6202 Final Project Report - Group 4

Divya Parmar, Pranay Bhakthula
The George Washington University

June 23, 2021

Table of Contents

Introduction

Kobe Bryant is widely regarded as one of the greatest National Basketball Association players of all-time. He is renowned for his work ethic, competitiveness, and ability to deliver in a game's most pressure-packed moments. Bryant won five NBA championships, was an 18-time NBA All-Star, and led the NBA in scoring twice. Kobe Bryant transcended the game of basketball and became a worldwide icon. The world was stunned when he tragically passed away in 2020.

As massive basketball fans ourselves, we were excited when we found a Kaggle dataset containing all shots taken by Bryant during a regular season or post-season game. The Kaggle dataset has information about the shot attempt (location, game period, made/missed, etc.), and we decided we could apply our machine learning knowledge to mine this data.

Our problem statement is to classify whether Bryant made or missed a given shot, based on variables provided about the shot attempt. This will both give us insight into Bryant's play style, but also about how to understand NBA player behavior more generally.

The dataset contains categorical variables (such as shot zone), and numeric variables (shot distance). This allows for us to practice pre-processing and scaling that comes with most neural network use cases. Furthermore, the categorical variables will provide the basis for our exploratory data analysis and help us select features to put into the model.

Dataset Description

This dataset comes from Kaggle, and it was very thorough with its information. There were 30,697 rows and 25 columns. These are the columns in our dataset.

Column	Data type	Description
action_type	object	Action taken as shot was released (running, turning around, etc)
combined_shot_type	object	More simplified version of shot attempt from previous column
game_event_id	int64	ID for game event
game_id	int64	ID which is unique for the game
lat	float64	Latitude of where the game was held
loc_x	int64	X coordinate on the court of where the shot was taken
loc_y	int64	Y coordinate on the court of where the shot was taken
lon	float64	Longitude of where the game was held
minutes_remaining	int64	Minutes remaining in the period of the game when shot was taken
period	int64	Period of the game
playoffs	int64	1 if game was in playoffs, 0 otherwise
season	object	Season in which game took place
seconds_remaining	int64	Seconds remaining in the period when shot was taken
shot_distance	int64	Distance of shot from the basket
shot_made_flag	float64	Whether the shot was made or not, 1.0 if made, 0.0 if missed
shot_type	object	Whether it was a two pointer or three pointer
shot_zone_area	object	Left side of court, right side of court, center, etc.
shot_zone_basic	object	In the paint, mid-range, three pointer, etc.
shot_zone_range	object	Less than 8 ft, 8-16 ft, etc. (categorical version of shot distance)
team_id	int64	ID of team he was playing for (Los Angeles Lakers)
team_name	object	Name of team he was playing for (Los Angeles Lakers)
game_date	object	Date when shot was taken
matchup	object	Matchup, ie (LAL @ DEN)
opponent	object	Name of opponent
shot_id	int64	Unique ID for the shot attempt

The data was extremely clean, with no missing values except for “shot made flag” in which 5,000 rows had withheld values for the Kaggle competition. We dropped those 5,000 rows and continued on with 25,697 rows for our modeling.

shot_made_flag	**Value Count**
0.0 (missed shot)	14,232
1.0 (made shot)	11,465
Nan (for Kaggle competition, we dropped these records)	5,000

We chose shot_made_flag as our target variable, as it fits our goal of classifying shots as made or missed (which was also the intent of the Kaggle competition).

For our analysis, we used a metric called *field goal percentage*. This can be applied to the overall dataset or subsets of the data, such as field goal percentage on jumpshots.

$$Field\ Goal\ Percentage\ = (Made\ Field\ Goals)\ /\ (Total\ Field\ Goals\ Attempted)$$

Exploratory Data Analysis

Overall Exploration

To understand Bryant's shooting, we overlaid his shot attempts on an image of a basketball court (Figure1). To do this, we used the loc_x and loc_y values in the dataset. The green dots are made shots, and the red dots are missed shots. The basket Bryant is shooting at is at the bottom center. We can also see the court lines such as the restricted area and three point area in our plot.

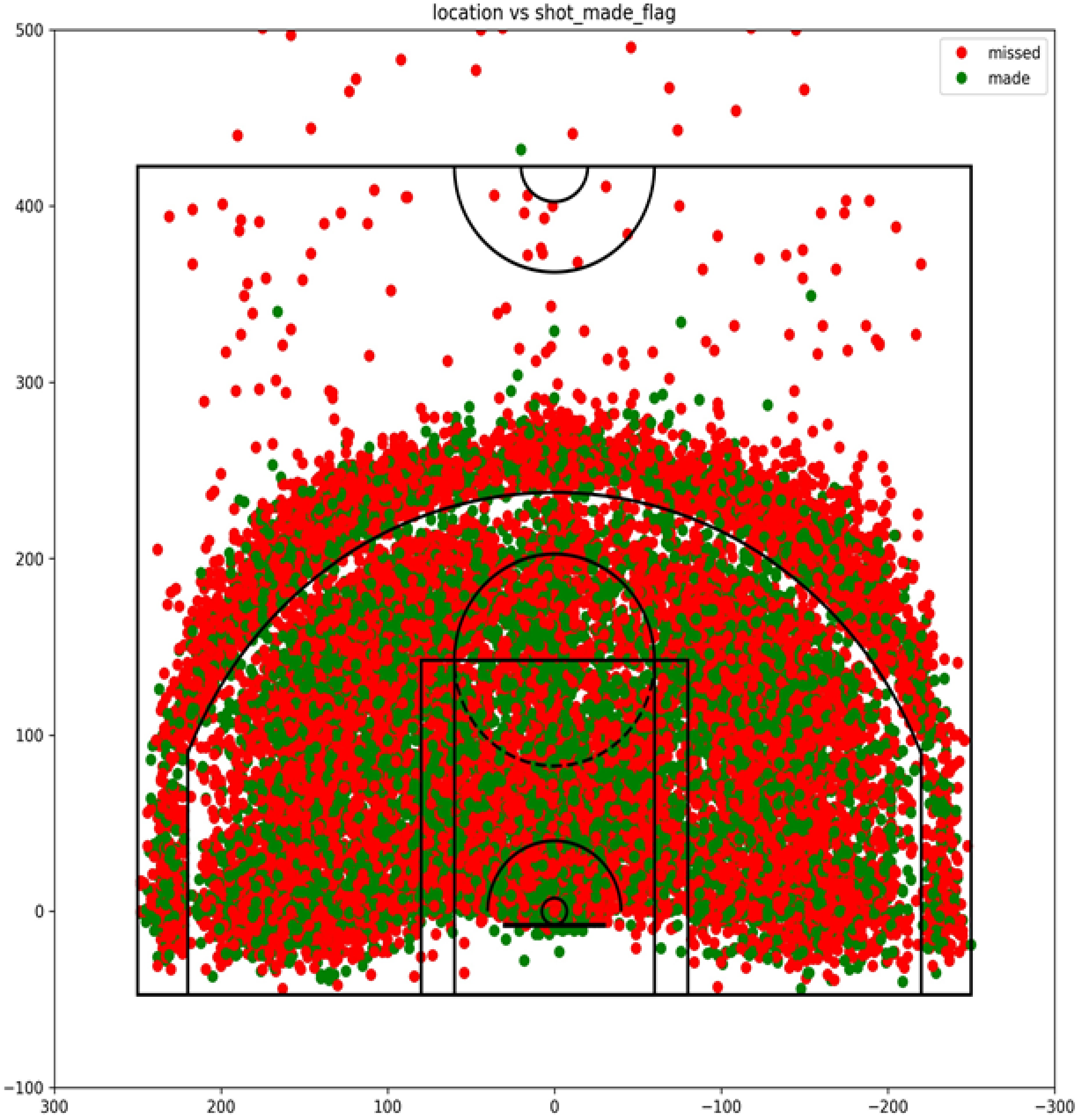


Figure1

To find more insight in the shot locations, we plotted hexagons that clustered nearby shots together (Figure2). The greener the hexagon, the higher percentage of shots Bryant made in that clustered area, with redder areas signifying a lower made percentage. We can see that closer shots are made at a higher rate than farther shots, although this is not extremely surprising. This metric is called *field goal percentage*.

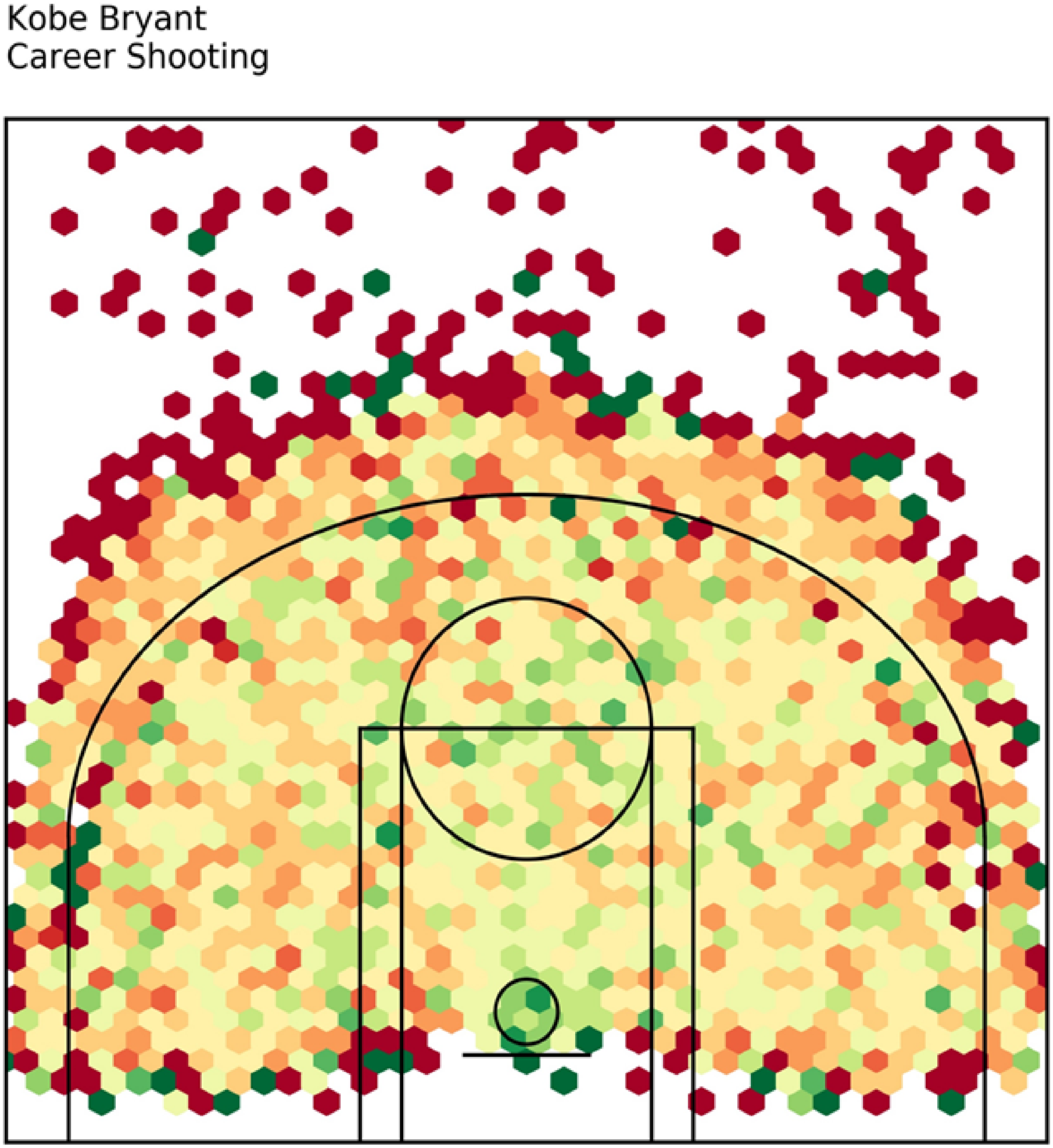
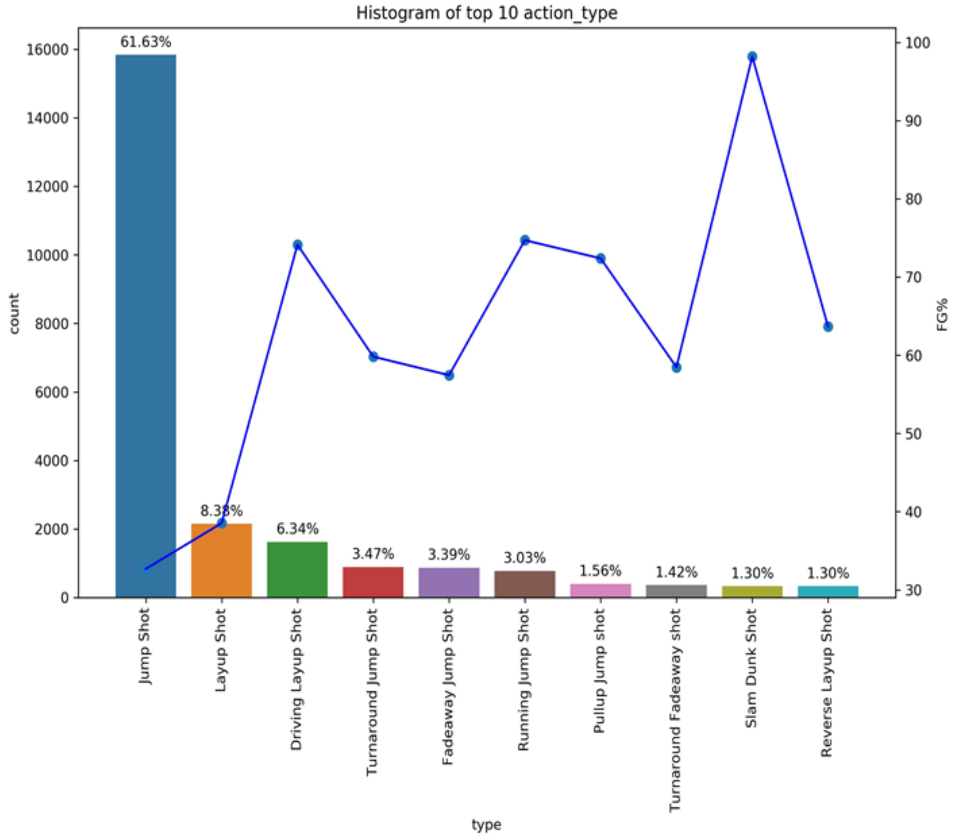


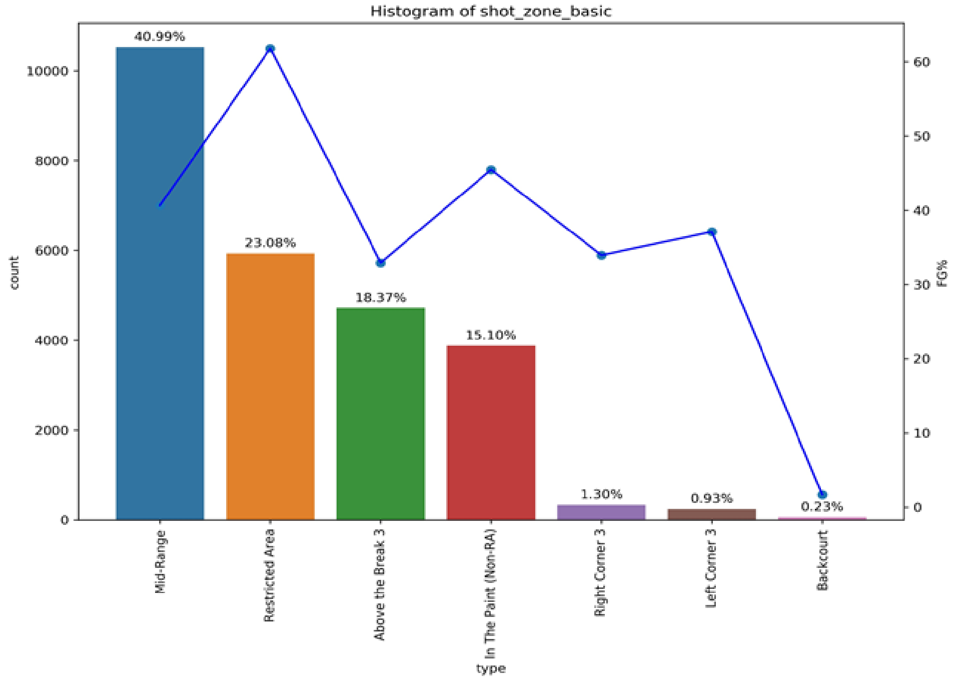
Figure2

Next, we wanted to see how many shots Bryant took across categories of different variables, and his shot efficiency in those categories.



