Predictive Modeling for NBA Game Outcomes

Keegan Burr, Information Science B.S., Senior
Rhett Lavender, Data Science B.S., Junior
Adalia Winters, Statistics and Analytics B.S., Junior
Isabella Yeager, Statistics and Analytics B.S., Sophomore

Department of Statistics and Operations Research
University of North Carolina at Chapel Hill

1. Data Information

The primary goal of this project is to use data science methods to design models for predicting three different variables–*Spread, Total,* and *OREB* (offensive rebounds)–in games played by the National Basketball Association (NBA) between March 7 and March 21, 2025. The following basic formulas define the variables we are trying to predict:

$$Spread = Home\ Points - Away\ Points$$
$$Total = Home\ Points + Away\ Points$$
$$OREB = Home\ OREB + Away\ OREB$$

Our starting data was obtained from a GitHub repository titled “NBA-Data-2010-2024” and created by Vitalii Korolyk. The repository contains various CSV files with NBA data from 2010 to 2024, with some files providing information related to player statistics and others offering insights into team performances and overall game outcomes. We decided against using any data from games during the postseason, as our objective is to predict variables derived from outcomes of regular season games. Naturally, postseason games are quite different from regular season games, as each team that makes the postseason is competing for an NBA championship. Coaches and analysts will prepare for each game as best they can, and players will give maximum effort on every possession. With regular season games generally being less competitive, using data from postseason games in our modeling and prediction process did not make sense. The first file in the repository we chose to work with and analyze is titled “regular_season_totals_2010_2024.csv”. Initially, this data set consisted of 33,316 samples where each sample represented one NBA team’s performance in an individual game described by summary statistics. The original 57 features included some contextual information such as the date of the game, where the game took place, and which team won the contest, but most of them described different team-level statistics over the course of the matchup. A team’s total number of field goals made and attempted, their total number of assists and turnovers, and their total number of offensive and defensive rebounds are just a few of the measures that were present within the data.

In an effort to consolidate the data and create the desired target variables, *Spread*, *Total*, and *OREB*, the overall structure of the data set was modified such that each sample now included game information for *both* teams involved. To perform said modification, the original data set was first split in half, creating two new data sets, one that included samples where the team was playing at home, and another consisting of samples where the team was the visitor. This split was done utilizing the MATCHUP feature, which for each game was a string indicating the two teams involved and where the game was being held. A prefix was then added to the names of the columns of the first data set, and a different prefix was given to the names of the columns in the second data set, creating a distinction between the home and visiting team’s statistics. Lastly, a merge was conducted using the unique eight-digit game identifier present in both data sets, and columns for each of the three target variables were created based on the given formulas.

1.1 Advanced Box Scores

The resulting data set contained 22 features for both the home and visiting teams representing basic metrics (points scored, total assists, number of offensive/defensive rebounds) that would appear in traditional NBA box scores. To improve the accuracy of our final predictions, we sought to incorporate more advanced measures, measures typically derived from the aforementioned basic metrics. To illustrate the necessity for these advanced measures, consider the FG% feature in the current data set which represents the percentage of field goals (two- or three-point shots) a team converts over the course of a game.

$$FG\% = \frac{total\ field\ goals\ made}{total\ field\ goals\ attempted}$$

Looking at a single sample from the data set, imagine the home team made 15 field goals while attempting 20, solely taking two-point shots. The visiting team also made 15 field goals on 20 attempts, but all of their field goals were three-point shots. The teams' field goal percentage would be identical, but the visiting team would have won the contest by a score of 45 to 30. The field goal percentage statistic fails to describe this difference.

However, in 2002, statistician and NBA assistant coach Dean Oliver developed the idea of 'effective field goal percentage' in his book, *Basketball on Paper*. This new measure was created to more effectively quantify the impact that made three-point field goals have on a team's offensive efficiency. The formula for effective field goal percentage (EFG%) is as follows, where FGM represents the number of field goals made (two- and three-point shots), 3FGM represents the number of three-point field goals made, and FGA represents the number of field goals attempted:

$$EFG\% = \frac{FGM + 0.5 * 3FGM}{FGA}$$

Applying this formula to our imaginary scenario above, the home team's EFG% would be 75 percent (the same as their FG%), while the visiting team's EFG% would be 112.5 percent! Clearly, EFG% is not a true percentage in the mathematical sense, but it does do a better job of measuring the impact of three-point field goals on a team's offensive capabilities. By including derived metrics like effective field goal percentage in our design process, we hoped to increase the predictive power of our created models.

To obtain these types of advanced metrics, we downloaded advanced box scores from the NBA's official statistics database. The structures of these advanced box scores were similar to the original data set in the sense that each sample represented one team's performance in a single game. This led us to largely repeat the splitting and merging process performed on the original data set. The downloaded box scores were split into two data sets, one where samples represented a team's performance at home, and the other where samples represented a visiting team's performance. Then, because these box scores contained less identifying information, we had to standardize the MATCHUP feature, which again was a string indicating the two teams involved and where the game was being held. Each string was modified to in general read: "HOME vs. AWAY", where HOME was the home team's three-letter abbreviation, and AWAY was the visiting team's three-letter abbreviation. Performing this modification in both the home team and visiting team data sets and adding appropriate home and away prefixes ('H_' and 'A_') to the names of the columns allowed us to then merge them back together on the MATCHUP feature.

