NBA Analytics: Powered by Spark

Pierre McWhannel**
pmcwhannel@uwaterloo.ca
University of Waterloo: Center for Computational Mathematics in Industry & Commerce
Waterloo, Ontario, Canada

Vincent Luong†
v3luong@uwaterloo.ca
University of Waterloo: Center for Computational Mathematics in Industry & Commerce
Waterloo, Ontario, Canada

Ren Yuan Xue‡
ryxue@uwaterloo.ca
University of Waterloo: Center for Computational Mathematics in Industry & Commerce
Waterloo, Ontario, Canada

Figure 1: Pre-dunk of Lebron James and his teammate Dwayne Wade running by with bravado

KEYWORDS

NBA, Analytics, Spark, Machine Leaning, Graph Analysis.

*Focus on Shot Prediction
†Focus on Graph Analysis
‡Focus on Player Analysis

CS 631, December 14, 2020, Remote

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Pierre McWhannel, Vincent Luong, and Ren Yuan Xue. 2021. NBA Analytics: Powered by Spark. In *CS 631 '20:Data-Intensive Distributed Analytics Project, Remote.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Analytics are used all the time in the NBA world. NBA teams hire analysts to look for trends in data to give their team an edge over others. We aim to find data on the NBA and provide our own insight in the basketball world. We look to provide a variety of insights about NBA teams and players — both insight catered to a hardcore fan, an NBA professional or even insight that is more "for fun" that can be used for trivia.

There are a lot of publicly available data for the NBA. Sites such as basketball-reference.com and NBA.com have plenty of statistics for anyone to use. We try to use all this data to provide interesting

analysis and insight in the basketball world. To do this, we use web-scraping techniques to take data from basketball-reference (BR), and process this data with the help of python and spark. There is an enormous amount of data on BR — there have been over 70 years of the NBA, and modern NBA seasons contain over 1000 games. With information on every player, every game, every team, we decide to use spark to speed up the process for processing all this data.

We use the basketball data in several different ways. One, we create interesting plots that showcase the data and visualize the relationships between players' features. Second, we create a graph with the data. We create a graph where is node represents a player, and an edge connects two players if they have played with each other. We perform classical graph algorithms on the graph and provide insight on graph statistics computed. Lastly, we create a machine learning model with the data. We use the data to predict whether a shot in an NBA game will be a *make* or *miss*. We directly use the data from BR to build the model, along with some new data derived from BR.

In section 2, we cover the overview of tasks and the flow of data between tasks and when, where, and why spark is used. In section 3, we talk more in-depth on how we scrape data from BR and how the data is processed. In section 4, we go over a variety of visualizations in our data to explore and analyze the patterns in players' metadata and "per-game" statistics. In section 5, we go over the graph created and what insight we gain. In section 6, we go over the entire process on how we create the machine learning model used to predict if a shot goes in or not with a focus on our new feature OPPFGPct. In section 7, summarize the accomplishments and leanings throughout our time on this project. In section 8, we explore more ideas on how to use basketball data that we did not have the time to do. The objectives of are project are as follows:

(1) Primary Objective: Learning Objectives
   - Learn how to web crawl and collect data with Spark.
   - Learn to implement graph algorithm in Spark.
   - Learn how to perform feature engineering with Spark.
(2) Secondary Objective: Goals
   - Create system of tools for web crawling basketball-reference.
   - Analyze trends of NBA player data.
   - Implement a graph algorithm in Spark.
   - Predicting whether a shot will be a *make* or *miss* and what features are important.

We've made all our code open-sourced at https://github.com/pmcwhannel/NBA-analytics.

2 OVERVIEW

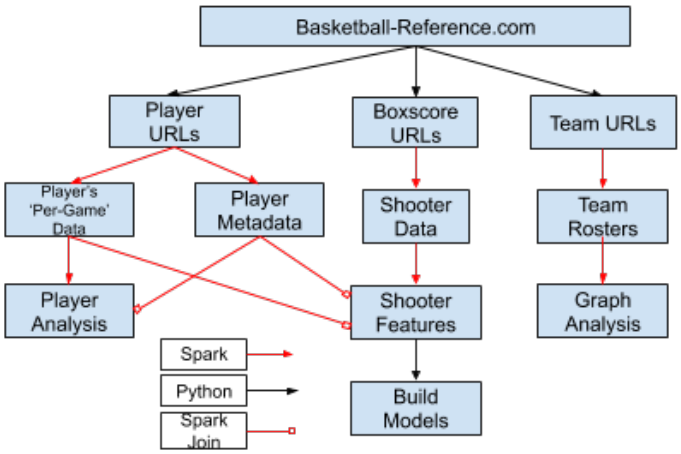


Figure 2: Procedural Layout of Data System

Basketball-Reference.com: Basketball Reference (BR) is a website which keeps track of many statistics and counts of NBA data. This data ranges from player data, game data, team data, season data, draft data, and more NBA related data. The data is not available in an accessible database; however, it can be extracted from the HTML code displaying the data. All our data was scraped from "https://www.basketball-reference.com" webpages this required using XPath to efficiently query the HTML source code; we originally used regular expressions to parse the HTML code but soon found this to be less efficient. However, we still used regular expressions on certain returned values from XPath such as extracting shot distance from the shot chart. All in all this is the root block in figure 2 for which all data for downstream tasks is sourced from. Meaning errors here will propagate to downstream tasks.