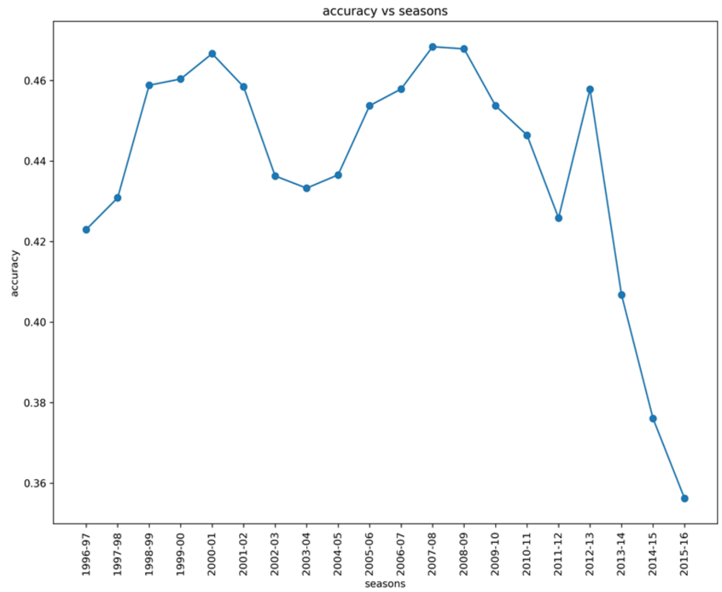
First, we can look at action_type, and we see there are many different values. We also see that Bryant’s field goal percentage (made shots as percentage of total shots) across these types varies greatly, meaning that this variable may have predictive power in our model.

Figure3



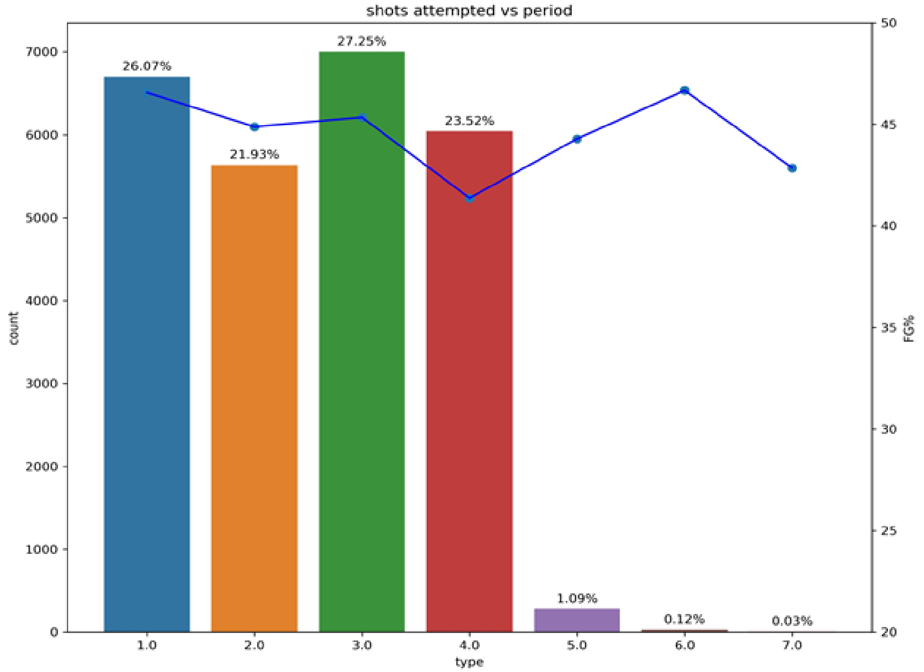
Next, we looked at the shot_zone_basic variable. Bryant is fairly balanced across zones, but his field goal percentage varies widely. This is another good feature to use in our model.

Figure4



Furthermore, we looked at the season variable. Bryant’s field goal percentage shows clear variation across seasons, meaning that we want to use this feature in the model.

Figure5



Lastly, we looked at the period variable, which refers to the quarter of the game. Bryant’s field goal percentage varies across periods, and we want to include this variable in our models.

Figure6

Additional EDA plots can be found in the appendix.

Opponent Analysis

Note: This analysis is not part of the EDA leading to modelling. This is a separate analysis to analyze Bryant's play from the opponent team's point of view in two of the championship years. We aim to demonstrate our curiosity about this dataset and the insights it holds.

We wanted to understand how Bryant played against specific teams in a couple of championship seasons (2002 and 2009), and why he played well or poorly. We chose these years specifically because in the 2002 season he had as a teammate another NBA legend, Shaquille O’Neal, who was more dominant in that year than Bryant. In contrast, in 2009, O’Neal was not on the team and Bryant was the most dominant player on his team.

In the season 2001-02, Bryant scored the most points against the Memphis Grizzlies, and he scored the least against the Charlotte Hornets.

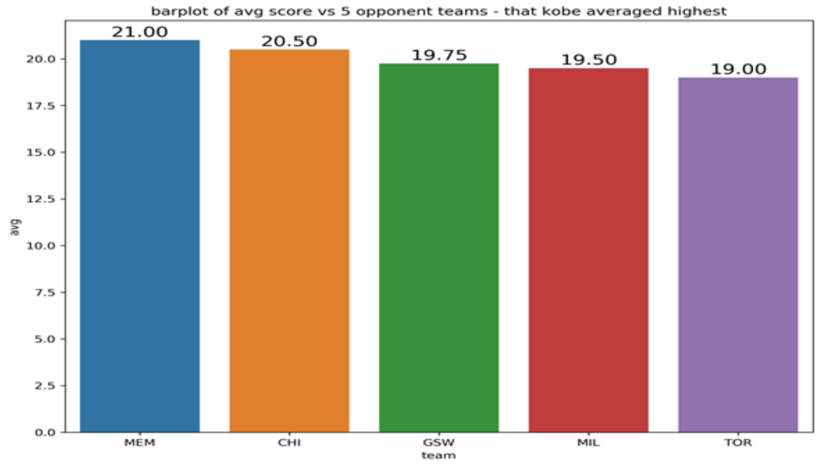


Figure7

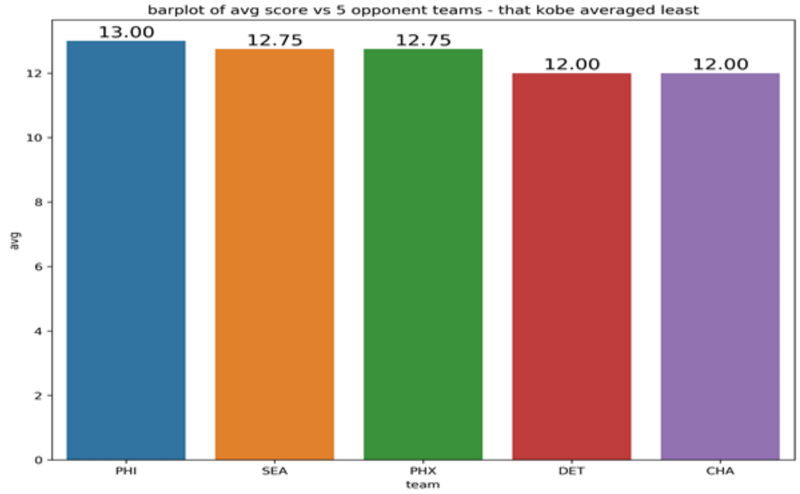
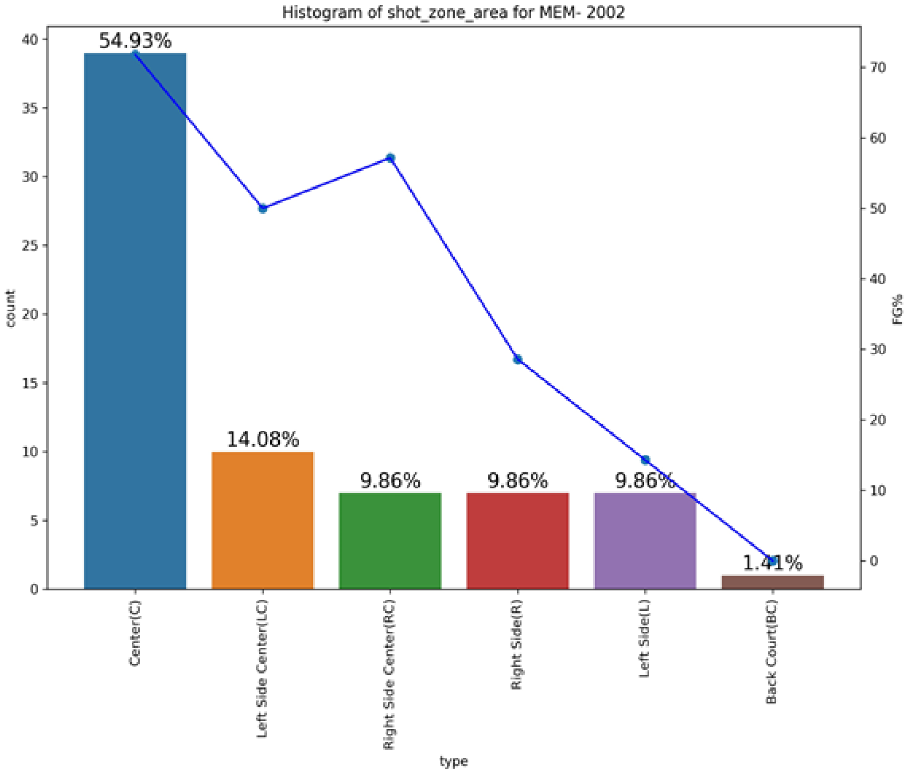


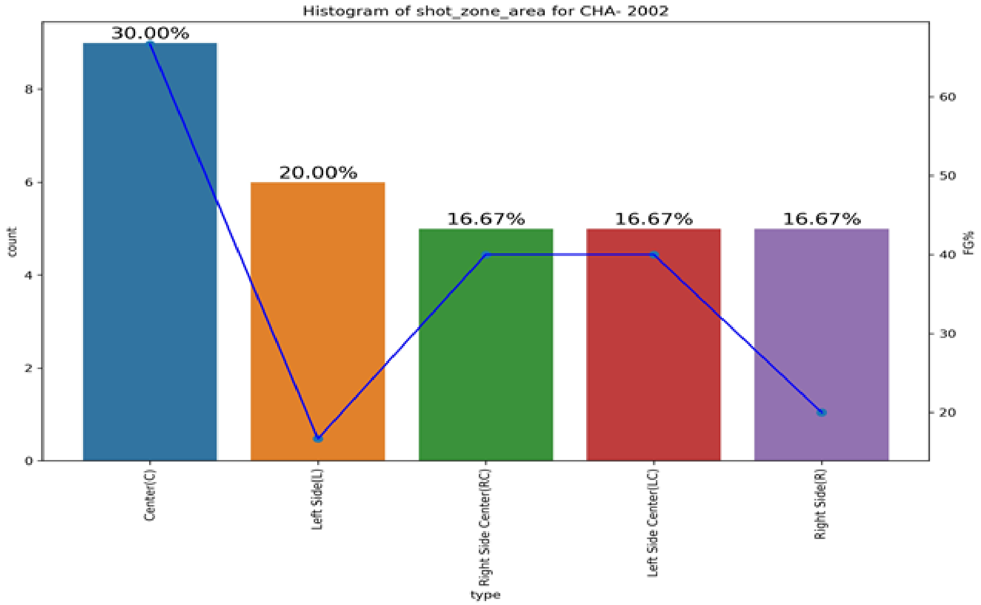
Figure8

Next, we look at the shot_zone_area differences between his best opponents and worst opponents.



Against Memphis, his best opponent.

Figure9



Bryant shoots much less from the center zone area against Charlotte. His field goal percentage is highest in the center zone, so that means he is getting higher quality shot attempts against Memphis.

Figure10

Defenses may want to push Bryant toward the right side, where he shoots a worse percentage.

When we plot this visually, we can see that Bryant shot more from the center and the restricted area against Memphis (Figure11), and there are many green points indicating hits. Against Charlotte (Figure12), there were more shots from the sides, which were missed.

