Predicting NBA Games With Combined Team and Player Data

By: Spencer Weston, Weijia Li, and Matt Kinkley

Section 1: Problem Statement

Any first attempt at predicting NBA games will utilize data gathered at the team level to generate predictions. However, aggregate team statistics do not acknowledge the presence or absence of each player in each game. If a highly impactful player is expected to miss tomorrow's game, team statistics created with the impactful player in the lineup will fail to account for the impactful player's absence. We attempt to solve this issue by clustering players and including the clusters as features in supervised models to predict NBA games.

Section 2: Objective

We aim to create the most predictive model possible as measured across binary classification of wins and losses and point estimates of each game's margin of victory (MOV). We compare our models with clustered player features to a linear regression on team statistics to measure the benefit of including player features.

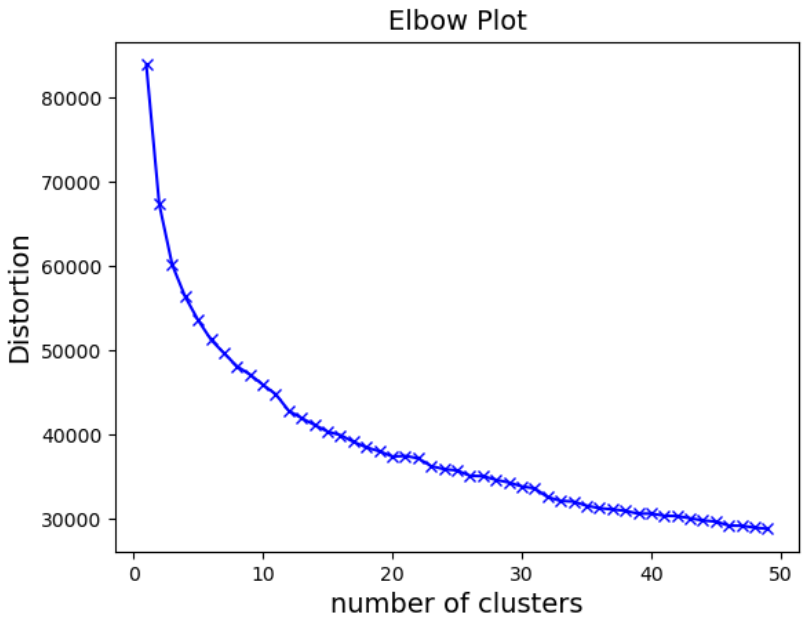
Section 3: Approach/methodology

We start with team and player based datasets. In the team dataset, we select relevant features and compute a rolling average of prior game statistics to determine feature values for the next game in the season. For the first 10 games of a season, we calculate a weighted average of the team's prior season and current season for each statistic. For the first game of each season, a team's statistics are determined from their previous season statistics. For the second game of each season, the weighted average weights the prior season's statistics at 90% and the current season's statistics at 10%. The weighted average shifts by 10% each game until the influence of the prior season drops to zero, and team statistics are computed only as a rolling average of current season performance.

For the player dataset, we select relevant features and compute 82 game rolling averages of the selected features. In the player dataset, the rolling averages allows each player's cluster to change overtime and in-response to not playing in games. In our data engineering, we consider two reasons a player receives a "Did Not Play" (DNP) in a given game. First, if the player received a DNP due to a "coach's decision", we assume (a) that the player was available to play and (b) did not play due to the coach's assessment of player quality. In this situation, we set all feature values to 0. If the player received a DNP for another reason (injuries, personal reasons, suspensions, etc.), we assume the player was unavailable. We assign null values to that player's statistics in that game, and do not include the game when calculating the player's 82 game rolling averages.

We then use K-means to cluster players and use an elbow plot to select the number of clusters. In this case, we selected 12 player clusters. We use one-hot encoding to transform the K-means output into features for supervised learning.

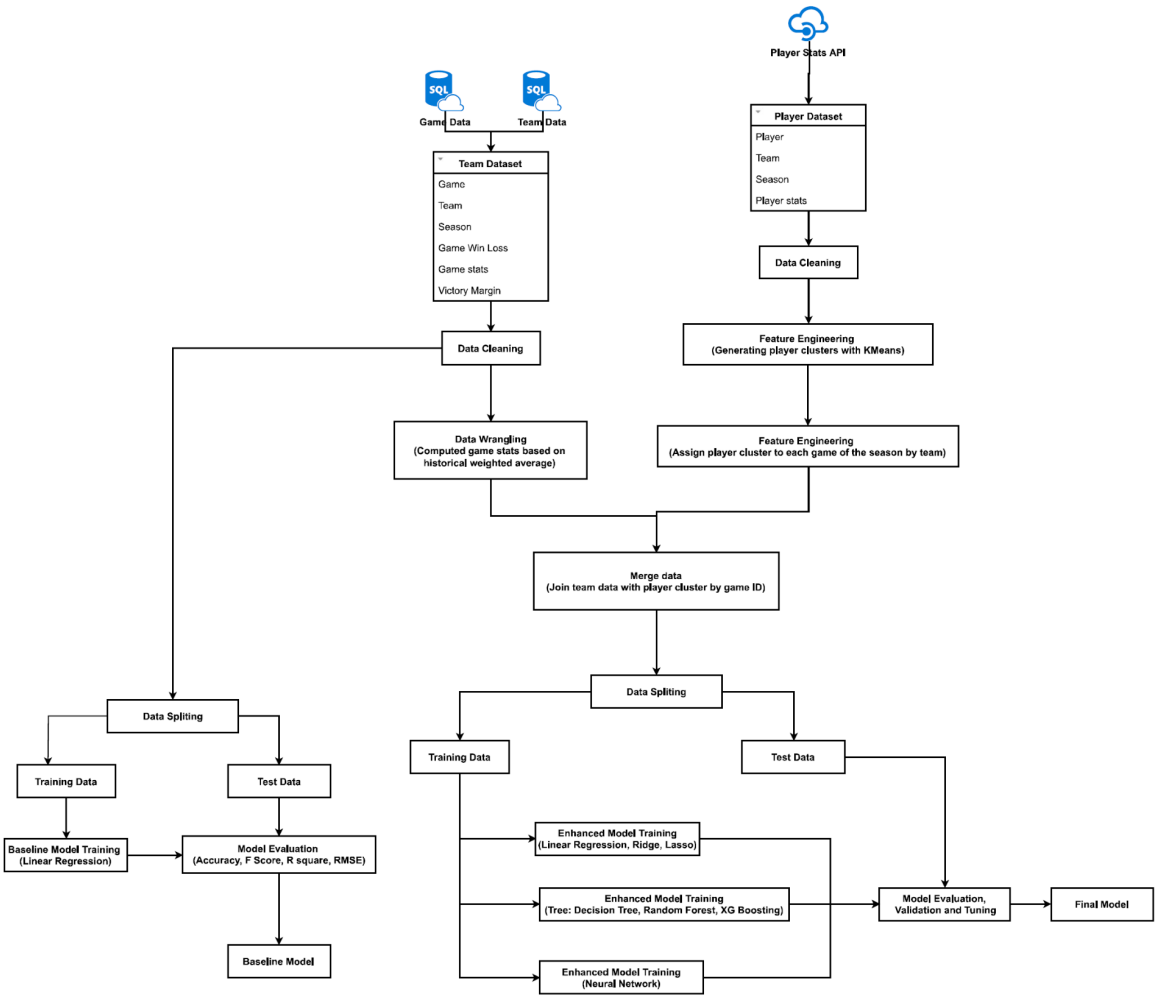
Figure 3.1: K-Means Elbow Plot of Distortion



