Index

Useful functions

- **plotly_data():** Obtains the data at any point in time, which is primarily useful for debugging purposes.
-

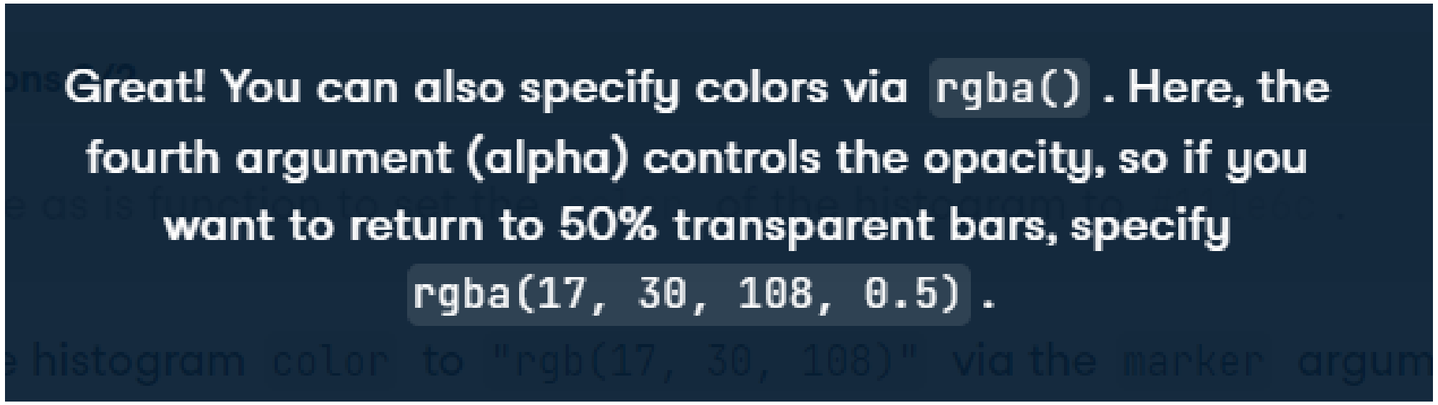
Basic transformations

layout()

- axes: type, labels, tick marks, transformations, etc.
- legend: position
- canvas: grid lines, background color
- size: height, width, margins

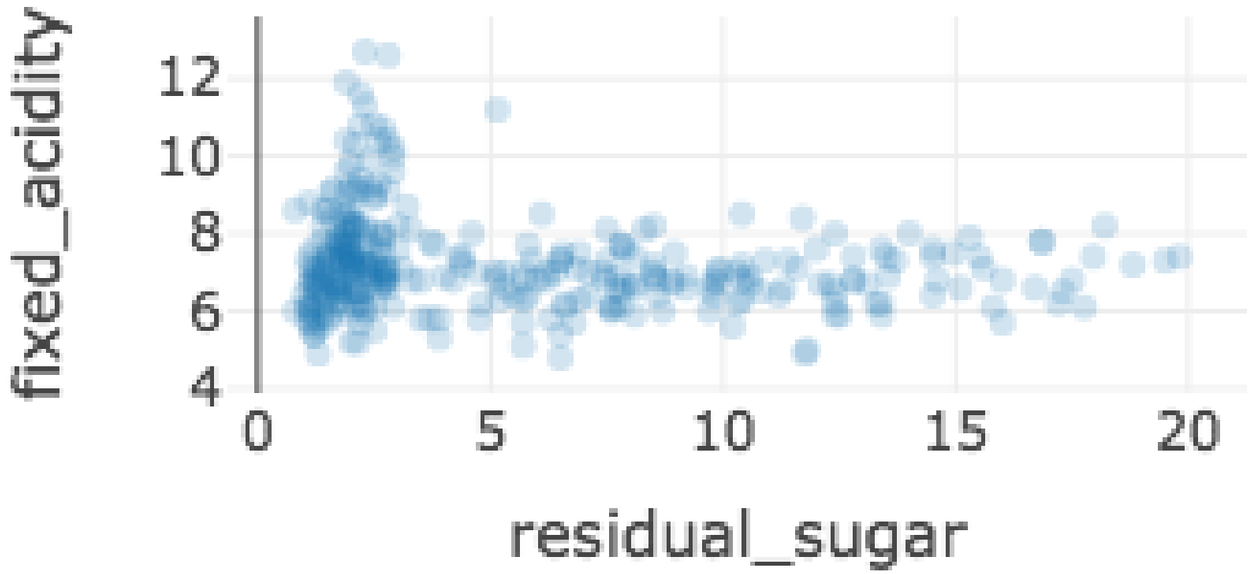
Marker options

- `opacity` `alpha`
- `color`
- `symbol` (sca er/box)
- `size` (sca er)
- `width` (bar/histogram)



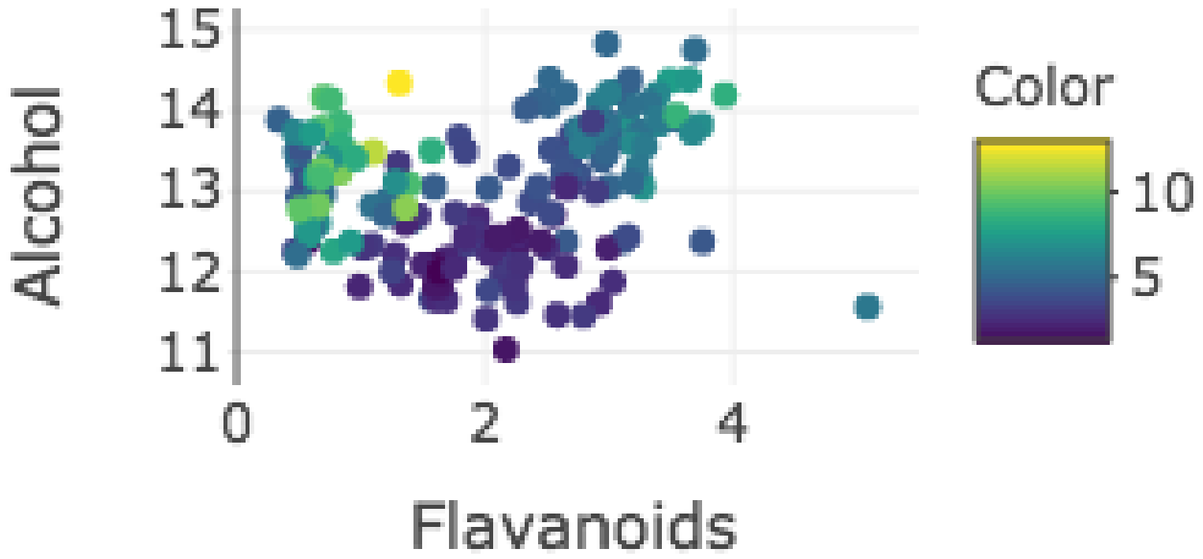
- stroke = I("black"): Color de contorno.
- span = I(2): Ancho de linea de contorno.

Opacity



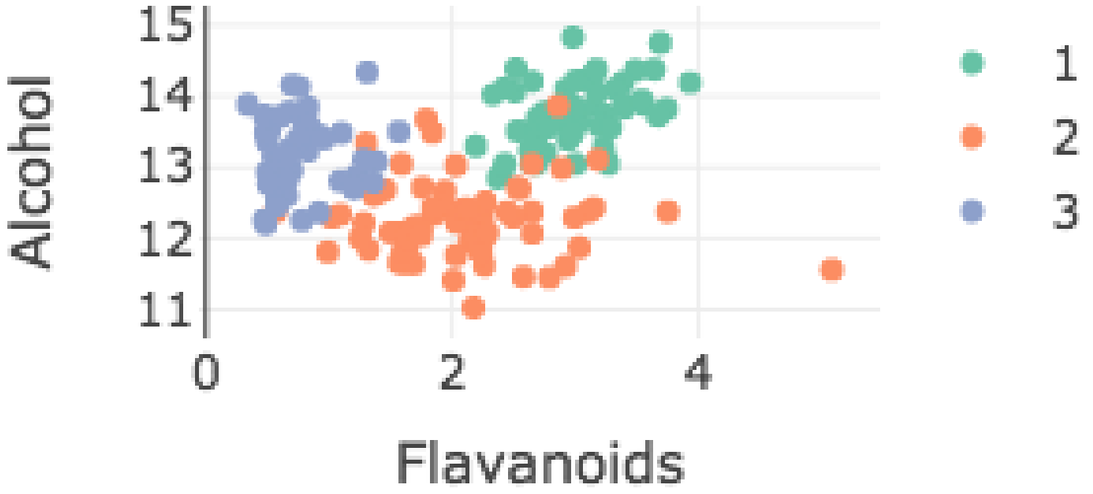

```
winequality %>%
  plot_ly(x = ~residual_sugar, y = ~fixed_acidity) %>%
  add_markers(marker = list(opacity = 0.2))           # Adjust opacity
```

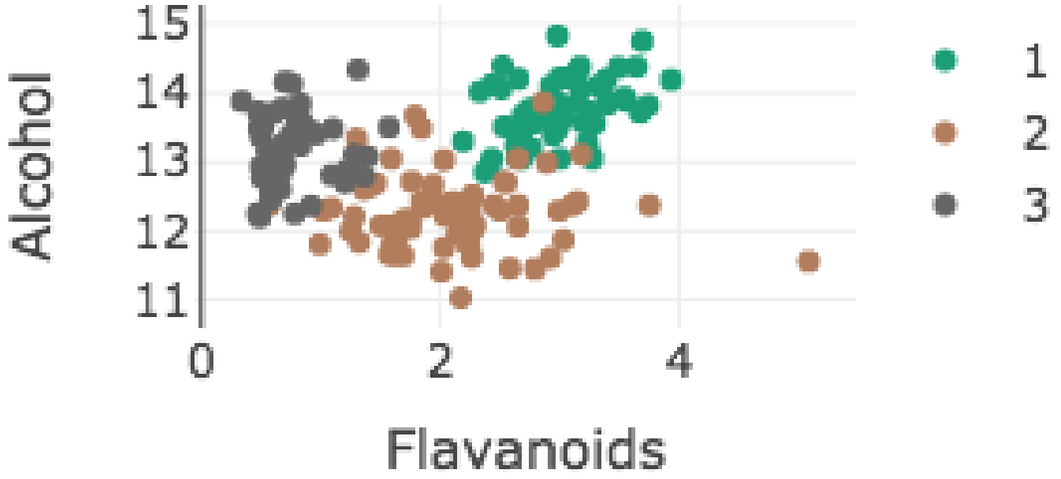
Adding a quantitative variable




```
wine %>%
  plot_ly(x = ~Flavanoids, y = ~Alcohol, color = ~Color) %>%
  add_markers()
```

RColorBrewer palettes





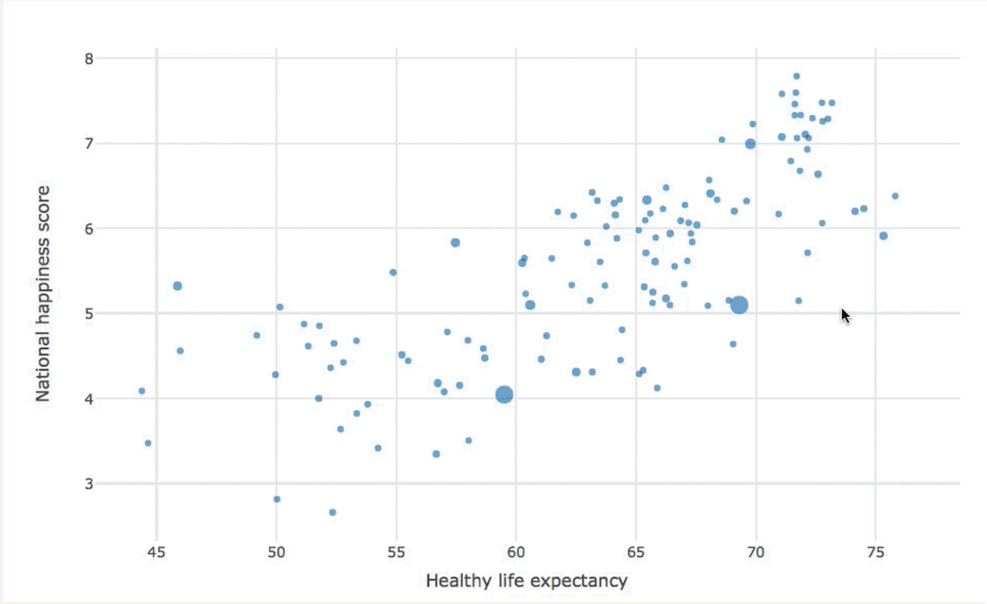

```
wine %>%
plot_ly(x = ~Flavanoids,
        y = ~Alcohol, color = ~Type) %>%
add_markers()
```

```
wine %>%
plot_ly(x = ~Flavanoids,
        y = ~Alcohol, color = ~Type) %>%
add_markers(colors = "Dark2")
```

```
add_markers(colors = c("orange", "black", "skyblue"))
```

Custom labels

```
happy %>%
  plot_ly(
    x = ~life.expectancy, y = ~happiness,
    hoverinfo = "text",
    text = ~paste("Country: ", country,
                  "</br> Population: ", population)
  ) %>%
  add_markers(size = ~population) %>%
  layout(
    xaxis = list(title = "Healthy life expectancy"),
    yaxis = list(title = "National happiness score")
  )
```



`hoverinfo =`

- `"all"`
- `"y"`
- `"x+y+z"`
- `"x"`
- `"x+y"`
- `"text"`

Custom labels with ggplot2

See the `tooltip` argument to ggplotly(). For instance, to show only the species name (e.g. `virginica` for the top right point) on hover:

```
g <- ggplot(tail(iris), aes(Petal.Length, Sepal.Length, text=Species)) + geom_point
ggplotly(g, tooltip="text")
```

Other examples:

```
ggplotly(g, tooltip="x")              # Petal.Length: 5.7
ggplotly(g, tooltip="Petal.Length")  # Petal.Length: 5.7
ggplotly(g, tooltip=c("x", "y"))
```

The last example will show the two-line tooltip

```
Petal.Length: 5.7
Sepal.Length: 6.7
```

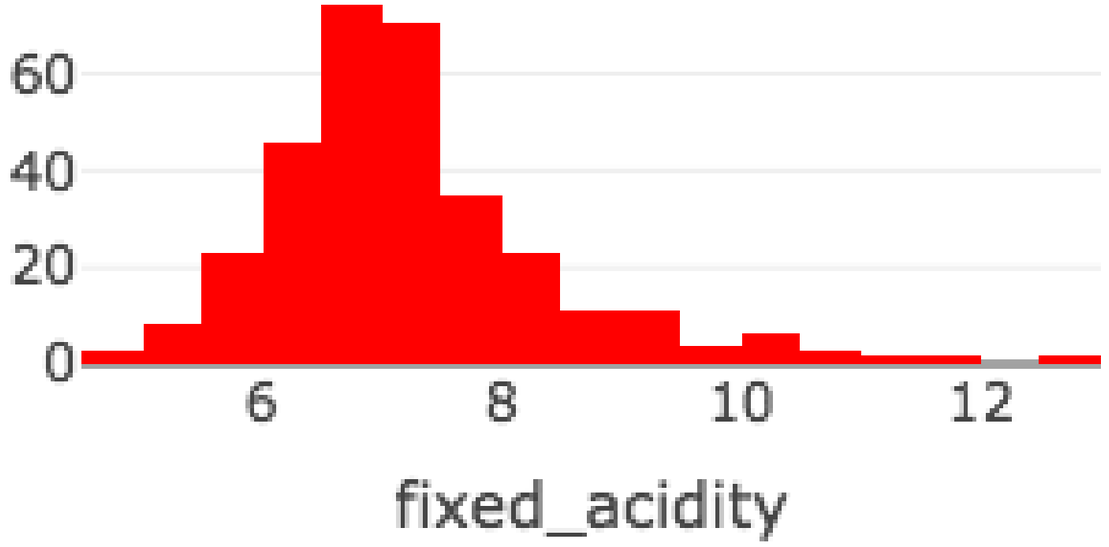
Static **bubble** charts

```
world_indicators %>%
   filter(year == 2014) %>%
   plot_ly(
     x = ~income, y = ~co2, hoverinfo = "text",
     text = ~country
   ) %>%
   add_markers(
      size = ~population, color = ~six_regions,
      marker = list(opacity = 0.5,
                    sizemode = "diameter",
                    sizeref = 2)
   )
```

Sizemode = “diameter”
Allows the size of the points to be governed by the diameter rather than the area. (Los hace mas grandes)

sizeref = 2
Reduce el tamaño de los circulos a la mitad.

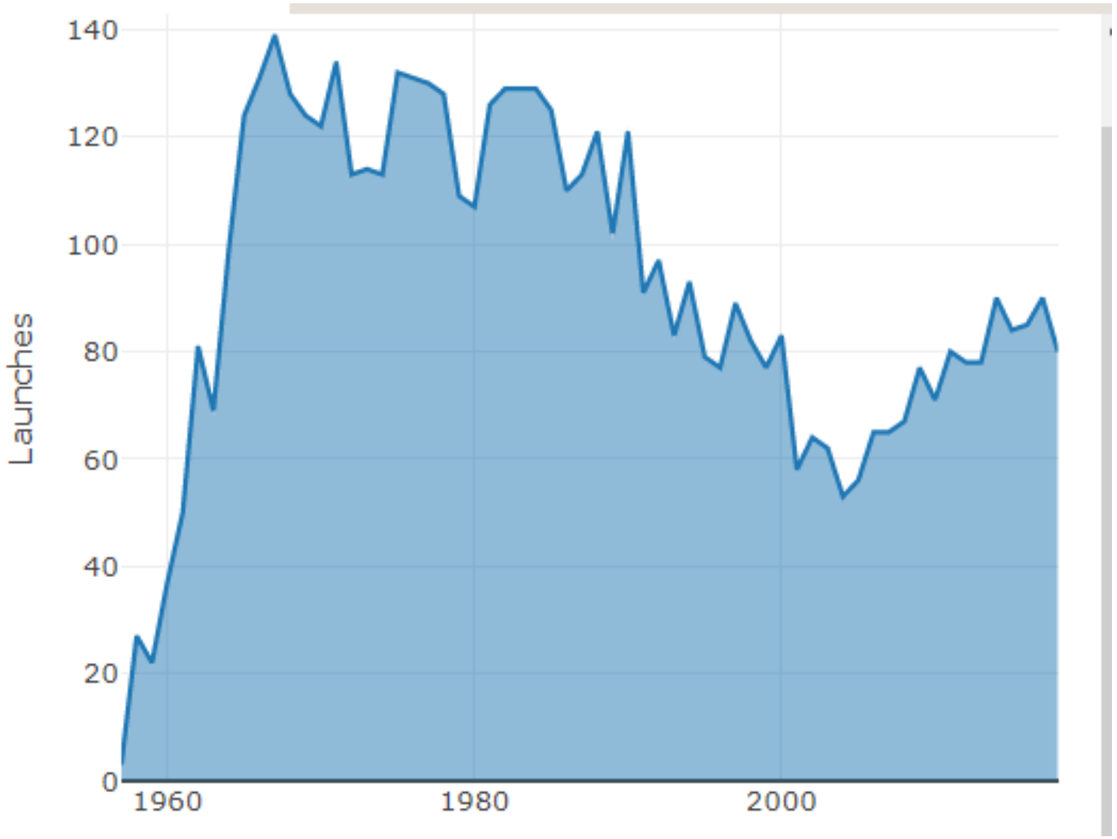
Color




```
winequality %>%
    plot_ly(x = ~fixed_acidity) %>%
    add_histogram(color = I("red")) # Setting color
```

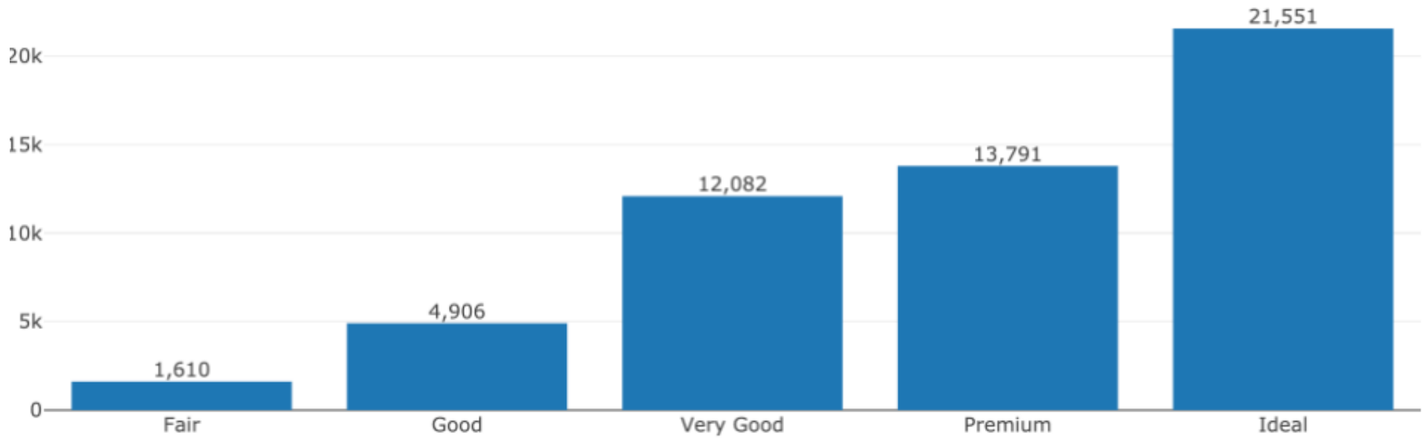
Fill area

```
# table of launches by year
launches_by_year <- launches %>%
  count(launch_year)

# create a filled area chart of launches
over time
launches_by_year %>%
  plot_ly(x = ~launch_year, y = ~n) %>%
  add_lines(fill = 'tozeroy') %>%
  layout(
    xaxis = list(title = "Year"),
    yaxis = list(title = "Launches")
  )
```

