

NBA Playoff Predictive Modeling Project

Abisai Lujan, Alexandra Myers, Banks Totten, Thadius Trone, Veronica
Upadhyay, Zhejun Zhu

March 7, 2025

1. Data Information

1.1 Starting Datasets

This paper examines NBA Games over 14 years to make game predictions for the 2025 season. We use datasets compiled by Vitalii Korolyk (NocturneBear on Github¹) from the repository “NBA-Data-2010-2024”, which contains comprehensive NBA data spanning from the year 2010 to 2024. We used all five datasets from the repository:

CSV File Name	Henceforth referred to as,
play_off_box_scores_2010_2024.csv	PO Box Scores
play_off_totals_2010_2024.csv	PO Totals
regular_season_box_scores_2010_2024_part_1.csv	RS Box Scores Pt 1
regular_season_box_scores_2010_2024_part_2.csv	RS Box Scores Pt 2
regular_season_box_scores_2010_2024_part_3.csv	RS Box Scores Pt 3
regular_season_totals_2010_2024.csv	RS Totals

1.2 Data Cleaning and Merging

This process was completed with R.

The `AVAILABLE_FLAG` (boolean value) columns from `PO Totals` had 336 missing values and 30 zeros and `RS Totals` 4454 missing values and 128 zeros; the observations with zeros were removed using `filter()`, because the `README` says the `AVAILABLE_FLAG` columns indicates if the data for a row is available. Also, since missing values in this column imply that the row does not contain complete information, we also removed the NA observations under `AVAILABLE_FLAG`. Additionally, `PO Box Scores` had several duplicate rows, so we used `distinct()` to keep the rows with unique values.

For `RS Totals`, we selected the columns relevant to opponent stats— `"FGM"`, `"FGA"`, `"FG_PCT"`, `"FG3M"`, `"FG3A"`, `"FG3_PCT"`, `"FTM"`, `"FTA"`, `"FT_PCT"`, `"OREB"`, `"DREB"`, `"REB"`, `"AST"`, `"TOV"`, `"STL"`, `"BLK"`, `"BLKA"`, `"PF"`, `"PFD"`, `"PTS"`, `"PLUS_MINUS"`, and `"PACE"`. We made a function called `add_opponent_stats()` which extracted team statistics and appended an “o” to the beginning of the column name to represent the opposing team’s value. With `left_join`, we merged this back into the original `RS Totals` dataset based on `GAME_ID` and `GAME_DATE`. To ensure opposing data stats are

¹ <https://github.com/NocturneBear/NBA-Data-2010-2024>

correct, we filtered for the years 2015-16, grouped by `TEAM_NAME` variable and calculated average points, rebounds, assists, and pace for the Teams and Opponents, and compared them to official records.

The `RS Box Scores Pt 1`, `RS Box Scores Pt 2`, and `RS Box Scores Pt 3` contain the same columns so we vertically appended these datasets with `bind_rows()`. After mutating columns `gameId` (from `RS Box Scores` and `PO Boxscores`) and `GAMEID` (from `RS Totals` and `PO Totals`) into numeric format with `as.numeric()`, we used `inner_join()` to merge `RS Box Scores` and `RS Totals` by their respective Game ID columns and respective Team ID columns. We also used `inner_join()` to merge `PO Box Scores` and `PO Totals` by their respective Game ID columns and respective Team ID columns. We ended by horizontally appending the two dataframes that result from these two inner joins processes using `bind_rows` since they share the same column names. We used `select(-)` to remove some columns which were only providing repeated information: `teamSlug`, `comment`, `jerseyNum`, `SEASON_YEAR`, `TEAM_ABBREVIATION`, `TEAM_NAME`, and `MATCHUP`.

We then introduced a column `opponentTricode`, which extracts the last three characters from column `matchup`. The dataset's columns were reorganized so the leftmost columns are `teamTricode`, `opponentTricode`, and then all other columns. Then, using `arrange()`, we arranged the values by the columns `teamId`, `game_date`, then `opponentTricode`. This produced a dataset with roughly 394,000 observations.

1.3 Additional Data Source

Although we cleaned both the player-level and game-level datasets, our primary focus was on game-by-game (team total) data based on the models we intended to create. As a result, we deliberately avoided incorporating external data sources that emphasized individual player metrics, such as injury data or advanced player statistics. One of the primary reasons for this decision was the inherent difficulty in reliably predicting player health during our forecast period (March 7th–21st, 2025). Since uncertainty in player availability and playing time would undermine the accuracy of player-specific models, a team-based approach was more appropriate. While team total data inherently contains fewer observations than player-level data, we determined that the tradeoff in training sample size was insufficient to compensate for the uncertainty in projecting individual player contributions.

With this rationale established, we sought to incorporate external data that could provide additional qualitative and quantitative context at the team level. The variables we introduced included the game's start time (`GAME_START_TIME`), attendance (`ATTENDANCE`), the ratio of attendance to arena capacity (`ATTENDANCE_RATIO`), whether either team was playing on consecutive nights (divided into two variables `HOME_BACK_TO_BACK` and `AWAY_BACK_TO_BACK`), and the distance the away team traveled from its home city (`AWAY_DISTANCE_FROM_HOME`). These variables were sourced from various publicly available datasets on Kaggle, including wyattowalsh's NBA-attendance-prediction

dataset on Github² and Logan Donaldson's stadiums dataset on Kaggle³. The following table helps explain our logical reasoning behind incorporating these specific pieces of outside data, and what part of the project we hoped they'd contribute to:

Outside Variable Name	Reasoning
GAME_START_TIME	A later starting time provides more rest/preparation for both teams potentially could lead to a higher skill game. An early start time also potentially hurts the away team who might have less time to prepare.
ATTENDANCE	Higher attendance represents a bigger crowd in favor of the home team, potentially increasing the "home court advantage" effect.
ATTENDANCE_RATIO	A smaller, fuller arena can feel louder than a larger, emptier one. Potentially compounding the "home court advantage" effect even more.
BACK_TO_BACK	Teams are required to play back-to-back games multiple times a season. Due to increased fatigue star players often don't play and the pace of the game can be slower.
AWAY_DISTANCE_HOME	Greater travel distance for the away team means more travel time, less preparation, potential jet lag, and increased fatigue, potentially benefiting the home team, especially in effort-driven stats like offensive rebounds.

For variables such as GAME_START_TIME, ATTENDANCE, and BACK_TO_BACK, which required no transformation, we directly integrated them into our dataset by adding new columns to both the regular season and playoff datasets. The datasets were linked using a unique identifier, AWAY_DATE, constructed by concatenating the away team's three-letter abbreviation with the game date (e.g., if the Chicago Bulls played on April 17, 2004, AWAY_DATE would be CHI_2004-04-17). Since no NBA team played in multiple cities on the same day—a fact we verified for our dataset's time span—we were able to successfully join these variables to the unique team-by-team data rows in the RS TOTALS and PO TOTALS datasets. The same process was applied to AWAY_BACK_TO_BACK and HOME_BACK_TO_BACK, but since both home and away versions were needed, we joined them using AWAY_DATE and HOME_DATE, following the same assumptions and logic.

To calculate ATTENDANCE_RATIO, we first matched each NBA team's home stadium capacity using the HOME_TEAM_ABBREVIATION. We then derived ATTENDANCE_RATIO by dividing ATTENDANCE by CAPACITY. For AWAY_DISTANCE_FROM_HOME, we introduced the variables AWAY_LAT, AWAY_LONG, HOME_LAT, and HOME_LONG, which contained the latitude and longitude coordinates for each stadium. These coordinates were merged into the dataset using AWAY_TEAM_ABBREVIATION and HOME_TEAM_ABBREVIATION. We then applied the distHaversine() function to compute the distance (in meters) between the two arenas, storing the result in the AWAY_DISTANCE_FROM_HOME column.

² <https://github.com/wyattowalsh/NBA-attendance-prediction/tree/master/data>

³ <https://www.kaggle.com/datasets/logandonaldson/sports-stadium-locations>

Once these variables were successfully integrated, we converted BACK_TO_BACK into a factor variable and included all five external variables in our model training and testing process. However, during model development, none of these variables were selected in models that performed feature selection or combination. Additionally, in models that included all available variables, none of the five external variables achieved statistical significance at any meaningful level. While further research is needed, the findings from this portion of the study suggest that these broader contextual variables contribute less predictive value to variables such as Spread, Total, and OREB Total than compared to traditional in-game basketball statistics.

