George Schoeffel
October 31, 2019
PPOL 563

Data Visualization Tutorial: Creating Spider Plots

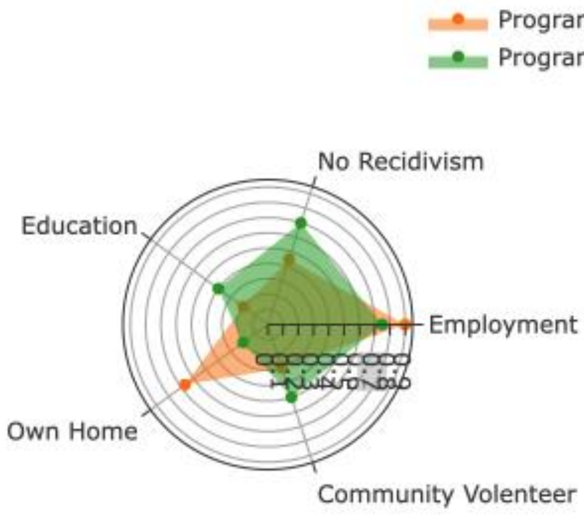
As a public policy student, I am always asking myself questions of how: How are we able to best fight poverty? How much does the economy have to grow in order to guarantee a president's reelection? How do we best combat a recession using the financial tools the government is able to control? The heart of these questions lies in the fact that a public policy student is often trying to find causal relationships in the data. Policy students want to know if raising the minimum wage, improving education, or creating more resources and programs is most effective at helping individuals living below the poverty line. Because of this, we often create graphs that demonstrate the relationships between two variables. For example, when measuring life expectancy versus individual GDP for a country, we may observe somewhat of a linear pattern, possibly indicating a causal relationship between the two variables. As a policy student, we like these graphs because we are able to create a story from it: improving the GDP for each individual could help increase life expectancy. Whether or not that is true is an entirely different conversation. But to begin having that conversation we need graphical tools like scatterplots, line plots, and histograms.

As an analyst for a professional soccer team, I often find myself needing to put on a different hat: a predictive hat. Although it would be untrue to say that we never care about the question of how (e.g. how one player is able to score ten more goals than another for the course of a season), our top interest is in making accurate predictions regardless of understanding why they happen; we want to know who will score more goals over the course of the season. This is why creating visualizations at Wycombe can often be so challenging as a public policy student. To this point I have created some dot charts that show goals scored on one axis and goals conceded on the other with the size of the dot being related to the ranking of each team, but that only shows us our past progress and is not very useful for the next game. One visualization that I have found to be useful that I have created is the spider chart (sometimes called the radar chart). This chart is useful because it serves as a valuable tool to either quickly and efficiently measure two individuals or see the advantages and disadvantages of different observations, while not necessarily visually demonstrating any form of relationship between these values and observations.

Throughout the rest of this document I will run the reader through the process of making a simple spider chart. Before moving to that stage however, I think it is useful to have a brief discussion regarding things to keep in mind when using spider charts. The first thing that I will mention is that it is helpful to scale variables in a spider chart. Using soccer data, there are a wide variety of variables where the domain can range from 0 to 32 (in the case of goals scored in the 2017 season), to 0 to 3240 (for minutes played in the 2017 season). For this reason, it makes sense to scale all variables for a spider plot, so that you do not need to have every axis be a different unit of measure. For the spider plots that I created, I scaled the data by dividing each observation by the maximum possible value of the domain. Therefore, if an individual scored 24 goals and played 1620 minutes (using the 32 goal max and the 3240 minute max), the values would be .75 and .50. Additionally, it's important to keep in mind which variables are used in a spider chart. For example, the soccer data that I will be showing has variables for goals, assists, points, points per match, dream teams, minutes played, and cost. For all of these except cost we desire a higher number. Having a higher cost, however, is not necessarily a positive thing. This is something to watch out for when using spider plots, as an individual may think it is better to have a higher cost player, when the reality is that teams would want a lower cost player. For variables like this, it may be important to rescale them in a way to account for this (in my data, I choose to use 1 minus scaled cost so that a higher value in the spider chart consists with a cheaper player).

Finally, I think it is important to consider when it is appropriate to use a spider plot. As described above, spider plots are not the best tools at describing causal relationships between two variables. Instead, they are useful at showing information about many variables from different observations. From a public policy perspective, a spider plot may be useful when considering two different programs to implement (see Figure 1 below).