Player URLs: This task was to save a text file with URLs to all NBA players webpages on BR. Players' webpages on BR displays personal data such as weight, height, shooting hand, nationality, and other personal attributes. Additionally, players' webpages have a variety of statistics for each season the player has played in the NBA, both aggregates and totals. Extracting these URLs was accomplished by looping over all the hyperlinks to player webpages from a player directory page using python. The output of this task is a text file with all players' hyperlinks and ordered alphabetically. We used python here since there are only 4803 player URLs.

Boxscore URLs: This task was to save a text file with URLs of all boxscores this provides access to the shot charts of the same game. A shot chart has information about who took the shot, whether they scored or not, and the location of the shot. This data is essential for building a supervised machine learning model to predict whether a shot will be *made* or *missed*. Unfortunately, shot chart data is not available for all NBA games in history but starts in the 96-97 season in a game between the Boston Celtics and Chicago Bulls (The first recorded shot was Michael Jordan's). The output of this task was the URLs to all boxscores starting after the aforementioned

game all the way to the end of the most recent season (19-20) and ordered chronologically. This task was achieved using python since there was 62,097. Spark could be used here but once we tested with python we already had the links stored to a textfile. In future tasks we weren't as naive as we were here and found ways to estimate or count how many links then decide.

Team URLs: This task was to save a text file with URLs to team rosters for a particular season. The webpages which the URLs link to here is a page with the roster of all players who played on the team at some point during the season. These URLs are used for the downstream task of the graph analysis. The ordering of the output in the text files is by season and then alphabetically this was achieved using python since the number of URLs was only 1573.

Players "Per-Game" Data: This task was to extract the regular season per-game data statistics of each season a player played. This was accomplished using Spark to extract in parallel the player *per-game* statistics by creating an RDD of the player URLs text file. The output of this task is a text file containing a single season's *per-game* statistics for each player and each season they played. One issue during the extraction of *per-game* statistics was certain older players had missing columns of data which caused problems as we used a fixed schema of 31 known columns to identify the information in each column within the mappers. These tables which did not fit our schema were filtered out; resulting in 1396 players *per-game* data discarded from the total of 4803 players in the NBA's history. The majority of these players whose "per-game" data statistics were lost played before circa 1970.

Players Metadata: This task was to extract player metadata pertaining to personal information that is not relevant to a season statistic. This includes data such as height, weight, nationality, shooting hand, and birth date. This data was processed from the player URLs using Spark and again using XPath in the mapping functions. This data was then saved using Spark's partitioned *saveAsTextFile* method.

Shooter Data: This task was to extract shooter data from the shot charts, this was done by converting the boxscore URLs into an RDD. Spark was then used to extract all the shots and some shooter features which were available with the shot charts. This process resulted in over 4.5 million shots and took approximately 5000 seconds and then was saved using *saveAsTextFile* once again, to avoid needing to re-run. This data will then be read later in the shooter features task before proceeding to its final downstream task of building a predictive model for shooting.

Team Rosters: This task was to process the team URLs in parallel using spark. Each URL is sent through a mapper which then emits an adjacency list for each player as a key and all other players as the list. Then in the reduce phase all players a player has played with for all seasons and teams they've played on can be reduced to a single adjacency list per player. This adjacency list per player is then saved using *saveAsTextFile* for later reading in during the downstream task of the graph analysis.

Player Analysis: In this task, we are interested in the distribution of players' heights and weights and how it affects players' and the correlations to players' performances on the field. First, we extracted heights and weights from the text files created in *Players Metadata*, then plot them both individually and with respect to the birth year of the players. Furthermore, we extract various features from *Players "Per-Game" data* files and join the height/weight data to the current RDD using the player ID key. Finally, we collect all the features and plot them against players weights and heights and provide insight on the results.

Shooter Features: This task utilizes the shooter data stored in the text file partitions from the *Shooter Data* task. In this task we will join *player "per-game"* and *players metadata* to each shooter. Additionally, in this task we developed a new feature we coined the Opponent Field Goal Pct (OPPFGPct) this is a measure for each season the field goal (FG) percentage of opposing players from all shot locations. The heuristic behind this feature is to provide information of where a team's defense is strong and weak. Ideally, this feature could be replaced by who a shooter was defended by; however, this information is not available on BR. This feature will be discussed in more detail in the *Shot Prediction* section as it is found to be the most important feature in the predictive model. The output of this task is all the shots and features saved using *saveAsTextFile*, then these files will be read into python to train our predictive model during the *build models* task.

Graph Analysis: In this task, we take the adjacency list created from the *Team Rosters* task and perform some analyses using common graph statistics. The adjacency list is read in as a spark RDD, and we use a spark implementation of Dijkstra's algorithm to analyze the graph created with the adjacency list.

Build Models: This task is to train and evaluate predictive models which predict whether a shot will score or not given shooter features. In this task a model is selected and variable importance is inferred from the results. The models are trained on the data from the *Shooter Features* task.