In both the team and player datasets, we organize features into home and away clusters. If we have *M* team features, the supervised team-based model has *2M* features where the first *M* features are the features of the home team and the *M+1* to *2M* features come from the away team. We also perform this process on player cluster features. The player clusters and team statistics are then joined to give the feature set for supervised models.

Once we generate the supervised learning dataset, we split the data into training (60%), validation (20%), and testing (20%) sets. Our supervised models include: Lasso regression, Ridge regression, Regression trees, Random forest with boosting, and Neural Networks We evaluate these models with the metrics specified in the *Evaluation Parameters* section.

Block Diagram:



Section 4 Datasets

We pulled two datasets from Kaggle: a player stats dataset and a team stats dataset. We extracted the per 36 minutes player stats from the NBA API [1] on the documentation of the Kaggle Basketball Dataset[2]. The player dataset contains 27,748 samples with 5 identifiers and 22 features. The identifiers in the dataset help us identify the stats of the player by season and team. The stats show the player's performance during a particular season in a given team. If a player was on more than one team for a season, season Totals for that player were used and partial team seasons dropped. This player stats dataset is later joined with the Basketball Dataset in Kaggle via SQLITE to get additional player attributes such as height and weight. NAs are removed from the player stats dataset since it only weighs a small percentage of the overall sample size. NAs from the player attributes are replaced by the average of that feature.

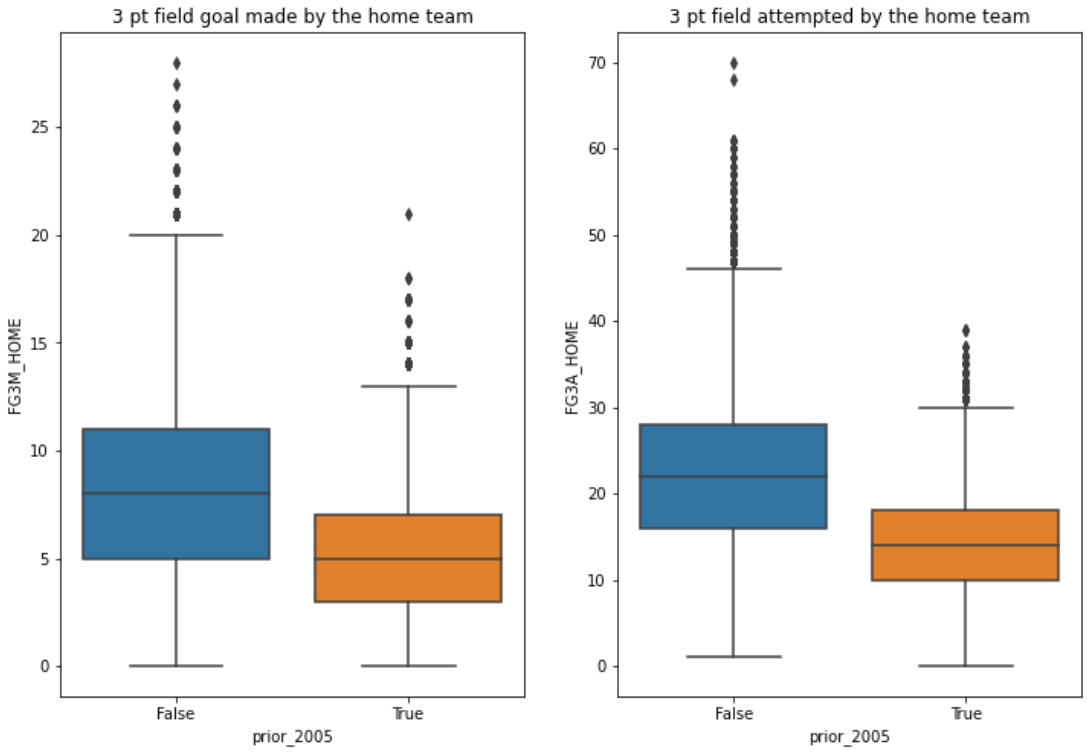
---

[1] https://github.com/swar/nba_api/blob/master/docs/nba_api/stats/endpoints/playercareerstats.md
[2] https://www.kaggle.com/wyattowalsh/basketball

The Team dataset is extracted from Kaggle Basketball Dataset's game table via SQLITE. There are 62,448 samples in this data since November 1996 with more than 100 features. The dataset contains identifiers for game, season and teams played and other team specific statistics for each game. Additional features are created to calculate margin of victory of the game and boolean win or loss indicator.

In 2005 the NBA updated rules to curtail hand-checking, clarify blocking fouls, and call defensive three seconds in order to "open up" the game. We explored any difference in player stats before and after the rule change, expecting to see the different play style result in more 3 point shots and less fouls, for example. *Figure 4.1* shows that the number of 3 point shots attempted and made in games after the rule change greatly increases. Due to this, we decided to remove all data prior to 2005 and focus on games playing by updated rules.

