a glyph that only has line properties, this will be the color of the line.
- When using a glyph that has line and fill properties, this will be the color of the line and the fill, with the alpha level of the fill reduced by 50%.
- If full control over fill and line color is desired, the fill_color and line_color attributes can be specified explicitly and will override color.

When color is NULL and fill_color or line_color are not specified, the color will be chosen from the theme.

Handling alpha

The alpha is a high-level plot attribute that sets the transparency of the glyph being plotted.

- When using a glyph that only has line properties, this will be the alpha of the line.
- When using a glyph that has line and fill properties, this will be the alpha of the line and the alpha of the fill will be set to 50% of this value.
- Individual fill and line alpha can be specified with fill_alpha and line_alpha and will override alpha.

Mapped plot attributes and legends

When specifying an input data frame for a layer through the data argument, columns of data can be used to specify various plot attributes such as color, etc. For example, with ly_points(..., data = iris, color = Species), the Species variable is used to determine how to color the points. Here, Species is "mapped" to the color attribute. Both continuous and categorical variables can be mapped. In the case of continuous variables, the range is cut into slices and attributes are applied to each interval. The mapping from the values of the variable to the actual plot attributes is determined based on the theme.

Additional parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill_color</td>
<td>color to use to fill the glyph with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green'</td>
</tr>
<tr>
<td>fill_alpha</td>
<td>transparency value between 0 (transparent) and 1 (opaque)</td>
</tr>
<tr>
<td>line_color</td>
<td>color to use to stroke lines with - a hex code (with no alpha) or any of the 147 named CSS colors, e.g 'green'</td>
</tr>
<tr>
<td>line_width</td>
<td>stroke width in units of pixels</td>
</tr>
<tr>
<td>line_alpha</td>
<td>transparency value between 0 (transparent) and 1 (opaque)</td>
</tr>
<tr>
<td>line_join</td>
<td>how path segments should be joined together 'miter' 'round' 'bevel'</td>
</tr>
<tr>
<td>line_cap</td>
<td>how path segments should be terminated 'butt' 'round' 'square'</td>
</tr>
<tr>
<td>line_dash</td>
<td>array of integer pixel distances that describe the on-off pattern of dashing to use</td>
</tr>
<tr>
<td>line_dash_offset</td>
<td>the distance in pixels into the line_dash that the pattern should start from</td>
</tr>
</tbody>
</table>
nyctaxihex

See Also


Examples

```r
rescale <- function(x)
  (x - min(x)) / diff(range(x))
figure() %>%
  ly_wedge(Sepal.Length, Sepal.Width, data = iris,
    end_angle = rescale(Petal.Length) * 2 * pi, color = Species,
    radius = 0.15, alpha = 0.5,
    hover = Species)
```

nyctaxihex

Hexagon binned counts of NYC taxi pickup locations

Description

Counts of NYC taxi pickups by location for January 2013, obtained from [here](#).

Usage

nyctaxihex

Examples

```r
## Not run:
gmap(title = "NYC taxi pickups January 2013",
  lat = 40.74, lng = -73.95, zoom = 11,
  map_type = "roadmap", width = 1000, height = 800) %>%
  ly_hexbin(nyctaxihex, alpha = 0.5,
    palette = "Spectral10", trans = log, inv = exp)
## End(Not run)
```
phantom_install

Description

Instructions for installing phantomjs

Usage

phantom_install()

describepal_color

Palettes for themes

Description

Palettes for themes

Usage

pal_color(colors)
pal_tableau(pal = "Tableau10")
pal_bk_glyph()
pal_gradient(
cols = c("#66C2A4", "#41AE76", "#238B45", "#006D2C", "#00441B"),
space = "rgb"
)
pal_size(min = 2, max = 20)
pal_bk_line_dash()
pal_bk_line_width()

Arguments

colors | a vector of colors to be used in the color palette
pal | palette name
cols | a vector of colors to be used in the color palette
space | passed on to colorRampPalette[grDevices]
min | minimum value
max | maximum value

phantom_install

Instructions for installing phantomjs

Description

Instructions for installing phantomjs

Usage

phantom_install()
**point_types**

Display glyph types available for `ly_points()`

**Usage**

```r
point_types(size = 25, color = "blue", width = 800, height = 450)
```

**Arguments**

- `size`: size of the glyph
- `color`: color to use for line and fill properties
- `width`, `height`: dimensions of output plot

**Examples**

```r
point_types()
```

---

**print_model_json**

Print the JSON of a Bokeh figure

**Description**

Print the JSON of a Bokeh figure

**Usage**

```r
print_model_json(fig, prepare = TRUE, pretty = TRUE, file = "", pbcopy = FALSE)
```

**Arguments**

- `fig`: figure to print
- `prepare`: logical - should the figure be sent through preparations that need to be done prior to plotting (TRUE), or printed as-is (FALSE)
- `pretty`: parameter passed on to `toJSON`
- `file`: parameter passed on to `cat`
- `pbcopy`: logical - if on OSX, should the results be passed to the clipboard (TRUE) instead of printed to the screen (FALSE)?
Examples

```r
# Not run:
p <- figure() %>% ly_points(1:10) %>%
  tool_pan(dimensions = "height")
print_model_json(p)

# End(Not run)
```

---

**rbokeh2html**

*Get the HTML content required to embed a Bokeh figure*

### Description

Get the HTML content required to embed a Bokeh figure

### Usage

```r
rbokeh2html(
  fig, 
  file = tempfile(fileext = ".html"),
  pretty = FALSE,
  secure = TRUE
)
```

### Arguments

- `fig`: figure
- `file`: html file name to write the figure to
- `pretty`: should the json model be pretty printed to the html file?
- `secure`: should https be used for cdn links?

### Examples

```r
p <- figure() %>% ly_points(1:10)
rbokeh2html(p)
```
rbokehOutput

Widget output function for use in Shiny

Description

Widget output function for use in Shiny

Usage

rbokehOutput(outputId, width = "100\%", height = "400px")

Arguments

- outputId: output variable to read from
- width: a valid CSS unit for the width or a number, which will be coerced to a string and have "px" appended.
- height: a valid CSS unit for the height or a number, which will be coerced to a string and have "px" appended.

