Exploring Data Visualization In R Using Plotly

Victor Xu (Post 02)

November 30, 2017

Introduction

Plotly is a web application and library for R, useful for creating interactive web graphics. More importantly, using Plotly we are not only able to create, but also easily share our data visualizations to viewers. In this post, I will focus on creating several different interactive graphs to explore the capabilities of Plotly. This is essentially my motivation for writing this post - to examine how to more easily create, sharable interactive visualizations in R.

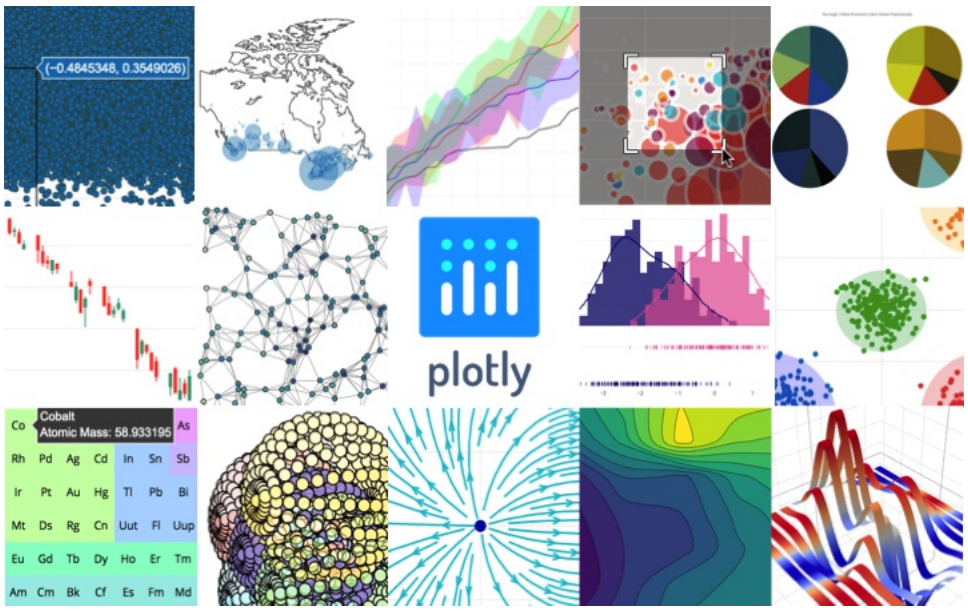

Image: Plotly logo and some sample visualizations the program is compatible with reproducing

More importantly, Plotly has yet to be addressed in Statistics 133 - the course I am submitting this post for. When it comes to creating visualizations in R, the most common processes include either ‘ggplot’ or ‘shiny’. Whereas ggplot is able to generate beautiful visualizations, when used in isolation it is not sharable. ‘Shiny’ does allow us to create interactive web apps; however, the process has a significant learning curve. Shiny can be compared to a dense encyclopedia - there’s plenty of flexibility and customization possible, but it’s up to the user to figure out all the small details. In situations where all we need are quickly sharable visualizations (side note: which are easy to embed as HTML into a blog or website as well), Plotly seems to be the more efficient, and arguably better option for the majority of everyday users.

Thus, Plotly presents an original opportunity to go beyond what I’ve learned from lectures, labs, and tutorials. So read on! Whether you’re a fellow student in Stat 133, aspiring data scientist, website designer, or just curious about expanding your knowledge - this post will hopefully be useful in providing some concrete examples about Plotly. Let our exploration begin!

Background

In our post, let’s start by observing the Plotly R library definitionally:

“Plotly’s R graphing library makes interactive, publication-quality graphs online. Examples include line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, and 3D (WebGL based) charts.” - Plotly, R Handbook

As you can see, there are plenty of visualizations which Plotly supports. Each type of visualization in separation is enough to warrant follow-up posts containing more demos. So as an exploratory post (to better organize and manage the length of this tutorial) we’ll cover the 3 most-common examples (3 different types of visualizations). We’ll start with a 2D scatter plot, progress into a histogram, and conclude with a 3D scatterplot.

