Upgrade your Visualizations to Make them Interactive with Plotly

Ben Pirie

October 26, 2017

```
# loading appropriate packages
library(readr)
library(ggplot2)
```

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

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(plotly)
```

```
## Warning: package 'plotly' 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
```

```
library(knitr)
```

```
# a quick note: throughout this post I will be using examples building from
# MLS data which you can access in the link under references
players <- read_csv("../data/MLS_players.csv")
```

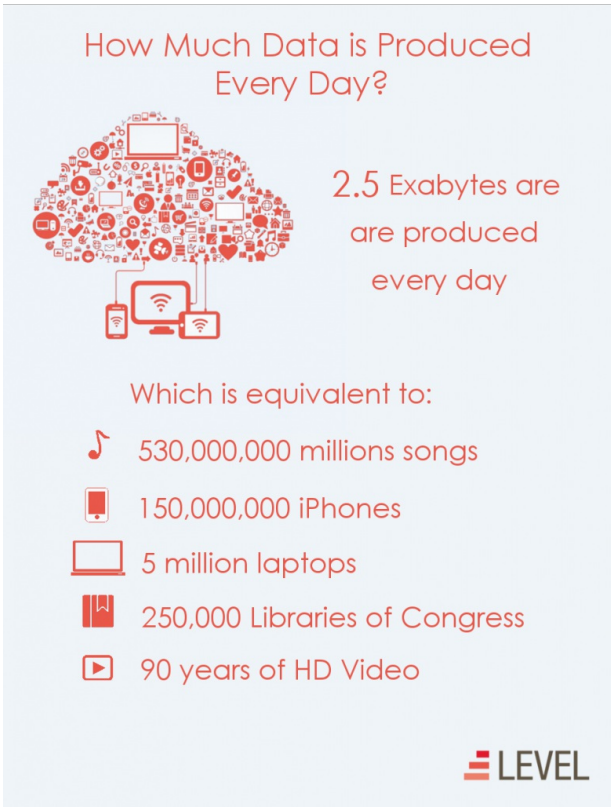
```
## Parsed with column specification:
## cols(
##   Last = col_character(),
##   First = col_character(),
##   Club = col_character(),
##   Pos = col_character(),
##   name = col_character(),
##   baseSalary = col_double(),
##   guarenteedComp = col_double(),
##   Team = col_character(),
##   Min = col_integer(),
##   Shts = col_integer(),
##   G = col_integer(),
##   xG = col_double(),
##   `G-xG` = col_double(),
##   KP = col_integer(),
##   A = col_integer(),
##   xA = col_double(),
##   `xG+xA` = col_double(),
##   unAst = col_double(),
##   touch = col_double()
## )
```

```
players <- mutate(players, Pos = as.factor(Pos))
teams <- read_csv("../data/MLS_teams.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer(),
##   Club = col_character(),
##   AvShtF = col_double(),
##   AvShtA = col_double(),
##   TSR = col_double(),
##   Poss = col_double(),
##   PPG = col_double(),
##   clubAbbr = col_character(),
##   division = col_character()
## )
```

```
## See spec(...) for full column specifications.
```

According to a post by Northeastern University we produce 2.5 Exabytes of data every day. Put into perspective, that is equivalent to streaming HD videos for 90 years straight. As has been proven by the algorithms of major tech companies such as Facebook, Google, and Amazon, this data is an incredible valuable resource. But nothing valuable comes easy; it's easy to miss correlations, trends, or outliers when scanning through rows and rows of numbers. However, what the telescope did for astronomy, the graph does for data analysis. While the graph is a great start, I hope to convince you that there is more we can do to perfect our data visualization skils.



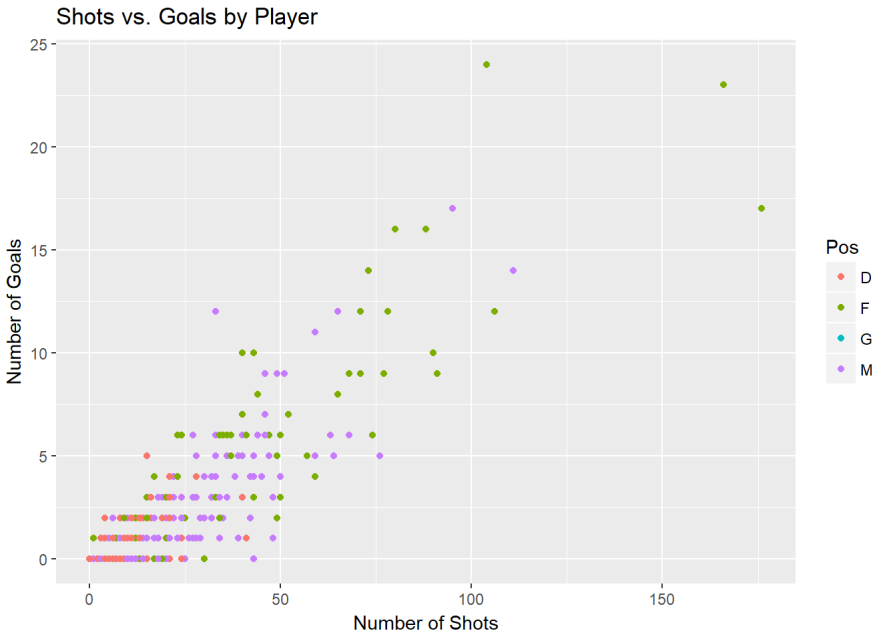
Motivation for Data Visualization

You've heard it a million times *a picture is worth a thousand words*. With all this data floating around, it will get harder and harder to make sense of it. Arguably, the most affective way to show important indings in your data is through graphs. See for yourself: which of the following gives you a better sense of what's going on?

