Week 8. Big Data Analytics
Visualization with `plotly` for R

Hyeonsu B. Kang
hyk149@eng.ucsd.edu

May 2016

1 Adding interactivity to graphs

Plotly is a collaboration platform for modern data science. It lets you share a fully web-based dashboards with your colleagues and create web-browser rendered interactive plots. In this session, we will go over some of the features in `plotly` such as overlaying graphs, adding embedded interactivity and rendering graphs on a web-browser with `Shiny`. For more information about `plotly`, please visit `https://plot.ly/`.

1.1 Embedded interactivity of graphs

To create a plotly visualization, we start with `plot_ly()`. If we use a default data set called `economics` and plotted the $\frac{\text{unemploy}}{\text{pop}}$ value with `plot_ly()` it automatically embeds interactivity within the graph.

```
> install.packages("plotly") # if you haven't installed the package
> library(plotly)
> p = plot_ly(economics, x = date, y = unemploy / pop)
> p
```

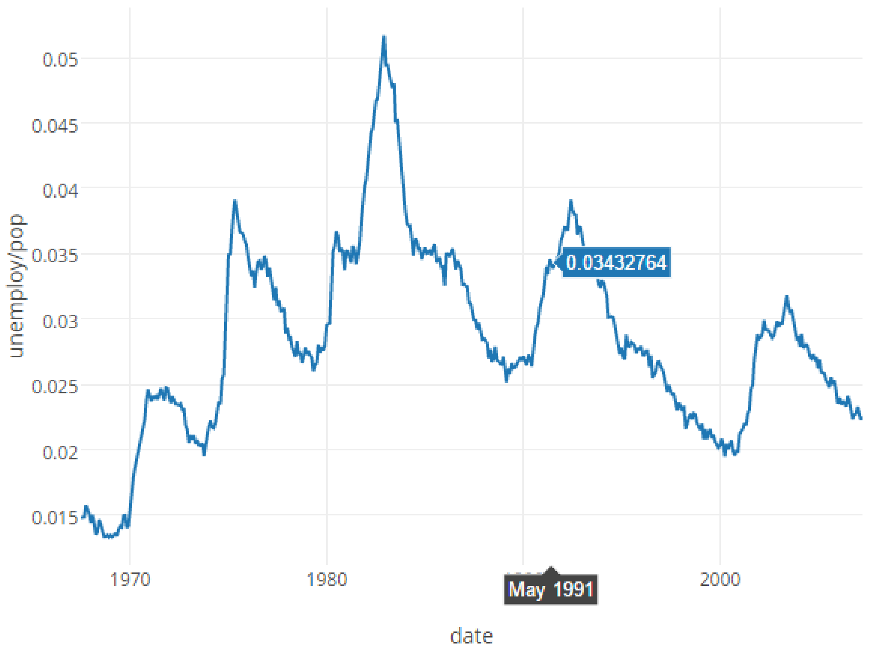

Figure 1: Graph with embedded interactivity and `add_trace()` for composing multi-plot graphs

A plotly visualization is composed of one (or more) trace(s), and every trace has a `type` (the default type is 'scatter'). The arguments or properties that a trace will respect (documented here) depend on its type. A scatter trace respects `mode`, which can be any combination of "lines", "markers", "text" joined with a "+".

```
> plot_ly(economics, x = date, y = unemploy / pop, type = "scatter", mode = "markers")
> plot_ly(economics, x = date, y = unemploy / pop, type = "scatter", mode = "markers+lines")
```

You can manually add a trace to an existing plot with `add_trace()`. In that case, you'll want to either `name` your traces, or hide the legend by setting `showlegend = FALSE`. Let us use a statistics function `loess()` to fit a polynomial surface (Fig. 2).

```
> m <- loess(unemploy / pop ~ as.numeric(date), data = economics)
> p <- plot_ly(economics, x = date, y = unemploy / pop, name = "raw")
> p <- add_trace(p, x = date, y = fitted(m), name = "loess")
> p
```

