Henry Robb
ITWS 4600

Make it or Break it

Abstract and Introduction

In the National Basketball Association, there are millions of dollars on the line every year that can be won or lost on the back of a teams' performance. So, every year it is the job of each teams' management to build the strongest roster possible. This task has changed a lot from the inception of the league back in the 1940s. At that point in time, many of the statistics that are taken for granted by most fans and scouts, did not even exist. Simple counting statistics like blocks were not even compiled until the 1970s! Because of this lack of statistics, many decisions were made by the "eye test", essentially based on what the scouts thought looked best. This can lead to great results, especially when the eye of the beholder is an experienced one, but it never had the consistency that sports teams desire.

Today, almost the opposite problem exists. There is such an extraordinary volume of data collected on each play and movement in the NBA that even for a paid staff of data scientists, choosing which are important and how to use said data can be a difficult choice. Because of this, I chose a very narrow lens through which to make my foray in Basketball Data Analytics. I looked at specific shot events in the NBA, and attempted to predict whether or not the attempt would convert successfully into a made basket. This analysis was well balanced because it is broad enough that there are significant quantities of data available to make such predictions, but it is also narrow enough that practical conclusions can be drawn by exploring this problem.

Data Description and Preliminary Analysis

I originally intended on doing a much broader data analysis of the entire NBA. This plan was rather foolhardy as it involved using a dataset with so many more columns then I could possibly know what to do with. With this dataset, I was going to attempt to find correlations between regular season statistics and post season success. However, it was simply too much.

In narrowing my focus, I found a Kaggle competition that is attempting to use various qualities about an individual shot attempt to predict whether it will be made. After analyzing the data set, I found there were a few things to note. There were very few noticeable missing rows or columns which demonstrates the care in the initial collection of this database. There were some features present that were going to be more important when coming up with an accurate algorithm for shot prediction. The more basic ones like the player's name (PLAYER_NAME), the team name (TEAM_NAME), the position of the player (POSITION), the home team (HOME_TEAM), and the away team (AWAY_TEAM). In addition to these string variables, there are also some numerical equivalents in the players' (PLAYER_ID), the teams' (TEAM_ID), and games' (GAME_ID) respective NBA IDs. There are also variables that are dependent on the specific shot. Some of the ones that may be helpful in building a model are the zones from which the shot was taken: there is a basic zone variable (BASIC_ZONE) which is more general, then a zone name (ZONE_NAME) feature that further divides the floor. Additionally, there are precise X and Y coordinates (LOC_X, LOC_Y) recorded for each shot instance. However, they are the two columns of data with the most missing points. This can be remedied by using the shot distance (SHOT_DISTANCE) in combination with the zone, rather than having to rely on the specific coordinates. Though there are more columns, the final one to mention now is the shot type (ACTION_TYPE), which describes whether the shot was a dunk, a jumper, and so on. This will

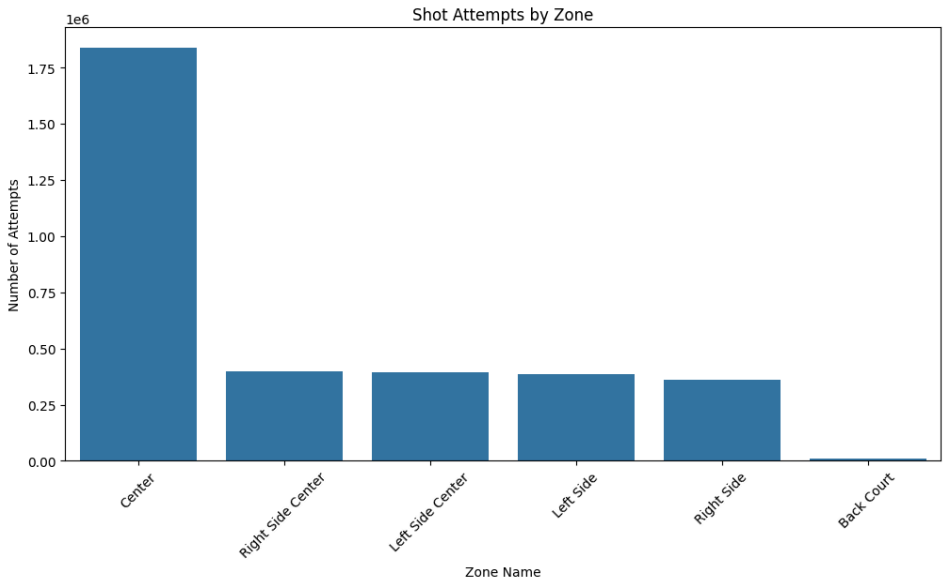
be a good sorting method for the shots as there are clear divides in the makability of each of them.

