SESUG Paper 210-2023

Translating Common Data Visualizations from PROC SGPLOT to ggplot2

Austin R. Brown, Ph.D., Kennesaw State University

ABSTRACT

Data visualizations are a powerful way of communicating information in a brief and aesthetically pleasing manner. Most every statistical software has some means of generating visualizations with some being easier to work with or modify than others. In SAS, `PROC SGPLOT` is a very useful procedure for generating visualizations. However, the modern data scientist is often required to be well-versed in multiple programming languages to meet the needs and demands of their organizations. For instance, R is a very popular statistical programming language that many employers want or need their data science teams to be familiar with. Thus, in this paper, it will be shown how common univariate and bivariate visualizations generated with `PROC SGPLOT` can be similarly generated with the ggplot2 package in R. Examples are given and thoroughly explained.

INTRODUCTION

Data visualization is becoming a more and more important tool in a data scientist's toolbox. And it makes sense as to why! When we consider how complex raw data can be in addition to the challenges presented in appropriately and contextually understanding and communicating mathematical and statistical computer output, it is understandable that a large proportion of the global population may not have experience in dealing with such things. So if we are to tell a story with our data, which is what we do as data scientists, then being able to effectively tell said story to a broad audience in an accessible manner seems of great importance.

Of course, the choice of data visualization is a function of the data type being analyzed. And moreover, the specific aesthetic elements of a visualization should be carefully chosen to well communicate the intended message. However, in modern day, the breadth of softwares which can generate data visualizations is numerous, with options including `PROC SGPLOT` in SAS, mathplotlib in Python, Tableau, or even Microsoft Excel. One very powerful and robust data visualization package is called "ggplot2" and is accessed through the open source statistical software package R. This useful package has been developed and utilized to create a wide variety of aesthetically pleasing visualizations which tell data stories. While similar visualizations can be generated with `PROC SGPLOT`, it may be useful for the modern data scientist to also know how to create them using ggplot2 in R.

The goal of this paper is demonstrate how common data visualizations, like bar charts, scatterplots, and boxplots, can be generated in `PROC SGPLOT` and then translated into ggplot2.

BAR CHARTS: COMMUNICATING "HOW MANY"

A common data story we have to tell is "how many." For instance, using the NYC Flights 2013 data (contained in the nycflights13 package in R), suppose I want to know how many flights from each of the three New York City airports (LGA, JFK, and EWR) had a destination of Charlotte, North Carolina (CLT). One way of communicating this data story is by using a bar chart. To do this with PROC SGPLOT, we have a few different options. First, we can generate a vertical bar chart, as generated by the below code and shown in Figure 1:

```
/* Generate a Vertical Bar Chart for flights departing to CLT */

/* Subset CLT flights */

proc sql noprint;
create table clt as
select origin,count(dest) as Frequency
from viz.flights
where dest = "CLT"
group by origin;
quit;

/* Generate PROC SGPLOT */

proc sgplot data=clt;
vbar origin/response=Frequency;
xaxis label="Airport of Origin";
yaxis label="Number of Departing Flights";
title "Number of Flights Departing to CLT";
title2 "From NYC Airports 2013";
run;
title;
title2;
```