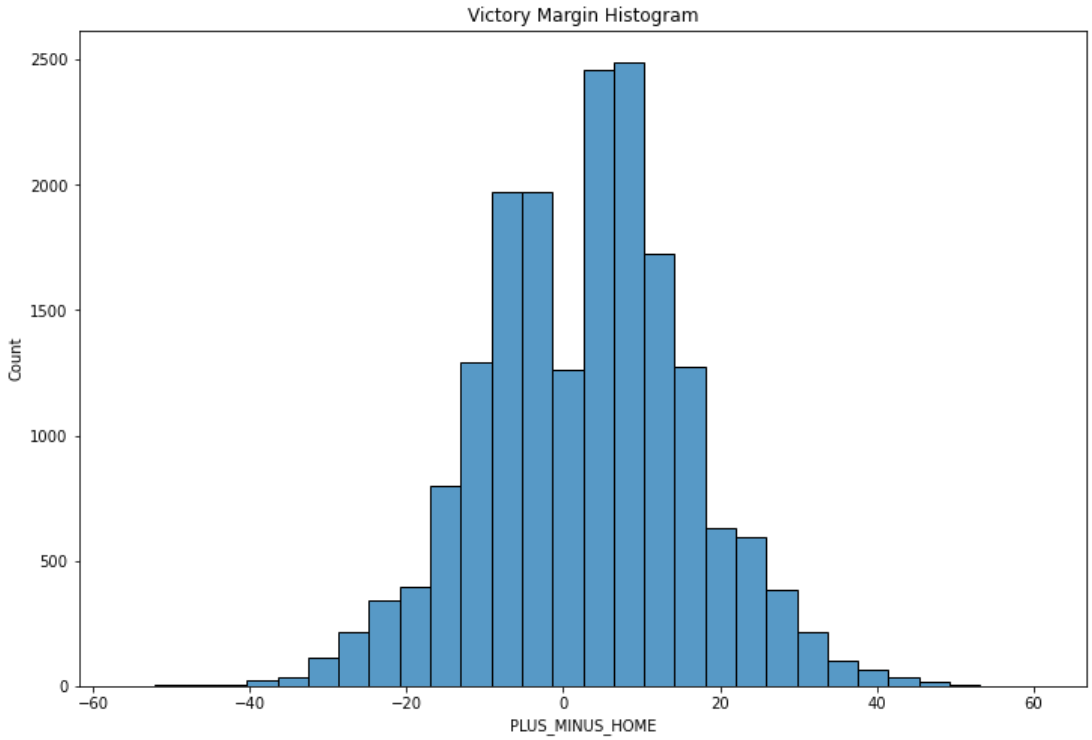
Figure 4.1: 2005 NBA Rule change effect on selected game statistics



As mentioned above in the Approach/methodology section, we used K-means clustering to identify the most suitable number of clusters to represent the players, observing a distinct decrease in distortion at 12 clusters. We mapped all players to the clusters by team to each game (12 clusters for home team and 12 clusters for away team). The final dataset to begin modeling for this project is generated by joining the created player cluster dataset to the team dataset. It contains 16,635 records, each representing a game with team and player cluster features and a final score outcome variable. We performed basic data cleaning on the final dataset by removing NAs, assigning appropriate data types to the variables, and analyzing potential outliers and identifying features that are linear combinations of other features (e.g. Points Made, Attempted, and Percentage are linear combinations of each other, so we dropped Points Made). Post data cleaning, there are 5 identifiers, 2 outcomes and 64 features regarding the historical team level performance statistics for each game and player cluster involved in

each game. The final dataset does not contain any outliers in the victory margin outcome variable - there is a normal distribution (*Figure 4.2*).

Figure 4.2: Histogram on outcome variable



Section 5: Baseline and Metrics for Evaluation

The teams dataset alone, having team performance per game with the score outcome, will serve as our benchmark. We fit a Linear Regression on normalized team statistics as our baseline model. If we can outperform this baseline by incorporating player clusters and other model types below, our efforts will be deemed worthwhile. Success will be indicated by the addition of player level data resulting in enhanced performance of an existing team model.

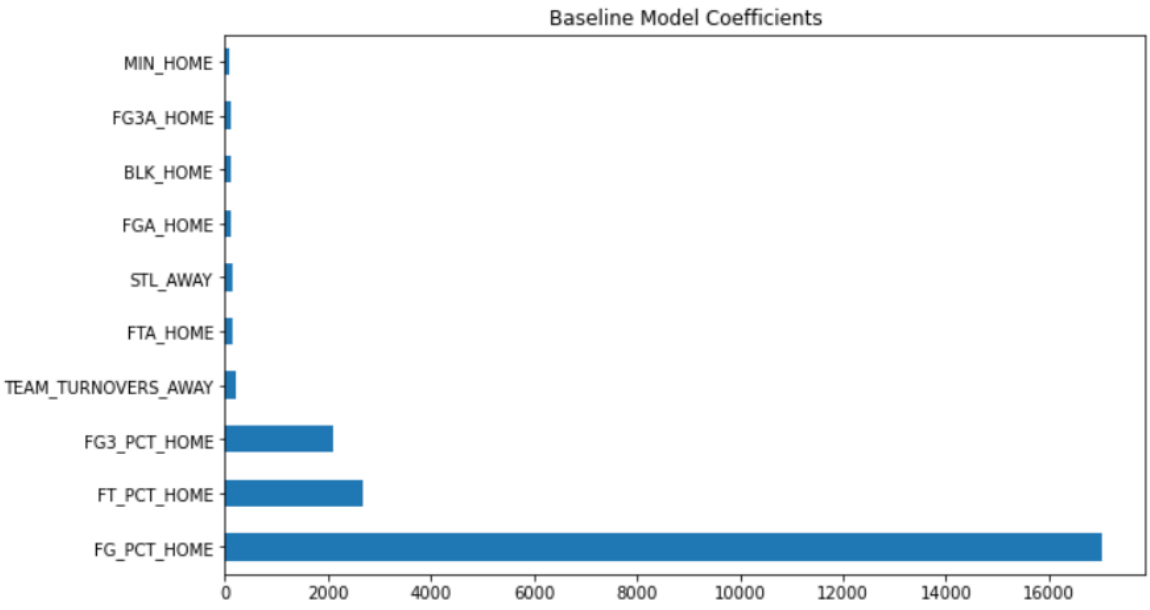
We will consider two types of evaluation metrics for our NBA game predictions. First, we employ F-scores and accuracy to measure model effectiveness in predicting the correct winner, a boolean outcome variable. Second, we measure root mean squared error and R^2 to evaluate our point estimate of the final margin of victory.

Section 6: Experiments

6.1 Baseline Model

To start, we fit a Linear Regression on the Teams dataset to serve as our benchmark. The data was normalized and cleaned by removing NAs as described above, but the baseline model was fit on only the aggregate team features, excluding the generated player clusters. We split the dataset into 80% Training and 20% Test. The most important features for this model are the Percentages for each method of scoring: Field Goal, Free Throw, and 3-Point. This is expected because these variables (linearly combined with the Attempts for each) directly translate to the number of points the Home team scored. Also, we can note that Field Goals have a much higher impact than Free Throws or 3-Points scored in determining the winner of a game.

Figure 6.1.1: Feature Importance in Baseline Regression



6.2 Regression Models

6.2.1 Linear Regression

Next, we incorporated the generated player clusters in our dataset and ran a new Linear Regression model. Now we can see cluster features in our coefficient attributes, indicating that some are higher in importance than others. We can take this to mean that some players are more impactful than others, as determined by their individual player attributes.