Examples

```r
## Not run:
library("shiny")
library("rbokeh")

ui <- fluidPage(
  rbokehOutput("rbokeh")
)

server <- function(input, output, session) {
  output$rbokeh <- renderRbokeh({
    # Use invalidateLater() and jitter() to add some motion
    invalidateLater(1000, session)
    figure() %>%
      ly_points(jitter(cars$speed), jitter(cars$dist))
  })
}

shinyApp(ui, server)
```

```r
library("shiny")
library("rbokeh")

ui <- fluidPage(
  rbokehOutput("rbokeh", width = 500, height = 540),
  textOutput("x_range_text")
)
```
server <- function(input, output, session) {
  output$rbokeh <- renderRbokeh({
    figure() %>% ly_points(1:10) %>%
    x_range(callback = shiny_callback("x_range"))
  })

  output$x_range_text <- reactive({
    xrng <- input$x_range
    if(!is.null(xrng)) {
      paste0("factors: ", xrng$factors, ", start: ", xrng$start,
        ", end: ", xrng$end)
    } else {
      "waiting for axis event..."
    }
  })
}

shinyApp(ui, server)

## End(Not run)

---

**renderRbokeh**  
*(Widget render function for use in Shiny)*

**Description**

Widget render function for use in Shiny

**Usage**

renderRbokeh(expr, env = parent.frame(), quoted = FALSE)

**Arguments**

- `expr` an expression that generates a rbokeh figure
- `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.

**See Also**

rbokehOutput for an example in Shiny
set_palette

Set palettes for various plot attributes

Description

Set palettes for various plot attributes

Usage

```
set_palette(
  fig,
  discrete_color = NULL,
  discrete_alpha = NULL,
  continuous_color = NULL,
  continuous_alpha = NULL,
  discrete_glyph = NULL,
  discrete_fill_color = NULL,
  discrete_line_color = NULL,
  discrete_text_color = NULL,
  discrete_fill_alpha = NULL,
  discrete_line_alpha = NULL,
  discrete_text_alpha = NULL,
  discrete_line_dash = NULL,
  discrete_line_width = NULL,
  discrete_size = NULL,
  continuous_glyph = NULL,
  continuous_fill_color = NULL,
  continuous_line_color = NULL,
  continuous_text_color = NULL,
  continuous_fill_alpha = NULL,
  continuous_line_alpha = NULL,
  continuous_text_alpha = NULL,
  continuous_line_dash = NULL,
  continuous_line_width = NULL,
  continuous_size = NULL
)
```

Arguments

- **fig**: figure to update theme palettes for
- **discrete_color**: a discrete color palette to override the theme (see details)
- **discrete_alpha**: a discrete alpha palette to override the theme (see details)
- **continuous_color**: a continuous color palette to override the theme (see details)
- **continuous_alpha**: a continuous alpha palette to override the theme (see details)
set_palette

discrete_glyph  a discrete glyph palette to override the theme
discrete_fill_color  a discrete fill_color palette to override the theme
discrete_line_color  a discrete line_color palette to override the theme
discrete_text_color  a discrete text_color palette to override the theme
discrete_fill_alpha  a discrete fill_alpha palette to override the theme
discrete_line_alpha  a discrete line_alpha palette to override the theme
discrete_text_alpha  a discrete text_alpha palette to override the theme
discrete_line_dash  a discrete line_dash palette to override the theme
discrete_line_width  a discrete line_width palette to override the theme
discrete_size  a discrete size palette to override the theme
continuous_glyph  a continuous glyph palette to override the theme
continuous_fill_color  a continuous fill_color palette to override the theme
continuous_line_color  a continuous line_color palette to override the theme
continuous_text_color  a continuous text_color palette to override the theme
continuous_fill_alpha  a continuous fill_alpha palette to override the theme
continuous_line_alpha  a continuous line_alpha palette to override the theme
continuous_text_alpha  a continuous text_alpha palette to override the theme
continuous_line_dash  a continuous line_dash palette to override the theme
continuous_line_width  a continuous line_width palette to override the theme
continuous_size  a continuous size palette to override the theme

details

Palettes specified in this function will override the existing theme and apply the specified attributes when they are not otherwise explicitly specified in a layer function. See the contents of bk_default_theme for an example of the theme elements this will update. As a convenience, if you use discrete_color, the palette will apply to all the discrete_***_color attributes unless those
are explicitly specified also. The same pattern is true for discrete_alpha, continuous_color, and continuous_alpha. For specifying discrete color palettes, the easiest thing to do is use \texttt{pal_color} with a vector of colors you want to use in the palette.

\textbf{Examples}

```r
figure() %>%
  ly_points(Sepal.Length, Sepal.Width, data = iris,
            color = Species, glyph = Species) %>%
  set_palette(discrete_color = pal_color(c("red", "blue", "green")))
```

\textbf{Description}

Set the theme for a figure

\textbf{Usage}

```
set_theme(fig, theme)
```

\textbf{Arguments}

- \texttt{fig}: a figure to set the theme for
- \texttt{theme}: theme

\textbf{Examples}

```
# manually specify a ggplot-like grid and background
figure() %>%
  ly_points(1:10) %>%
  theme_plot(background_fill_color = "#E6E6E6",
             outline_line_color = "white") %>%
  theme_grid(c("x", "y"), grid_line_color = "white",
             minor_grid_line_color = "white",
             minor_grid_line_alpha = 0.4) %>%
  theme_axis(c("x", "y"), axis_line_color = "white",
             major_label_text_color = "#7F7F7F",
             major_tick_line_color = "#7F7F7F",
             minor_tick_line_alpha = 0, num_minor_ticks = 2)
```

```
# or use the built in ggplot theme (under development)
figure(data = iris, legend = "top_left", tools = NULL) %>%
  ly_points(Sepal.Length, Petal.Length, color = Species) %>%
  set_theme(bk_ggplot_theme)
```
## Not run:
# or to set the theme for all future plots
options(bokeh_theme = bk_ggplot_theme)

figure() %>%
  ly_points(1:10)

figure() %>%
  ly_boxplot(1:10)

## End(Not run)

---

**shiny_callback**  
*Specify a Shiny callback*

### Description
Specify a Shiny callback

### Usage
```r
shiny_callback(id)
```

### Arguments
- `id`: a name that will be made available in your Shiny app as `input$id`

### Note
Depending on the type of callback you are using (selection, range, hover, tap), the value of `input$id` will change. The best way to get familiar with what to expect as these values is to debug inside your Shiny app and inspect the contents. You can also use `custom_callback` to write your own custom callbacks that can register other data in your Shiny app. To see what the callbacks look like for each callback type, see, for example, the contents of `rbokeh::handle_range_callback.shinyCallback`.