1.4 Spread Variables Engineered

Spread is the measure of the points earned between Home and Away teams. A positive spread means the Home team earned more points, and a negative spread means the Away team earned more points. Spread uses the equation:

$$\text{Spread} = \text{Home Points} - \text{Away Points}$$

This variable was introduced in the dataset with `Spread = total_points_home - total_points_away`. In addition to Spread we added four more variables to the dataset:

Name	Column	Formulation
Rebound Difference	<code>rebound_diff</code>	<code>total_rebounds_home - total_rebounds_away</code>
Assist Difference	<code>assist_diff</code>	<code>total_assists_home - total_assists_away</code>
Turnover Difference	<code>turnover_diff</code>	<code>total_turnovers_home - total_turnovers_away</code>
Steal Difference	<code>steal_diff</code>	<code>total_steals_home - total_steals_away</code>

Rebound difference (`rebound_diff`) is the disparity in total rebounds between the home and away teams, reflecting each team's ability to secure possessions and generate second-chance scoring opportunities. Assist difference (`assist_diff`) is the gap in assists, highlighting differences in ball movement and offensive efficiency, as teams with higher assist counts tend to create more high-percentage scoring chances. Turnover difference (`turnover_diff`) is the imbalance in turnovers committed by each team, where a negative turnover differential suggests the home team is losing more possessions and potentially allowing more points off turnovers. Steal difference (`steal_diff`) is the variation in steals, indicating which team is more effective at disrupting the opponent's offense and generating fast-break opportunities.

1.5 Total Variable Engineered

Total is the measure of the total points earned among the Home and Away teams. It represents the combined offensive output of both teams in a game and is calculated as follows:

$$Total = Home Points + Away Points$$

Since raw point totals alone do not fully capture the factors influencing a game's scoring outcome, we added five additional variables that reflect offensive efficiency, pace, and possession-based metrics:

Name	Column	Formulation
Estimated Possessions	EP	$FGA + (0.44 * FTA) - OREB + TOV$
Offensive Rebounding Percentage	OREB_PCT	$OREB / (OREB + \odot DREB)$
Free Throw Ratio	FTR	$PTS / (FGA + (0.44 * FTA) - OREB + TOV)$
Effective Field Goal Percentage	EFG_PCT	$(FGM + 0.5 * FG3M) / FGA$
Pace	PACE	$(FGA + (0.44 * FTA) - OREB + TOV) / (MIN / 48)$

To better predict total points, several key metrics were created to capture a team's scoring opportunities, efficiency, and pace of play. Estimated Possessions (EP) approximates how often a team has possession of the ball, providing a baseline for scoring opportunities. Offensive Rebounding Percentage (OREB_PCT) reflects a team's ability to generate second-chance plays, increasing scoring potential. Free Throw Ratio (FTR) indicates how frequently a team attempts free throws relative to total shots, highlighting an efficient scoring method since free throws do not consume game time. Points Per Possession (PPP) measures scoring efficiency per possession, offering a more complete assessment than points per shot. Effective Field Goal Percentage (EFG_PCT) adjusts shooting efficiency by factoring in the extra value of three-pointers. Lastly, Pace estimates the number of possessions per game, with faster teams naturally creating more scoring chances. Together, these metrics help predict total points by capturing key elements of volume, efficiency, and play style.

1.6 OREB Variable Engineered

Offensive rebounds (OREB) measure the number of times a team regains possession of the ball after missing a field goal attempt. The ability to secure offensive rebounds is crucial in extending possessions and creating second-chance scoring opportunities. The equation for calculating OREB is:

$$OREB = Home OREB - Away OREB$$

We introduced several new variables to help improve the prediction of OREB, similar to the Spread methodology. These new variables capture key aspects of team and player performance related to rebounding ability:

Name	Column	Formulation
------	--------	-------------

Offensive Rebound Rate	oreb_rate	$\text{total_oreb_home} / (\text{total_oreb_home} + \text{total_dreb_away})$
Defensive Rebound Rate	dreb_rate	$\text{total_dreb_home} / (\text{total_dreb_home} + \text{total_oreb_away})$
Minutes Played Differential	min_diff	$\text{total_minutes_home} - \text{total_minutes_away}$

These variables are essential for modeling OREB because they account for team tendencies, rebounding efficiency, and relative performance against opponents. The rebound differential helps identify teams that dominate the glass, while offensive and defensive rebound rates give a more precise measure of rebounding efficiency relative to the opponent's opportunities. Minutes played differential helps capture variations in lineup rotations that could impact rebounding performance.

1.7 Rolling Averages for Predictive Modeling

When developing our predictive models, we initially trained them using game-by-game team total data. However, we quickly encountered a major limitation: while using in-game statistics to predict another statistic within the same game is valid for correlation analysis, it is not feasible for actual prediction. If a model relies on data from a game to predict an outcome from that same game, it cannot be used to generate forecasts before the game occurs, undermining the entire purpose of this prediction-based research project.

To address this, we implemented cumulative rolling averages for every numeric statistic in our dataset. This approach assumes that a team's past performance provides the best available estimate of its future performance. By using cumulative averages instead of raw game-by-game values, we ensured that our models could generate realistic, data-informed predictions for future games.

1.7.1 Overview of Implementation of Rolling Averages

We began by computed rolling averages for each numeric variable in our dataset, resetting the cumulative calculation at the start of each NBA season. This reset was necessary due to significant roster changes during the offseason (such as drafts, trades, and free-agent signings), which make it unreliable to assume a team's statistical profile remains the same from one season to the next.

A key challenge arose when dealing with early-season games, where the cumulative average is based on only a few observations. This posed a risk of instability in the rolling averages, as small sample sizes could skew the data and affect model training. To mitigate this, we implemented a weighted rolling average for the first 20 games of each season, blending data from the previous season with early-season statistics. This weighted average was computed using the formula:

$$(21 - n) \times 0.05 \times \text{Previous Season Average} + 0.05 \times (n - 1) \times \text{Current Season Cumulative Average}$$

where n represents the current game number within the season. This ensured a smooth transition from prior season data to current-season performance, with the first game relying entirely on the previous

season, the second game being 95% previous season data and 5% current season data, the third game being 90% previous season and 10% current, and so on. By game 21, the rolling average relied entirely on the current season's data.

For the first 20 games in our dataset (beginning of 2012-13 NBA season), where no prior season data was available, we used a cumulative average of the season itself. To validate our rolling average calculations, we compared the final season-long averages for all statistics against the official recorded statistics from Basketball Reference. After confirming that our calculated averages matched official statistics within an acceptable rounding error, we finalized our rolling average implementation.

1.7.2 Rolling Average Calculation in Code

To generate the rolling averages, we processed our dataset using the following steps:

1. Compute previous-season averages for each team and season.
2. Assign previous-season data to the corresponding team's current season games.
3. Apply rolling logic based on the three-phase weighted average formula.
4. Ensure cumulative averages reset at the start of each season to reflect roster changes.
5. Round all rolling averages to three decimal places for consistency.

The implementation was handled in R, where we first grouped our dataset by team and season, arranged games in chronological order, and computed cumulative means for each numeric column. The rolling logic was applied using `case_when()`, ensuring the correct weighting for early-season games. The final processed dataset included rolling averages for all relevant statistics, which were then used exclusively in model training to ensure reliable and informed predictions during our forecast period from March 7th to March 21st, 2025.

2. Methodology for *Spread*

We considered four models to predict spread. The models were built upon a Train dataset— setting the seed to 123, we split the dataset into training (80%) and testing (20%), where we used the R function `createDataPartition()` from the `caret` package to ensure a balanced split.

2.1 Multiple Linear Regression Model

To predict Spread, our multiple linear regression model used the rebound difference, assist difference, turnover difference, and steal difference. The model yields a Mean Absolute Error (MAE) of 3.577517, meaning that on average predicted Spread differs from actual Spread by 3.58 points. The model yields an R squared of 0.6002, meaning 60.02% of the variance in Spread can be explained by the selected variables. The model is still relatively strong but has room for improvement. Adding additional predictors could improve the model, but it also increases the risk of overfitting. Additionally, while some interaction variables did not change the MAE at all (i.e. rebound difference interacted with assist difference plus the other two variables), others just had a mild increase. For instance, a model with an interaction between turnover difference and steal difference (`rebound_diff + assist_diff + turnover_diff * steal_diff`) resulted in an MAE of 11.28, which isn't a significant enhancement of prediction accuracy. Hence, the final multiple linear regression model for predicting Spread includes just the differences in rebounds, assists, turnovers, and steals as predictor variables.