Finally, in supporting material, there are the results of the other individuals who attempted the challenge. There is a leaderboard present for the competition that will allow me to benchmark against what the best challengers could accomplish. I anticipate that I will not be able to generate any algorithms as accurate as theirs, simply because I am less experienced, but I will be able to compare to know if certain methods are at least on the right track.

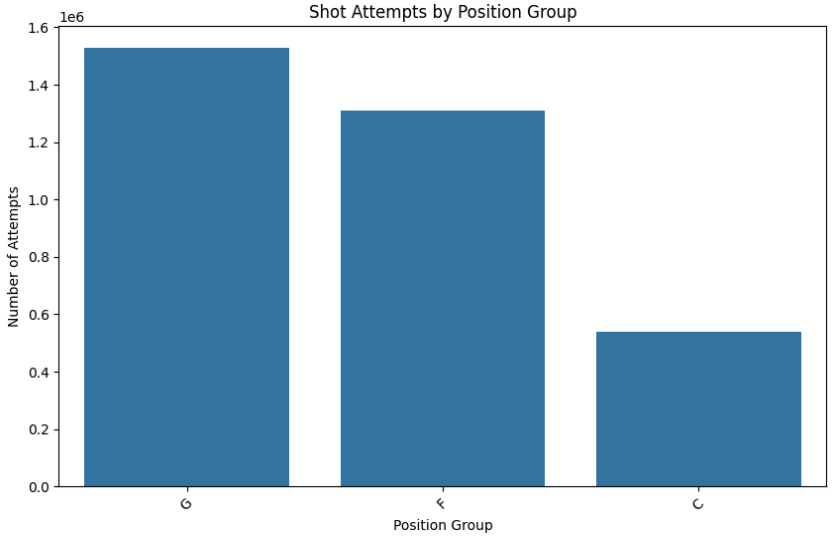
Exploratory Analysis

As mentioned in the prior sections, the data is very comprehensive, and therefore, there was no cleaning of note required. There were a significant number of object variables, mostly in the form of strings, so we will look at some visualizations of some of the more important variables when it comes to shot making.

For our first visualization, we can look towards which zones are most prominently shot from. Seeing a breakdown of the ZONE_NAMEs will be helpful in deciding whether they are unique enough to use in distinguishing accuracy.

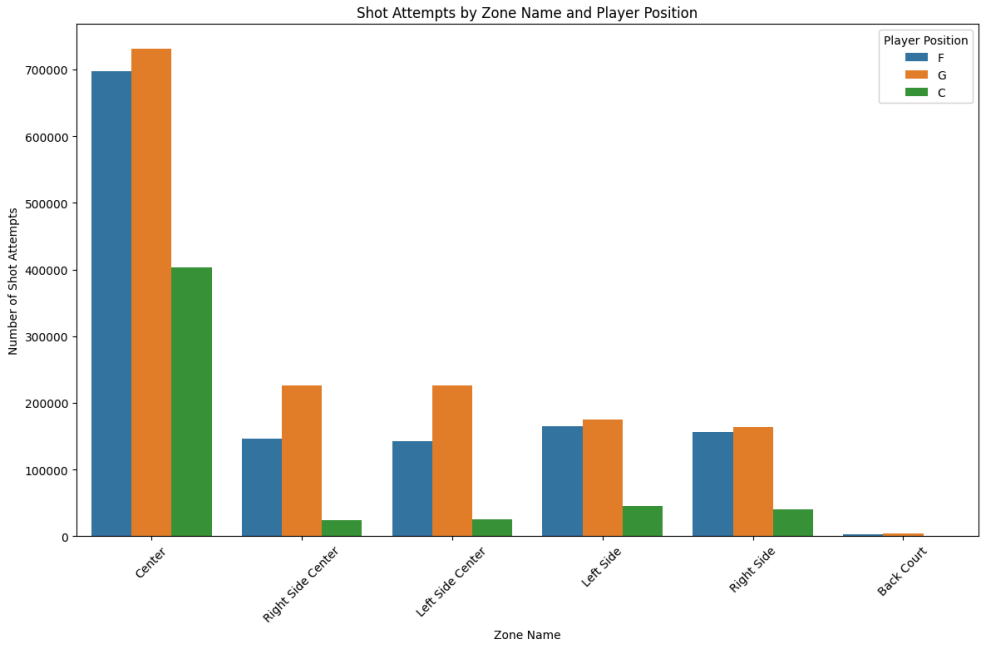


As we can see in the chart, a huge majority of shots are taken from the center of the court. Because of this large majority, it makes sense to attempt to use another category to see if any unique aspects appear. Next, we will inspect the shot attempts per each position group:

