

# More Data Visualization: A Brief Introduction to Plotly

Over the span of this course, we have mainly covered two packages for the purpose of data visualization: *ggplot* and *ggvis*. Although both packages are extremely useful for the purpose of data visualization, each comes with its own pros and cons. While *ggplot* is a nice, easy-to-use package for basic data visualization, its lack of interactivity can be somewhat limiting for both developers and users. Conversely, *ggvis* is a package that allows for much more complex visualizations since it implements interactive charts and graphs although, in my opinion, at the expense of “user-friendliness.” The purpose of this post is to introduce a somewhat middle ground in the form of *plotly*, an R package that creates interactive data visualizations in a syntax that is relatively similar to that of *ggplot*.

## Background:

Created in 2012, Plotly is an open source visualization library specializing in interactive graphing and headquartered in Montreal, Canada.<sup>1</sup> As a result of its construction using the Python scripting language, the package is very much language agnostic, meaning that instead of being saved as a traditional .png file, all charts are stored as JSON files which can be read by a wide variety of languages such as JavaScript and R. Because of this nontraditional storage form, all graphs and charts created using *plotly* are embeddable and editable on the web, so unlike *ggvis*, interactive graphs can be run on knit html files.<sup>2</sup> Below are a few examples to familiarize yourself with *plotly* syntax and compare its usability with that of *ggplot* and *ggvis*.

```
#Packages needed for Examples.  
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
```







```
library(ggplot2)  
library(ggvis)
```

```
##  
## Attaching package: 'ggvis'
```

```
## The following objects are masked from 'package:plotly':  
##  
## add_data, hide_legend
```

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

## Basic Syntax:

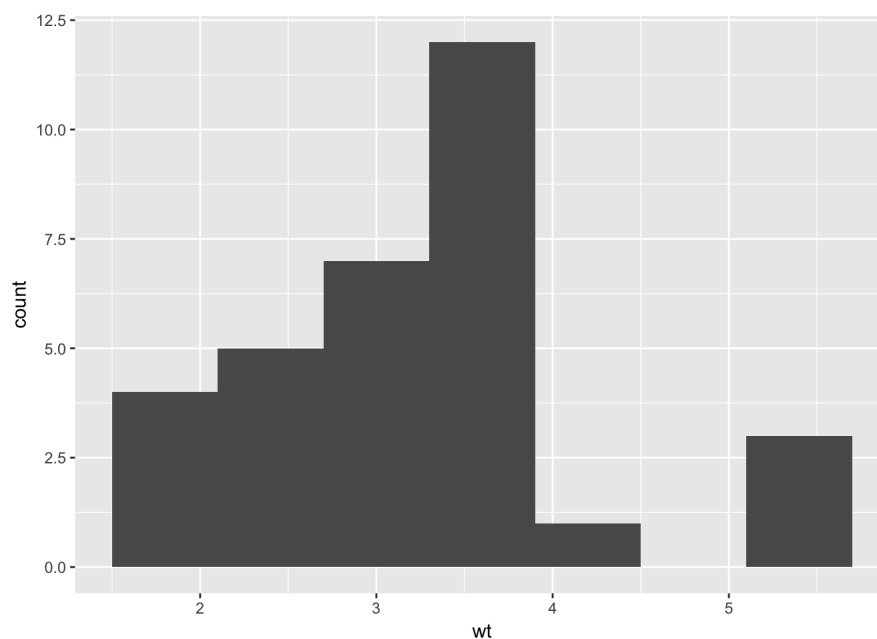
BASIC CHARTS	
 Line Plots	 Bubble Charts
<pre>plot_ly (   x = c( 1, 2, 3 ),   y = c( 5, 6, 7 ),   type = 'scatter',   mode = 'lines' )</pre>	<pre>plot_ly (   x = c( 1, 2, 3 ),   y = c( 5, 6, 7 ),   type = 'scatter',   mode = 'markers',   size = c( 1, 5, 10 ),   marker = list(     color = c( 'red', 'blue',     'green' )))</pre>
 Scatter Plots	 Heatmaps
<pre>plot_ly (   x = c( 1, 2, 3 ),   y = c( 5, 6, 7 ),   type = 'scatter',   mode = 'markers' )</pre>	<pre>plot_ly (   z = volcano ,   type = 'heatmap' )</pre>
 Bar Charts	 Area Plots
<pre>plot_ly (   x = c( 1, 2, 3 ),   y = c( 5, 6, 7 ),   type = 'bar',   mode = 'markers' )</pre>	<pre>plot_ly (   x = c( 1, 2, 3 ),   y = c( 5, 6, 7 ),   type = 'scatter',   mode = 'lines',   fill = 'tozeroy' )</pre>