2.2 Ridge Model

Ridge regression is a type of linear regression that helps prevent overfitting by adding a penalty term to the model's coefficients. This penalty term is controlled by lambda, which shrinks coefficient values toward zero, reducing the impact of less important predictors; the model maintains all predictors in the model but just shrinks their influence, making it useful when predictors exhibit multicollinearity.

For this model, we used the same four predictor variables: rebound difference, assist difference, turnover difference, and steal difference. Figure 1 shows the coefficient shrinkage in our model as lambda increases, with each colored line representing a variable. On the left side of the plot, where the lambda is very small, the coefficients resemble their ordinary least squares regression values, and therefore we see varying influences of each predictor. As lambda increases, all coefficients shrink toward zero, though at different rates. The model was trained using cross-validation with the `cv.glmnet` function, which automatically selects an optimal lambda value that minimizes prediction error. Our ridge regression model yielded a MAE of 13.93756, meaning that on average, the predicted Spread differed from the actual Spread by 13.94 points.

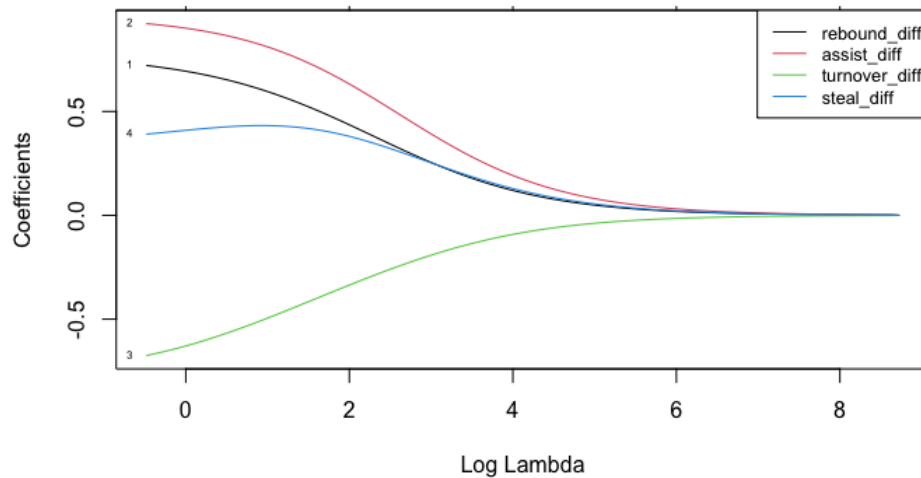


Figure 1: Coefficient Shrinkage as Lambda Increases

2.3 Random Forest Model

The random forest model uses a combination of decision trees, where each tree makes a set of predictions on a slightly different subset of the data (subset through bootstrapping). The final prediction set is based on the combination of these individual predictions, or “bootstrap aggregating”. One decision tree for our NBA dataset may have a root node at turnover difference, where a turnover difference less than or equal to -2.5 pivots to a check at Rebound Difference, and Assist Difference otherwise (see Figure 2). From there, a rebound difference less than or equal to 5 yields a spread of -4 (and 2 otherwise), while an assist difference less than or equal to 3.5 yields a spread of 6 (and 10 otherwise).

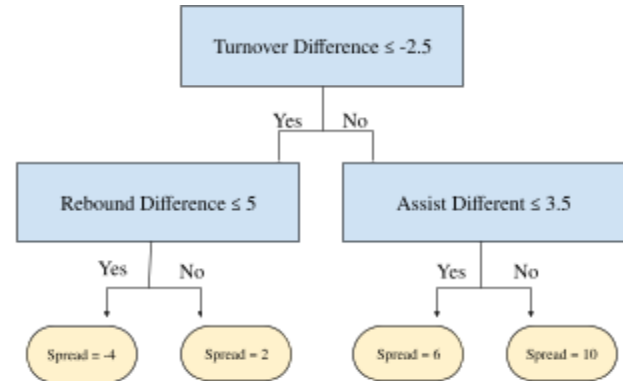


Figure 2: One possible decision tree within our Random Forest collection

The variables in our random forest model were rebound difference, assist difference, turnover difference, and steal difference, and we generated 500 trees. With `mtry` set to 2, meaning each tree will consider only two of the four variables at each split, we ensured randomness between the trees. This model yields an MAE of 10.89731038, meaning that on average predicted Spread differs from actual Spread by 10.9 points.

2.4 The Best Model: Extreme Gradient Boosting (xgBoost)

The extreme gradient boosting model also uses a combination of decision trees, with the exception that they descend by a gradient. Extreme gradient boosting models (i.e. xgBoost) are known for their speed, efficiency, and ability to scale well with large datasets. Since decision trees are prone to overfitting, xgBoost modeling is used to create more robust models. xgBoost modeling combines individual weak

trees—models that perform better than by random chance—to form a stronger model, then each new model is trained to correct errors made by the previous models. After many cycles of this, these weak models are converted to strong models. These models descend by a gradient, which is used to minimize the loss when training new models, thereby, improving the model’s performance.

When we began our process of creating the xgBoost model, we experimented with different techniques to create our model. We first explored a model by splitting data into training/testing sets. We manually selected specific features to include. We removed all variables with “_rank” at the end, and put the remaining variables in the appropriate DMatrix used for xgBoost models. After running 300 rounds and yielding an MAE of 5.98, we realized this was not the best method as the model was trained and tested using data from the same dataset. Therefore we decided to approach the model in a different way and use cross-validation and incorporate all numeric variables. We used the function `test_xgboost_model` that was developed when we created our model for TOTAL, which accepted the data, target column, number of rounds, number of cross-validation folds, early stopping rounds, and parameters. We selected our target column as “Spread” and used the same parameters from our first xgBoost model.

2.4.1 Parameter Experimentation

Objective:

The objective parameter specifies the learning objective and task used when training the model. With our target being spread, we decided to use the default objective “reg: squarederror”. We initially used “reg: absoluteerror”, thinking that was what needed to be used to evaluate mean absolute error. However, upon further investigation, we realized that “reg: absoluteerror” was not what we expected and not optimal because it refreshed the leaf value after decision tree construction. Therefore, we used the default throughout the rest of our model training.

Eval_metric:

This parameter is used to define the evaluation metric used to evaluate model performance. We set `eval_metric` as “mae” since we are evaluating the mean absolute error of the model. We used this as the parameter value throughout the investigation of finding the best model.

Max_depth:

This parameter refers to the maximum depth of each decision tree. With a higher `max_depth` the complexity of the model increases, but its tendency to overfit also increases. We set this parameter to the default, 6, to make sure our model was complex, but did not overfit.

Eta:

`Eta`, also known as the learning rate, relates to the step size shrinking used to prevent overfitting and from adjusting too aggressively. This parameter shrinks the weight of each feature in order to make the process of boosting more accurate. We initially set the value to 0.1, but reduced it to 0.05 after

experimenting. Doing this slows the training, while simultaneously increasing the accuracy and generalization.

Subsample:

The parameter `subsample` determines the ratio of training data randomly sampled for each round during the model's process. It is also used to prevent overfitting. We initially set this parameter to 0.5, but upon building a new model, we increased it to 0.8 to allow for more randomness and fitting, but making sure it did not overfit.

Colsample_bytree:

`Colsample_bytree` determines the subsample ratio of columns during each round when constructing each tree. It occurs once for every tree that is constructed. This parameter also prevents overfitting by reducing the feature dependence and increasing model diversity. We set `colsample_bytree` equal to 0.8 to allow for the best performance while simultaneously preventing overfitting.

