1

What is {plotly}?

`{plotly}` is an open source graphing library which runs via the open source JavaScript graphing library `{plotly.js}`. It allows the user to create high quality, interactive graphs. This includes scatter plots, histograms, heatmaps and many more! In this workshop we will cover:

- Interactive scatter plots and box plots.
- Adding custom controls such as buttons and sliders to plots.
- Interactive geographical plots.
- Integration of `{plotly}` with `{ggplot2}`

`{plotly}` is an extensive package and we will merely scratch the surface of its capabilities in this course. More information about the package can be found at the `{plotly}` website (https://plotly.com/r/)

1.1 Installation

For this course we recommend that you have the most recent version of `{plotly}` installed (version 4.9.3).

1.1.1 Download from CRAN

Use the `install.packages()` function to install the `{plotly}` R package from CRAN. This version may not be the latest version, so we recommend downloading from Github using the instructions below where possible.

```
install.packages("plotly")
```

1.1.2 Download from GitHub

Alternatively, you can install the latest development version of `{plotly}` from GitHub via the `{devtools}` R package:

```
devtools::install_github("ropensci/plotly")
```

2

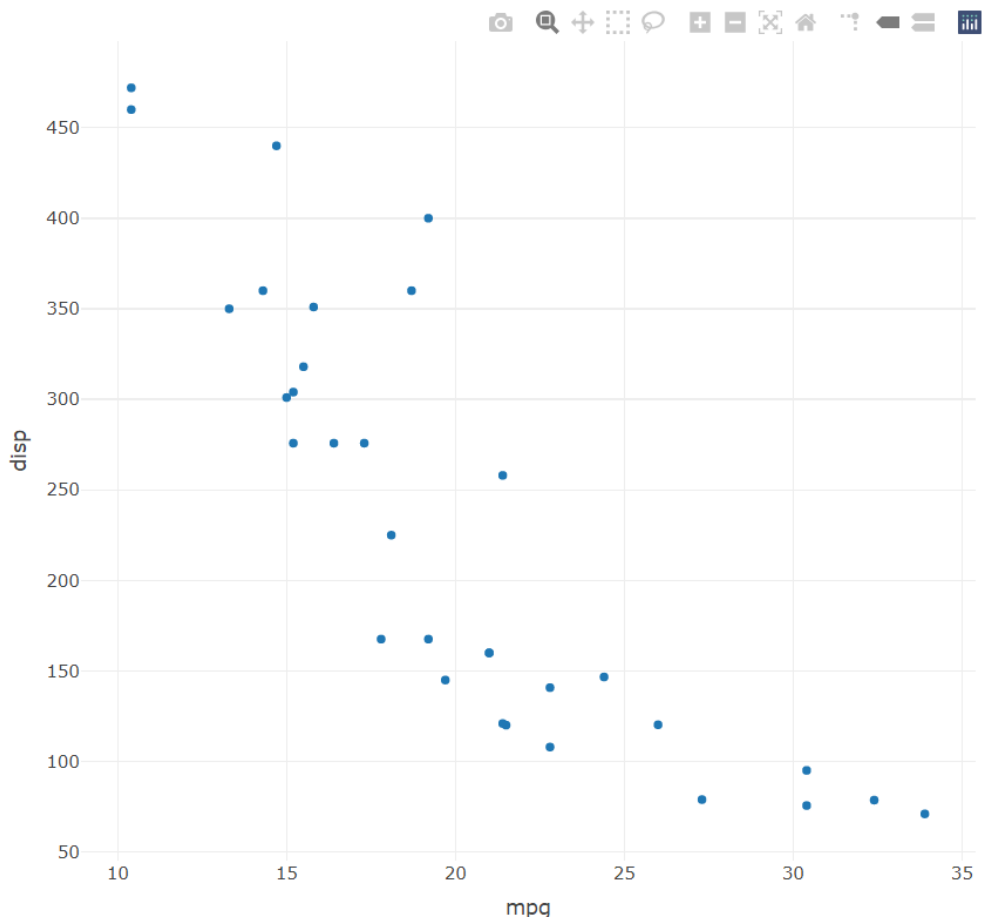
Creating an Interactive Plot

2.1 Interactivity

Let's see an example of a simple interactive plot:

```
library(plotly)
fig <- mtcars %>%
  plot_ly(x = ~mpg, y = ~disp)
fig
```

The interactive plot will appear in either a separate window within RStudio or under the Viewer tab, as opposed to the Plots tab.



The plot's interactivity can be accessed via its tool bar:

The functionality provided by this toolbar from left to right are as follows:

- Downloading the plot as a png file.
- Panning across the map.
- Selecting all points using a box.
- Selecting all points using a lasso.
- Zooming in and out on our plots.
- Resetting the view.

Moreover, hovering over individual points displays their coordinates.