With two data sets–one containing our so-called basic metrics and the target variables, the other containing the advanced measures from the NBA's statistics database–spanning the same period of time (2010-2024), we could now perform a merge to create a singular data set with both simple and derived statistics. This merge was performed using the standardized MATCHUP strings and GAME_DATE objects present in both data sets. The resulting data set was composed of 16,658 samples and 86 features.

1.2 Rolling Averages

When making final predictions of *Spread*, *Total*, and *OREB* for NBA games between March 7 and March 21, the values for these basic and advanced statistics are entirely unknown. Thus, the next logical step was the engineering of new features, features that could actually be used in making our predictions. These features had to somehow represent a team's overall performance in games leading up to the games in which we were actually making predictions on. One simple way to examine a team's past performance is to create averages in different statistical categories over a specific period of time. For example, to estimate the number of points a given team might score in their next game, one could choose their estimate to be the average number of points that team has scored in their last three games. This three-game rolling average of points scored roughly represents the given team's overall scoring performance recently. In our analysis, we first chose to create five-game rolling averages for each numeric variable present in the data set. In other words, a column for each statistical category representing a team's performance in said category in their past five games was added to the existing data set. The rolling averages were easily calculated for each numeric variable using the Python data science package *pandas*' groupby() and rolling() functions. For the first five games of the season, these rolling averages were not calculated, as going into those games, teams had not yet played five games. Repeating this process, we also engineered 10-game rolling averages for each numeric feature, hoping that by examining a team's performance over a larger window of games we might boost our final model's predictive power. The creation of a 10-game rolling average made possible the formulation of our next type of predictive feature.

1.3 Momentum Indicators

Anyone who watches sporting events on a consistent basis has probably seen the concept of momentum in action. While momentum is subjective in nature, and its influence on the outcome of a game varies from sport to sport, it has been observed and studied by researchers and analysts for years. Basketball, in particular, is known as “a game of runs”, meaning that teams typically do not just alternate scores, but instead, go on more unbalanced stretches (or runs) of scoring, stretches that are undeniably impacted by momentum. Take for example, a basketball game played in Carmichael Arena (Chapel Hill, NC) between the University of North Carolina at Chapel Hill and Duke University on March 2, 1974. Duke was winning the contest by eight points, 86-78, with just 17 seconds to go. Then, UNC-CH went on an 8-0 run forcing overtime, where they went on to win the game 96-92. Some might call it a miracle, but others might argue that UNC-CH’s players fed off the energy of the home crowd and began to play better as the score got closer and closer, eventually flipping in their favor. There are countless other examples of momentum being a factor in basketball at all levels of play. We sought to capture this phenomenon by engineering a new type of variable, which we referred to as ‘momentum indicators’. A momentum indicator for any given metric was calculated as such, where AVG5 represents a team’s five-game rolling average for the given metric, and where AVG10 represents a team’s 10-game rolling average for the given metric:

$$MOMENTUM = \frac{AVG5 - AVG10}{AVG10}$$

Momentum indicators then represent a percent change, ranging from zero to one. They quantify how much better a team is performing in a statistical category over their past five games relative to their past 10 games. If a team’s momentum indicator for points scored takes on a large negative value, this suggests that the team’s scoring has been lower than usual as of late. We would predict this team to score slightly fewer points than usual in their upcoming matchups based on the calculated trend. These measures of momentum are particularly useful in capturing the effects of shooting streaks and slumps, lineup or position changes, and offensive and defensive scheme adjustments made during the season.

1.4 Win Percentages

Up to this point, our data set has averages over two different lengths of the NBA regular season, as well as measures of momentum in both simple and advanced aspects of a basketball game. The rolling averages, particularly in areas such as points scored, steals and blocks, and offensive rebound percentage, will serve as a baseline for our *Spread*, *Total*, and *OREB* predictions, giving us a solid idea of how each team has been performing in different factors over the course of the given season. Our measures of momentum will capture performance trends for each team in each category. Additionally, including these indicators may help account for injuries, trades, and scheme adjustments that occur mid-season, all of which influence our target variables. There is not a predictive feature in the data set that serves as a measure of overall team strength, however. It is essential that we understand the skill level of each team, especially in predicting *Spread*. If the best team in the NBA plays the worst team in the NBA, it is likely that the spread of that contest will be relatively large. To remedy this, we decided to track the win percentage of each team over the course of each season in the data set. It was necessary to calculate win percentage as opposed to total number of wins because the 2024-25 regular season is actively ongoing, and teams have played different numbers of games. This was more easily done using the original data set, where each sample quantified one NBA team's performance in a single game, not two. The WL binary indicator, which was originally a string–'W' if the home team won the game, 'L' if the home team lost the game–was mapped to take on a value of one if the home team won the game, and zero if the home team lost the game. Again utilizing the *pandas* groupby() function, this time in combination with another *pandas* function, cumsum(), a series containing win totals for each team during each season was created. Then the *pandas* cumcount() function was used to create a series containing the number of games played by each team during each season. The WIN_PCT feature, which represents a team's win percentage going into a given game represented by its corresponding sample, was then added to the original data set by dividing the win totals series by the games played series. Finally, the splitting and merging process instituted previously was performed once again to appropriately create H_WIN_PCT and A_WIN_PCT columns in the current data set, which track the home and visiting teams' winning percentage over the course of a given season, respectively.