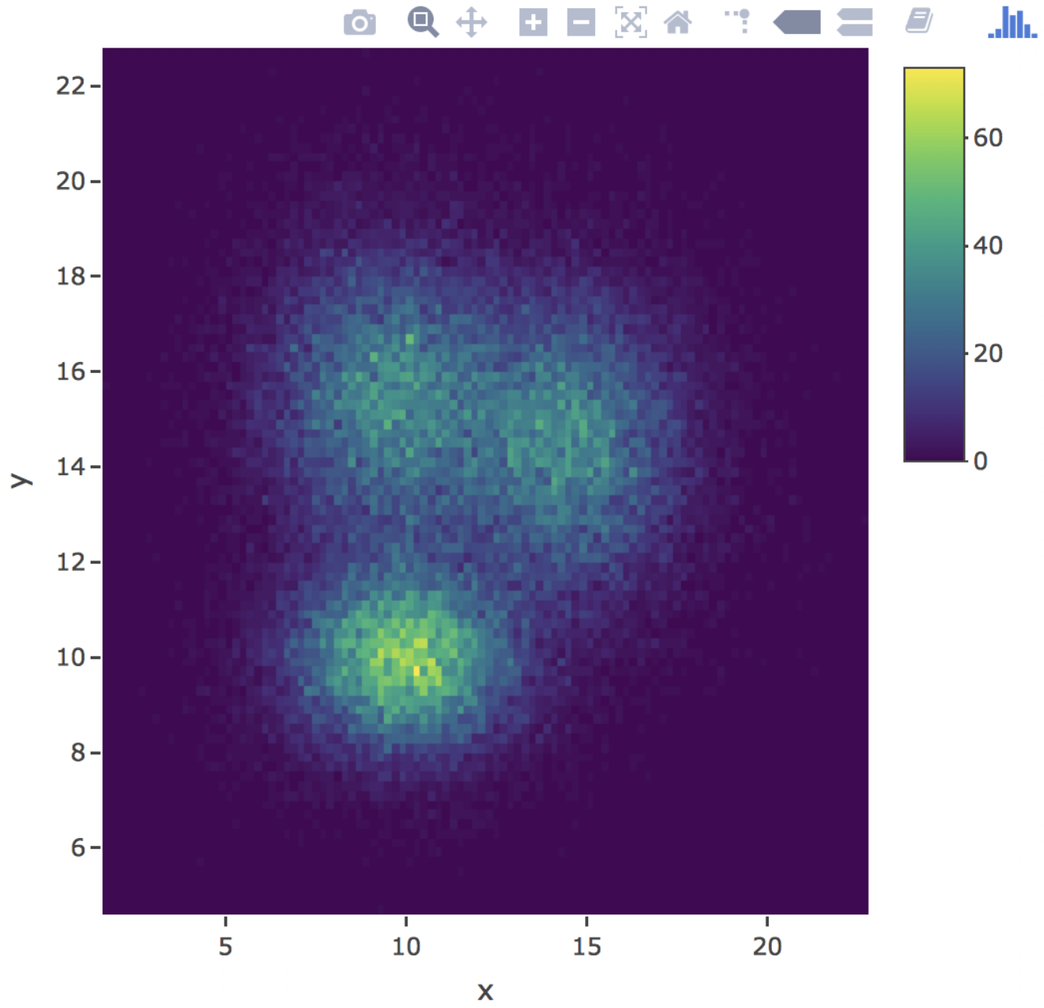


geom_bar

```
diamonds %>%
  plot_ly(x = ~cut) %>%
  add_histogram() %>%
  group_by(cut) %>%
  summarise(n = n()) %>%
  add_text(
    text = ~scales::comma(n), y = ~n,
    textposition = "top middle",
    cliponaxis = FALSE
  )
```



Changing the bins

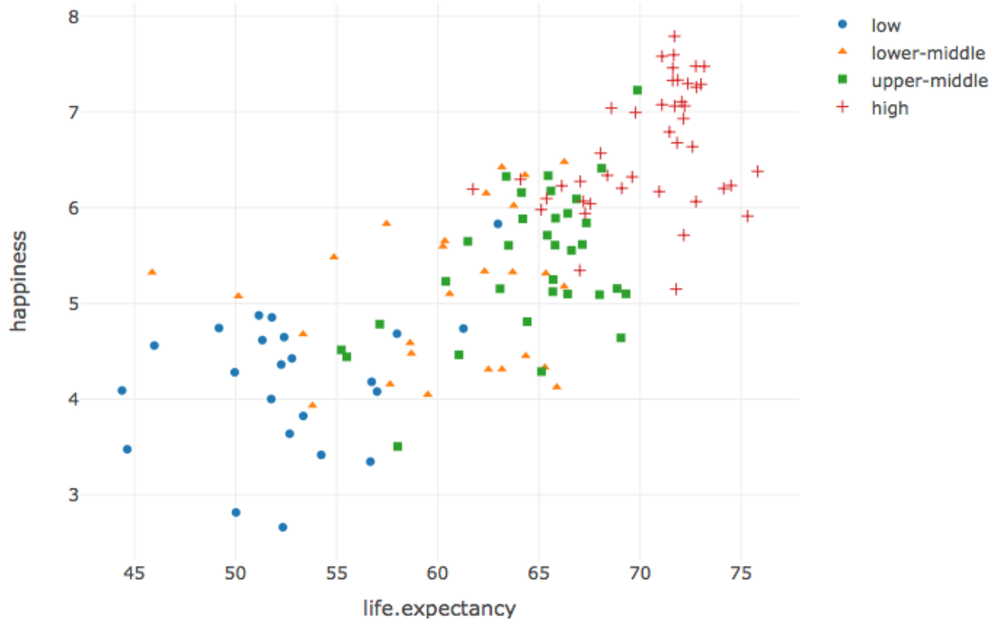


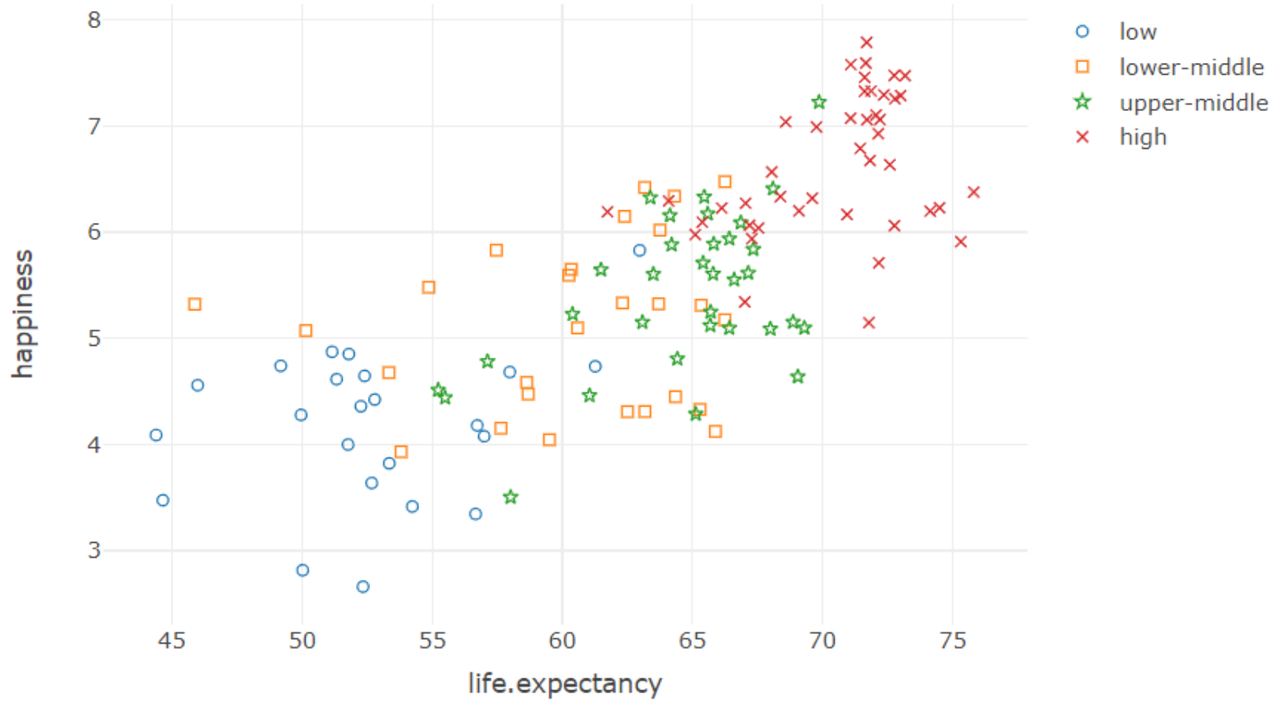

```
sim_data %>%
  plot_ly(x = ~x, y = ~y) %>%
  add_histogram2d(nbinsx = 200, nbinsy = 100)
```

Glyph symbol

```
happy %>%
  plot_ly(x = ~life.expectancy,
          y = ~happiness) %>%
  add_markers(symbol = ~income)
```

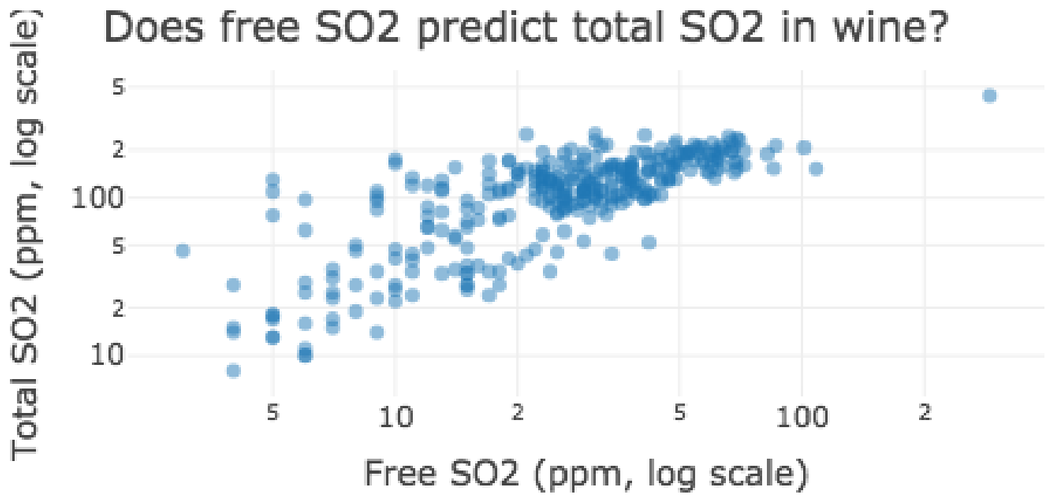
```
# Fill in the specified plotting symbols
happy %>%
  plot_ly(x = ~life.expectancy, y = ~happiness) %>%
  add_markers(symbol = ~income, symbols = c("circle-open", "square-open",
                                            "star-open", "x-thin-open"))
```





Layout transformations

Adding labs and transforming axes

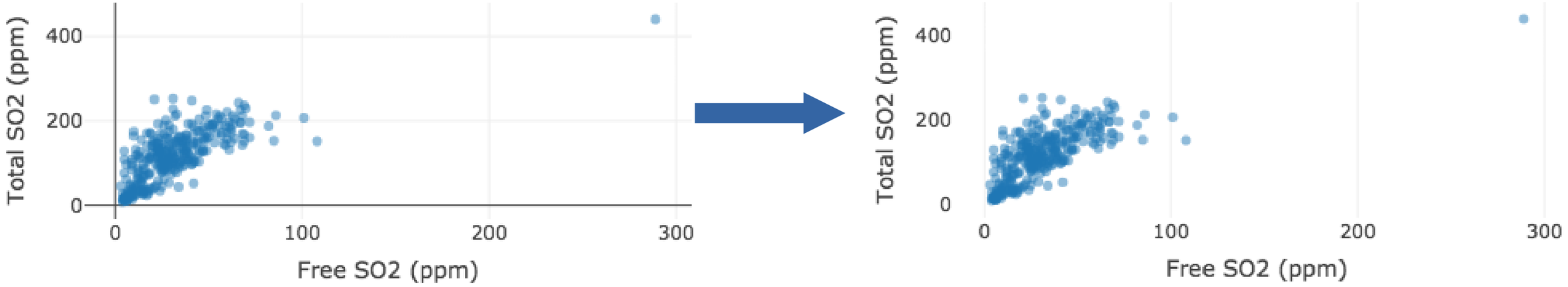



```
winequality %>%
  plot_ly(x = ~free_so2, y = ~total_so2) %>%
  add_markers(marker = list(opacity = 0.2)) %>%
  layout(xaxis = list(title = "Free SO2 (ppm, log scale)", type = "log"),
         yaxis = list(title = "Total SO2 (ppm, log scale)", type = "log"),
         title = "Does free SO2 predict total SO2 in wine?")
```

Other possible axis types include linear, date, and category.

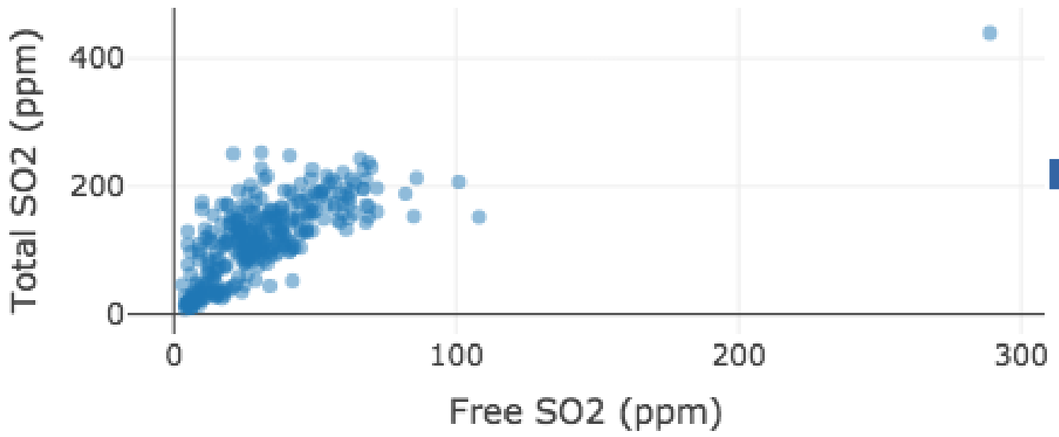
barmode = "overlay", showlegend = FALSE

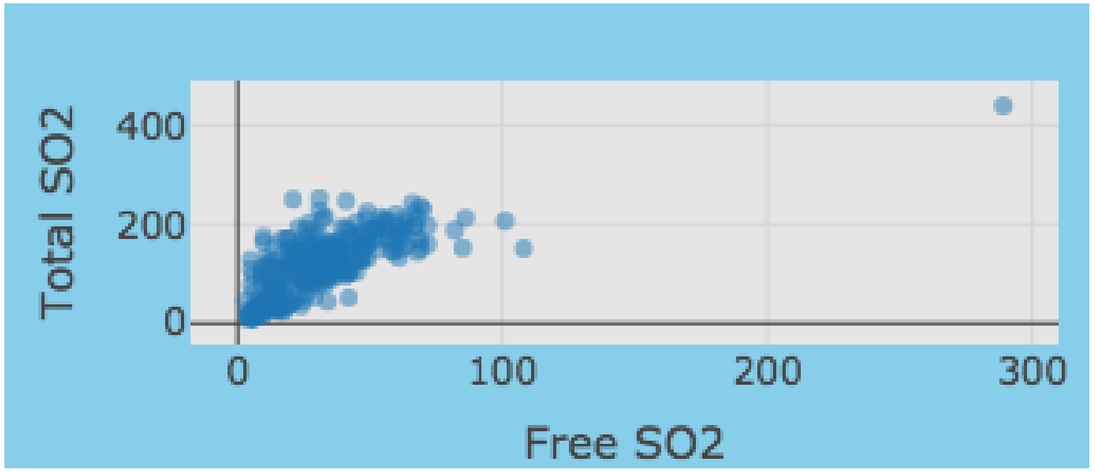
Customizing the grid




```
winequality %>%
  plot_ly(x = ~free_so2, y = ~total_so2) %>%
  add_markers(marker = list(opacity = 0.5)) %>%
  layout(xaxis = list(title = "Free SO2 (ppm)", zeroline = FALSE),
         yaxis = list(title = "Total SO2 (ppm)", zeroline = FALSE,
                      showgrid = FALSE))
```

Customizing the canvas



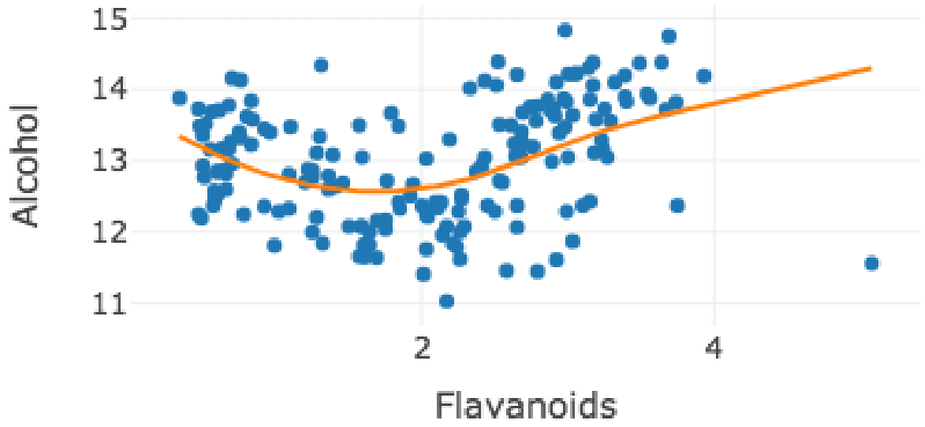



```
winequality %>%
  plot_ly(x = ~free_so2, y = ~total_so2) %>%
  add_markers(marker = list(opacity = 0.5)) %>%
  layout(xaxis = list(title = "Free SO2 (ppm)"),
         yaxis = list(title = "Total SO2 (ppm)"),
```

```
         plot_bgcolor = toRGB("gray90"),
         paper_bgcolor = toRGB("skyblue"))
```

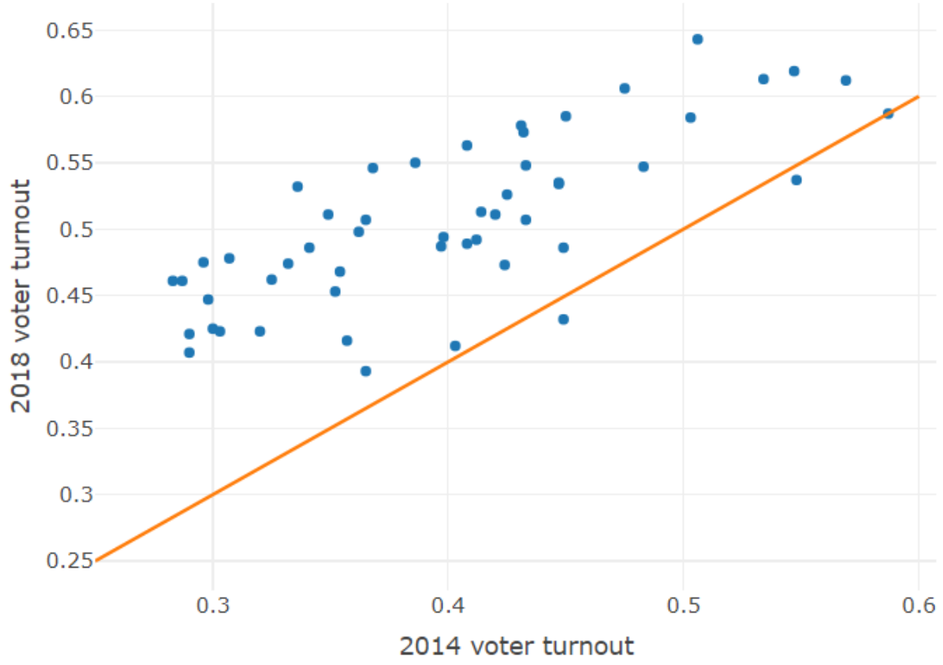
Adding smooth lines

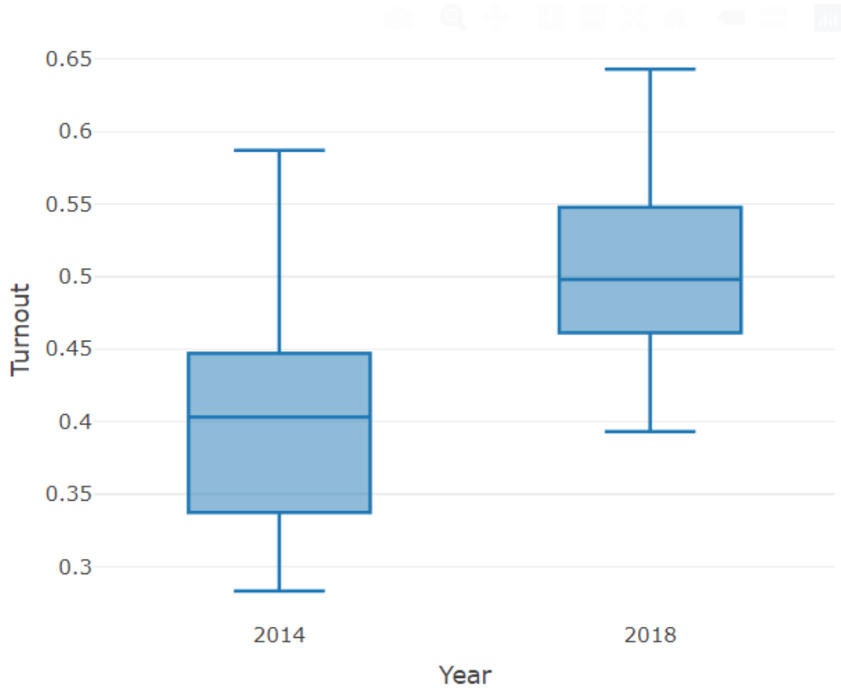
Adding a smoother




```
m <- loess(Alcohol ~ Flavanoids, data = wine, span = 1.5)
wine %>%
  plot_ly(x = ~Flavanoids, y = ~Alcohol) %>%
  add_markers() %>%
  add_lines(y = ~fitted(m)) %>%
  layout(showlegend = FALSE)
```

