October Post

Helen Deng

10/28/2017

Plotly: An Online Data Analytics and Visualization Tool

Introduction

Throughout the course of Stat133, we've learned various approaches to visualizing data. Whether it's Base R's `plot()` method generating a simple and effective scatter of the data, or ggplot's endless opportunities to create beautifully colored graphs, it seems as though we've thoroughly learned all methods for data exploration. But, there's actually more we haven't covered! R has a package, *plotly*, that changes these flat graphs into interactive visuals for the user. Plotly offers six main products for online graphing and analytics: Plot.ly, API libraries for various coding languages, figure convertors, Plotly Apps for Google Chrome, Plotly.js JavaScript library, and Plotly Enterprise. Plotly uses the *htmlwidget* framework, which allows plots to work without an internet connection.

In this blog post, I will be focusing mostly on Plot.ly. Plot.ly has a graphical user interface to make creating graphs faster and more efficient. The graphs we've created in Base R or ggplot aren't catered towards an audience. Instead, the flat images are sometimes difficult to interpret if either the visual wasn't created well or if the creator didn't include a comprehensive enough description. With visuals created through plot.ly, the audience can hover over the graph, zooming in and out at will, panning around the data, and more!

Motivation

As I was researching various topics I could write the post on, plotly jumped out as an extremely important and useful package to know! For me, good visuals are one of the most important aspects of data analysis. If a visual is difficult to interpret, any analysis drawn from it becomes questionable. Even if a visual is created effectively, sometimes the actual graphic is unappealing. Once I stumbled upon this package after frantically googling topics, it was a no brainer that I'd write my first post on plotly.

All of the data visualized in this blog post is from data sets that default ship with R.

Basics of Plotly

Plotly is similar to the ggplot's construction method of *adding layers*. Every time you utilize plotly, the main function will be `plot_ly()`. The first argument will be the data frame (or matrix) that you are drawing the columns from. The following arguments in the function will vary depending on which graphing method you choose to use (similar to ggplot). Let's explore some of the basics of plotly by looking at a scatterplot!

First, let's load the package.

```r
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
```

Simple Scatter

The data set we will be visualizing is under “Locations of Earthquakes off Fiji”. This data frame has 1000 observations on 5 numeric variables: lat (latitude), long (longitude), depth (depth in kilometers), mag (Richter magnitude), and stations (number of stations reporting).
The next step we can take to create a scatterplot is to call the `plot_ly()` function with the parameters we would want to visualize.

```
plot_ly(quakes, x = ~lat, y = ~long) %>% layout(title = "Latitude vs. Longitude of Earthquakes off Fiji")
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```

This function call creates a default scatter plot. The `layout()` function just adds a title to the scatter for a clearer visual!

Sidenote: While using plotly, you’ll probably find the pipe operator to be extremely useful! Without it, method calls would be more similar to our beginning use of dplyr. Too many function calls in one line would get too confusing! With the pipe operator, plotly syntax becomes more similar to ggplot’s “+” with adding layers. In the previous method call, you could clearly see that I created a scatter, and then set the title. If I hadn’t used the pipe operator, the syntax would change to
`layout(plot_ly(quakes, x = ~lat, y = ~long), title = "Latitude vs. Longitude of Earthquakes off Fiji")`. This method calls the same output, but is much less user friendly.

Even from this simple function call, the interactive graphic is already much more alluring than the ones we’ve created with ggplot. **Try spending some time familiarizing yourself with the possibilities of plotly! You can hover over specific data points, zoom in, and press the various icons to explore further.**

Plotly Capabilities

Plotly pairs with various `add_*()` functions such as `add_markers()`, `add_lines()`, `add_histogram()`, `add_surface()`, etc. to create a customized visual. To change a simple scatterplot into a line plot, we can utilize the `add_lines()` function. Let’s try it!

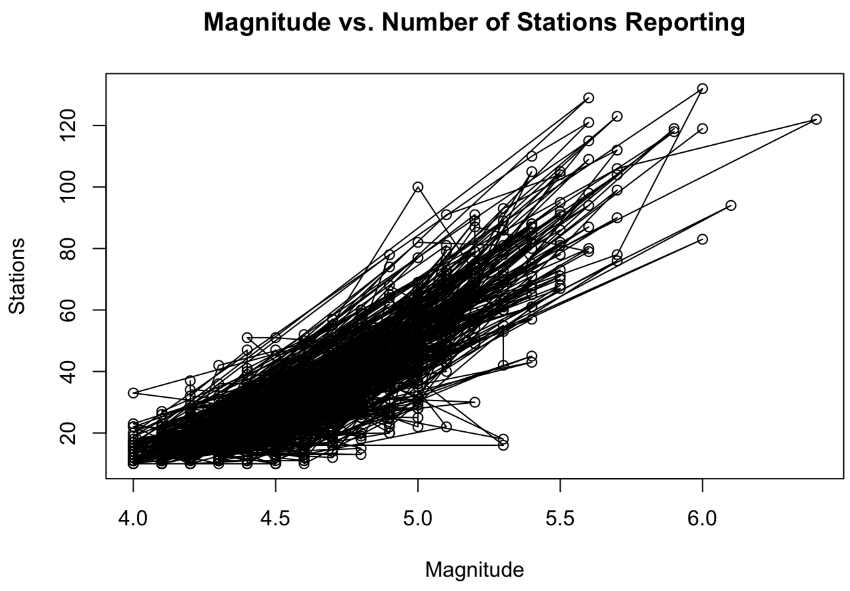
Line Plot

Because it wouldn’t make sense to connect lines between the latitude and longitude of earthquakes, let’s try looking at the relationship between the earthquakes’ magnitude and number of stations reporting on it.