Plotly constructs its graphs using two main functions: `plot_ly` and `ggplotly`. The first, `plot_ly`, is a visualization function like we've seen before that takes in a data frame, x and/or y coordinate mappings, type of visualization, and various other optional arguments to customize your graph.<sup>3</sup>

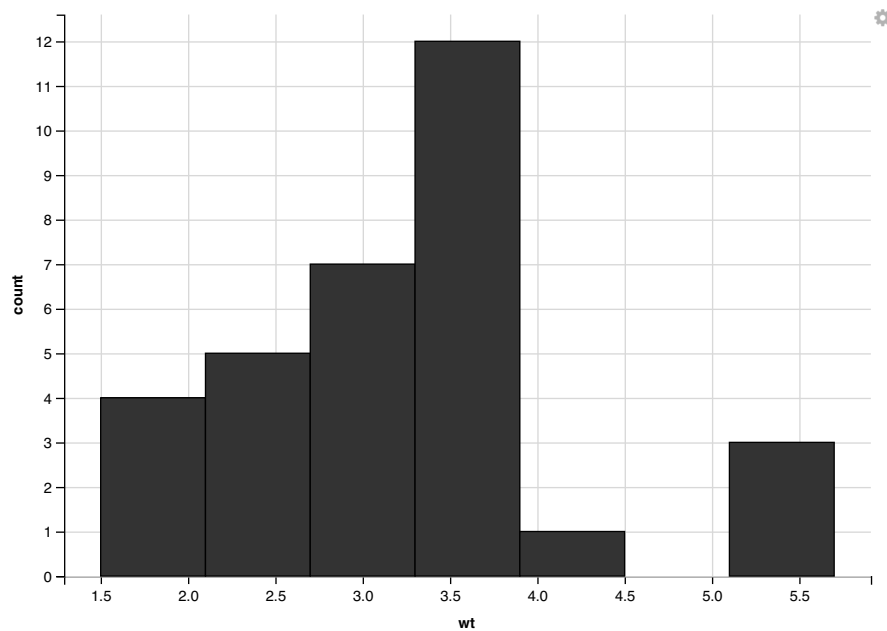
## Example 1) Distribution of Weights

**Goal:** Create a graph that shows distribution of car weights using the built-in `mtcars` data set's `wt` column.

```
#Graph using 'ggplot2'.
ggplot(data = mtcars,aes(x = wt))+geom_histogram(binwidth = .6)
```



```
#Graph using 'ggvis'.
ggvis(data=mtcars,x=-wt)%>%
layer_histograms(width = .6)
```



```
#Graphs using 'plot_ly'.
plot_ly(data=mtcars,x=-wt,type = "histogram",xbins=list(start=1.5,end=6,size=.6))
```

As you can see, `plot_ly` eliminates the need for secondary functions like `geom_histogram` or `layer_histograms` by declaring its type and format within the function itself. Also, as in the example above, all variable attributes, such as those declared in `xbins`, must be declared in the form of a list.<sup>4</sup> Additionally, functions like `layout` can be used with `plot_ly` for greater graphic control.

```
plot_ly(data=mtcars,x=-wt,type = "histogram",xbins=list(start=1.5,end=6,size=.6))%>%
  layout(xaxis=list(title = 'Car Weights',showgrid=TRUE,nticks=10),
        yaxis=list(title = 'Frequency', showgrid=TRUE,nticks=24))
```

Notice that all graph accurately depict the given data in accordance with the desire binwidth of .6. However, as you can see if you alter the binwidth of the given *plotly* graph, failure to alter the starting and endpoints as well will result in a slightly different graph. In order to circumvent this problem, *plotly* contains a second visualization function **ggplotly** which takes in any *ggplot* chart as an argument and displays it in an interactive format.

```
#Assign ggplot graph to 'ggplotly'.
plot <- ggplot(data = mtcars,aes(x = wt))+geom_histogram(binwidth = .6)
#Call ggplotly on plot object.
ggplotly(plot)
```

```
## We recommend that you use the dev version of ggplot2 with `ggplotly()`
## Install it with: `devtools::install_github('hadley/ggplot2')`
```