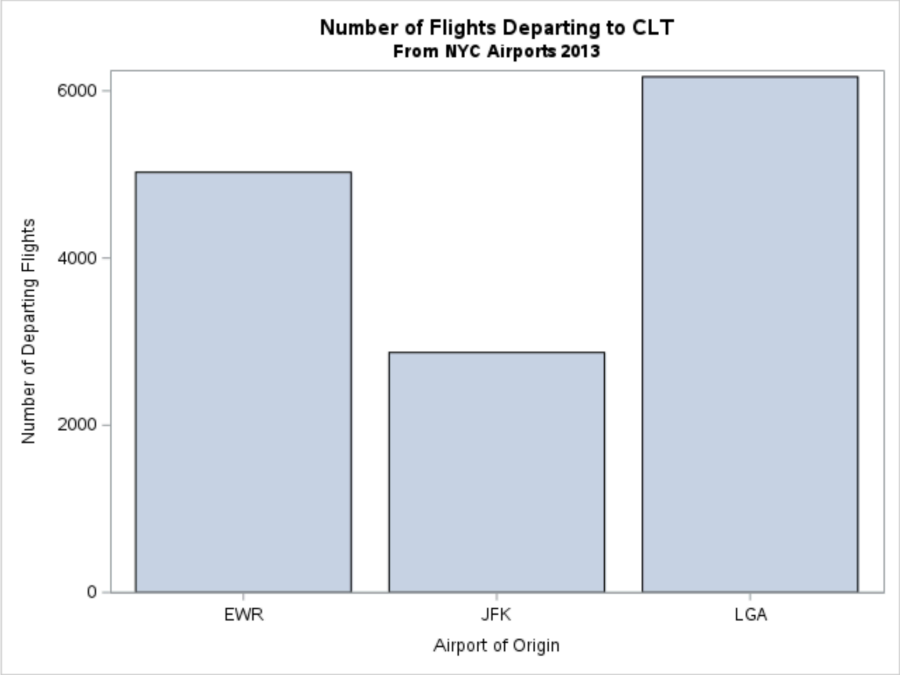

Figure 1. Vertical Bar Chart Generated with PROC SGPLOT

Now, how do we go about generating a comparable vertical bar chart using ggplot2 in R? We go through the exact same procedures, just in a different programming language. After I've read in the data (called "flights"), I subset into an aggregated data table, as I did with PROC SQL above, but by using a package for data wrangling called dplyr, which has a comparable syntax to SQL queries:

```
clt_dest <- flights |>
  dplyr::filter(dest == 'CLT') |>
  dplyr::group_by(origin) |>
  dplyr::count() |>
  dplyr::rename(Frequency = n)
```

Now to generate the visualization using ggplot2, we use:

```
## Vertical Bars ##

library(ggplot2)

clt_dest |>
  ggplot(aes(x=origin,y=Frequency)) +
  geom_bar(stat='identity') +
  labs(x = "Airport of Origin",
       y = "Number of Departing Flights",
       title = "Number of Flights Departing to CLT",
       subtitle = "From NYC Airports 2013") +
  theme_classic() +
  theme(plot.title=element_text(hjust=0.5),
        plot.subtitle=element_text(hjust=0.5))
```

The way ggplot2 works is by building visualizations in layers, much like painting a picture. Each visualization begins with a data source, which is clt_dest in our case. Then, we build our "canvas" using what I refer to as the "global ggplot statement." Here, we specify, among other things, what variable goes on the x-axis (origin here), and which variable goes on the y-axis (Frequency in our case). Then to add "layers" to our data painting, we use the "+" operator. As `PROC SGPLOT` has different statements to generate different visualizations (e.g., `vbar, series`, etc.), ggplot2 has different geom prefixed functions. For instance, to generate a bar chart in this case we use geom_bar. The argument stat='identity' within the function performs the same functionality as `response=Frequency` within the `vbar` statements. The labs function is self-explanatory. The addition of the theme_classic function changes the overall aesthetic of the visualization and then the final theme function center-justifies the title and subtitle, respectively. The resulting visualization is given by Figure 2.