```
#showing data numerically
players[, c("Pos", "Shts", "G")]
```

```
## # A tibble: 398 x 3
##       Pos  Shts     G
##    <fctr> <int> <int>
##  1      D    15     0
##  2      F    71     9
##  3      M    35     2
##  4      M    48     3
##  5      F    88    16
##  6      D    40     3
##  7      F    52     7
##  8      M     1     0
##  9      D     5     0
## 10      D     9     0
## # ... with 388 more rows
```

```
#showing data visually
ggplot(players, aes(x = Shts, y = G)) +
  geom_point(aes(col = Pos)) +
  labs(x = "Number of Shots", y = "Number of Goals",
       title = "Shots vs. Goals by Player")
```



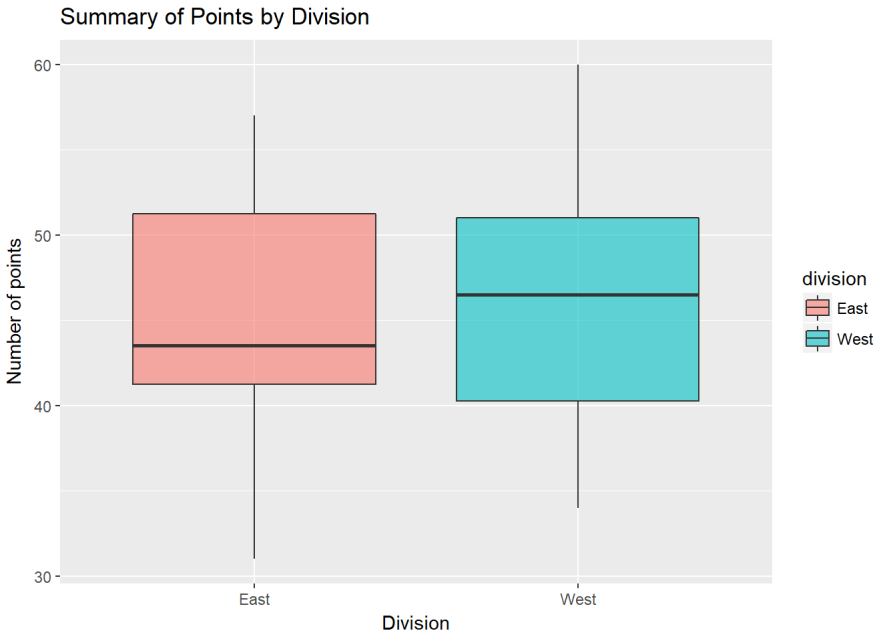
While the table and graph contain all the same information, it is much easier to quickly observe the relationships in graphical format than it is by simply reading through the data. In this case, we can easily see that a player who takes more shots also tends to score more goals. Furthermore, the breakdown between positions (defender, forward, midfielder, goalie) is also clearly indicated by the color of the points in the scatter plot.

Your choice of graphical format is not limited to scatter plots. Let's take a look at another example, this time comparing the spread of points between East and West divisions.