We can then manipulate the newly created *plotly* object just as before<sup>5</sup>:

```
ggplotly(plot)%>%
  layout(xaxis=list(title = 'Car Weights',showgrid=TRUE),
         yaxis=list(title = 'Frequency', showgrid=TRUE))
```

```
## We recommend that you use the dev version of ggplot2 with `ggplotly()`
## Install it with: `devtools::install_github('hadley/ggplot2')`
```

For the most part, *plotly* is a very simple yet powerful package. Now, for a more complex example:

## Example 2) Passengers per Month by Year

**Goal: Show visualization of number of passengers per month for every year using built-in AirPassengers data.**

For this example, we will be using a scatterplot to show number of passengers per month, setting the months as the x-axis and number of people as the y-axis.

First, since AirPassengers is not in the data frame form that we want, we need to clean the data. Create a new data frame passengers with the first column being the vector of months and each subsequent column the vector of passengers by month for every year.

```
#For appearances abbreviate month names.
month.name <- substr(month.name,1,3)

passengers=data.frame(Month=month.name)
n=1
for (year in 1949:1960){
  passengers[,as.character(year)]=AirPassengers[n:(n+11)]
  n=n+12
}

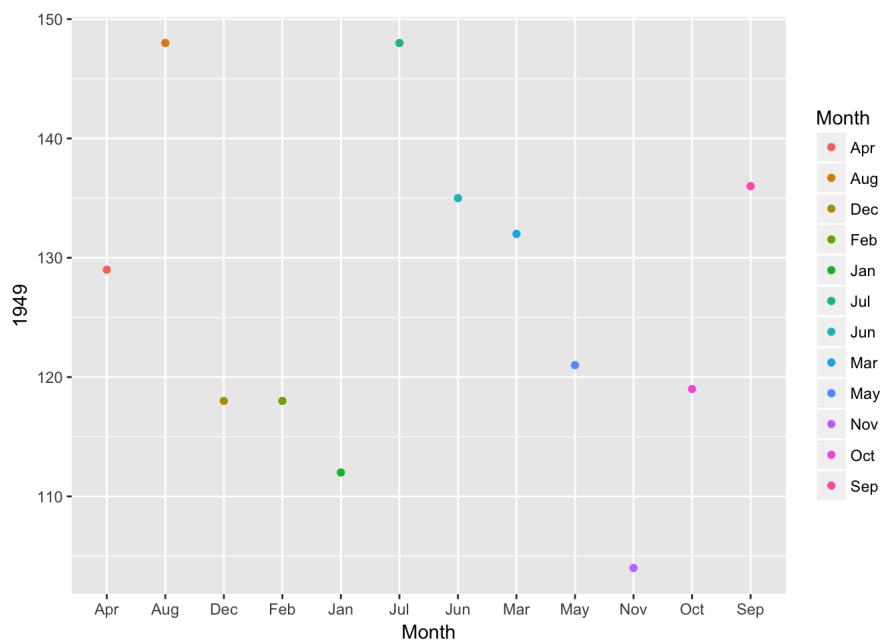
head(passengers)
```

```
##   Month 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960
## 1  Jan   112   115   145   171   196   204   242   284   315   340   360   417
## 2  Feb   118   126   150   180   196   188   233   277   301   318   342   391
## 3  Mar   132   141   178   193   236   235   267   317   356   362   406   419
## 4  Apr   129   135   163   181   235   227   269   313   348   348   396   461
## 5  May   121   125   172   183   229   234   270   318   355   363   420   472
## 6  Jun   135   149   178   218   243   264   315   374   422   435   472   535
```