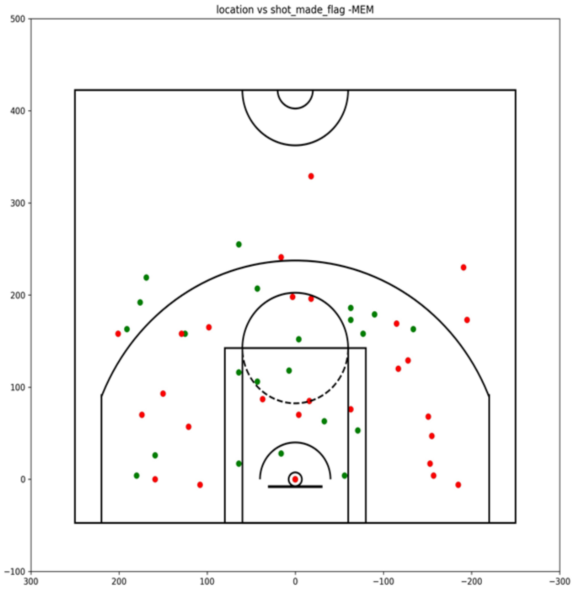


Figure11

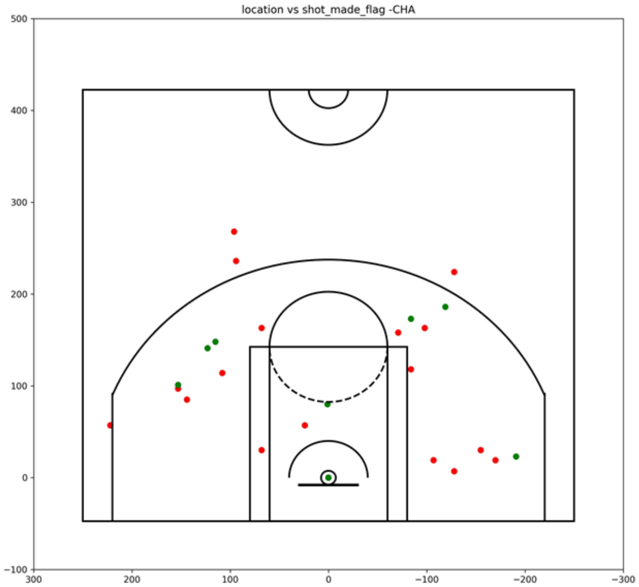


Figure12

Next, we looked at the 2008-09 season, in which he did not have O’Neal as a teammate. Bryant scored the most points against the New York Knicks, and he scored the least against the Washington Wizards.

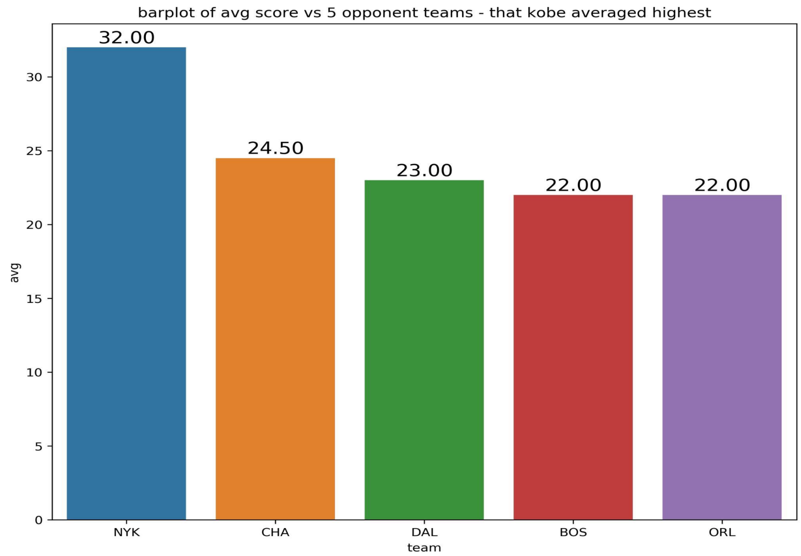


Figure13

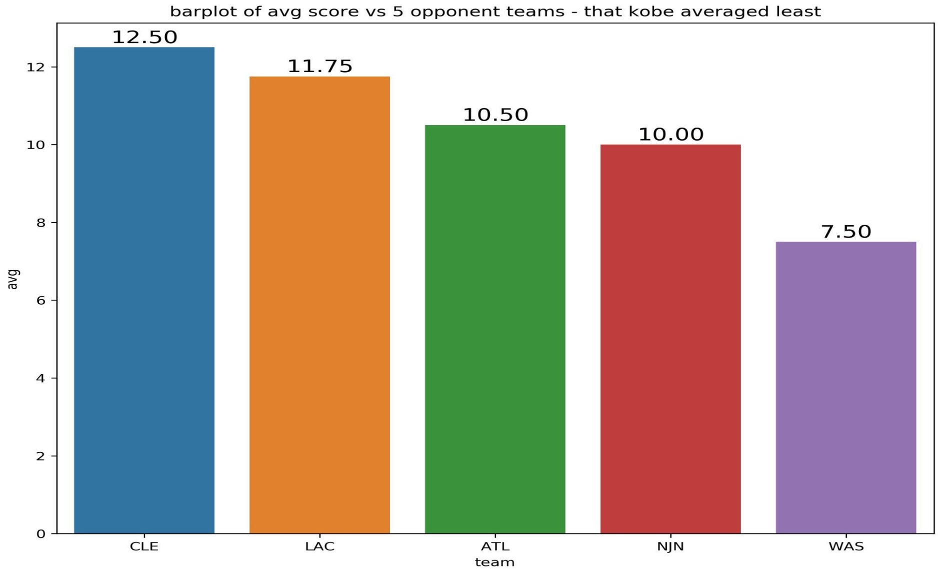


Figure14

Next, we look at the shot_zone_area differences between his best opponents and worst opponents.

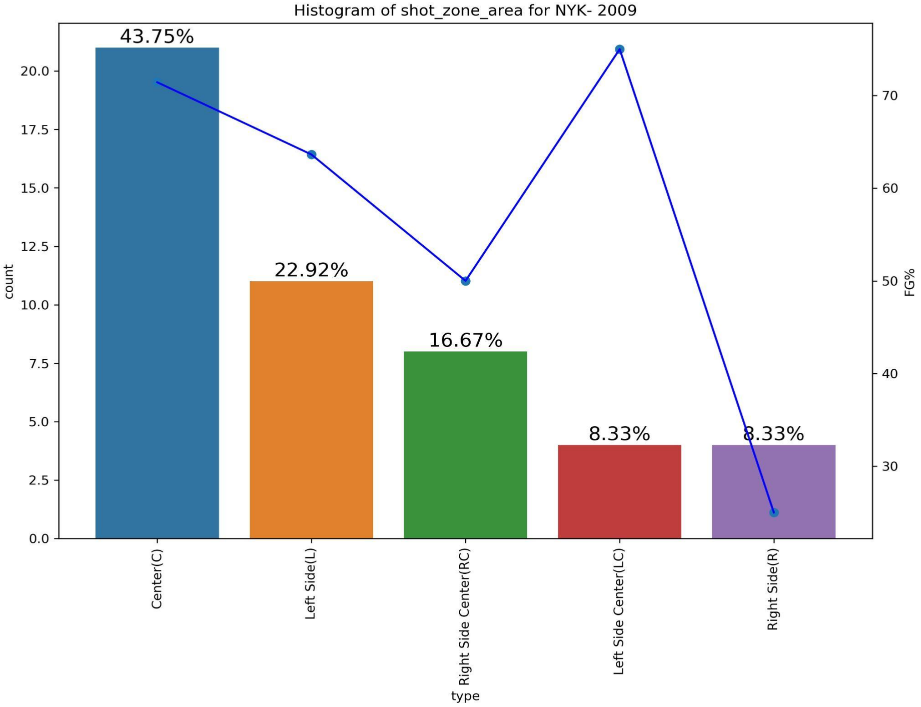


Figure15

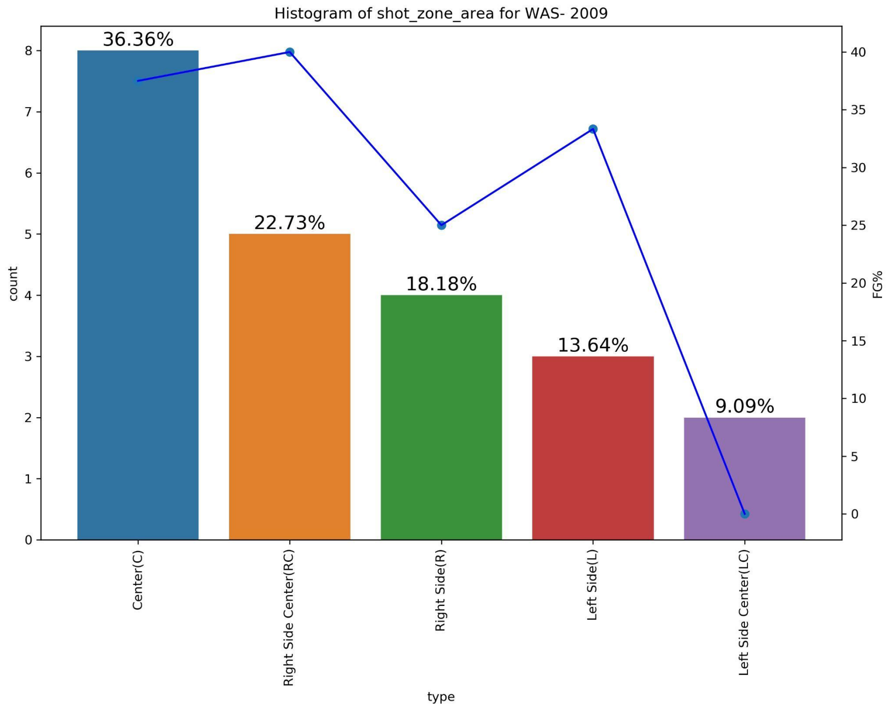


Figure16

Against the New York Knicks, Bryant shot most of his shots from center and left side. which were most effective. Against the Washington Wizards, he shot from center the most but next best shots were from the right side. So we can say Bryant is less effective from the right side than from the left side.

So defenses may want to push Bryant toward the right side, where he shoots a worse percentage.

When we plot this visually, we can see that Bryant shot more from the center and the In the paint area against knicks (Figure17), and there are many green points indicating hits. Against Wizards (Figure18), there were more shots from the sides, which were missed.
So in order to defend better against Bryant, we need to push him to the right side or make him shoot from longer distances.

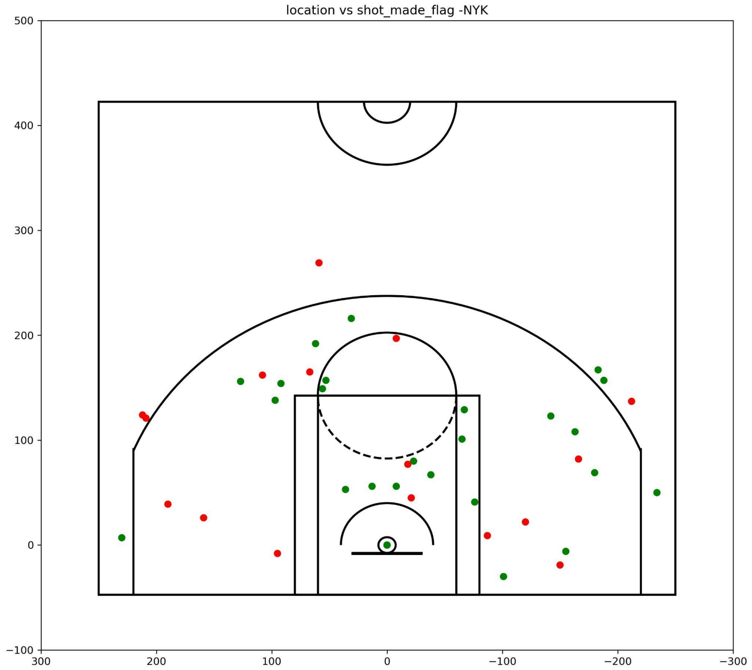


Figure17

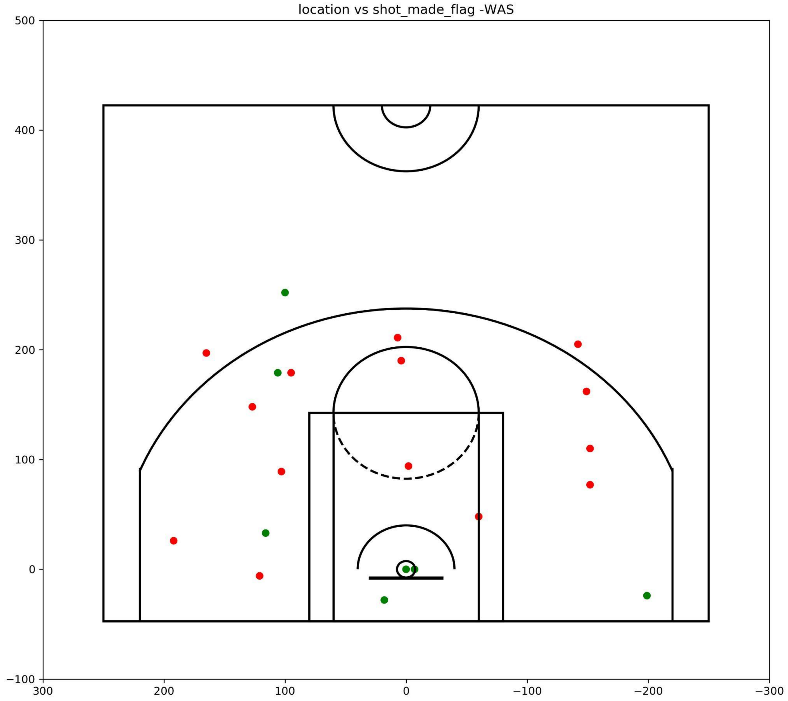


Figure18

Now we will look into the modelling part of the project and find the best model to predict the shot_made_flag. The variables which are of importance are chosen from the EDA analysis done before the opponent analysis part.

Machine Learning Algorithms

Multi-Layer Perceptron -- [Source: NNDesign, Martin Hagan]
As this was a classification problem, we chose a mutli-layer perceptron (MLP) for our neural network architecture. A perceptron is a single neuron with a weight, bias, and hardlims transfer function (see Figure13). A perceptron can create a classification decision boundary (see Figure14).