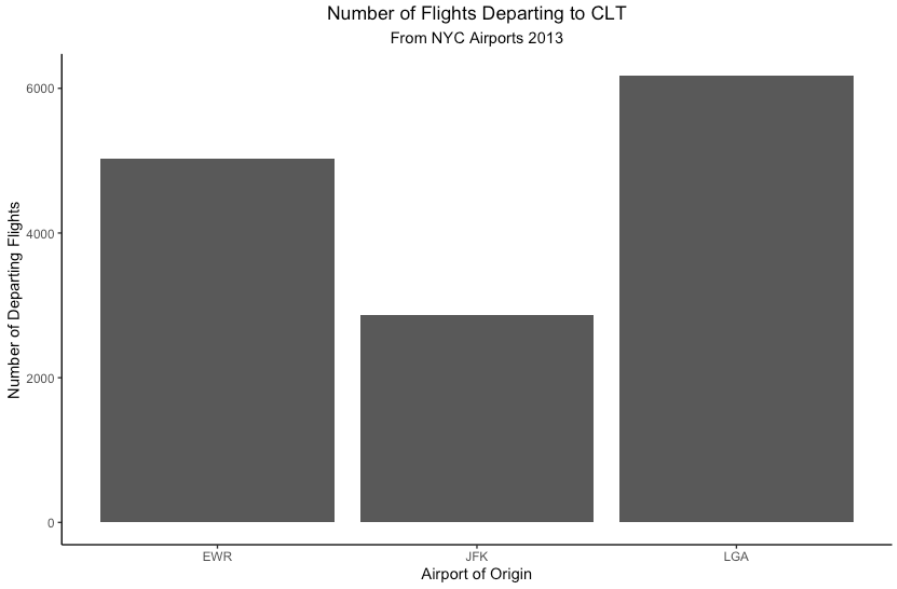

Figure 2. Vertical Bar Chart Generated with ggplot2

CREATING HORIZONTAL, ORDERED BAR CHARTS

In some instances, it may be more effective to communicate "how many" using a horizontal bar chart rather than a vertical bar chart. In the case of `PROC SGPLOT`, this is a matter of changing `vbar` to `hbar`. For ggplot2, it is simply switching the variables in the global ggplot function. But with this type of visualization, it is sometimes desirable to order the bars in descending order. So in this case, we may want LGA to be the first or top bar in the horizontal bar chart, EWR would be second, and JFK would be the bottom or last bar. To do this using `PROC SGPLOT`, we can use the below code which generates Figure 3:

```
/* Generating a horizontal, ordered bar chart */
proc sgplot data=clt;
hbar origin/response=Frequency
            categoryorder=respdesc;
yaxis label="Airport of Origin";
xaxis label="Number of Departing Flights";
title "Number of Flights Departing to CLT";
title2 "From NYC Airports 2013";
run;
title;
title2;
```

