Post 2

Timothy Hsu

December 3rd, 2017

Data Visualization

Data visualization is an important component of analyzing data. For those who are not strong on the technical end, it is hard to read and manipulate raw data. Instead, it is much easier to understand data when it is presented in a *visual* way. Thus, it is important that we, as coders, have the ability to not only manipulate data, but also put it in a way that everyone can understand. Luckily, instead of having to implement our own library, a lot of open-source libraries already exist for developers to use such as `ggplot2`.

Ggplot2

Ggplot2 is a public graphing library that makes it easier to make graphs such as barplots, scatterplots, density plots, and so much more. However, some of the plots that ggplot2 offers are not often used (e.g. `geom_tile`, `geom_rug`, etc). The most used plots are `geom_histogram`, `geom_point`, and `geom_density`.

How to use ggplot2

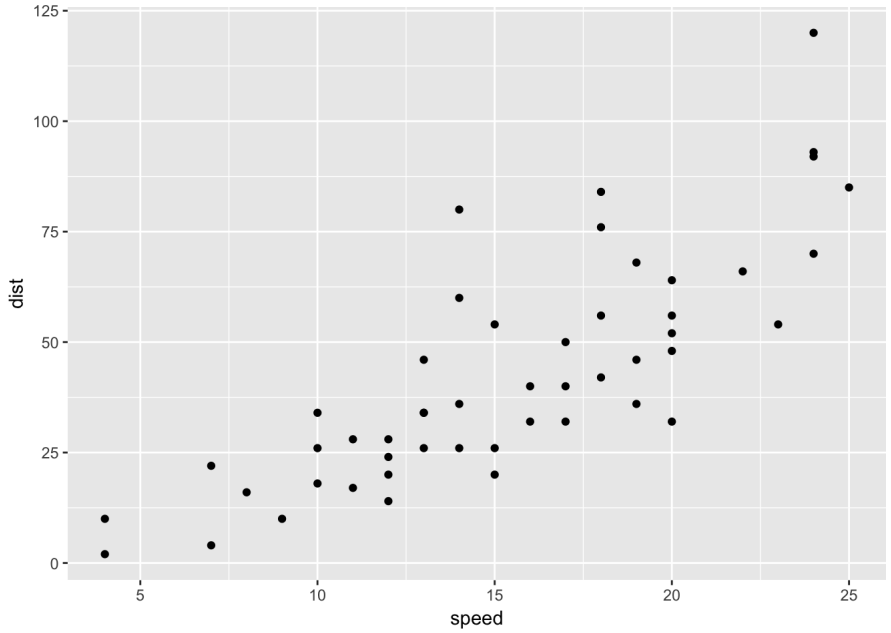
To use ggplot2, you need to first download the package by running `install.packages("ggplot2")`. Afterwards, load the library by executing

```
library(ggplot2)
```

After downloading and importing the library, you will now have access to all functions defined inside the `ggplot2` library. Creating plots is the interesting part. As the coder, you have free reign to create whatever plot you so choose but you must keep in mind what data the plots will need. For example, lets build a simple scatterplot.

To build a scatterplot, we will need information for both the y and the x axis.

```
ggplot(cars, aes(x=speed, y=dist)) + geom_point()
```