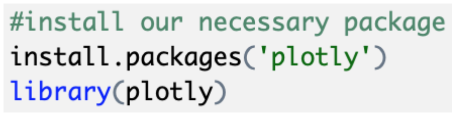
Figure 1: Success Rate of Prison Programs



For hypothetical example, let's imagine that we are choosing between two different programs used in a prison to be put into law as mandatory for all prisons in the United States. When trying to determine which program is desired, using a spider plot would allow us to see the benefits of both programs. If our top priorities as policymakers are not having individuals end up back in jail and continuing education after prison, then perhaps Program 2 may be more useful. However, if employment and seeing program attendees own houses is more important, than Program 1 may be more useful. I want to reinforce a point made above here again: it is important to note that these variables do not show any form of causal relationship between the data.

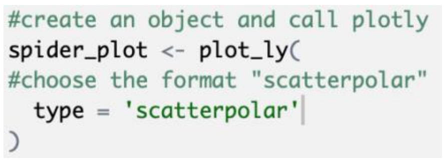
Creating Your Own Spider Plot Visualization

In order to build a spider plot, you will first need to have a cleaned and prepared data set. You will also need to have an updated version of R, since that is the language we will be using. Once that is all ready, we will need to install and call Plotly, which is a part of RShiny.

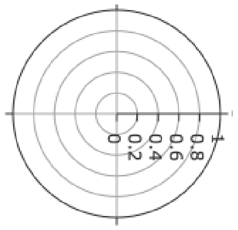


```
#install our necessary package
install.packages('plotly')
library(plotly)
```

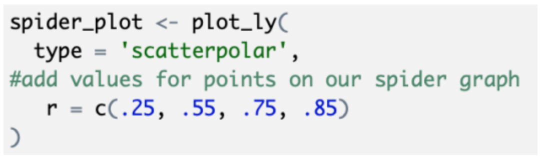
We choose to use Plotly because we believe it is easiest for a user to create and understand, although other packages are available. [1] Creating spider graphs in Plotly is a very layered process: we first specify general information that gives a template for our graphic, then we add data to put points on our graph, add axes to the plot to demonstrate the variables, and finally we add layers to connect the points to complete the spider plot. After installing Plotly and calling it using the library command seen above, we are ready to begin creating our spider plots. To create a spider plot with Plotly, we will first create a plot and specify that we want our plot in a "scatterpolar" format. We do this by using the following code, which results in the following output:



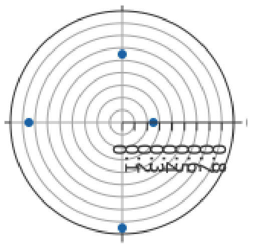
```
#create an object and call plotly
spider_plot <- plot_ly(
#choose the format "scatterpolar"
  type = 'scatterpolar'
)
```



This code give the computer a set of specific instructions on how to display information, preparing it to receive points on a polar coordinates graph as seen above. If we were to create a graphic at this current point, it would be an empty polar graph as we see above. This is because we have not given the computer any information as to where to add our points. In order to do that, we will add another layer to the plot to show our data. In this brief introduction example, we will concatenate some data. Now if we run the code again, we will see that we have our same polar graph, but now with some points added to the graph based on the values we created.



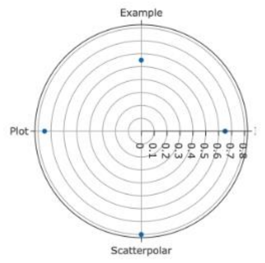
```
spider_plot <- plot_ly(
  type = 'scatterpolar',
#add values for points on our spider graph
   r = c(.25, .55, .75, .85)
)
```



Although having these points is a good start, a spider plot is not very valuable without proper labels. In order to add labels to the graph, we must add another line, which we call theta, which will contain our labels for our axes. Also note than when adding thetas, we must match the last observation with the first so that R understand that we are ending on the same point we are beginning with.

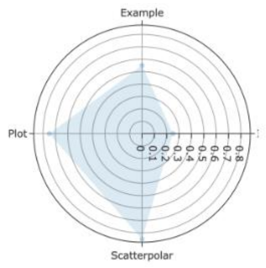
[1] Note that you can also use other packages like https://www.r-graph-gallery.com/142-basic-radar-chart.html

```
spider_plot <- plot_ly(
  type = 'scatterpolar',
#notice we had added the value .25 at the end to match the first value
  r = c(.25, .55, .75, .85, .25),
#add our value of theta and end with the same as our first value
  theta = c('Is a', 'This', 'Plot', 'Scatterpolar', 'Is a')
)
```



Now it is important to add more layers to the graphic. For this next step it is helpful to look at the full documentation in order to get the spider plot to be exactly to your liking.2 In our example that we give, we will connect the points on the graph and add a fill by using the "to fill" command, and we will also change the opacity of the fill by using the opacity command. As we can see below, we now have a functional spider plot! For more features, check the full documentation on Plotly's website.

```
spider_plot <- plot_ly(
  type = 'scatterpolar',
  r = c(.25, .55, .75, .85, .25),
  theta = c('Is a', 'This', 'Plot', 'Scatterpolar', 'Is a'),
#add our toself fill option to fill space between points
  fill = 'toself',
#add opacity for aesthetic purposes
  opacity = .25)
```