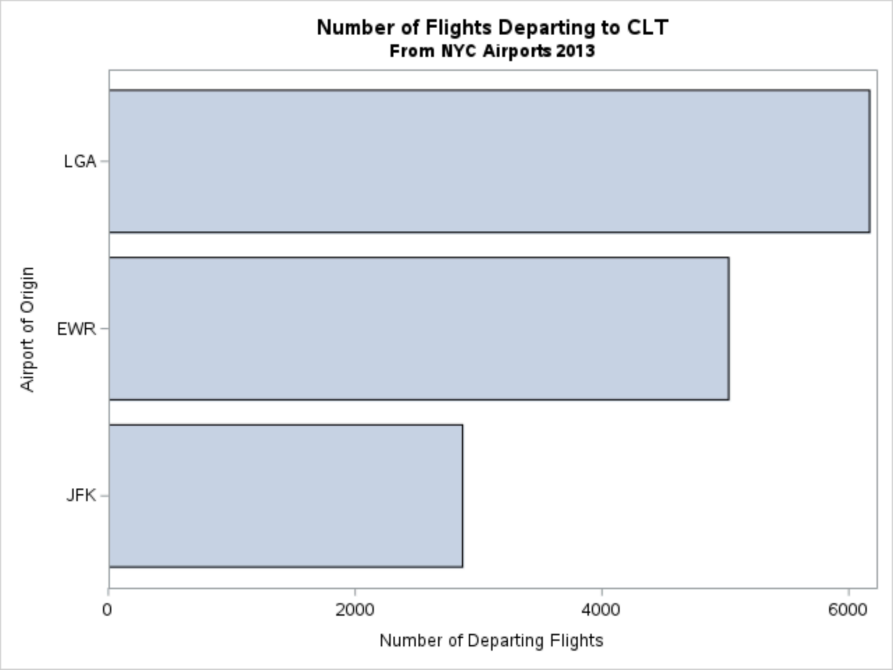

Figure 3. Horizontal, Ordered Bar Chart Generated with PROC SGPLOT

Notice here that the only major additional piece of code required to order the bars was `categoryorder=respdesc`. This is similar to the required changes to the ggplot2 code. In that case, we make use of the reorder function to reorder the categories on the y-axis by frequency. The code used to generate Figure 4 is given below:

```
## Converting to a horizontal, ordered bar chart ##

clt_dest |>
ggplot(aes(x=Frequency,y=reorder(origin,Frequency)))
geom_bar(stat='identity') +
labs(x = "Airport of Origin",
     y = "Number of Departing Flights",
     title = "Number of Flights Departing to CLT",
     subtitle = "From NYC Airports 2013") +
theme_classic() +
theme(plot.title=element_text(hjust=0.5),
      plot.subtitle=element_text(hjust=0.5))
```

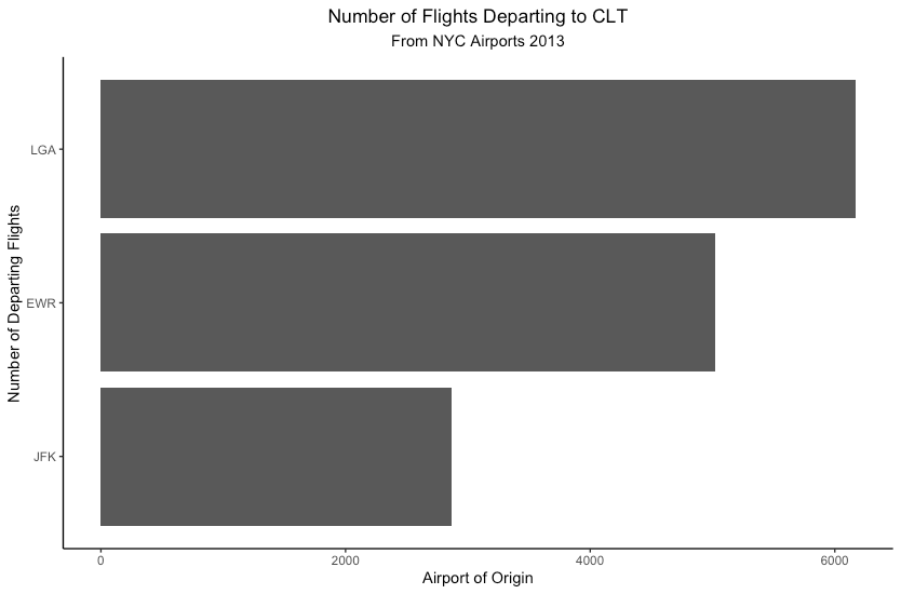

Figure 4. Horizontal, Ordered Bar Chart Generated with ggplot2

SCATTERPLOTS: COMMUNICATING ASSOCIATION

It is also often desirable to communicate how two quantitative variables (say departure delay and arrival delay for flights with a destination of Charlotte, NC) relate to each other. Obviously, we can do this quantitatively using the correlation coefficient, but we can also do so using a scatterplot. As we know, a scatterplot converts pairs of data points into x and y coordinates to plot on a Cartesian plane. In doing so, we can visually assess the nature of the association between our two quantitative variables. To do so in `PROC SGPLOT` we can use the below code which generates Figure 5.

```
/* Subset Data */

proc sql noprint;
create table clt2 as
select dep_delay, arr_delay, origin,
dest from viz.flights
where dest="CLT";
quit;
```

```
/* Generate Scatterplot */

proc sgplot data=clt2;
scatter x=dep_delay y=arr_delay;
xaxis label="Departure Delay (in minutes)";
yaxis label="Arrival Delay (in minutes)";
title "Association Between Departure and Arrival Delays";
title2 "For Flights Departing to CLT from NYC Airports";
run;
title;
title2;
```