2.2 Plot Structure

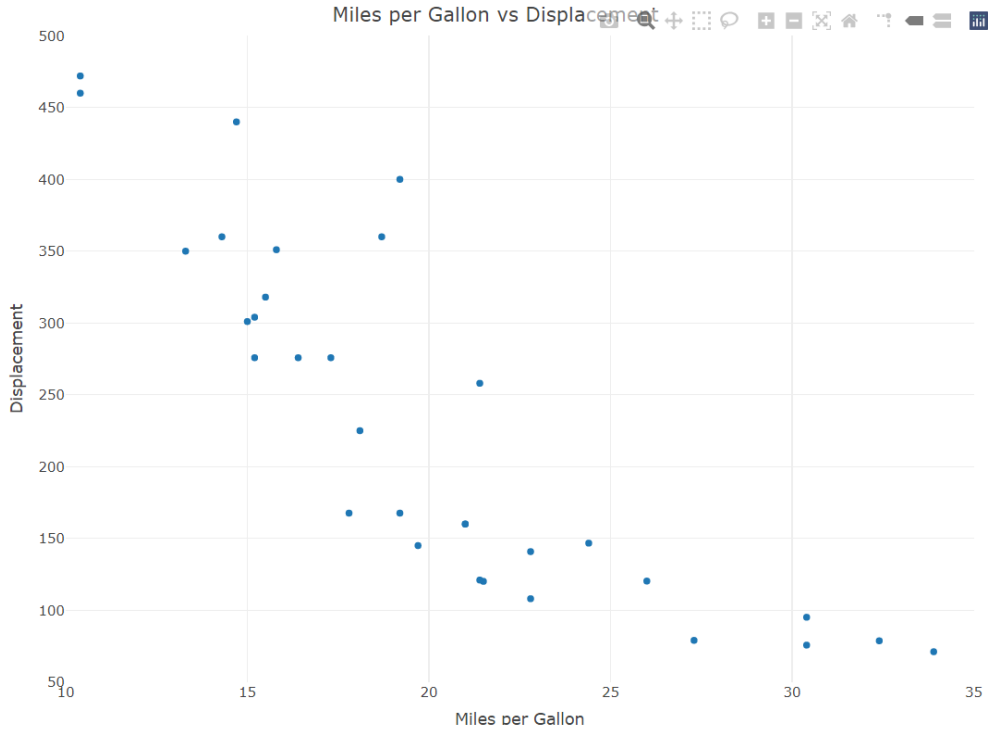
Charts in `{plotly}` can be described by three sets of functions: `plot_ly`, `layout`, and `add_trace` (`add_*`). `plot_ly` can be thought of as the base which allows R objects to be mapped to the Plotly library. `layout` is used to control the chart title, axis labels and scales. The `add_trace` function creates a geometry layer called a trace which is added to the chart. In a similar way to the structure of `{ggplot2}` plots, multiple traces can be added on one plot. There are many `add_*` functions. Running the following code provides a list of these functions:

```
stringr::str_subset(objects("package:plotly"), pattern ="^add_")
#  [1] "add_annotations"         "add_area"
#  [3] "add_bars"                "add_boxplot"
#  [5] "add_choropleth"          "add_contour"
#  [7] "add_data"                "add_fun"
#  [9] "add_heatmap"             "add_histogram"
# [11] "add_histogram2d"         "add_histogram2dcontour"
# [13] "add_image"               "add_lines"
# [15] "add_markers"             "add_mesh"
# [17] "add_paths"               "add_pie"
# [19] "add_polygons"            "add_ribbons"
# [21] "add_scattergeo"          "add_segments"
# [23] "add_sf"                  "add_surface"
# [25] "add_table"               "add_text"
# [27] "add_trace"
```

2.2.1 layout

We can use the `layout` function to set axis labels and ranges. To do this we use the `xaxis` and `yaxis` arguments which require lists:

```
fig <- mtcars %>%
  plot_ly(x = ~mpg, y = ~disp) %>%
  layout(
    title = "Miles per Gallon vs Displacement",
    xaxis = list(title = "Miles per Gallon",
                 range = c(10, 35)),
    yaxis = list(title = "Displacement",
                 range = c(50, 500))
  )
fig
```