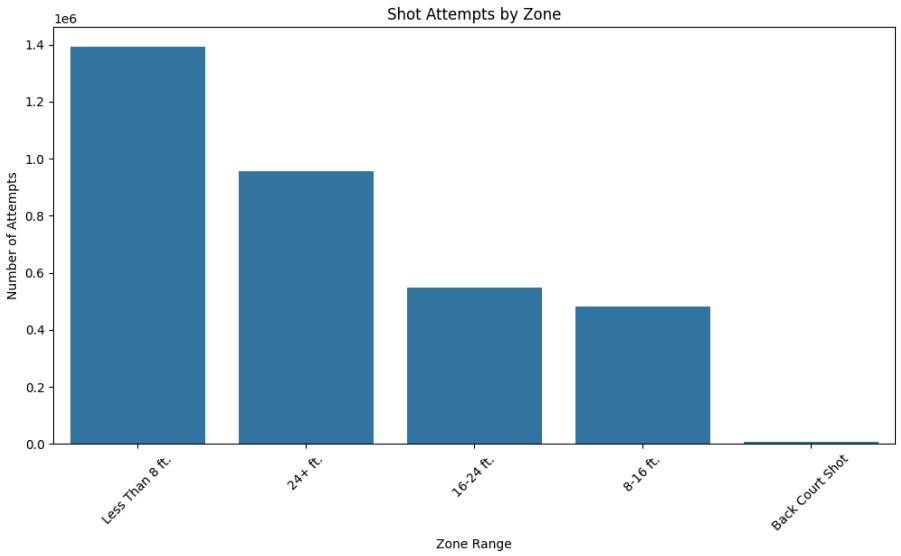


The guards are clearly the position group who are tasked with shooting the most. They are followed by the forwards and finally the centers. In order to have a greater understanding of each

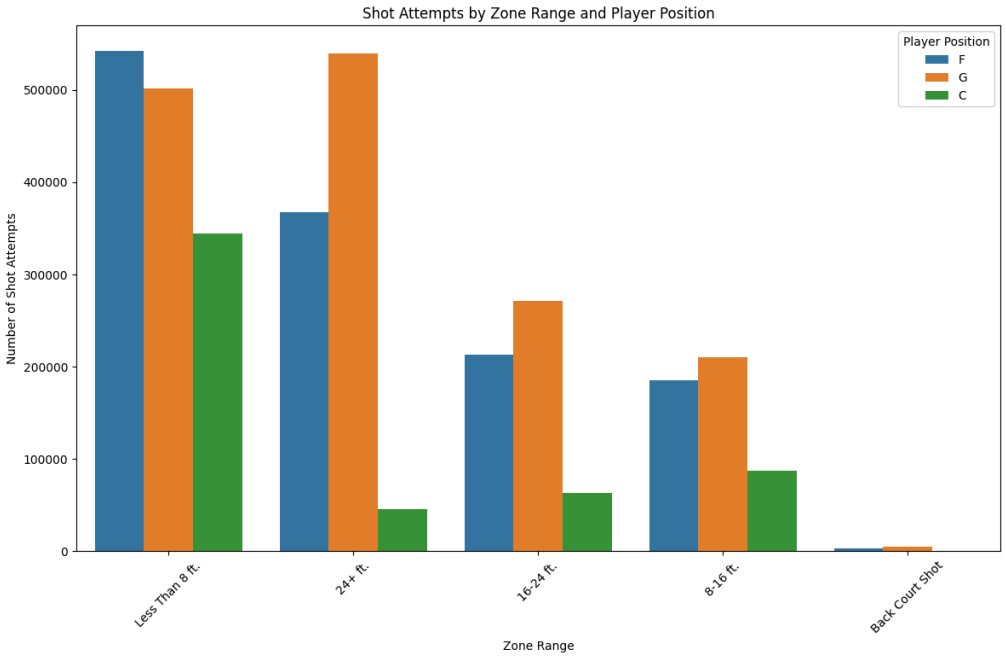
group's shot selection, we can combine the first two visualizations to find each group's shot attempts in each zone.



This chart follows very well with the basic understanding of the positions in the NBA where the center position is predominantly shooting from right underneath the rim in the center, and shooting very little elsewhere. Additionally, it also shows that the guards and forwards shoot heavily from the center, which makes sense considering our shot attempts by zone visualization, but these two positions maintain higher shooting numbers in the other zones, which also aligns with the shot attempts per position visualization. Next we will look at shot attempts by range, rather than zone to build a deeper understanding of the positions shot locations.



