Data Science – Final Project Script

**Intro:**

For my final project, I decided to use various MLB data sets in order to find out which aspects of stadiums correlated best with the number of homeruns hit in each stadium. For this, I used one data set that encompassed the dimensions of all MLB stadiums (LF, CF, RF distances) from ballparksofbaseball.com, and another that calculated the surface area of the outfields of each stadium from fangraphs.com. I also gathered the cumulative offensive statistics from each ballpark from 2018 from baseball-reference.com. The multiple linear regression model using surface area and outfield dimensions did not turn out to be as successful as I thought. This got me thinking what IS a good predictor of homeruns.

I decided to branch out a little from my original question and sought to find out what the best predictor of homerun distance was using individual player data from the 2018 season. This model considered various Statcast (a program powered by AWS) statistics such as exit velocity, launch angle, average hit distance, hard hit (95+ mph) percentage and barrel percentage. Given players Statcast statistics, the model was able to predict the average distance of a players homeruns with a less than 1% error.

Finally, I decided to get back to my original question involving stadiums. I wanted to use a combination of the dimensions/surface area and the offensive statistics for each stadium to project whether the stadium was in favor of the hitter or pitcher. I built a K nearest neighbor model which classified each of the 30 major league stadiums into one of five categories: very hitter friendly, hitter friendly, neutral, pitcher friendly and very pitcher friendly.

Now I will walk you through my code and explain how I built each model and the conclusions to be drawn from each.

**Block #1 – Packages:**

This first chunk of code simply loads in all necessary packages for me to complete my analysis.

**Block #2 – Ballpark Offensive Stats Data:**

This chunk of code takes a CSV containing the offensive statistics of each park I downloaded from baseball-reference.com and loads it into an R dataframe. After loading in the data, I removed three observations from the dataframe. These three rows were simply stadiums that hosted exhibition games or hosted MLB teams in a foreign country. These should be removed because each of these rows only held data for a couple games throughout that year, as opposed to the 30 MLB stadiums holding data for 81 games each. Lastly, I had to rename two of the values found in the first column, which held the stadium name for each observation. I had to do this because I needed to keep the stadium names consistent from data set/dataframe to data set/dataframe. This allowed me to directly compare the offensive statistics found in this dataframe with the dimensions for the corresponding stadium held in a separate dataframe.

**Block #3 – Outfield Area’s and Total Surface Area Data:**

This chunk of code reads in and converts a JSON file of outfield surface area data acquired using import.io to a dataframe. This process is relatively straight forward, as I simply set the URL for the file, read it in to R as a JSON doc, and convert it directly to a dataframe. I removed the entire first column from this dataframe because it simply contained the URL from where the data was acquired.

**Block #4 – Outfield Dimensions Data:**

This chunk of code reads in XML data containing various dimensions of each stadium acquired by using the Selector Gadget tool and specifies the unique nodes I am looking for in order to build the complete dataframe. To start, I set the URL to the site I was working with. From there, you read the page into R and break it down using HTML. Each set of code found below this specifies a specific node of data from the website using the unique HTML identifier.

The first line in each of these small chunks of code specifies the node to be extracted from the page, and then simply converts these to text. The code then removes the first element from each vector because this is the header value from the table on the website.

The issue with this data set is it included the three dimensions (LF, CF, RF) all combined into one column. In order to build the model, I wanted, I had to break this one column into three columns each with their own unique distance value from home plate. Thankfully the values were in the format ("330-L, 400-C, 330-R"), which made it easy for me to use regular expressions to break each individual value into its own column.

Once I had all data adequately cleaned, I created a list of the five columns of data I had extracted in order to turn the list into a matrix. This allowed me to easily build a dataframe from the matrix using as.data.frame(). Next, I added the correct names to each column as headers and sorted the dataframe by team (A-Z). The last thing I did was add a column called “AvgFenceDist” which simply added the three dimensions and divided them by three.

**Block #5 – Statcast Data**

This next chunk of code reads in the Statcast data which I acquired using the selector tool Data Miner, which made it really easy to select entire tables of data at once. This tool simply allowed me to select the table I was looking for, and automatically converted it to a CSV for me. The only cleaning I needed to perform on this data was removing the NA values, using na.omit().

**Block #6 – Team Standardization and Master Dataframe**

The first thing I do here in this chunk of code is create a character vector with each major league team’s abbreviation in alphabetical order. This will allow me to append this vector to each of my dataframes in order to make a standard key in each. I then used the cbind() function to append this vector to the left of each dataframe I have created above.

Next, I wanted to create a master dataframe which combined each of the previous three using the team abbreviation key. I did this in two parts using the merge function, because I you either can’t merge three dataframes at once or I couldn’t figure out how. So first, I used team abbreviation as a key to combine the first two dataframes. Then I simply repeated this process combining the third dataframe with the combination of the first two, again using team abbreviation as the key.

**Block #7 – Database Creation and Tables**

This chunk of code sets up a new database in SQLite called “baseballDB.” It then connects to the database and proceeds to create four tables within the database for each of my three dataframes, as well as the combined master dataframe.

**Block #8 – Homerun Outlier Detection**

This set of code simply creates a histogram of the homerun totals for all 30 MLB stadiums in order to visually determine outliers. It is obvious that there are two outliers of these 30 data points. You can see this because all stadiums were above the 140HR total, with the exception of the two totals between 120-130 range.