Figure 6.2.1.1: Top 5 Coefficients from Clustered LR

Features	Estimated Coefficients
TOTAL_TURNOVERS_HOME	5.856242e+14
TOTAL_TURNOVERS_AWAY	3.954055e+14
cluster_7h	1.493754e+12
cluster_1a	1.492424e+12
cluster_3a	5.343541e+10

This Linear Regression model did not perform significantly better or worse in the test set than it did in the training set - explained further below. Therefore we do not immediately suspect overfitting. Nonetheless, we experiment with regularization, specifically ridge and lasso regression to try and optimize our prediction.

6.2.2 Ridge Regression

While normal regression returns unbiased regression coefficients, Ridge regression allows us to regularize, or push our estimated coefficients towards zero in an attempt to penalize those not adding much value. This decreases model complexity while still retaining features, and would be ideal in a scenario where we want to decrease model complexity.

Using GridSearch we tested for alpha (L2) values between 10e-8 and 10e8, optimizing for the model with the lowest RMSE. Ultimately a value closest to 0.0001 was selected, resulting in a marginally improved RMSE. However when fitting a model using this optimized alpha, the results were not improved.

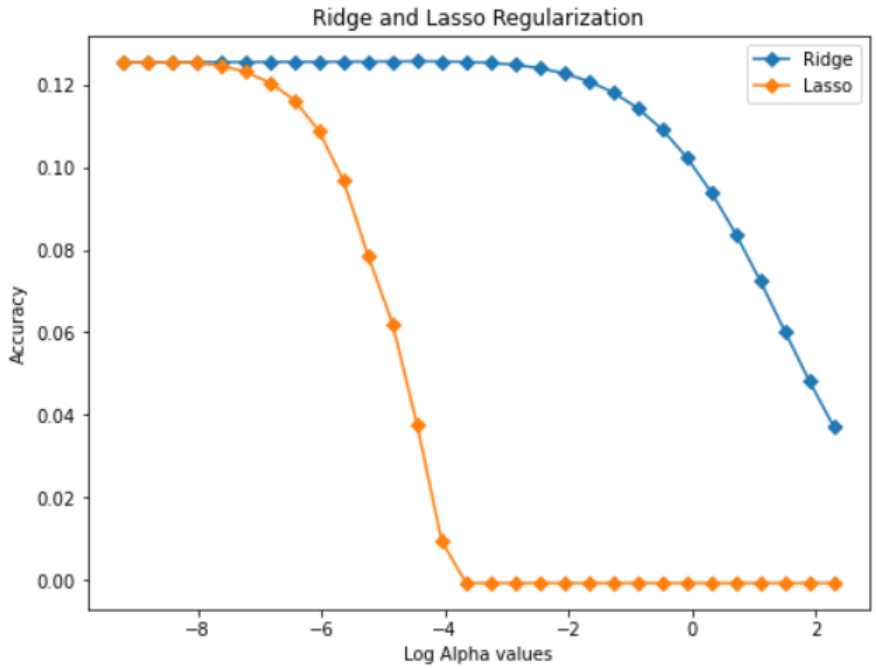
6.2.3 Lasso Regression

Lasso regularization contrasts with Ridge in that it actually sets the less important coefficients to zero, allowing us to leave predictors out of the model. Lasso tends to do well if there are a small number of significant parameters and the others are close to zero, which our coefficient chart seems to indicate.

Again we tested for different lambda (L1) regularization values, ultimately arriving at a value closest to 1e-5. This again seemed to arrive at an improved RMSE, but when fitting a new model using the alpha value did not improve results.

Below in Figure XX we can see that both L1 and L2 regularization parameters performed best while at their lowest, and accuracy decreased as alphas were increased. This essentially shows us that regularization has had little, if any, beneficial effect for our model.

Table 6.2.3.1: Accuracy by Regularization Strength for Regularized Regressions



6.3 Tree-Based Models

Three models were trained with tree-based machine learning algorithms: basic decision tree, ensemble method with Random Forest and boosting method with XG Boosting. The features used for the tree-based models are standardized. PCA was also explored; however the model performance was not better with PCA and it's not used. The parameters used for each model are identified by hyperparameter tuning with 3-fold cross validation.

The decision tree model is trained via DecisionTreeRegressor. Decision tree is a type of Supervised Machine Learning where the data is continuously split according to the pre-defined parameter. The best decision tree model after hyperparameter tuning are shown in *Table 6.3.1.*

Ensemble method with RandomForestRegressor consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a prediction and the prediction with the most votes becomes our model's prediction. The best RandomForestRegressor model after hyperparameter tuning are shown in *Table 6.3.2.*

Boosting method with XGBoosting combines a set of weak learners into a strong learner to minimize training errors. In boosting, a random sample of data is selected, fitted with a model and then trained sequentially—that is, each model tries to compensate for the weaknesses of its predecessor. The best XGBoosting model after hyperparameter tuning are shown in *Table 6.3.3*

Table 6.3.1: Decision Tree Hyperparameter Tuning

Hyperparameter	**Values for Parameter Tuning**	**Selected Parameter**
max_depth	"best", "random"	10
max_features	["auto","log2","sqrt",None]	auto
max_leaf_nodes	[None,40, 80, 120]	80
min_samples_leaf	[10,20,30,40]	20
min_weight_fraction_leaf	[0.01,0.1]	0.01
splitter	["best","random"]	random

Table 6.3.2: Random Forest Hyperparameter Tuning

Hyperparameter	**Values for Parameter Tuning**	**Selected Parameter**
bootstrap	[**True**]	True
max_depth	[30, 50, 70, 72, 74, 90]	70
max_features	[20, 30, 40, 50]	20
min_samples_leaf	[5, 10, 14, 18, 20]	18
min_samples_split	[5, 7, 11, 13, 14, 15]	14
n_estimators	[500, 700, 1000, 1500, 2200, 2400, 2600]	2400

Table 6.3.3: XGBoost Hyperparameter Tuning

Hyperparameter	**Values for Parameter Tuning**	**Selected Parameter**
colsample_bytree	[0.3, 0.5, 0.7, 0.9]	0.9
learning_rate	[0.01,0.05, 0.1]	0.01
max_depth	[3, 5, 7, 9, 13]	5
min_child_weight	[3, 5, 7, 9, 13]	9
n_estimators	[300, 500, 900,1200,1500]	900
objective	['reg:squarederror']	reg:squarederror
subsample	[0.3, 0.5, 0.7, 0.9]	0.9

6.4 Neural Network

The neural network was implemented with Tensorflow and Keras. The network attempts to predict the margin of victory or loss for the home team. To generate a binary win or loss outcome from the network, a positive value becomes a predicted win for the home team and a negative value becomes a predicted loss for the home team. We first use Bayesian Optimization to determine the optimal hyperparameters then retrain the model with the selected hyperparameters to determine the optimum number of training epochs.