2.4.2 Training the Model

After determining the best parameter values, we trained our model. The first thing we did before we trained our model, was select all the rolling average variables, those starting with "roll_" (all numeric variables), as our features and convert them into a matrix (`numeric_features`), since xgBoost modeling uses matrices. Then we standardized the features to ensure the scales are compatible. The next steps in our modeling process consisted of creating an optimized data structure for the xgBoost to use, taking the standardized features and extracting our target column

(`point_spread`). We then performed a 10-fold cross validation on our training data. We used the parameters previously stated, and ran 300 rounds with early stopping. Afterwards, we extracted the test mean absolute error (MAE) from the cross-validation results, the best iteration and its corresponding MAE for that round. Our final model achieved an MAE of 10.62181. After performing cross-validation, the final xgBoost model was trained on the entire dataset using the best iteration from the cross-validation as the number of rounds. We created a feature importance plot containing all the variables to determine

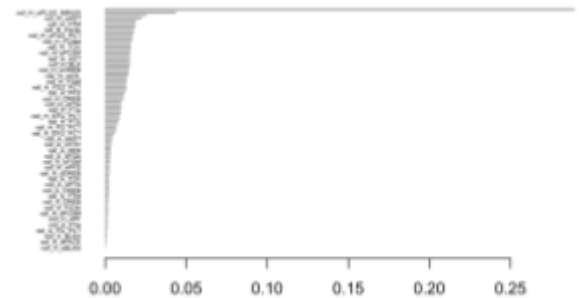


Figure 3: Feature Importance plot containing all variables

which features were most useful based on the trained model (Figure 3). The variables that were most significant were `roll_H_PLUS_MINUS`, `roll_H_oPLUS_MINUS`, `roll_H_BLKA`, and `roll_H_oAST`. Each of these variables has an importance factor greater than 0.02, insinuating their significance to the model. To clearly see these variables in plot form, we created an additional feature importance plot showing the significant variables plus two more (Figure 4). This shows that the top four variables are significant with

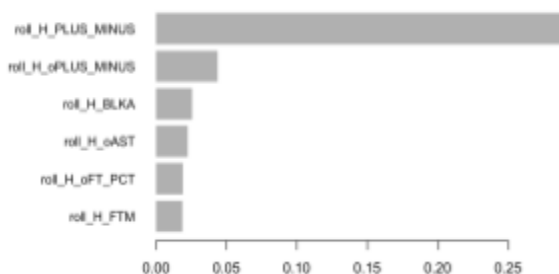


Figure 4: Reduced feature importance plot

importance values greater than 0.02. From these findings, we determined that all the rolling average variables in the dataset provided the best results for our model, along with cross-validation being the best method of training to create a model in predicting spread.

2.5 Generating Predictions

To generate predictions for Spread for NBA games from March 7th–21st, we utilized a dataset found on Kaggle⁴ titled “Complete Historical NBA Data and Player Box Scores,” which provided up-to-date data through March 6, 2025. Although we initially computed rolling averages for each team’s statistical measures using a three-phase logic (see Section 1.7), by March 6, 2025 every team had played at least 20 games. This allowed us to simplify our process by computing the cumulative average across all games played to date. These cumulative averages serve as robust estimates of team performance, reflecting the overall performance level of teams and players up to the prediction date.

Once the cumulative averages were calculated, we merged these team-level metrics into our prediction dataset by aligning the home and away teams with their corresponding averages. This integration resulted in a comprehensive feature set, with each game record containing both home team and away team statistics formatted to match the features used during model training.

For prediction, we leveraged our pre-trained xgBoost model along with the saved preprocessing object and feature names file. The new data were standardized using the original centering and scaling parameters before being fed into the model, ensuring consistency between the training and prediction phases. Finally, the xgBoost model generated predictions for the spread of the game scores, thereby completing our forecast pipeline.

4

<https://www.kaggle.com/datasets/eoinamoore/historical-nba-data-and-player-box-scores?resource=download&select=TeamStatistics.csv>

3. Methodology for *Total*

3.1.1 Multilinear Regression Model: Forward Selection

The forward selection method is a stepwise approach to building a multilinear regression model by iteratively adding the most significant predictor variables. The process begins with an intercept-only model and sequentially incorporates the variable that provides the greatest improvement based on the Akaike Information Criterion (AIC). AIC balances model fit and complexity, favoring models that achieve better predictive performance with fewer parameters to mitigate overfitting. This approach was particularly useful in our case, as we had over 50 potential predictor variables to consider when modeling TOTAL.

To implement forward selection, we began by initializing an intercept-only model using `step (lm ())` with the direction set to “forward.” This allowed us to progressively add predictor variables based on their statistical significance and contribution to explaining TOTAL. To ensure a robust evaluation of model performance and to mitigate overfitting, we employed 10-fold cross-validation. In this process, the dataset was divided into 10 folds; for each iteration, the forward selection model was trained on 9 folds and validated on the remaining fold. The final model was chosen once no additional predictors improved the model fit (as determined by AIC). Given that this model architecture features inherent feature selection with relatively few parameters to tune, our main focus was on evaluating its predictive performance compared to other baseline models.

We assessed the predictive accuracy using the Mean Absolute Error (MAE) computed from the 10-fold cross-validation, which resulted in an MAE value of 15.13492. The final model included the following variables along with their respective slopes, standard errors, t-values, and p-values:

Variable	Slope	Standard Error	t Value	p Value
Intercept	28.36897	7.47672	3.794	0.000148
roll_TOTAL	0.79746	0.02774	28.752	< 2e-16
roll_oFG3A	0.8617	0.09471	9.098	< 2e-16
roll_oFG3M	-1.95093	0.25654	-7.605	2.94E-14
roll_FG3_PCT	1.86152	16.78695	0.111	0.911704
roll_FG3A	0.70248	0.22641	3.103	0.001919
roll_PACE	1.71609	0.22757	7.541	4.81E-14
roll_EP	-1.55739	0.23152	-6.727	1.77E-11
roll_FTR	-10.62172	3.86274	-2.75	0.005967
roll_FG3M	-1.68424	0.63527	-2.651	0.008025

roll_STL	-0.3154	0.13065	-2.414	0.015782
roll_oBLK	-0.28547	0.15962	-1.788	0.073719

3.1.2 Multilinear Regression Model: Backward Selection

Backward selection follows a similar stepwise approach as forward selection but in reverse. Instead of starting with an intercept-only model, it begins with the full multilinear regression model containing all potential predictor variables and iteratively removes the least significant ones. While this method ensures that all relevant variables are initially considered, it comes with its own set of trade-offs.

One advantage of backward selection in this context is that it accounts for the full range of available statistics from the outset of the model's creation, reducing the risk of excluding influential variables too early in the process. For example, a variable like roll_STL (rolling average of steals) might not seem significant in predicting TOTAL in a simple model but could be important when combined with other team defensive metrics. However, a key drawback is that backward selection can struggle with collinearity, where multiple predictors provide overlapping information, potentially leading to the retention of redundant variables. To combat this, we compared the selected predictors to those from other models, ensuring that there wasn't too much inflation in the number of variables for the backwards-based model.

Implementation of backward selection started with a `stepAIC()` model that included all possible predictors from our dataset and systematically removed the least significant ones based on AIC, by setting the direction to “backward.” The process continued until eliminating additional variables no longer improved model performance. To ensure robust prediction across our dataset, we again utilized 10-fold cross validation, using the same process as the forward model previously. Since backward selection inherently determines the final feature set, our primary focus was again on evaluating its predictive accuracy and comparing it to forward selection and other baseline models.