Example 1: 2D Scatterplot

To begin, let’s install the Plotly library package in R, by calling install.packages(“plotly”):

```
#This is currently commented out, as the package has already been installed once on my computer
#install.packages("plotly")
```

Next, let’s create the scatterplot. A basic scatterplot is simple to make, and has the benefit of tooltips (advanced settings) when the user’s mouse hovers over it.

We can specify a scatterplot by indicating that mode = “markers” in the plot_ly function (from the Plotly library). For this example, we’ll draw on a standard dataset built into R, mtcars (Motor Trend Car Road Tests).

From official R documentation, this dataset was “Extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).” In our sample scatterplot, we’ll have weight on the x axis, and mpg (miles/gallon) on the y axis. Calling on ‘layout’, we are able to customize the plot with titles and axis labels.

```
#Calling on the library
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 3.4.3
```

```
## Loading required package: ggplot2
## Warning: package 'ggplot2' was built under R version 3.4.2
```

```
##
## 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
```

```r
#Creating the scatterplot with Plot_ly
plot_ly(mtcars, x = ~wt, y = ~mpg, mode = "markers", text = text) %>%
        layout(title = "Plot 1: mpg vs. weight for mtcars sample dataset",
         xaxis = list(title = "weight (in 1000's pounds)"),
         yaxis = list(title = "mpg (miles per gallon)"))
```

```
## 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
```

Discussion/Reflection:

Immediately, we are able to see that the plot is interactive, even within the knitted HTML document containing this post! We are able to hover over specific data points for greater specificity (coordinates). Furthermore, we are able to click/hold anywhere on the plot, and drag out a square. For this command, you should see only the square (which you're creating) highlighted, while the background is darkened. This is Plotly's way of allowing users to quickly zoom in on specific parts in the plot.

Pretty cool! Be sure to try it out on the scatterplot we have just created above.

From here, there are many ways we can share the plot we created. The direct way (i.e. working in RStudio under a R Notebook), would be knitting the document as HTML. The HTML file containing the interactive Plotly visuals can be shared online, published, sent as email attachments, etc. in the traditional way. For advanced users, it's also possible to integrate Plotly into Shiny to create an advanced custom web app, although that goes beyond the scope of this current post. Lastly, because Plotly (in itself) contains a web application potion, it's possible to store visualizations using simple services on "plot.ly". Note that this process could involve a subscription cost, segmented into categorized models such as "student" or "professional"-tier. Using these paid services, we can store our visualizations onto the cloud for simplified retrieval and collaboration between subscribers. In this sense, the web app portion of Plotly can be thought of as a separate, "freemium" (business plan) option.

Going back to our original scatterplot example, we can of course, adjust other elements like color or size of the plotted points. In fact, this process is similar to how we'd adjust any ordinary scatterplot in R. As a quick demonstration, here's the scatterplot after playing around with some of these adjustments:

```r
#Calling on the library
library(plotly)

#Creating the scatterplot with Plot_ly
#This time, we are adjusting the color of the points to be continuous (spectrum)
#--with the scale (legend) displayed to the right of the plot


#Furthermore, we adjusted the size of the points, to be correlated with 'hp'
#--which is a variable in the mtcars dataset for 'gross horsepower' (of a car)


plot_ly(mtcars, x = ~wt, y = ~mpg, mode = "markers",
        color = ~disp, size = ~hp) %>%
        layout(title = "Plot 2: mpg vs. weight for mtcars sample dataset (modified)",
         xaxis = list(title = "weight (in 1000's pounds)"),
         yaxis = list(title = "mpg (miles per gallon)"))
```

```
## 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
```

Example 2:

The following example covers visualization with histograms (and by extension, barplots). For this example, we'll draw on another standard (public) database, called 'iris'. Here's a description of what the database contains, from the official R documentation guide:

"This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica."

Iris is a dataframe with 150 cases (rows) and 5 variables (columns) named: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, and Species.