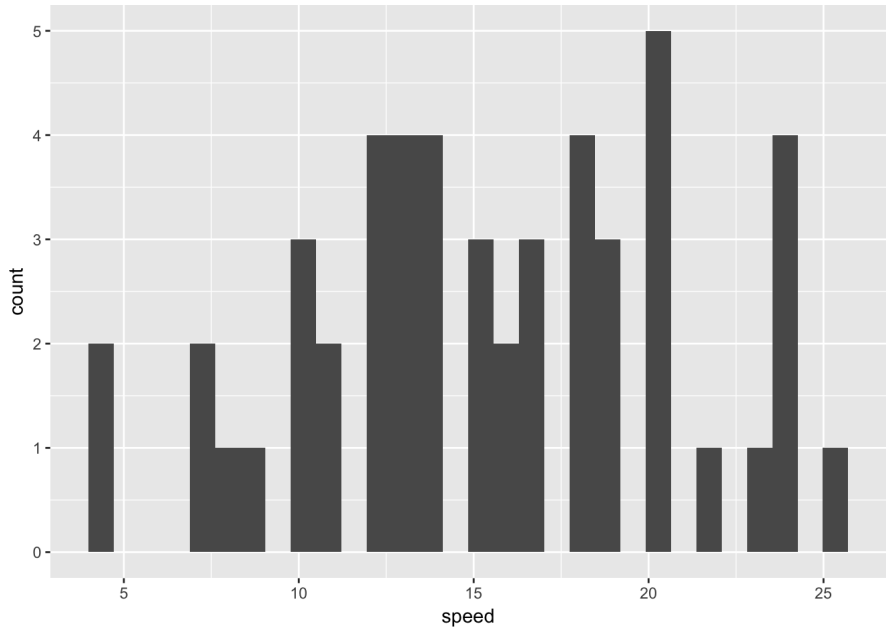


This chunk of code will plot a *scatterplot* using the data from the data frame `cars` using the speed of the car as the x-axis and the distance as the y-axis. However, not all plots will require 2 data points to plot a graph. Let's look at making a histogram.

Changing from a scatterplot to a histogram is relatively simple. To change the plot type, change `geom_point` to `geom_histogram`. However, to preserve the nature of a histogram, you can only have *one* axis. In fact, if you try to pass in two sets of data, your console will complain and say that it can't accept two sets of data. Thus, we will want to use just one set of data and in this example, we'll use `cars$speed`.

```
ggplot(cars, aes(x=speed)) + geom_histogram()
```

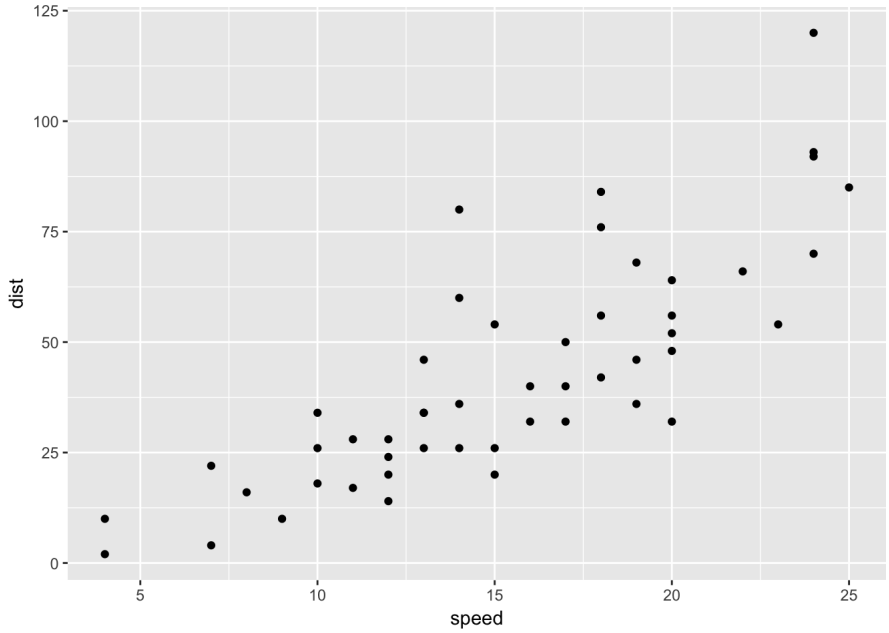
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Aesthetics

A large part of data visualization is making it easy to read. But what makes a visual “easy” to read? This ultimately boils down to be the *aesthetics* of the visual. Is it easy to tell what points of data are the main focus? Is it easy to tell what each point of data refers to? To answer these questions, I will be using the same scatterplot example from above.

Consider this graph:



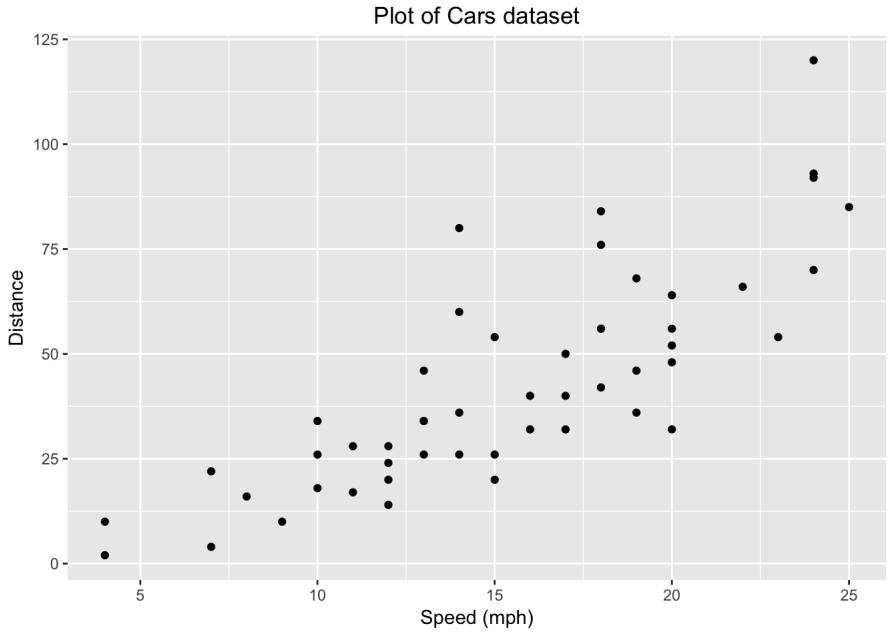
What does this graph tell you about the data if you didn’t have access to the code? Well we know that the x-axis is tracking speed and the y-axis is tracking dist… We really don’t know much here. Some questions a viewer might have are