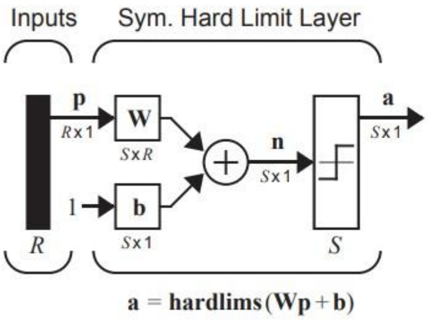


Figure 3.1 Single-Layer Perceptron

Figure19

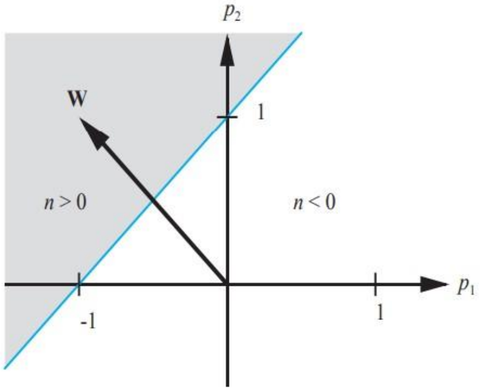


Figure 3.3 Perceptron Decision Boundary

Figure20

The MLP architecture below allows us to create intricate decision boundaries and classify complex datasets. [Source: researchgate.net, Mohamed Zahran]

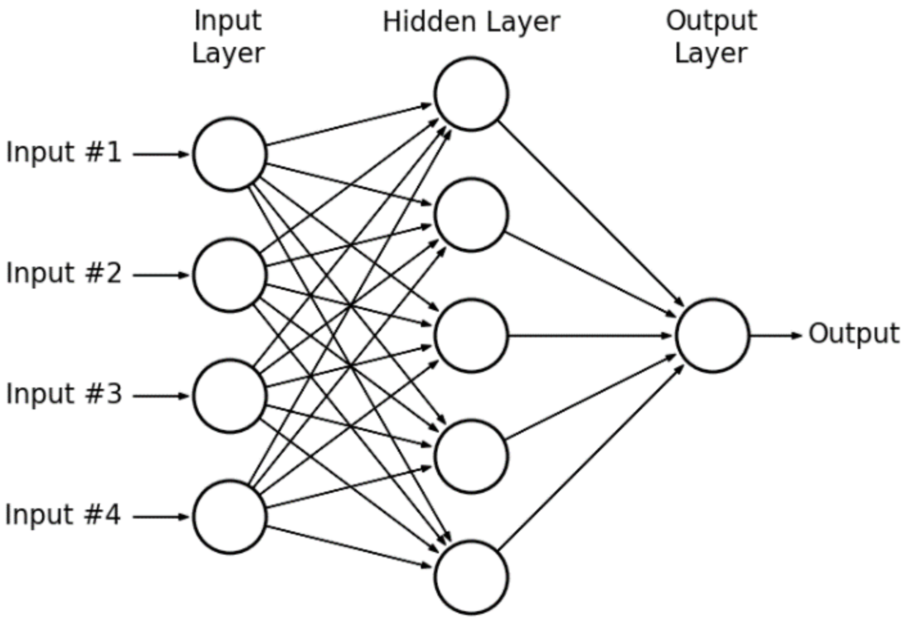


Figure21

Random Forest -- [Source: mygreatlearning.com, Great Learning Team]
We decided to compare our neural network to a classical model, as we want to see if these newer techniques can perform better than established mathematical models.

The Random Forest (RF) classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction. Moreover, a random forest technique has a capability to focus both on observations and variables of a training data for developing individual decision trees and take maximum voting for classification and the total average for regression problem respectively. It also uses a bagging technique that takes observations in a random manner and selects all columns which are incapable of representing significant variables at the root for all decision trees. In this manner, a random forest makes trees only which are dependent on each other by penalising accuracy. We have a rule of thumb which can be implemented for selecting sub-samples from observations using random forest. If we consider 2/3 of observations for training data and p be the number of columns then:

1. For classification, we take sqrt(p) number of columns
2. For regression, we take p/3 number of columns.

The above thumb rule can be tuned in case of increasing the accuracy of the model.

Random Forest pseudocode:

```
1. Randomly select "k" features from total "m" features.
    a. Where k << m
2. Among the "k" features, calculate the node "d" using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat 1 to 3 steps until "l" number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees.
```

Model Metrics

We looked at the following metrics in deciding how effective our models were:

Precision

Precision measures the rate of false positives [blog.floydhub.com].

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Figure22

Recall

Recall measures the rate of false negatives [blog.floydhub.com].

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Figure23

F1-score

F1-score weights precision and recall [blog.floydhub.com].

$$F1 - Score = 2 * \frac{Recall * Precision}{Recall + Precision}$$

Figure24

Accuracy

Accuracy is simply what percentage of observations were classified correctly [blog.floydhub.com].

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative}$$

Figure25

Mean Squared Error

MSE compares the value of the model output to the true target value (NNDesign).

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t-\mathbf{x}^T\mathbf{z})^2]$$

Figure26

ROC AUC

This curve measures the tradeoff between specificity and sensitivity [blog.floydhub.com].

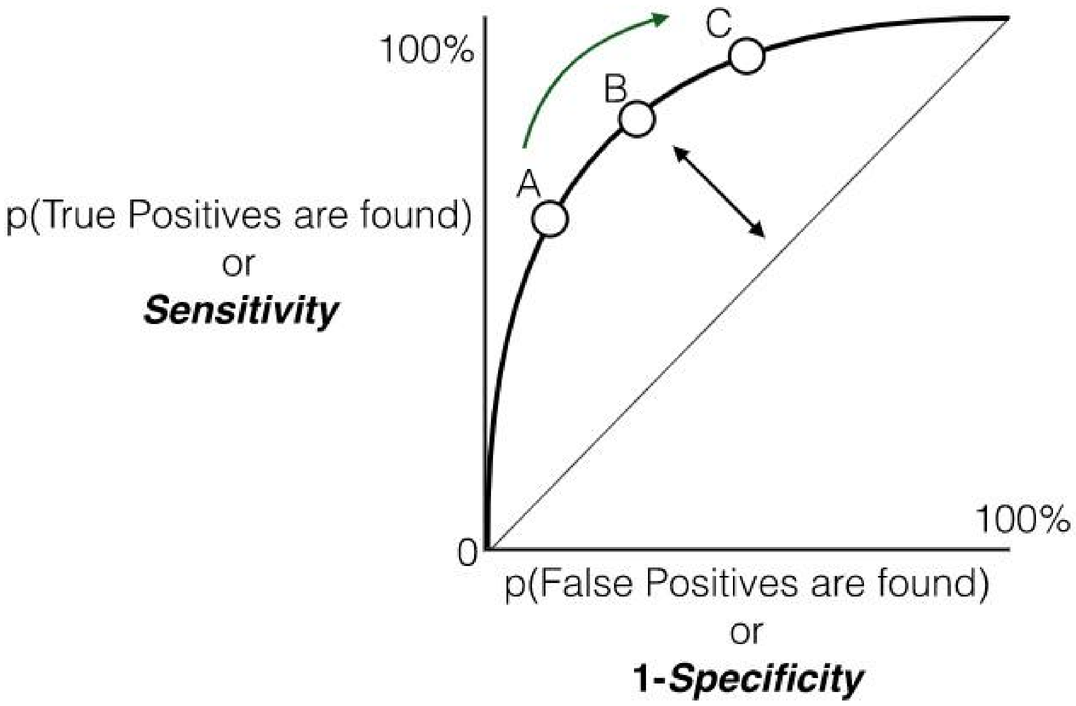


Figure27

Experimental Setup

Pre-Processing

First, we had to pre-process our data to prepare it for the model.

To start, we dropped columns such as shot_id, latitude/longitude, seconds remaining, and the opposing team which either provided no information or had high cardinality.

The remaining variables that made our model are:

- 'action_type'
- 'Combined_shot_type'
- 'period'
- 'playoffs'
- 'season'
- 'shot_distance'
- 'shot_type'
- 'shot_zone_area'
- 'shot_zone_basic'
- 'shot_zone_range'

Next, we transformed features as needed to prepare for the model.

- We cut off outliers in shot_distance by setting the max distance to 40 ft
  ```
  data1['shot_distance'] = np.where(data1['shot_distance'] >= 40, 40, data1['shot_distance'])
  ```
- We transformed shot_zone_area into the correct alphabetical order for the label encoder variable. We want the order to be Left Side → Center → Right Side
  ```
  data1['shot_zone_area'].replace('Center(C)', 'CC', inplace=True)
  data1['shot_zone_area'].replace('Right Side Center(RC)', 'DD', inplace=True)
  data1['shot_zone_area'].replace('Right Side(R)', 'EE', inplace=True)
  data1['shot_zone_area'].replace('Left Side Center(LC)', 'BB', inplace=True)
  data1['shot_zone_area'].replace('Left Side(L)', 'AA', inplace=True)
  ```
- We also transformed other variables for alphabetical order purpose
- We label encoded and min-max scaled all our features.

We completed our pre-processing by label encoding our target variable (shot_made_flag).

Modelling

Initially, we wanted to compare a neural network to a classical model. Therefore, we created two random forest classifiers and two multi-layer perceptron classifiers:

- Model 1 - Random forest on all features
- Model 2 - Random forest on top six features (found with feature importance in full model)
- Model 3 - Multi-layer perceptron on all features
- Model 4 - Multi-layer perceptron on top six features (found in random forest model)

Next, we focused solely on the multi-layer perceptron. We compared the performance of the default MLPClassifier library in sklearn versus changing specific parameters. Here are the parameters we modified:

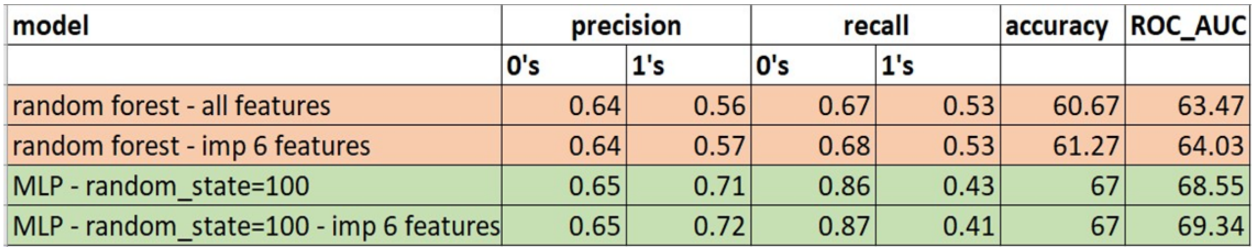
Parameter	**Default**	**Other options tested**
activation	relu	logistic
hidden_layers	100	(100 , 100), (100 , 100 , 100), (100 , 100 , 100 , 100)
solver	adam	sgd
momentum	0.9	0.95
early_stopping	False	True

Figure28

Using these parameters, we created 10 additional models. We compared the 11 models to see how they performed relative to each other.

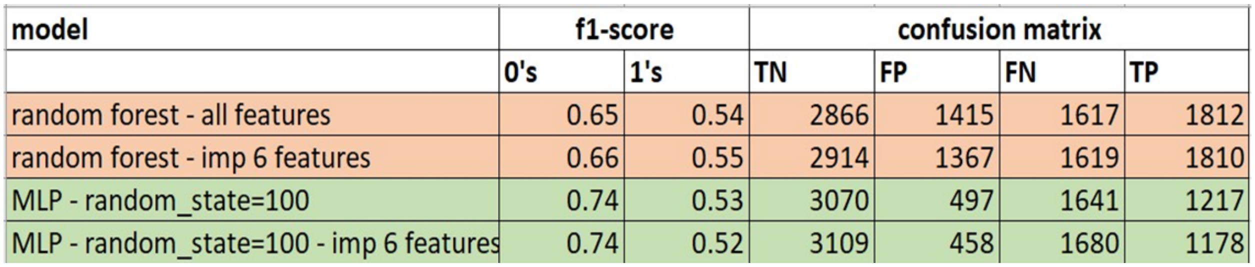
Results

In comparing metrics for our two random forest models versus multi-layer perceptrons, the MLP model performs better across most metrics such as AUC, accuracy, and precision. The only area in which RF does better is recall for our positive class (made shot). RF classifies more positives in general, and seems to do better at avoiding false negatives.



model	precision		recall		accuracy	ROC_AUC
	0's	1's	0's	1's		
random forest - all features	0.64	0.56	0.67	0.53	60.67	63.47
random forest - imp 6 features	0.64	0.57	0.68	0.53	61.27	64.03
MLP - random_state=100	0.65	0.71	0.86	0.43	67	68.55
MLP - random_state=100 - imp 6 features	0.65	0.72	0.87	0.41	67	69.34

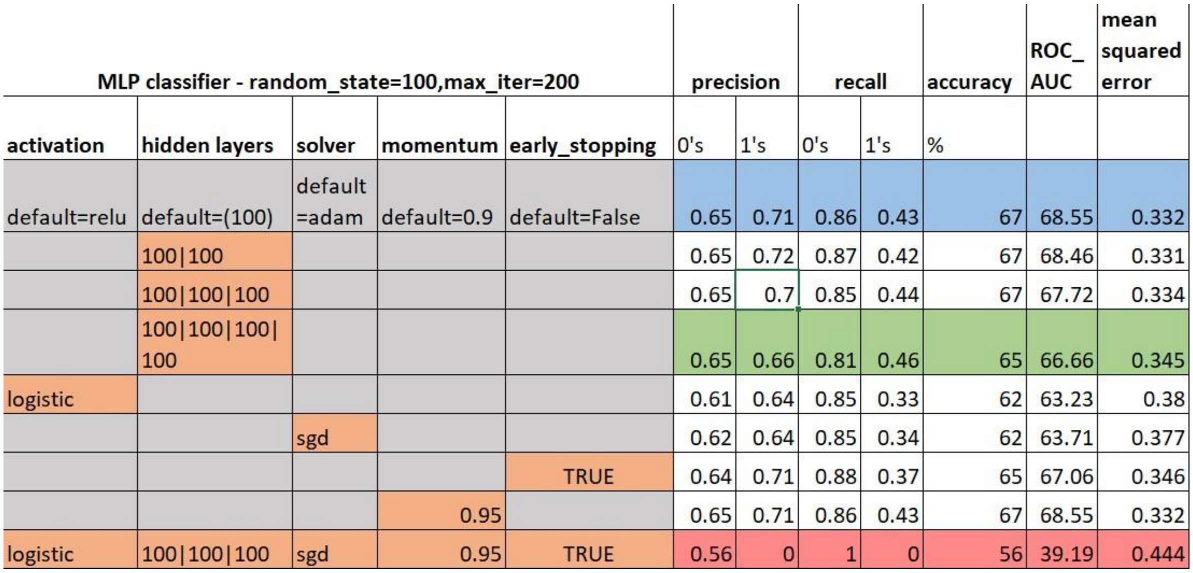
Figure29



model	f1-score		confusion matrix			
	0's	1's	TN	FP	FN	TP
random forest - all features	0.65	0.54	2866	1415	1617	1812
random forest - imp 6 features	0.66	0.55	2914	1367	1619	1810
MLP - random_state=100	0.74	0.53	3070	497	1641	1217
MLP - random_state=100 - imp 6 features	0.74	0.52	3109	458	1680	1178