3 DATA COLLECTION: WEB SCRAPING

There is a lot of publicly available data about the NBA on the internet. We choose to gather our data from the website basketball-reference.com (BR). On BR, there is a webpage for every player in the NBA, past or present, for every game that has occurred in the NBA, and for every team in the NBA in each NBA season. We needed to find an efficient way scrape all this data in a usable way. We started by categorizing the different data we needed — data on players, data on teams, data on games. In each categorization, we created a 4-step process for getting all the data we needed. As an example, suppose our categorization is on player data. We start by collecting the URLs of every single player's page. Second, we visit a player's page find our data of interest on the page. Once we know what data we want, we create a XPath expression to grab the HTML elements of that data. XPath is a querying language designed for HTML/XML. This XPath is made sure to be extendable to any other player's page. Lastly, we parse through the HTML elements into a format that can be easily interpreted by spark. The XPath expression and HTML parsing is then done on every single player's webpage gathered in step 1.

In the first step, there is a page that lists all players and their respective webpages. (There are similar pages for the other categories) On that page, a little bit of webscraping is done with XPath to gather all the URLs of the player's pages.

The second step in this process is quite simple, we visit a webpage to find the data we would like to collect. BR contains a lot of

information, a page for a player contains multiple tables, along with various metadata that are not in tables. We simply decide which data on each page would be useful for our analyses.

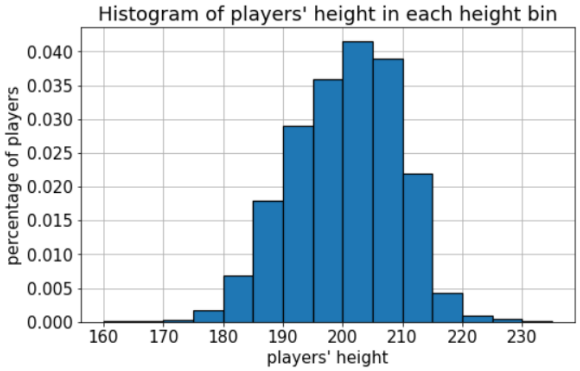
The third step requires some knowledge of XPath, a querying language for HTML/XML. We create a XPath expression filter the HTML elements on the page for pieces of information we want on a player's page. If the table was used to represent this data (such as with yearly averages of a player), then the XPath expression would capture all the rows in the tables. Sometimes, the table was not well formatted on the HTML pages, such as with a player's metadata. Information such as height, weight, country of birth, draft number, were not always available, or not always in the same place. This led to more general XPath's that capture more HTML elements than necessary, and requiring more work when parsing through the HTML elements.

The fourth step is to parse through the HTML elements found with XPath into a format that can be easily read into an RDD. This meant turning tables into lists of lists, so that once read into an RDD, as long as we know the format of the data, the data can be easily manipulated. The HTML elements were stored in a Element object from the *lxml* library, so learning of *lxml* was required.

Once we finish this process, we used the XPath expression obtained in step 3 and the parsing in step 4 on every single player's page. The scraping of each page can be done separately, so utilizing spark sped up this process. There are tens of thousands of webpages for the games played in the NBA, so utilizing spark to speed up this process was essential. Once all the data has been processed for a single category, it was saved for future use in our analyses.

4 PLAYER ANALYSIS

First, histograms of players' heights and weights are created. The figure below suggests that the heights and weights of players follows a normal distribution as expected. Specifically, the mean and standard deviation of NBA players' height is 197.6 *cm* and 9.23 *cm*. Whereas the mean and standard deviation of their weight is 209.34 *lb* and 26.20 *lb* , respectively.



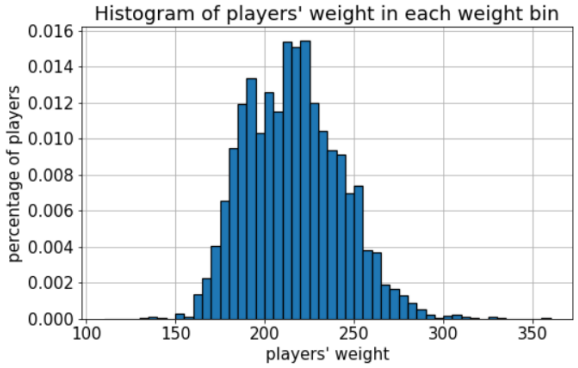
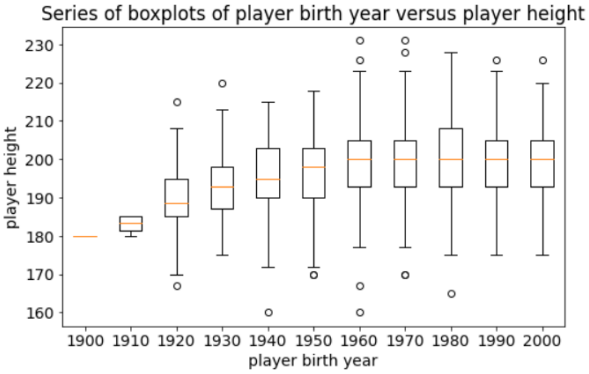


Figure 3: Histogram of NBA players Heights and Weights

Then, we take players birth year into consideration. Since the players birth years ranging from 1900 to 2000, we created 10 bins to group the birth years. Once the years are rounded to the nearest 10, we use a dictionary with keys and values being the bins and the heights and weights. With that, we can plot a series of box plots. According to the figure below, we see its evident players' heights and weights increase over the years.