We see a very similar trend to that of the shooting zone in the shooting range. Most of the shots are taken closer to the basket, and as the range gets further, the number of shot attempts taken from there decline. Our next step will again be breaking it down by position group.



In this chart again, we see the prominence of the center shooting very close to the basket. But this chart also demonstrates that forwards only have their shot number superiority very close to the basket, and once we are outside 8 feet, the guards take the most shots.

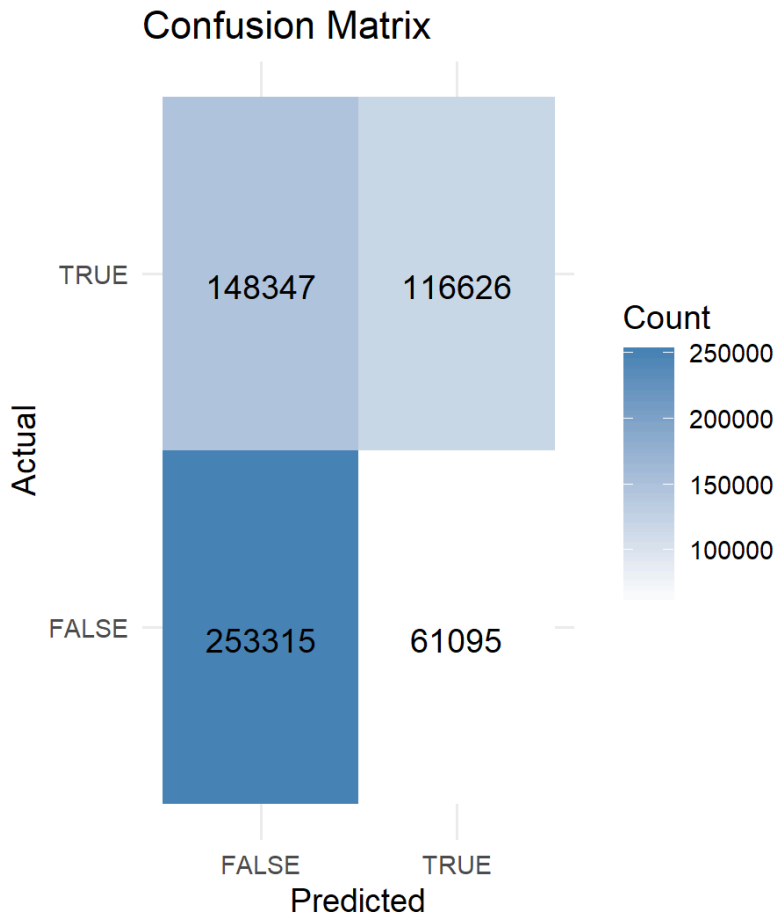
All of these figures will account for the best features for this problem. We must take into consideration the center prominence near the rim, and the guards prominence away from the rim and penalize their odds of making a basket outside their familiar territories appropriately. All of this to say, this is a complex problem, and these graphs get us just a bit closer to building a solution.

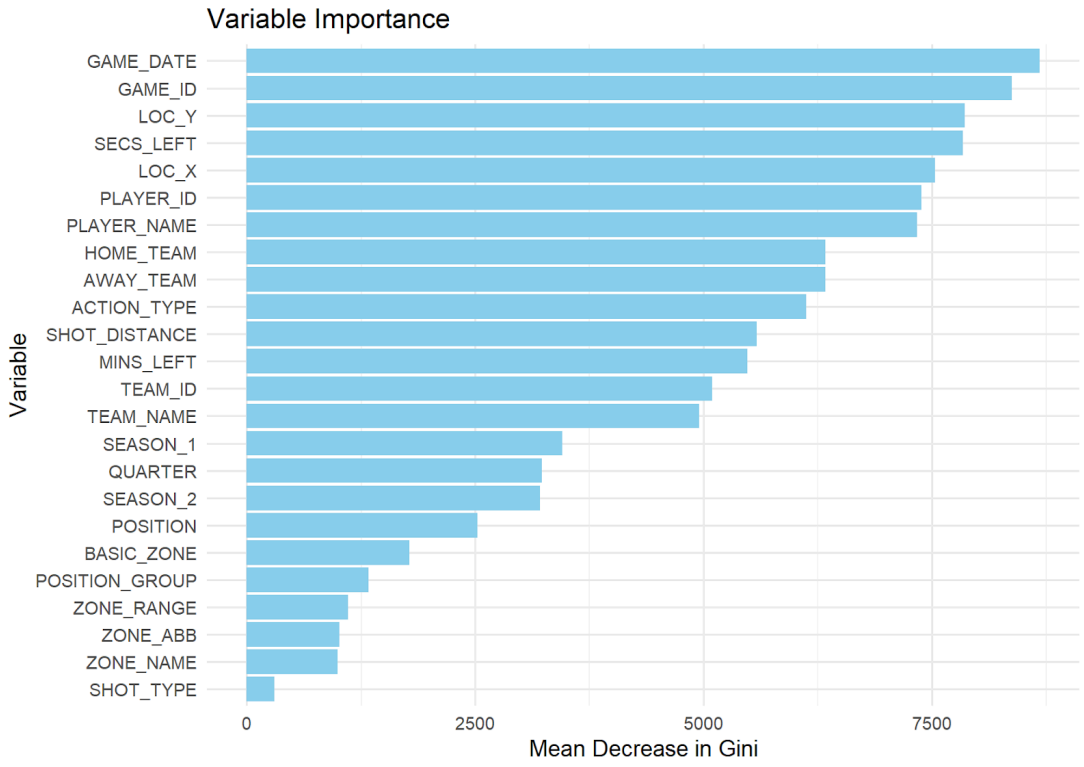
Model Development and Application of model(s)