---

**sub_names**  
*Retrieve and properly parse all data*

### Description
Retrieve and properly parse all data

### Usage
```r
sub_names(fig, data, arg_obj, process_data_and_names = TRUE)
```
theme_axis

Arguments

fig figure to be used
data data to be used
arg_obj args object supplied by grab
process_data_and_names boolean to determine if the data and x_name and y_name should be post processed

Value

list of three groups: data, info, and params

theme_axis Override theme parameters for axis attributes

Description

Override theme parameters for axis attributes

Usage

theme_axis(
  fig,
  which = c("x", "y"),
  num_minor_ticks = 5,
  axis_label_standoff = NULL,
  axis_label_text_align = "left",
  axis_label_text_alpha = 1,
  axis_label_text_baseline = "bottom",
  axis_label_text_color = "#444444",
  axis_label_text_font = "Helvetica",
  axis_label_text_font_size = "12pt",
  axis_label_text_font_style = "normal",
  axis_line_alpha = 1,
  axis_line_cap = "butt",
  axis_line_color = "black",
  axis_line_dash = NULL,
  axis_line_dash_offset = 0,
  axis_line_join = "miter",
  axis_line_width = 1,
  major_label_orientation = "horizontal",
  major_label_standoff = NULL,
  major_label_text_align = "left",
  major_label_text_alpha = 1,
  major_label_text_baseline = "bottom",
  major_label_text_color = "#444444",
)
major_label_text_font = "Helvetica",
major_label_text_font_size = "12pt",
major_label_text_font_style = "normal",
major_tick_in = NULL,
major_tick_line_alpha = 1,
major_tick_line_cap = "butt",
major_tick_line_color = "black",
major_tick_line_dash = NULL,
major_tick_line_dash_offset = 0,
major_tick_line_join = "miter",
major_tick_line_width = 1,
major_tick_out = NULL,
minor_tick_in = NULL,
minor_tick_line_alpha = 1,
minor_tick_line_cap = "butt",
minor_tick_line_color = "black",
minor_tick_line_dash = NULL,
minor_tick_line_dash_offset = 0,
minor_tick_line_join = "miter",
minor_tick_line_width = 1,
minor_tick_out = NULL,
pars = NULL
)

Arguments

fig  
figure to modify

which  
which grids to apply attributes to ("x" and/or "y")

num_minor_ticks  
number of minor ticks

axis_label_standoff  
(integer) The distance in pixels that the axis labels should be offset from the tick labels.

axis_label_text_align  
('left', 'right', 'center') The text align of the axis label.

axis_label_text_alpha  
(numeric) The text alpha of the axis label.

axis_label_text_baseline  
('top', 'middle', 'bottom', 'alphabetic', 'hanging') The text baseline of the axis label.

axis_label_text_color  
(color) The text color of the axis label.

axis_label_text_font  
(string) The text font of the axis label.

axis_label_text_font_size  
(string - e.g. '12pt') The text font size of the axis label.

axis_label_text_font_style  
('normal', 'italic', 'bold') The text font style of the axis label.
theme_axis

axis_line_alpha
  (numeric) The line alpha of the axis line.
axis_line_cap  ('butt', 'round', 'square') The line cap of the axis line.
axis_line_color  (color) The line color of the axis line.
axis_line_dash  The line dash of the axis line.
axis_line_dash_offset  (integer) The line dash offset of the axis line.
axis_line_join  ('miter', 'round', 'bevel') The line join of the axis line.
axis_line_width  (integer) The line width of the axis line.

major_label_orientation
  ('horizontal', 'vertical', or angle in degrees) What direction the major label text should be oriented. If a number is supplied, the angle of the text is measured from horizontal.

major_label_standoff
  (integer) The distance in pixels that the major tick labels should be offset from the associated ticks.

major_label_text_align
  ('left', 'right', 'center') The text align of the major tick labels.

major_label_text_alpha
  (numeric) The text alpha of the major tick labels.

major_label_text_baseline
  ('top', 'middle', 'bottom', 'alphabetic', 'hanging') The text baseline of the major tick labels.

major_label_text_color
  (color) The text color of the major tick labels.

major_label_text_font
  (string - 'Helvetica') The text font of the major tick labels.

major_label_text_font_size
  (string - e.g. '12pt') The text font size of the major tick labels.

major_label_text_font_style
  ('normal', 'italic', 'bold') The text font style of the major tick labels.

major_tick_in
  (integer) The distance in pixels that major ticks should extend into the main plot area.

major_tick_line_alpha
  (numeric) The line alpha of the major ticks.

major_tick_line_cap
  ('butt', 'round', 'square') The line cap of the major ticks.

major_tick_line_color
  (color) The line color of the major ticks.

major_tick_line_dash
  The line dash of the major ticks.

major_tick_line_dash_offset
  (integer) The line dash offset of the major ticks.
major_tick_line_join
  ('miter', 'round', 'bevel') The line join of the major ticks.

major_tick_line_width
  (integer) The line width of the major ticks.

major_tick_out
  (integer) The distance in pixels that major ticks should extend out of the main plot area.

minor_tick_in
  (integer) The distance in pixels that minor ticks should extend into the main plot area.

minor_tick_line_alpha
  (numeric) The line alpha of the minor ticks.

minor_tick_line_cap
  ('butt', 'round', 'square') The line cap of the minor ticks.

minor_tick_line_color
  (color) The line color of the minor ticks.

minor_tick_line_dash
  The line dash of the minor ticks.

minor_tick_line_dash_offset
  (integer) The line dash offset of the minor ticks.

minor_tick_line_join
  ('miter', 'round', 'bevel') The line join of the minor ticks.

minor_tick_line_width
  (integer) The line width of the minor ticks.

minor_tick_out
  (integer) The distance in pixels that major ticks should extend out of the main plot area.

pars
  optionally specify a named list of all parameters - useful when dealing with theme lists

Examples

# manually specify a ggplot-like grid and background
figure() %>%
  ly_points(1:10) %>%
  theme_plot(background_fill_color = "#E6E6E6",
             outline_line_color = "white") %>%
  theme_grid(c("x", "y"),
             grid_line_color = "white",
             minor_grid_line_color = "white",
             minor_grid_line_alpha = 0.4) %>%
  theme_axis(c("x", "y"),
             axis_line_color = "white",
             major_label_text_color = "#7F7F7F",
             major_tick_line_color = "#7F7F7F",
             minor_tick_line_alpha = 0, num_minor_ticks = 2)

# or use the built in ggplot theme (under development)
figure(data = iris, legend = "top_left", tools = NULL) %>%
  ly_points(Sepal.Length, Petal.Length, color = Species) %>%
  set_theme(bk_ggplot_theme)
theme_grid

## Not run:
# or to set the theme for all future plots
options(bokeh_theme = bk_ggplot_theme)

figure() %>%
  ly_points(1:10)

figure() %>%
  ly_boxplot(1:10)

## End(Not run)

---

theme_grid

_override theme parameters for grid attributes_

### Description

Override theme parameters for grid attributes

### Usage

```r
theme_grid(
  fig,
  which = c("x", "y"),
  band_fill_alpha = 1,
  band_fill_color = "gray",
  grid_line_alpha = 1,
  grid_line_cap = "butt",
  grid_line_color = "black",
  grid_line_dash = NULL,
  grid_line_dash_offset = 0,
  grid_line_join = "miter",
  grid_line_width = 1,
  minor_grid_line_alpha = 1,
  minor_grid_line_cap = "butt",
  minor_grid_line_color = "black",
  minor_grid_line_dash = NULL,
  minor_grid_line_dash_offset = 0,
  minor_grid_line_join = "miter",
  minor_grid_line_width = 1,
  pars = NULL
)
```