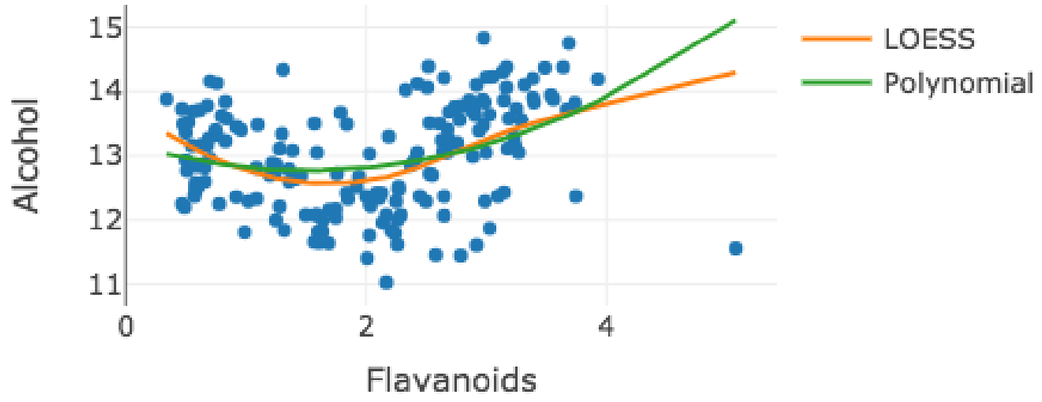
Adding a line y = x






```
# Add the line y = x to the scatterplot
p %>%
  add_lines(x = c(0.25, 0.6), y = c(0.25, 0.6)) %>%
  layout(showlegend  = FALSE)
```

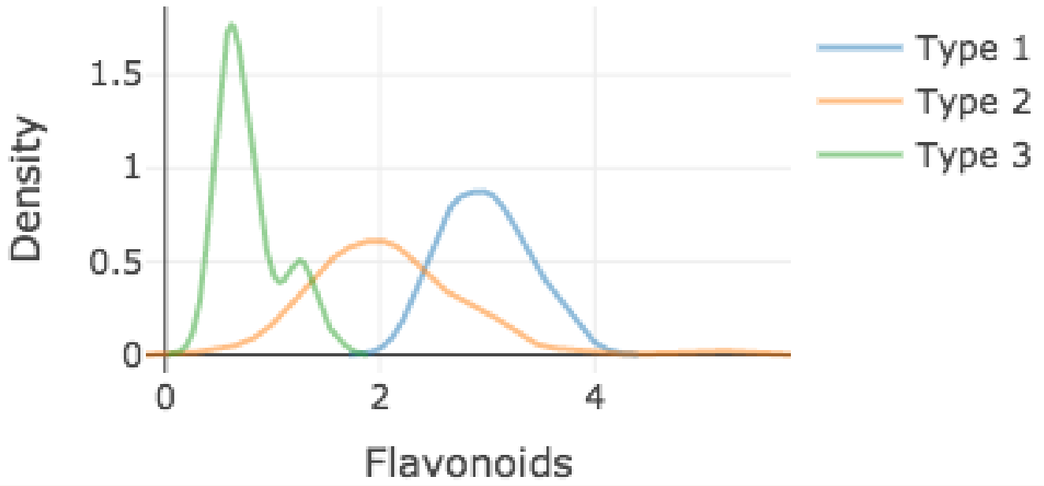
Adding a second smoother




```
m2 <- lm(Alcohol ~ poly(Flavanoids, 2), data = wine)
wine %>%
  plot_ly(x = ~Flavanoids, y = ~Alcohol) %>%
  add_markers(showlegend = FALSE) %>%
  add_lines(y = ~fitted(m), name = "LOESS") %>%
  add_lines(y = ~fitted(m2), name = "Polynomial")
```

Layering densities

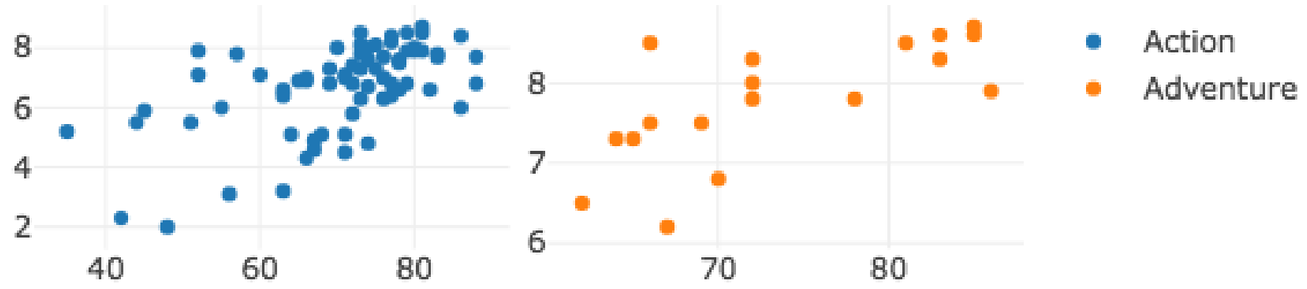
```
d1 <- filter(wine, Type == 1)
d2 <- filter(wine, Type == 2)
d3 <- filter(wine, Type == 3)
density1 <- density(d1$Flavanoids)
density2 <- density(d2$Flavanoids)
density3 <- density(d3$Flavanoids)
plot_ly(opacity = 0.5) %>%
  add_lines(x = ~density1$x, y = ~density1$y, name = "Type 1") %>%
  add_lines(x = ~density2$x, y = ~density2$y, name = "Type 2") %>%
  add_lines(x = ~density3$x, y = ~density3$y, name = "Type 3") %>%
  layout(xaxis = list(title = 'Flavonoids'),
         yaxis = list(title = 'Density'))
```



Arranging plots

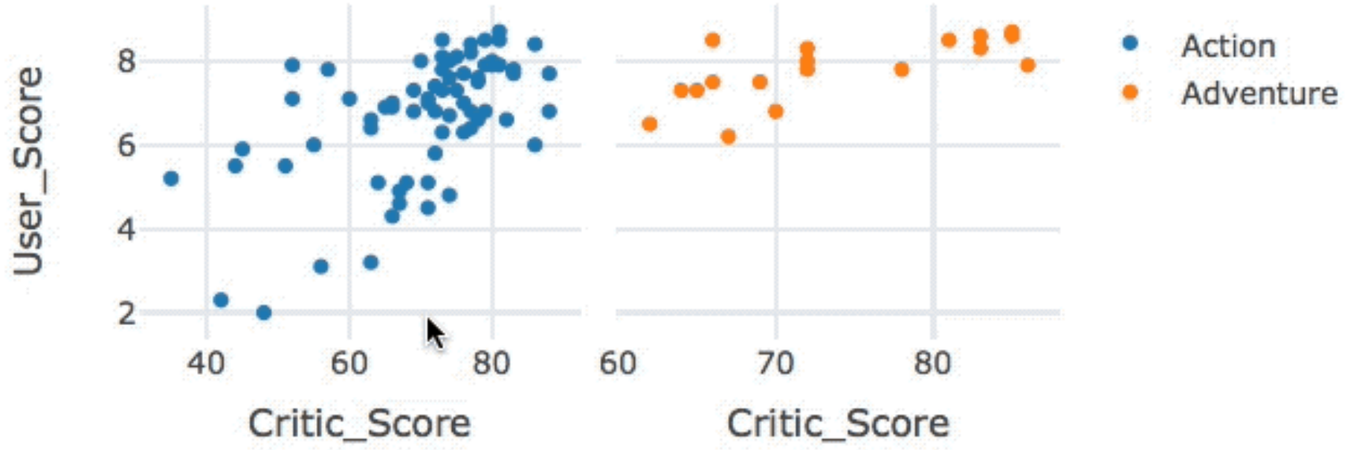
Facing plots manually

```
p1 <- plot_ly(x = ~Critic_Score, y = ~User_Score) %>%
  add_markers(name = ~Genre)

p2 <- vgsales2016 %>%
  filter(Genre == "Adventure") %>%
  plot_ly(x = ~Critic_Score, y = ~User_Score) %>%
  add_markers(name = ~Genre)

subplot(p1, p2, nrows = 1)
```



Defining interactivity of a faced plot

```
subplot(p1, p2, nrows = 1, shareY = TRUE, shareX = TRUE)
```

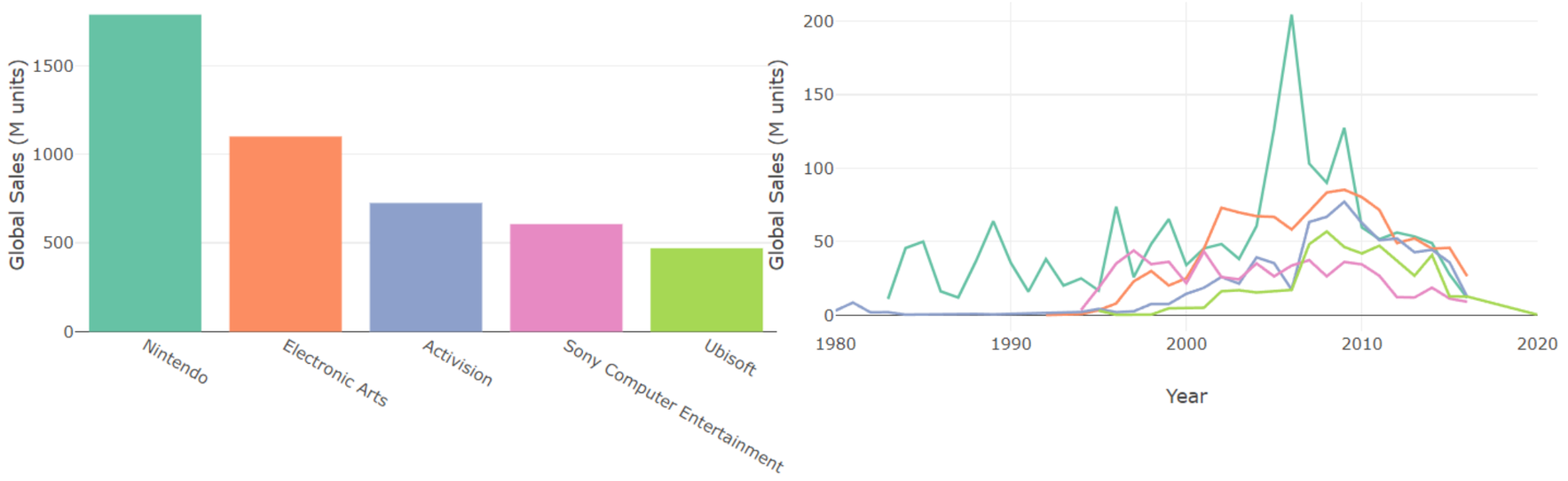


- Sharing an axis leads to linked interactivity
- If linked interactivity is not desired: use `titleX` and `titleY` arguments

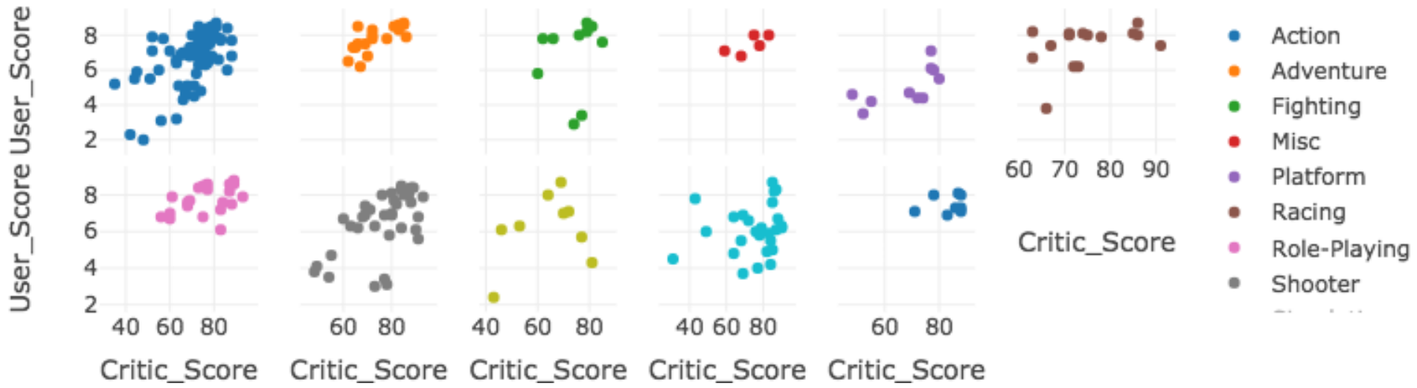
Defining interactivity of a faced plot

```
# Add x-axis and y-axis labels, and a title to  sp2
sp2 %>%
  layout(
    xaxis = list(title = ""),
    xaxis2 = list(title = "Year"),
    yaxis = list(title = "Global Sales (M units)"),
    yaxis2 = list(title = "Global Sales (M units)")
  )
```

Excellent! Clear axis labels are essential when sharing your work. If the y-axis on the left starts to creep into the other plot, consider adding a larger margin around the plots such as `margin = 0.2`.



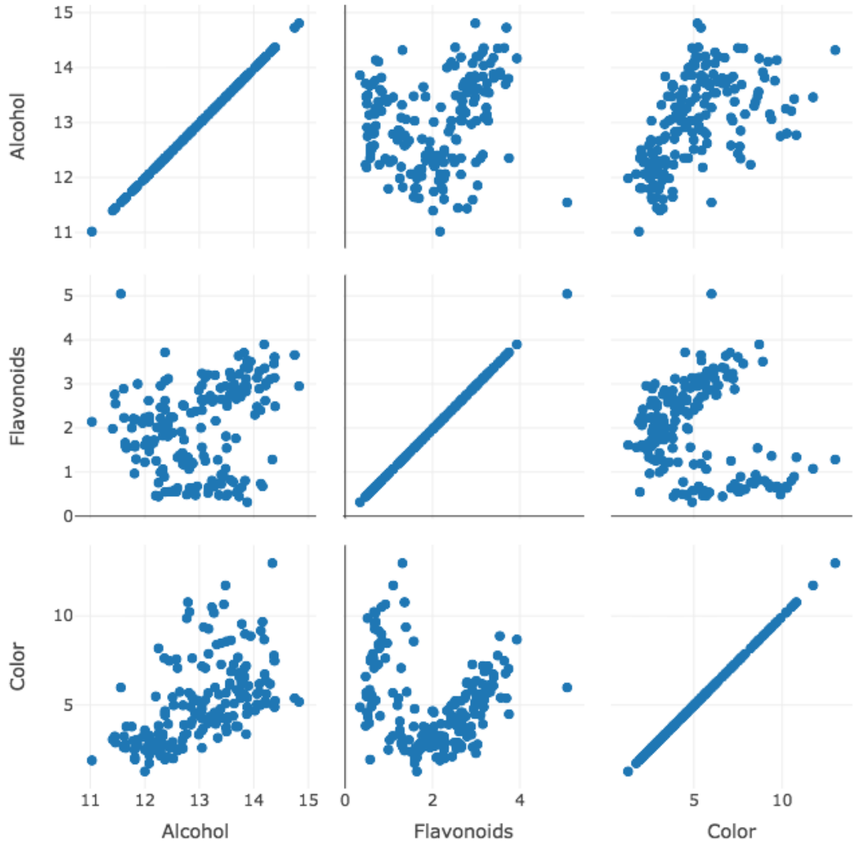
Facing many plots at once




```
library(dplyr)
vgsales2016 %>%
  group_by(region) %>%
  do(plot = plot_ly(data = ., x = ~Critic_Score, y = ~User_Score) %>%
      add_markers(name = ~Genre)
    ) %>%
  subplot(nrows = 2)
```

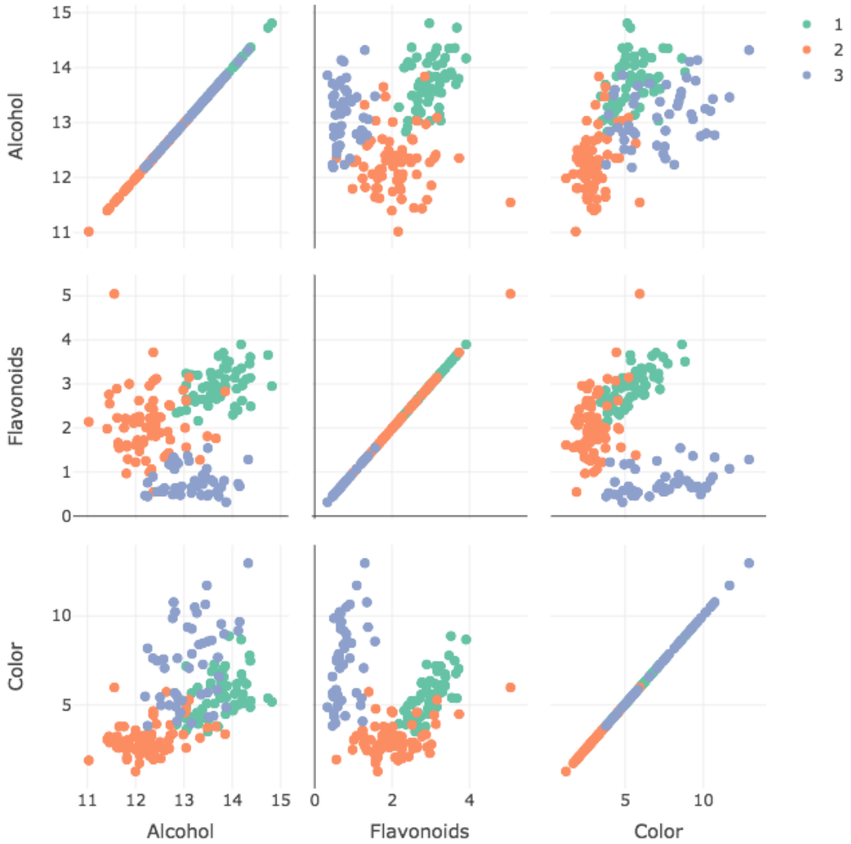
Wine SPLOM

```
wine %>%
  plot_ly() %>%
  add_trace(
    type = 'splom',
    dimensions = list(
      list(label='Alcohol', values=~Alcohol),
      list(label='Flavonoids', values=~Flavanoids),
      list(label='Color', values=~Color)
    )
  )
```



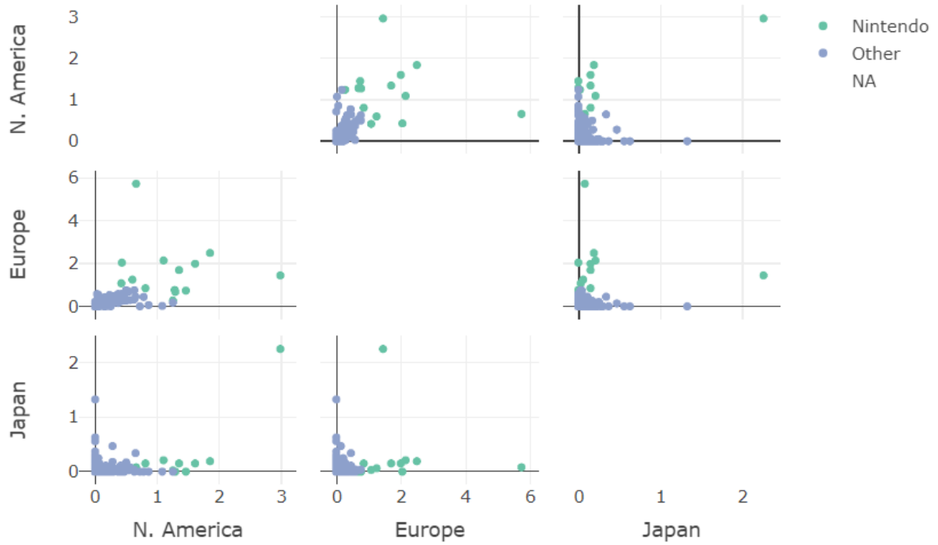
Adding color

```
wine %>%
  plot_ly(color = ~Type) %>%
  add_trace(
    type = 'splom',
    dimensions = list(
      list(label='Alcohol', values=~Alcohol),
      list(label='Flavonoids', values=~Flavanoids),
      list(label='Color', values=~Color)
    )
  )
```