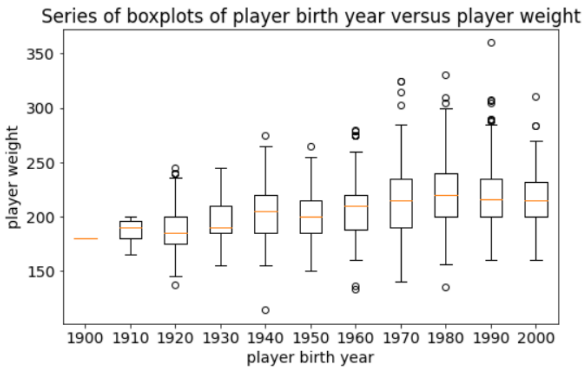


Figure 4: Series of box plots of player birth year versus players' heights and weights

Finally, we investigate the relationship between players' heights, weights and their career statistics. In particular, we explored players' per game points, assists, blocks and free-throws. We join these features with heights and weights of the players using the player ID key and calculate the correlation between them. Notably, we see moderate positive correlations between heights/weights and per game blocks and moderate negative correlations between heights/weights and per game assists.

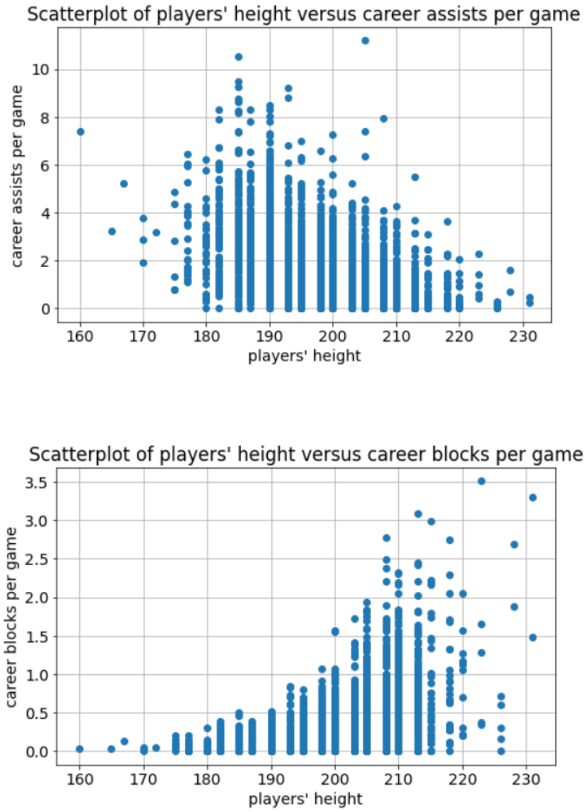


Figure 5: Scatter plots of per game assists and blocks versus heights

5 GRAPH ANALYSIS

With the data from basketball reference, a goal is to create a $G = (V, E)$, where there is a vertex for every player who has played in the NBA. We connect two players u, v with an undirected edge uv if player u has played with v in his lifetime. We call this graph the *played-with* graph. This graph is then used to compute statistics such as the *Shaq* number — a basketball version of the Erdős number. The "Shaq" number of a player u is the distance of the shortest path from Shaquille O'neal to u in G.

5.1 Graph Creation

To create our G from above, we start by scraping all team rosters from basketball-reference.com. To do this, we gather all the team page URLS for each year in the NBA. In total, there are around 1500 of these links. Then, we use spark to create an RDD with a list of these links. The next step is to scrape the team roster from each of these URLs. These rosters are stored in an RDD so that we can now create the adjacency list of G using the RDD's *flatMap* method. This created a graph of over 4000 vertices and over 100,000 edges. Due to the size of this graph, we decided to build this graph with the help of spark. Our process is easily parallelizable, enabling spark to speed up the graph creation process easily.

5.2 Analysis

Once our graph G is created, we look to find player's "Shaq" number, their "LeBron" number, and other various distance from one player to another. To do so, we need to find the shortest path from Shaq's vertex to every other vertex. We do this by implementing Dijkstra's algorithm in spark (See section 5.2.1). You can see the various player's "Shaq" number along with several other numbers in figure 1.

After computing player's "Shaq" number, we noticed some other interesting properties of the "played-with" graph. The "played-with" graph is connected. Every player is connected to every other player. Intuitively, we were unsure if this was true. There are players who only play in the league for a season or two on a single team and then leave the NBA. Furthermore, the NBA has been around for so long that we thought there would be a high chance of an isolated bubble of players. However, this is not the case, and it turns out the "played-with" graph is a hugely connected 4000 vertex graph where the average degree is around 60. In fact, the maximum "Shaq" number of a player is only 7!