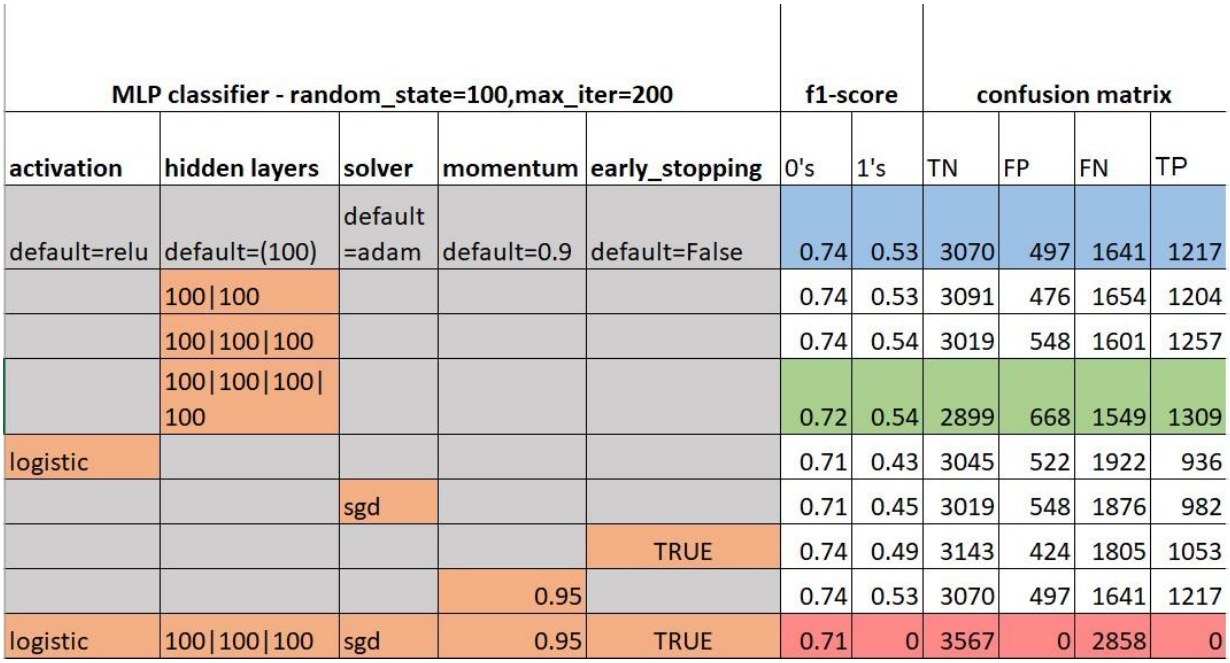
Figure30

Next, we have only the neural network with parameter changes across models. For parameters, grey indicates where we used the default value, and orange indicates that the given parameter was specified as non-default.



MLP classifier - random_state=100,max_iter=200					precision		recall		accuracy	ROC_AUC	mean squared error
activation	hidden layers	solver	momentum	early_stopping	0's	1's	0's	1's	%		
default=relu	default=(100)	default =adam	default=0.9	default=False	0.65	0.71	0.86	0.43	67	68.55	0.332
	100|100				0.65	0.72	0.87	0.42	67	68.46	0.331
	100|100|100				0.65	0.7	0.85	0.44	67	67.72	0.334
	100|100|100|100				0.65	0.66	0.81	0.46	65	66.66	0.345
logistic					0.61	0.64	0.85	0.33	62	63.23	0.38
		sgd			0.62	0.64	0.85	0.34	62	63.71	0.377
				TRUE	0.64	0.71	0.88	0.37	65	67.06	0.346
			0.95		0.65	0.71	0.86	0.43	67	68.55	0.332
logistic	100|100|100	sgd	0.95	TRUE	0.56	0	1	0	56	39.19	0.444

Figure31



MLP classifier - random_state=100,max_iter=200					f1-score		confusion matrix			
activation	hidden layers	solver	momentum	early_stopping	0's	1's	TN	FP	FN	TP
default=relu	default=(100)	default =adam	default=0.9	default=False	0.74	0.53	3070	497	1641	1217
	100|100				0.74	0.53	3091	476	1654	1204
	100|100|100				0.74	0.54	3019	548	1601	1257
	100|100|100|100				0.72	0.54	2899	668	1549	1309
logistic					0.71	0.43	3045	522	1922	936
		sgd			0.71	0.45	3019	548	1876	982
				TRUE	0.74	0.49	3143	424	1805	1053
			0.95		0.74	0.53	3070	497	1641	1217
logistic	100|100|100	sgd	0.95	TRUE	0.71	0	3567	0	2858	0

Figure32

The base model (highlighted in blue) does quite well across metrics. Adding more layers generally seems to help, and our four-layer model (highlighted in green) does the best with recall of our positive class.

In contrast, models with the logistic activation function did poorly. They did poorly in classifying the positive class, and the final model predicted negative for all values. The two logistic models had the highest mean squared error across the entire set.

One possibility is that, by stopping at 200 iterations, we did not allow our more complex MLP models to converge. However, even increasing the iterations up to 10,000 did not change our MLP results.

Summary

For Random Forest model: (before and after feature reduction)

1. Accuracy of the model before feature reduction = 60.67% and after feature reduction accuracy = 61.27%.

2. From the classification report of models:

    1. Before feature reduction, F1 score for 0's = 0.65 and f1 score for 1's = 0.54 and After feature reduction, F1 score for 0's = 0.66 and f1 score for 1's = 0.55.

    2. Before feature reduction, Precision for 0's = 0.64 and precision for 1's = 0.56 and After feature reduction, Precision for 0's = 0.64 and precision for 1's = 0.57.

    3. Before feature reduction, Recall for 0's = 0.67 and recall for 1's = 0.53 and After feature reduction, Recall for 0's = 0.68 and recall for 1's = 0.53

3. Area under the curve for the model before feature reduction, AUC = 63.47% and for the model after feature reduction, AUC = 64.03%, A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.

4. From confusion matrix, for the model before feature reduction, TN=2866 and TP=1812 and for the model after feature reduction, TN=2914 and TP=1810.

We can observe that f1-scores, precision and recall rate is high for 0's in this model. The reason is because there are more number of observations which have 0's in target variable than those which have 1's as we have observed the histogram of target variable in EDA analysis.

The model after feature reduction has better performance than the model with all the features so we will choose this model as the best random forest model to which we compare default MLP model.

For Multilayer Perceptron(MLP) model – default model: (before and after feature reduction)

1. Accuracy of the model before feature reduction = 67% and after feature reduction accuracy = 67%.

2. From the classification report of models:

    1. Before feature reduction, F1 score for 0's = 0.74 and f1 score for 1's = 0.53 and After feature reduction, F1 score for 0's = 0.74 and f1 score for 1's = 0.52.

    2. Before feature reduction, Precision for 0's = 0.65 and precision for 1's = 0.71 and After feature reduction, Precision for 0's = 0.65 and precision for 1's = 0.72.

    3. Before feature reduction, Recall for 0’s = 0.86 and recall for 1’s = 0.43 and After feature reduction, Recall for 0’s = 0.87 and recall for 1’s = 0.41.

3. Area under the curve for the model before feature reduction, AUC = 68.55% and for the model after feature reduction, AUC = 69.34%, A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.

4. Mean squared error for the model before feature reduction = 0.332 and for the model after feature reduction = 0.332.

5. From confusion matrix, for the model before feature reduction, TN=3070 and TP=1217 and for the model after feature reduction, TN=3109 and TP=1178.

we can observe that all the metrics show better results in the default MLP model before feature reduction than random forest model. Since our focus in the project is more on TPs, the MLP model has low TP values compared to random forest model and MLP model after feature reduction.

For Multilayer Perceptron(MLP) model – hidden layer = (100,100,100,100) model:

This model has the highest TP value among all the MLP model we used.

1. Accuracy of the model = 65%

2. From the classification report of models:

    1. F1 score for 0’s = 0.72 and f1 score for 1’s = 0.54

    2. Precision for 0’s = 0.65 and precision for 1’s = 0.66

    3. Recall for 0’s = 0.81 and recall for 1’s = 0.46

3. Area under the curve for the model AUC = 66.66%. A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.

4. Mean squared error = 0.345.

5. From confusion matrix, TN=2899 and TP=1309

Though the most performance metrics shows decrease in performance this model has the highest among all the MLP models used with TP=1309 and also has high f1-score for 1’s.

For Multilayer Perceptron(MLP) model – hidden layer = (100,100,100), activation=logistic, solver=sgd, momentum=0.95, early_stopping=True:

We can see from the results tables that by addition of activation=logistic, solver=sgd, early_stopping=True parameters individually on default model as depreciated the performance and this model has all those parameters included together.

This model has the worst performance metrics among all the MLP models used.

1. Accuracy of the model = 56%

2. From the classification report of models:

    4. F1 score for 0’s = 0.71 and f1 score for 1’s = 0

    5. Precision for 0’s = 0.56 and precision for 1’s = 0

    6. Recall for 0’s = 1 and recall for 1’s = 0

3. Area under the curve for the model AUC = 39.19%. A descent model should have AUC value above 0.8. So, we can say that this model is not the best model.

4. Mean squared error = 0.444, highest for all the models.

5. From confusion matrix, TN=3567 and TP=0

This model just predicted every value as 0 so as to obtain accuracy of 56%. We can observe that all the performance metrics for 1’s are zeroes. This is the worst performing model among all the MLP models used

Conclusion

Using a Kaggle dataset, we set out to classify Kobe Bryant's shot attempts based on various shot features. We hoped to honor Bryant's legacy and better understand his game and style. First, we decided to do exploratory analysis of his shots and do an analysis of his best seasons.

We saw that Bryant's field goal percentage (made shots as a percentage of total shots) varied by shot distance, shot type, and shot zone. We decided to put those features into our model. For classification, we used both random forest (RF) and multi-layer perceptron (MLP) models.

We found that MLP performed better than RF across most metrics, and this validates our decision to proceed with a neural network approach to this problem.

When looking at MLP models specifically, we experimented with activation function, hidden layers, solver, momentum and early stopping. We found the base model performed extremely well, while changing most parameters led to a decrease in model performance (the logistic activation function was extremely poor). The exception was adding layers, which generally improved performance.

Our models did much better than simply guessing, which would have predicted "miss" and have been accurate ~55% of the time. Our best model had 67% accuracy.

Future work on this area would include time series analysis. This would mean using past shots from the same game as well as previous games against the same opponent. That could be added into an ensemble model to improve our classification.

References

- Shot chart number one
  - https://towardsdatascience.com/make-a-simple-nba-shot-chart-with-python-e5d70db45d0d
- Shot chart number two
  - http://savvastjortjoglou.com/nba-shot-sharts.html
- Multi layer perceptron architecture
  - https://www.researchgate.net/figure/A-hypothetical-example-of-Multilayer-Perceptron-Network_fig4_303875065
- Definition of metrics
  - https://blog.floydhub.com/a-pirates-guide-to-accuracy-precision-recall-and-other-scores/#f1-score
- Textbook (for various information about neural networks)
  - https://hagan.okstate.edu/NNDesign.pdf
- Dataset from Kaggle
  - https://www.kaggle.com/c/kobe-bryant-shot-selection/data
- Kobe Bryant’s legacy
  - https://nba.nbcsports.com/2016/04/12/kobe-bryants-legacy-in-his-own-words/
- Random forest description
  - https://www.mygreatlearning.com/blog/random-forest-algorithm/

Appendix

Technical Considerations:

1. Installation of Python 3.5 & above, Pycharm and Anaconda are necessary
2. Packages like numpy, pandas, matplotlib, sklearn need to be installed in the computer in order to execute the code

GitHub Repo Link:

All the documents related to this project are included in the following repo link:
https://github.com/dparmar16/DATS_6202_Final_Project_Group4

EDA Charts:

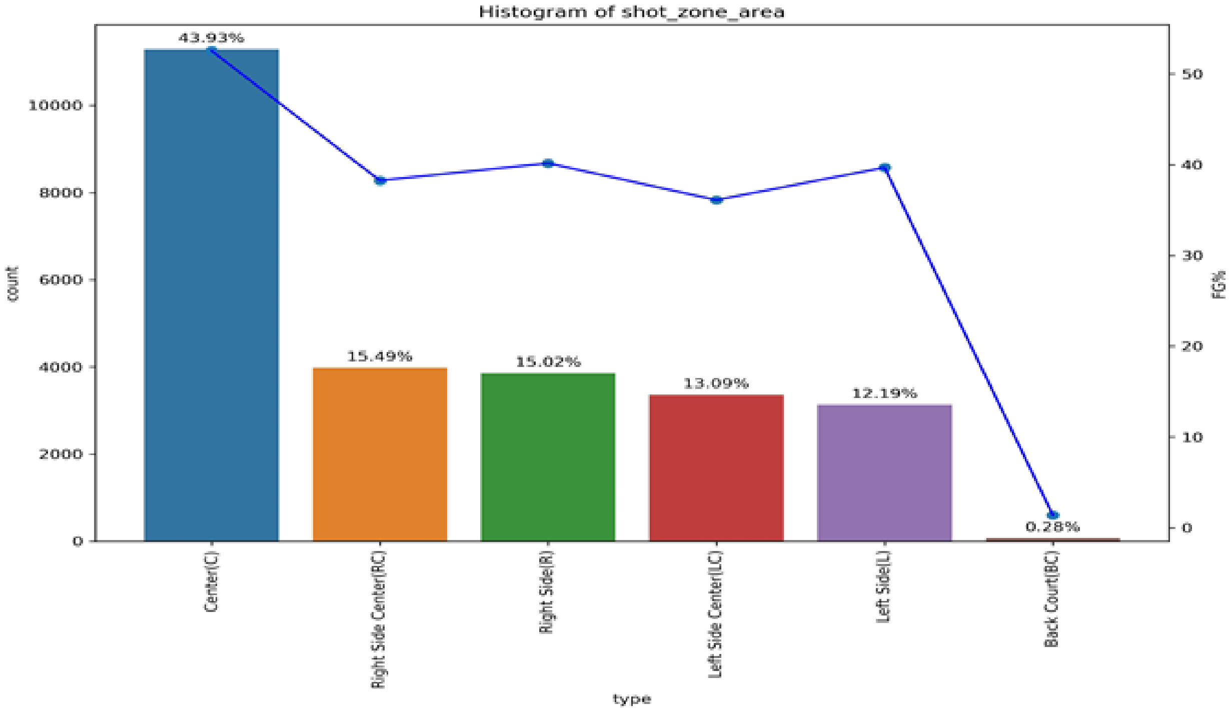


Figure33 - Distribution of shot_zone_area variable overall. We see a variation in FG% across groups.