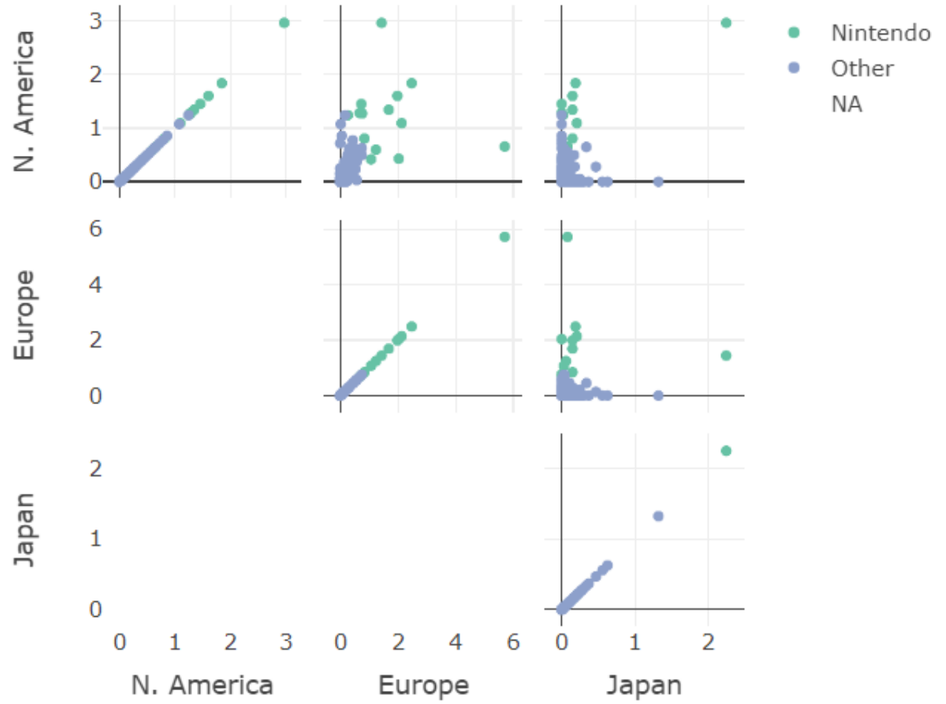
Deleting the diagonal panels

```
wine %>%
  plot_ly(color = ~Type) %>%
  add_trace(
    type = 'splom',
    dimensions = list(
      list(label='Alcohol', values=~Alcohol),
      list(label='Flavonoids', values=~Flavanoids),
      list(label='Color', values=~Color)
    )
  ) %>%
  style(diagonal = list(visible = FALSE))
```



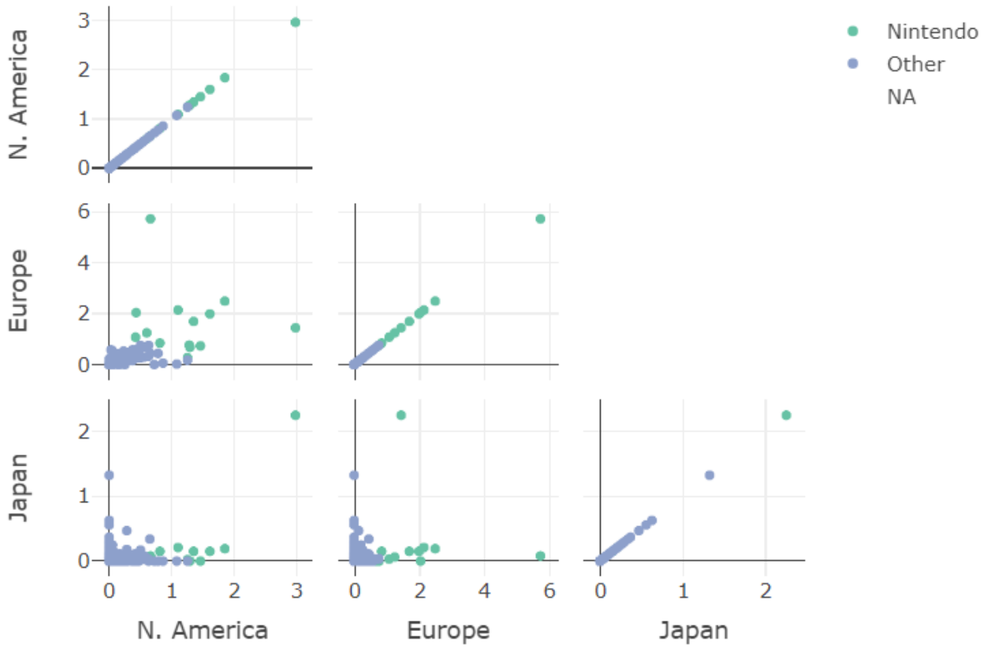
Displaying only the upper triangle of plots

```
wine %>%
  plot_ly(color = ~Type) %>%
  add_trace(
    type = 'splom',
    dimensions = list(
      list(label='Alcohol', values=~Alcohol),
      list(label='Flavonoids', values=~Flavanoids),
      list(label='Color', values=~Color)
    )
  ) %>%
  style(showlowerhalf = FALSE)
```



Displaying only the lower triangle of plots

```
wine %>%
  plot_ly(color = ~Type) %>%
  add_trace(
    type = 'splom',
    dimensions = list(
      list(label='Alcohol', values=~Alcohol),
      list(label='Flavonoids', values=~Flavanoids),
      list(label='Color', values=~Color)
    )
  ) %>%
  style(showupperhalf = FALSE)
```



Well done! If you want a clear and concise scattterplot matrix, consider plotting only the lower triangle. That reduces redundant information while keeping the axis labels and tick marks in a useful location on the chart.

Creating US maps

Mapping options

```
scope = "usa"
```

- "world" | "usa" | "europe" | "asia" | "africa" | "north america" | "south america"

```
projection = list(type = "mercator")
```

- "conic conformal" | "mercator" | "robinson" | "stereographic" | and 18 more...

```
scale = 1
```

- Larger values = tighter zoom

```
center = list(lat = ~c.lat, lon = ~c.lon)
```

- Set `c.lat` and `c.lon` to center the map

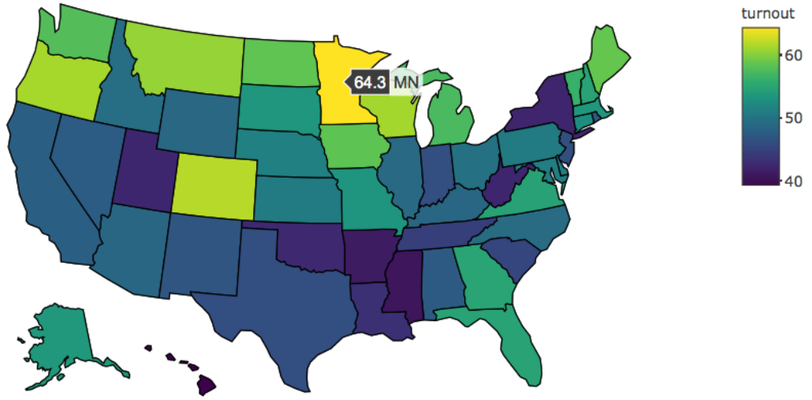
Limitation of plot_geo()

locationmode:

```
"USA-states" | "ISO-3" | "country names"
```

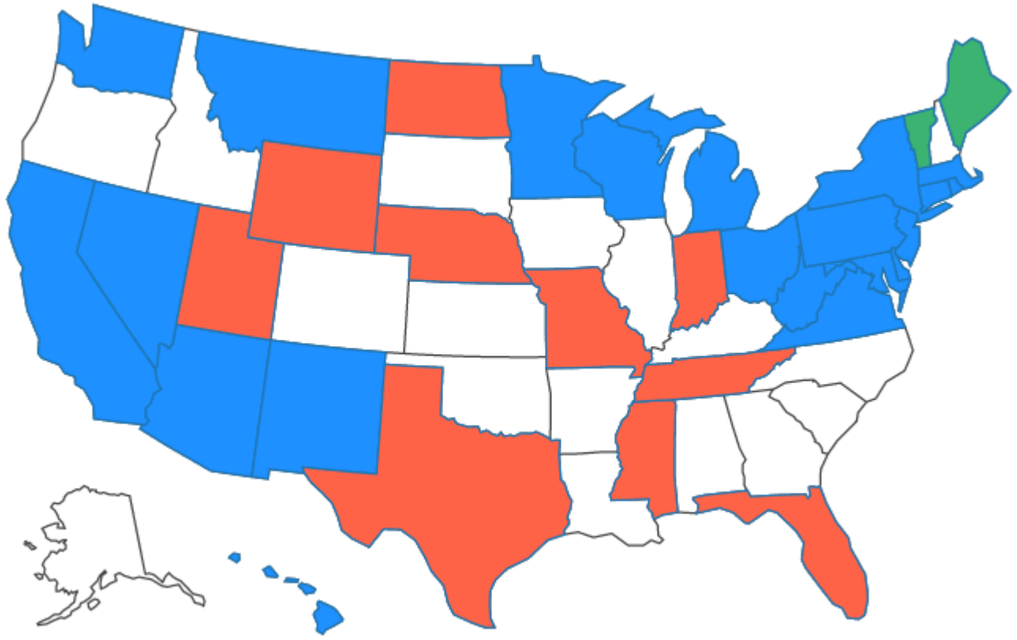
Choropleth maps in plotly

```
turnout %>%
  plot_geo(locationmode = 'USA-states') %>%
  add_trace(
    z = ~turnout,                    # Sets the color values
    locations = ~state.abbr          # Matches cases to polygons
  ) %>%
  layout(geo = list(scope = 'usa')) # Restricts map only to USA
```



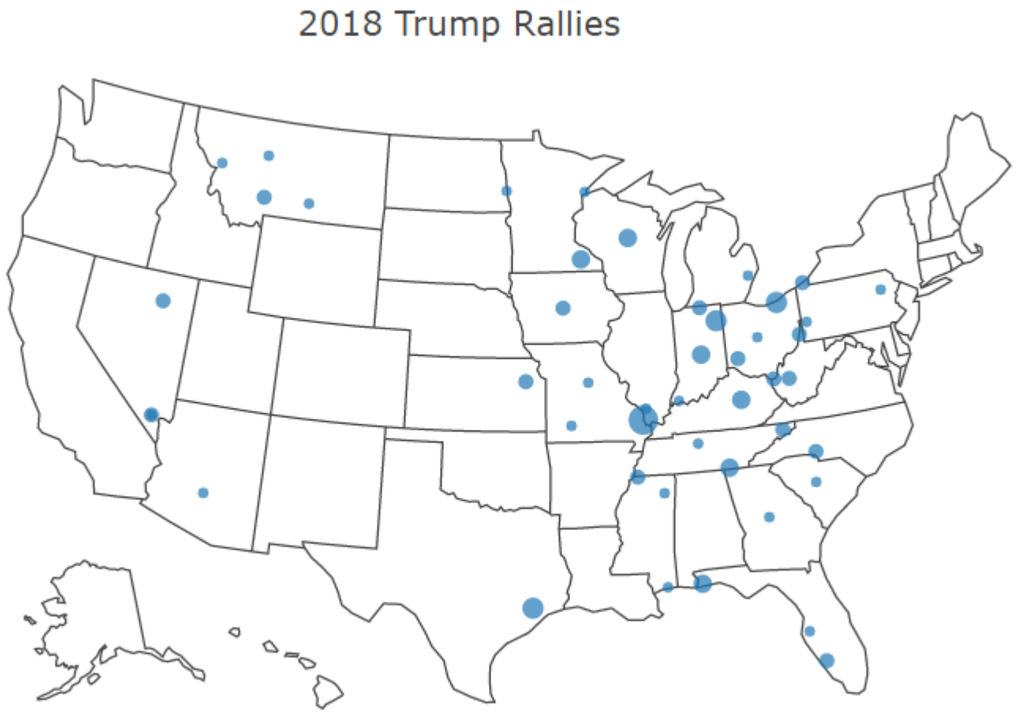
Choropleth maps in plotly

```
# Create a choropleth map displaying the Senate results
# Note: Party is a factor
senate_winners %>%
  plot_geo(locationmode = 'USA-states') %>%
  add_trace(z = ~as.numeric(party), locations = ~state,
    colors = c("dodgerblue", "mediumseagreen", "tomato"),
    hoverinfo = "text",
    text = ~paste("Candidate:", name, "<br>",
                  "Party:", party, "<br>",
                  "% vote:", round(pct.vote, 1))
  ) %>%
  layout(geo = list(scope = 'usa')) %>%
  hide_colorbar()
```



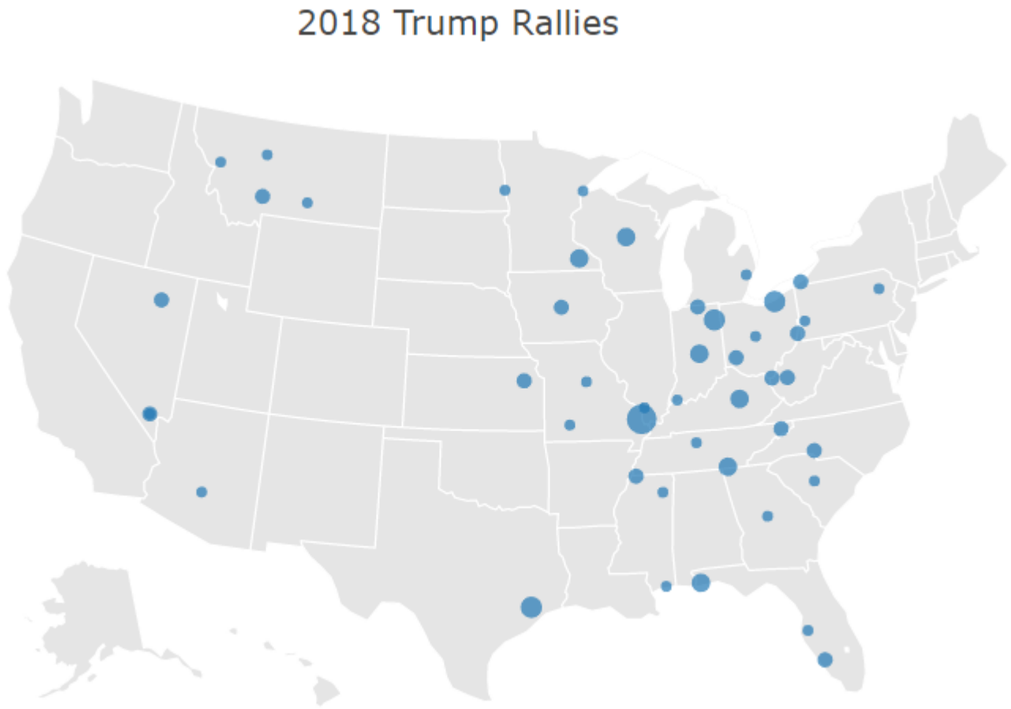
Plotting points

```
# Map President Trump's rallies in 2018
rallies2018 %>%
  plot_geo(locationmode = 'USA-states') %>%
  add_markers(
    x  = ~long, y = ~lat, size = ~no.speakers,
    hoverinfo = "text", text = ~paste(city, state, sep = ",")
  ) %>%
  layout(title = "2018 Trump Rallies",
         geo = list(scope = 'usa'))
```



Geo options

```
# Customize the geo layout
g <- list(scope = 'usa',
          showland = TRUE, landcolor = toRGB("gray90"),
          showlakes = TRUE, lakecolor = toRGB("white"),
          showsubunit = TRUE, subunitcolor = toRGB("white"))

# Apply the geo layout to the map
rallies2018 %>%
  plot_geo(locationmode = 'USA-states') %>%
  add_markers(
    x = ~long, y = ~lat, size = ~no.speakers,
    hoverinfo = "text", text = ~paste(city, state, sep = ",")
  ) %>%
  layout(title = "2018 Trump Rallies", geo = g)
```



More options

Joining data frames

```
glimpse(us_states)
```

```
Observations: 15,537
Variables: 5
$ long   <dbl> -87.46201, -87.48493, -87.52503, ...
$ lat    <dbl> 30.38968, 30.37249, 30.37249, ...
$ group  <dbl> 1, 1, 1, ...
$ order  <int> 1, 2, 3, ...
$ region <chr> "alabama", "alabama", "alabama", ...
```

```
glimpse(turnout)
```

```
Observations: 51
Variables: 7
$ state       <fct> Alabama, Alaska, Arizona, Ar...
$ state.abbr  <fct> AL, AK, AZ, AR, ...
$ turnout2018 <dbl> 0.474, 0.537, 0.486, 0.412, ...
$ turnout2014 <dbl> 0.332, 0.548, 0.341, 0.403, ...
$ ballots     <int> 1725000, 280000, 2385000, 89...
$ vep         <int> 3641209, 521777, 4910625, 21...
$ vap         <int> 3802714, 554426, 5519036, 23...
```

Joining data frames

```
turnout <- turnout %>%
  mutate(state = tolower(state)) # make state names lowercase
states_map <- left_join(us_states, turnout, by = c("region" = "state"))
```

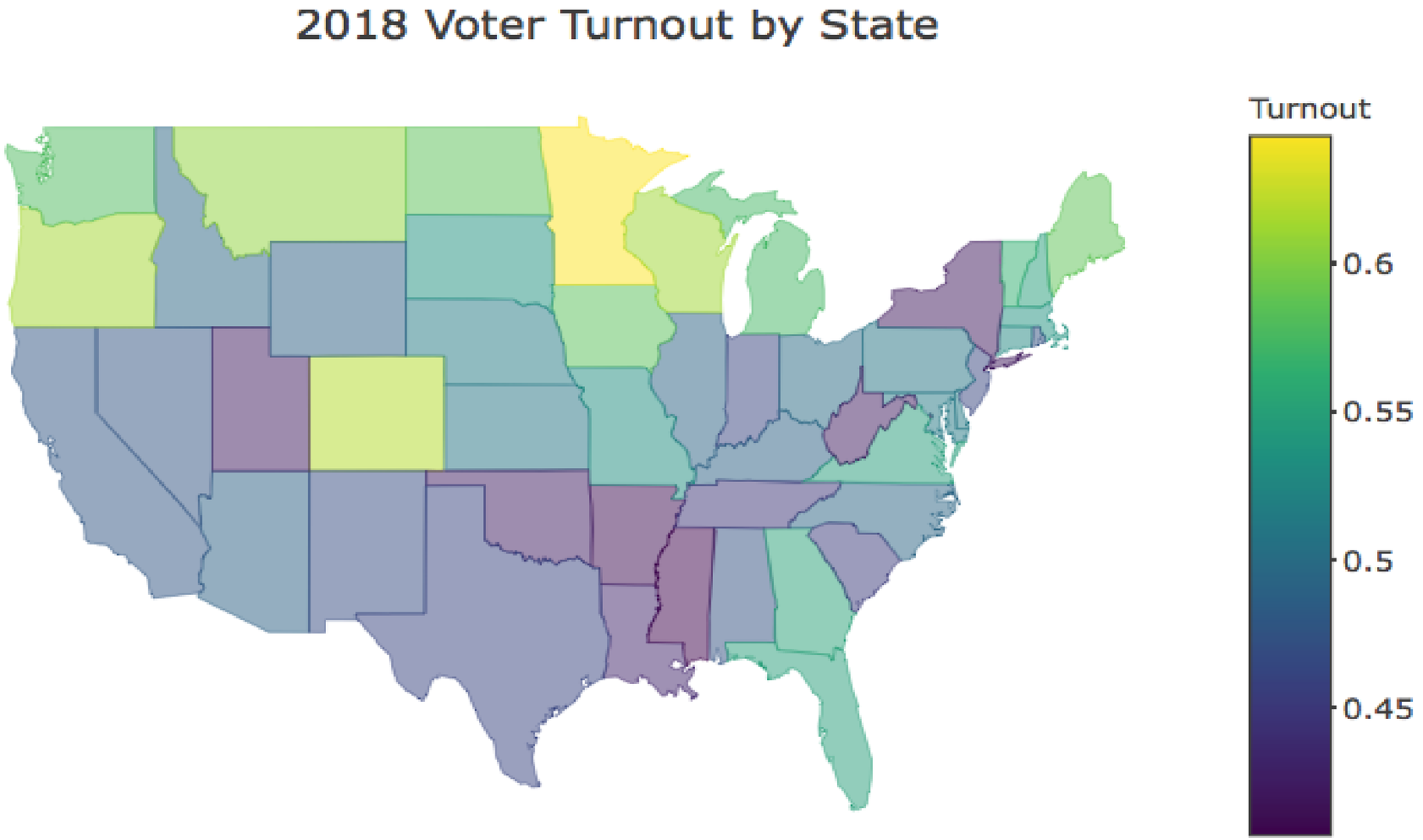
```
Observations: 15,537
Variables: 11
$ long        <dbl> -87.46201, -87.48493, -87.52503, -87.53076...
$ lat         <dbl> 30.38968, 30.37249, 30.37249, 30.33239, 30...
$ group       <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
$ order       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...
$ region      <chr> "alabama", "alabama", "alabama", "alabama"...
$ state.abbr  <fct> AL, AL, AL, AL, AL, AL, AL, AL, AL, AL, AL...
$ turnout2018 <dbl> 0.474, 0.474, 0.474, 0.474, 0.474, 0.474, ...
$ turnout2014 <dbl> 0.332, 0.332, 0.332, 0.332, 0.332, 0.332, ...
...
```

From polygons to map

```
states_map %>%
  group_by(group) %>%
  plot_ly(
    x = ~long,
    y = ~lat,
    color = ~turnout2018, # variable mapped to fill color
    split = ~region       # no more than one fill color per polygon
  ) %>%
  add_polygons(
    line = list(width = 0.4),
    showlegend = FALSE
  )
```