We assessed the predictive accuracy using the Mean Absolute Error (MAE) computed from the 10-fold cross-validation, obtaining a value of 15.12943, slightly better than forward selection. The final model included the following variables with their respective slopes, standard error, t value, and p-value:

Variable	Slope	Standard Error	t Value	p Value
Intercept	49.0835	14.1676	3.464	0.000532
roll_FGM	3.307	0.5965	5.545	2.97E-08
roll_FGA	2.2313	1.2839	1.738	0.082231
roll_FG3A	0.6842	0.1049	6.521	7.10E-11
roll_FTM	2.8556	0.7842	3.641	0.000272
roll_FT_PCT	-30.1056	17.0395	-1.767	7.73E-02
roll_OREB	-2.101	1.2831	-1.637	0.101549

roll_TOV	2.3633	1.2974	1.822	0.068519
roll_STL	-0.3449	0.1466	-2.352	1.87E-02
roll_BLK	-204.917	136.8045	-1.498	1.34E-01
roll_PFD	302.7643	161.7467	1.872	0.061239
roll_PLUS_MINUS	301.9449	163.6383	1.845	0.065019
roll_PACE	1.7803	0.2305	7.724	1.16E-14
roll_oFGM	3.7271	0.6113	6.097	1.09E-09
roll_oFG3A	0.8437	0.1061	7.954	1.87E-15
roll_oFTM	1.8869	0.2929	6.441	1.20E-10
roll_oSTL	-0.3669	0.256	-1.433	0.151861
roll_oBLK	-0.3892	0.1872	-2.079	0.037639
roll_oBLKA	204.944	136.8114	1.498	0.134143
roll_oPF	-302.9603	161.7543	-1.873	0.061083
roll_oPTS	603.6442	327.2772	1.844	0.065129
roll_EP	-3.7842	1.3206	-2.866	0.004167
roll_TOTAL	-302.7912	163.6388	-1.85	0.064271

3.1.3 Multilinear Regression Model: Stepwise Selection

Stepwise selection combines elements of both forward and backward selection, allowing for a more dynamic feature selection process. This approach begins with an intercept-only model (like forward selection) but at each step, it reevaluates the significance of previously added variables, removing any that become statistically insignificant as new variables are introduced. By incorporating both inclusion and exclusion criteria, stepwise selection seeks to strike a balance between model complexity and predictive power.

To implement stepwise selection, we set up a `stepAIC()` model and set the direction to "both", allowing for forward inclusion and backward elimination at each iteration. Thus the process began with an intercept-only model and iteratively added or removed predictors based on AIC until no further improvements in model fit could be made. As with the previous methods, we use 10-fold cross validation in the training/testing of our model.

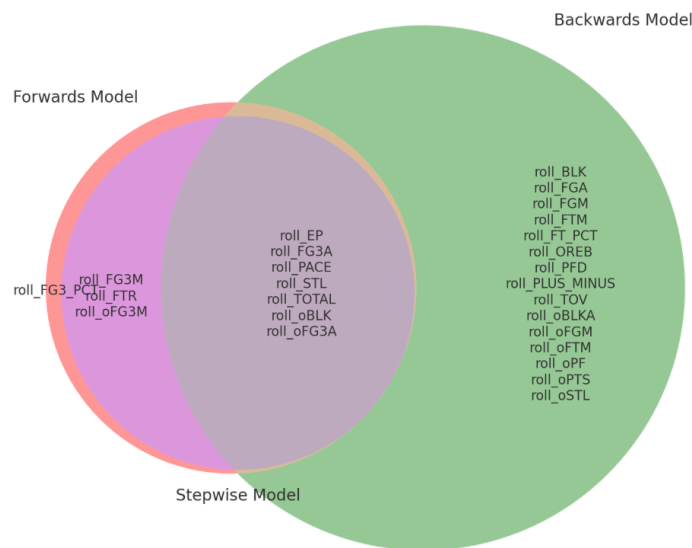
We assessed model performance using MAE computed from the 10-fold cross-validation, obtaining a value of 15.13485, almost identical to that of the MAE for the forward selection model. The final model included the following variables with their respective slopes, standard error, t value, and p-value:

Variable	Slope	Standard Error	t Value	p Value
Intercept	29.00441	4.80245	6.04	1.56E-09
roll_TOTAL	0.7973	0.0277	28.785	< 2e-16

roll_oFG3A	0.86153	0.0947	9.098	< 2e-16
roll_oFG3M	-1.95026	0.25647	-7.604	2.95E-14
roll_FG3A	0.67942	0.08954	7.588	3.35E-14
roll_PACE	1.71621	0.22757	7.542	4.78E-14
roll_EP	-1.55694	0.23148	-6.726	1.78E-11
roll_FTR	-10.64048	3.85897	-2.757	0.00583
roll_FG3M	-1.61905	0.24071	-6.726	1.78E-11
roll_STL	-0.31477	0.13053	-2.412	0.01589
roll_oBLK	-0.28653	0.15933	-1.798	0.07213

Overall, the three models largely used fairly similar predictors. All three share a core set of 7 variables: roll_TOTAL, roll_oFG3A, roll_FG3A, roll_PACE, roll_EP, roll_STL, and roll_oBLK. The Forwards Model includes a total of 11 variables by adding 4 extra predictors (roll_oFG3M, roll_FG3_PCT, roll_FTR, and roll_FG3M) to this common set. The

Stepwise Model uses 10 variables, matching the Forwards Model except for the omission of roll_FG3_PCT. In contrast, the Backwards Model employs 22 predictors, with 15 of these (roll_FGM, roll_FGA, roll_FTM, roll_FT_PCT, roll_OREB, roll_TOV, roll_BLK, roll_PFD, roll_PLUS_MINUS, roll_oFGM, roll_oFTM, roll_oSTL, roll_oBLKA, roll_oPF, and roll_oPTS) being unique to that model. Although the Backwards Model achieved the best MAE, its improvement of only 0.005 over the other models comes at the cost of more than twice as many predictors, raising the question of whether the marginal



gain is worth the increased complexity. Fortunately, subsequent models later surpassed the MAE of all three regression examples, saving us the pressure of making this decision when it came to generating our TOTAL predictions.

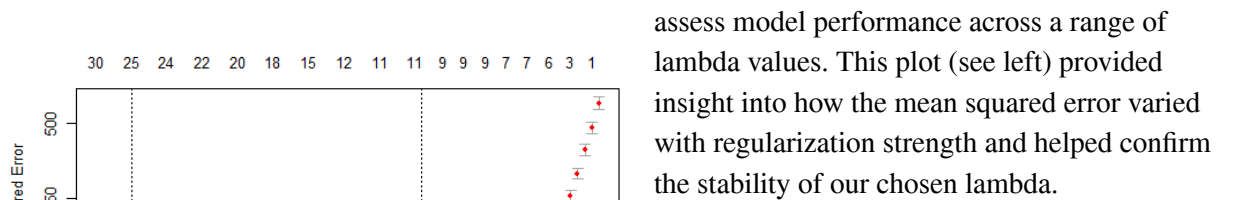
3.2 Elastic Net Regression Model

In our research of potential models, we frequently encountered discussions on Ridge and Lasso regression. Although both methods offer clear benefits, their approaches appeared similar to our earlier

multivariate selection-based models, until we discovered the Elastic Net model. Elastic Net combines L1 (Lasso) and L2 (Ridge) penalties to simultaneously perform regularization and variable selection, thereby merging the strengths of both techniques.

This dual penalty is particularly advantageous in the context of basketball statistics, where multicollinearity among rolling averages (e.g., roll_TOTAL, roll_PTS, roll_STL) can obscure the true influence of each predictor. In our analysis, we experimented with the alpha parameter, which balances the L1 and L2 penalties, by testing values from 0.1 to 0.9 in increments of 0.1. We found that variations in alpha had little impact on model performance, with the MAE fluctuating by no more than +/- 0.005. Consequently, we set alpha to 0.5 in our final model to achieve an even balance between the two penalties and employed 10-fold cross-validation, using mean squared error (MSE) as the performance metric, to identify the optimal regularization parameter (lambda).

The implementation involved preparing a predictor matrix from our rolling statistics data and constructing a response vector for TOTAL. We utilized the glmnet() package to fit the model, with cross-validation guiding us to the best lambda value. Additionally, we plotted the cross-validation error curve to visually



assess model performance across a range of lambda values. This plot (see left) provided insight into how the mean squared error varied with regularization strength and helped confirm the stability of our chosen lambda.

The Elastic Net model’s performance was evaluated using Mean Absolute Error (MAE), resulting in an MAE of 15.13417 across all folds. Not only does this approach help mitigate overfitting by shrinking coefficients, but it also

inherently performs feature selection, potentially highlighting the combinations of basketball performance metrics most critical to predicting TOTAL.