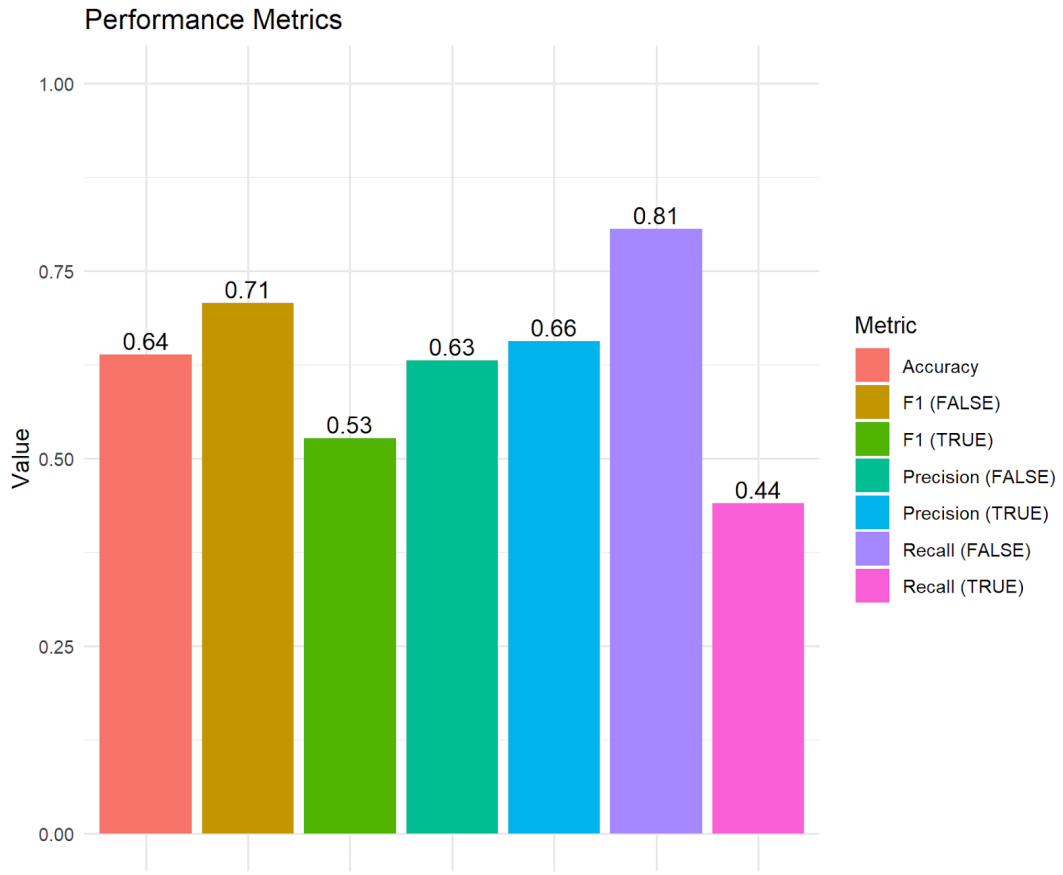
When making these models, my first thought was to use a k nearest neighbors algorithm. Because of their ability to clump examples with similar features, I believed that the shot prediction task would have been perfect as similar shots should have similar likelihoods of going in the basket. I ran into issues rather quickly with this method. The first of which was the format of the data. It is all nominative data in alphanumeric format! This was extremely difficult to work with. My solution was to convert the three columns I thought to be the most important, ACTION_TYPE, POSITION, DISTANCE to their own data frame and ensure they were all in numerical format. This solved the formatting issue, but then I started running into issues with the algorithm itself. While using the train algorithm and cross validation for the train control, I kept finding there were too many ties for the algorithm to ever complete a full run without erroring out. After a few more tries using different features and other values of k, I eventually concluded that this strategy was not going to be able to work on this dataset. Even beyond the simple errors I was running into, there was also a bigger issue. One does not really train a knn model, one instead has to process all the training points, and then process all of the test points. This likely would have been a death sentence for my poor computer as the dataset is composed of more than three million points. So, it is probably for the best that I had to abandon the strategy before attempting a proper run.