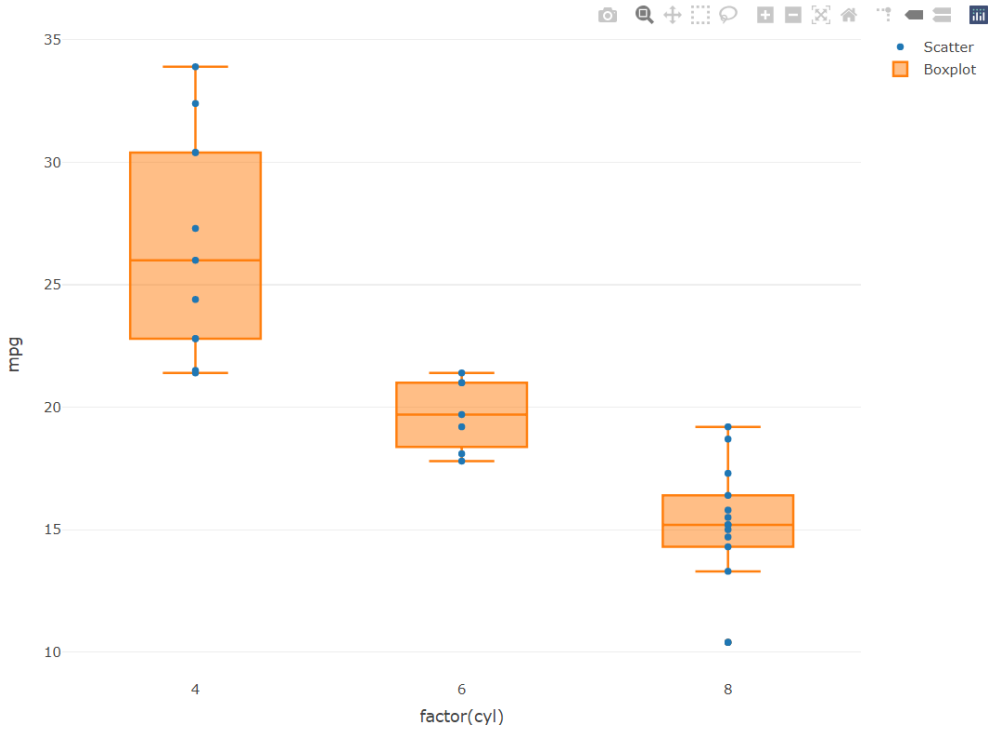


2.2.2 add_*

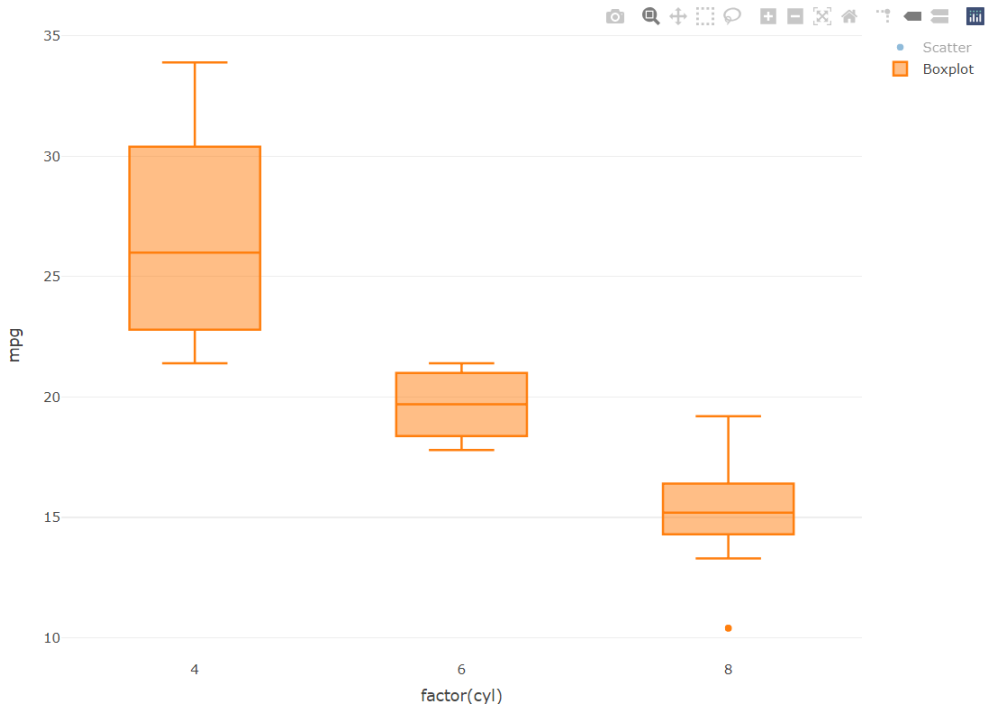
In the previous two examples we did not use any `add_*` functions, however it still created the plot. This is because `plot_ly` is smart and will automatically create a trace from the default and x,y arguments specified. We can use `plot_ly` with `add_*` functions to plot multiple traces (geometries). For example, we can use the function `add_boxplot` to overlay a box plot over a scatter plot described in `plot_ly`:

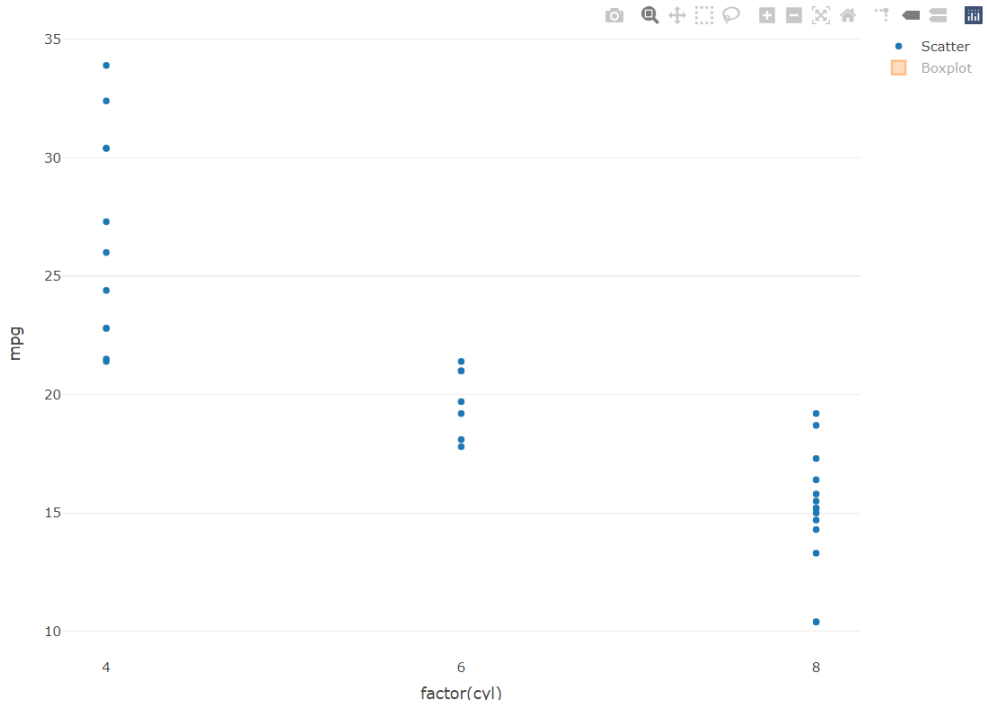
```
fig <- mtcars %>%
  plot_ly(x = ~factor(cyl), y = ~mpg, type = "scatter", name = "Scatter")
          %>%
  add_boxplot(name = "Boxplot")
fig
```

2.2 Plot Structure



Here we have a legend with the options “Scatter” and “Boxplot”. These were set using the `name` argument in `plot_ly` and `add_boxplot`. This legend is interactive: clicking on either option will remove/add that trace from/to the plot.