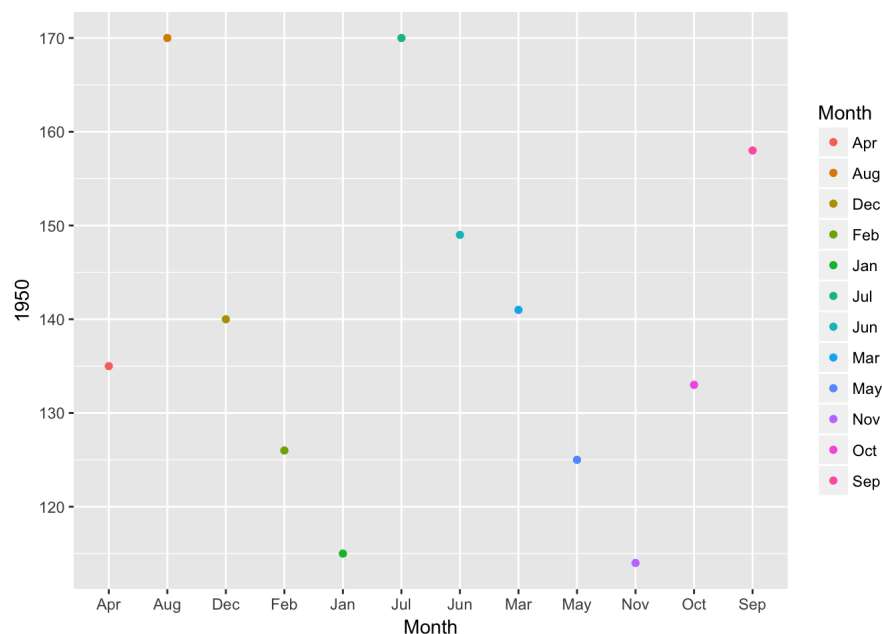
## 1) GGPlot

The first method we will consider involves the use of *ggplot*.

```
ggplot(passengers,aes(Month,`1949`,color=Month))+geom_point()
```



```
ggplot(passengers,aes(Month,`1950`,color=Month))+geom_point()
```



Since this package is not interactive, the only possible way to visualize the number of passengers by month for each year is to create a new chart for each subsequent year, which is not very efficient.

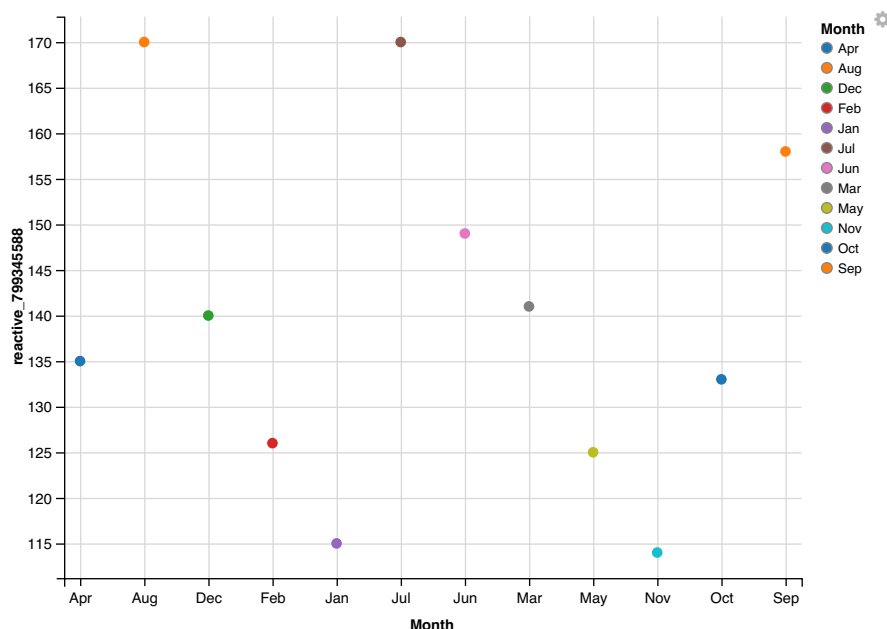
## 2)GGvis

The second method we will consider is using the *ggvis* library to create an interactive scatterplot that changes its points according to the chosen year.

```
#slider that returns column name of passengers according to desired year.
slider <- input_slider(1949,1960,1950, map=function(x) {as.name(names(passengers[x-1947]))})

ggvis(passengers,x = ~Month,
      y=slider) %>%
  layer_points(fill=~Month)
```

```
## Warning: Can't output dynamic/interactive ggvis plots in a knitr document.
## Generating a static (non-dynamic, non-interactive) version of the plot.
```



This interactive graph is much more efficient than the previous rendering and does exactly as it is intended when run through the console. However, since *ggvis* was designed mainly as a tool for shiny apps, running this code on a knitted html file will result in a static graph.