1.5 Betting Lines

The target variables *Spread* and *Total* are quantities that are commonly bet on by individuals, primarily in American football and basketball. Betters might choose to put some amount of money on one team to 'cover' the spread or on the two teams to combine to score fewer points than the 'over/under' line. The spread can be defined as the number of points one team is projected to win by. A team covers the spread if they win by *more* than the value of the spread. So, if the set spread is seven in favor of the home team, and the home team beats the visitor by more than seven points, the home team has covered the spread. The over/under line (or total) is a quantity that represents the total number of points the two teams in the game are projected to combine to score. Imagine the over/under line is 220.5 for some NBA game. If an individual "bets the under" on this line, they believe that the two teams will combine to score less than 220.5 points. The spread and the over/under line are set by professional oddsmakers and meant to be fairly accurate so that it is difficult for bettors to come to a final decision, maximizing profit for sports gambling companies. Therefore, it is reasonable to believe that the addition of closing spread and over/under lines for each game in the data set would aid our final models in predicting *Spread* and *Total*.

Historical spread and over/under lines were pulled in from the sports betting website "covers.com" via Scrapy, a web-scraping framework written in Python. A short script that manipulated the structure of HTML elements of each team's webpage for each season wrote the betting lines into a JSON file alongside identifying information. These identifiers–things like the date of the game, home and visiting team abbreviations, and where the game was played–allowed for a relatively simple merge process. After reading in the JSON file to a *pandas* dataframe, some columns had to be renamed for the merge, and duplicate samples were dropped. It was then possible to merge the betting information data with our current working data set on the GAME_DATE, H_TEAM_ABBREVIATION, and A_TEAM_ABBREVIATION features.

1.6 Player Fatigue

The NBA regular season consists of 82 games played over the course of roughly seven months. Within the two-week period of games we are making predictions for, each of the association's 30 teams play anywhere from seven to nine times. Teams can even wind up playing on back-to-back days. A regular season of this nature turns player fatigue into a much more influential factor. It seems plausible that a team's performance might be a bit worse than average if they have played three games in five days, possibly traveling hundreds of miles in that timeframe as well. It was this reasoning that led to the creation of additional features related to player fatigue. The most simple of those is the rest day variable, which represents the number of days since a team's last game. Separate rest day variables were created for both the home and visiting teams by subtracting the team's previous game date from the current game date being looked at. We thought this variable would be useful in understanding how fatigue plays a role in how well each individual plays as well as how the team plays as a whole, and knowing how many rest days a team has had could help us understand how a team will perform, as one would assume players normally play better with a longer rest period. This variable can also account for changes in coaching style. If games are closer together, a coach may decide to increase player rotation or make other changes that will influence the *Spread*, *Total*, and *OREB*.

Then, to enhance the model's interpretability, we created a rest-days difference variable (REST_DAYS_DIFF), representing the difference in rest days between the home and away teams. The goal of adding this variable was to simplify the model's ability to analyze the relative advantage or disadvantage in rest, providing a more direct measure of a fatigue imbalance between the two different teams.

Finally, with rest days, we created two binary indicator variables to capture whether either team was playing a back-to-back in their schedule. Denoted by H_BACK_TO_BACK and A_BACK_TO_BACK, these variables took a value of 1 if the home or away team played a game the previous day. In most cases where the team did not play the previous day, a 0 was filled in for that indicator. The reason behind including these variables, although similar to the rest days variables, was to ensure any model chosen could distinguish particularly high-fatigue situations from general short-rest scenarios, accounting for scenarios where teams might experience heightened fatigue, potentially impacting gameplay intensity, player endurance, and strategic decisions.

1.7 Historical Matchups

To capture specific trends present within certain matchups, we created three historical matchup variables SPREAD_matchup_last5, TOTAL_matchup_last5, and OREB_matchup_last5. These variables represent the average spread, total points, and offensive rebounds, respectively, from the last five games in which the same two teams competed against one another. Our hypothesis for creating these was that historical matchups would offer valuable insights by uncovering patterns that might persist in future games. In order to make these variables, we had to first normalize the matchup column to avoid inconsistencies in the order of team names. By applying a sorting function in Python to standardize matchup names, we ensured that the data was accurate regardless of which team was “home” or “away” in the previous matchups. Similar to our other created variables, we then utilized rolling averages within our grouped data to calculate the mean of these metrics over the last five games of our normalized matchups.