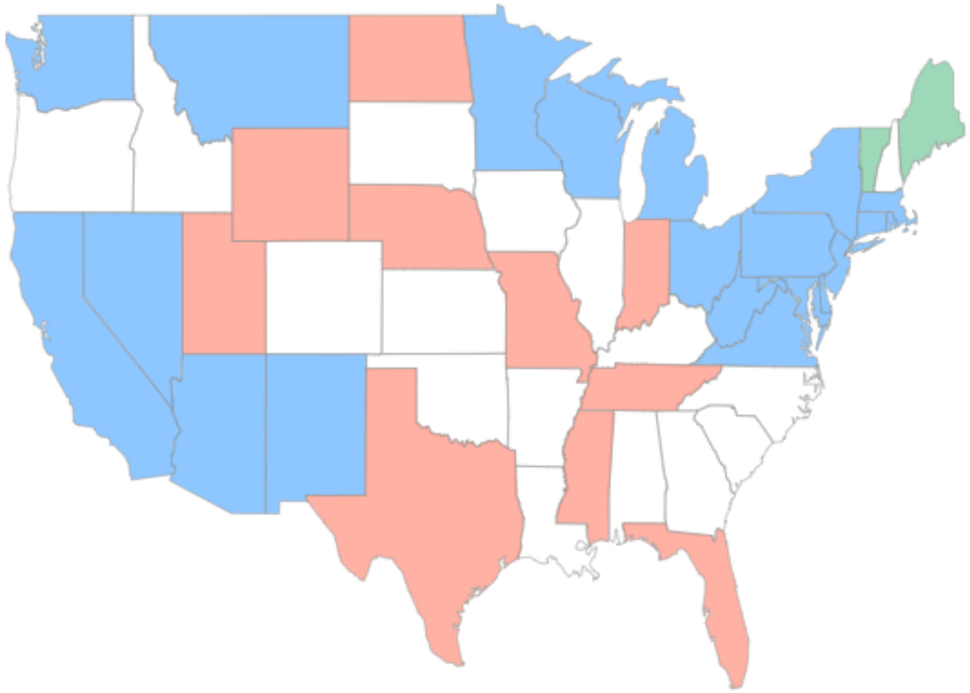
Polishing your map

```
state_turnout_map %>%
  layout(
    title = "2018 Voter Turnout by State",
    xaxis = list(title = "", showgrid = FALSE,
                 zeroline = FALSE, showticklabels = FALSE),
    yaxis = list(title = "", showgrid = FALSE,
                 zeroline = FALSE, showticklabels = FALSE)
  ) %>%
  colorbar(title = "Turnout")
```


2018 Voter Turnout by State
Turnout
0.6
0.55
0.5
0.45

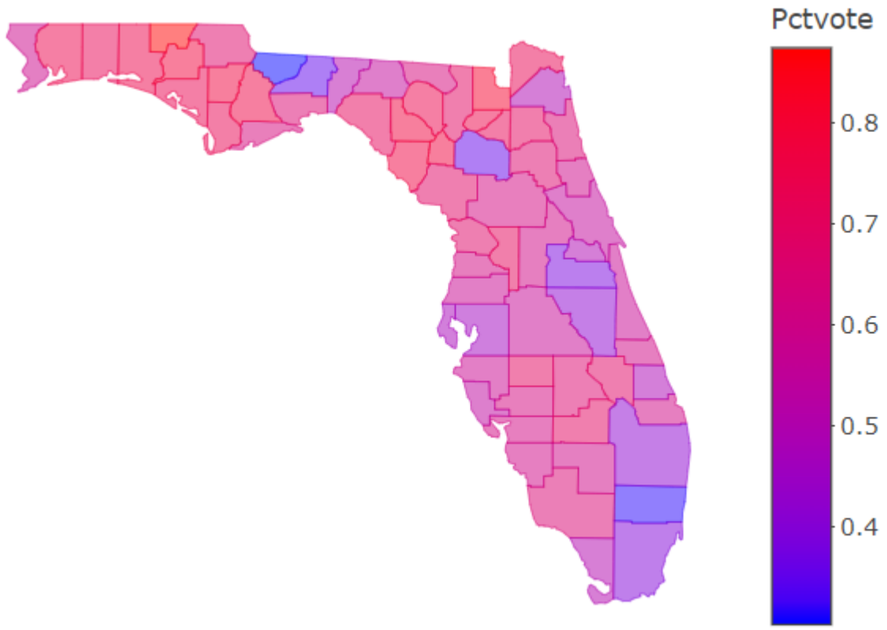
Polishing your map

```
# Define the layout settings to polish the axes
map_axes <- list(titles = "", showgrid = FALSE,
                 zeroline = FALSE,showticklabels = FALSE)

# Apply the layout to both axes
senate_map %>%
  group_by(group) %>%
  plot_ly(x = ~long, y = ~lat, color = ~party, split = ~region,
          colors = c("dodgerblue", "mediumseagreen", "tomato")) %>%
  add_polygons(line = list(width = 0.4, color = toRGB("gray60")),
               showlegend = FALSE) %>%
  layout(xaxis = map_axes,
         yaxis = map_axes)
```



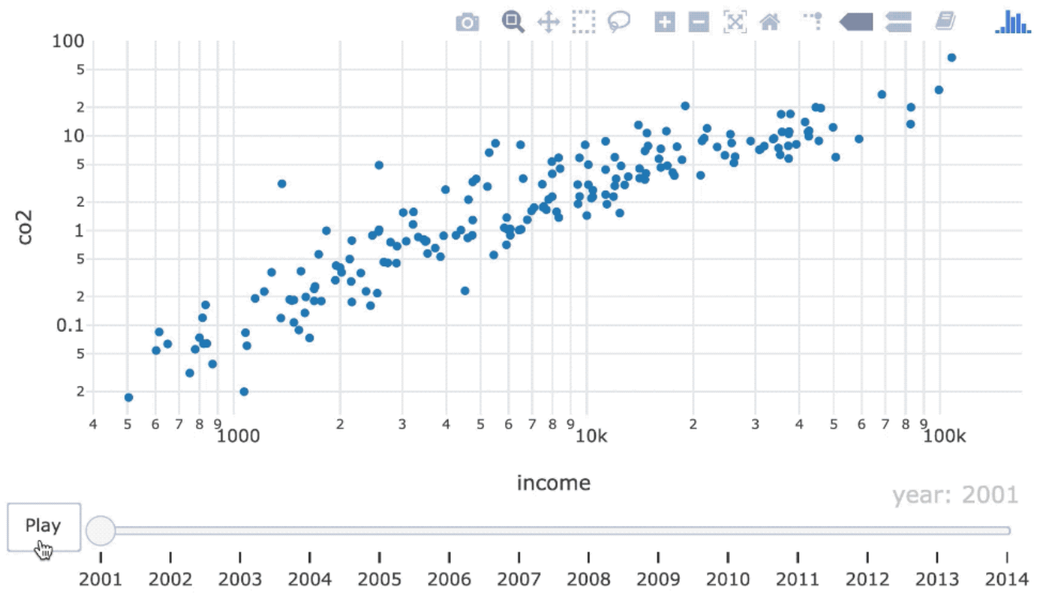
Polishing your map

```
# Join the fl_boundaries and fl_results data frames
senate_vote <- left_join(fl_boundaries, fl_results,
                         by = c("subregion" = "CountyName"))

# Specify the axis settings to polish the map
map_axes <- list(titles = "", showgrid = FALSE,
                 zeroline = FALSE, showticklabels = FALSE)

# Create a polished county-level choropleth map of Pctvote
senate_vote %>%
  group_by(group) %>%
  plot_ly(x = ~long, y = ~lat,
          color = ~Pctvote,
          split = ~subregion) %>%
  add_polygons(line = list(width = 0.4),
               showlegend = FALSE,
               colors = c("blue", "red")) %>%
  layout(xaxis = map_axes, yaxis = map_axes)
```



Animating charts

Animation options



transition: specifies the duration of the smooth transition between frames in milliseconds. By default, this is set equal to the frame, resulting in a smooth transition between frames with no focus on a static graphic.

frame: specifies (milliseconds) between frames, including the time used to transition between frames.

easing argument specifies the type of transition used between frames.

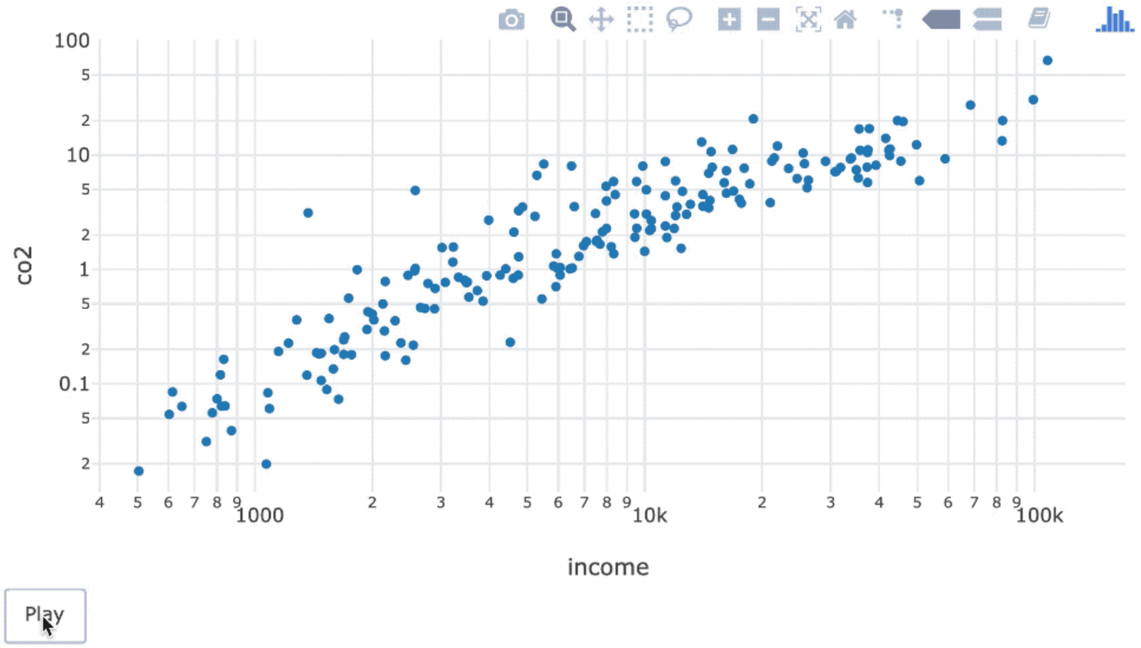
redraw specifies whether the entire graphic should be redrawn. In some cases specifying redraw = FALSE will result in far smoother transitions.

```
ani <- world_indicators %>%
  plot_ly(x = ~income, y = ~co2) %>%
  add_markers(frame = ~year,
              ids = ~country,
              showlegend = FALSE) %>%
  layout(xaxis = list(type = "log"),
         yaxis = list(type = "log")) %>%
  animation_opts(
      frame = 500,
      transition = frame,
      easing = "linear",
      redraw = TRUE )
```

Animation (easing) options

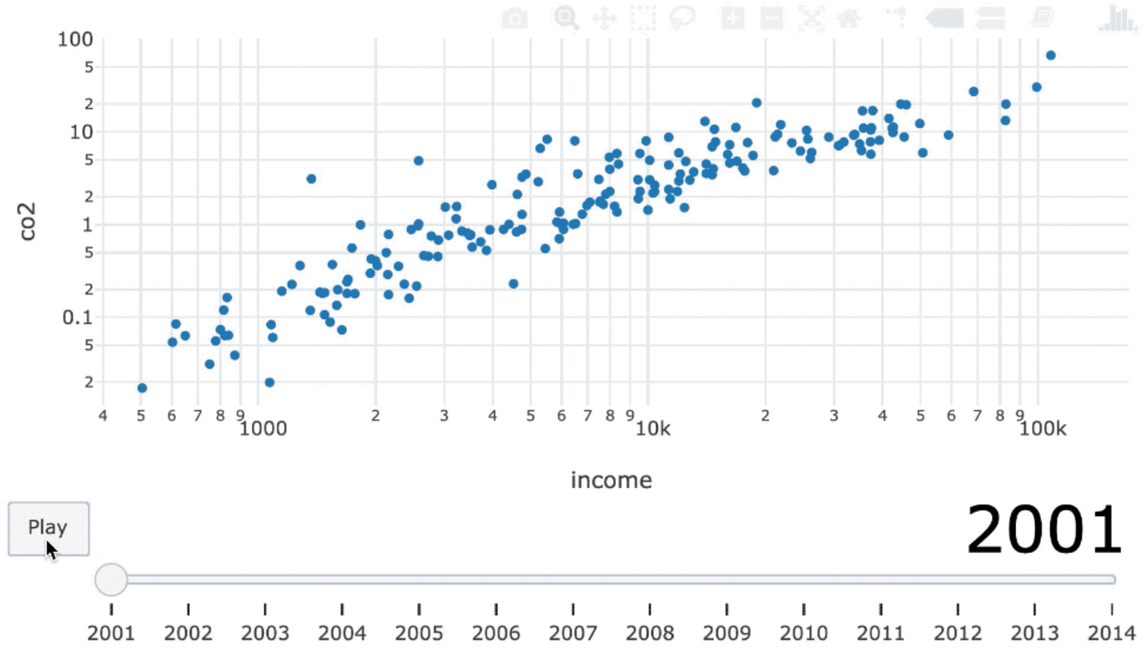
'linear'	'quad'	'cubic'	'sin'	'exp'	'circle'	'elastic'	'back'	'bounce'	Suffix
									-in
									-out
									-in-out

Removing the slider




```
ani %>%
  animation_slider(hide = TRUE)
```

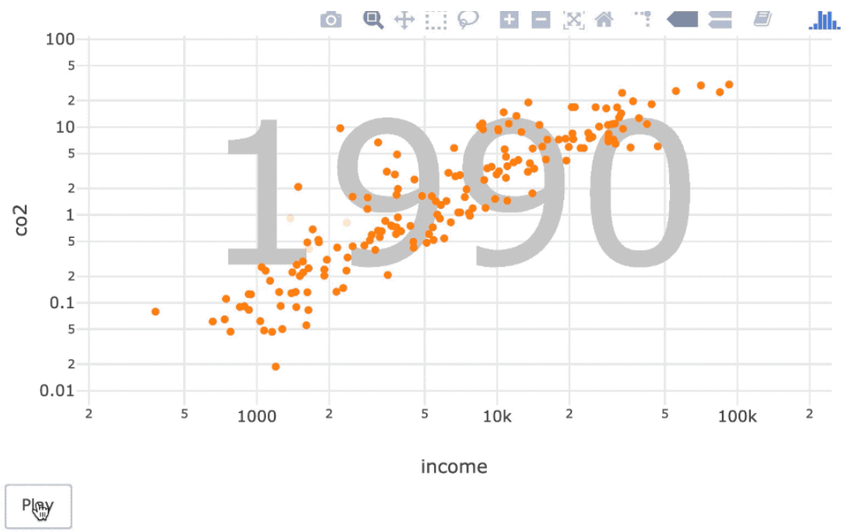
Editing slider text




```
ani %>%
  animation_slider(
     currentvalue = list(prefix=NULL,
        font = list(
           color = "black",
           size = 40
        )
     )
  )
```

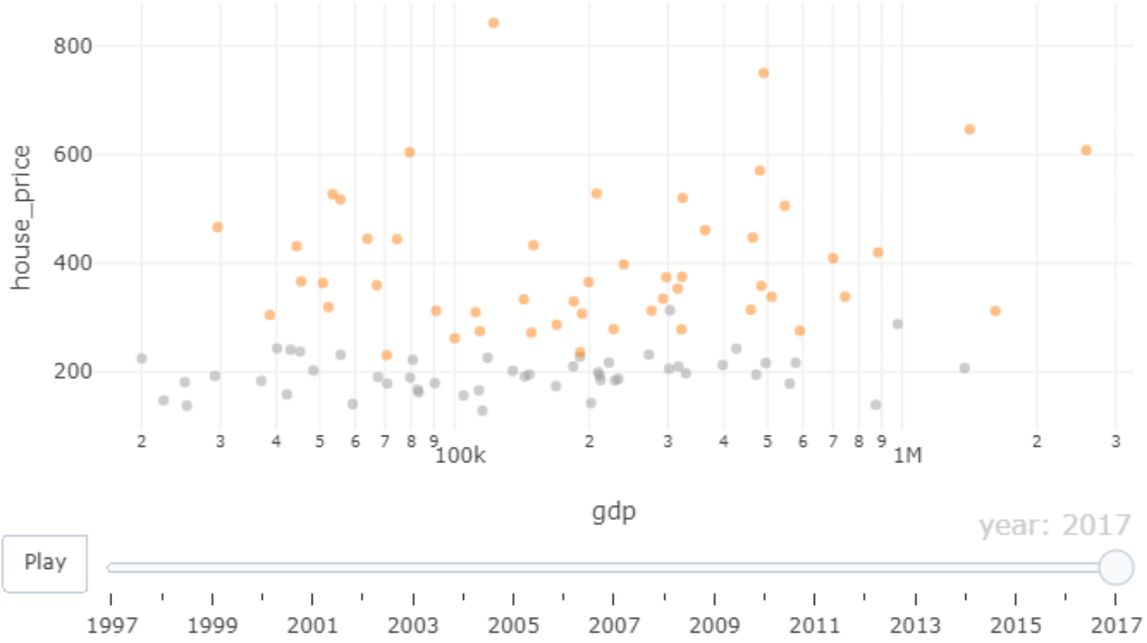
Adding animated layers

```
world_indicators %>%
  plot_ly(x = ~income, y = ~co2) %>%
  add_text(
    x = 6500, y = 1, text = ~year, frame = ~year,
    textfont = list(size = 150, color = toRGB("gray80"))
  ) %>%
  add_markers(frame = ~year, ids = ~country) %>%
  layout(
    xaxis = list(type = "log"), yaxis = list(type = "log"),
    showlegend = FALSE
  ) %>%
  animation_slider(hide = TRUE)
```



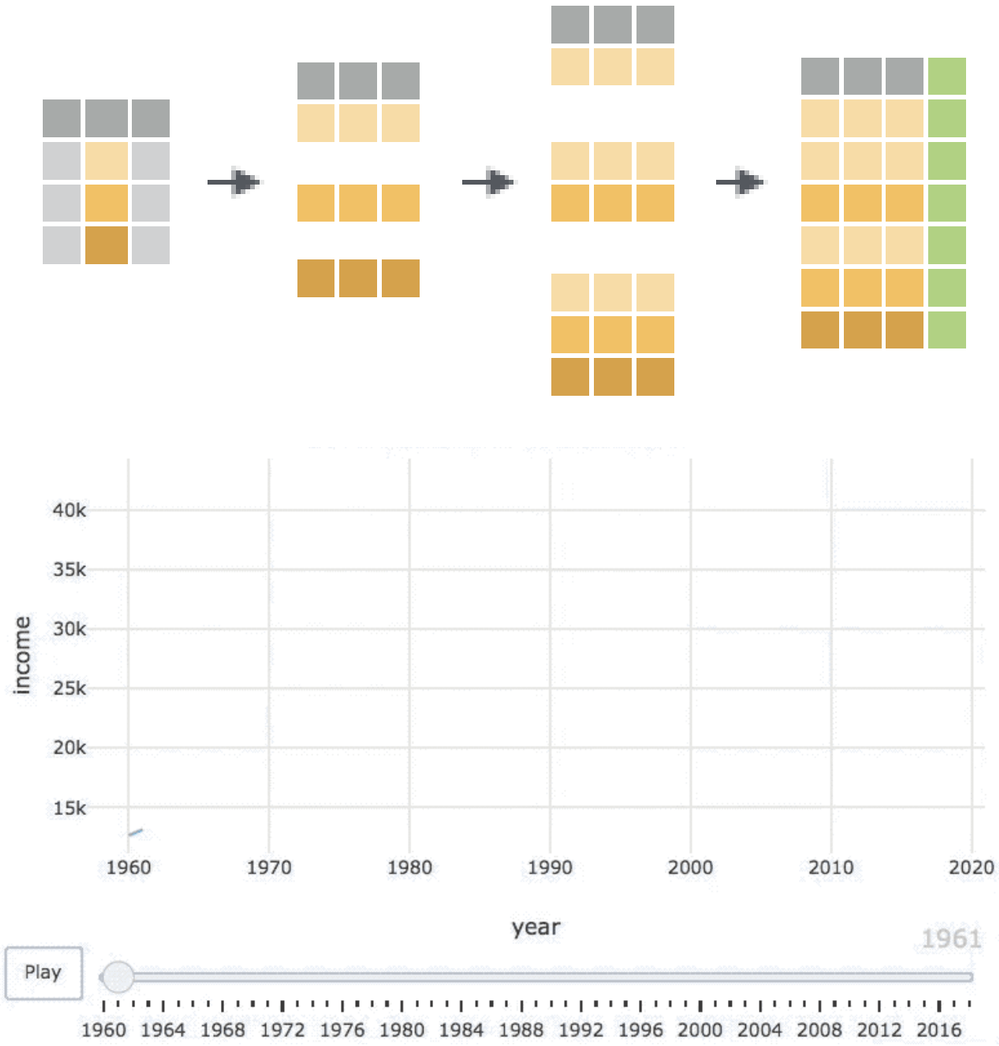
Adding a base line

```
# extract the 1997 data
us1997 <- us_economy %>%
  filter(year == 1997)

# create an animated scatterplot with baseline from 1997
us_economy %>%
  plot_ly(x = ~gdp, y = ~house_price) %>%
  add_markers(data = us1997,
              marker = list(color = toRGB("gray60"),
              opacity = 0.5)) %>%
  add_markers(frame = ~year, ids = ~state,
              data = us_economy,
              showlegend = FALSE, alpha = 0.5) %>%
  layout(xaxis = list(type = "log"))
```



Cumulative animation

```
library(dplyr)
library(purrr)
belgium %>%
  split(.$year) %>%
  accumulate(~bind_rows(.x, .y)) %>%
  set_names(1960:2018) %>%
  bind_rows(.id = "frame") %>%
  plot_ly(x = ~year, y = ~income) %>%
  add_lines(
    frame = ~frame, showlegend = FALSE
  )
```



What if we ignore the baseline issue?

```
monthly_logs %>%
  split(f = .$dec_date) %>%
  accumulate(., ~bind_rows(.x, .y)) %>%
  bind_rows(.id = "frame") %>%
  plot_ly(x = ~dec_date, y = ~downloads) %>%
  add_lines(color = ~package, frame = ~frame, ids = ~package)
```

```
Warning message:
In p$x$data[firstFrame] <- p$x$frames[[1]]$data :
  number of items to replace is not a multiple of replacement length
```