### Arguments

- **fig** Figure to modify
- **which** Which grids to apply attributes to ("x" and/or "y")
band_fill_alpha
   The fill alpha of alternating bands between Grid lines.
band_fill_color
   The fill color of alternating bands between Grid lines.
grid_line_alpha
   The line alpha of the Grid lines.
grid_line_cap  ('butt', 'round', 'square') The line cap of the Grid lines.
grid_line_color
   The line color of the Grid lines.
grid_line_dash
   The line dash of the Grid lines.
grid_line_dash_offset
   The line dash offset of the Grid lines.
grid_line_join  ('miter', 'round', 'bevel') The line join of the Grid lines.
grid_line_width
   The line width of the Grid lines.
minor_grid_line_alpha
   The line alpha of the minor Grid lines.
minor_grid_line_cap  ('butt', 'round', 'square') The line cap of the minor Grid lines.
minor_grid_line_color
   The line color of the minor Grid lines.
minor_grid_line_dash
   The line dash of the minor Grid lines.
minor_grid_line_dash_offset
   The line dash offset of the minor Grid lines.
minor_grid_line_join  ('miter', 'round', 'bevel') The line join of the minor Grid lines.
minor_grid_line_width
   The line width of the minor Grid lines.
pars
   optionally specify a named list of all parameters - useful when dealing with
text

Examples

# manually specify a ggplot-like grid and background
figure() %>%
   ly_points(1:10) %>%
   theme_plot(background_fill_color = "#E6E6E6",
              outline_line_color = "white") %>%
   theme_grid(c("x", "y"), grid_line_color = "white",
              minor_grid_line_color = "white",
              minor_grid_line_alpha = 0.4) %>%
   theme_axis(c("x", "y"), axis_line_color = "white",
              major_label_text_color = "#7F7F7F",
              major_tick_line_color = "#7F7F7F",
              minor_tick_line_alpha = 0, num_minor_ticks = 2)
# or use the built in ggplot theme (under development)
figure(data = iris, legend = "top_left", tools = NULL) %>%
  ly_points(Sepal.Length, Petal.Length, color = Species) %>%
  set_theme(bk_ggplot_theme)

## Not run:
# or to set the theme for all future plots
options(bokeh_theme = bk_ggplot_theme)
figure() %>%
  ly_points(1:10)
figure() %>%
  ly_boxplot(1:10)
## End(Not run)

---

### theme_legend

Override theme parameters for legend attributes

#### Description

Override theme parameters for legend attributes

#### Usage

```r
theme_legend(
  fig,
  background_fill_alpha = 0.95,
  background_fill_color = "#fff",
  border_line_alpha = 0.5,
  border_line_cap = "butt",
  border_line_color = "black",
  border_line_dash = NULL,
  border_line_dash_offset = 0,
  border_line_join = "miter",
  border_line_width = 1,
  glyph_height = 20,
  glyph_width = 20,
  label_height = 20,
  label_standoff = 15,
  label_text_align = "left",
  label_text_alpha = 1,
  label_text_baseline = "bottom",
  label_text_color = "#444444",
  label_text_font = "Helvetica",
  label_text_font_size = "12pt",
)```
```
label_text_font_style = "normal",
label_width = 50,
legend_padding = 10,
legend_spacing = 3,
pars = NULL
)

Arguments

fig figure to modify
background_fill_alpha (numeric) background color alpha of plot
background_fill_color (color) background color of plot
border_line_alpha The line alpha for the legend border outline.
border_line_cap (‘butt’, ‘round’, ‘square’) The line cap for the legend border outline.
border_line_color The line color for the legend border outline.
border_line_dash The line dash for the legend border outline.
border_line_dash_offset The line dash offset for the legend border outline.
border_line_join (‘miter’, ‘round’, ‘bevel’) The line join for the legend border outline.
border_line_width The line width for the legend border outline.
glyph_height The height (in pixels) that the rendered legend glyph should occupy.
glyph_width The width (in pixels) that the rendered legend glyph should occupy.
label_height The height (in pixels) of the area that legend labels should occupy.
label_standoff The distance (in pixels) to separate the label from its associated glyph.
label_text_align (‘left’, ‘right’, ‘center’) The text align for the legend labels.
label_text_alpha The text alpha for the legend labels.
label_text_color The text color for the legend labels.
label_text_font The text font for the legend labels.
label_text_font_size The text font size for the legend labels.
theme_plot

\begin{verbatim}
label_text_font_style
    ('normal', 'italic', 'bold') The text font style for the legend labels.
label_width
    The width (in pixels) of the area that legend labels should occupy.
legend_padding
    Amount of padding around the legend.
legend_spacing
    Amount of spacing between legend entries.
pars
    optionally specify a named list of all parameters - useful when dealing with
    theme lists
\end{verbatim}

Examples

\begin{verbatim}
figure(legend_location = "top_left") >>
ly_points(1:10, legend = "a") >>
theme_legend(border_line_width = 2)
\end{verbatim}

theme_plot

Override theme parameters for general plot attributes

Description

Override theme parameters for general plot attributes

Usage

\begin{verbatim}
theme_plot(
    fig,
    pars = NULL,
    background_fill_color = "white",
    background_fill_alpha = 1,
    border_fill_color = "white",
    border_fill_alpha = 1,
    outline_line_alpha = 1,
    outline_line_cap = "butt",
    outline_line_color = "black",
    outline_line_dash = NULL,
    outline_line_dash_offset = 0,
    outline_line_join = "miter",
    outline_line_width = 1,
    min_border = 50,
    min_border_bottom = 50,
    min_border_left = 50,
    min_border_right = 50,
    min_border_top = 50
)
\end{verbatim}
Arguments

**fig**
figure to modify

**pars**
optionally specify a named list of all parameters - useful when dealing with theme lists

**background_fill_color**
(color) background color of plot

**background_fill_alpha**
(numeric) background color alpha of plot

**border_fill_color**
(color) fill color of border area of plot

**border_fill_alpha**
(numeric) fill color alpha of border area of plot

**outline_line_alpha**
(numeric) The line alpha for the plot border outline.

**outline_line_cap**
('butt', 'round', 'square') The line cap for the plot border outline.

**outline_line_color**
(color) The line color for the plot border outline.

**outline_line_dash**
The line dash for the plot border outline.

**outline_line_dash_offset**
(integer) The line dash offset for the plot border outline.

**outline_line_join**
('miter', 'round', 'bevel') The line join for the plot border outline.

**outline_line_width**
(integer) The line width for the plot border outline.

**min_border**
(integer) A convenience property to set all the min_X_border properties to the same value. If an individual border property is explicitly set, it will override min_border.