In the end, we decided to include these variables in our data set for multiple reasons. First, it helped us capture unique team-specific tendencies, such as rivalry games or stylistic matchups that may lead to consistently higher or lower-scoring games. In particular, applying this approach to offensive rebounding provided our model with an advantage by offering deeper insights into how specific player matchups between teams influence rebounding outcomes. Additionally, by making these variables rolling averages, it considers the most recent interactions between teams, which is critical because recency often influences predictability. Lastly, averaging the last five matchups helped us smooth out outliers in single-game data, making it a more reliable indicator for the model. Ultimately, incorporating these past matchup variables was to help our model go beyond the generic league-wide trends and truly understand the specific historical dynamics in unique matchups.

1.8 Updated Box Scores

One of the most crucial steps taken in building our data set for the eventual fitting, evaluation, and implementation of various predictive models was the incorporation of team-level statistics from the 2024-25 (current) NBA regular season. After all, our goal is to predict specific outcomes of a set of NBA games played *during* the 2024-25 season, so it follows that the inclusion of the most recent data would make a positive difference in predictive power. To obtain this information, both traditional and advanced team box scores for the 2024-25 season so far were downloaded from the official NBA statistics database. After merging the traditional statistics with the advanced metrics (once again performed using the GAME_DATE and MATCHUP features), rolling averages, momentum indicators, win percentages, closing betting lines, fatigue-related variables, and historical matchup features were still missing from the temporary data set representing the current season's games. To remedy this, the feature engineering procedures described in previous sections were simply reapplied to the temporary data set, creating columns representing all of our predictive features. The temporary data set was then appended to the larger data set (which contained game information for NBA regular seasons between 2010 and 2024) using the *pandas* function, concat().

1.9 Finalization and Train-Test Split

At this point, our current working data set consisted of 211 features we might use in predicting *Spread*, *Total*, and *OREB*. With this many features, both multicollinearity and overfitting become a concern. Utilizing a correlation matrix, we were able to identify pairs of highly correlated predictor variables and remove one of the two from the data set. In our analysis, a pair of predictors qualified as highly correlated if the absolute value of their calculated correlation coefficient was greater than 0.9. By removing highly correlated predictors, we reduce redundancy and inherently increase model interpretability. Overfitting is also combatted as the number of features present in the data is lowered.

For better or for worse, the game of basketball at the professional level has changed dramatically since 2010. According to "basketball-reference.com", during the 2010-11 NBA regular season, on average, teams attempted 18 three-pointers per game. This season, on average, teams are taking 37.5 three-pointers per game, more than *double* the number of attempts per game roughly 15 years ago. Three-point field goal percentage has remained fairly constant over this 15-year span, but with significantly higher numbers of three-pointers being taken in the modern NBA, teams are *making* higher numbers of three-pointers each game, leading to a league-wide increase in scoring. In fact, during the 2010-11 regular season, teams across the association scored on average 99.6 points per game. Currently, teams are, on average, scoring 113.6 points per game. Various rule changes, explosive financial growth, and the rise of analytics and data-driven decision-making over the past 15 years have also contributed to the evolution of the NBA. Taking all of this into consideration, it was clear that the scope of our analysis had to be reduced in terms of time.

It was decided that our predictive models would be based on the past five NBA regular seasons, as well as the current (2024-25) regular season. By the 2019-20 regular season, three-point shooting had become a critical part of each game, and the league average in points scored was quite high. It made more sense to base our predictions for a set of games played during the 2024-25 season on more recent, similar games and seasons. After removing games played before the 2019-20 regular season from our data set, the samples were initially split into a training and testing data set based on the season in which each sample occurred. Samples from the 2019-20, 2020-21, 2021-22, 2022-23, and 2023-24 seasons were placed into the train set, and the samples from the 2024-25 season were placed into the test set. We quickly realized the methodology by which our train/test split was performed was flawed. Since our final predictions are to be made on NBA games played between March 7 and March 21, which is just a two-week period of time, it did not make sense for our train set to include *all* games from the past five regular seasons, nor did it make sense for our test set to just include games from the current regular season. The training data set was altered to contain only the games played between the start of the regular season up to March 6 for each season since 2019-20. The testing data set was modified such that it was now composed of only the games played between March 7 and 21 for each season since 2019-20, excluding the 2024-25 season. This split more effectively reflected the circumstances in which we are making our final predictions. Samples that contained any NaN (empty) values were then dropped from both the train and test set. NaN values were largely present in samples towards the beginning of a season, where five- and 10-game rolling averages could not be accurately calculated. The finalized training set consisted of 4,815 samples, while the finalized testing set consisted of 449 samples, resulting in a rough 90/10 train/test split.