My next strategy was to go with the rf tree. I thought that this was going to be a far more successful strategy as it is much stronger in handling nominative data. Though, I did not make it through the development process here without error either. The first error I ran into was my

blank data. I was originally under the impression that rf trees would run fine as long as there was no blank data in the classification column; I was wrong. This frustrated me for a long time, but eventually, I decided I had more than enough data that even if I removed all rows with na data, I would still have more than I could possibly use. This assumption turned out to be quite accurate. After I omitted all of the na rows, I ran into a storage issue. With the size of the training set I originally chose and the proximity parameter set to true was leading to the rf function attempting to create a vector with an absurd size of 56 gigabytes. Let me repeat that: 56 gigabytes! So, as I have not been able to sneak a supercomputer into my laptop, I was unable to run under those parameters. This led to my removing the proximity parameter. This more than halved the size of the vector to a still huge 22 gigabytes. After this, I came to the rather obvious conclusion that I was being too ambitious with the size of the training set. I tenthed the data set and attempted to train the rf model again. This time, it worked! Here are some graphical representations of my results:



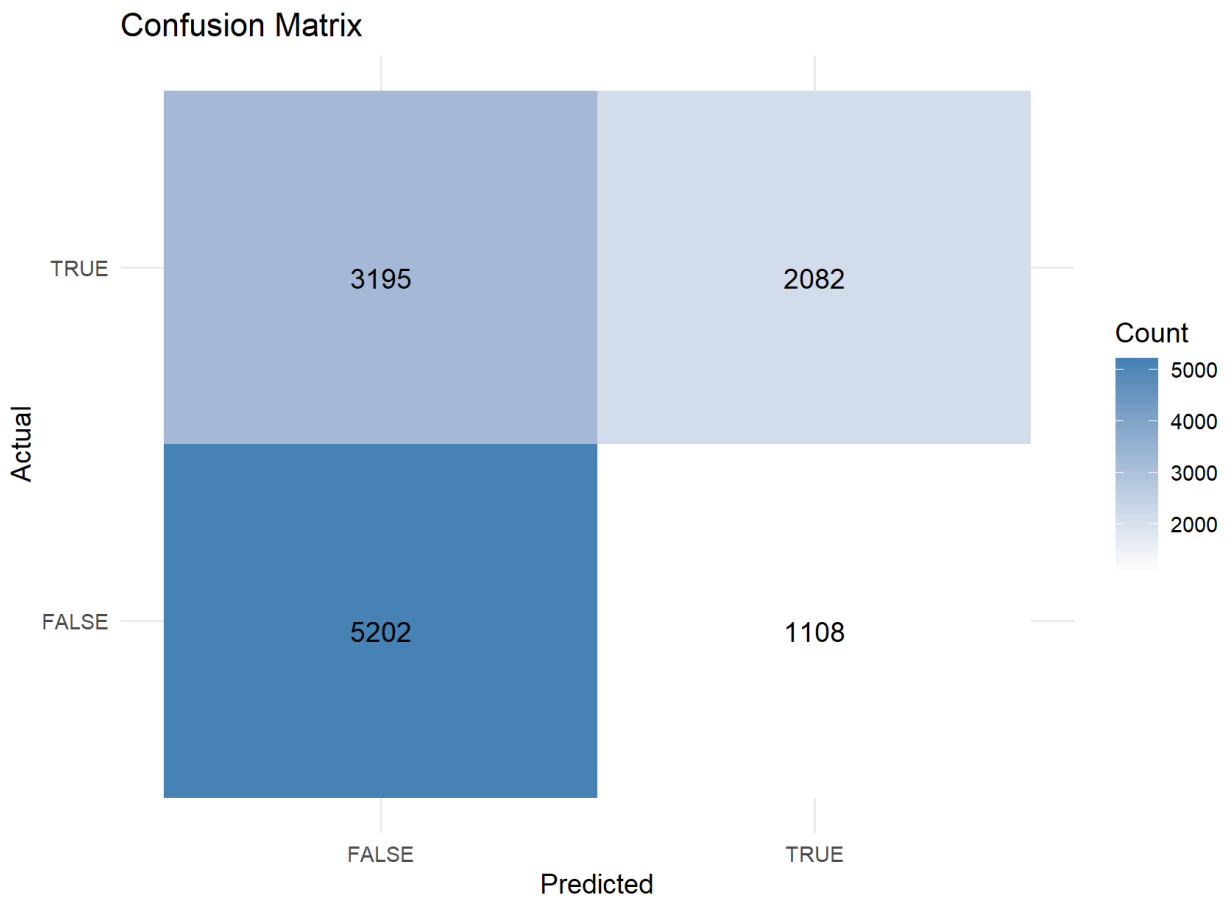

Variable Importance
GAME_DATE
GAME_ID
LOC_Y
SECS_LEFT
LOC_X
PLAYER_ID
PLAYER_NAME
HOME_TEAM
AWAY_TEAM
ACTION_TYPE
SHOT_DISTANCE
MINS_LEFT
TEAM_ID
TEAM_NAME
SEASON_1
QUARTER
SEASON_2
POSITION
BASIC_ZONE
POSITION_GROUP
ZONE_RANGE
ZONE_ABB
ZONE_NAME
SHOT_TYPE
Variable
0
2500
5000
7500
Mean Decrease in Gini

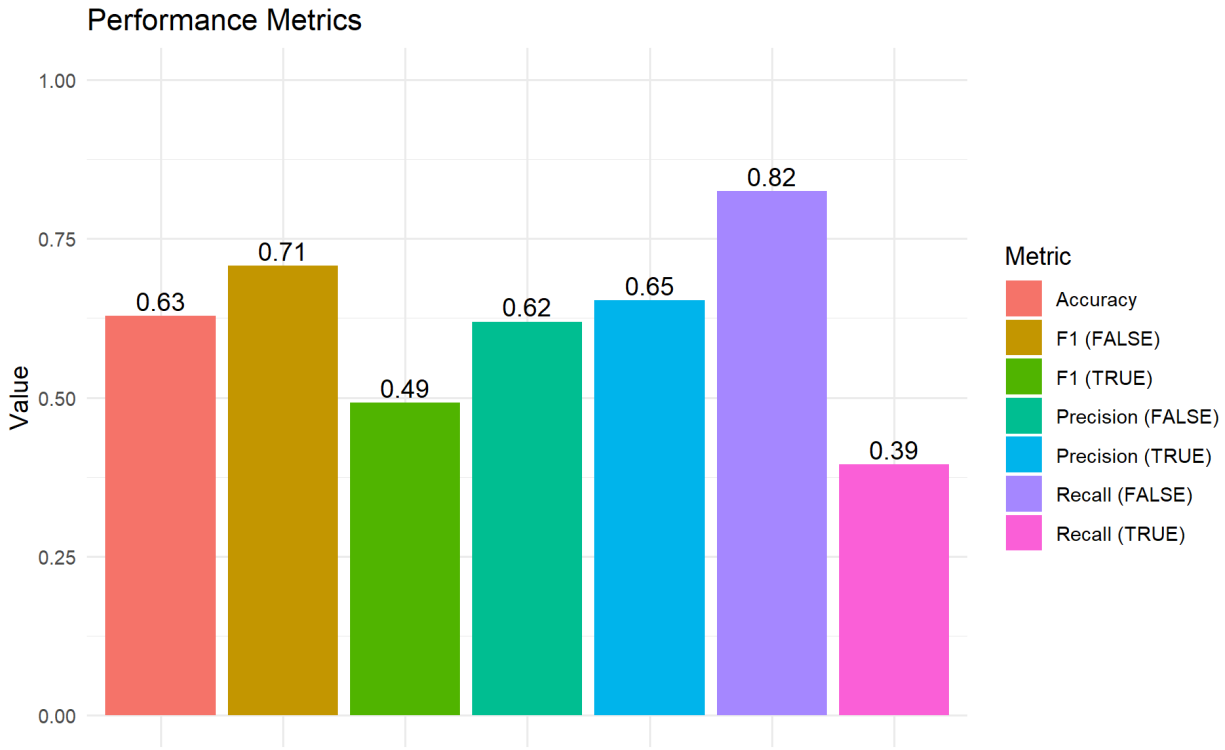

Performance Metrics
1.00
0.75
0.50
0.25
0.00
Value
0.64
0.71
0.53
0.63
0.66
0.81
0.44
Metric
Accuracy
F1 (FALSE)
F1 (TRUE)
Precision (FALSE)
Precision (TRUE)
Recall (FALSE)
Recall (TRUE)

I will discuss the results further in the conclusion section, but based purely on accuracy, I was pretty satisfied with the results.