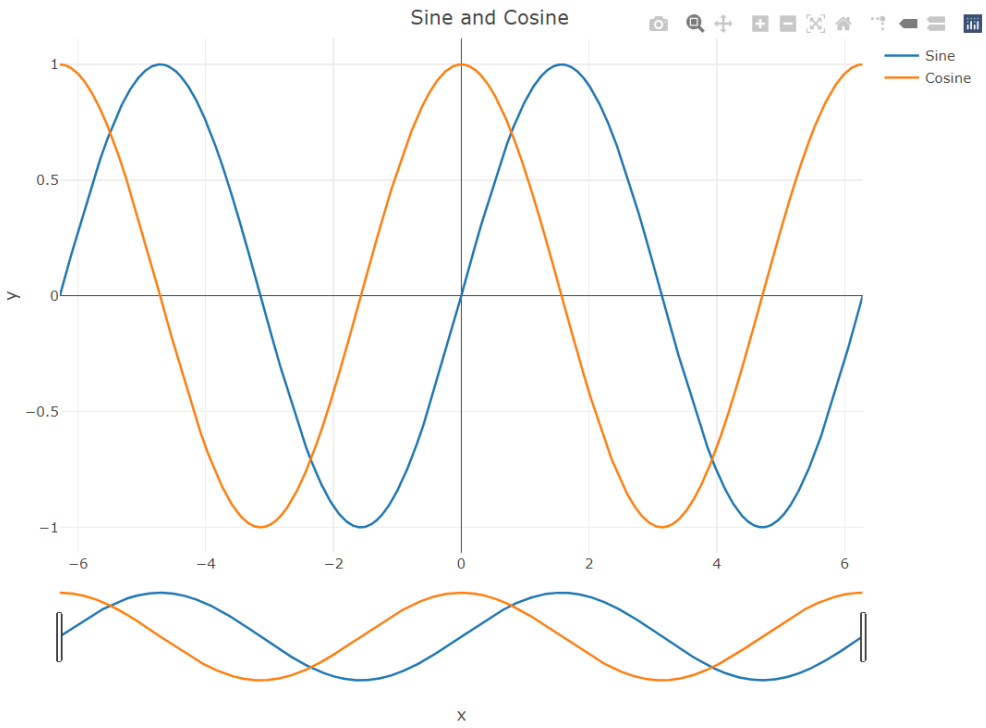
2.3 Custom Controls

There are a range of custom controls that can be added to an interactive plot such as buttons and sliders.

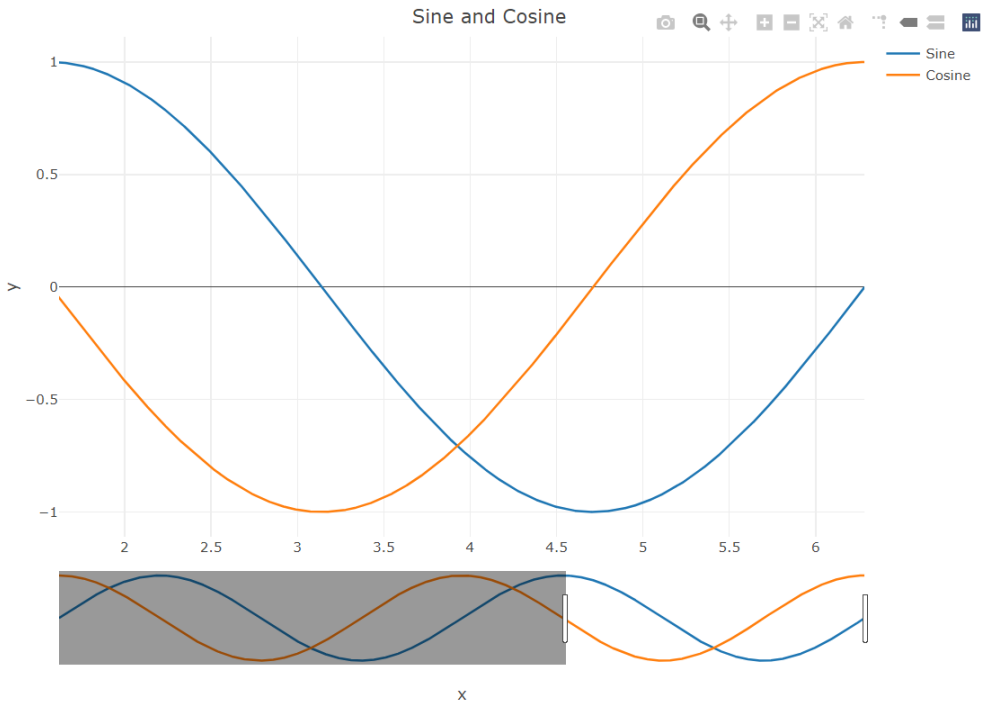
The following example produces an interactive plot of the sine and cosine functions using the `add_lines` and `layout` functions with a range slider below the x axis:

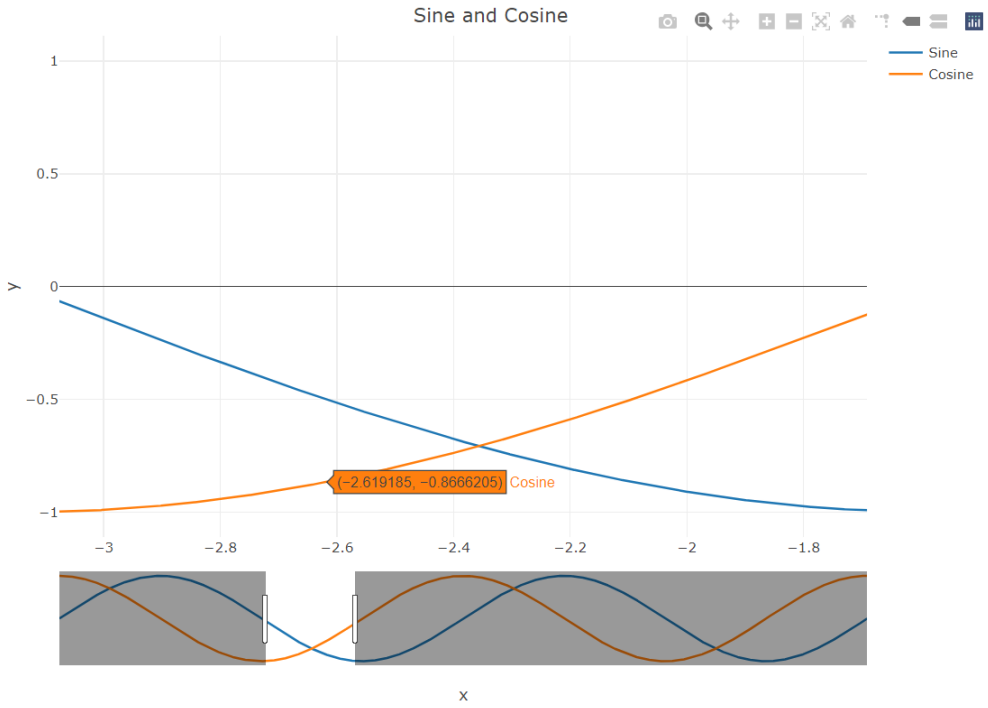
```
# Create data frame
x = seq(-2*pi, 2*pi, by = 1/1000)
df <- data.frame(x = x, y = sin(x),
                 z = cos(x))

# Interactive plot
fig <- df %>%
  plot_ly(x = ~x) %>%
  add_lines(y = ~y, name = "Sine") %>%
  add_lines(y = ~z, name = "Cosine") %>%
  layout(
    title = "Sine and Cosine",
    xaxis = list(rangeslider = list(type = "x"))
  )
fig
```

2.3 Custom Controls



This slider allows us to zoom into a specific range on the x axis.





Using the airquality data set, plot an interactive scatter plot of Temp against Ozone.

3

Scatter Plots on Maps

3.1 Basic Scatter on Map

Often when we are presented with geographical data it is easier to interpret if it is formatted as a map. This can be easily achieved using the `{plotly}` package.

First we need some geographical data to work with. Lets load in some data about the precipitation in the US in June 2015.

```
library(plotly)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
df <- read.csv(

          'https://raw.githubusercontent.com/plotly/datasets/master/2015_06_30_precipit
          ation.csv'

  )
```

If we view the data we can see that it includes `Lat` and `Long` columns. These are required for plotting geographical data.

```
head(df)
```

```
##       Hrapx    Hrapy     Lat       Lon Globvalue
## 1  272.3333 670.2500 48.4113 -112.8352    0.0875
## 2 1546.5000 195.1667 18.0057  -65.8040    0.0892
## 3  262.2500 674.2500 48.6163 -113.4784    0.0908
## 4  252.5000 674.6667 48.5379 -114.0702    0.0933
## 5  302.1667 671.4167 48.5843 -111.0188    0.0942
## 6  305.7500 670.0833 48.6120 -110.7939    0.0950
```

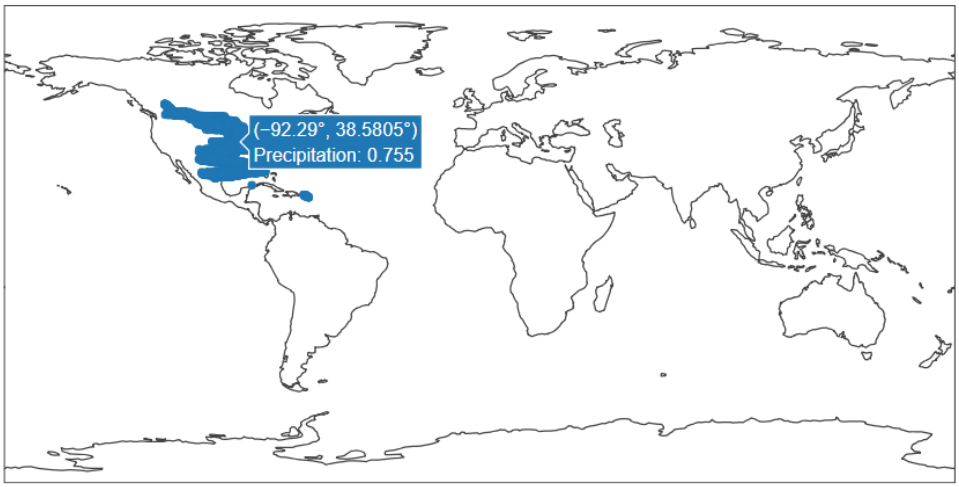
To create a basic plot all we need is the `plot_geo` function from the `{plotly}` package, where we pass in our dataframe and the names of the latitude and longitude columns. This creates a plot with interactive characteristics, which can be accessed using the toolbar. Additionally, we are able to see the labels associated with each point by hovering over them with the mouse. By default this is the latitude and longitudinal coordinates.

```
fig <- df %>%
  plot_geo(lat = ~Lat, lon = ~Lon)
fig
```

We can now customize the plot to make it more informative, for example, we can change the labels of each point using the `add_markers` function.

```
fig <- fig %>%
  add_markers(
    text = ~paste("Precipitation:", Globvalue)
    )
fig
```

Now, when we hover over a point, as well as seeing the geographical coordinates, we can see the name of the level of precipitation at each location.