2. Methodology for $Spread$

To predict the *Spread* of the NBA games in the 2024-25 season between March 7 and March 21, we first tested a multitude of modeling techniques. We looked at simple linear regression and implemented regularization and stepwise algorithms for variable selection. We also investigated the effect of interaction terms in these linear models. Lastly, we attempted to implement an ensemble machine learning method to find the best model for predicting *Spread*.

2.1 Linear and Stepwise Regression

A linear model using only the home and visiting team names as its predictors served as a simple baseline. This model evaluated on our testing set after being fitted on the training set produced a mean absolute error (MAE) of 11.1669. Stepwise regression was then used to find the best linear model utilizing a combination of forward selection and backward elimination algorithms. Traditionally, stepwise regression is evaluated based on Akaike's Information Criterion (AIC), but we adapted the process to track MAE at each step. An initial model was constructed using all predictors. After a predictor was removed or added and the new model was evaluated on the testing set, the MAE was calculated and stored. The model that produced the lowest MAE, which was 10.8671, was kept as the final model.

2.2 Regularized Regression

Next, we looked at ridge, lasso, and elastic net models. Ridge regression introduces an additional penalty to the model to shrink coefficients, which helps to reduce overfitting. Coefficients do not reach zero, which means that all variables remain in the model, but their impact is controlled. Unlike ridge regression, lasso regression can shrink coefficients to exactly zero, so only the most important variables are kept in the model. After fitting a lasso regression model, we were able to identify the best predictors and create interaction terms based on them. One of the interaction terms created was between H_WIN_PCT and A_WIN_PCT. Our reason behind the creation of this variable was that if two teams with a high win percentage play each other, the *Spread* will be smaller than if a team with a high win percentage plays a team with a low win percentage. Another term we created was between H_NetRtg_AVG_10 and A_NetRtg_AVG_10. This term looks at both the efficiency of the offense and defense for each team's last ten games. The reason for the creation of this term is very similar to the previous term. If both teams have had a high net rating in their most recent games, this could predict a closer *Spread*. Elastic net regression is a mix of the previous two models discussed. It handles predictors better when they are highly correlated as compared to lasso regression and helps to reduce the excessive shrinking that can occur with lasso regression. For these model types, the training and testing data described in section 1.9 needed to be split further. To prepare the data for the models the training data was split into two data sets named "X_Train" and "Y_Train". X_Train contained the training data set with all features except *Spread.* Y_Train contained only the data from the training set for *Spread* only. These are the inputs in the model. The same was repeated for the testing data set. X_Test was used when generating predictions, and Y_Test was used to compare the predictions to the actual values.

The parameter lambda controls the strength of the penalty for all three models, as lambda increases, the penalty increases. The default number of lambdas tried is 100 when using cv.glmnet to perform regularization. For each value of lambda, the model was fitted, and its cross-validation error was computed. The lambda used in the final model produced the lowest MAE after cross-validation. Alpha is the parameter that specifies the type of regularization, either L1 or L2 regularization. In ridge regression, alpha is set to zero which means lambda is proportional to the square of the coefficients (L2 regularization). With lasso regression, which is when alpha is set to be one, lambda is proportional to the absolute value of the coefficients (L1 regularization). For elastic net regression, alpha is set to be between zero and one, and the penalty combines L2 regularization and L1 regularization. The standardized parameter was set equal to true because without this the penalty will not be applied evenly to each predictor. To find the best alpha value, instead of looking at all 3 model types separately, we looped through values from zero to one with a step size of 0.01. Cross-validation was performed for each alpha and matching best lambda and was evaluated based on MAE. The alpha value that produced the lowest MAE, an elastic net model because alpha was equal to 0.57, was chosen. The resulting MAE was 10.4345.

2.3 Random Forest

After fitting and evaluating a variety of linear models, we chose to implement an ensemble method called random forest. An ensemble method combines the predictions of multiple models, commonly referred to as 'weak learners', to make stronger, more accurate predictions. The random forest algorithm builds multiple decision trees using bootstrapped samples of the data. Each tree created is trained on a different subset of the data due to the use of bootstrapping, making each tree unique. When creating each tree a subset of features is selected randomly to be used in splitting the data rather than using all available features, adding diversity to the trees. The starting point for choosing the size of the subset of features to be used in the creation of each individual tree is typically the square root of the data set's total number of features. Once all *n* decision trees are built, each tree makes its own prediction. Since we are dealing with a regression problem, the random forest's final prediction is the average of the *n* decision trees' predictions. The randomness in data samples and feature selection helps to prevent the model from overfitting, in turn making the predictions more accurate and reliable.