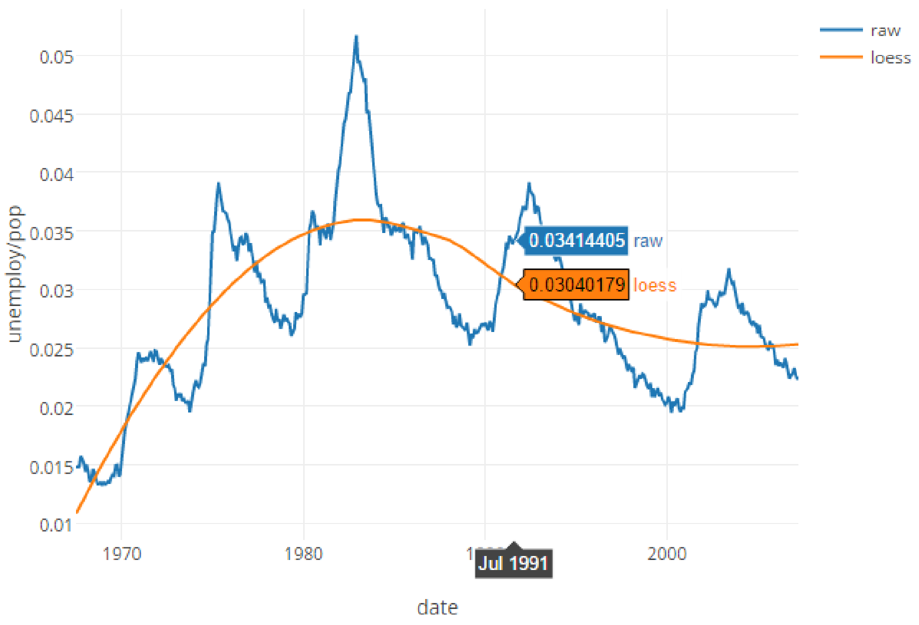

Figure 2: Graph with a fitted polynomial surface (line)

`Plotly` was designed with a pipeable interface in mind, so you can also use the `%>%` operator to modify your plots:

```
> p <- economics %>%
+     plot_ly(x = date, y = unemploy / pop) %>%
+     add_trace(x = date, y = fitted(m)) %>%
+     layout(showlegend = F)
> p
```

Furthermore, `plot_ly()`, `add_trace()`, and `layout()`, all accept a data frame as their first argument and output a data frame. As a result, we can inter-weave data manipulations and visual mappings in a single pipeline.

```
> p <- economics %>%
+     transform(rate = unemploy / pop) %>%
+     plot_ly(x = date, y = rate) %>%
+     subset(rate == max(rate)) %>%
+     layout(
+         showlegend = F,
+         annotations = list(x = date, y = rate, text = "Peak", showarrow = T)
+     )
> p
```

1.2 Overlaying graphs using histograms

Basic histograms in `plotly` is pretty simple by assigning the type of the graph in `plot_ly()`

```
> data = rnorm(50)
> plot_ly(x = data, type = "histogram")
```

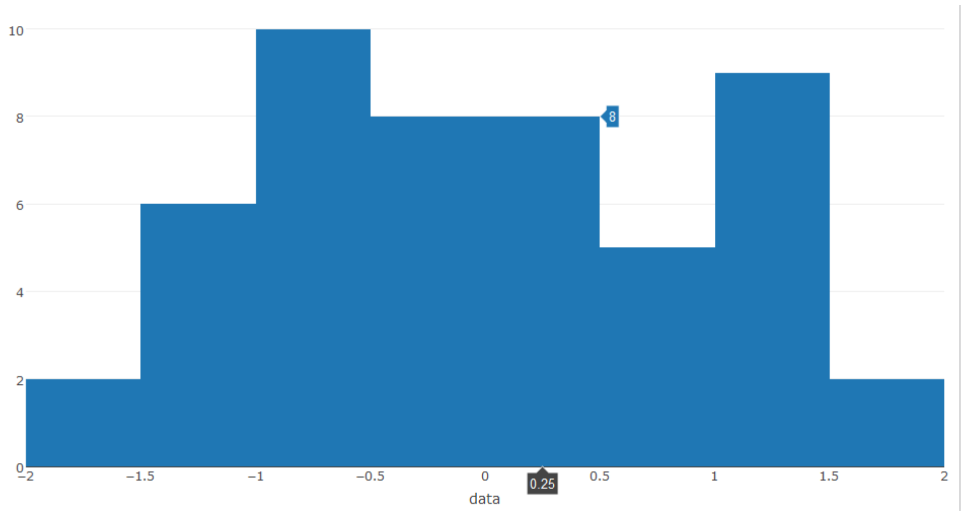

Figure 3: Default histogram

As before, this histogram comes with interactivity, and as you hover your mouse pointer over one of the bars in the graph, it shows both the x-axis value and the count of the corresponding bar. To render two histograms simultaneously, you can use the add_trace() function as followS:

```
> data1 = rnorm(500)
> data2 = rnorm(500) + 1
> plot_ly(x = data1, type = "histogram") %>%
+     add_trace(x = data2, type = "histogram")
```