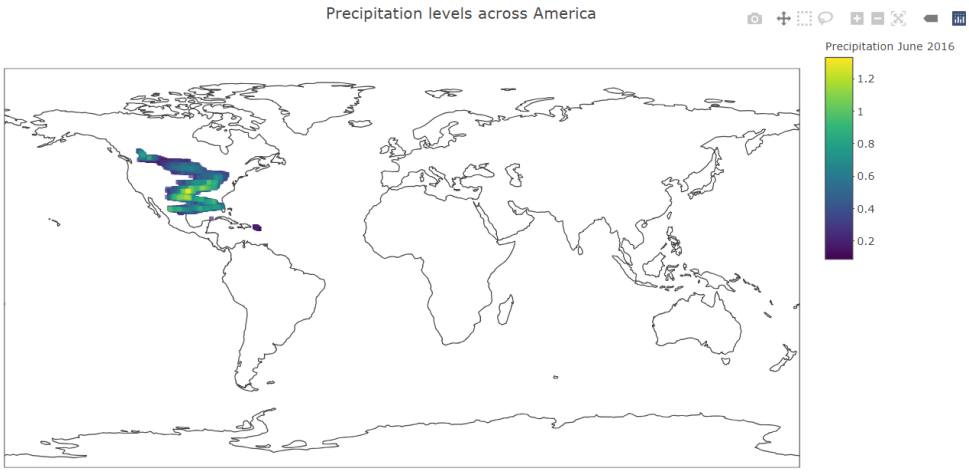
We can also customize the markers on the plot using this same function.

```
fig <- df %>%
  plot_geo(lat = ~Lat, lon = ~Lon) %>%
  add_markers(
    text = ~paste("Precipitation:", Globvalue),
    color = ~Globvalue,
    symbol = I("square"),
    size = I(8),
    hoverinfo = "text",
    opacity = I(0.8))

fig
```

Here, we have changed the shape of the markers to squares, decreased their size, changed the label to only what is specified in the text parameter and changed the colour based on the value in the `precipitation` column.

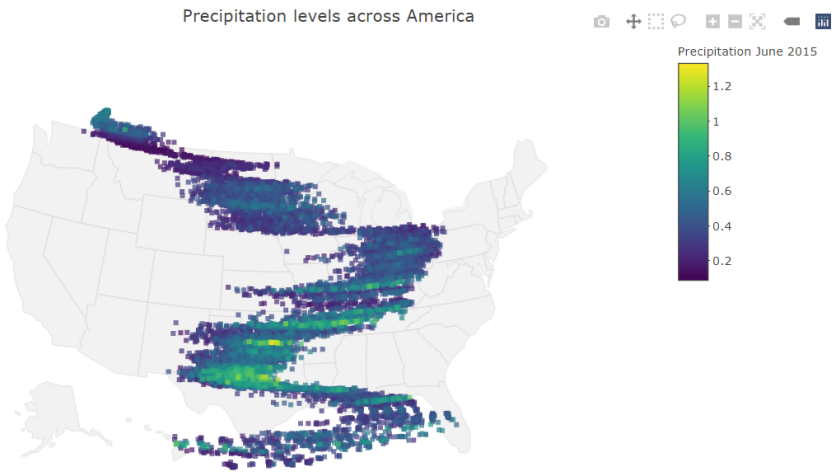
We can also add titles to our plot and our colour bar.

```
fig <- fig %>%
  colorbar(title = "Precipitation June 2015") %>%
  layout(
    title = 'Precipitation levels across America'
  )

fig
```



We can apply further customization to the layout of the visualization using the `geo` argument in the layout function. `geo` provides a list of arguments to the layout object, a full list can be found here: https://plotly.com/r/reference/#Layout_and_layout_style_objects.

```
# geo styling
g <- list(
  scope = 'usa',
  projection = list(type = 'albers usa'),
  showland = TRUE,
  landcolor = toRGB("gray95"),
  subunitcolor = toRGB("gray85"),
  countrycolor = toRGB("gray85"),
  countrywidth = 0.5,
  subunitwidth = 0.5
)

fig <- fig %>%
  layout(
    geo = g
  )

fig
```



The arguments in the `geo` list have the following functionality:

Parameter	Description
`scope`	Sets the scope of the map. In this case it focuses it on the USA.
`projection`	Sets the projection.
`showland`	Sets whether or not land masses are filled in color.
`landcolor`	Sets the landmass color.
`subunitcolor`	Sets the color of the subunits boundaries.
`subunitwidth`	Sets the stroke width (in px) of the subunits boundaries.
`countrycolor`	Sets line color of the country boundaries.
`countrywidth`	Sets line width (in px) of the country boundaries.