**Block #9 – Outlier Identification Using Database**

This chunk of code uses the database created above to select important information on both of the two outliers above. I first send a query to retrieve the stadium name, team abbreviation and homerun totals for the two stadiums with the least number of homeruns. I then send another query to retrieve the stadium dimensions for each of the two stadiums. I then added a column to the outfieldAreaDF that uses the rank() function to assign a ranking for each stadium based on the surface area of their outfield. I then selected each rank for the two teams and added the suffix “th” to the end of their rank. Finally, I used cbind() to combine these two vectors of data into a dataframe, and named the rows accordingly.

**Block #10-11 – Plot Outfield Area vs. Homeruns**

This code block simply plots the homeruns hit on the x-axis and the outfield area of each stadium on the y-axis, and adds labels to the graph. It then adds a regression line to the plot, using the coefficients found from the lm() function. You can see that the regression line is decreasing, which makes sense because as the total outfield area decreases you would expect more homeruns to be hit in that given stadium.

However, this model ends up not being such a great predictor as I thought it may be. It has an adjusted r-squared value of just 0.05449, and a p-value of over 0.11. These two simple statistics show that this model is not super accurate and there is little correlation. Because of this, I decided to skip predictions using this model and went on to create more accurate models in order for more accurate predictions.

**Block #12-14 – Statcast Multiple Linear Regression**

The first chunk of code here creates training and testing sets to be used for the multiple linear regression model. I used sample.split() to assign TRUE/FALSE values to a new column in order to split the data.

The next chunk of code starts by setting up the multiple regression model by inputting the formula to predict the Average Homerun Distance for each of the 700 player data points I have. I built this model using the training data set created above. From there, I used the step function to create the best model possible given all the variables. I used backwards regression to build the optimal model, meaning that the model started by including all variables and from there eliminated those which weren’t significant predictors.

Step returned the optimal model as: AvgHRdist ~ ExitVeloFBLD + AvgDist + BarrelPct

Once the optimal model was built, I used the predict() function and the testing data set to predict the average homerun distance for each player in the testing set. Once the average homerun distances were predicted, I calculated the percent error of the model. This took the predicted values, minus the actual values, divided by the actual values. I did this for each of the players in the testing data set, and took the mean of this in order to find the percent error on the entire testing set. This model was very accurate, with a percent error of just .0053%.

**Blocks #15-19 – KNN Model for Stadium Classifications**

This first chunk of code copies the masterDF to the knnMaster dataframe in order to normalize the values while maintaining the original data in the masterDF. To get the data set ready I converted a few columns to characters and removed other columns that would be useless/couldn’t be normalized. I then selected all columns that were factors using sapply(), and used lapply() to convert all the factor columns first to characters, then to numeric.

The next chunk is the code that normalizes all the values in the dataframe. It starts by setting the formula as a function, and then lapply() again to apply the function to all the columns in dataframe that needed to be normalized. I then take these normalized values and make a new dataframe with them, and used cbind() to append the column of team abbreviations to the left of the normalized dataframe in order to be able to keep track of each observation.

The next chunk of code creates a new column that classifies each stadium into five categories (very hitter friendly, hitter friendly, neutral, pitcher friendly, very pitcher friendly) based on how many homeruns were hit in each stadium. Lastly, it takes this column and makes it a factor.

The next set of code splits the data into training and testing the function createDataPartition(). It then creates the training and testing labels, which are the classification categories saying whether the stadium is hitter/pitcher friendly.

Now that we have everything set up, the next set of code creates and executes the K Nearest Neighbor model. The k value in the knn() function line is the number of neighbors to be considered by the model. Once the model was set up and executed, I put the results into a table comparing the actual classifications with those predicted by the model. The model predicted the classifications of the testing data set with a 60% accuracy.

**Blocks #20-24 – Neural Network**

These chunks of code set up and execute a neural network to predict the homeruns hit in each stadium based on all the stadium dimensions, outfield surface areas and offensive statistics. I start by copying in the dataframe used in my KNN model and take a 60% sample of the observations to create training and testing sets. Lastly, I take all non-numeric columns of the dataframe and convert them to numeric, again using sapply() and lapply().

The next block of code takes the min and max of each column in order to scale the dataframe. I then use the scale function and the min and max’s of each column to create a new scaled dataframe.

The next chunk of code takes the same sample as above, but from the scaled dataframe. Now that everything is ready to go, I finally create and plot the neural network using the training data from the scaled dataframe.

The next set of code uses the neural network created above to predict the homerun values for each stadium in the testing data set. However, the results of this prediction are scaled due to the input of scaled data. In order to compare to the actual homerun totals, I unscaled the data in the reverse fashion I previously scaled it.

Finally, to visualize the results I plotted the actual homerun totals vs. the predicted and unscaled homerun totals. You can see that this model was very accurate, as nearly all predicted points fall perfectly on the plotted line. The mean squared error of these predictions was less than 1, at 0.9468.

I converted this Mean Squared Error to a percent by dividing the error by the average homerun distance in order to compare the effectiveness of this model against the multiple linear regression model. Both models had a percent error, based on the mean squared error, of roughly 2% showing that they were equally accurate in predicting homerun distances and quantities per stadium.