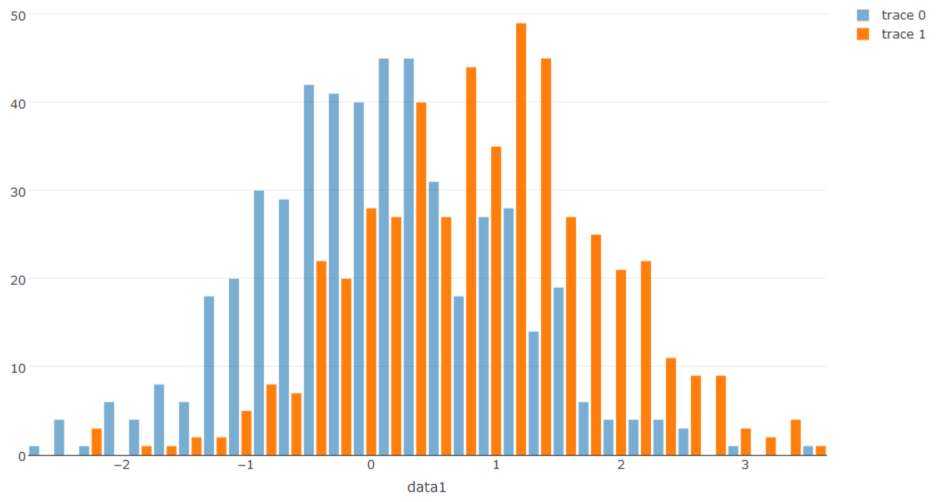

Figure 4: Non-overlayed histogram

However, including an overlay function can give you a different outlook of the graph as follows

```
> plot_ly(x = data1, opacity = 0.6, type = "histogram") %>%
+     add_trace(x = data2, opacity = 0.6, type = "histogram") %>%
+     layout(barmode = "overlay")
```

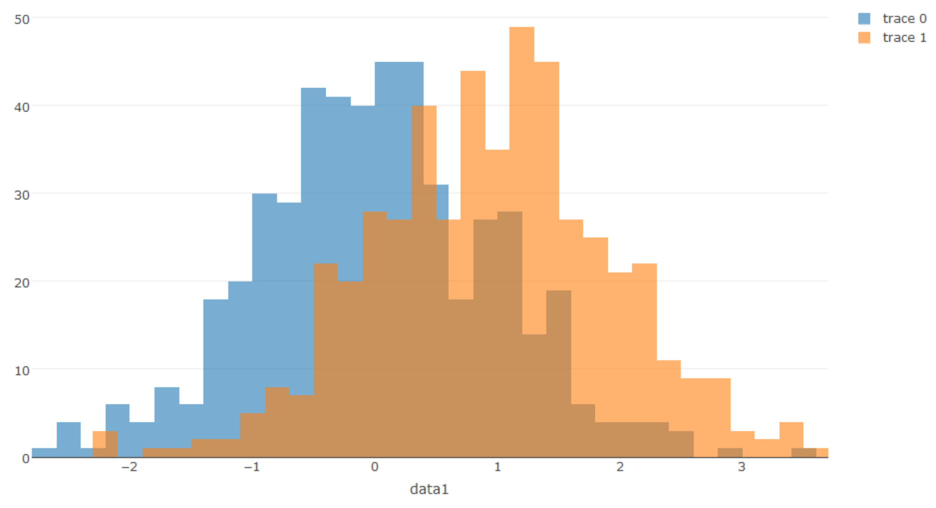

Figure 5: Overlayed histogram

1.3 Scatter plots

Creating a scatter plot needs only specification of the `mode` argument in the `plot_ly()` function to "markers". In addition, specifying the marker color can be done by specifying the `marker` argument

```
> data <- read.csv("https://goo.gl/jZLWh7")
> data <- data[order(data$Men), ]
> p <- plot_ly(data, x = Men, y = School, name = "Men",
+                mode = "markers", marker = list(color = "blue")) %>%
+        layout(
+                title = "Male earnings",
+                xaxis = list(title = "Annual Salary (in thousands)"),
+                margin = list(l = 100),
+                markermode = "overlay"
+        )
> p
```