4

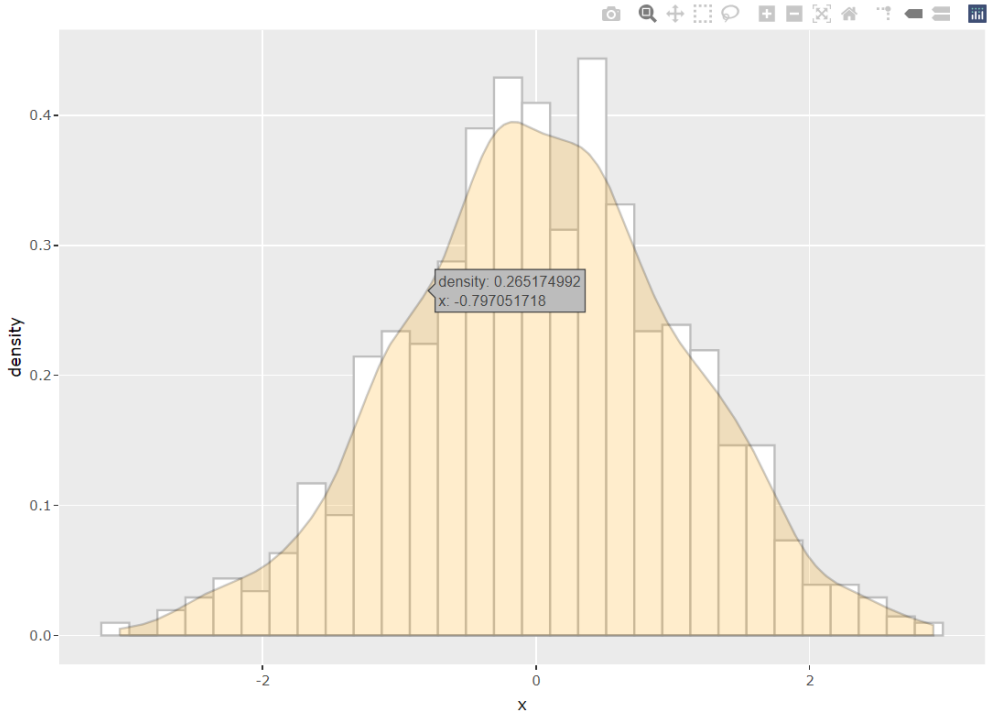
Integration with {ggplot2}

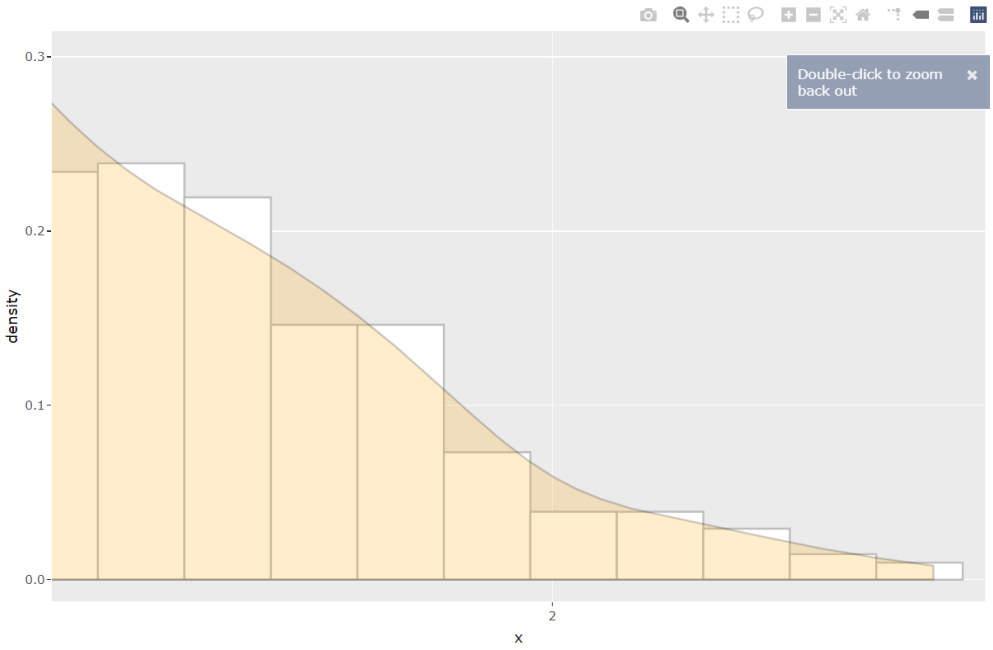
`{plotly}` allows for each integration with `{ggplot2}`. A ggplot object can be transformed into an interactive plot by calling the function `ggplotly`. Here are a couple of examples:

4.1 Example 1

The following takes a histogram created using `{ggplot2}` and transforms it into an interactive plot. Hovering over the bars and overlay we can see their coordinates and we can move the axis, zoom in and select parts of the plot using the toolbar as discussed previously.

```
library(tidyverse)
set.seed(23)
df <- data.frame(x = rnorm(1000))

# ggplot object
p <- df %>%
  ggplot(aes(x = x)) +
  geom_histogram(aes(y = ..density..), colour = "gray", fill = "white") +
  geom_density(fill = "orange", alpha = 0.2)

# Create interactive plot
ggplotly(p)
```