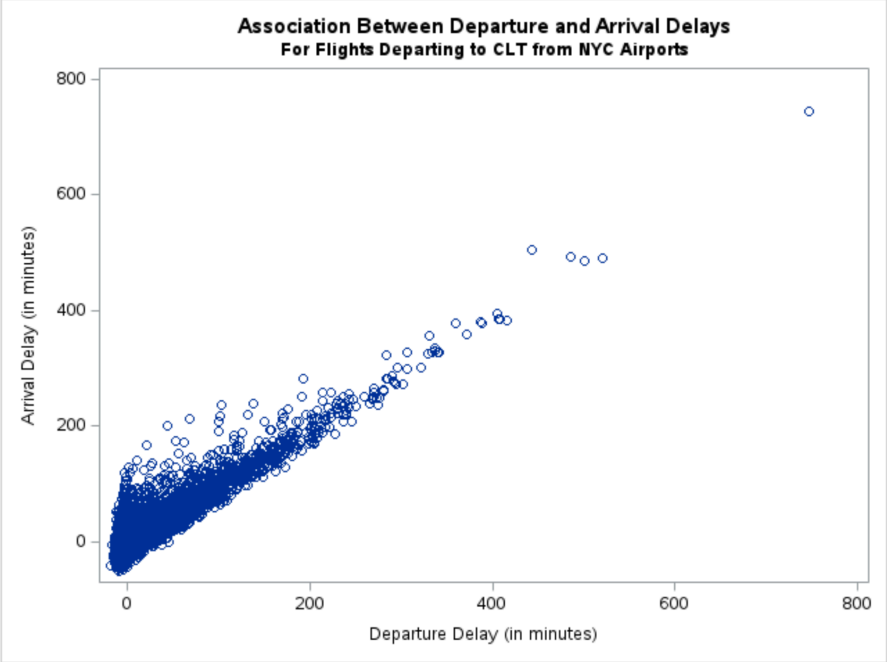

Figure 5. Scatterplot Generated with PROC SGPLOT

Here, the `PROC SGPLOT` syntax is relatively straightforward. To generate a scatterplot, we must invoke the `scatter` statement and then specify the variable which goes on the x-axis and which goes on the y-axis. The ggplot2 syntax is incredibly similar. Here, we would use the geom_point function, which is the ggplot2 analogue to the scatter statement. The below code generates Figure 6, which is a very similar visualization to Figure 5:

```
## Scatterplot Comparing Dep_Delay and Arr_Delay ##

## Subset Data ##
clt2 <- flights |>
  dplyr::select(dep_delay, arr_delay, origin,dest) |>
  dplyr::filter(dest=="CLT")
```

```
## Generate Scatterplot ##
clt2 |>
  ggplot(aes(x=dep_delay,y=arr_delay)) +
  geom_point() +
  labs(x="Departure Delay (in minutes)",
       y="Arrival Delay (in minutes)",
       title="Association Between Departure and Arrival Delays",
       subtitle="For Flights Departing to CLT from NYC Airports") +
  theme_classic() +
  theme(plot.title=element_text(hjust=0.5),
        plot.subtitle=element_text(hjust=0.5))
```