Variables Used	Coefficients
(Intercept)	-45.874
roll_FG_PCT	6.978
roll_FG3A	0.124
roll_FG3_PCT	-31.718
roll_OREB	0.175
roll_AST	0.059
roll_STL	-0.381
roll_BLK	-0.224
roll_PFD	-0.135
roll_PACE	0.965

roll_oFG_PCT	53.460
roll_oFG3A	0.312
roll_oFG3_PCT	-35.079
roll_oFTM	0.146
roll_oFT_PCT	10.950
roll_oOREB	0.254
roll_oDREB	-0.003
roll_oTOV	-0.245
roll_oSTL	-0.244
roll_oBLK	-0.136
roll_oBLKA	0.003
roll_oPF	-0.051
roll_oPTS	0.261
roll_oPACE	0.271
roll_PPP	54.197
roll_TOTAL	0.154

The final model incorporated 25 variables, roughly half of the available features. Notably, it excluded `roll_EP` (the rolling average of Estimated Possessions), despite this variable being one of the core seven predictors selected by all three stepwise multiple linear regression models.

Ultimately, the MAE of the Elastic Net model was comparable to that of the forward and stepwise selection models, and slightly higher than that of the backward selection model. While Elastic Net provided an alternative modeling approach, its performance suggests that we must further investigate more complex models capable of handling the non-linear relationships inherent in our dataset.

3.3 The Best Model: xgBoost Model

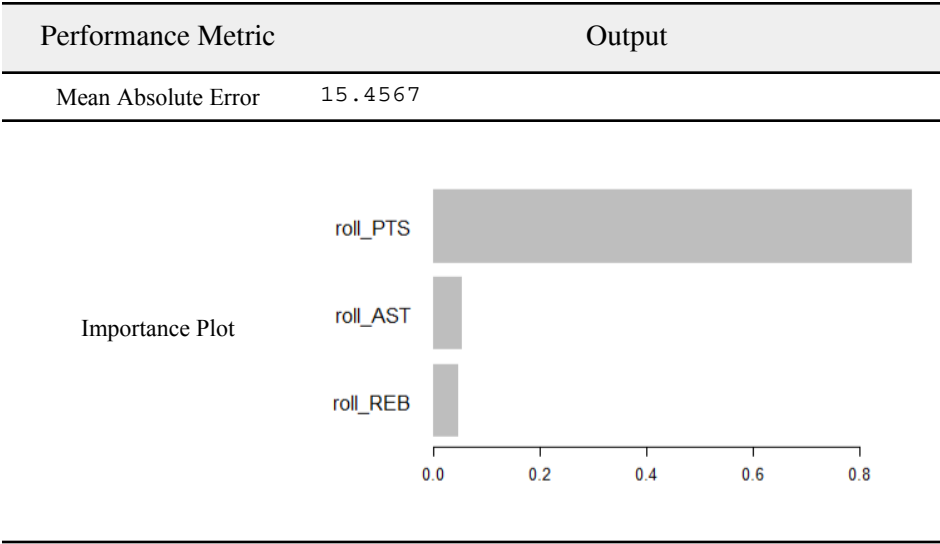
In our research of potential models, we recognized early on that xgBoost was among the most promising options for predicting TOTAL, a statistic influenced by a plethora of basketball performance factors. While others in our research group explored xgBoost for the prediction of different statistics, this section details our specific implementation, parameter tuning, and iterative experimentation aimed at minimizing Mean Absolute Error (MAE) for TOTAL.

To facilitate systematic experimentation, we developed a function (`test_xgboost_model`) which accepts the dataset, target column, number of rounds, cross-validation folds, early stopping rounds, and a list of parameters. This function not only allowed us to control the features processed, but also provided a standardized workflow for model testing. The process involved standardizing the selected features, converting them into an xgBoost-compatible DMatrix, and performing 10-fold cross-validation with early stopping. After cross-validation, the model was trained on the full dataset using the optimal number of rounds, and feature importance scores were generated to assess the contributions of each predictor.

3.3.1 Feature Selection and Filtering

Unlike our previous regression-based models that inherently removed or added less/more significant features, xgBoost does not perform inherent feature selection/elimination, but instead assigns importance scores and creates importance plots for all predictors. This allowed us to experiment with various subsets of features to establish robust baselines and controls based on what we observed in these scores. Our approach to filtering and refining these baselines is detailed below:

Initial Baseline: We started with a small set of fundamental statistics (roll_PTS, roll_AST, and roll_REB) that are conceptually linked to TOTAL, though none ranked among the most predictive features in our regression models. Despite their presumed limitations, this traditional statistical model performed surprisingly well, achieving an MAE of 15.4567, as shown in the following results:

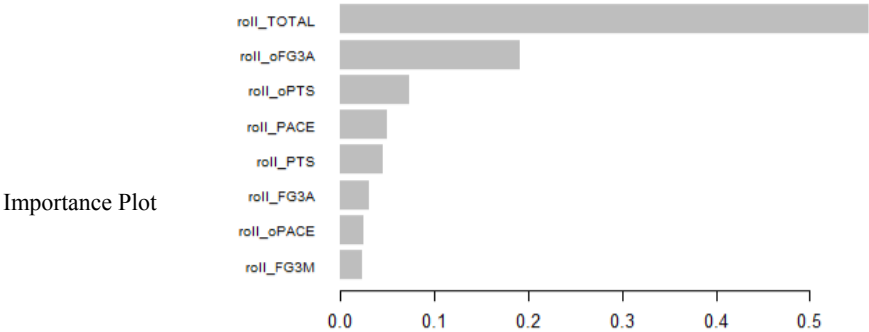


Lower-end Baseline: Given the surprisingly strong MAE of our initial baseline compared to the regression-based models, we aimed to establish a true lower bound. To do so, we tested a model using more obscure and independent features, such as roll_OREB_PCT, roll_oAST, and roll_oPFD, none of which were selected in any previous regression models. This configuration resulted in an MAE of 16.269, helping define the potential performance spectrum of our xgBoost models. However, this result also underscored the strength of xgBoost, as it was still able to construct a reasonably functional predictive model despite relying on seemingly unrelated independent variables.

Full Feature xgBoost Total Model Table Displaying Top 10 Variables by Importance Score				
Feature	Gain	Cover	Frequency	Importance
roll_TOTAL	0.392	0.108	0.039	0.392
roll_oFG3A	0.221	0.046	0.018	0.221
roll_oPTS	0.062	0.025	0.024	0.062
roll_PTS	0.036	0.021	0.017	0.036
roll_PACE	0.034	0.028	0.022	0.034
roll_oPACE	0.013	0.027	0.018	0.013
roll_FG3A	0.013	0.060	0.034	0.013
roll_FG3M	0.010	0.029	0.037	0.010
roll_PFD	0.009	0.029	0.027	0.009
roll_oOREB	0.009	0.023	0.027	0.009

Feature Subset Model: Building on the results of our full model, we next decided to next test a refined model using only features with an importance score greater than 0.01. This subset (comprising roll_TOTAL, roll_oFG3A, roll_oPTS, roll_PTS, roll_PACE, roll_oPACE, roll_FG3A, and roll_FG3M) resulted in an MAE of 15.1615. Notably, roll_TOTAL remained the most influential variable, with an importance score of 0.5633. The rankings of the remaining variables remained largely unchanged, aside from slight percentage increases due to the reduced feature set.

Performance Metric	Output
Mean Absolute Error	15.1615



Based on these experiments, we determined that the full variable set provided additional insights with minimal extra computation time. Therefore, in our subsequent analyses and reporting, we use the full variable set as our default, while still occasionally experimenting with reduced subsets for specific features.

3.3.2 Parameter Experimentation

Unlike our previous models, which focused primarily on variable selection, xgBoost requires careful tuning to optimize its handling of non-linear relationships and complex interactions. Thus, the xgBoost model architecture includes several tunable parameters that can significantly influence performance. To systematically evaluate each parameter, we followed a structured approach similar to our feature selection process. We established a control model using the same variables while modifying only one parameter at a time. We then tested its lower and upper bounds, comparing results against the default setting. If all three configurations produced similar MAE values across validation folds, we concluded that the parameter had minimal impact in the context of predicting TOTAL and reverted it to its default setting. Conversely, for parameters that showed meaningful improvements, we conducted further refinements to determine optimal ranges.

While some parameters had a significant impact under specific conditions (such as smaller datasets, fewer variables, or limited computational resources) our final model primarily relies on settings close to the defaults. Nonetheless, we document our full tuning process below, including plots, figures, and statistical analyses, to illustrate the effects of each parameter and provide insight into their role in model refinement:

Objective:

The objective parameter specifies the learning task and associated loss function. Since our target variable, TOTAL, remained unchanged, we kept this parameter at its default setting for regression-based xgBoost models: "reg:squarederror". We recognize that using squared error instead of mean absolute error (MAE) is a potential limitation in our experimentation. However, xgBoost in R does not natively support MAE as an objective function, and our attempts to implement a custom MAE function were unsuccessful. While this discrepancy may introduce some minor differences, we believe its overall impact on model performance is minimal.