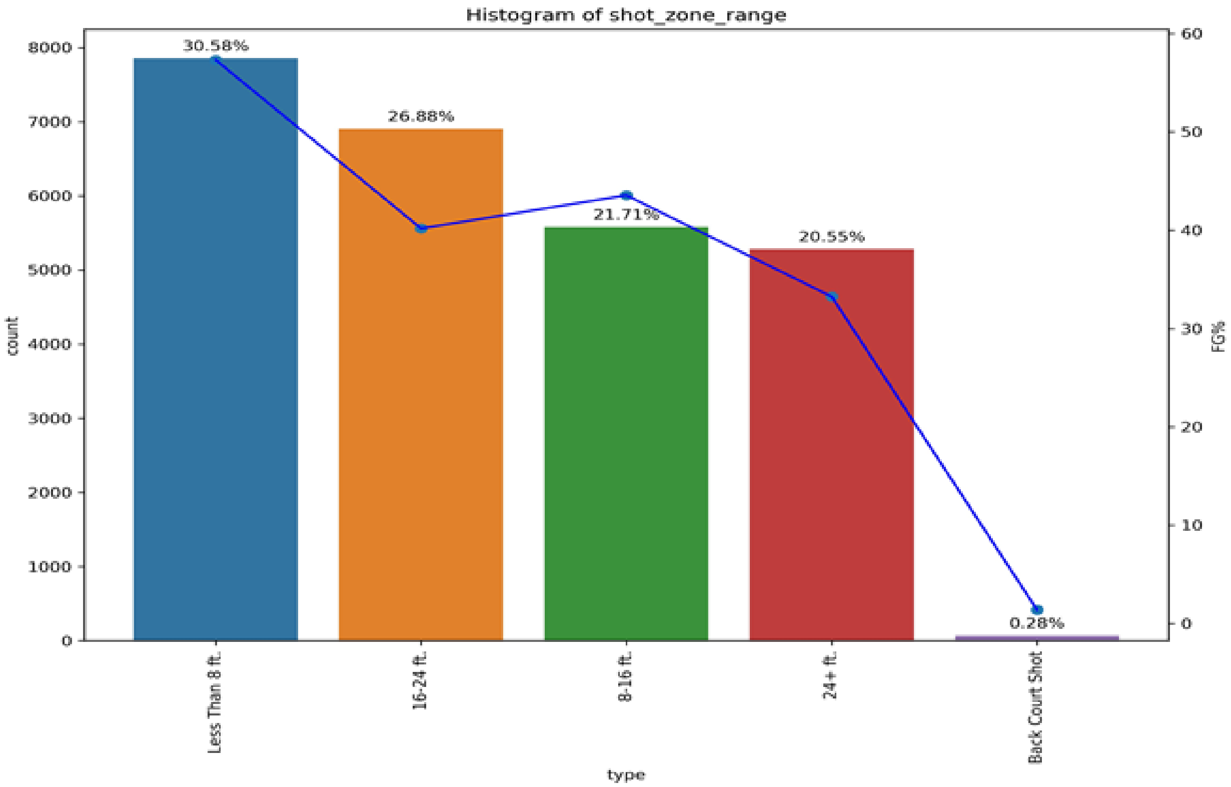


Figure34 - Distribution of shot_zone_range variable overall. We see a variation in FG% across groups.

We also have additional plots for our Opponent Analysis, which can be seen in the coming pages.

Opponent Analysis Charts for 2002 NBA Championship Season

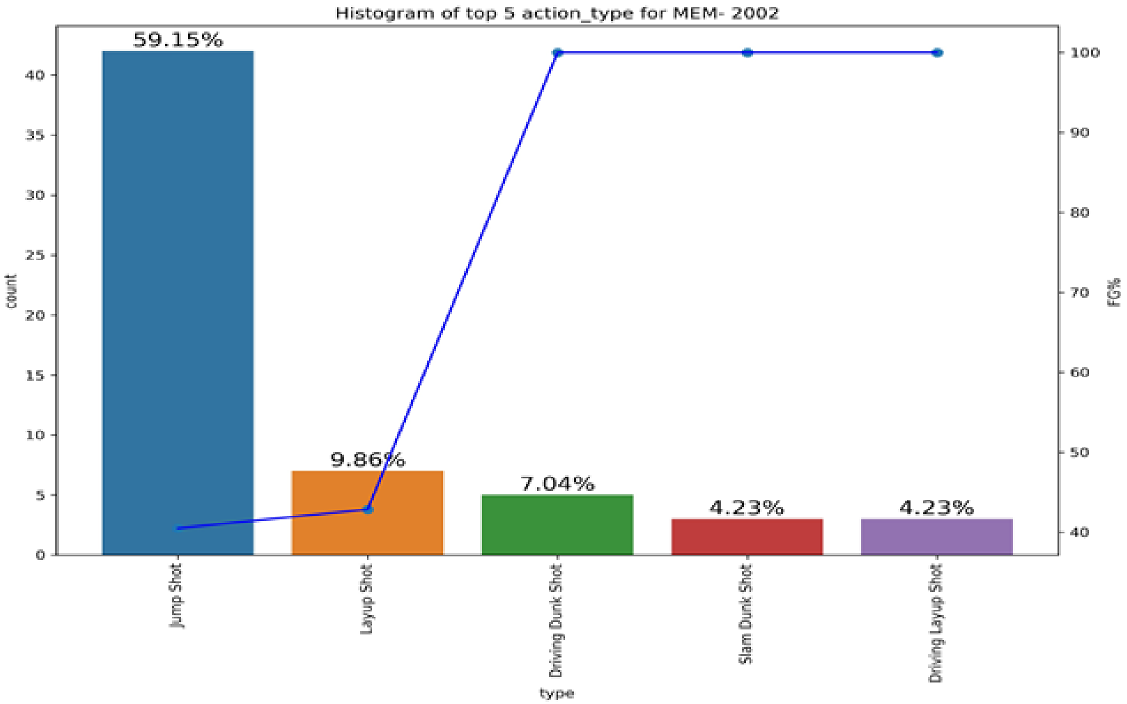


Figure35 - Bryant's action type against his best opponent (Memphis)

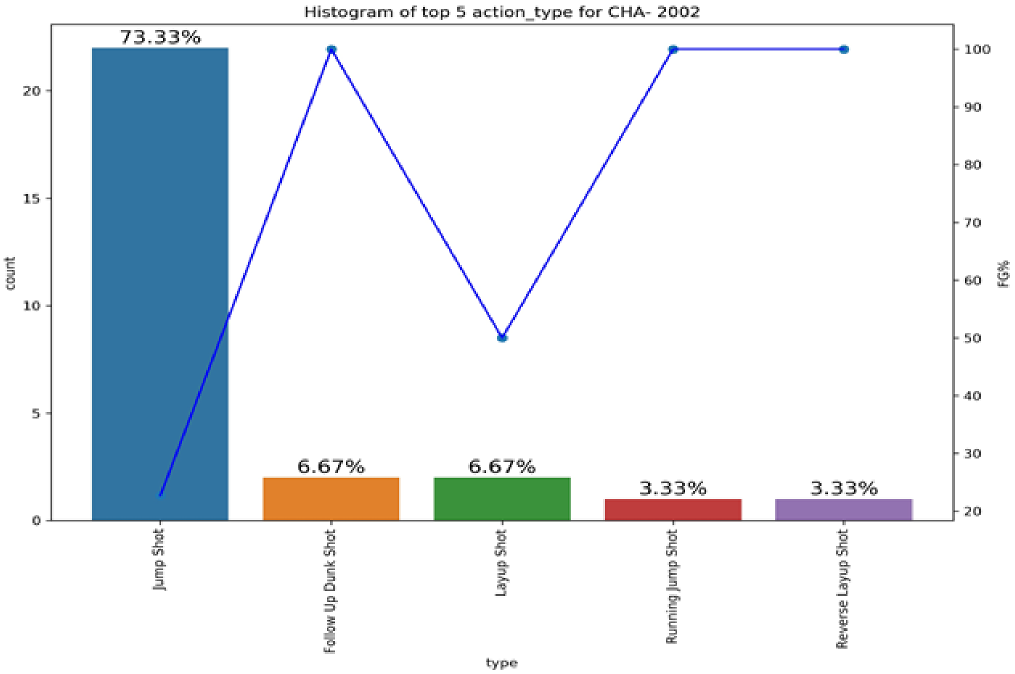


Figure36 - Bryant's action type against his worst opponent (Charlotte)

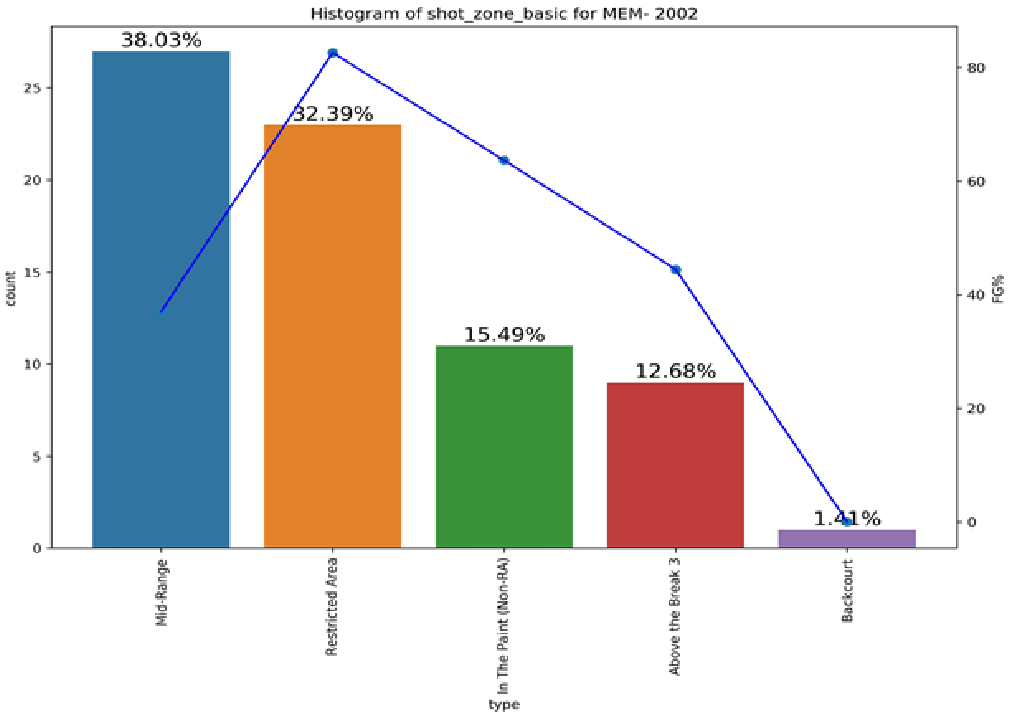


Figure37 - Bryant's shot zone basic against his best opponent (Memphis)

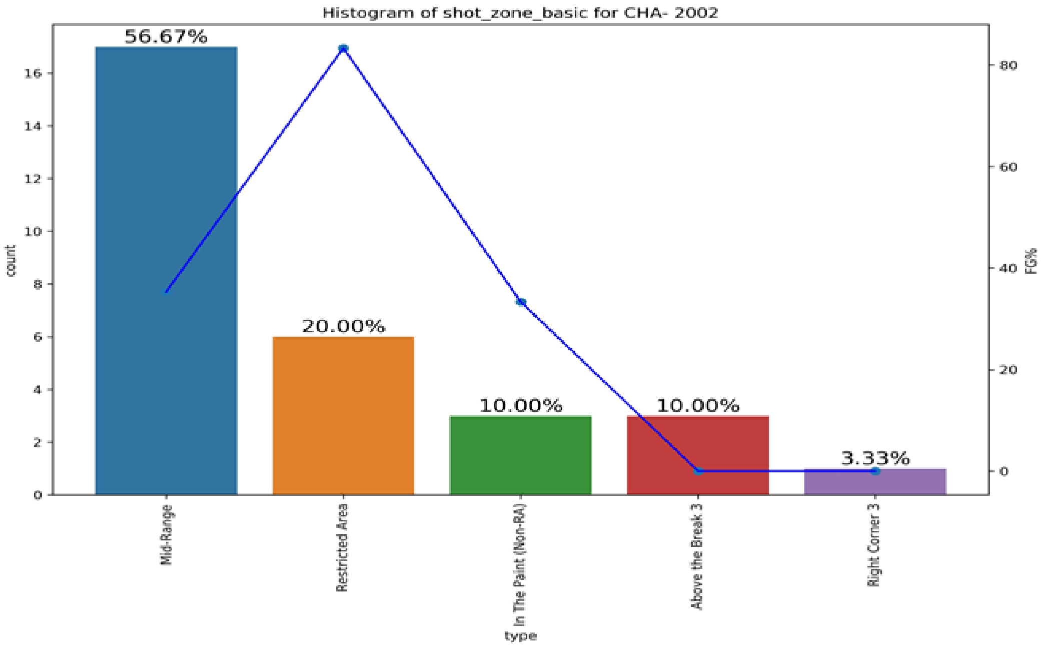


Figure38 - Bryant's shot zone basic against his worst opponent (Charlotte)

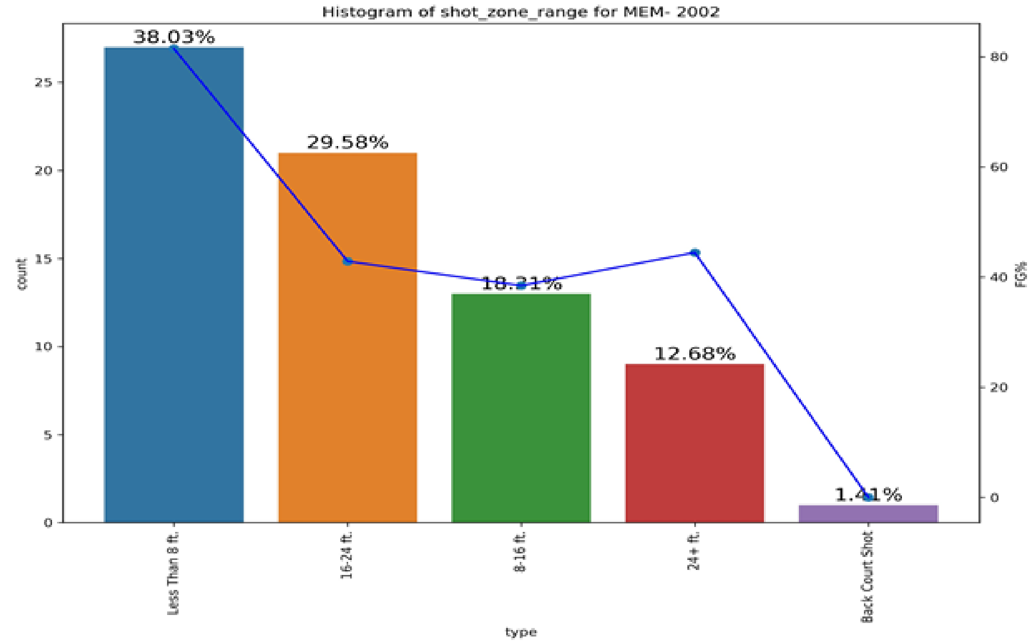


Figure39 - Bryant’s shot zone range against his best opponent (Memphis)