We chose to fit a very simple random forest regressor using the Python library *sklearn*. All parameters took on default values excluding the criterion by which data splits were evaluated in the creation of each decision tree. By default, decision tree splits are constructed so that mean squared error is minimized, but in this case, we sought to minimize mean absolute error. Setting parameters to the default meant that our random forest regressor was composed of 100 decision trees (*n_estimators* = 100), the depth of each decision tree was "unbounded" (*max_depth* = None), and the minimum number of samples required to create a split within a decision tree was two (*min_samples_split* = 2). Given more time and computational power, it would have been worthwhile to test different values for the parameters of the random forest model, as the right parameters can make a significant difference in terms of predictive power. This basic random forest regressor produced a mean absolute error value of approximately 10.4855.

2.4 Best Model

Our best model for predicting *Spread* was the elastic net model. This model had the lowest MAE compared to all other models created. Although the MAE for the random forest model was extremely similar, that model was much more complex, making the elastic net model the better choice.

Table 1: Parameters and MAE for the best model for *Spread*

Best Model	*Alpha*	*Lambda*	*MAE*
Elastic Net	.57	0.3921	10.4345

Table 2: Coefficients of variables in the best model for *Spread*

Variable Name	*Coefficient*
Intercept	3.5611
H_PLUS_MINUS_AVG_10	0.0583

H_NetRtg_AVG_10	0.0995
H_PIE_AVG_10	0.0111
H_WIN_PCT	11.4422
A_PLUS_MINUS_AVG_10	-0.0675
A_NetRtg_AVG_10	-0.1010
A_PIE_AVG_10	-0.0880
A_WIN_PCT	-7.0292
SPREAD_matchup_last5	0.0971

The variables that were most useful in predicting *Spread* were H_NetRtg_AVG_10, A_NetRtg_AVG_10, H_WIN_PCT, A_WIN_PCT, and SPREAD_matchup_last5. These variables appeared in both the stepwise and elastic net regression models. H_WIN_PCT, A_WIN_PCT, and SPREAD_matchup_last5 also showed up in the top five most important features in the random forest's decision-making, along with closing_spread and A_PLUS_MINUS_AVG_10. In the elastic net model, the larger coefficients for both the home and away WIN_PCT variables, suggest that they are important predictors for *Spread*. The SPREAD_matchup_last5 variable represents the average spread over the last 5 times the teams played each other and was described in detail in the data information section. This variable was created specifically to help predict spread, and once it was added to the dataset and all models were rerun, it significantly improved the MAE for any model it was included in. All interaction variables we created were useless in predicting *Spread,* as they were not chosen as predictors in any of the models. Both 5 and 10 game rolling averages were created for all the original statistics in the data, but all the 5 game rolling average variables were useless in each of the models we created, only the 10 game rolling averages were chosen to be included.

2.5 Creating Predictions for *Spread*

In order to generate predictions for future games using the elastic net model shown above, each variable in Table 2 needed to be filled in for the set of games we were predicting. To address this challenge of unknown variables, we decided to backfill the data. Specifically, for each metric needed in the prediction, we populated the variables with the most recent data available for each team. This approach involved identifying the latest completed game for each team and using those performance metrics as inputs for the prediction model.

While this backfilling strategy allowed our model to generate predictions when fresh data was not yet available, we do recognize that this approach has its limitations. Backfilling essentially assumes that a team's most recent performance is representative of its current form, which, as we know, may not always hold true due to things like injuries, trades, changing team dynamics, etc. Additionally, this method does not account for short-term trends or momentum that could significantly change, although fortunately, for our spread model, variables like momentum were not used. Ultimately, while backfilling was a practical and effective solution given the constraints, we would have liked to explore alternative strategies for handling unknown variables.

3. Methodology for $Total$

To predict the *Total* score of the upcoming NBA games, we took a systematic, multi-step approach, considering various model types, transformations, and evaluation metrics. We leaned on a traditional linear model as a baseline and then used regularization techniques to improve our predictions.

3.1 Basic Linear and Stepwise Regression

We began with a simple linear regression model, using a subset of predictors, as the baseline for comparison with more advanced methods. The predictions include basic features like home team and away team identifiers. The linear model resulted in an MAE of 15.6371, which helped establish a performance benchmark for other models.

We then applied stepwise regression, a more refined approach to building linear models. By utilizing forward selection and backward elimination, we evaluated combinations of predictors to identify the most impactful variables. Unlike traditional approaches that focus on AIC (Akaike Information Criterion), we focused on optimizing the MAE at each step. After several iterations, the final model—optimized for MAE—yielded an MAE of 14.0023.

3.2 Regularized Regression

Next, we explored regularization techniques—ridge regression, lasso regression, and elastic net regression—to prevent overfitting and improve the model's generalizability. The alpha parameter specifies the type of regularization applied. Ridge regression, which is when alpha is zero , applies an L2 penalty that shrinks coefficients toward zero but does not eliminate predictors entirely. This approach helped to reduce model complexity without discarding valuable features. Lasso regression, which is when alpha is one, applies an L1 penalty, which can shrink some coefficients to zero, effectively eliminating less relevant features. Elastic net regression, which is when alpha is between zero and one, combines both L1 and L2 penalties, providing a more balanced solution. This model is particularly useful when predictors are highly correlated, as it tends to perform better than lasso in such scenarios. The data for the models was prepared as described in the previous section, but with *Total*, instead of *Spread* being isolated, and the standardization parameter was set equal to true.