The network architecture was determined using Bayesian Optimization across the hyperparameters detailed in Table 7.4.1. The Bayesian Optimizer runs up to 200 training epochs per configuration, tests 100 hyperparameter configurations, and optimizes for mean squared error on the validation set. The validation set consists of 20% of the training data. The tuner will also stop training on any configuration if 25 epochs pass without an improvement in the mean squared error.

Table 6.4.1: Hyperparameter Tuning Options With Bayesian Optimization

Hyperparameter	**Values for Parameter Tuning**	**Selected Parameter**
Dropout (Boolean)	True or False	False
Dropout Percent (if Dropout == True)	[0.05, 0.1, 0.25, 0.5]	NA
L2 Regularization Strength	[0.0001, 0.001, 0.01, 0.1, 0.25, 0.5]	0.25
Number of Layers	[1, 2, 3]	1
Nodes per Layer	Range(8, 64, step=4)	36
Learning Rate	[0.1, 0.01, 0.001, 0.0001]	0.01

After hyperparameter optimization, we determine the optimum number of training epochs. Selecting the hyperparameters that minimize mean square error on the validation set, we retrain the model for 300 epochs using the same 80/20 split of the data into training and validation sets. After retraining, we identify the number of training epochs that minimize mean squared error on the validation set. The selected hyperparameters are shown in Table 7.4.1, and the optimum number of training epochs is 38.

We then rebuild the model with the selected hyperparameters and train for the selected number of epochs (38) on the full training set. This is our final neural network model. We will evaluate this model's performance on the test set.

Section 7: Results

7.1 Baseline Model

These are baseline scores from our Teams dataset - no clusters added.

Training

Accuracy	F1 Score	R Square	RMSE
0.631274	0.725789	0.091382	12.663492

Test

Accuracy	F1 Score	R Square	RMSE
0.646228	0.740004	0.100825	12.72275

7.2 Linear Regression with Player Clusters

Training

Accuracy	F1 Score	R Square	RMSE
0.644424	0.728046	0.135042	12.3555

Test

Accuracy	F1 Score	R Square	RMSE
0.650135	0.736532	0.136191	12.47004

The results of this Regression with added player clusters is hopeful, as we can see Accuracy and F1 Score increase by about 0.01 each. R Squared is well improved over our baseline model, and Root Mean Squared Error has decreased as well. We can thus initially assert that adding player clusters has helped explain more variance in our dataset, as well as improve our win/loss predictor.

Comparing scores between the Test and Training set for this run, we see that Accuracy, F1, and R-Squared have increased similarly to that of the baseline model test/training sets. Root Mean Squared Error is again higher in the test set. Because RMSE is particularly sensitive to outliers as it squares error, this might be due to “outlier” data points. We do not suspect overfitting because the test set is performing similarly to the training set.

Ridge Regression with optimized alpha

Training

Accuracy	F1 Score	R Square	RMSE
0.6445	0.728182	0.134505	12.359335

Test

Accuracy	F1 Score	R Square	RMSE
0.648933	0.735866	0.134769	12.480303

As mentioned above, GridSearch revealed an optimal alpha value where a lower RMSE was returned. However as we now fit a new model using that parameter, we find that it actually performs worse than our normal Linear Regression.

Lasso Regression with optimized alpha

Training

Accuracy	F1 Score	R Square	RMSE
0.644424	0.728296	0.134394	12.360127

Test

Accuracy	F1 Score	R Square	RMSE
0.649234	0.736152	0.134952	12.478983

Scores from our Lasso Regression are more in line with our original Linear Regression model, but again have no significant improvement. If we had to choose one, we would go with the original Linear Regression as it is simplest and needs no parameter tuning.

7.3 Tree-Based Models

The performance of the decision tree model is slightly overfitted on the training dataset for the outcome variable of margin of victory by comparing RMSE and R Square on outcome variables predicted from training and testing dataset. The performance of the decision tree model is consistent for the training dataset for the outcome variable of win or lose of the game by comparing the accuracy and F1 score on outcome variables predicted from training and testing dataset. However, the decision tree model did not outperform the baseline.

Evaluation metrics for the decision tree model on training data:

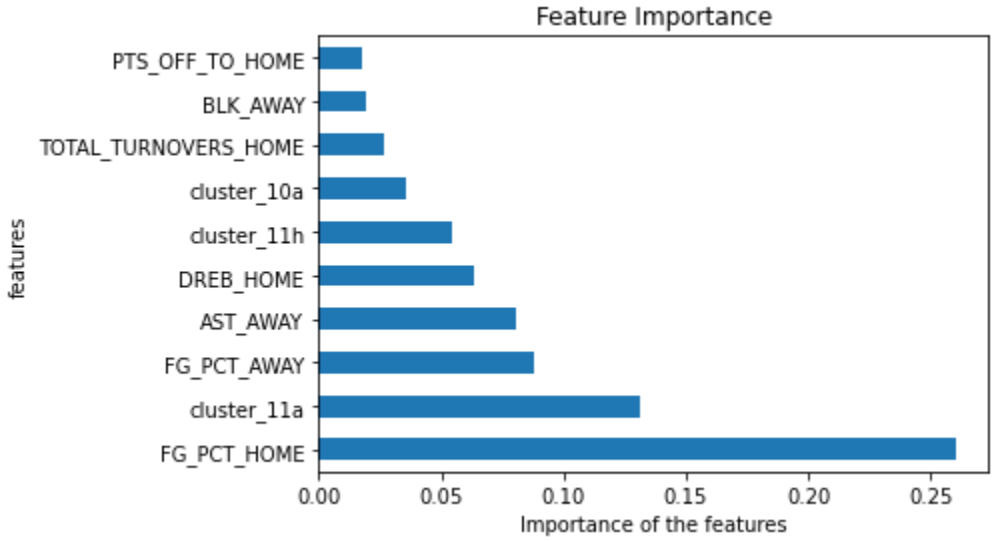
Accuracy	F1 Score	R Square	RMSE
0.631425	0.729379	0.090377	12.670495

Evaluation metrics for the decision tree model on testing data:

Accuracy	F1 Score	R Square	RMSE
0.634806	0.736271	0.061603	12.997274

Figure 7.3.1 shows the feature importance of the decision tree model.

Figure 7.3.1: Feature Importance of the Decision Tree Model



The performance of the random forest model is overfitted on the training dataset for the outcome variable of margin of victory by comparing RMSE and R Square on outcome variables predicted from training and testing dataset. The performance of the random forest model is also overfitted on the training dataset for the outcome variable of win or lose of the game by comparing the accuracy and F1 score on outcome variables predicted from training and testing dataset. The random forest model outperformed the baseline model.