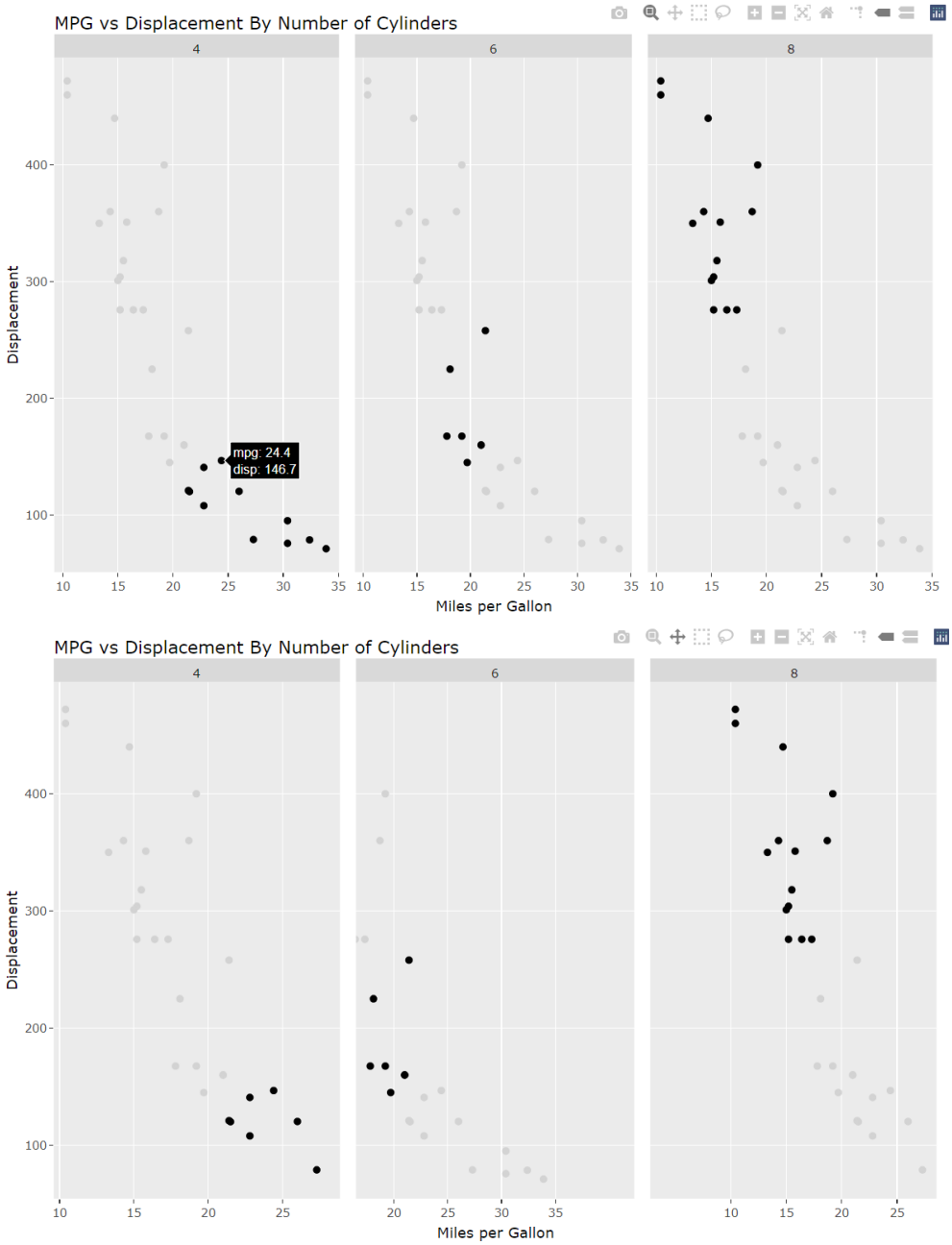




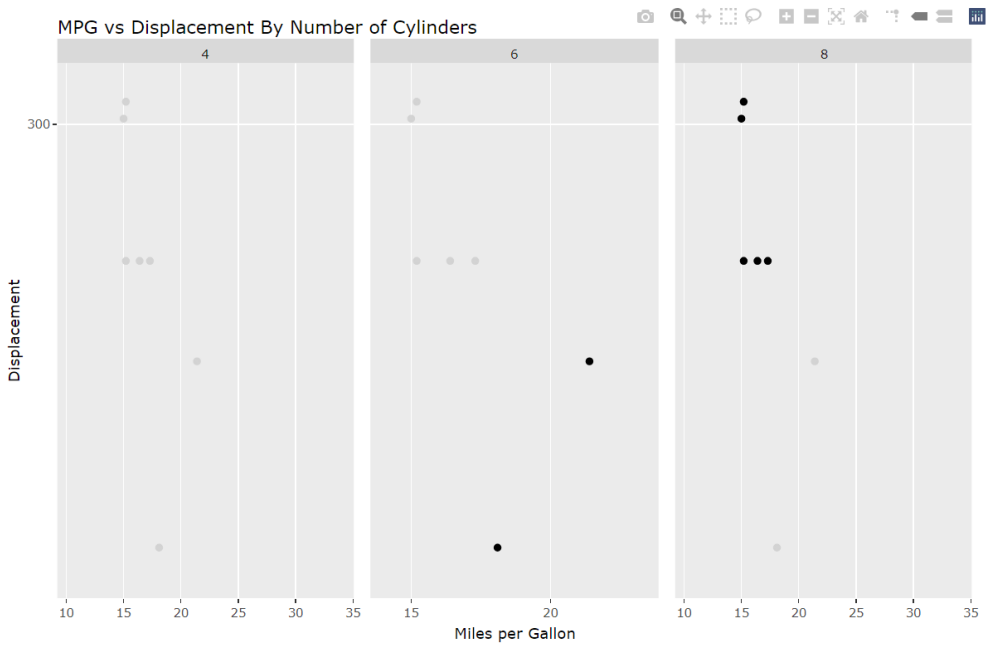
4.2 Example 2

The following takes a faceted plot created using `{ggplot2}` and transforms it into an interactive plot. Hovering over the points we can see their coordinates. We can move the axis: moving the x axis of one facet does not change the others; moving the y axis of one facet moves the y axis for all three. We can also zoom in to parts of a facet, which in turn zooms the other facets, and select parts of the plots.

```
car_copy <- select(mtcars, -cyl)

# ggplot object
p <- ggplot() +
  geom_point(data = car_copy, aes(x = mpg, y = disp),
             color = "lightgrey") +
  geom_point(data = mtcars, aes(x = mpg, y = disp)) +
  facet_grid( ~ cyl) +
  ggtitle("MPG vs Displacement By Number of Cylinders") +
  xlab("Miles per Gallon") +
  ylab("Displacement")

# Create interactive plot
ggplotly(p)
```


MPG vs Displacement By Number of Cylinders
4
6
8
Displacement
400
300
200
100
mpg: 24.4
disp: 146.7
10
15
20
25
30
35
Miles per Gallon
MPG vs Displacement By Number of Cylinders
4
6
8
Displacement
400
300
200
100
10
15
20
25
30
35
Miles per Gallon

4.2 Example 2


MPG vs Displacement By Number of Cylinders
4
6
8
300
Displacement
10
15
20
25
30
35
15
20
10
15
20
25
30
35
Miles per Gallon