**min_border_bottom**
(integer) Minimum size in pixels of the padding region below the bottom of the central plot region. This is a minimum. The padding region may expand as needed to accommodate titles or axes, etc.

**min_border_left**
(integer) Minimum size in pixels of the padding region to the left of the central plot region. This is a minimum. The padding region may expand as needed to accommodate titles or axes, etc.

**min_border_right**
(integer) Minimum size in pixels of the padding region to the right of the central plot region. This is a minimum. The padding region may expand as needed to accommodate titles or axes, etc.

**min_border_top**
(integer) Minimum size in pixels of the padding region above the top of the central plot region. This is a minimum. The padding region may expand as needed to accommodate titles or axes, etc.
Examples

# manually specify a ggplot-like grid and background
figure() %>%
  ly_points(1:10) %>%
  theme_plot(background_fill_color = "#E6E6E6",
             outline_line_color = "white") %>%
  theme_grid(c("x", "y"), grid_line_color = "white",
             minor_grid_line_color = "white",
             minor_grid_line_alpha = 0.4) %>%
  theme_axis(c("x", "y"), axis_line_color = "white",
             major_label_text_color = "#7F7F7F",
             major_tick_line_color = "#7F7F7F",
             minor_tick_line_color = "white",
             minor_tick_line_alpha = 0, num_minor_ticks = 2)

# or use the built in ggplot theme (under development)
figure(data = iris, legend = "top_left", tools = NULL) %>%
  ly_points(Sepal.Length, Petal.Length, color = Species) %>%
  set_theme(bk_ggplot_theme)

## Not run:
# or to set the theme for all future plots
options(bokeh_theme = bk_ggplot_theme)

figure() %>%
  ly_points(1:10)

figure() %>%
  ly_boxplot(1:10)

## End(Not run)

### theme_title

Override theme parameters for general plot attributes

#### Description

Override theme parameters for general plot attributes

#### Usage

theme_title(
  fig,
  pars = NULL,
  background_fill_color = "white",
  background_fill_alpha = 1,
  border_fill_color = "white",
  border_fill_alpha = 1,
  align = "left",
)
text_alpha = 1,
  text_baseline = "bottom",
  text_color = "#444444",
  text_font = "Helvetica",
  text_font_size = "12pt",
  text_font_style = "normal"
)

Arguments

fig            figure to modify
pars           optionally specify a named list of all parameters - useful when dealing with
               theme lists
background_fill_color
               (color) background color of plot
background_fill_alpha
               (numeric) background color alpha of plot
border_fill_color
               (color) fill color of border area of plot
border_fill_alpha
               (numeric) fill color alpha of border area of plot
align
               ('left', 'right', 'center') The text align for the plot title.
text_alpha
               The text alpha for the plot title.
text_baseline
               ('top', 'middle', 'bottom', 'alphabetic', 'hanging') The text baseline for the plot
               title.
text_color
               (color) The text color for the plot title.
text_font
               (string) The text font for the plot title.
text_font_size
               (string - e.g. '12pt') The text font size for the plot title.
text_font_style
               ('normal', 'italic', 'bold') The text font style for the plot title.

Examples

figure(title = "asdf") %>%
  ly_points(1:10) %>%
  theme_title(text_color = "red")

---

Add "box_select" tool to a Bokeh figure

Description

Add "box_select" tool to a Bokeh figure
tool_box_select

Usage

tool_box_select(
    fig,
    callback = NULL,
    ref_layer = NULL,
    line_color = "black",
    line_alpha = 1,
    fill_color = "lightgrey",
    fill_alpha = 0.5,
    line_width = 2,
    line_dash = c(4, 4),
    level = "overlay"
)

Arguments

fig figure to modify

callback a callback to be applied to this tool - either a character string of javascript
code or any one of debug_callback, shiny_callback, console_callback,
custom_callback

ref_layer name of the layer that the callback should be applied to

line_color, line_alpha, fill_color, fill_alpha, line_width, line_dash, level
parameters to control the look of the selection bounding region

Note

Tools can be easily specified as a vector of tool names in the tools argument when instantiating
a figure. In this case, they are added with defaults. Explicitly calling these tool_ functions will
manually add the tool to a figure and allow additional specification of parameters.

See Also

Other tools: tool_box_zoom(), tool_crosshair(), tool_hover(), tool_lasso_select(), tool_pan(),
tool_reset(), tool_resize(), tool_save(), tool_tap(), tool_wheel_zoom()

Examples

figure() %>% ly_points(1:10) %>%
tool_box_select()
tool_box_zoom

Add "box_zoom" tool to a Bokeh figure

Description

Add "box_zoom" tool to a Bokeh figure

Usage

```r
tool_box_zoom(
  fig,
  line_color = "black",
  line_alpha = 1,
  fill_color = "lightgrey",
  fill_alpha = 0.5,
  line_width = 2,
  line_dash = c(4, 4),
  level = "overlay"
)
```

Arguments

- **fig**: figure to modify
- **line_color**, **line_alpha**, **fill_color**, **fill_alpha**, **line_width**, **line_dash**, **level**: parameters to control the look of the selection bounding region

Note

Tools can be easily specified as a vector of tool names in the `tools` argument when instantiating a `figure`. In this case, they are added with defaults. Explicitly calling these `tool_` functions will manually add the tool to a figure and allow additional specification of parameters.

See Also

Other tools: `tool_box_select()`, `tool_crosshair()`, `tool_hover()`, `tool_lasso_select()`, `tool_pan()`, `tool_reset()`, `tool_resize()`, `tool_save()`, `tool_tap()`, `tool_wheel_zoom()

Examples

```r
figure() %>% ly_points(1:10) %>%
  tool_box_zoom()
```
tool_crosshair  
*Add "crosshair" tool to a Bokeh figure*

**Description**

Add "crosshair" tool to a Bokeh figure

**Usage**

```r
tool_crosshair(fig)
```

**Arguments**

- `fig` figure to modify

**Note**

Tools can be easily specified as a vector of tool names in the `tools` argument when instantiating a `figure`. In this case, they are added with defaults. Explicitly calling these `tool_` functions will manually add the tool to a figure and allow additional specification of parameters.