Player	Shaq #	Bill Russel #	Steve Nash #	Kyle Lowry #
Larry Bird	2	2	2	3
LeBron James	1	5	2	2
Stephen Curry	2	5	2	2
Luka Doncic	3	5	2	2
DeMar DeRozan	2	5	2	1
Brian Scalabrine	2	4	2	2
Khem Birch	3	6	3	2

Table 1: Shaq # of various players

5.2.1 Spark Implementation of Dijkstra's Algorithm. In our "played-with" graph, we analyze the graph using Dijkstra's algorithm. Due to the size of the graph, we implement Dijkstra's algorithm in spark to speed up the process. Dijkstra's algorithm parallelizes well, especially in a MapReduce setting. Below is the pseudo-code of the MapReduce implementation that we implemented in spark. We use $flatMap$ and $reduceByKey$ in spark to replicate the algorithm.

Input: Graph $G = (V, E)$, and node $u \in V$

(1) Initialize distances of all vertices $v \in V, v \neq u$ as ∞. Distance of $u = 0$ Let d_v be the distance of v.

(2) **Map:** For each vertex $v \in V$ with distance $\neq 0$:
- for each neighbour n_v of v, omit $(n_v, d_v + 1)$

(3) **Reduce:** for all pairs $(v, d_1), (v, d_2), ...(v, d_n)$, $d_v = \min d_i$
(4) Repeat (2-3) until all distance are finite, or the number of vertices with distance ∞
(5) Output all distances

6 SHOT PREDICTION

This section walks through the development of a predictive model for shot prediction. They key development phases of this model are data wrangling, feature engineering, and experiments and results.

6.1 Data Wrangling

Firstly, a list of box score links were collected into a text file. Next, a Spark job was created to extract relevant data from the shot charts. Through this process 4,750,345 shots were collected. The features extracted from the shot charts were as follows:

(1) Player key: $Unique$
(2) Quarter: $1, 2, 3, 4$
(3) Make/Miss: $make/miss$
(4) Shot location: (x, y), $pixels$
(5) Time remaining: $MM : SS$
(6) Game score: *shooter's team score - opponent's team score*
(7) Distance from hoop: $feet$
(8) Player's team: *3-letter abbreviation*
(9) Home team: *3-letter abbreviation*
(10) Away team: *3-letter abbreviation*
(11) Season (represented by the year the season ended in)

In separate jobs, data was collected for player per game statistics and player metadata (i.e. weight, height, shooting hand, etc..). These were then read in as RDDs and joined on the temporary key of *Player Key + Season* and *Player Key* respectively. The shot chart data from which shooting locations is extracted is a 500x472 (horizontal x vertical) pixels in size. We didn't see any reason to scale to a more common distance unit, therefore, shot location data is in terms of pixels from a datum in the top left corner of the shot chart amd ranges in $x \in [0, 500]$ and $y \in [0, 472]$.

6.2 Feature Engineering

As seen in the previous subsection those were the primary feature extracted from the shot charts. In addition to those the features gained during the join of the player *per-game* statistics and metadata data were as follows:

(1) Position: $PG/SG/SF/PF/C$
(2) 2P%: $\frac{2PM}{2PA}$
(3) 3P%: $\frac{3PM}{3PA}$
(4) eFG%: $\frac{FGM+0.5 \cdot 3PM}{FGA}$
(5) Weight: kg
(6) Height: cm
(7) Shooting hand: $Right/Left$

The *per-game* statistics were joined using a *player key + season* temporary key to ensure the statistics of the shooter only for the season when the shot took place were added. The player metadata was joined using solely the *player key* as these don't change between seasons. These features required some cleaning as would be expected since they are coming from a raw source and categorical features required being one-hot encoded.

Additionally, a new supervised feature was integrated coined Opponent Field Goal Percentage (OPPFGPct). The purpose is to have a representation of the difficulty of a shot location against a particular team since given a location in a real game there would most likely be a certain defender covering the shooter. This feature would require calculating an aggregated statistic of similar shots against the same team as the shooter's shot that we wish to add the feature to. To elaborate this requires looking for all shooters who took a shot against the same team in the same season. Next a shot similarity is needed to obtain a weighted field goal percentage for the specific shot location of all shooters. The first idea we had was to use a players shot location given by (x, y) in pixels and for the given shooter weight all other shots by a Gaussian distribution centered on the shooter's shot location. Then all other shots taken by players within the same season against the same team would be weighted according to the the value of the likelihood for their position when the Gaussian is placed on the shooter of interests position. The issue with this technique was that for each shooter this would need to be recalculated and there are approximately 82 games and 80 shots against a single team each season meaning 6,560 evaluations for each shot (4,750,345). After letting this map-reduce job run for a few hours we aborted this plan. The calculation of this technique would be as follows:

$$OPPFGPct(s_i) = \frac{(\sum_{k=1}^{n} f(x_i, d_k))_{make}}{(\sum_{k=1}^{n} f(x_i, d_k))_{make} + (\sum_{k=1}^{m} f(x_i, d_k))_{miss}}$$

$$f(x_i, d_k) = \frac{1}{\sigma\sqrt{2\pi}} \exp \frac{-1}{2}(\frac{d_k}{\sigma})^2$$