Evaluation metrics for the random forest on training data:

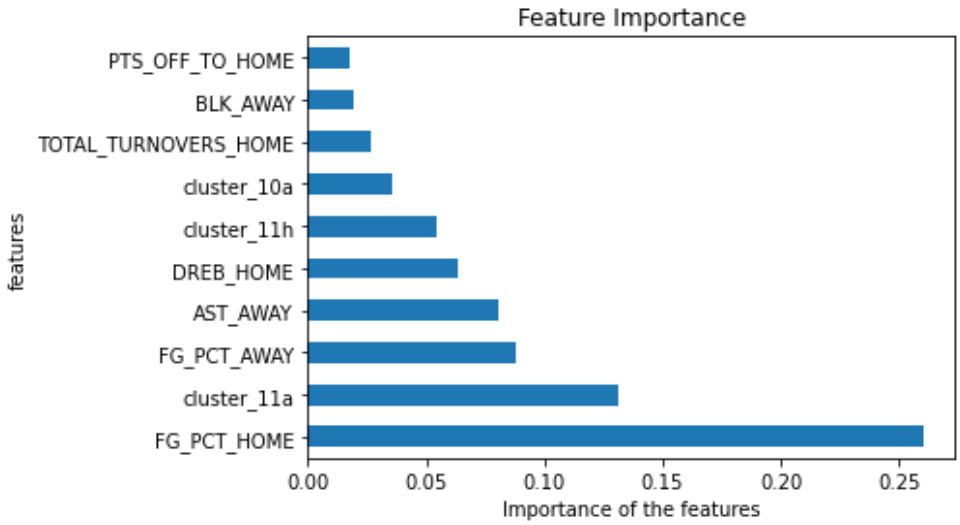
Accuracy	F1 Score	R Square	RMSE
0.752931	0.813033	0.35238	10.691114

Evaluation metrics for the random forest on testing data:

Accuracy	F1 Score	R Square	RMSE
0.645927	0.741213	0.109036	12.664531

Figure 7.3.2 shows the feature importance of the random forest model.

Figure 7.3.2: Feature Importance of the Random Forest Model



The performance of the boosting model is overfitted on the training dataset for the outcome variable of margin of victory by comparing RMSE and R Square on outcome variables predicted from training and testing dataset. The performance of the boosting model is also overfitted on the training dataset for the outcome variable of win or lose of the game by comparing the accuracy and F1 score on outcome variables predicted from training and testing dataset. The boosting model outperformed the baseline model. In addition, the boosting model is also the best performed tree-based model. However, the boosting model has a tradeoff on training time.

Evaluation metrics for XGBoosting on training data:

Accuracy	F1 Score	R Square	RMSE
0.717388	0.785098	0.29884	11.124265

Evaluation metrics for XGBoosting on testing data:

Accuracy	F1 Score	R Square	RMSE
0.659152	0.745854	0.133189	12.491686

Figure 7.3.3 shows the feature importance of the boosting model. *Figure 7.3.4* shows the predicted outcome and actual outcome for both the training and testing dataset.

Figure 7.3.3: Feature Importance of the Boosting Model

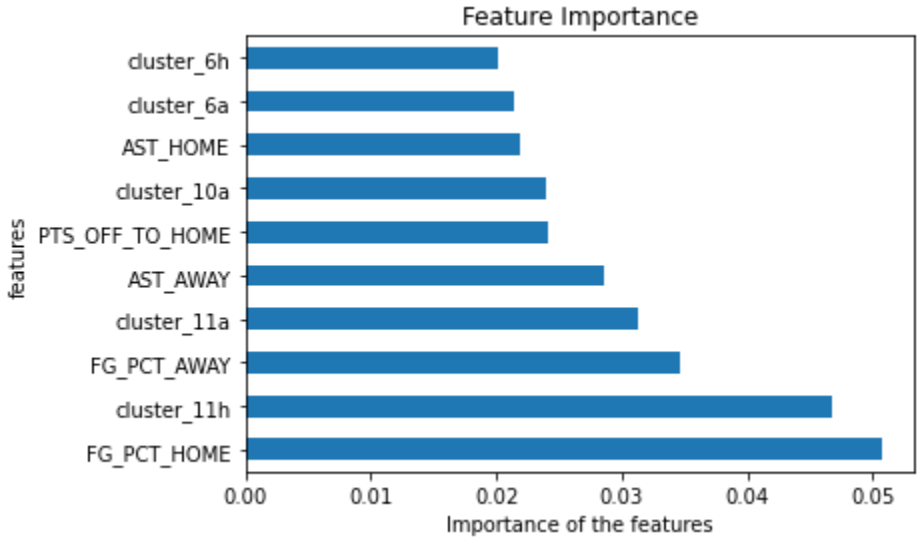
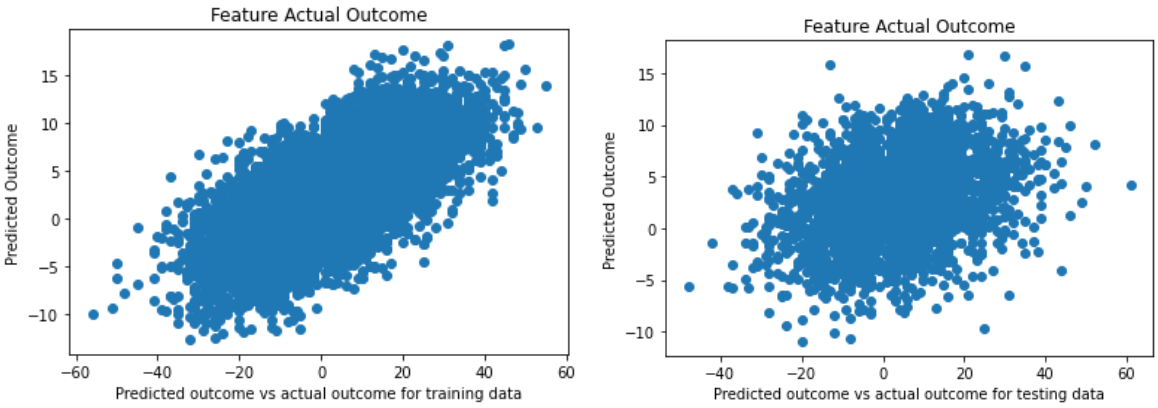