**See Also**

Other tools: `tool_box_select()`, `tool_box_zoom()`, `tool_hover()`, `tool_lasso_select()`, `tool_pan()`, `tool_reset()`, `tool_resize()`, `tool_save()`, `tool_tap()`, `tool_wheel_zoom()`

**Examples**

```r
figure() %>% ly_points(1:10) %>%
tool_crosshair()
```

tool_hover  
*Add "hover" tool to a Bokeh figure*

**Description**

Add "hover" tool to a Bokeh figure

**Usage**

```r
tool_hover(fig, callback, ref_layer)
```
tool_lasso_select

Add "lasso_select" tool to a Bokeh figure

Description
Add "lasso_select" tool to a Bokeh figure

Usage

```r
tool_lasso_select(
  fig,
  callback = NULL,
  ref_layer = NULL,
  line_color = "black",
  line_alpha = 1,
  fill_color = "lightgrey",
)```
tool_pan

```r
fill_alpha = 0.5,
line_width = 2,
line_dash = c(4, 4),
level = "overlay"
```

Arguments

- `fig` figure to modify
- `callback` a callback to be applied to this tool - either a character string of javascript code or any one of `debug_callback`, `shiny_callback`, `console_callback`, `custom_callback`
- `ref_layer` name of the layer that the callback should be applied to
- `line_color`, `line_alpha`, `fill_color`, `fill_alpha`, `line_width`, `line_dash`, `level` parameters to control the look of the selection bounding region

Note

Tools can be easily specified as a vector of tool names in the `tools` argument when instantiating a `figure`. In this case, they are added with defaults. Explicitly calling these `tool_` functions will manually add the tool to a figure and allow additional specification of parameters.

See Also

Other tools: `tool_box_select()`, `tool_box_zoom()`, `tool_crosshair()`, `tool_hover()`, `tool_pan()`, `tool_reset()`, `tool_resize()`, `tool_save()`, `tool_tap()`, `tool_wheel_zoom()`

Examples

```r
figure() %>% ly_points(1:10) %>%
  tool_lasso_select()
```

---

tool_pan Add "pan" tool to a Bokeh figure

Description

Add "pan" tool to a Bokeh figure

Usage

```r
tool_pan(fig, dimensions = "both")
```
tool_reset

Arguments

fig figure to modify
dimensions a vector specifying whether the pan tool should pan with respect to the x axis ("width") and the y axis ("height") or "both"

Note

Tools can be easily specified as a vector of tool names in the tools argument when instantiating a figure. In this case, they are added with defaults. Explicitly calling these tool_ functions will manually add the tool to a figure and allow additional specification of parameters.

See Also

Other tools: tool_box_select(), tool_box_zoom(), tool_crosshair(), tool_hover(), tool_lasso_select(), tool_reset(), tool_resize(), tool_save(), tool_tap(), tool_wheel_zoom()

Examples

# only pan on x axis
figure() %>% ly_points(1:10) %>%
  tool_pan(dimensions = "height")

---

tool_reset Add "reset" tool to a Bokeh figure

Description

Add "reset" tool to a Bokeh figure

Usage

tool_reset(fig)

Arguments

fig figure to modify

Note

Tools can be easily specified as a vector of tool names in the tools argument when instantiating a figure. In this case, they are added with defaults. Explicitly calling these tool_ functions will manually add the tool to a figure and allow additional specification of parameters.

See Also

Other tools: tool_box_select(), tool_box_zoom(), tool_crosshair(), tool_hover(), tool_lasso_select(), tool_pan(), tool_resize(), tool_save(), tool_tap(), tool_wheel_zoom()
Examples

```r
figure() %>% ly_points(1:10) %>%
tool_resize()
```

---

tool_resize  
Add "resize" tool to a Bokeh figure

Description

Add "resize" tool to a Bokeh figure

Usage

```r
tool_resize(fig)
```

Arguments

- `fig`: figure to modify

Note

Tools can be easily specified as a vector of tool names in the `tools` argument when instantiating a `figure`. In this case, they are added with defaults. Explicitly calling these `tool_` functions will manually add the tool to a figure and allow additional specification of parameters.

See Also

Other tools: `tool_box_select()`, `tool_box_zoom()`, `tool_crosshair()`, `tool_hover()`, `tool_lasso_select()`, `tool_pan()`, `tool_reset()`, `tool_save()`, `tool_tap()`, `tool_wheel_zoom()`

Examples

```r
figure() %>% ly_points(1:10) %>%
tool_resize()
```
### tool_save

Add "save" tool to a Bokeh figure

**Description**

Add "save" tool to a Bokeh figure

**Usage**

```r
tool_save(fig)
```

**Arguments**

- `fig`: figure to modify

**Note**

Tools can be easily specified as a vector of tool names in the `tools` argument when instantiating a `figure`. In this case, they are added with defaults. Explicitly calling these `tool_` functions will manually add the tool to a figure and allow additional specification of parameters.

**See Also**

Other tools: `tool_box_select()`, `tool_box_zoom()`, `tool_crosshair()`, `tool_hover()`, `tool_lasso_select()`, `tool_pan()`, `tool_reset()`, `tool_resize()`, `tool_tap()`, `tool_wheel_zoom()`

**Examples**

```r
figure() %>% ly_points(1:10) %>%
tool_save()
```

### tool_selection

Add "selection" tool callback to a Bokeh figure

**Description**

This adds a selection callback to be used with the box select or lasso select tools.

**Usage**

```r
tool_selection(fig, callback, ref_layer)
```
Arguments

- **fig** figure to modify
- **callback** a callback to be applied to this tool - either a character string of javascript code or any one of `debug_callback`, `shiny_callback`, `console_callback`, `custom_callback`
- **ref_layer** name of the layer that the callback should be applied to

---

**tool_tap**

*Add "tap" tool to a Bokeh figure*

---

Description

Add "tap" tool to a Bokeh figure

Usage

```
tool_tap(fig, callback, ref_layer)
```

Arguments

- **fig** figure to modify
- **callback** a callback to be applied to this tool - either a character string of javascript code or any one of `debug_callback`, `shiny_callback`, `console_callback`, `custom_callback`
- **ref_layer** name of the layer that the callback should be applied to

Note

Tools can be easily specified as a vector of tool names in the `tools` argument when instantiating a `figure`. In this case, they are added with defaults. Explicitly calling these `tool_` functions will manually add the tool to a figure and allow additional specification of parameters.

See Also

Other tools: `tool_box_select()`, `tool_box_zoom()`, `tool_crosshair()`, `tool_hover()`, `tool_lasso_select()`, `tool_pan()`, `tool_reset()`, `tool_resize()`, `tool_save()`, `tool_wheel_zoom()`

Examples

```
figure() %>%
  ly_points(1:10, lname = "points") %>%
  tool_tap(debug_callback("points"), "points")
```
tool_wheel_zoom  
*Add "wheel_zoom" tool to a Bokeh figure*

---

**Description**

Add "wheel_zoom" tool to a Bokeh figure

**Usage**

```r
tool_wheel_zoom(fig, dimensions = "both")
```

**Arguments**

- `fig`: figure to modify
- `dimensions`: a vector specifying whether the wheel_zoom tool should zoom with respect to the x axis ("width") and the y axis ("height") or "both"

**Note**

Tools can be easily specified as a vector of tool names in the `tools` argument when instantiating a `figure`. In this case, they are added with defaults. Explicitly calling these `tool_` functions will manually add the tool to a figure and allow additional specification of parameters.

**See Also**

Other tools: `tool_box_select()`, `tool_box_zoom()`, `tool_crosshair()`, `tool_hover()`, `tool_lasso_select()`, `tool_pan()`, `tool_reset()`, `tool_resize()`, `tool_save()`, `tool_tap()`

**Examples**