- What data set are we looking at?
- What do these points represent?

Clearly, this graph doesn’t tell the viewer much. Thus, in order to improve the user experience, we need to add more details to make things more clear. Some things we may want to add are

- Giving the graph a title
- Detail the x and y axis a bit more

Now consider this graph:



This new graph now gives the viewers a little bit more context of what the graph is trying to describe. As the vewier, we now know that the graph is depicting a dataset about cars. This tells us that these points are likely cars that are being plotted. We also know that the x-axis is tracking the speed in mph. We can also make this look a bit more fancier by changing the colors of the points, changing the shapes, and etc but these are to the designer's taste.

Taking it a step further.

Ggplot is a powerful tool but it can be made even stronger through the use of extensions. For this example, we're going to look at utilizng the `ggedit` extension. To download `ggedit`, run

```
install.packages('ggedit')
# https://github.com/metrumresearchgroup/ggedit
```

Now you might be wondering what `ggedit` does. This extension behaves somewhat similar to what we have learned in class in regards to shiny apps. `ggedit` gives users the ability to *dynamically* change inputs to ggplot through means of the user interface (UI). This is a powerful tool because `ggedit` essentially requires the developer to write less code than needed for a shiny app in order to achieve the same behavior.

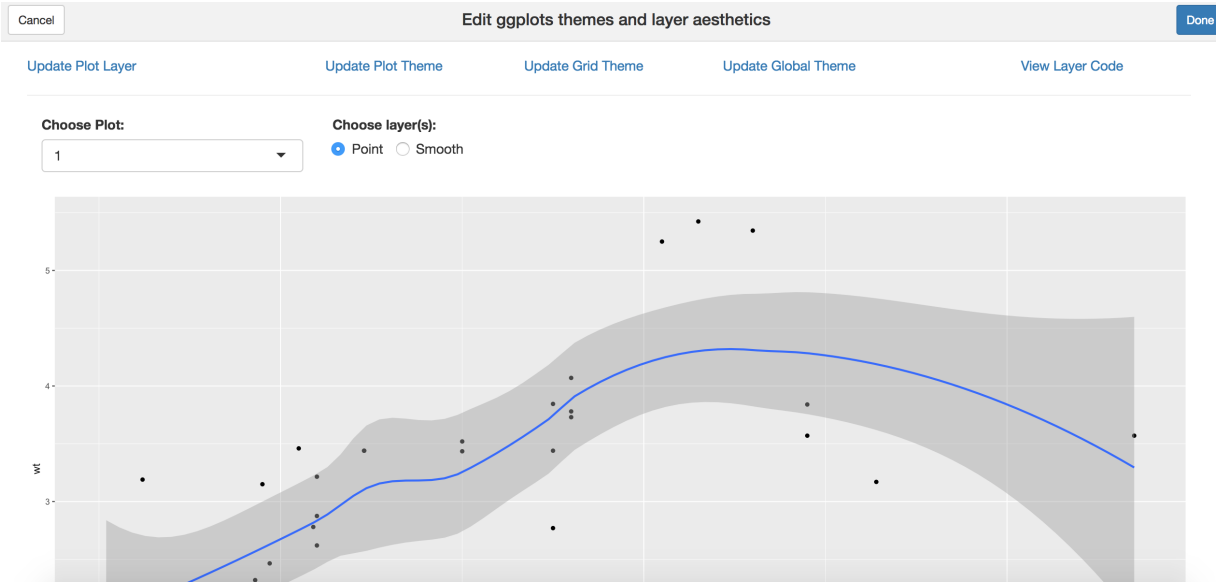
Let's take a look at this example:

```
library('ggedit')
library(ggplot2)
p <- ggplot(mtcars, aes(x = hp, y = wt)) + geom_point() + geom_smooth()
p2 <- ggedit(p)
```