To find the best alpha value, we looped through values zero to one with a step of size .01. We utilized cross-validation of different lambda values for each model and its corresponding alpha parameter, which controls the balance between L1 and L2 regularization. After rigorous cross-validation, we found that elastic net regression with an alpha value of .15 minimized the MAE, which was calculated at 13.8486.

3.3 Best Model

Our best model for predicting *Total* was the elastic net model. This model produced the lowest MAE out of all the models created.

Table 3: Parameters and MAE for the best model for *Total*

Best Model	*Alpha*	*Lambda*	*MAE*
Elastic Net	.15	2.1258	13.8486

Table 4: Coefficients of variables in the best model for *Total*

Variable Name	*Coefficient*
Intercept	35.6551
H_PACE_AVG_5	0.0596
H_STL_AVG_10	-0.0996
H_OffRtg_AVG_10	0.0407
H_PACE_AVG_10	0.0156
A_PACE_AVG_5	0.0285
A_FGA_AVG_10	0.0210
A_PACE_AVG_10	0.0022
closing_total	0.6832
TOTAL_matchup_last5	0.0888

After finalizing the best model, we assessed which predictors had the most significant influence on *Total*. Key predictors for *Total* included H_STL_AVG_10, closing_total, and TOTAL_matchup_last5. H_STL_AVG_10 measures the home team's steals average over the ten games and has a negative coefficient of -0.0996, suggesting a negative impact on the home team's outcome. closing_total measures the total points predicted for the game based on past betting data, and its coefficient of 0.6832 indicates that it has a strong positive impact on the outcome. Lastly, TOTAL_matchup_last5 with a coefficient of 0.0888, indicates the importance of the total points scored by both teams in their last five matchups. Some variables, such as HOME_AVG_REBOUND and AWAY_AVG_REBOUND, did not contribute to the prediction of *Total*. These variables were either too weakly correlated with the outcome or provided redundant information already captured by other features.

3.4 Creating Predictions for *Total*

When generating predictions for future games using the elastic net model, we again utilized the backfilling approach for unknown variables. For missing data, we used the most recent game metrics, assuming that the team performance in their latest game was the most relevant and similar to the current state of the team. All the rolling average variables were similarly done from their latest matchups to provide a snapshot of those historical trends.

4. Methodology for $OREB$

To predict *OREB* for the NBA games in the 2024-25 season from March 7 to March 21, we used a mix of regression techniques, as well as one ensemble method in an attempt to minimize MAE. *OREB* is a variable that is quite different from the *Spread* and *Total* variables in the sense that it takes on a smaller range of values. It is also related to an entirely different area of basketball. *Spread* and *Total* are calculated using the number of points scored by both teams, meaning that predictive features such as the 10-game rolling average of three-point field goals made for each team might be useful in generating predictions. *OREB* should in theory depend on a different set of features than *Spread* and *Total*. It is also entirely possible that our current working data set does not include a great set of features for predicting *OREB*. In retrospect, it may have been useful to create variables related to physical characteristics of each team's players, as measures such as height and weight can play a critical role when it comes to rebounding.

4.1 Linear Regression

We started with a simple linear model to have a baseline to compare more complex models to. This model contained just the home and away team names as predictors and produced an MAE of 4.2175 when evaluated on the testing data. Next, we used a mix of forward selection and backward elimination, tracking MAE at each step, to find the best linear model to predict *OREB*. The resulting model produced an MAE of 4.1247.

4.2 Generalized Linear Models

OREB is a count variable that measures the number of offensive rebounds that occur in a game, which is why the next modeling technique we looked at was Poisson regression. This is a type of generalized linear model that is used to model count data. One of the key assumptions of Poisson regression is that the variance is equal to the mean, if it is not, this suggests that the data is overdispersed. This is because in the probability mass function that defines the Poisson distribution, lambda represents both the variance and the mean. The variance of *OREB* in our data set was 28.37, and the mean was 20.77, which means the data is overdispersed. When data is overdispersed, a different model, such as negative binomial regression would be better suited for the data. The negative binomial regression, which is a generalization of Poisson regression, introduces an additional parameter that accounts for the extra variance. The glm.nb function was used to fit the negative binomial regression model. A mix of forward and back selection was used to determine the model that produced the best MAE. After a predictor was added or taken away, the model was fitted on the testing set and evaluated on the training set described in the data information section. The end model had an MAE of 4.0505.