```r
# only zoom on x axis
figure() %>% ly_points(1:10) %>%
  tool_wheel_zoom(dimensions = "height")
```

---

widget2gist  
*Export htmlwidget plot to a gist*

---

**Description**

Export htmlwidget plot to a gist
widget2gist

Usage

widget2gist(
  widget_string,
  name,
  created = NULL,
  description = "",
  license = c("none", "apache-2.0", "bsd-2-clause", "bsd-3-clause", "cc-by-4.0",
             "cc-by-nc-4.0", "cc-by-nc-nd-4.0", "cc-by-nc-sa-4.0", "cc-by-nd-4.0", "cc-by-sa-4.0",
             "cddl-1.0", "epl-1.0", "gpl-2.0", "gpl-3.0", "lgpl-2.1", "lgpl-3.0", "mit",
             "mpl-2.0"),
  border = TRUE,
  scrolling = FALSE,
  secure = TRUE,
  view = TRUE
)

Arguments

widget_string a string containing R code to create an htmlwidget
name name of the gist
created optional string for a "Created by" to precede the README
description optional text to go in README.md to describe the gist
license license under which gist is released - one of those accepted here: https://bl.ocks.org/licenses.txt
border should the bl.ocks.org iframe have a border?
scrolling should the bl.ocks.org iframe scroll?
secure should https be used for cdn links?
view should the resulting gist be opened in the browser on bl.ocks.org?

Note

This requires that you have a github personal access token stored as an environment variable GITHUB_PAT. See gist_create for more information.
Also note that this currently can’t handle thumbnails but we are looking into ways to do that.

Examples

## Not run:
widget2gist("figure() %>% ly_points(1:10)", name = "test")

## End(Not run)
widget2png  Make a static png file for an htmlwidget

Description

Make a static png file for an htmlwidget

Usage

widget2png(p, file, timeout = 500)

Arguments

p htmlwidget object
file where to save png file
timeout plot render timeout in milliseconds (see details)

Details

This uses phantomjs (https://phantomjs.org) to render your htmlwidget in a headless browser and take a screenshot of it, creating a static output. This assumes that phantomjs has been installed on your machine and is available as a system call. For plots that take longer to load and render, you may need to increase the value of timeout. Note that this function is experimental.

Examples

```r
## Not run:
path <- tempfile(fileext = ".png")
figure(tools = NULL) %>%
  ly_points(1:10) %>%
  widget2png(path)
## End(Not run)
```

x_axis  Customize x axis of a Bokeh figure

Description

Customize x axis of a Bokeh figure
Usage

```r
x_axis(
  fig,
  label,
  position = "below",
  log = FALSE,
  grid = TRUE,
  desired_num_ticks = NULL,
  num_minor_ticks = 5,
  visible = TRUE,
  number_formatter = c("basic", "numeral", "printf"),
  power_limit_high = 5,
  power_limit_low = -3,
  precision = NULL,
  use_scientific = TRUE,
  format = NULL
)
```

Arguments

- `fig` figure to modify
- `label` axis label
- `position` where to place the axis (either "above" or "below")
- `log` logical or integer - if TRUE, a log axis with base 10 is used - if an integer, a log axis with base of that integer will be used
- `grid` logical - should a reference grid be shown for this axis?
- `desired_num_ticks` desired target number of major tick positions to generate across the plot range
- `num_minor_ticks` number of minor ticks
- `visible` should axis be shown?
- `number_formatter` Bokeh numeric tick label formatter ("basic", "numeral", or "printf"); ignored if log is TRUE
- `power_limit_high` (int) Limit the use of scientific notation to when log(x) >= value. Only applicable when number_formatter is "basic".
- `power_limit_low` (int) Limit the use of scientific notation to when log(x) <= value. Only applicable when number_formatter is "basic".
- `precision` (int) How many digits of precision to display in tick labels. Automatically determined if not specified. Only applicable when number_formatter is "basic".
- `use_scientific` (logical) Whether to ever display scientific notation. If True, then when to use scientific notation is controlled by power_limit_low and power_limit_high. Only applicable when number_formatter is "basic".
- `format` Specification of format options. Specification depends on the value of number_formatter - see "details" below.
Details

format parameter. When number_formatter is "basic" and the axis type is datetime, format specifies how to display tick values from a continuous range as formatted datetimes. See DatetimeTickFormatter. When number_formatter is "numeral", format specifies a human-readable format string. See NumeralTickFormatter. When number_formatter is "printf", format is a printf-style format string. See PrintfTickFormatter.

See Also

Other axes: y_axis()

Examples

```r
figure() %>%
  ly_points(rexp(1000), rexp(1000)) %>%
  x_axis(label = "x", log = TRUE) %>%
  y_axis(label = "y", log = TRUE)

figure() %>%
  ly_points(2 ^ (1:10)) %>%
  y_axis(log = 2)

# disable scientific tick labels
figure() %>%
  ly_points(rnorm(10), rnorm(10) / 1000) %>%
  y_axis(use_scientific = FALSE)

# specify datetime tick labels
# the appropriate datetime units are automatically chosen
big_range <- seq(as.Date("2012-01-01"), as.Date("2012-12-31"), by = "days")
small_range <- seq(as.Date("2012-01-01"), as.Date("2012-02-01"), by = "days")

figure() %>%
  ly_lines(big_range, rnorm(366)) %>%
  x_axis(label = "Date", format = list(months = "%b-%Y", days = "%d"))

figure() %>%
  ly_lines(small_range, rnorm(32)) %>%
  x_axis(label = "Date", format = list(months = "%b-%Y", days = "%d"))

# specify numeric tick labels
figure() %>%
  ly_points(rnorm(10), rnorm(10) * 10000) %>%
  y_axis(number_formatter = "numeral", format = "0,000")

figure() %>%
  ly_points(rnorm(10), rnorm(10) * 100) %>%
  y_axis(number_formatter = "printf", format = "%0.1f%%")
```
**Description**

Update x axis range in a Bokeh figure

**Usage**

```r
x_range(fig, dat = NULL, callback = NULL)
```

**Arguments**

- `fig` : figure to modify
- `dat` : either a vector (min, max) if the axis is numeric, or a vector of values if the axis is categorical. In the latter case, the order in which the values are supplied is how they will be arranged on the axis.
- `callback` : TODO

**See Also**

Other ranges: `y_range()`

**Examples**

```r
# get data from Duluth site in 'barley' data
du <- subset(lattice::barley, site == "Duluth")