$$d_k = \sqrt{(x_{s_i} - x_{s_k})^2 + (y_{s_i} - y_{s_k})^2}, i \neq k$$

We needed to find a way to avoid needing 6,560 per shot. The solution we came up with was to slice the court into a fixed grid a priori. This requires calculating a OPPFGPct for each team a priori and storing a value of a 2D list under a key of *opponent + season* in a map-reduce job. Next the RDD containing a 2D list of opposing shot percentages for each season against each team can be joined to the RDD of shots on a temporary *opponent + season* key made for each shooter's shot. Then in a following map job the OPPFGPct can be added by simply indexing the 2D list in constant time. This technique is significantly more efficient for computation though we anticipate it is not as accurate as a Gaussian weighting scheme. The number of partitions the grid is cut into is a tuning parameter as more grids would mean each shots considers only more local shots and when larger it gets closer to the average shooting percentage against a team regardless of location. There is a caveat here with his feature and that is it has to use the labels to calculate as seen in the following equation.

$$OPPFGPct(X_{i,j}) = \frac{s_m}{s_a}$$

$$s_m, s_a \in X_{i,j}$$

Where s_m are the total shots made and s_a total shots attempted from within the same partition of the grid $X_{i,j}$.

Since the labels are used we do not explore very fine grids as this would certainly over-fit the results. Additionally, the grid sizes were selected to fit the 500x472 to best ensure the same number of pixels in each partition though not exactly equal.

6.3 Data Analysis

Plots in this section were all generated using 50,000 randomly sampled shots from the pool of over 4.5 million. Additionally, all work isn't shown for examining all the features we originally had as mentioned in the previous sections. This is because lots ended up being pretty useless; however, we were fortunate to have a few work out as we will discuss in this section and the next.

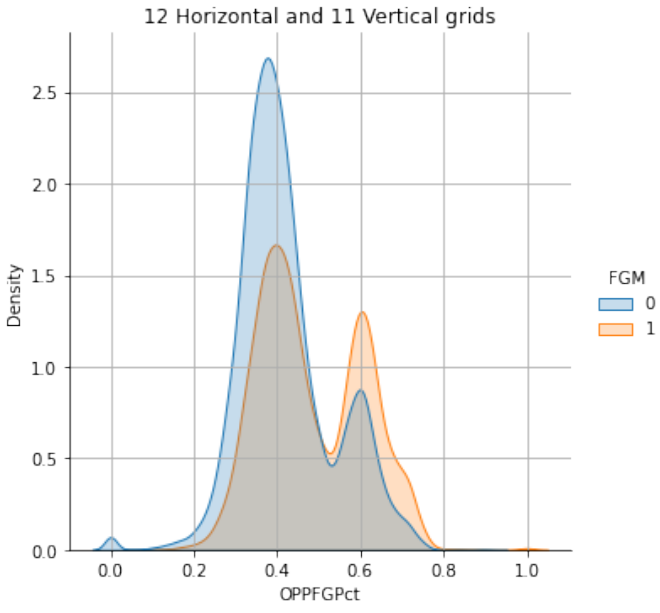


Figure 6: Distribution of OPPFGPct for grid partitioning of 12/11

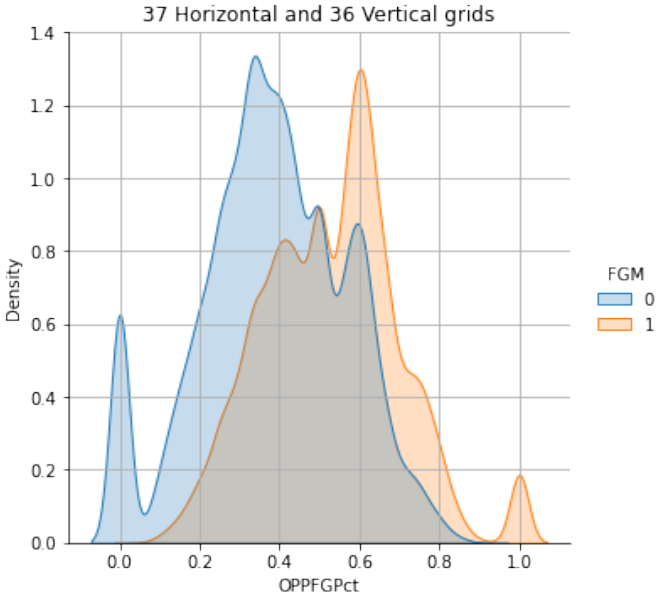


Figure 7: Distribution of OPPFGPctfor grid partitioning of 37/36

Comparing figures 6,7 the effects of increasing the grid resolution results in a cleaner separation of the *make* and *miss* distributions over OPPFGPct. We also have done this for partitioning of 18/17 and 29/28, and found this trend to be consistent. The take away here is that increasing the grid resolution leads to OPPFGPct being more dependent on shots that take place in closer proximity. This seems to better separate the distributions suggesting that shot location against particular teams is an important effect on the likelihood of scoring, which intuitively makes some sense.