Overlaying the women's earnings data in this plot can be done with `add_trace()`

```
> p <- plot_ly(data, x = Men, y = School, name = "Men",
+                mode = "markers", marker = list(color = "blue")) %>%
+       add_trace(x = Women, y = School, name = "Women",
+                 mode = "markers", marker = list(color = "pink")) %>%
+       layout(
+               title = "Gender earnings disparity",
+               xaxis = list(title = "Annual Salary (in thousands)"),
+               margin = list(l = 100),
+               markermode = "overlay"
+       )
> p
```

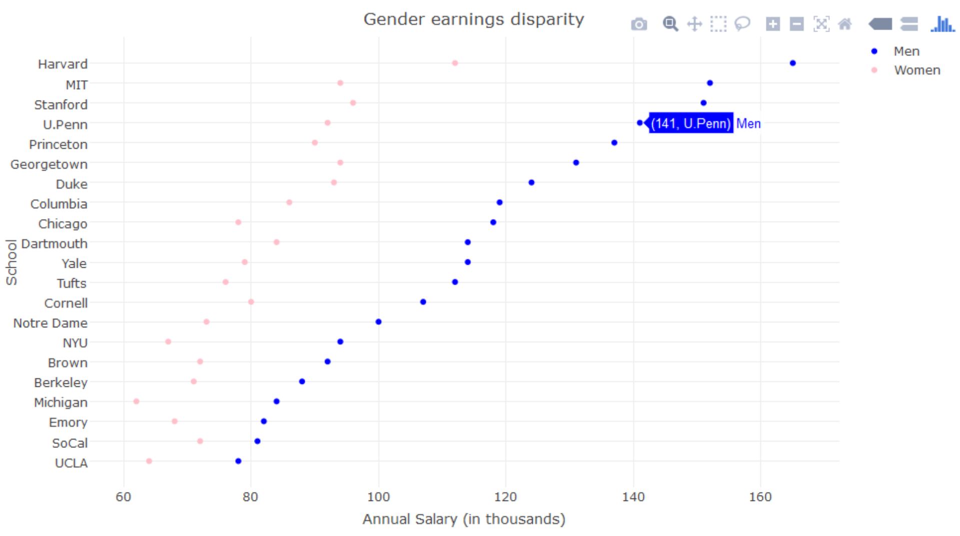

Figure 6: Scatter plot of gender earnings disparity in the decreasing order of men's earnings

The same task can be done differently by modifying the underline structure of the data set. Using the `tidyr`'s `gather()` function, for example, we can gather the separate Men and Women columns under a new Sex column. Then, plotting these can be done by assigning the `color` argument in the `plot_ly()` function:

```
> library(tidyr)
> data <- read.csv("https://goo.gl/jZLWh7")
> data <- data[order(data$Men), ]
# key, value and columns to be gathered under the key column
> gather(data, Sex, value, Women, Men) %>%
+     plot_ly(x = value, y = School, mode = "markers",
+             color = Sex, colors = c("pink", "blue")) %>%
+     layout(
+         title = "Gender earnings disparity",
+         xaxis = list(title = "Annual Salary (in thousands)"),
+         margin = list(l = 100)
+     )
```

Moreover, adding another layer of graph is also possible. Suppose, for example, that we would like to indicate the gap between genders more explicitly. Adding a line graph on top of it is partly overlaying graphs, and thus we can use `add_trace()` as follows:

```
> gather(data, Sex, value, Women, Men) %>%
+     plot_ly(x = value, y = School, mode = "markers",
+             color = Sex, colors = c("pink", "blue")) %>%
+     add_trace(x = value, y = School, mode = "lines",
+               group = School, showlegend = F, line = list(color = "gray")) %>%
+     layout(
+         title = "Gender earnings disparity",
+         xaxis = list(title = "Annual Salary (in thousands)"),
+         margin = list(l = 100)
+     )
```