4.3 Regularized Regression

We next looked at ridge, lasso, and elastic net regression. These three regression types help to reduce multicollinearity and place higher importance on the most valuable variables. The alpha parameter determines the type of regularization applied. L2 regularization is applied if alpha is zero , L1 regularization if alpha is one, or a mix of both types if alpha is between zero and one. The lambda value determines the strength of the penalty that reduces coefficient values. To prepare the data for the model both, the training and testing sets were split into two data sets each. The X data set for both training and testing contained every variable except *OREB*. The Y data set contained only the *OREB* variable for both training and testing. The two training data sets were the inputs in the model. The X_Test set was what the model was evaluated on and the Y_Test set was used to evaluate MAE. The standardized parameter was set to be true to evenly penalize each predictor. To find the alpha and matching lambda value that resulted in the lowest MAE, a loop that went through alpha values from zero to one with a step size of .01 was run. The resulting model had an MAE of 4.0050.

4.4 Adaptive Boosting

Although the ensemble method adaptive boosting, or AdaBoost, is typically most effective when applied to binary classification problems, we decided to test an AdaBoost model for predicting *OREB*. This is due to the fact that the *OREB* target variable takes on a much smaller range of discrete values than the *Spread* and *Total* target variables. The AdaBoost algorithm is similar to the random forest algorithm tested in our *Spread* methodology in the sense that it combines weak learners to formulate stronger, more reliable predictions. Unlike random forest, the algorithm begins by assigning equal weights to all of the train set samples. These weights represent the 'importance' of each sample during the training process. Then, a decision tree of maximum depth *n* is fitted, where *n* is the total number of predictive features in the train set. Next, 'total error' is calculated, which is defined as the number of incorrectly predicted samples divided by the total number of samples. In the case of a regression problem, the tree's predictions do not need to match up with the true values present in the test set *exactly*. Instead, a margin of error is specified in the creation of the model, and a prediction is counted as incorrect if it falls outside this margin. AdaBoost then works to quantify this tree's 'influence' based on the total error. Influence (α) is calculated as such:

$$\alpha = \frac{1}{2} ln(\frac{1 - TotalError}{Total\ Error})$$

Using the tree's influence value, new sample weights are determined by the following formula, where w_{new} is a new sample weight and w_{old} is an existing sample weight:

$$w_{new} = w_{old} \cdot e^{\pm\alpha}$$

The sign of the influence value used in the formula depends on whether or not the sample the new weight is being calculated for was "correctly" predicted. After each weight is recalculated, the weights are normalized by dividing each weight by the total sum of all weights. Based on the new sample weights, 'bins' are formed for each sample. The first sample's bin takes on values beginning at zero and ending at the value for the sample's weight. The second sample's bin takes values starting where the first sample's bin ended, and ending at the sum of the starting value and the second sample's weight. Essentially, each bin covers a range of values equal to its corresponding sample's weight. So, samples with larger weights will have larger bins. Now, bootstrapping is utilized to create a new data set to fit a second decision tree on. A random number between zero and one (inclusive) is generated for the number of samples in the train set. Depending on which bin these random values fall within, a sample is chosen and added to the new data set. This translates to: more heavily weighted samples are more likely to be featured in the new data set.

The algorithm has now returned to its starting step, the fitting of a decision tree. The process detailed above (calculation of influence, new sample weights, and creation of a bootstrapped data set) is performed for a predetermined number of iterations. In our case, the algorithm was performed 50 times to create 50 decision trees, each fit on a new data set that was constructed in a manner to minimize previous trees' mistakes. Using the Python library *sklearn*, we fit an AdaBoost regressor with default parameter values. The number of estimators (trees), the type of estimator, weight applied to each regressor at each iteration, and loss function can be easily manipulated, but much like with the implementation of the random forest algorithm, time and computational power were an issue. Our basic AdaBoost regressor fit to predict *OREB* yielded a mean absolute error value of approximately 4.1359.

4.5 Best Model

The model created using lasso regression was chosen as our best model for predicting *OREB*. The lasso regression model produced a slightly lower MAE, and was a much simpler model with less terms then the models produced through negative binomial regression and AdaBoost.

Table 5: Parameters and MAE for the best model for *OREB*

Best Model	*Alpha*	*Lambda*	*MAE*
Lasso	1	0.1048	4.0050

Table 6: Coefficients of variables in the best model for *OREB*

Variable Name	*Coefficient*
Intercept	11.6715
H_OREB_AVG_10	0.2407
A_OREB_AVG_10	0.2919
OREB_matchup_last5	0.1776

The three variables in Table 6 were the most important variables for predicting *OREB*. When the alpha value was lowered, which resulted in a lower MAE, H_OREB_AVG_5 and closing_total were added to the model, but had very small coefficients and made the predictions worse. Neither H_OREB_AVG_5 or A_OREB_AVG_5 showed up in any of the models created, making them useless in predicting *OREB*.

4.6 Creating Predictions for *OREB*

To generate *OREB* predictions for the future games between March 7 and March 21 using the lasso model, we once more used the backfilling approach for unknown variables. For missing data, we used the most recent game metrics, assuming that the team performance in their latest game was the most relevant and similar to the current state of the team. The rolling average variables were similarly taken from their latest matchups to provide a snapshot of historical trends.