My next attempt was to use a support vector machine. I thought this would be a good balance as it was something more complex than a tree or a simpler linear regression model. I immediately started running into the memory issues that I had seen previously in the rf tree attempts. My first try using all of the training data available to me, it was attempting to use a vector with a size of 98 gigabytes. Again, I unfortunately do not own a super computer, so I had to make it more efficient. The biggest change I had to make was to do with the GAME_DATE and PLAYER_NAME. These had to be changed because their dimensionality was so astronomical, making it impossible to make a model of any reasonable size. The way I got around this for players was by ensuring that each player had a certain number of shot attempts present in the dataset I was using, and if they did not, their stats were relegated to player "other". For the game dates, I simply sorted them by their month and the day of the week they were played. This drastically reduced the dimensionality from the thousands of dates the games were originally played on to the 7x12 possibilities they could have been sorted into after the conversion.

After all of this preprocessing, I was finally ready to train. At first I was planning on trying to use a polynomial kernel. I thought this would be best as it would provide me with the ability to match a very complex boundary. However, after several attempts and a lot of downsizing, I found this just was not that feasible. I was letting the program run for hours at a time, but it would never actually finish running. I considered going to a linear model, but eventually concluded that it would not be powerful enough for how complex of a task this is, especially when considering how many variables are present. Finally, I settled on a radial kernel. This was not my first choice, but it was the one that made the most sense considering the limited computing power I had at my disposal and the complexity of the data set. It still took an extremely long time to run, however, it did run, and that's what counts. You will find the resulting graphs below:


Confusion Matrix
TRUE
FALSE
Actual
3195
2082
5202
1108
FALSE
TRUE
Predicted
Count
5000
4000
3000
2000


Performance Metrics
1.00
0.75
0.50
0.25
0.00
Value
0.63
0.71
0.49
0.62
0.65
0.82
0.39
Metric
Accuracy
F1 (FALSE)
F1 (TRUE)
Precision (FALSE)
Precision (TRUE)
Recall (FALSE)
Recall (TRUE)

Conclusions and Discussion

The first observation that jumps out to me is the similarity in the accuracy of the results. The support vector machine achieved an accuracy of 63% whereas the RF trees were able to achieve an accuracy of 64%. It is rather absurd to me how different these two methods are yet how they seem to have found a very similar asymptote when it comes to accuracy. A huge limiting factor for this project was the time I had available and the computing power I had at my disposal. If one of those were greater, I believe that accuracy I could have achieved would have jumped significantly. I am under this belief because I achieved a minimum accuracy of 63% for both of my models and only used a very minute portion of the data. In fact, for the support Vector machine, after the downsizing I had to do, I really only ended up using about 2% of the data I had at my disposal. Now, this is still an extraordinarily large amount seeing as the base data set had more than 3 million rows to begin with, but there was still a lot of data that I did not get to properly include in the creation of the predictive models I ultimately ended up with.

Still, in the realm of what could have been done with greater compute power, I think the support vector machine could have been far more successful if I was able to use the polynomial kernel, especially if I was using a very high polynomial kernel. If you remember, I only ended up choosing the radial kernel as it was a better compromise when it came to run time on my machine, but I do not believe it had the greatest ceiling of all of the kernel options.

All of that being said, I am extraordinarily impressed with and proud of the results that I was able to generate. Considering that I was not using any fancy neural networks or reinforcement learning techniques, I think these are great results. A minimum of a 63% accuracy is rather impressive considering what the data set is representing. It is not representing something very certain to begin with. Most people, when they shoot a basketball, do not even know themselves whether or not it is going in. So to be able to see a data set with a number of values, granted a very elaborated curated number of values, describing each shot and how it happened and being able to predict, more than half the time correctly, whether or not the shot is going in is a rather impressive feat for a not extremely complex R program.

To sum it all up, the results were pretty impressive for an undergraduate student's data analytics final project. More than half the time I was able to predict whether or not a shot were to fall, while barely even breaking the surface when it came to the data that was available to me. Thus I think it was a very successful first outing and if I were to go at it again, I would bring a better laptop and be ready to wait a while to compute some far more powerful models.