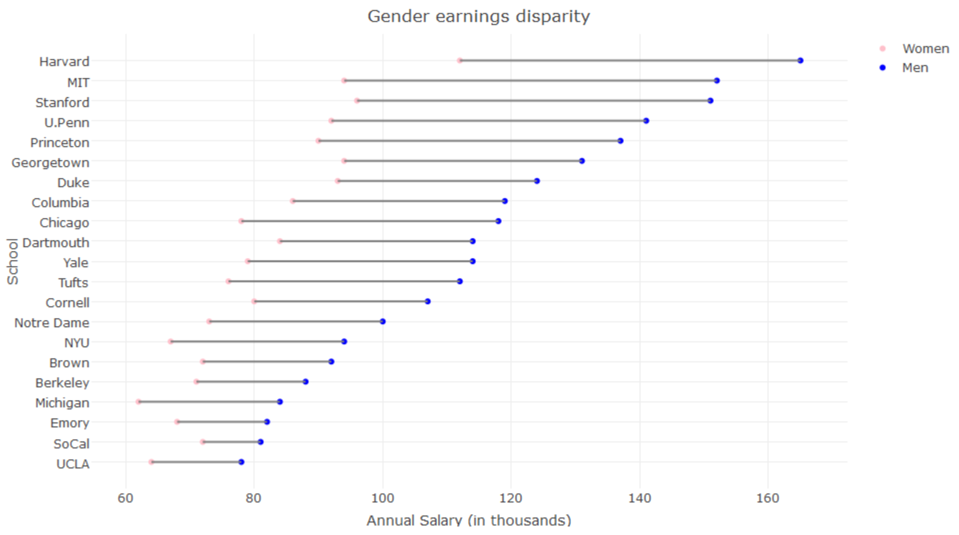

Figure 7: Scatter plot of gender earnings disparity in the decreasing order of men's earnings

Or how about ordering by the size of the earning gap?

```
> data <- data[order(data$gap), ]
> gather(data, Sex, value, Women, Men) %>%
+     plot_ly(x = value, y = School, mode = "markers",
+             color = Sex, colors = c("pink", "blue")) %>%
+     add_trace(x = value, y = School, mode = "lines",
+               group = School, showlegend = F, line = list(color = "gray")) %>%
+     layout(
+         title = "Gender earnings disparity",
+         xaxis = list(title = "Annual Salary (in thousands)"),
+         margin = list(l = 100)
+     )
```

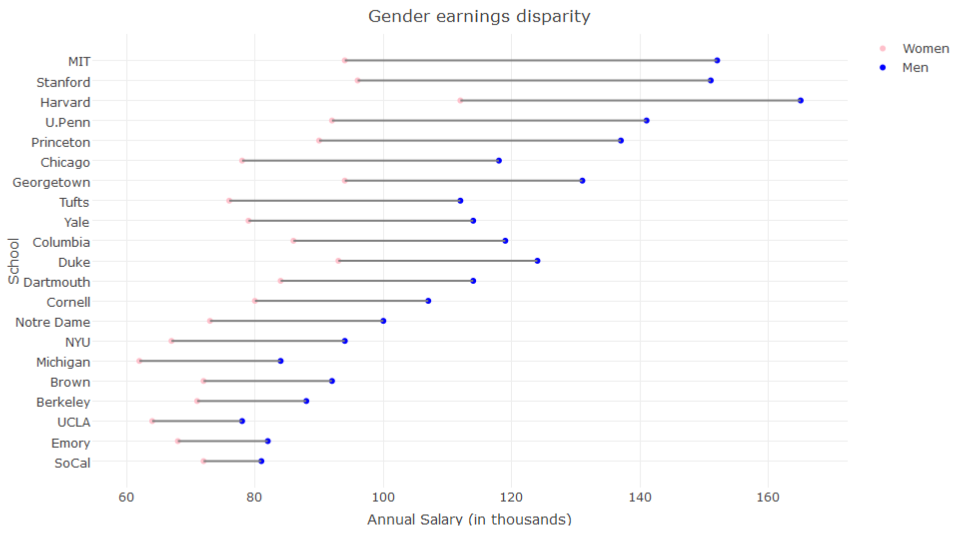

Figure 8: Scatter plot of gender earnings disparity in the decreasing order of earnings gaps

1.4 Adding UI control components with `plotly`

Please download the stock visualization code from `https://goo.gl/lkBbnM`. Adding UI control components for more complete interactivity, such as buttons or slider bars (and many more that were not included here) can be done by specifying the corresponding arguments in `layout()`. Here, the buttons directly manipulate the size of a step on the x axis and thus different options are specified in the list in the `buttons` argument inside of `rangeselector`. Creating a small component view of a range slider can be done by specifying the `rangeslider` inside of the `xaxis` argument.