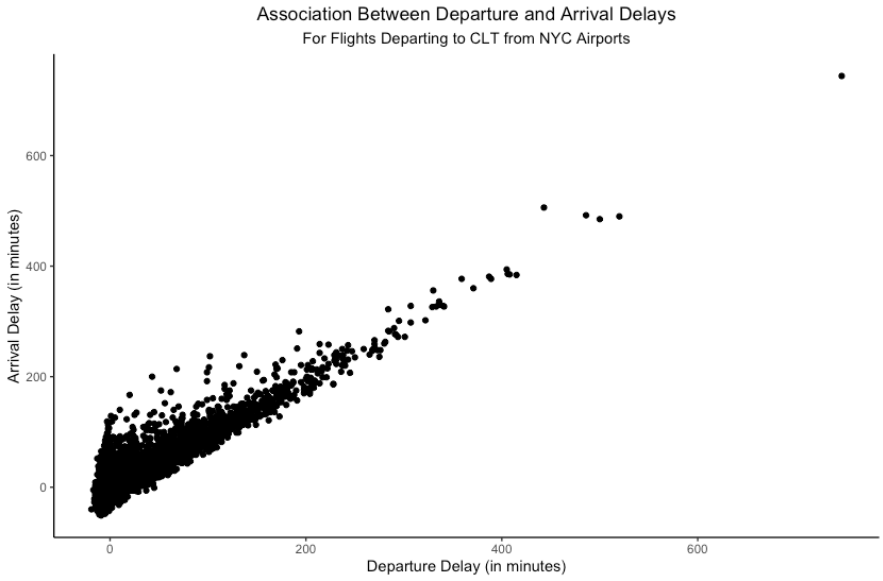

Figure 6. Scatterplot Generated with ggplot2

ADDING A REGRESSION LINE TO SCATTERPLOTS

Since it is common to use simple linear regression to describe the relationship between two quantitative variables, it may be useful to plot the fitted regression line on our scatterplot, overlaying the points. To do so using both `PROC SGPLOT` and ggplot2 is quite simple. In the case of `PROC SGPLOT`, we simply add a `reg` statement which has the same syntax as the scatter statement.

```
/* Add a Simple Linear Regression Line */

proc sgplot data=clt2;
scatter x=dep_delay y=arr_delay;
reg x=dep_delay y=arr_delay;
xaxis label="Departure Delay (in minutes)";
yaxis label="Arrival Delay (in minutes)";
title "Association Between Departure and Arrival Delays";
title2 "For Flights Departing to CLT from NYC Airports";
run;
title;
title2;
```

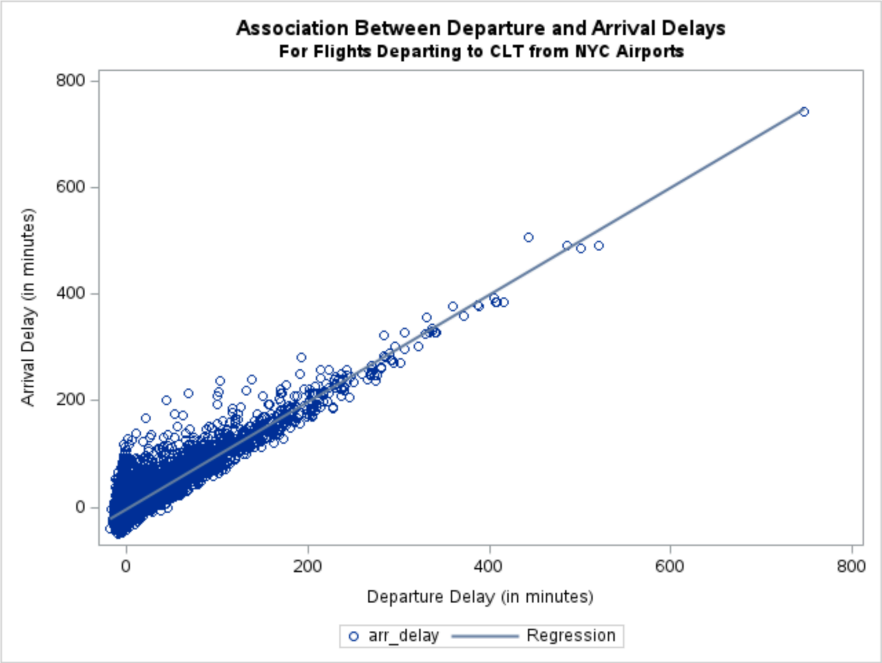

Figure 7. Scatterplot with Simple Linear Regression Line Generated with PROC SGPLOT

For ggplot2, we add another geom called geom_smooth. Since a simple linear regression line is not the only type of regression line we can fit to our data, we have to specify that we want a “linear model” or “lm” for short. SE=F suppresses the standard errors from rendering on the plot.

```
## Adding a Simple Linear Regression Line ##

clt2 |>
  ggplot(aes(x=dep_delay,y=arr_delay)) +
  geom_point() +
  geom_smooth(method='lm',se=F) +
  labs(x="Departure Delay (in minutes)",
       y="Arrival Delay (in minutes)",
       title="Association Between Departure and Arrival Delays",
       subtitle="For Flights Departing to CLT from NYC Airports") +
  theme_classic() +
  theme(plot.title=element_text(hjust=0.5),
        plot.subtitle=element_text(hjust=0.5))
```