Figure 7.3.4: Predicted Outcome vs Actual Outcome



7.4 Neural Network

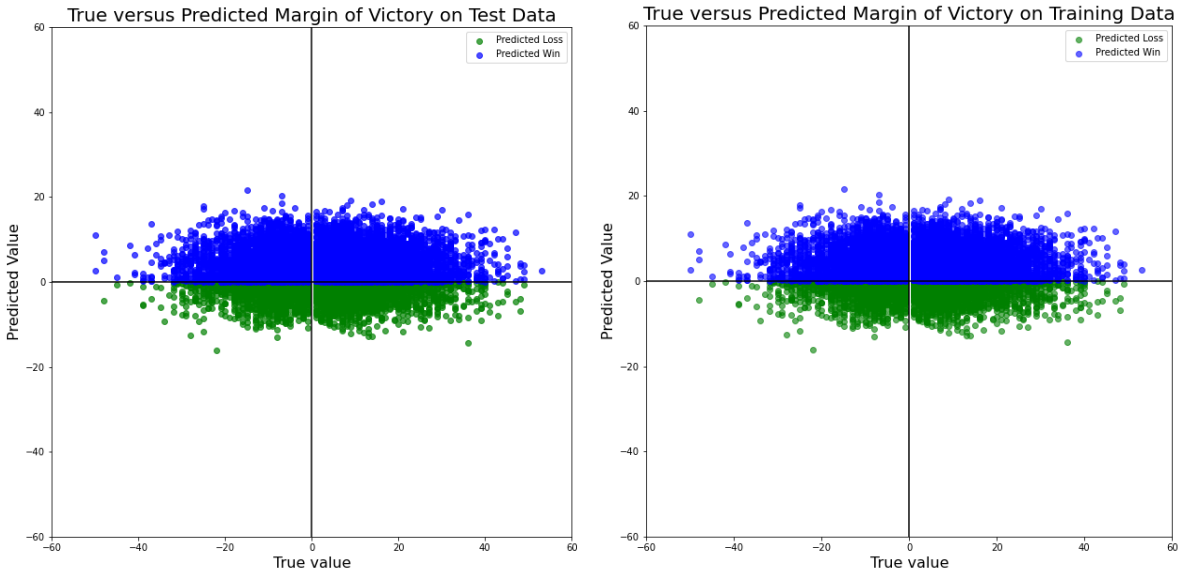
Comparison of the test and training sets shows a modest reduction in performance for the test set. Notably, root mean squared error, a transformation of mean squared error that hyperparameter tuning optimized for, is near equivalent across training and test sets as shown in *Table 7.4.1*.

Scatter plots of predicted versus true values for both the training and test sets show that the neural network predictions have less variance than the true values (*Figure 7.4.1)*. The standard deviation of the test predictions is approximately 4.5 whereas the standard deviation in the test data is approximately 13.1. Our R^2 of 0.117 on the test set is another indicator that we only capture a small portion of the total variance in MOV.

Table 7.4.1: Neural Network Results on Training and Test dataset

Metric	Training Data	Test Data
Accuracy	0.650	0.631
F1 Score	0.736	0.718
R Squared	0.146	0.117
RMSE	12.343	12.342

Figure 7.4.1: Scatterplots of Neural Network MOV Predictions versus True MOV



Section 8: Discussion

8.1 Cluster Analysis

We clustered players under the theory that clusters would be a dense representation of players' contributions to winning basketball games. We also expected players to be clustered by quality and that higher quality clusters would be more influential. In this section, we will check this assumption with a focus on the clusters identified as most important by the XGboost model: 6, 10, and 11.

We begin by looking at some of the notable players in clusters 6, 10, and 11. *Table 8.1.1* includes selected players who are members of the important clusters for at least 50% of their

games played. Cluster 6 contains hall of fame level big men, and Cluster 10 contains hall of fame level point guards. Cluster 11, the most important cluster, contains an array of players who won the NBA's Most Valuable Player (MVP) award. Of the 15 seasons in our dataset, cluster 11 contains the MVP for 12 of the seasons. The 3 remaining MVP's were won by Steve Nash and Derrick Rose who belong to the second most important cluster, cluster 10. MVP awards going back to the 1999-2000 season were won by Shaquille O'Neal, Allen Iverson, Tim Duncan, and Kevin Garnett who are included in clusters 6 and 10.

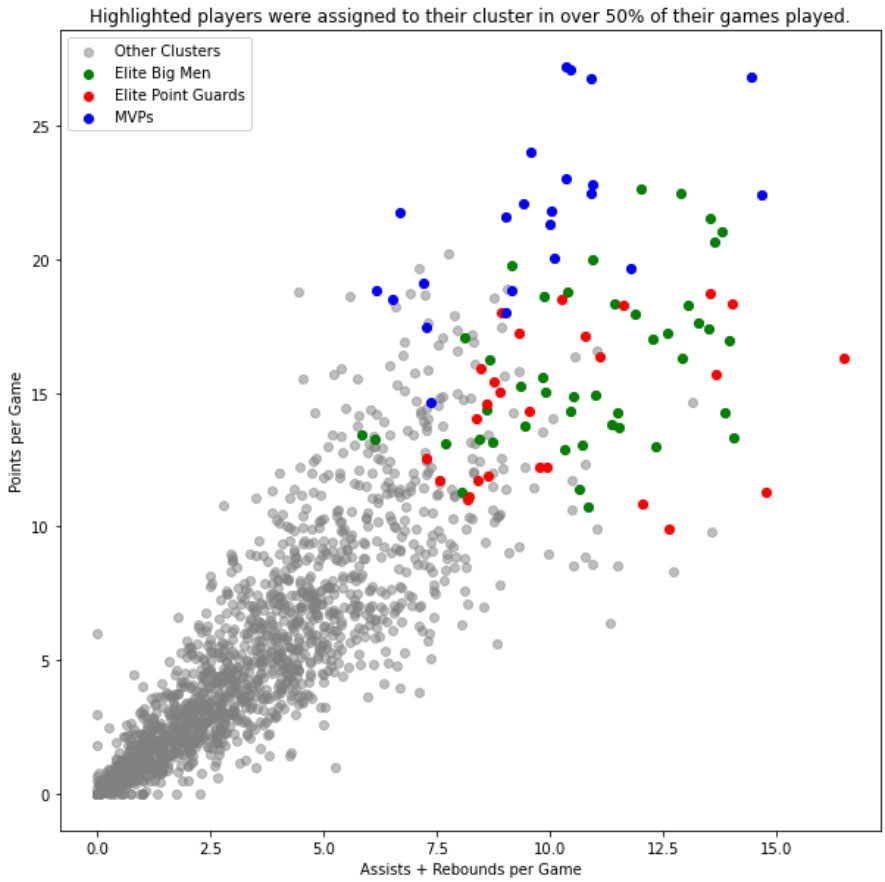
Table 8.1.1: Notable Players by Cluster

Cluster Number	Notable Players	Cluster Description
6	Tim Duncan, Kevin Garnett, Shaquille O'Neal, Anthony Davis, Chris Webber, Chris Bosh, Amar'e Stoudemire	Elite Big men
10	John Wall, Steve Nash, Derrick Rose, Trae Young, Gary Payton, Jason Kidd, Tony Parker	Elite Point Guards
11	Kobe Bryant, Kevin Durant, Lebron James, Stephen Curry, James Harden, Dirk Nowitzki, Giannis Antetokounmpo, Russell Westbrook, Allen Iverson	Most Valuable Players

In *Figure 8.8.1*, we show a scatter plot of the most influential clusters versus all other clusters across rebounds, points, and assists. We observe that players in these clusters score more points and generate more rebounds and/or assists than other players in the dataset. This supports our expectation that the best players would create the informative clusters.