```
plot_ly(quakes, x = ~mag, y = ~stations) %>% add_lines() %>% layout(title = "Magnitude vs. Number of Stations Repo
rting")
```

The reason this graph looks so wonky is because the Richter scale works by tenths of a number. This explains why it looks like there are strange vertical lines. However despite this irregularity, the data makes sense! As the magnitude of earthquakes increases, so does the number of stations. If we try to recreate this in Base R, the plot looks messed up.

```
plot(quakes$mag, quakes$stations, type = "o", main = "Magnitude vs. Number of Stations Reporting", xlab = "Magnitu
de", ylab = "Stations")
```



Base R doesn't know how to connect the data points correctly when multiple points have the same x-value and different y-values. ggplot deals with this data set better and creates the same graphic as plotly, but loses the graph's interactive behavior.

Histogram

Similar to Base R and ggplot, plotly can also create histograms. We can take a look at the histogram of depths of earthquakes!

```
plot_ly(quakes, x = ~depth) %>% add_histogram() %>% layout(title = "Histogram of the Depths of Earthquakes")
```

Although we've learned how to plot a similar histogram in both Base R and ggplot, neither of those platforms allow the user to retrieve the height of the bins without either looking at the raw data or estimating from the graph. With plotly, a simple hover over the bars gives you both the height and center of the bin!

Merging Multiple Graphs

A capability not found at all in Base R or ggplot is merging various graphs of your choosing. Plotly has the function `subplot()` to create a customized graphic of graphs. For example, let's say we just combine all of the plots we've explored so far! There's no way that we could achieve this even in ggplot. However with Plotly, we can achieve this goal with a simple method call.

```
subplot(plot_ly(quakes, x = ~lat, y = ~long, name = "scatter"),
        plot_ly(quakes, x = ~mag, y = ~stations) %>% add_lines(name = "line plot"),
        plot_ly(quakes, x = ~depth) %>% add_histogram(name = "histogram")) %>%
  layout(title = "Mashing of Graphs")
```