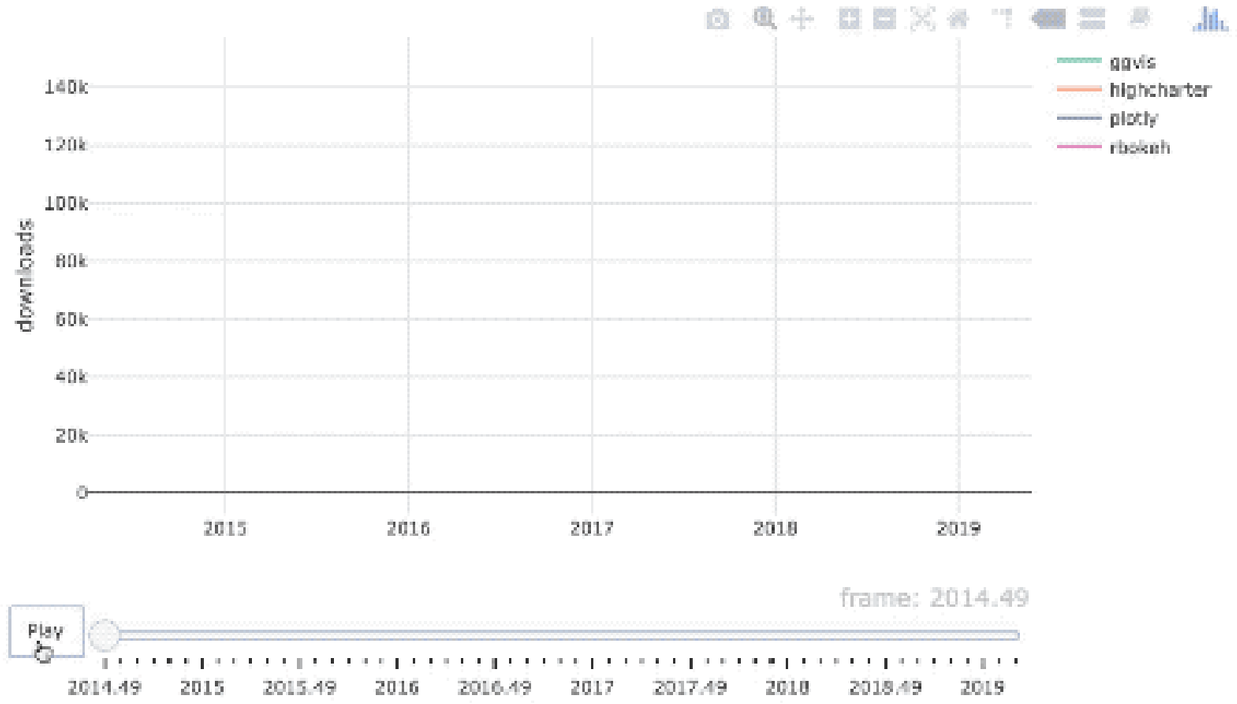
Completing the data set

```
library(tidyr)
complete_logs <- monthly_logs %>% complete(package, dec_date, fill = list(downloads = 0))
arrange(complete_logs, dec_date)
```

```
## A tibble: 228 x 4
  package     dec_date date       downloads
  <chr>          <dbl> <date>         <dbl>
1 ggvis          2014. 2014-06-30      1344
2 highcharter    2014. NA                 0
3 plotly         2014. NA                 0
4 rbokeh         2014. NA                 0
5 ggvis          2015. 2014-07-31      2120
# … with 223 more rows
```

Animating the completed data

```
complete_logs %>%
  split(f = .$dec_date) %>%
  accumulate(., ~bind_rows(.x, .y)) %>%
  bind_rows(.id = "frame") %>%
  plot_ly(x = ~dec_date, y = ~downloads) %>%
  add_lines(color = ~package, frame = ~frame, ids = ~package)
```



Linking views with crosstalk

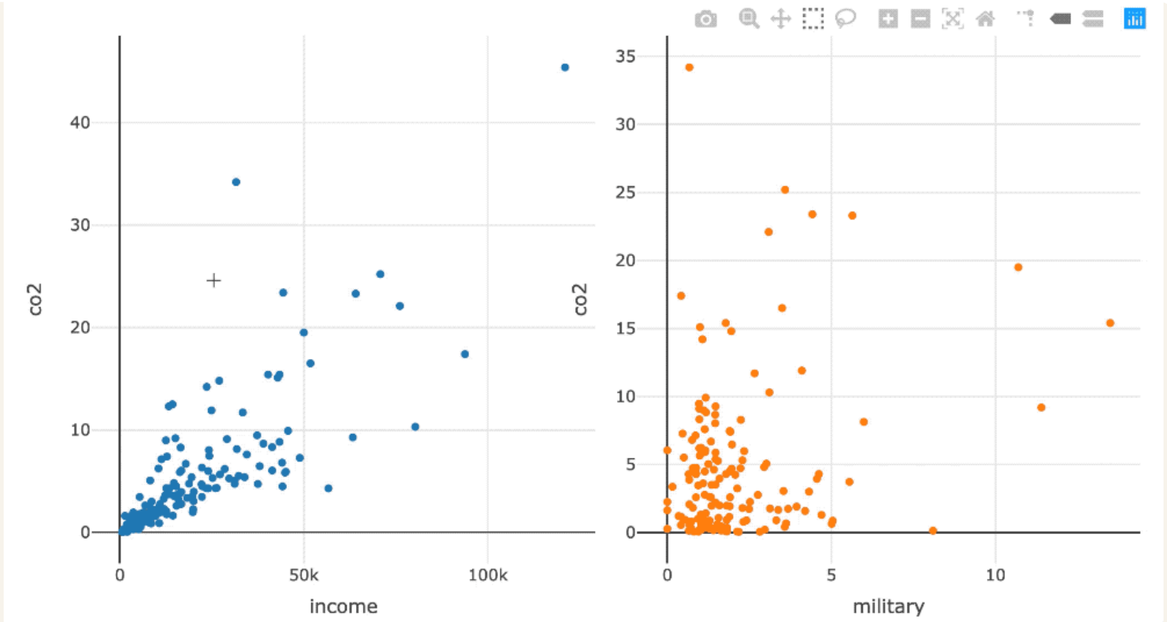
highlight() options

Argument	Description
`on`	selection event: `'plotly_click'`, `'plotly_hover'` or `'plotly_selected'`
`off`	event to turn o selection: `'plotly_doubleclick'`, `'plotly_deselect'`, or `'plotly_relayout'`
`persistent`	Should selections be persisent? `TRUE` or `FALSE`
`dynamic`	Add a widget to change colors? `TRUE` or `FALSE`
`color`	string of color(s) to use for highlighting selections
`selectize`	Add a selectize.js widget for selecting keys? `TRUE` or `FALSE`

selected = attrs_selected(opacity = 0.3)

Linked views

```
library(crosstalk)
shared_data <- SharedData$new(world2014)
p1 <- shared_data %>%
  plot_ly(x = ~income, y = ~co2) %>%
  add_markers()
p2 <- shared_data %>%
  plot_ly(x = ~military, y = ~co2) %>%
  add_markers()
subplot(p1, p2, titleX = TRUE, titleY = TRUE) %>%
  hide_legend() %>%
  highlight(on = "plotly_selected")
```

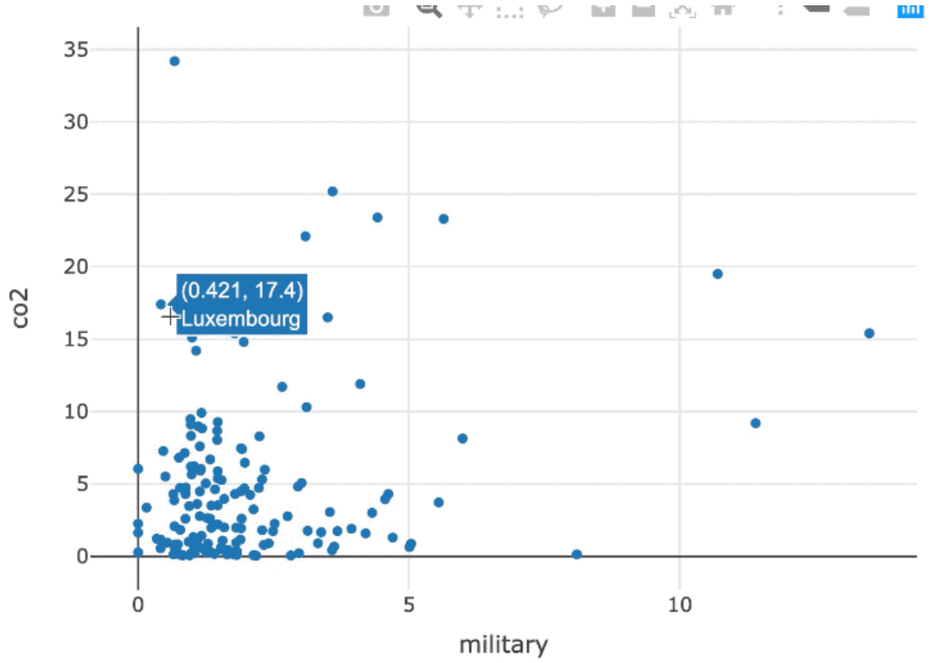


selectize = TRUE: agrega una lista desplegable para seleccionar Distintas categorías

persistent = TRUE: Keeps many things selected at first time.

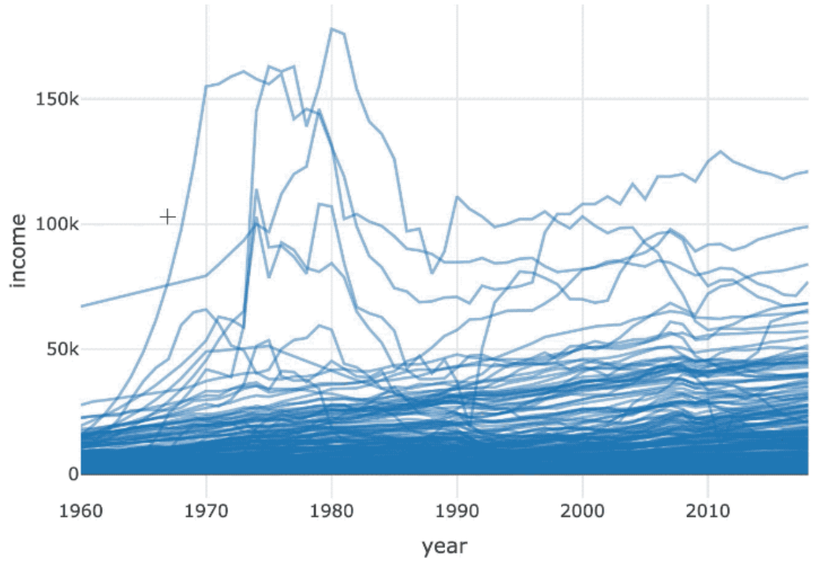
dynamic = TRUE: we can change color by selecting

Selecting groups of points on a scatterplot




```
world_indicators %>%
  filter(year == 2014) %>%
  SharedData$new(~six_regions) or  highlight_key(~six_regions) %>%
  plot_ly(x=~military, y = ~co2, text = ~country) %>%
  add_markers()
```

Selecting individual time series

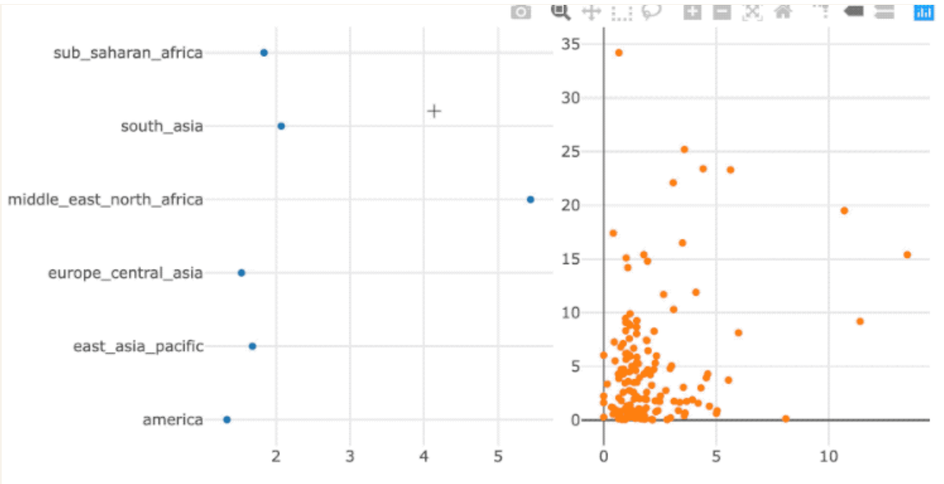


Create a `SharedData` object with a key

```
world_indicators %>%
  SharedData$new(key = ~country) %>%
  plot_ly(x = ~year, y = ~income, alpha = 0.5) %>%
  group_by(country) %>%
  add_lines()
```

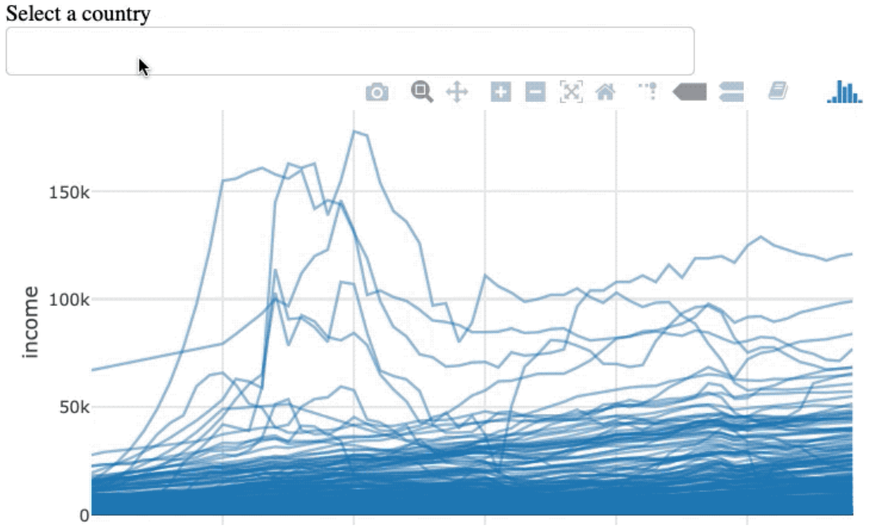
Linking a summary and detailed view

```
shared_data <- world_indicators %>%
  filter(year == 2014) %>%
  SharedData$new(key = ~six_regions)
p1 <- shared_data %>%
  plot_ly() %>%
  group_by(six_regions) %>%
  summarize(avg.military = mean(military, na.rm = TRUE)) %>%
  add_markers(x = ~avg.military, y = ~six_regions)
p2 <- shared_data %>%
  plot_ly(x=~military, y = ~co2, text = ~country) %>%
  add_markers()
subplot(p1, p2) %>% hide_legend()
```



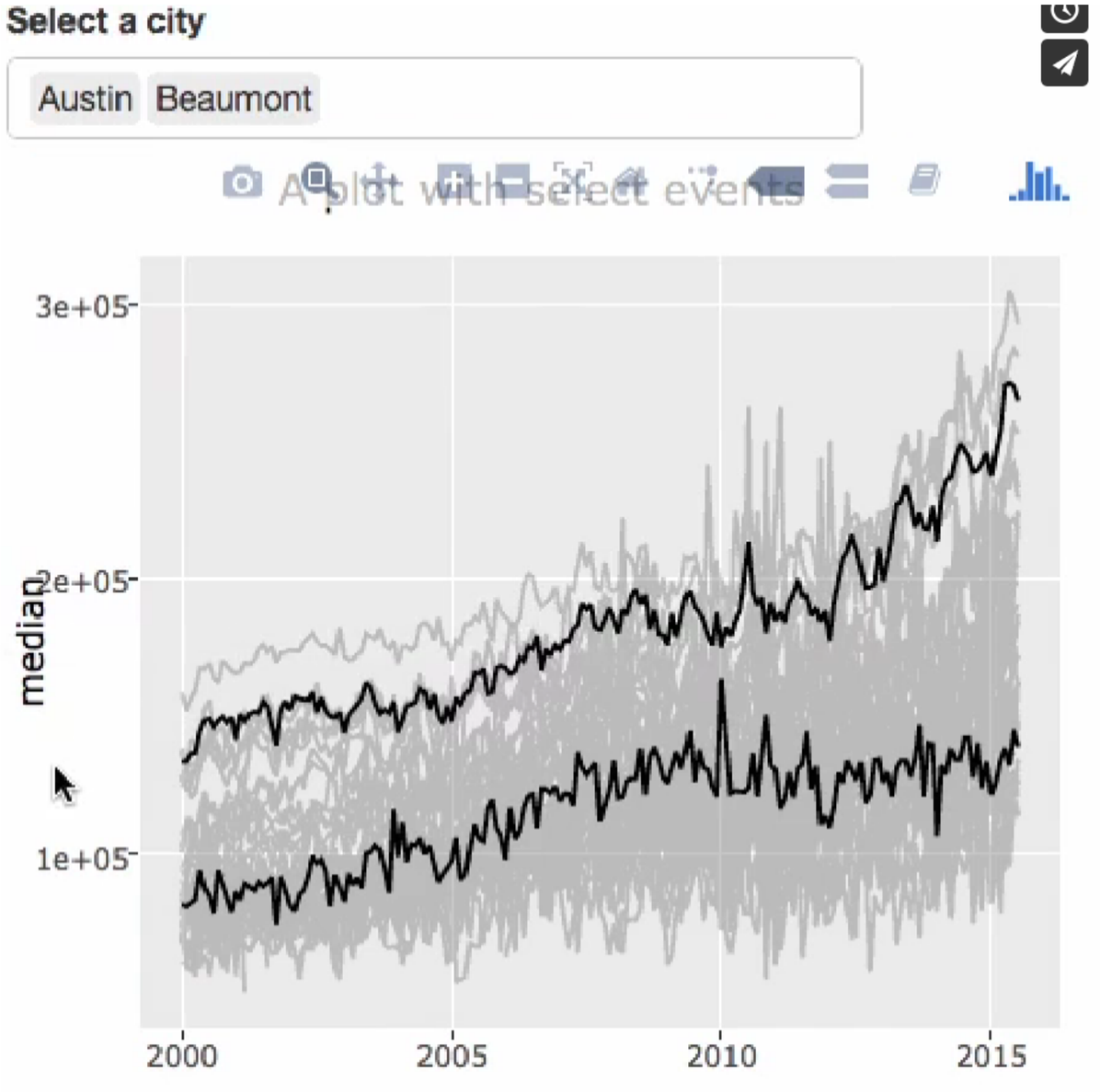
Indirect manipulation

```
world_indicators %>%
  SharedData$new(key = ~country, group = "Select a country") %>%
  plot_ly(x = ~year, y = ~income, alpha = 0.5) %>%
  group_by(country) %>%
  add_lines() %>%
  highlight(selectize = TRUE)
```



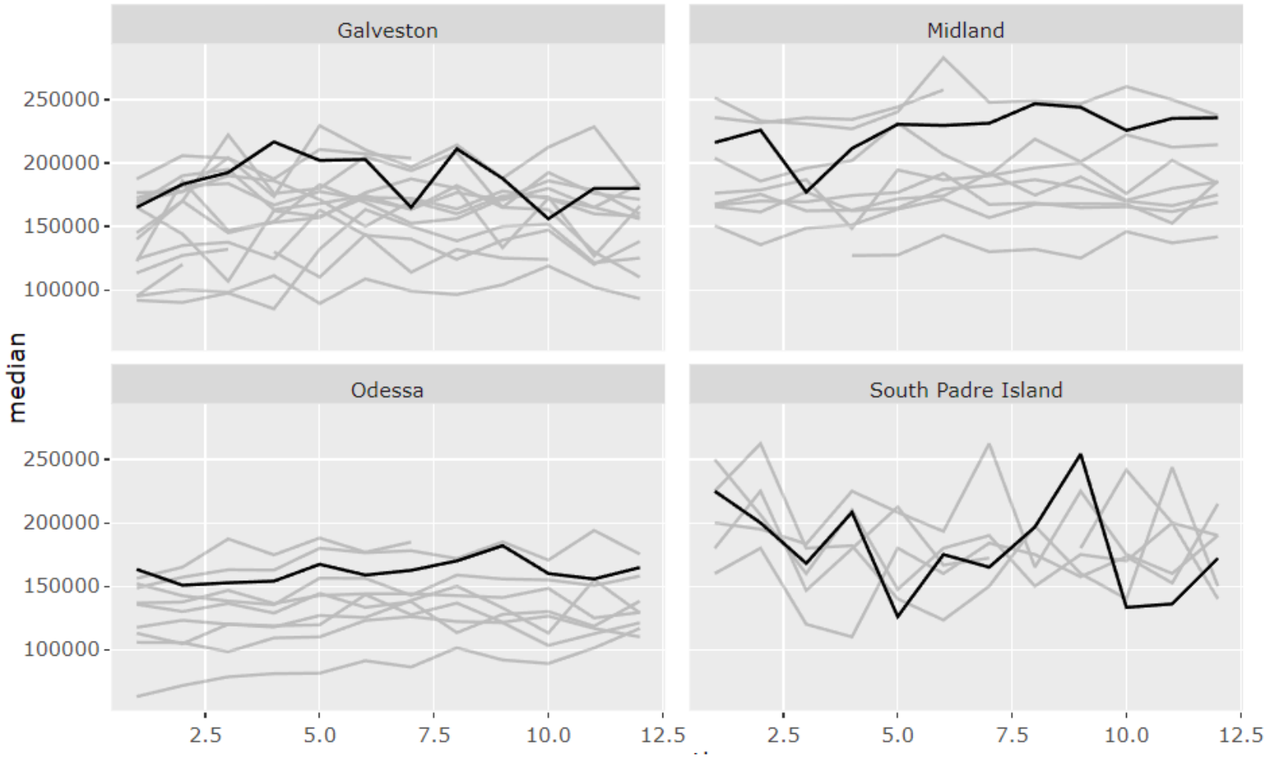
Using ggplot2

```
tx2 <- highlight_key(txhousing, ~city, "Select a city")
gg <- ggplot(tx2) + geom_line(aes(date, median, group = city))
select <- highlight(
  ggplotly(gg, tooltip = "city"),
  selectize = TRUE, persistent = TRUE
)
```