Listing 1: Microsoft and Apple's stock price visualization using R

```
library(plotly)
library(quantmod)

# Download some data
getSymbols(Symbols = c("AAPL", "MSFT"))
ds <- data.frame(Date = index(AAPL), AAPL[,6], MSFT[,6])

# Graph
plot_ly(ds, x = Date, y = AAPL.Adjusted, mode = "lines + markers", name = "Apple") %>%
  add_trace(x = Date, y = MSFT.Adjusted, name = "Microsoft") %>%
  layout(
    title = "Stock Prices",
    xaxis = list(
              rangeselector = list(
                        buttons = list(
                          list(
                            count = 3,
                            label = "3 mo",
                            step = "month",
                            stepmode = "backward"),
                          list(
                            count = 6,
                            label = "6 mo",
                            step = "month",
                            stepmode = "backward"),
                          list(
                            count = 1,
                            label = "1 yr",
                            step = "year",
                            stepmode = "backward"),
                          list(
                            count = 1,
                            label = "YTD",
                            step = "year",
                            stepmode = "todate"),
                          list(step = "all")
                        )
              ),

        rangeslider = list(type = "date")
    ),

    yaxis = list(title = "Price")
  )
```

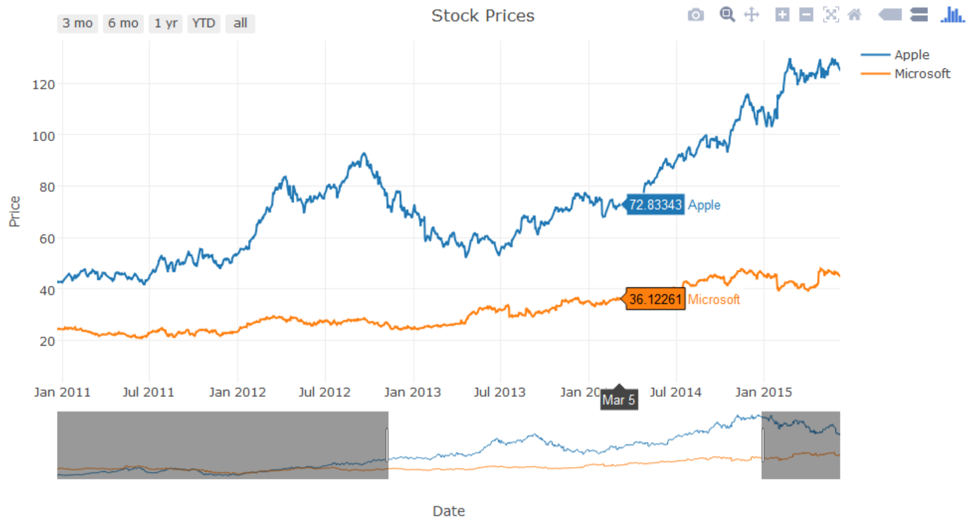

Figure 9: Microsoft and Apple's stock price visualization

1.5 Rendering interactive graphs in a web browser

Using `plotly` with `shiny` lets you render your interactive graphs on a web browser as well as publish them to the internet. Although we will not cover how to publish the graphics to the internet in this session, let us see how to put your graphs on a web browser. First, we need to install another R package called `shiny` and then have two R scripts called `ui.R` and `server.R`. You can download the scripts from `https://goo.gl/HACyvo` (`ui.R`) and `https://goo.gl/3gsND6` (`server.R`). You can manually save the files from the given URLs and locate them in your project folder, or you can use the following lines to do so:

```
> install.packages("shiny")
> library(shiny)
> download.file("https://goo.gl/HACyvo", "ui.R")
> download.file("https://goo.gl/3gsND6", "ui.R")
```

Once you have them saved in the project folder, open one of them in RStudio, and click the "Run App" button. A new window will pop up and if you click the "Open in Browser" button on the new window, the graphics will be transferred to your default web browser.

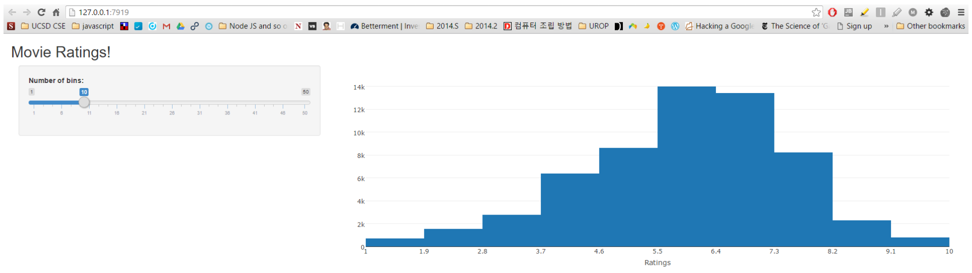

Figure 10: Plotly with Shiny, rendering the graph on a web browser