Figure 10 (see below too large to fit here) is a pair-plot where *made* shots are coloured orange and *missed* shots are blue. The diagonal of the plot shows the distribution for the variable on the x-axis. The off-diagonal plots are scatter plots of their respective y and x axes. On the diagonal we see the distance distribution suggests when players are close they are more likely to score. Additionally on the diagonal the OPPFGPct plot shows some insight on how the shots where most people score from are generally correlated to a player also making a shot from that location. In the off-diagonal we see a high correlation between eFG%,2P% and eFG%,3P%, this is expected as eFG% is a combination of the other two. Interestingly 2P% and 3P% don't seem correlated, this may be explained by players who score a lot from within 5 ft such as centers but are poor marksman. A great example of such a player is Shaq with a career 3P% of 4.5% while having a 2P% of 58.3%. After reviewing all off-diagonal plots we anticipated OPPFGPct to be the best variable for predicting whether a player will make a shot or not as it separates the *makes* from the *misses* the most.

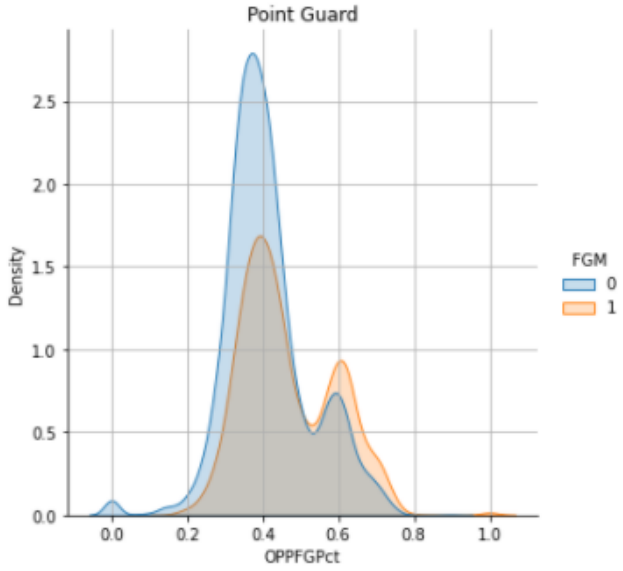


Figure 8: Distribution of OPPFGPct for Shots by Point Guards

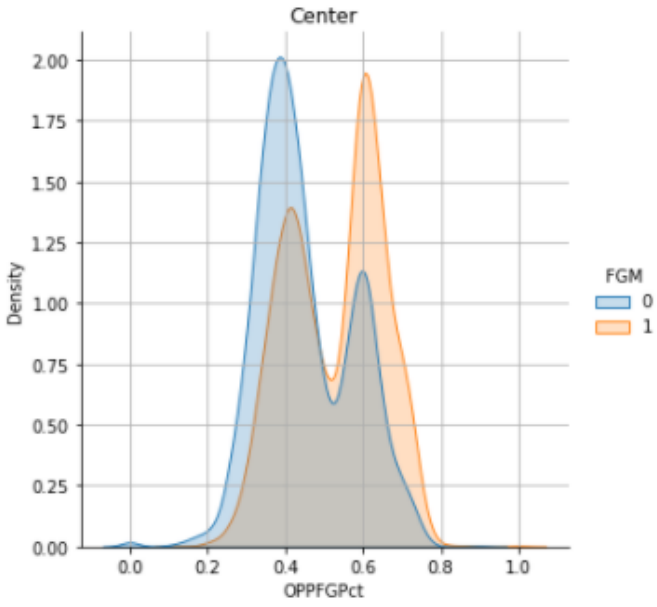


Figure 9: Distribution of OPPFGPct for Shots by Centers

Comparing figures 8,9 we observe that there is a interesting difference between the OPPFGPCt distributions of point guards and centers. This suggests when the OPPFGPct for a center is above 0.5 they are much more likely to score than a point guard with a OPPFGPct above 0.5. This may be motivation to build models for the two positions separately; we will explore this in the next section.

We did build a a random forest with positions one-hot encoded though it made no significant difference.

6.4 Models, Experiments, and Results

The models used were *Random Forest* (RF) with 200 trees and a max depth of 3 and *Logistic Regression* (LR) without any penalization as it didn't make any significant differences when validating. We chose these two models as RF can create non-linear decision boundaries and LR is limited to linear decision boundaries, therefore, if we saw RF significantly outperform LR it would indicate non-linear boundaries and models of higher complexity are likely to be useful. However, we did not find this as RF and LR are mostly within 1% of one another. The features selected to use were *GameScore*, *OPPFGPct*, *Distance*, and *eFG%*. Most of there features demonstrated marginal gains compared to naively guessing in the evaluation set alone, except OPPFGPct which provided a jump of 7% when alone, see table 2. These features did provide a slight increase when added to OPPFGPct. The OPPFGPct partitioned with 37 horizontal and 36 vertical grid sections was used (37/36) for the experiments in tables 2,3, and 5.