In summation, our clusters appear to provide the information we expect them to. The best players are clustered together, and the clusters with the best players are the most influential in our supervised models. However, we can likely improve our clusters with several changes. First, since our clusters are input into a supervised learning model, we can use supervised model performance to inform how many clusters we generate. Second, a gaussian mixture model (GMM) would allow fractional assignment of players to clusters. Players change clusters over time as their performance changes, and the GMM may better capture performance changes over time. A GMM model may also allow a player to contribute multiple "skill sets" to the extent that clusters are organized around a player's ability to contribute discrete skills such as scoring, shooting, defense, or rebounding.

Figure 8.1.1: Box Score Statistics of Most Influential Clusters by Players' Career Statistics



8.2 Results Analysis

In *Table 8.2.1*, we present the results of all models compared to the baseline. XGBoost appears to provide the best overall performance. It is the only model that outperforms the baseline regression on every metric. It also has the highest accuracy and F1 Score. The linear regression is the preferred regression model as the Lasso and Ridge regressions do not improve results on any of our metrics. The neural network provides the lowest RMSE indicating it generates the most accurate point estimates of MOV.

Gambling environments are an obvious use case for our models. Sportsbooks (i.e. providers of betting opportunities) provide two ways to bet on the final outcome of a game. First, they offer a "money line" which is the odds that a given team will win. Second, they offer a "spread", an estimate of the game's MOV, and allows the bettor to predict rather the MOV will be higher or lower than the spread. If we consider these two use cases, we anticipate the XGBoost model would perform best on the moneyline since it predicts each games winner with the highest accuracy. For the spread, we anticipate the neural network model to perform best since its point estimates of the MOV are most accurate among our models.

Table 8.2.1: Results Summary on Test Data

	Baseline Linear Regression	Linear Regression with Clusters	Lasso Regression	Ridge Regression	Decision Tree	Random Forest	XGBoost	Neural Network
Accuracy	0.646	0.650	0.649	0.649	0.635	0.646	0.659	0.631
F1 Score	0.740	0.737	0.736	0.736	0.736	0.741	0.746	0.718
R Squared	0.101	0.136	0.135	0.135	0.062	0.109	0.133	0.117
RMSE	12.723	12.470	12.479	12.480	12.997	12.665	12.492	12.342

Section 9: Constraints and Limitations

Time was our foremost constraint. Our extensive data engineering process absorbed the majority of our time on this project. This restricted the time allotted to data collection, feature engineering, feature selection, and model selection. For example, more time would allow us to perform hyperparameter tuning on the number of clusters into our model. We chose to use 12 clusters with visual analysis of a dispersion graph. A more advanced approach would use the supervised model error to inform the number of clusters chosen.

All model predictions contained less variance than the observed MOV indicating a need for more and/or more informative features. We consider additional features in *Section 11.*

Section 10: Standards Comparison

Other researchers have generated models to predict NBA games and reported similar results. Cheng et al. (2016)[3] achieved 67.68% accuracy across eight seasons by using a maximum entropy model on box score team statistics. However, this paper focuses on predicting NBA playoff games and only evaluates accuracy. Jones (2016)[4] achieved 62% accuracy on regular season games using a linear regression model. However, this model was developed to predict playoff games as well.

[3] Cheng, Ge; Zhang, Zhenyu; Kyebambe, Moses N; Kimbugwe, Nasser. *Predicting the Outcome of NBA Playoffs Based on the Maximum Entropy Principle*. <https://www.researchgate.net/publication/312236952_Predicting_the_Outcome_of_NBA_Playoffs_Based_on_the_Maximum_Entropy_Principle>

[4] Jones, Eric S. *Predicting outcomes of NBA Basketball Games.* <https://library.ndsu.edu/ir/bitstream/handle/10365/28084/Predicting%20Outcomes%20of%20NBA%20Basketball%20Games.pdf?sequence=1&isAllowed=y>

Though not a published paper, the work by Weiner (2021)[5] is the most analogous model to our own. He differs by including Elo scores, a quality ranking metric, for each team, and examines similar linear, logistic, and random forest models. Weiner achieved an accuracy of 67.15% with a random forest model based on team statistics. Relevant to our study, Weiner also attempted to use aggregated player statistics to predict games. This model only achieved an accuracy of 58.66%. All of our models using clustered player data achieved higher accuracy. This may be an indication that clustering players into features is a productive approach for modeling individual players' contributions to winning games.

Section 11: Future Work

We believe many avenues to improve predictions exist. As mentioned, a more sophisticated clustering strategy might generate better player features. We can also consider more advanced modeling of player statistics. Consider a rookie playing in their first season. In the current framework, this player will not have 82 games of statistics to include in their rolling average until after the entire first season. We might employ a Bayesian model, based on the player's draft position, college statistics, and physical attributes, to estimate the rookie's statistics until they've played a sufficient number of NBA games. The NBA statistics community has also created many different statistics to evaluate player quality. We might try to incorporate these into our model. In particular, more advanced defensive statistics might be helpful as the box score metrics we incorporated tend to have a weak correlation with a team's overall defensive performance.

At the team level, we might add features related to the schedule and location of games. For each team, the frequency of games vary over time. When the team plays more frequently, the team has less time to recover between games. This effect is accentuated if the frequent games align with periods of travel where players move between cities on red eye flights. Assessing the effect of game frequency and travel would likely improve our predictions. We might want to also include location specific effects for each game. For example, the Denver Nuggets play at a higher altitude with thinner air than the rest of the league which increases their home field advantage. Weiner's (2016) results suggest that the inclusion of ELO ratings may be beneficial.

In summation, many avenues of exploration remain for improving our model. However, including player features through clustering appears to be a promising strategy to handle the presence and absence of players in a way aggregate team statistics cannot.

[5] Weiner, Josh. *Predicting the Outcome of NBA Games With Machine Learning.* <https://towardsdatascience.com/predicting-the-outcome-of-nba-games-with-machine-learning-a810bb768f20>