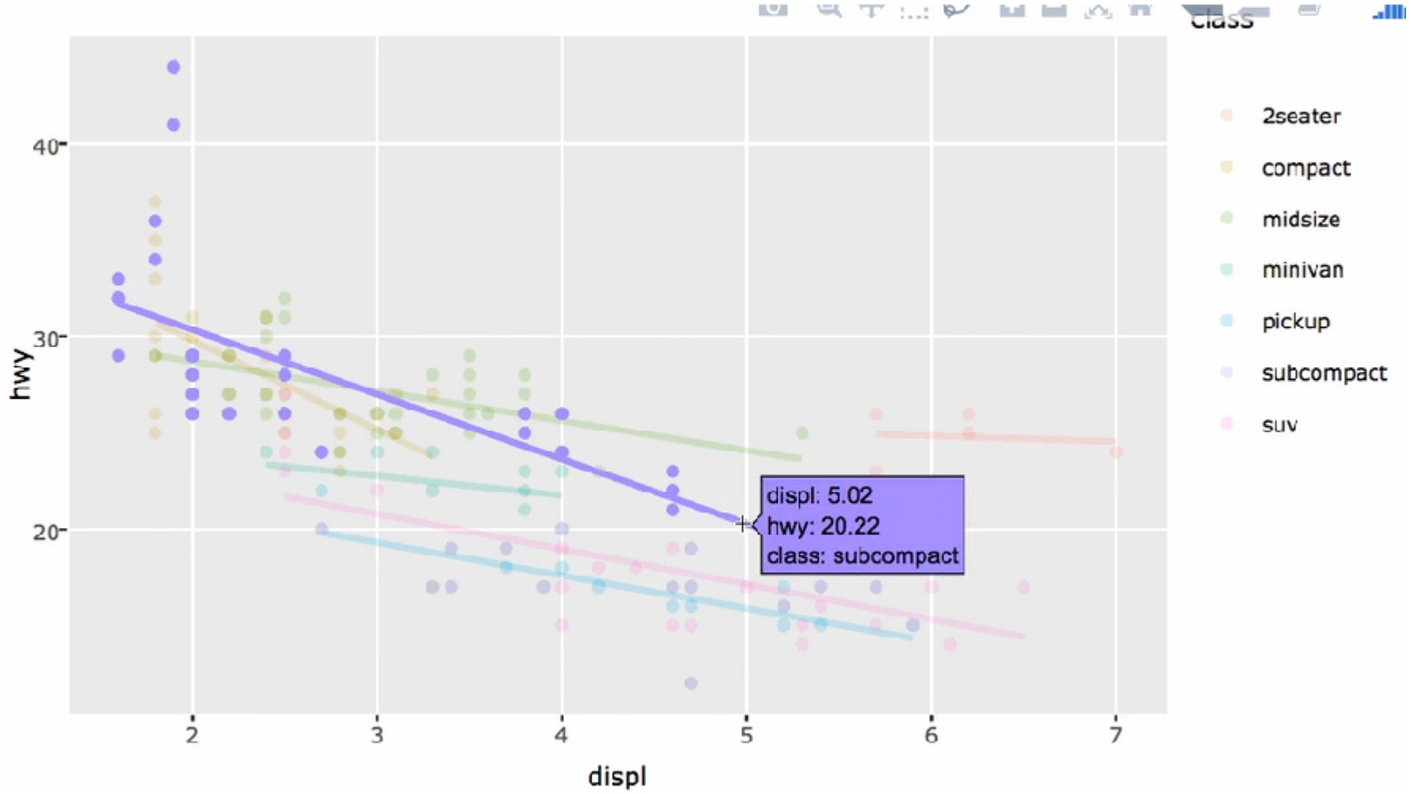
Using ggplot2

```
library(dplyr)
cities <- c("Galveston", "Midland", "Odessa", "South Padre Island")
txsmall <- txhousing %>%
  select(city, year, month, median) %>%
  filter(city %in% cities)

txsmall %>%
  highlight_key(~year) %>% {
    ggplot(., aes(month, median, group = year)) + geom_line() +
      facet_wrap(~city, ncol = 2)
  } %>%
  ggplotly(tooltip = "year")
```



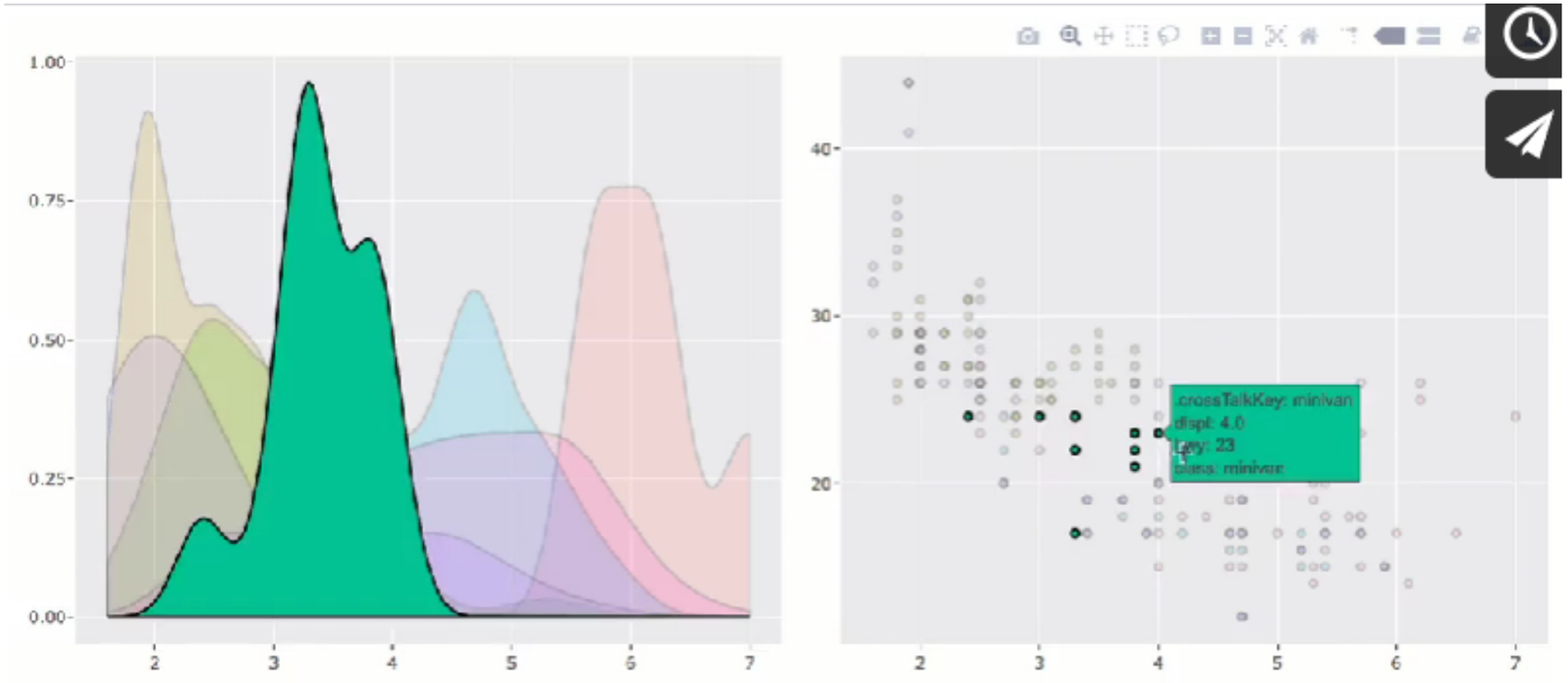
Using ggplot2

```
m <- highlight_key(mpg)
p <- ggplot(m, aes(displ, hwy, colour = class)) +
    geom_point() +
    geom_smooth(se = FALSE, method = "lm")
ggplotly(p) %>% highlight("plotly_hover")
```



Using ggplot2

```
m <- highlight_key(mpg, ~class)

p1 <- ggplot(m, aes(displ, fill = class)) + geom_density()

p2 <- ggplot(m, aes(displ, hwy, fill = class)) + geom_point()

subplot(p1, p2) %>% hide_legend() %>% highlight("plotly_hover")
```

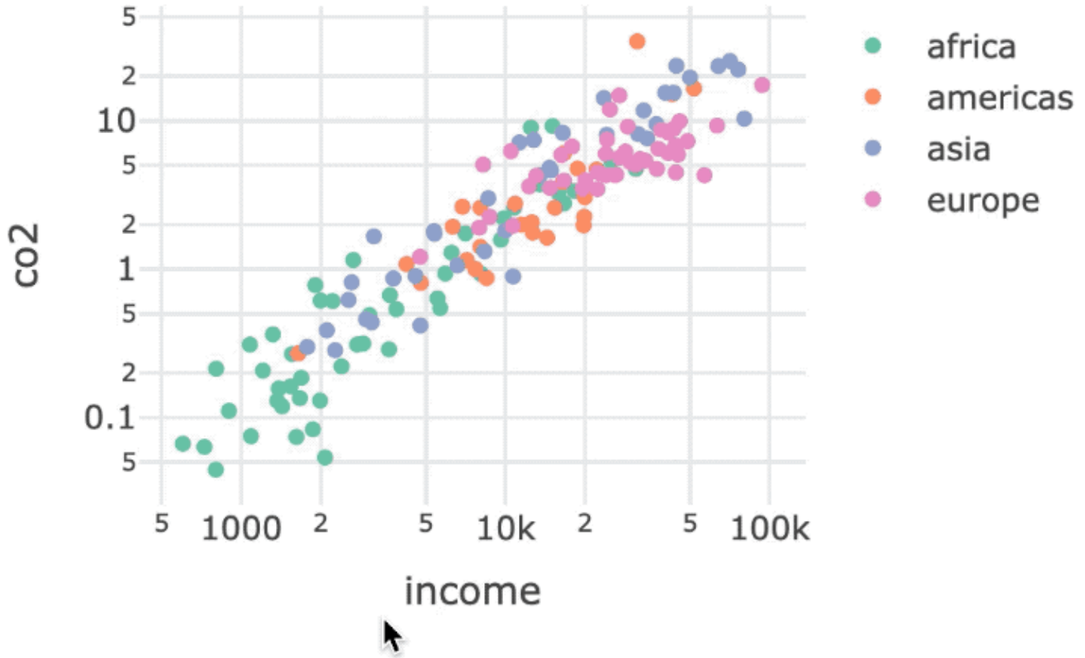


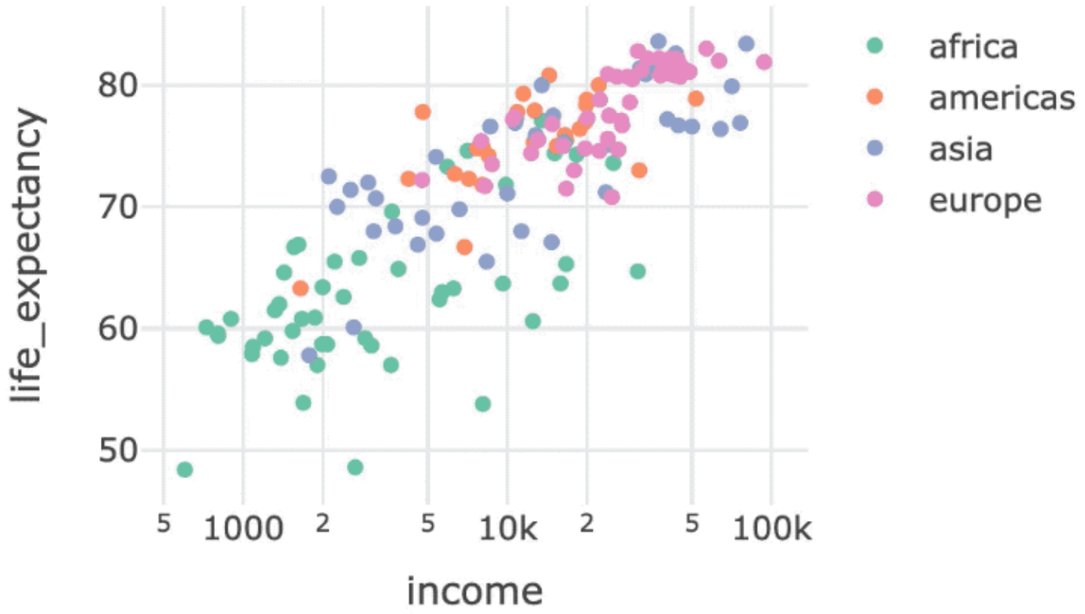
Creating filters

bscols() for column layouts

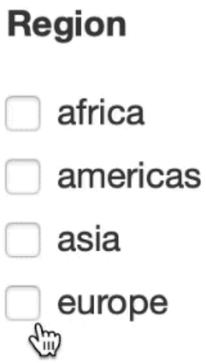
```
library(plotly)
library(crosstalk)
shared_data <- world2014 %>% SharedData$new()
p1 <- shared_data %>%
  plot_ly(x=~income, y = ~co2, color = ~four_regions) %>%
  add_markers() %>%
  layout(xaxis = list(type = "log"), yaxis = list(type = "log"))
p2 <- shared_data %>%
  plot_ly(x=~income, y = ~life_expectancy, color = ~four_regions) %>%
  add_markers() %>%
  layout(xaxis = list(type = "log")))
bscols(p1, p2)
```

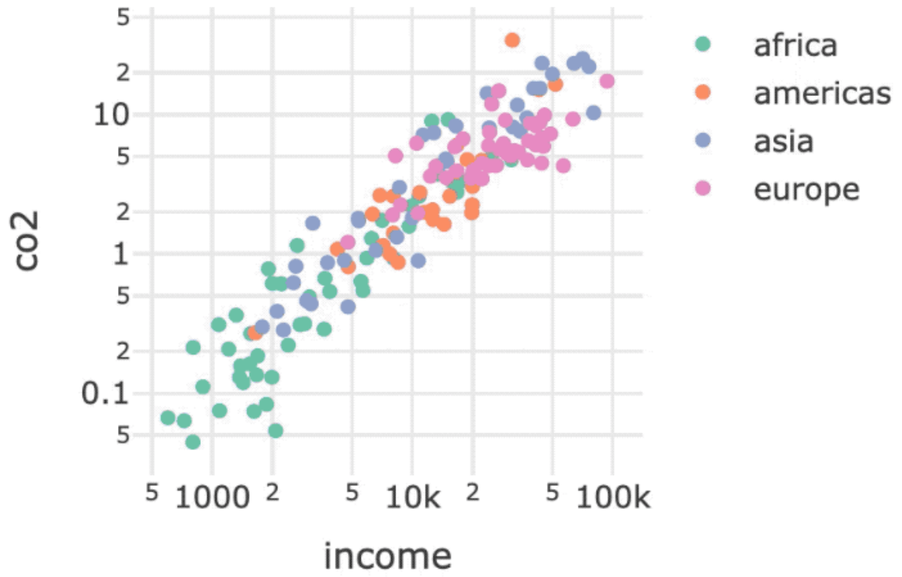
bscols() for column layouts





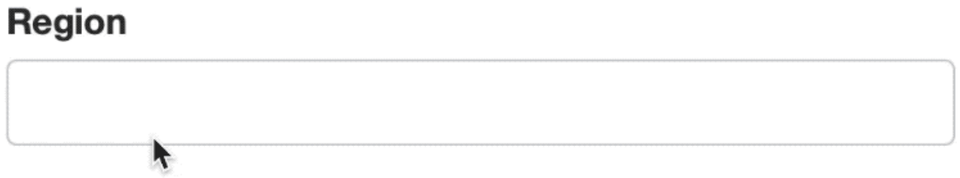
Adding filters: Checkboxes

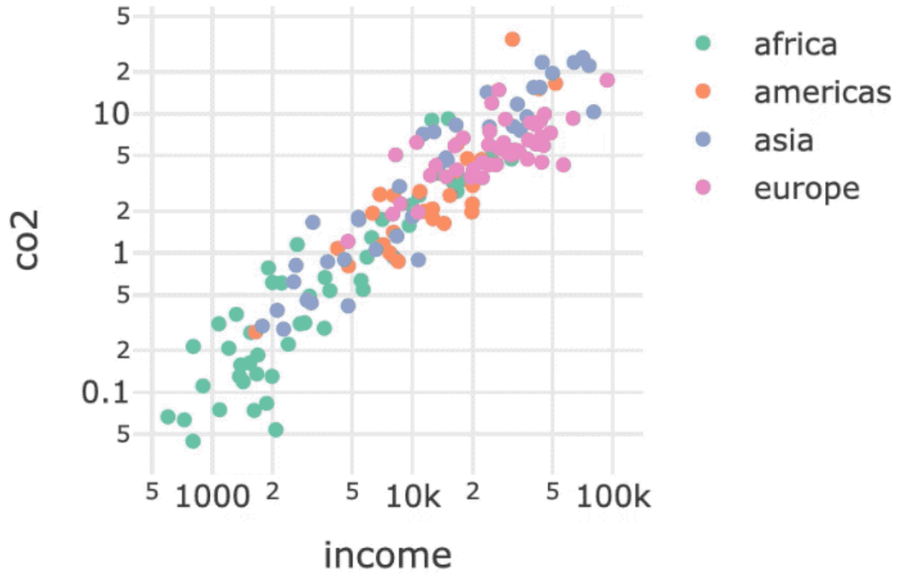





```
bscols(filter_checkbox(id = "four_regions", label = "Region",
                       sharedData = shared_data, group = ~four_regions),
       p1)
```

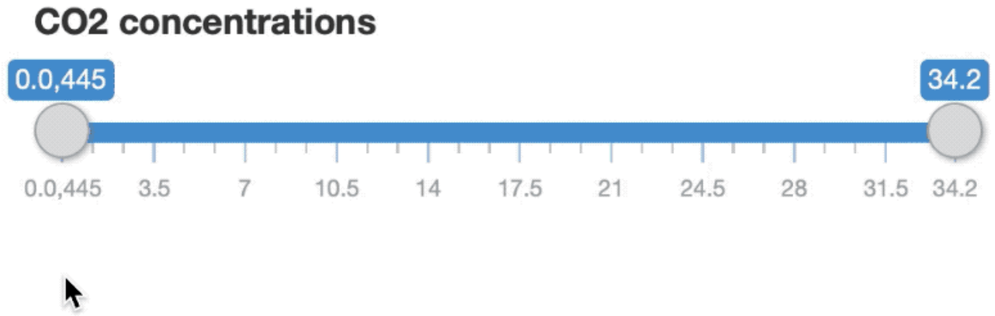
Adding filters: Select box

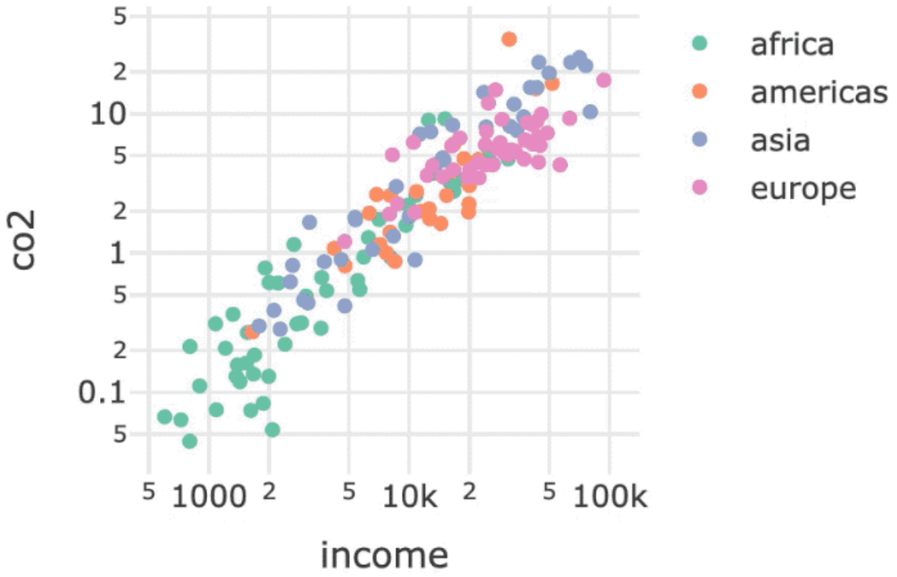





```
bscols(filter_select(id = "four_regions", label = "Region",
                     sharedData = shared_data, group = ~four_regions),
       p1)
```