```
## No trace type specified:
##   Based on info supplied, a 'scatter' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#scatter
```

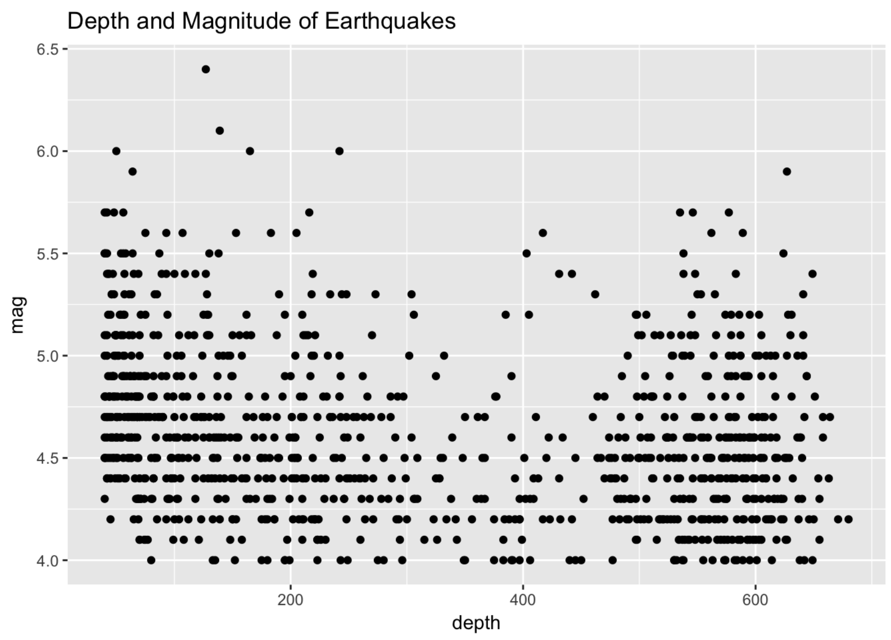
```
## No scatter mode specifed:
##   Setting the mode to markers
##   Read more about this attribute -> https://plot.ly/r/reference/#scatter-mode
```

By calling this function and organizing these graphs next to each other, we can analyze all three at the same time without constantly scrolling back and forth through the file.

Data Animation

Another feature of plotly is how it supports key frame animations through a `frame` argument. This attribute of plotly is another one of the six main products: figure convertors. We can combine ggplot with plotly using the function `ggplotly()` to make an awesome interactive animation! To do this, we first plot with ggplot as we would normally. Let's assume we want to look more closely at the relationship with the depth and magnitude of earthquakes.

```
ggplot(quakes, aes(depth, mag)) + geom_point() + ggtitle("Depth and Magnitude of Earthquakes")
```



Based on our previous line plot with magnitude vs. number of stations, we already know that there is a positive correlation between the two variables. But, let's assume we didn't know! What if we wanted to visualize if both greater depth and magnitude correlated to a greater number of stations? We can do this with a sliding animation through plotly. To accomplish this, we call `ggplotly()` around our original ggplot call, and we add a `frame` argument inside of the `geom_point()`.

```
ggplotly(ggplot(quakes, aes(depth, mag)) + geom_point(aes(frame = stations)) + ggtitle("Depth and Magnitude of Ear
thquakes Across Number of Stations"))
```

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

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

Woo! Go ahead and play with the slider and play button. Through this graphic, we can tell that although magnitude increases with the number of stations reporting, depth appears much more random and doesn't follow a pattern. Through the combination of ggplot and plotly, we could find a relationship between these three variables very quickly.

Extra Cool Function in plot_ly

The `add_surface()` function changes a matrix into an actual 3D figure. For this example, I'll use the volcano data set that gives topographic information on a volcano in Auckland. If we use the same command for a simple scatter, a 2D density plot appears.

```
plot_ly(z = ~volcano) %>% layout(title = "2D Density Heatmap of Volcano")
```

```
## No trace type specified:
##   Based on info supplied, a 'heatmap' trace seems appropriate.
##   Read more about this trace type -> https://plot.ly/r/reference/#heatmap
```

With the addition of the `add_surface()` function, we can bring the plot to life!

```
plot_ly(z = ~volcano) %>% add_surface() %>% layout(title = "3D Topography of Volcano")
```

Conclusion

Although the commands we’ve learned through base R and ggplot are effective at creating visuals, plotly brings them to life. Instead of looking at a graphic someone else created, plotly’s GUI allows the audience to have some hands on interaction with the data visuals. Even if the users have no experience in data or statistics, plotly allows them to have some interest by exploring the data themselves. We can also use plotly to merge multiple graphs or create a key frame animation of our data. Plotly expands past the capabilities of base R and ggplot as a extremely effective data analytics and visualization Tool

References

- Plotly History
- Plotly Overview
- Basic Visualizations
- Data Animation
- 3D Surface Plots
- htmlwidgets
- Fiji Earthquake Data
- Maunga Whau Volcano Data
- Plotly Logo