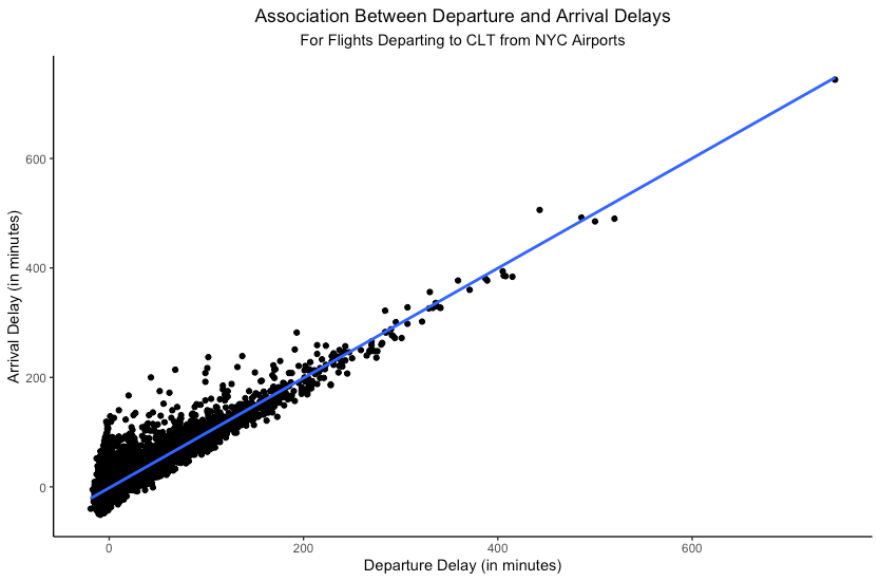

Figure 8. Scatterplot with Simple Linear Regression Line Generated with ggplot2

BOXPLOTS: COMMUNICATING DIFFERENCES IN VARIATION AND LOCATION

One of the beautiful aspects of a boxplot (sometimes called a "box and whiskers plot") is that we are able to visually identify several pieces of information in a glance. Remember, a boxplot is a visual representation of the empirical quartiles (the box) as well as the extreme values (the "whiskers"). For instance, suppose I wanted to use the `SASHELP.CARS` data table to compare the variation and location of highway miles per gallon by the type of vechile (e.g., sedan, SUV, etc.) A boxplot may be an appropriate way of doing so. Using `PROC SGPLOT`, we can use the following code to generate Figure 9:

```
/* Boxplot to Compare Location and Variation
/* of Highway MPG between Car Type */

proc sgplot data=sashelp.cars;
vbox mpg_highway/group=type;
xaxis label="Car Type";
yaxis label="Highway Miles Per Gallon";
title "Highway Miles Per Gallon by Car Type";
run;
title;
```