Eval_metric:

This parameter defines the evaluation metric used during training to assess model performance. Since MAE (Mean Absolute Error) was the designated metric for this research project, we consistently used "mae" across all experiments.

Max_depth:

Max_depth controls the maximum depth of individual trees, affecting both model complexity and the risk of overfitting. We found this parameter to be one of the most influential in determining final model accuracy and, interestingly, one of the few that could be optimized to a specific middle value.

Rather than an exact optimization process, we incrementally increased `max_depth` one step at a time while keeping all other parameters and features constant. This approach consistently reduced the final testing MAE, until a certain point where further increases caused MAE to rise again. Through repeated testing, we identified `max_depth = 5` as the most effective setting, closely aligning with the `xgBoost` default of 6. This became our new standard for `max_depth` moving forward.

We believe this effect is largely due to how `max_depth` influences feature selection. Lower values limit the number of variables selected, allowing the model to more reliably focus on the most important features for predicting `TOTAL`. Conversely, increasing `max_depth` led to a more evenly distributed feature importance plot, giving additional weight to less influential variables. This shift appears to have reduced model accuracy, as the deeper trees struggled to prioritize the most relevant predictors. The results of this process are shown below:

Max Depth Value	3	4	5	6	8	10	15
MAE	15.1222	15.113	15.0981	15.1146	15.113	15.137	15.3134
Top Feature Importance Score	0.718	0.682	0.678	0.636	0.429	0.435	0.303

Eta:

Also known as the learning rate, `eta` controls the step size at each boosting iteration, balancing the speed of convergence with the risk of overshooting the optimal solution. Based on our experience, learning rate is particularly important when working with computationally intensive models or those prone to overfitting. However, since our dataset does not contain hundreds of variables or hundreds of thousands of training rows, computational time was not a major concern.

However we did observe that lower `eta` values led to more stable and lower MAE values, as longer training times and additional iterations allowed the model to refine itself more effectively. Conversely, higher `eta` values significantly reduced training time but resulted in higher MAE values, and introduced slight inconsistencies, with MAE fluctuating by approximately ± 0.01 between runs. However, we did find that setting `eta` around 0.05 produced results comparable to low `eta` models without requiring excessive computation time.

In terms of feature selection, the learning rate had minimal impact on which features were prioritized or their respective importance scores. Across different `eta` values, the top 8–10 most important features remained consistent, with `roll_TOTAL` always emerging as the most influential. In lower `eta` models, `roll_TOTAL` typically had an importance score of approximately 0.55, while in higher `eta` models, it was slightly elevated at around 0.6.

For general model testing, we used $\eta = 0.05$ to balance efficiency and performance. However, once a model demonstrated high-end performance, we re-ran it with a lower learning rate to achieve optimal MAE. Our final xgBoost model relies on this lower eta setting to reach its best performance. The results of our high and low eta tests are detailed below:

Performance Metric	0.01 ETA Value Results	0.05 ETA Value Results	0.25 ETA Value Results
Mean Absolute Error	15.0563	15.0693	15.1655
Test Mean Absolute Error at Round 5	203.6191	165.6848	50.9108
Test Mean Absolute Error at Round 10	193.64297	128.2105	17.9621
Test Mean Absolute Error at Round 50	129.5607	20.3077	Already Finished
Test Mean Absolute Error at Round 100	78.4116	15.0856	Already Finished
Test Mean Absolute Error at Round 500	15.0781	Already Finished	Already Finished
Final Training Round	617	130	31

Subsample and colsample_bytree serve somewhat similar functions, as observed in our experimentation. When one was set low and the other high, their effects largely canceled out, but when both were set to either low or high values, we noticed significant impacts on feature importance and model MAE. Due to this interaction, we primarily adjusted them in tandem throughout our experiments, and we discuss them together in the following analysis:

Subsample:

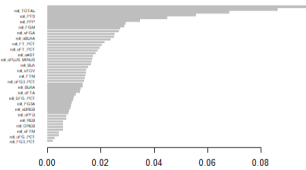
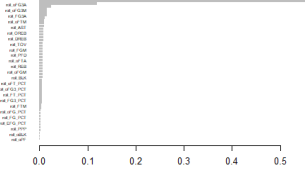
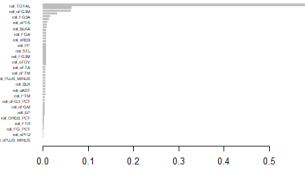
This parameter specifies the fraction of the training data to be randomly sampled for each boosting round, which can help reduce overfitting.

Colsample_bytree:

Colsample_bytree denotes the fraction of features (columns) to be randomly sampled for each tree, influencing the diversity of the trees.

Both of these parameters play a significant role in xgBoost's feature selection process. While the default value of 0.8 for both generally performed well in identifying the most important variables in our dataset, lowering them led to greater diversity in feature selection. This resulted in a more even distribution of feature importance scores but ultimately produced weaker models, as indicated by higher MAE values.

This finding reinforced that, in the context of our dataset, certain variables are crucial for predicting TOTAL, and limiting the model’s ability to focus on them led to poorer performance. While higher values for these parameters can sometimes increase the risk of overfitting, our cross-validation results showed minimal discrepancies between training and test MAE. This consistency gave us confidence that increasing these parameters would enhance model accuracy without significantly raising the risk of overfitting. You can see the results of this experimental process below, where models with lower values for these parameters resulted in a far more evenly distributed importance plot, with a significantly larger number of features holding similar importance scores at the top.

Performance Metric	0.1 subsample and colsample_bytree Results	0.8 subsample and colsample_bytree Results	0.9 subsample and colsample_bytree Results
Mean Absolute Error	15.2977	15.1018	15.0902
Importance Plot			
Total Feature by Importance Score (Importance Score)	roll_Total (0.0969)	roll_Total (0.5715)	roll_Total (0.6772)
Number of Features with Importane Score >10% of Top Feature Importance Score	32	1	0

0.1 subsample and colsample_bytree Top 5 Features by Importance		0.8 subsample and colsample_bytree Top 5 Features by Importance		0.9 subsample and colsample_bytree Top 5 Features by Importance	
Feature	Importance Score	Feature	Importance Score	Feature	Importance Score
roll_TOTAL	0.0969	roll_TOTAL	0.5715	roll_TOTAL	0.6772
roll_oPACE	0.0718	roll_oFG3A	0.1054	roll_oFG3A	0.0505
roll_oFG3M	0.0699	roll_PACE	0.0279	roll_PACE	0.0327
roll_oPTS	0.0620	roll_PTS	0.0191	roll_FG3A	0.0169
roll_FGA	0.0578	roll_FG3A	0.0162	roll_oFG3M	0.0095

3.3.3 Best Iteration and Final Model

After extensive experimentation with both feature sets and parameter tuning, our iterative process led us to identify the best iteration of the xgBoost model using 10-fold cross-validation. This final model, selected through early stopping when MAE stabilized, represents our most optimized approach for predicting TOTAL.

Interestingly, the max_depth parameter in the final model deviated from our earlier findings, where a depth of 5 initially appeared optimal. Through multiple rounds of testing and retesting, we observed that increasing max_depth to 8 consistently reduced MAE by approximately 0.05 compared to previous runs using max_depth = 5. Further experimentation is needed to determine the underlying cause of this discrepancy and whether deeper interactions between parameters influence this outcome.