Features	LR Acc (%)	RF Acc (%)
GameScore	56.12	56.30
Distance	60.53	62.02
eFG%	55.59	55.48
OPPFGPct	65.62	65.70
OPPFGPct + Distance	65.85	65.65
All 4	**66.22**	**66.24**

Table 2: Feature ablation study (37/36)

The larger class proportion was 55.16% in table 2 as can be seen *GameScore* and *eFG%* make marginal gains alone although when *all 4* are considered the best results are yielded. Here a training and test split of 33k/17k were used.

train/test	LR Acc (%)	RF Acc (%)
670/330	62.73	62.73
6.7k/3.3k	66.06	**66.45**
67k/33k	66.16	66.08
670k/330k	**66.60**	66.44

Table 3: Accuracy of models with increasing dataset size (37/36)

Table 3 shows how the model accuracies change for LR and RF change as the dataset increases in size. As can be seen once the training set is in the tens of thousands the gain in accuracy is diminishing or in-existent. Due to this result in the following experiments we stay below 50k training examples for all model variations.

Grid Partitions	LR Acc (%)	RF Acc (%)	π_{max} (%)
12/11	62.24	62.67	55.32
18/17	63.60	63.58	55.29
29/28	64.37	64.54	55.18
37/36	**66.55**	**66.46**	55.62

Table 4: Accuracy of models for different grid partitions

Table 4 shows the accuracies for LR and RF trained on 2/3 of a randomly sampled 50,000 shots and evaluated against the remaining 1/3. The π_{max} represent the percent of the higher proportion class from the *make* or *miss* classes. This proportion provides an idea of what the accuracy would be given predicting solely the predominant class in the evaluation set. We observe the accuracy increases as anticipated from the discussion of figures 6 and 7 previously. The grid could be further partitioned, however then the results may be inaccurate in reality, since a single player's contribution to the feature becomes more prominent.

Positions	LR Acc (%)	RF Acc(%)	π_{max} (%)	train/test
PG	65.77	65.77	57.28	28k/14k
SG	66.66	66.94	57.27	32k/16k
SF	**67.35**	**67.05**	56.03	27k/13k
PF	66.45	66.63	54.22	28k/14k
C	65.65	66.08	**50.15**	21k/10k

Table 5: Accuracy of models for different positions (37/36)

Table 5 shows the varying accuracies of the two models when trained and tested solely on data of a single position. Both models exhibit their highest accuracies on predicting shots of a small forward (SF); however, it is important to look at the π_{max} as the difference between the accuracy and π_{max} for the SF is 11% while the difference between the center (C) and π_{max} is 15%. Since the larger the π_{max} is the easier it should be to achieve larger accuracies. This suggests predicting whether a shot will be *made* or *missed* is hardest for centers, however, the models relatively perform best at predicting their shots. Data for this task was selected from a randomly sampled pool of 200,000 samples then filtered for the desired position before training, the train/test column shows the splits for the training and evaluation.

We anticipate the accuracies are seemingly so low since in the machine learning problem $y = f(x) + \epsilon$, where y is the label and $f(x)$ is the model we wish to learn. The best possible error rate is called the Bayes error rate and is dependent on the magnitude of noise induced by ϵ. We believe the ϵ is most likely large in this problem since players even in practice don't consistently *make* or *miss* shots. For example if a player had a ground truth of even an 80% shot making regardless of position and all other factors the best accuracy would only be 80%. This is not too say we reached the Bates error rate; I am optimistic there is more data and thereby features to improve our results. The results from this work simply show we can most definitely do better than random guessing based on the prior.

7 CONCLUSION

Throughout this project we faced many challenges from learning to web-scrape, how to best store our data, when to use python or spark, and finally how to implement the spark tasks. Additionally, we got to design our own "data pipeline" all the way from the raw source being BR to the downstream tasks. We learned that cleaning the data to avoid transformations *raising errors* takes up the majority of the time and learning how to effectively test spark code saves a TON of time. Moreover, We analyzed NBA players' heights weights and investigated the correlations between players' career statistics and their heights and weights. The predictive model building was a lot of fun and the feature developed seemed promising as a predictor. The "played-with" graph was very interesting to explore, especially once we put the graph into a graph visualiser such as Gephi. Computing statistics on the "played-with" graph gave new insights on players that we never thought of before.

8 FUTURE WORK

All tasks could benefit from more data being collected by developing more URL lists and corresponding spark code to extract data. In player analysis, We only discussed correlations between two variables. However, we can extend the studies to correlations between multiple variables which would show more insights.

Having discovered that the accuracy of the model saturates around $\tilde{1}$0,000 training samples it seems possible that the Gaussian technique of estimating the OPPFGPct could be performed for a subset of data. Another avenue to explore in the future is to train the model with shot data from one time period and then test it on data from another to see how the models performance will respond. Additionally, during the addition of OPPFGPct we further explore a way to remove the current shooters shot from the aggregate, this would then allow us to explore finer grid resolutions with no worry of actually over-fitting. Like other tasks this task would surely benefit from more data to try and create more features.

For graph analysis, more work can be done by creating different graphs. Another interesting idea is to have a directed edge uv if player u has beat v in a game. Some interesting insight can arise from this graph, such as the player with has beat/lost to the most players. Finding different approaches on how to construct a graph would be an interesting task for the future.

Figure 10: Pair plot of the main features