The first three lines should seem pretty familiar. What we're interested in is the fourth line

```
p2 <- ggedit(p)
```

Here, we're telling `ggedit` to operate on our graph `p` and when we execute it, we'll get an operatable app. If you execute the code, you will get something that looks like this:



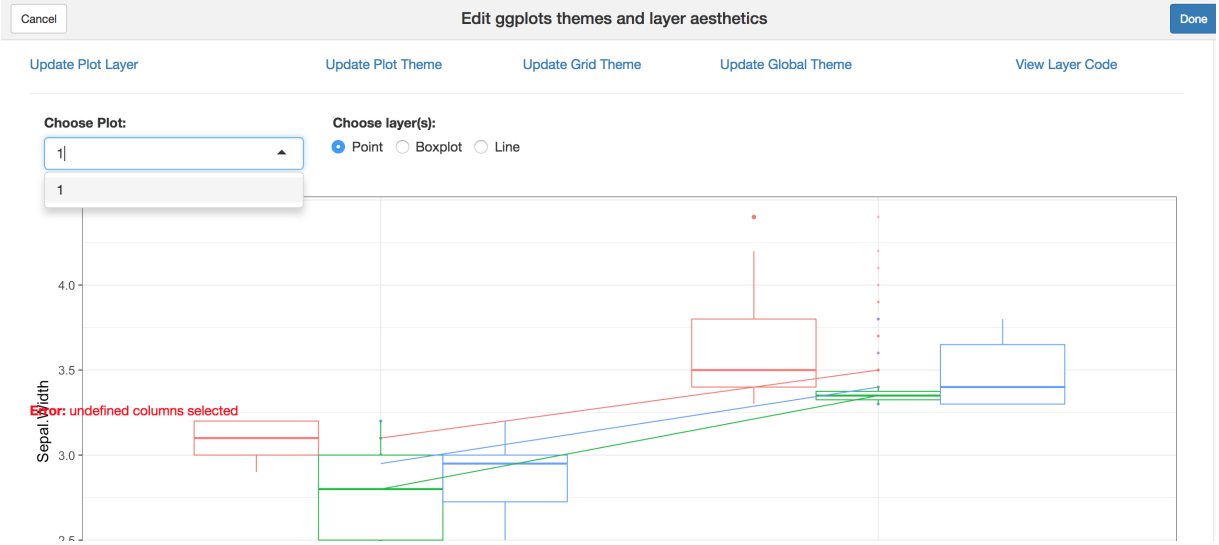
Already, you can see that a lot of functinality is readily available including the ability to update the theme of your graph, choose different layers, vary the plot, and more. However, while a lot of these buttons exist, not all of it is in working condition with our sample code. Let's delve a little

deeper into how to use `ggedit`.

```
plot <-
  iris%>%
  ggplot(aes(x=cut(Sepal.Width,2),y=Sepal.Width)) +
  aes(colour=Species,group=Species)+
  geom_point(alpha = 0.5,shape = 16, size = 1) +
  geom_boxplot(aes(group=NULL), varwidth = FALSE,notch = FALSE, show.legend = TRUE)+
  stat_summary(fun.y = 'median', geom = "line")+
  theme_bw(base_size = 16) +
  theme(legend.position = "bottom",
        legend.box = "vertical", legend.direction = "horizontal",
        axis.text.x = ggplot2::element_text(angle = 90,hjust = 1, vjust = 0.5),
        legend.title = element_blank())

ggedit(plot)
```

Using this graph, we are able to produce a more sophisticated app.



So how are we producing this? Well looking at the code, we can see that we're essentially using using `ggplot` as normal while specifying what types of graphs we would like to produce. Specifically, we're creating a boxplot, scatterplot, and we're also adding a theme.

Conclusion

As we have seen, visual representation is a powerful tool that allows for viewers to better understand the data presented to them. With valuable libraries such as `ggplot2`, it is fairly easy for developers to make pretty graphs and for more docuementation on `ggplot2`, you can checkout the official documentation at http://ggplot2.tidyverse.org/reference/.

Moreover, we have also seen now that a lot of powerful extensions built on top of popular libraries exist as well. These tools greatly simply a developer's life by reducing the amount of code needed to achieve the same result (as seen by our example `ggedit`).