```
#showing data numerially
select(teams, c(Club, PTS, division)) %>% arrange(PTS)
```

```
## # A tibble: 20 x 3
##                      Club   PTS division
##                     <chr> <int>    <chr>
##  1           Chicago Fire    31     East
##  2         Houston Dynamo    34     West
##  3       Columbus Crew SC    36     East
##  4   San Jose Earthquakes    38     West
##  5 Vancouver Whitecaps FC    39     West
##  6        Orlando City SC    41     East
##  7     Philadelphia Union    42     East
##  8 New England Revolution    42     East
##  9       Portland Timbers    44     West
## 10        Montreal Impact    45     East
## 11         Real Salt Lake    46     West
## 12            D.C. United    46     East
## 13   Sporting Kansas City    47     West
## 14    Seattle Sounders FC    48     West
## 15             LA Galaxy    52     West
## 16            Toronto FC    53     East
## 17       New York City FC    54     East
## 18     New York Red Bulls    57     East
## 19        Colorado Rapids    58     West
## 20             *FC Dallas    60     West
```

```
#showing data visually
ggplot(teams, aes(division, PTS)) +
  geom_boxplot(aes(fill = division), alpha = 0.6) +
  labs(x = "Division", y = "Number of points",
       title = "Summary of Points by Division")
```



From the graph we immediately notice that the average team in the West has more points than the average team in the East. We also see that extreme positive values are more likely in the West and extreme negative values are more likely in the East. At first glance, we may be sparked to do further anlysis on whether the West is a stronger division than the East. Again, while all of these insights could also be made looking through the rows of data, we can come to conclusions much quicker using a good visualization.

Hopefully by now you are convinced that graphically displayed information has a the potential to convey more information quickly than tabular information. In fact, you may already even have experience creating beautiful graphs. In practicing with Base-R's plot() function, or using Ggplot2, you may have noticed that the graph does have some limits. The obvious next question is then: *how can I do better than a graph?*

Interactive Graphs

In a presentation by Karl Broman titled "Why aren't all of our graphs interactive?", he oulines several benefits to moving away from static visualizations:

- Allows for better exploration:
    - Ability to tune parameters
    - More easily identify outliers
    - Use one fancy plot rather than 100 static ones
- Better handling of big data:
    - Don't rely on summary statistics
    - Compress your information
    - Allow for zooming in on dense figures
    - Better exploration

The features that most stick out here are the ability to tune your parameters and manipulate the graph as you explore them, and the ability to reduce the number of graphs you need to build.

For example, many times one graph may not sufficiently describe your findings. You may choose to pair it with another graph, a description, or a table. We can limit the need for these additional tools by using interactive plots. These will allow users to drill deeper into the data and isolate variables they might be specifically curious about. In a sense, an interactive graph almost allows you to perform all your data analysis visually, rather than simply demonstrating your results visually.

Others have latched onto this concept, and you can find many blogs discussing the advantages of interactive graphs. This blog describes many of the same benefits as Karl Broman for using "dynamic graphs":

- quicker to produce new graphs once you have setup the first
- easier to update an existing graph or chart with new data
- interactivity makes your graphs more engaging to consumers of information

At this point, I'm sure you're on the edge of your seat wondering, "how can I make an interactive graph?!"

Introduction to Plotly

Meet Plotly - a simple R package which helps you make interactive plots like a pro! Just like we make ggplot objects, we can also make plotly objects in R - they are both based on the principal of *Grammar of Graphics*.

There are two main ways to create Plotly objects.

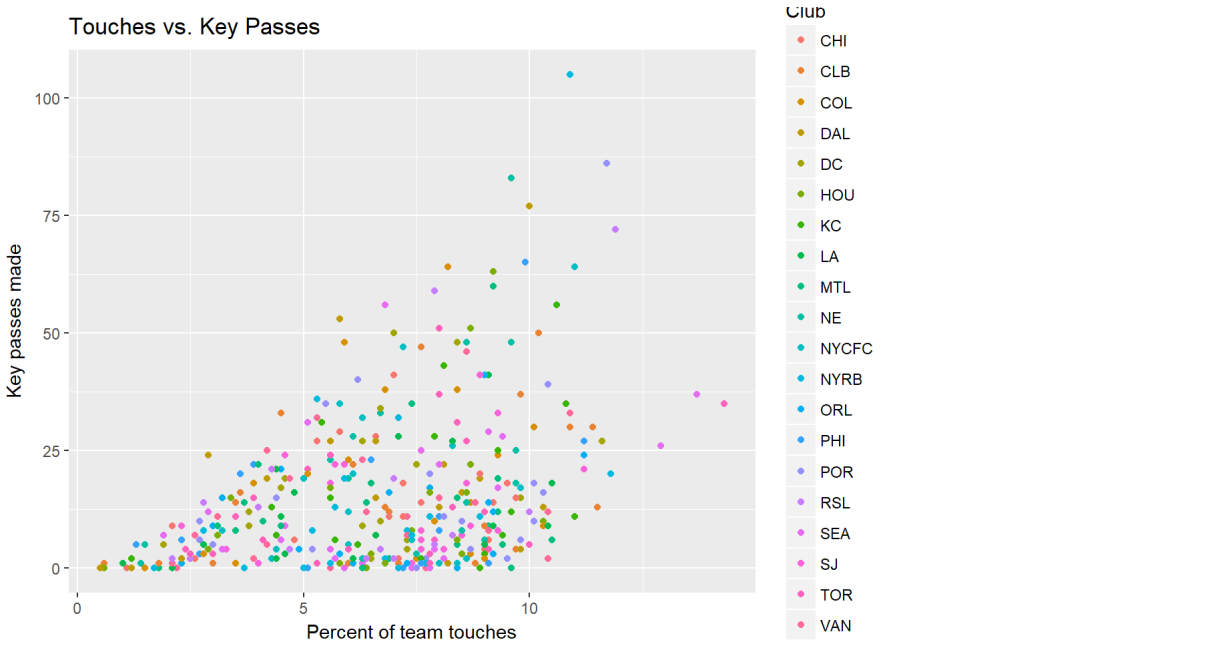
1. The first way to do this is using the plot_ly() function, which transforms data into a plotly object.
2. The second way, which should appeal to those with experience creating ggplot objects, is to use the ggplotly() function. This handy function transforms any ggplot object into a plotly object.

To take advantage of the Ggplot2 skills you may have already developped, I will dive into Plotly using this ggplotly() function.

Example

To introduce different ways you may want to use the Plotly package, I will run through an example of building an interactive graph. The data I am using is data from the 2016 season of Major Leauge Soccer (MLS), and it can be found here. Suppose we are curious about whether there is a correlation between the percent of team touches a player has and the number of key passes he makes. This could tell us if those players who demand a lot of possession of the ball are using that possession wisely. We can build the following graph:

```
# building the ggplot
touchVsKP <- ggplot(players, aes(x = players$touch, y = players$KP)) +
                      geom_point(aes(col = Club)) +
                      labs(x = "Percent of team touches",
                           y = "Key passes made",
                           title = "Touches vs. Key Passes")
touchVsKP
```



From this we can notice that players who have a greater percent of their team's touches tend to use these touches wisely in making key passes. I additionally colored the points by team in case a viewer is curious as to how the trend follows from team to team. However, there are many so many different teams that it's difficult to isolate some colors from others. It would be nice to be able to isolate one or two teams at a time.

We could simply make a graph for any combination we may want, but that would take time and be a repetitive process. Instead, we can take advantage of the ggplotly function in the Plotly R package.

The simplest way to take advantage of the Plotly package is to call its function ggplotly() on any ggplot graph you have made, as follows:

```
#converting the ggplot into a plotly interactive graph
ggplotly(touchVsKP)
```

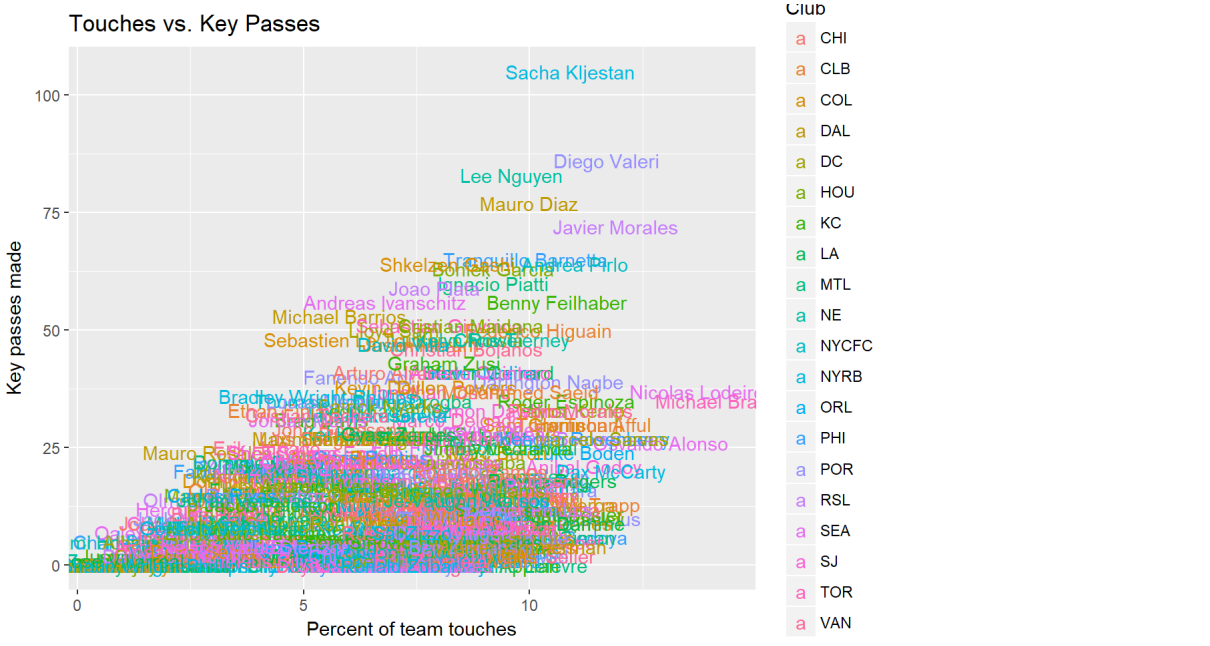
```
## We recommend that you use the dev version of ggplot2 with `ggplotly()`
## Install it with: `devtools::install_github('hadley/ggplot2')`
```

This first step towards mastering interactive graphs could not be easier. With one simple line of code we take our static graph and add the ability to hover over data points for more information, and isolate, hide, or compare the data of specific teams (along with several other functions). Here are a couple things to try out:

- Click a team in the legend to hide its associated points
- Double-click a team in the legend to isolate its associated points
- Hover over points to retrieve information about them
- Select a portion of the plot to zoom into by clicking and dragging

Now that you've gotten a taste of the power of interactive graphs, you won't want to stop just yet. While that last graph was definitely an improvement there is still more that we can do! Wouldn't it be nice to know which players made up which points? Returning back to our trusty friend Ggplot2, we can do this by adding a geom_label layer like so:

```
#building a ggplot with labels for points
graphLabels <- ggplot(players, aes(x = players$touch, y = players$KP)) +
               geom_text(aes(label = name, col = Club)) +
               labs(x = "Percent of team touches", y = "Key passes made",
                   title = "Touches vs. Key Passes")
graphLabels
```



Well that didn't turn out so well… There are too many data points for this type of labeling to be useful - most of the labels are unreadable. Perhaps if I apply the Plotly function ggplotly() all will be magically solved.

```
#converting the ggplot into a plotly interactive graph
ggplotly(graphLabels)
```

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

Although there is no visual improvement, we can now hover over a point and get the player's name information along with what we had previously. Ideally we could keep our scatterplot with points and have this name information appear upon hovering over a point. In fact, Plotly allows us to do just this!

```
#ggplot with information we want to appear upon hovering over points
graphHoverInfo <- ggplot(players, aes(x = players$touch, y = players$KP,
                                 label = name)) +
                         geom_point(aes(col = Club)) +
                         labs(x = "Percent of team touches",
                              y = "Key passes made",
                              title = "Touches vs. Key Passes")

#converting the ggplot into a plotly interactive graph
ggplotly(graphHoverInfo)
```

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

As you can see, this simply requires us to edit the ggplot object and allow the variable *name* to be accessible by ggplotly() as a label. the Plotly package takes care of the rest!

Suppose we wanted to only show the name of the player upon hovering over the point; we could implement the following code:

```
#plotly graph that shows player name's upon hovering over points
ggplotly(graphHoverInfo, tooltip = "name")
```

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

We can use the tooltip option to choose any piece of data we want to show - as long as it has been made available by including it in the label aesthetic of the ggplot object.

Our final graph here on the surface does not look much different than the ggplot object we started with. However, the added functionality from converting it into a Plotly object makes it much more useful for analysis and allows us to avoid staring at rows and rows of data. There are many, many ways to take advantage of the Plotly package. My research and ideas came out of this introductory book, which hightlights some of the key features of Plotly. But this is just the beginning - Plotly is filled with useful tricks, all of which help you take data visualization to the next level - I encourage you to explore!

References

- Data: http://www.americansocceranalysis.com/
- Karl Broman Slides: https://www.biostat.wisc.edu/~kbroman/talks/InteractiveGraphs/
- Blog on Dynamic Graphs: https://www.liquidlight.co.uk/blog/article/bring-data-to-life-with-dynamic-graphs-charts/
- Plotly references: https://plot.ly/r/
- Intro to Plotly book: https://plotly-book.cpsievert.me/index.html
- Blog discussing uses for Plotly: http://www.r-graph-gallery.com/2017/06/07/get-the-best-from-ggplotly/
- Image: http://www.northeastern.edu/levelblog/2016/05/13/how-much-data-produced-every-day/