# plot with default ranges
p <- figure(width = 600) %>%
  ly_points(yield, variety, color = year, data = du)
p

# y axis is alphabetical

# manually set x and y axis (y in order of 1932 yield)
p %>%
  x_range(c(20, 40)) %>%
  y_range(du$variety[order(subset(du, year == 1932)$yield)])
```
Customize x axis of a Bokeh figure

Description
Customize x axis of a Bokeh figure

Usage
`y_axis(fig, label, position = "left", log = FALSE, grid = TRUE, desired_num_ticks = NULL, num_minor_ticks = 5, visible = TRUE, number_formatter = c("basic", "numeral", "printf"), power_limit_high = 5, power_limit_low = -3, precision = NULL, use_scientific = TRUE, format = NULL)`

Arguments
- `fig` figure to modify
- `label` axis label
- `position` where to place the axis (either "left" or "right")
- `log` logical or integer - if TRUE, a log axis with base 10 is used - if an integer, a log axis with base of that integer will be used
- `grid` logical - should a reference grid be shown for this axis?
- `desired_num_ticks` desired target number of major tick positions to generate across the plot range
- `num_minor_ticks` number of minor ticks
- `visible` should axis be shown?
- `number_formatter` Bokeh numeric tick label formatter ("basic", "numeral", or "printf"); ignored if log is TRUE
- `power_limit_high` (int) Limit the use of scientific notation to when log(x) >= value. Only applicable when number_formatter is "basic".
y_axis

power_limit_low (int) Limit the use of scientific notation to when log(x) <= value. Only applicable when number_formatter is "basic".

precision (int) How many digits of precision to display in tick labels. Automatically determined if not specified. Only applicable when number_formatter is "basic".

use_scientific (logical) Whether to ever display scientific notation. If True, then when to use scientific notation is controlled by power_limit_low and power_limit_high. Only applicable when number_formatter is "basic".

format Specification of format options. Specification depends on the value of number_formatter - see "details" below.

See Also

Other axes: x_axis()

Examples

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```

```r
ggplot() +
  geom_point(aes(x, y)) +
  x_axis(label = "x", log = TRUE) +
  y_axis(label = "y", log = TRUE)
```
y_range(number_formatter = "printf", format = "%0.1f%%")

---

**y_range**  
_Update y axis range in a Bokeh figure_

### Description

Update y axis range in a Bokeh figure

### Usage

```r
y_range(fig, dat = NULL, callback = NULL)
```

### Arguments

- **fig**: figure to modify
- **dat**: either a vector (min, max) if the axis is numeric, or a vector of values if the axis is categorical. In the latter case, the order in which the values are supplied is how they will be arranged on the axis.
- **callback**: TODO

### See Also

Other ranges: `x_range()`

### Examples

```r
# get data from Duluth site in 'barley' data
du <- subset(lattice::barley, site == "Duluth")

# plot with default ranges
p <- figure(width = 600) %>%
  ly_points(yield, variety, color = year, data = du)
p
# y axis is alphabetical

# manually set x and y axis (y in order of 1932 yield)
p %>%
  x_range(c(20, 40)) %>%
y_range(du$variety[order(subset(du, year == 1932)$yield)])
```
Description

Pipe figures

Arguments

<table>
<thead>
<tr>
<th>lhs</th>
<th>rhs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a Bokeh figure</td>
<td>a layer to add to the figure</td>
</tr>
</tbody>
</table>
Index

* axes
  x_axis, 112
  y_axis, 116

* data
  elements, 8
  flightfreq, 12
  nyctaxihex, 79

* layer functions
  ly_abline, 18
  ly_annular_wedge, 21
  ly_annulus, 23
  ly_arc, 26
  ly_bar, 28
  ly_bezier, 31
  ly_boxplot, 33
  ly_contour, 35
  ly_crect, 37
  ly_curve, 40
  ly_density, 42
  ly_hist, 45
  ly_image, 47
  ly_image_url, 48
  ly_lines, 49
  ly_map, 51
  ly_multi_line, 53
  ly_oval, 54
  ly_patch, 57
  ly_points, 59
  ly_polygons, 62
  ly_quadratic, 64
  ly_quantile, 66
  ly_ray, 68
  ly_rect, 70
  ly_segments, 72
  ly_text, 74
  ly_wedge, 76

* ranges
  x_range, 115
  y_range, 118

* tools
  tool_box_select, 100
  tool_box_zoom, 102
  tool_crosshair, 103
  tool_hover, 103
  tool_lasso_select, 104
  tool_pan, 105
  tool_reset, 106
  tool_resize, 107
  tool_save, 108
  tool_tap, 109
  tool_wheel_zoom, 110

%%, 119

b_eval, 5
bk_default_theme, 4
bk_ggplot_theme (bk_default_theme), 4
bokeh_render_json, 4
boxplot.stats, 34

cat, 81
catjitter, 5
colorRampPalette, 80
console_callback, 6, 101, 104, 105, 109
contourLines, 36
curve, 41
custom_callback, 6, 8, 88, 101, 104, 105, 109
data_name_list, 7
debug_callback, 6, 7, 101, 104, 105, 109
density, 42

elements, 8

figure, 9, 14, 101–110
figure_data, 12
flightfreq, 12

get_object_refs, 13
gist_create, 111
gmap, 11, 13, 15
ly_wedge, 11, 20, 23, 25, 27, 30, 33, 35, 36, 39, 41, 43, 46, 48, 49, 51, 53, 54, 57, 59, 61, 64, 66, 68, 70, 72, 74, 75, 76

tool_save, 11, 101–107, 108, 109, 110
tool_selection, 108
tool_tap, 101–108, 109, 110
tool_wheel_zoom, 11, 101–109, 110

widget2gist, 110
widget2png, 112

x_axis, 112, 117
x_range, 115, 118

y_axis, 114, 116
y_range, 115, 118

map, 52

nyctaxihex, 79

pal_bk_glyph (pal_color), 80
pal_bk_line_dash (pal_color), 80
pal_bk_line_width (pal_color), 80
pal_color, 80, 87
pal_gradient (pal_color), 80
pal_size (pal_color), 80
pal_tableau (pal_color), 80

par, 19, 20, 27, 32, 33, 36, 41, 43, 50, 51, 54, 65, 69, 73

phantom_install, 80
point_types, 60, 81
print_model_json, 4, 81

qnorm, 66

qunif, 66

rbokeh (rbokeh-package), 3
rbokeh-package, 3
rbokeh2html, 82
rbokehOutput, 83, 84
renderRbokeh, 84

set_palette, 85
set_theme, 87
shiny_callback, 88, 101, 104, 105, 109
sub_names, 88

toJSON, 81
tool_box_select, 11, 100, 102–110
tool_box_zoom, 11, 101, 102, 103–110
tool_crosshair, 11, 101, 102, 103, 104–110
tool_hover, 101–103, 103, 105–110
tool_lasso_select, 11, 101–104, 104, 106–110
tool_pan, 101–105, 105, 106–110