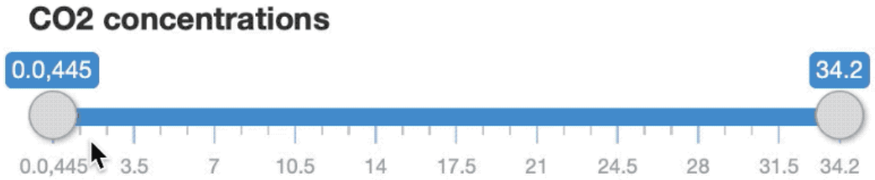
Adding filters: Sliders

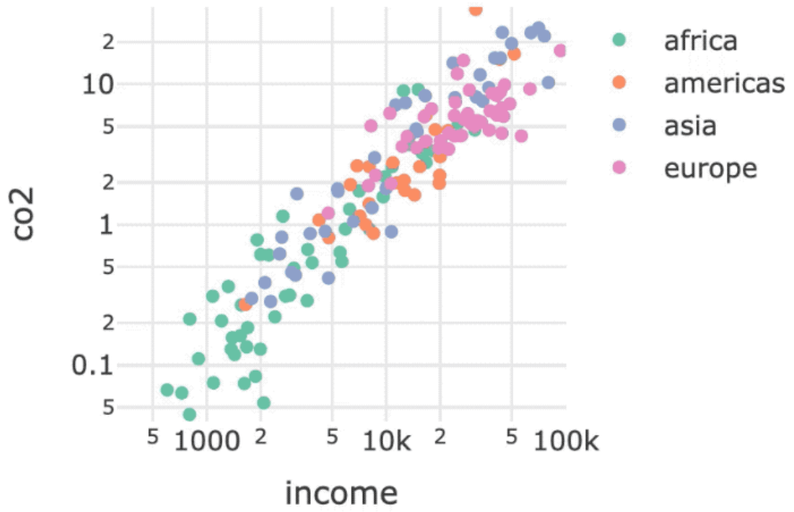





```
bscols(filter_slider(id = "co2", label = "CO2 concentrations",
                     sharedData = shared_data, column = ~co2),
       p1)
```

Fixing the range of your axes






```
bscols(filter_slider(id = "co2", label = "CO2 concentrations",
                     sharedData = shared_data, column = ~co2),
       p1 %>% layout(xaxis = list(range = c(2.5, 5)),
                     yaxis = list(range = c(-1.4, 1.55)))
)
```

Putting the pieces together

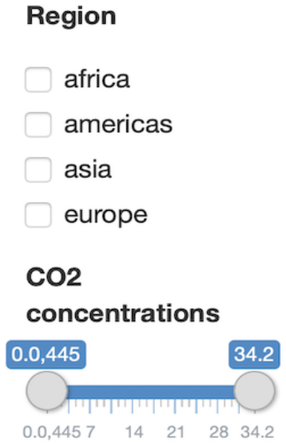
```
bscols(widths = c(2, 5, 5),
  list(
    filter_checkbox(
      id = "four_regions", label = "Region",
      sharedData = shared_data, group = ~four_regions ),
    filter_slider(
      id = "co2", label = "CO2 concentrations",
      sharedData = shared_data, column = ~co2 ) ),
   p1, p2 )
```

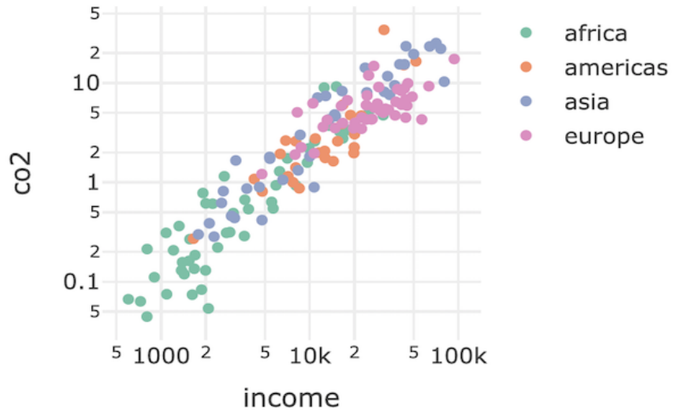
Great! `bscols()` makes it easy to combine plotly charts and other HTML widgets. To control the height of a `plotly` chart, add the `height` argument (in pixels) to the `plot_ly()` command.

Si agrupamos los plots en una **list** se crea una columna.

nrow: Define la cantidad de columnas necesarias.

widths = c(NA, 5, NA): Los NA dejan que los plots respectivos tengan el espacio sobrante.





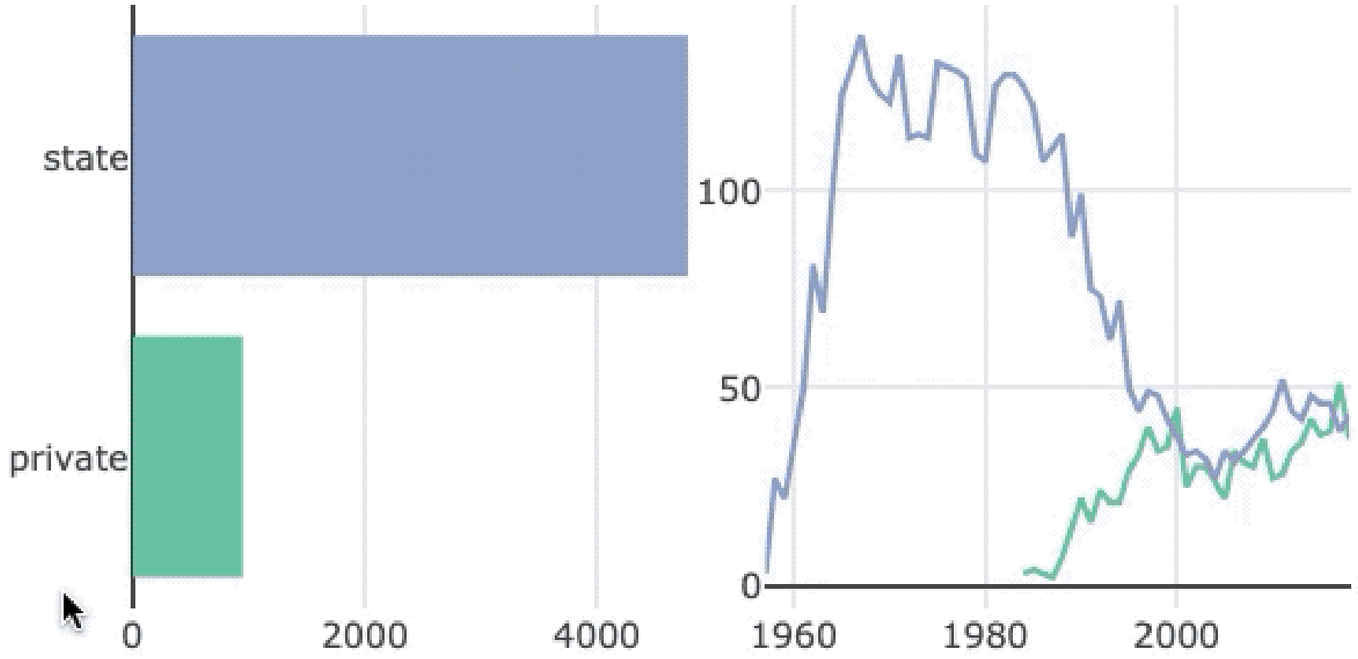
SharedData

```
shared_launches <- SharedData$new(launches, key = ~agency_type)
line_chart <- shared_launches %>%
  plot_ly(x = ~launch_year, y = ~n, color = ~agency_type) %>%
  count(launch_year, agency_type) %>%
  add_lines()  %>%
  hide_legend()
bar_chart <- shared_launches %>%
  plot_ly(y = ~fct_reorder(agency_type, n), x = ~n, color = ~agency_type) %>%
  count(agency_type) %>%
  add_bars() %>%
  layout(barmode = "overlay", yaxis = list(title = "")) %>%
  hide_legend()
```

barmode so that only a single bar for each year

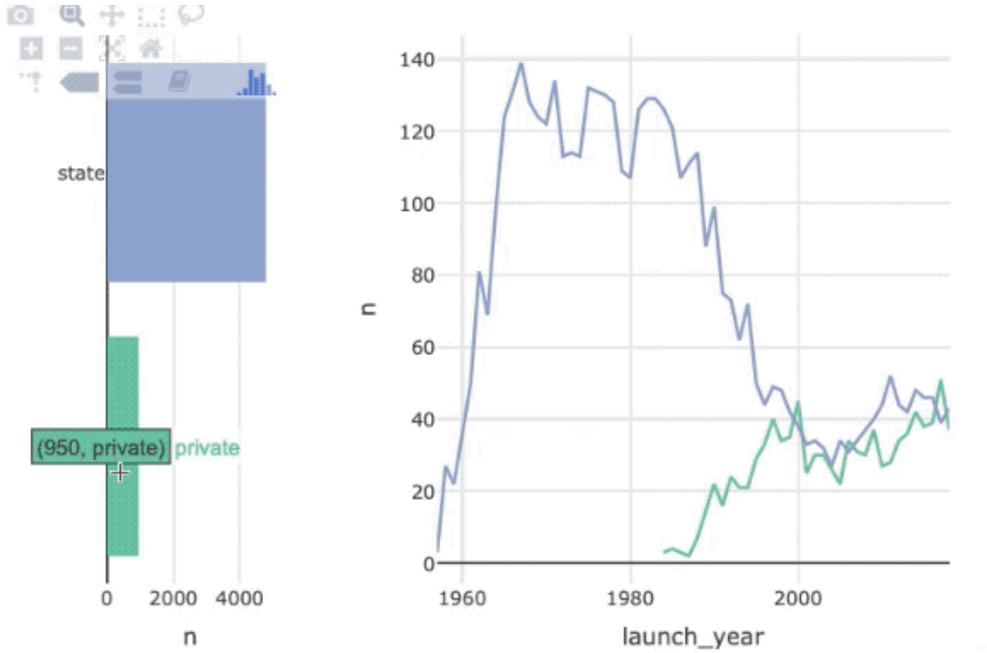
Linking views with subplot()

```
subplot(bar_chart, line_chart) %>%
  hide_legend() %>%
  highlight()
```



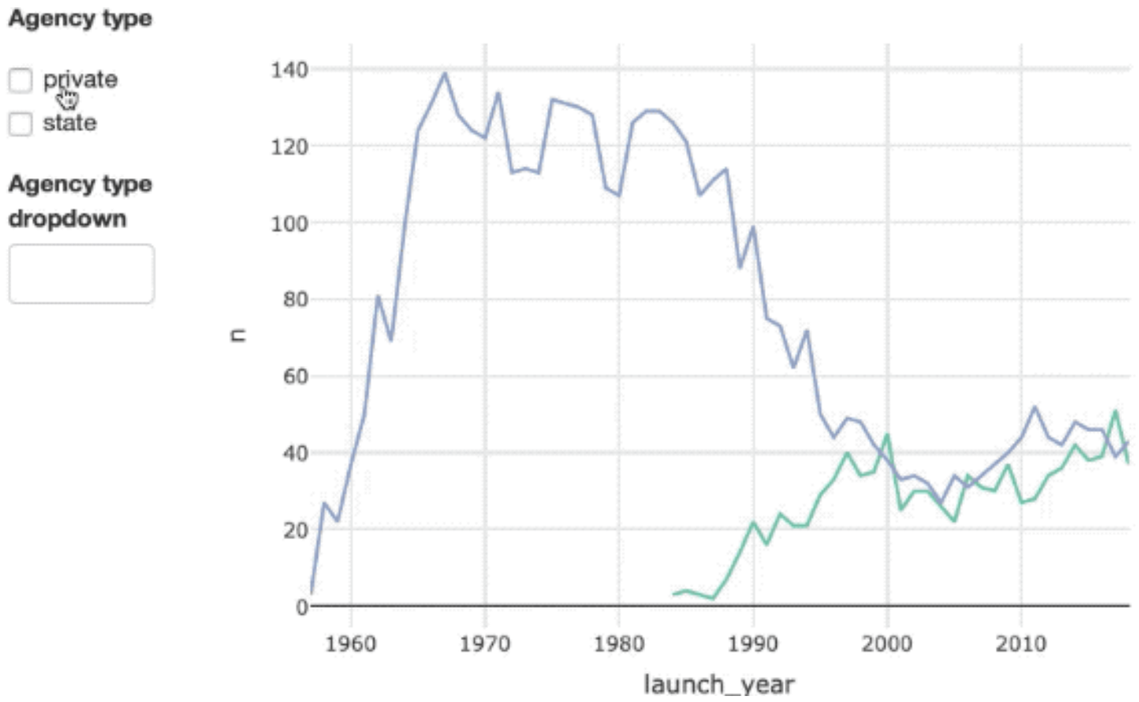
Linking views with bscols()

```
bscols(
  widths = c(4, NA),
  launch_state %>% highlight(),
  launch_ts %>% highlight()
)
```



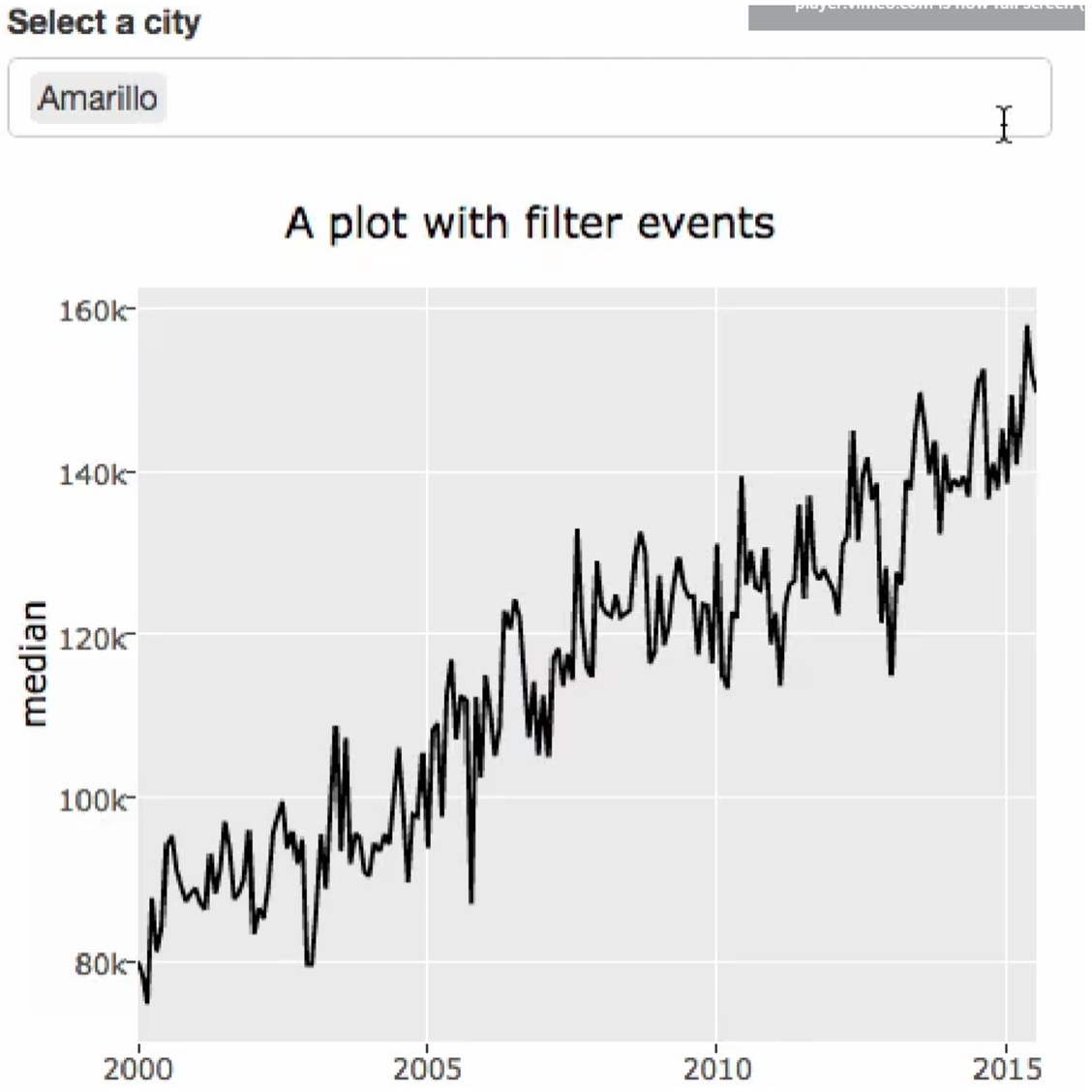
Selector widgets

```
bscols(widths = c(2, NA),
  list(filter_checkbox(id = "agency", label = "Agency type", shared_launches, ~agency_type),
       filter_select(id = "agency2", label = "Agency type dropdown", shared_launches, ~agency_type)),
  line_chart %>% highlight(on = "plotly_selected", off = "plotly_deselect")
)
```



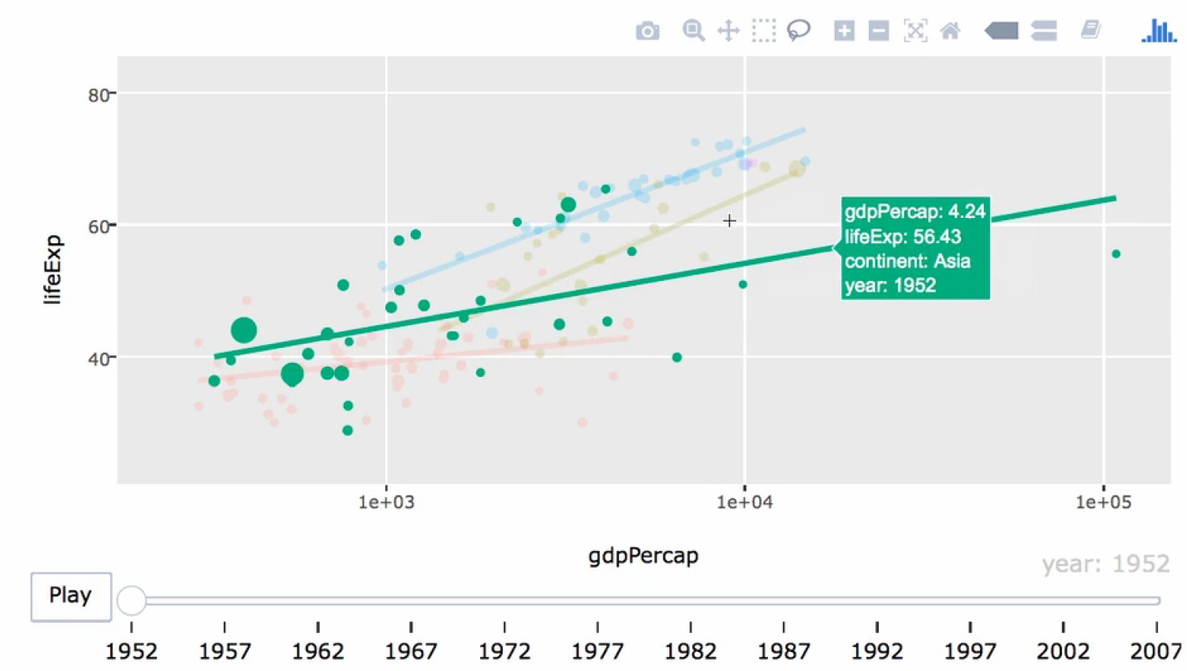
Using ggplot2

```
# generally speaking, use a "unique" key for filter,
# especially when you have multiple filters!
tx <- highlight_key(txhousing)
gg <- ggplot(tx) + geom_line(aes(date, median, group = city))
filter <- bscols(
  filter_select("id", "Select a city", tx, ~city),
  ggplotly(gg, dynamicTicks = TRUE),
  widths = c(12, 12)
)
```



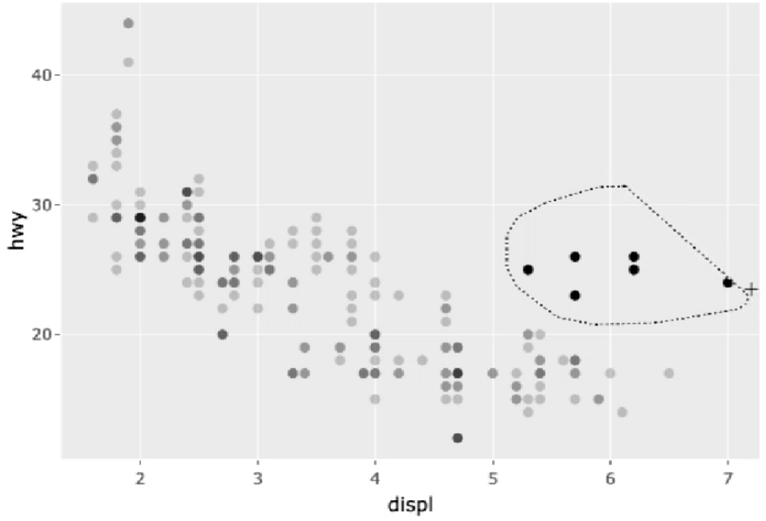
Linking animated views

```
g <- highlight_key(gapminder, ~continent)
gg <- ggplot(g, aes(gdpPercap, lifeExp,
  color = continent, frame = year)) +
  geom_point(aes(size = pop, ids = country)) +
  geom_smooth(se = FALSE, method = "lm") +
  scale_x_log10()
highlight(ggplotly(gg), "plotly_hover")
```



Linking a plot and table

```
library(plotly)

m <- highlight_key(mpg)

p <- ggplot(m, aes(displ, hwy)) + geom_point()

gg <- highlight(ggplotly(p), "plotly_selected")

crosstalk::bscols(gg, DT::datatable(m))
```



Show 10 entries

	manufacturer	model	displ	year	cyl	trans
1	audi	a4	1.8	1999	4	auto(l5)
2	audi	a4	1.8	1999	4	manual(m5)
3	audi	a4	2	2008	4	manual(m6)
4	audi	a4	2	2008	4	auto(av)
5	audi	a4	2.8	1999	6	auto(l5)
6	audi	a4	2.8	1999	6	manual(m5)
7	audi	a4	3.1	2008	6	auto(av)
8	audi	a4 quattro	1.8	1999	4	manual(m5)
9	audi	a4 quattro	1.8	1999	4	auto(l5)
10	audi	a4 quattro	2	2008	4	manual(m6)

Showing 1 to 10 of 234 entries Previous

Where to go from here

Explore more interactive plo ing libraries

- lea et
- highcharter
- trelliscope
- rbokeh

Learn Shiny

- plotly for R, by Carson Sievert
- DataCamp courses on Shiny