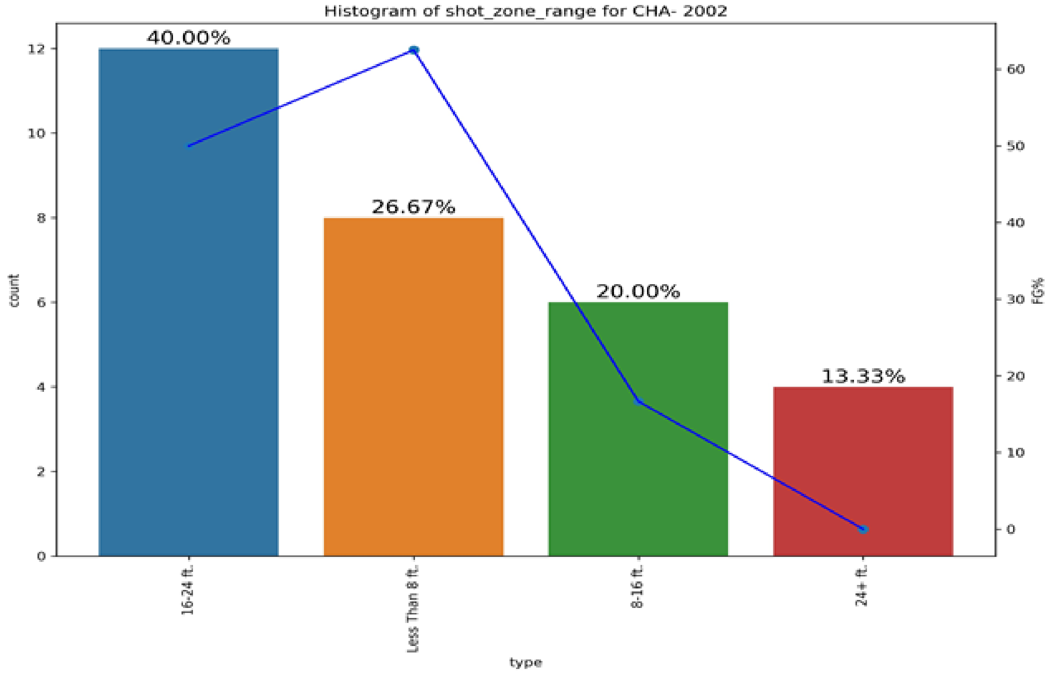


Figure40 - Bryant’s shot zone range against his worst opponent (Charlotte)

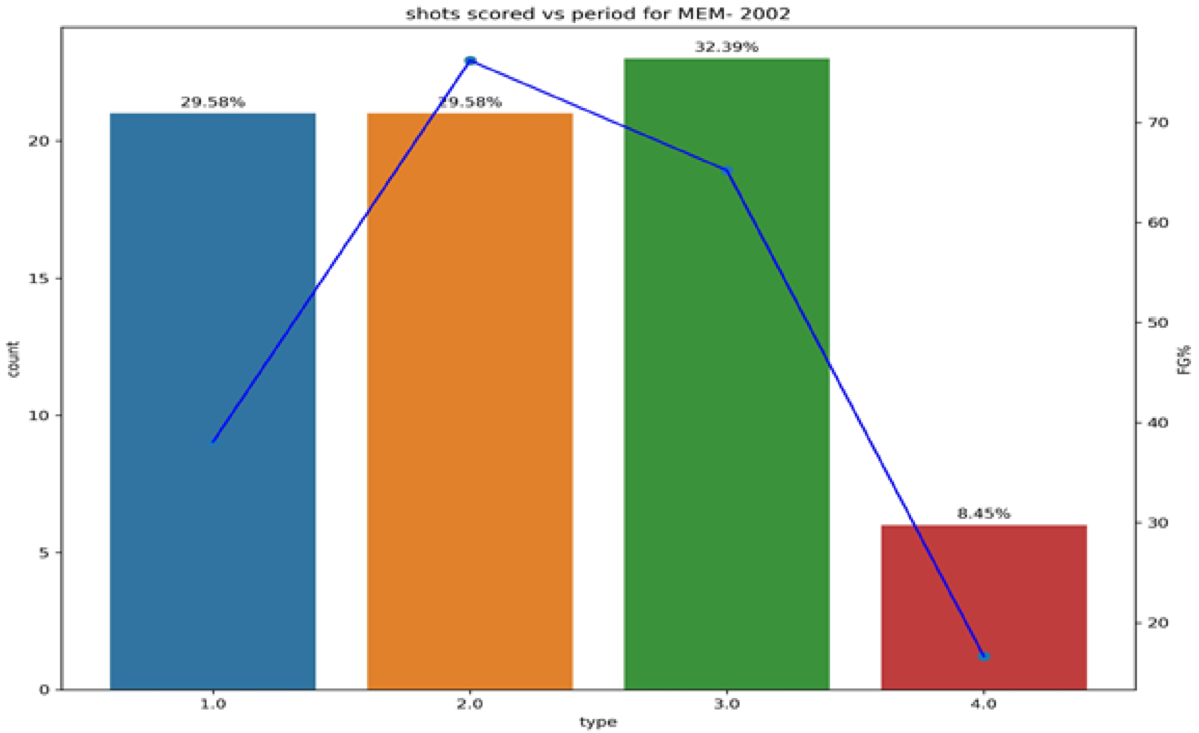


Figure41 - Bryant's shots by period against his best opponent (Memphis)

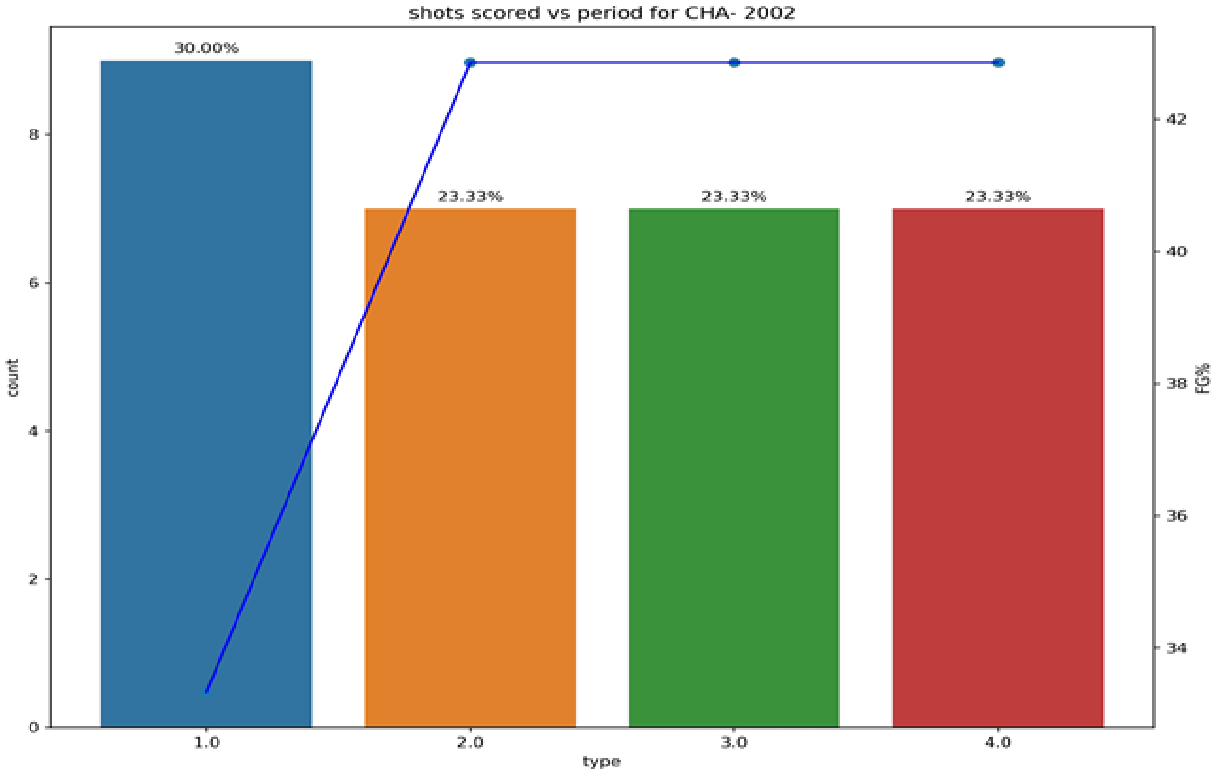


Figure42 - Bryant's shots by period against his worst opponent (Charlotte)

Opponent Analysis Charts for 2009 NBA Championship Season

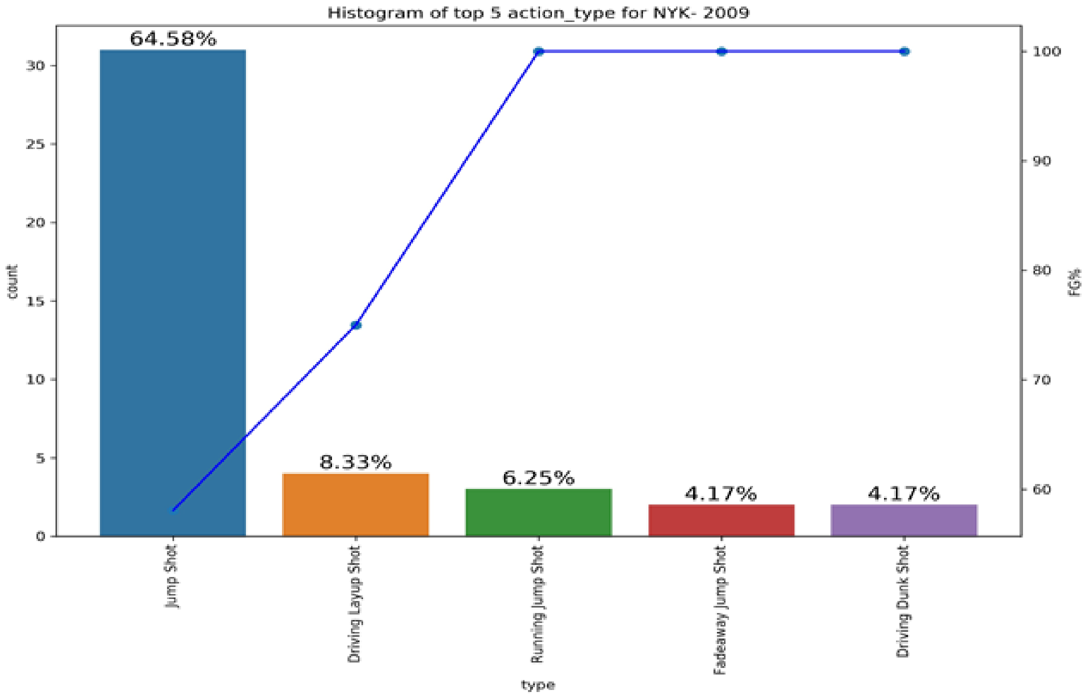


Figure43 - Bryant’s action type against best opponent (NYK)

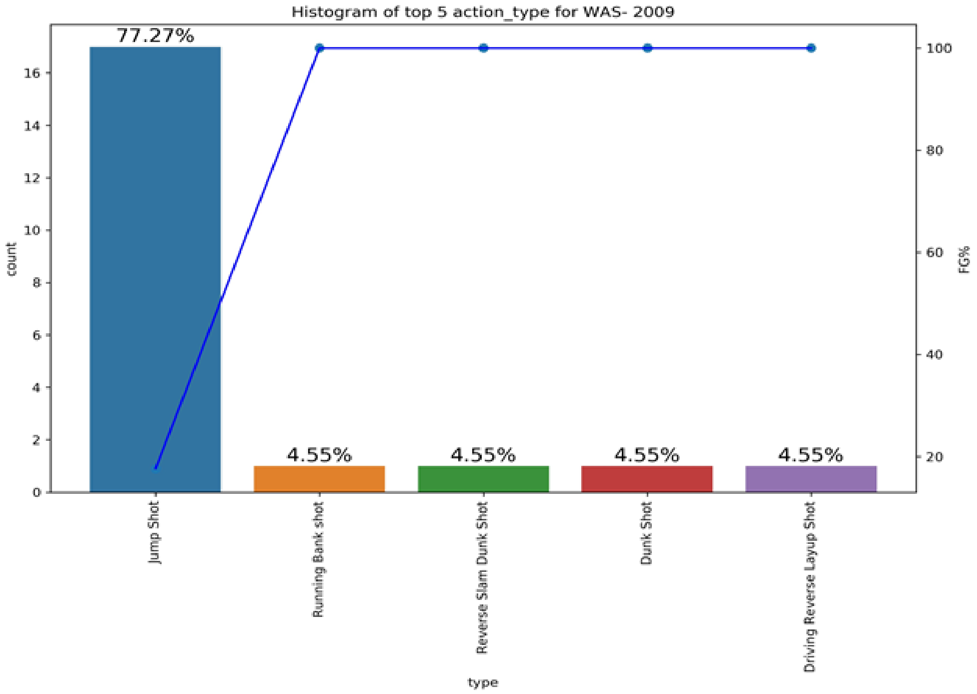


Figure44 - Bryant’s action type against worst opponent (WAS)