## 3)Plotly

To use *plotly*, as you will later see, it will be much easier first reorganize the data frame in only 3 columns: Number(the number of passengers), Month, and Year. (Although it may seem counterproductive to reorganize the data at first, realize that the the data analysis cycle is not linear and “re-cleaning” data to suit the developer’s needs is common.)

```
passengers = data.frame(Number=AirPassengers)
passengers[ 'Month' ]=month.name
a <- rep(1949,12)
for (year in 1950:1960){
a=c(a,rep(year,12))}
passengers[ 'Year' ]=a

head(passengers)
```

```
##   Number Month Year
## 1    112   Jan 1949
## 2    118   Feb 1949
## 3    132   Mar 1949
## 4    129   Apr 1949
## 5    121   May 1949
## 6    135   Jun 1949
```

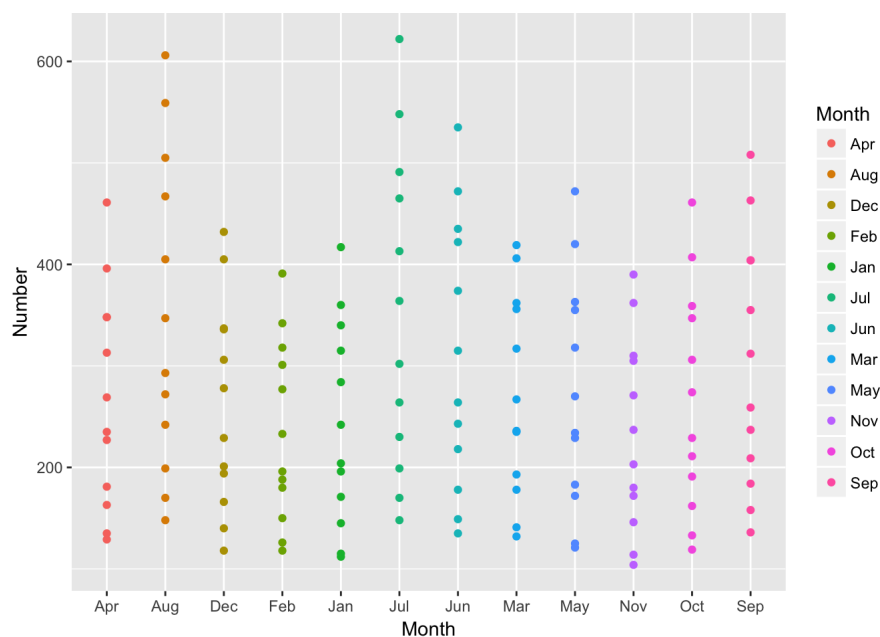
Once we have “re-cleaned” the data, we are ready to graph it using *plotly*. In this case, it will be easier to construct our chart using *ggplot* first. When using *ggplotly*, a slider will automatically be created according to the values dictated in the aesthetic element ‘frame’ of our initial plot, p. Statically, the graph will show all possible points of the data frame but, interactively (when we run p through *ggplotly*), it will switch from year to year.

```
plot <- ggplot(passengers, aes(x=Month, y=Number,color=Month)) +
  geom_point(aes(frame=Year))
```

```
## Warning: Ignoring unknown aesthetics: frame
```

```
plot
```

```
## Don't know how to automatically pick scale for object of type ts. Defaulting to continuous.
```



```
ggplotly(plot)
```

```
## We recommend that you use the dev version of ggplot2 with `ggplotly()`
## Install it with: `devtools::install_github('hadley/ggplot2')`
## Don't know how to automatically pick scale for object of type ts. Defaulting to continuous.
```

Note: Although in this case it is more efficient to use *plotly*, there are still drawbacks. For example, if we were to construct the chart using `plot_ly` we would need to define all parameters for the slider ourselves which is a somewhat complex process you can read more about [here](#).<sup>6</sup>

In conclusion, *plotly* is a very useful and powerful package which allows not only for the creation of interactive plots but also the conversion of static charts into an interactive format. Its ability to run on multiple platforms allows for greater collaboration across multiple programming languages as well. While I have covered some of the more basic functions of *plotly*, there are many more features that exist from basic barcharts to complex interactive maps and 3D graphs which will be useful for more advanced data analysis.<sup>7</sup>

## References:

1:<https://en.wikipedia.org/wiki/Plotly>

2:<https://plot.ly/d3-js-for-r-and-shiny-charts/>

3:[https://images.plot.ly/plotly-documentation/images/r\\_cheat\\_sheet.pdf](https://images.plot.ly/plotly-documentation/images/r_cheat_sheet.pdf)

4:<https://plot.ly/r/#basic-charts>

5:<https://plot.ly/r/reference/>

6:<https://plot.ly/r/sliders/>

7:<https://moderndata.plot.ly/interactive-r-visualizations-with-d3-ggplot2-rstudio/>