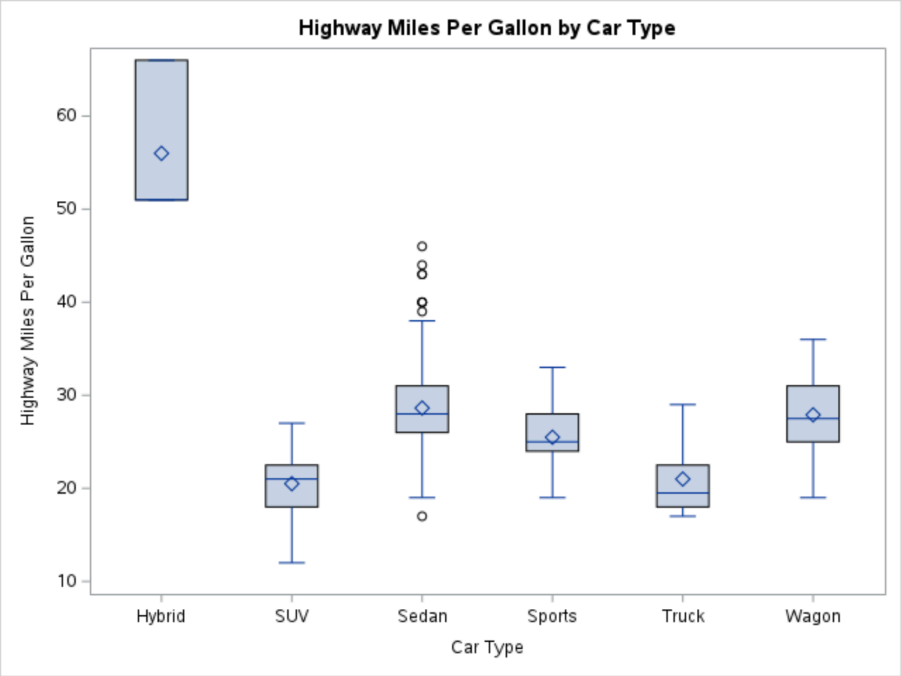

Figure 9. Boxplots Generated with PROC SGPLOT

To create the same visualization in ggplot2, I can use the below code to generate Figure 10:

```
## Generating a Boxplot ##

## Read in SAS Help Cars ##

cars <- readxl::read_xlsx("SAS Help Cars.xlsx")

cars |>
  ggplot(aes(x=Type,y=MPG_Highway)) +
  geom_boxplot() +
  labs(x="Type",
       y="Highway Miles Per Gallon",
       title="Highway Miles Per Gallon by Car Type") +
  theme_classic() +
  theme(plot.title=element_text(hjust=0.5),
        plot.subtitle=element_text(hjust=0.5))
```

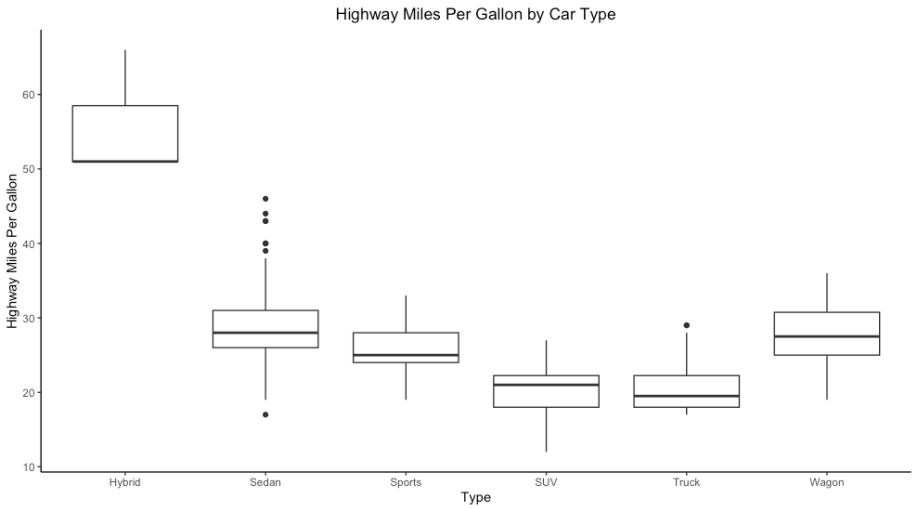

Figure 10. Boxplots Generated with ggplot2

CONCLUSION

Data visualization is a useful and powerful tool modern data scientists have in their data science toolbox. As mentioned, effective data visualizations can serve as quick and accessible mediums through which a data story can be told. Because of the variety of statistical software programs which exist, it is useful for a data scientist to be familiar with multiple mediums through which data visualizations can be generated, including `PROC SGPLOT` in SAS, matplotlib in Python, and of course, ggplot2 in R. In this paper, it was demonstrated how commonly used data visualizations can be generated in `PROC SGPLOT` and how they can similarly be generated using the ggplot2 package in R. Of course, this paper serves as an introduction to ggplot2 functionality. There is much, much more which can be done and modified, even among the existing figures presented here. As one delves deeper into both the functionality of ggplot2 as well as `PROC SGPLOT`, it becomes apparent that some modifications are less verbose to implement than others between the two programs. However, knowing which tools are most useful and simplest to implement in a given situation is useful for any data scientist to know.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Austin R. Brown, Ph.D.
School of Data Science and Analytics
Kennesaw State University
abrow708@kennesaw.edu