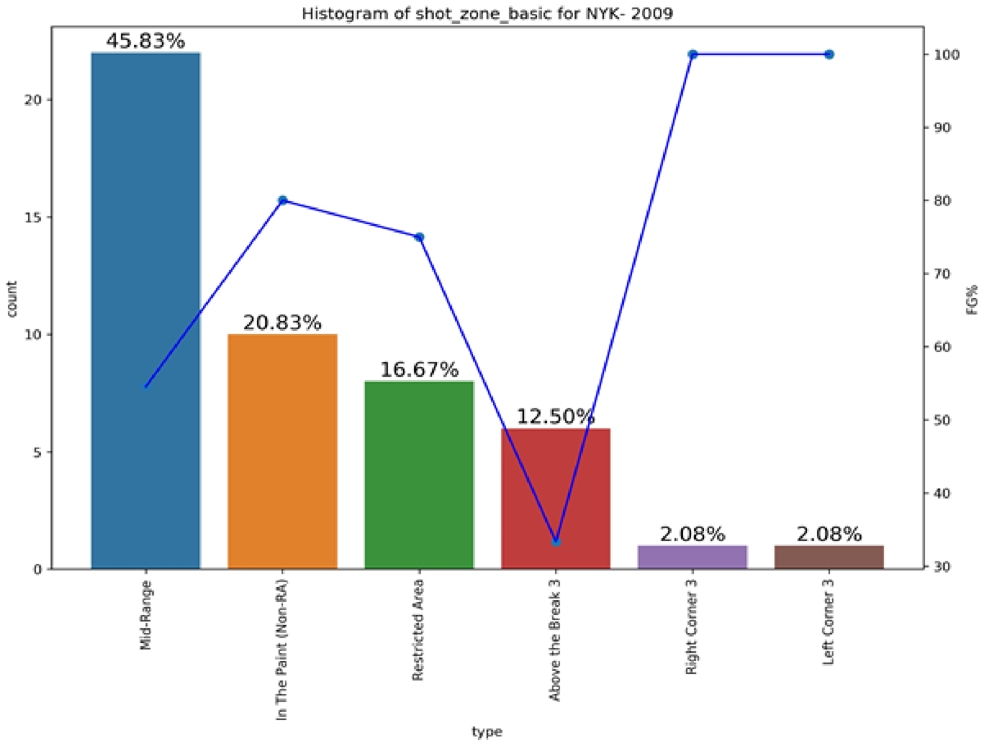


Figure45 - Bryant’s shot zone basic against best opponent (NYK)

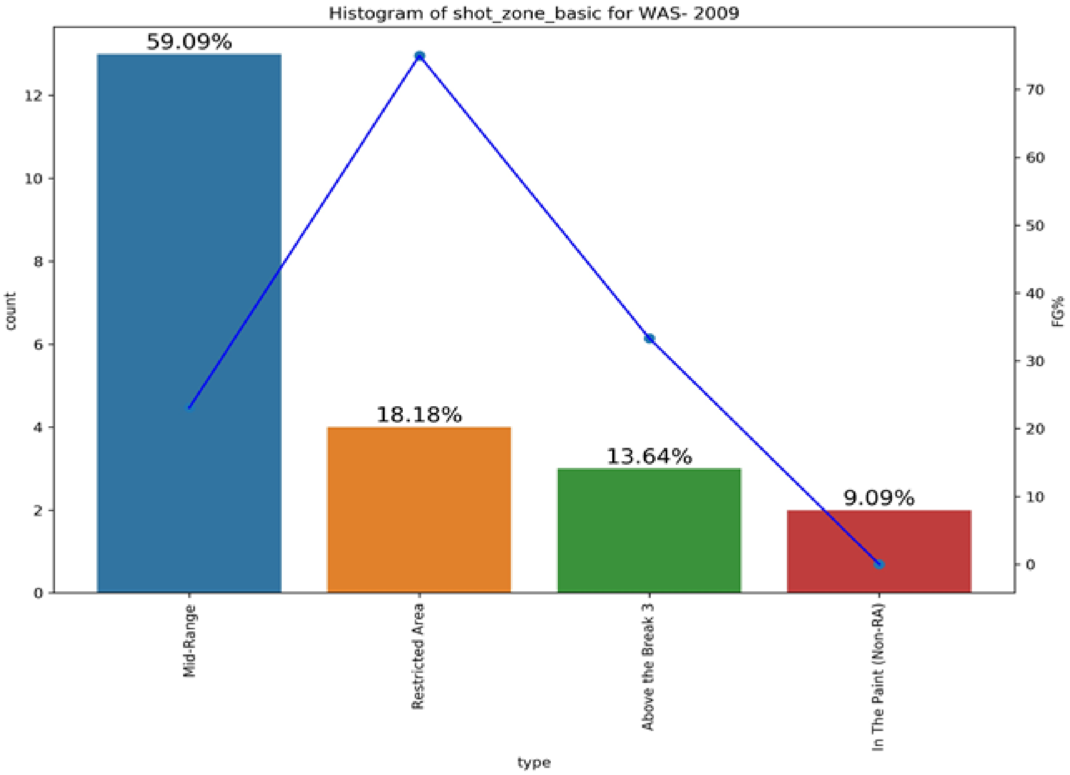


Figure46 - Bryant’s shot zone basic against worst opponent (WAS)

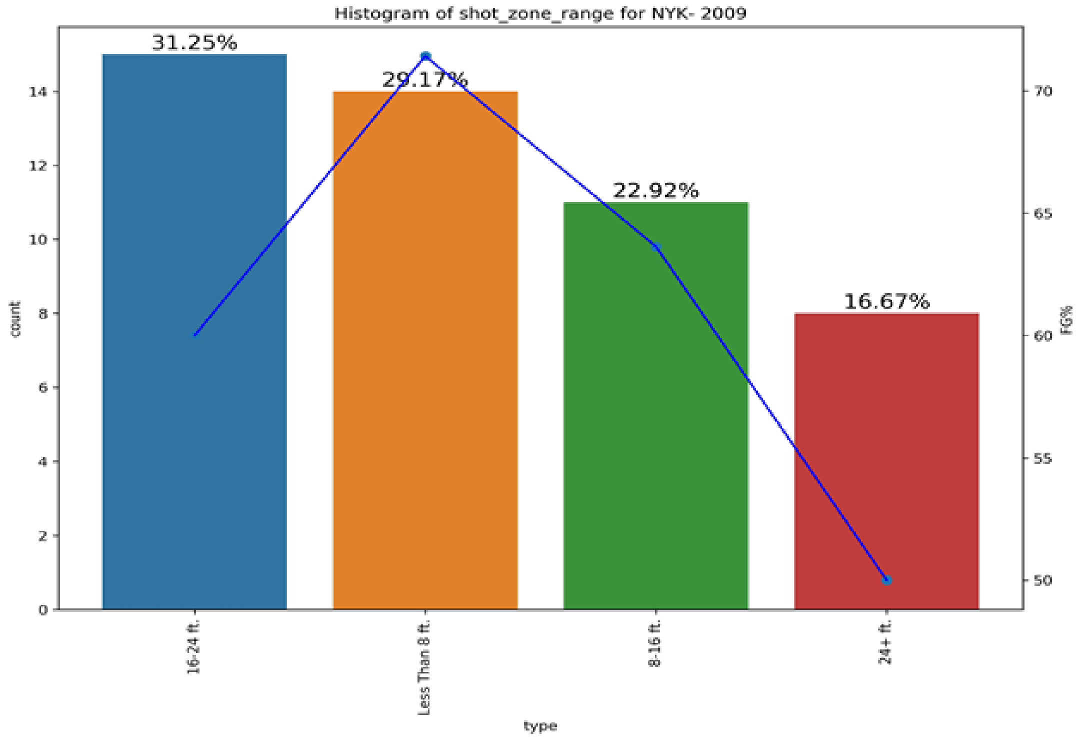


Figure47 - Bryant's shot zone area against best opponent (NYK)

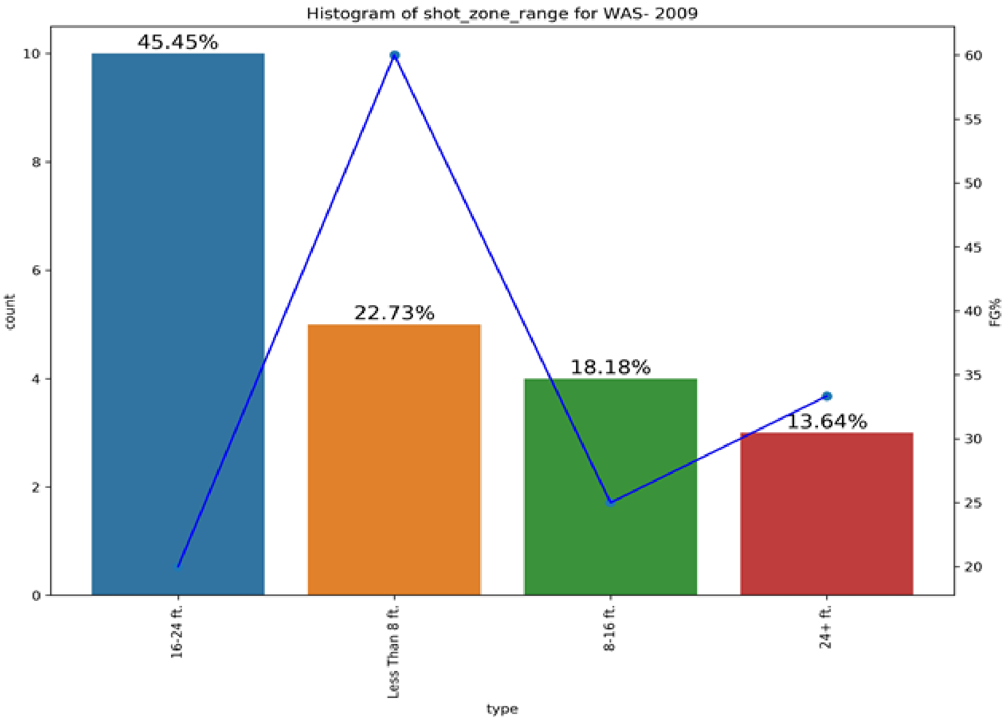


Figure48 - Bryant's shot zone area against worst opponent (WAS)

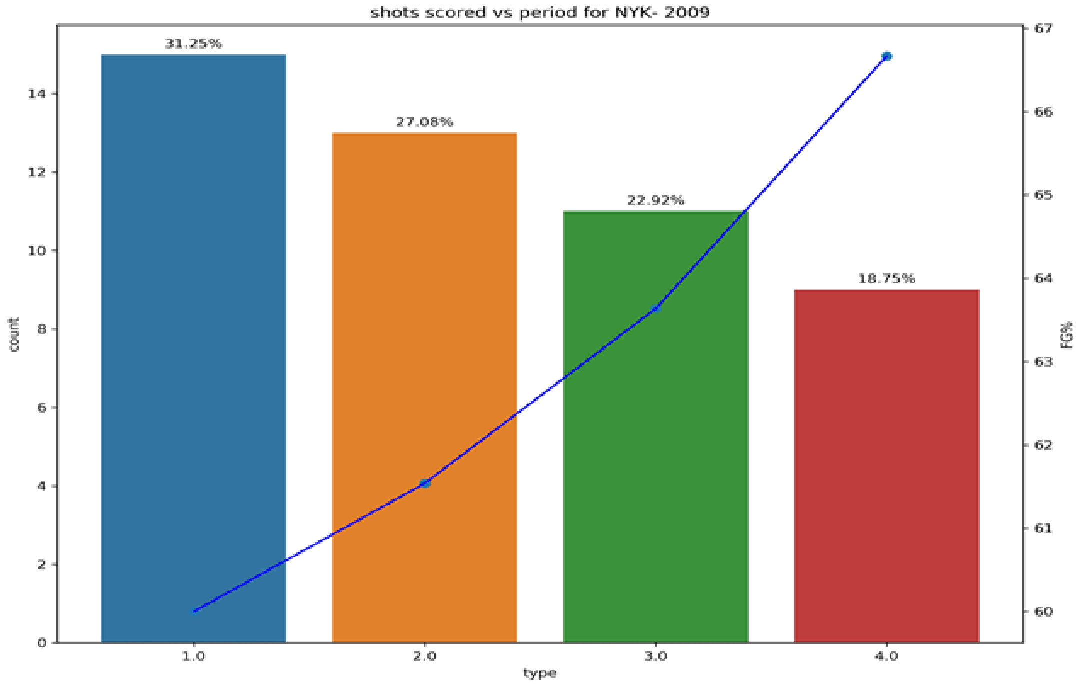


Figure49 - Bryant’s shots by period against best opponent (NYK)

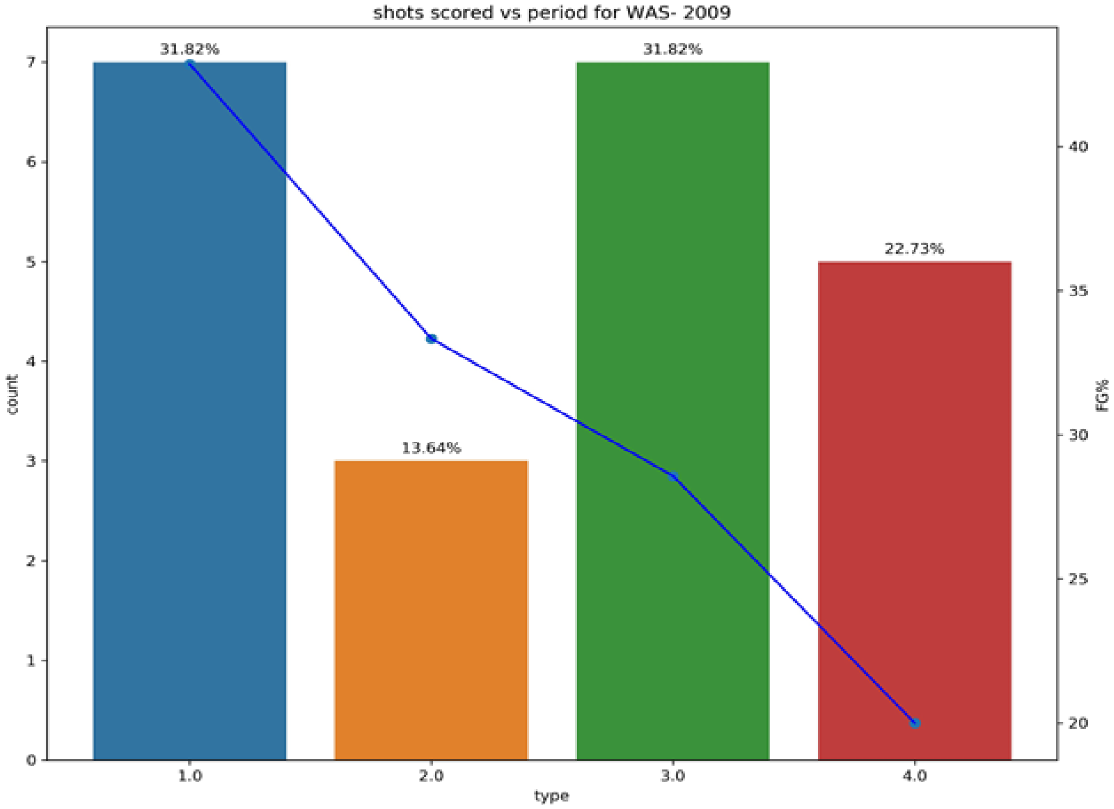


Figure50 - Bryant’s shots by period against worst opponent (WAS)