In our second example, we are plotting the Species (of flower) with their respective Sepal Lengths. We can create the plot using the type = "histogram" argument in the plot_ly function. We will also make the bar's colors equal to Species (making the plot easier to read).

```r
#Calling on the 'iris' dataset
library(datasets)
data("iris")

plot_ly(data=iris, x = ~Species, y = ~Sepal.Length, type = "histogram",
        color = ~Species) %>%
        layout(title = "Plot 3: Sepal.Length for Species on a histogram")
```

Discussion/Reflection:

Again, we are able to hover over parts of the plot (for more details), and zoom in by dragging a square using the cursor. We can observe that the plot is scaled automatically, with a helpful legend (explaining what the colors correspond to) on the right of the plot.

What's really cool is that you can click on the legend (try double clicking on the word 'setosa' or the green box within the legend). This reflows the plot so that only elements containing that specific species ('setosa') is displayed. You can double click on the 'setosa' label again, to return to a complete plot with everything displayed.

Now try clicking on 'setosa' just one time (instead of double clicking). You should notice that instead of only bringing up results with setosa, now the plot reflows to show everything EXCEPT for setosa. In other words, we have removed setosa from the interactive visualization! This feature makes it handy to compare results between differing species, say we want to only look at setosa and versicolor together. Our dynamic barplot allows this to be possible, eliminating what we don't want, from view.

Example 3:

The final example covers 3D scatterplots! We can create 3D scatterplots using the type = "scatter3d" argument in the plot_ly function.

In this example, we are creating a simple, fictitious dataset containing 100 data points. We will define the variables temp (which contains 100 random normals with some mean and standard deviation), pressure (also containing 100 random normals), and dtime (simply the numbers from 1 to 100).

Now that we have our 3 variables defined, we can specify them for the x, y, z dimensions in plot_ly function. The code and output are as follows:

```r
#Seed for random number generation
set.seed(2017-11-30)
#Defining the temp variable
temp <- rnorm(100, mean = 30, sd = 5)
#Defining the temp variable
pressure <- rnorm(100)
#Defining the dtime variable
dtime <- 1:100

#Plotting, with color as a function of temp
plot_ly(x = temp, y = pressure, z = dtime,
        type = "scatter3d", mode = "markers", color = temp) %>%
        layout(title = "Plot 4: A 3D Scatterplot Based on Temperature, Pressure, and Time")
```

Discussion/Reflection:

This interactive scatterplot can be viewed from different angles (in 3D space) by clicking and dragging the mouse around. Give it a try!

Like with the plots we made before, scaling is done automatically without additional adjustments needed. Again, we are able to zoom in and out - although the easiest way I've found is through a multitouch gesture (pinching in/out on the track pack of a computer), now that clicking is reserved for panning movements in 3D space. Hovering the mouse at any particular point allows us to see the precise coordinates in x, y, z. The interactive nature of Plotly really comes alive in these complex 3D plots, where a sense of depth is easier understood through animation and movement.

Conclusion & Take-Away:

Congratulations!

By making it to the end of this post, you've obtained a deeper understanding of how Plotly can be be used to generate interactive visualizations, both in 2D and 3D space. Implicitly, we can observe how simple it is to use Plotly and generate dynamic visualizations. Small chunks of code can produce powerful plots which gives the viewer freedom to explore more.

Explicitly, Plotly's strengths seem to shine even brighter in 3D visualizations, where depth information is conveyed through movement.

Again, the simplicity and flexibility of Plotly - both in the production and sharing process - makes it a great alternative (or conjunction) to standard ggplot/Shiny. In the end, it makes the process of examining complex datasets more humanized and engaging. In my book, that's a win.

References:

1. https://plot.ly/r/
2. https://cran.r-project.org/web/packages/plotly/plotly.pdf
3. https://www.analyticsvidhya.com/blog/2017/01/beginners-guide-to-create-beautiful-interactive-data-visualizations-using-plotly-in-r-and-python/
4. https://www.youtube.com/watch?v=loRDhzQ9SOE
5. https://github.com/ropensci/plotly
6. http://neondataskills.org/R/Plotly
7. https://plotly-book.cpsievert.me/installation.html
8. https://www.rdocumentation.org/packages/datasets/versions/3.4.1/topics/mtcars