As described above, we know that one of the greatest benefits of a spider plot is the ability to quickly look at two or more different observations. We will give some examples using real data from the Premier League 2017 season with 4 different goal scoring forwards. To begin our spider plot, we use the same first line of code, indicating that we are calling Plotly and using a "scatterpolar" format. After doing this, we will put in our aesthetic options that are universal for all observations. For our example, this will include the "to fill" feature, and setting the opacity to .5.

```
spider_plot <- plot_ly(
  type = 'scatterpolar',
  fill = 'toself',
  opacity = .5
)
```

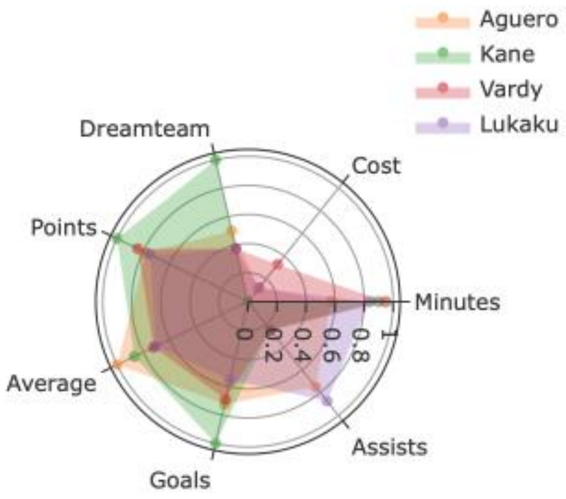
As a side step, we choose to create objects to hold the data we will be using later in our spider graphs. This step is not necessary as data can be put directly into the graphic as seen in the example with the concatenate function, but improves readability for examples.

2 For the documentation for spider plots, see https://plot.ly/r/reference/#scatterpolar

```
#Create our first object to hold data
Aguero = c(data[1,2], 1 - data[1,3], data[1,4], data[1,5], data[1,6], data[1,7], data[1,8], data[1,2])
#Create our second object to hold data
Kane = c(data[2,2], 1 - data[2,3], data[2,4], data[2,5], data[2,6], data[2,7], data[2,8], data[2,2])
#Create our third object to hold data
Lukaku = c(data[3,2], 1 - data[3,3], data[3,4], data[3,5], data[3,6], data[3,7], data[3,8], data[3,2])
#Create our fourth object to hold data
Vardy = c(data[4,2], 1 - data[4,3], data[4,4], data[4,5], data[4,6], data[4,7], data[4,8], data[4,2])
```

Since we are wanting to add multiple different observations to our graph, we want to layer each observation in piece by piece. We do this by using the piping, coded as %>%, in our graphic. We add each observation as a "trace", and each observation must include a list of values to plot (the value for r), a list of thetas (for our axes), and a label for each different observation. While adding more observations to the graph, we continue to use the piping function until we have all observations that we desire. The code for such a spider graph should look like something below, with the following output.

```
spider_plot <- plot_ly(
#use our code from above
  type = 'scatterpolar',
  fill = 'toself',
  opacity = .5
) %>%
#Add a trace for our first observation
  add_trace(
#Add our values from the object created above
    r = Aguero,
#Add the labels to our graph
    theta = c('Minutes', 'Cost', 'Dreamteam', 'Points', 'Average', 'Goals', 'Assists', 'Minutes'),
#Name our observation for the legend
    name = 'Aguero'
  ) %>%
#Repeat for our second observation
  add_trace(
    r = Kane,
    theta = c('Minutes', 'Cost', 'Dreamteam', 'Points', 'Average', 'Goals', 'Assists', 'Minutes'),
    name = 'Kane'
  ) %>%
#Repeat for our third observation
  add_trace(
    r = Vardy,
    theta = c('Minutes', 'Cost', 'Dreamteam', 'Points', 'Average', 'Goals', 'Assists', 'Minutes'),
    name = 'Vardy'
  ) %>%
#Repeat for our fourth observation
  add_trace(
    r = Lukaku,
    theta = c('Minutes', 'Cost', 'Dreamteam', 'Points', 'Average', 'Goals', 'Assists', 'Minutes'),
    name = 'Lukaku'
  )
```



Conclusion

In conclusion, spider plots can be helpful tools in comparing multiple results from different observations that do not rely on casual relationships. Creating spider plots is not difficult, but data should be properly cleaned and prepared before use to create the best plots possible. In order to create spider plots, one must first download and call a package (e.g. Plotly), build the graphic by layers, first calling for a polar format with points and choosing to fill between the points and then adding the data and axes for all observations. Finally, aesthetic options can be added in order to optimize your spider plot.