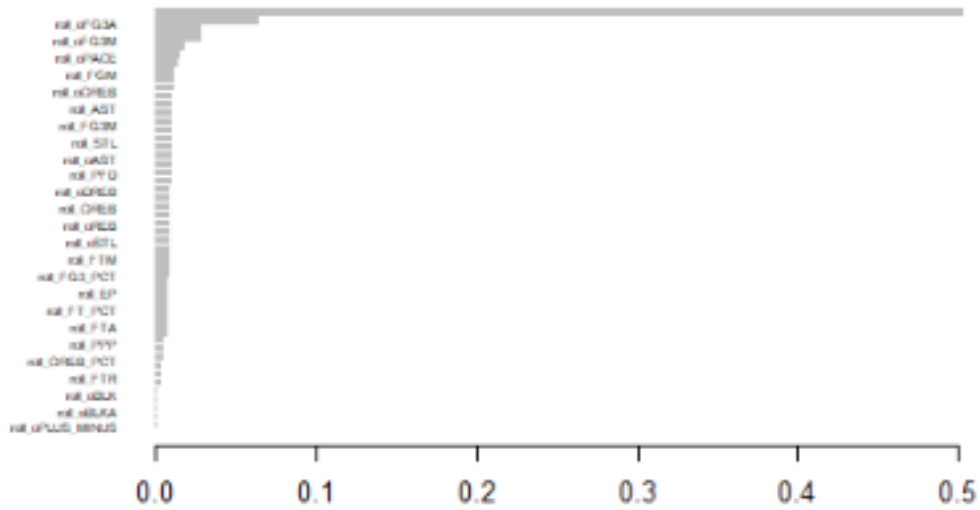
The following set of parameters consistently produced the most accurate xgBoost model for predicting TOTAL based on MAE. Our final iteration achieved an MAE of 15.0121, with the top twenty features and their importance scores displayed in the table below.

Parameter	Value
objective	"reg:squarederror"
eval_metric	"mae"
max_depth	8
eta	0.01
subsample	0.9
colsample_bytree	0.9

Performance Metric	Output
--------------------	--------

Mean Absolute Error15.0121

Importance Plot



Top Feature by Importance
(Importance Score)roll_TOTAL
(0.5041)

Final Stopping Round616

Final xgBoost Model Table Displaying Top 20 Variables by Importance Score				
Feature	Gain	Cover	Frequency	Importance
roll_TOTAL	0.5041	0.0932	0.0245	0.5041
roll_oFG3A	0.0645	0.0509	0.0215	0.0645
roll_PACE	0.0287	0.0416	0.0198	0.0287
roll_oFG3M	0.0280	0.0324	0.0208	0.0280
roll_FG3A	0.0176	0.0550	0.0306	0.0176
roll_oPACE	0.0154	0.0251	0.0154	0.0154

roll_PTS	0.0137	0.0197	0.0166	0.0137
roll_FGM	0.0110	0.0150	0.0913	0.0110
roll_FGA	0.0109	0.0251	0.0481	0.0109
roll_OREB	0.0108	0.0234	0.0224	0.0108
roll_BLK	0.0107	0.0182	0.0254	0.0107
roll_AST	0.0106	0.0337	0.0244	0.0106
roll_BLK_A	0.0104	0.0248	0.0234	0.0104
roll_FG3M	0.0103	0.0198	0.0346	0.0103
roll_OPT	0.0103	0.0187	0.0165	0.0103
roll_STL	0.0102	0.0182	0.0248	0.0102
roll_TOV	0.0100	0.0151	0.0242	0.0100
roll_OAST	0.0099	0.0193	0.0217	0.0099
roll_DREB	0.0099	0.0258	0.0243	0.0099
roll_PFD	0.0098	0.0259	0.0206	0.0098

3.4 Generating Predictions

To generate predictions for total score for NBA games from March 7th–21st, we utilized a dataset found on Kaggle titled “Complete Historical NBA Data and Player Box Scores,” which provided up-to-date data through March 6, 2025. Although we initially computed rolling averages for each team’s statistical measures using a three-phase logic (see Section 1.7), by March 6, 2025 every team had played at least 20 games. This allowed us to simplify our process by computing the cumulative average across all games played to date. These cumulative averages serve as robust estimates of team performance, reflecting the overall performance level of teams and players up to the prediction date.

Once the cumulative averages were calculated, we merged these team-level metrics into our prediction dataset by aligning the home and away teams with their corresponding averages. This integration resulted in a comprehensive feature set, with each game record containing both home and away team statistics formatted to match the features used during model training.

For prediction, we leveraged our pre-trained XGBoost model along with the saved preprocessing object and feature names file. The new data were standardized using the original centering and scaling parameters before being fed into the model, ensuring consistency between the training and prediction phases. Finally, the XGBoost model generated predictions for the total game scores, thereby completing our forecast pipeline.

4. Methodology for *OREB*

4.1 Poisson Regression Model

The Poisson regression model is commonly used for predicting count-based data such as *OREB*. To build this model, we used the `glm()` function in R with `poisson` as the family parameter. The predictors included rebound differential, offensive rebound rate, defensive rebound rate, and minutes played differential. Running `summary()` on this model showed that rebound differential and offensive rebound rate had the strongest impact on *OREB*. However, the model exhibited signs of overdispersion, meaning the variance was significantly larger than the mean. The Mean Absolute Error (MAE) was 3.95, indicating that predictions differed from actual values by an average of 3.95 rebounds per game.

4.3 Random Forest Regression Model

We also tested a Random Forest model, which captures complex interactions between variables. This model used the same predictors but applied a non-parametric approach through decision trees. We set `ntree` equal to 500 and `mtry` equal to 2 to ensure each tree used a random subset of variables at each split. The model resulted in an MAE of 3.45, which was better than Poisson regression but slightly worse than Negative Binomial Regression. While Random Forest provided flexibility in capturing interactions, it lacked interpretability compared to regression models.

4.4 Gradient Boosting Machines (GBM)

We implemented a GBM model, which improves predictions iteratively by minimizing errors in weak learners. This model achieved an MAE of 3.18, making it one of the best-performing models. It effectively handled nonlinear relationships but required careful tuning of hyperparameters such as the number of trees and interaction depth.

4.5 Neural Networks

We also explored a feedforward neural network using five hidden nodes and a regularization term to prevent overfitting. While the model captured complex patterns in the data, it had an MAE of 3.30, which was worse than the GBM and Negative Binomial Regression. Given its high computational cost and only moderate performance improvement, we did not select it as our final model.

4.6 The Best Model: Negative Binomial Regression

We applied a Negative Binomial Regression model using the `glm.nb()` function from the MASS package in R. This approach was chosen over the standard Poisson model due to the presence of overdispersion in our dataset—where the variance exceeded the mean—a common issue in count-based data. The Negative

Binomial model, by incorporating a dispersion parameter, provided a more flexible framework for capturing the variability in OREB.

Our results indicated that rebound differential remained the most influential predictor of OREB, while offensive rebound rate was slightly less significant compared to the Poisson model. The model's predictive performance improved, as reflected by the Mean Absolute Error (MAE), which decreased to 3.20. This means that, on average, our predictions deviated from actual OREB counts by 3.20 rebounds, marking a notable improvement over the Poisson regression model.

Although Gradient Boosting Machines (GBM) produced the lowest MAE (3.18), we ultimately selected the Negative Binomial Regression model for several reasons. First, since OREB is a discrete count variable, Negative Binomial Regression is more appropriate for handling overdispersed count data. Unlike GBM, which operates as a black-box machine learning model, Negative Binomial Regression offers better interpretability, allowing us to examine the impact of individual predictors on OREB. Additionally, we evaluated model selection criteria, such as the Akaike Information Criterion (AIC), which favored the Negative Binomial model over alternatives, indicating a better fit to the underlying data structure.

During our model evaluation process, we prioritized both predictive accuracy and model suitability for count-based data. While machine learning models like GBM can often yield slightly lower error metrics, their lack of interpretability makes them less ideal for analytical purposes. The Negative Binomial Regression model provided the best balance between accuracy (MAE = 3.20), interpretability, and suitability for count-based data, making it the optimal choice for predicting OREB.

4.7 Generating Predictions

To generate predictions for future games (March 7 - March 21, 2025), we first prepared the dataset by ensuring all necessary predictor variables—rebound differential, offensive rebound rate, defensive rebound rate, and minutes played differential—were updated with the latest available statistics. Missing values were handled using imputation methods to ensure data consistency before making predictions.

Next, we loaded the trained Negative Binomial Regression model and applied it to the dataset containing upcoming matchups. Using the `predict()` function in R, we computed the expected OREB values for each game, ensuring that predictions were generated on a per-game basis rather than team aggregates. This process involved passing the new dataset through the trained model, which applied the estimated coefficients to the input features to calculate expected OREB values for each game.

After generating the predictions, we conducted a brief sanity check by comparing the predicted OREB values against historical averages to ensure the outputs were within a reasonable range.