Call Of Duty Data Analysis

Andrew Eross, Owen Wassel, Jack Messina, Khang Le

2025-12-02

1 Introduction

Call of Duty is one of the most popular first person shooter games of all time. As a result, players from all over the world continue to play and try to improve their abilities to win more and more games. Today, we will be investigating the statistics of two Call of Duty Players, Player1 and Player2. We will also be diving into data from the different game modes of Call of Duty: Domination, Kill Confirmed, Hardpoint, and TDM. One of our goals is to learn more about what this data means, and most importantly want to answer some research objectives.

The objectives are as follows:

1. Which game mode is likely to reach the score limit?
2. What variables are predictors of TotalXP?
3. Predictive Match Outcome (Win/Loss)

In this project, we will be applying various statistical methods to answer these questions/objectives with an emphasis on the predictive modeling techniques kNN, Random Forest, and XGBoost. Before we get into those more advanced techniques, we must first explore the data and do necessary wrangling/tidying in order to apply the methods we listed. Objectives 1 and 2 will serve as exploratory data analysis (EDA) objectives to get a solid understanding of the data we are working with to ensure a correct execution of kNN, Random Forest, and XGBoost.

2 Data Summary and Exploratory Data Analysis

2.1 Variable Description

There are a total of 3 datasets that we have and will be working with: Player1, Player2, and GameModes. First, let's explore the Player1 and Player2 datasets.

Player1 and Player2 data are structured in a similar fashion and contain the same variables so it will be eaisest to observe them at the same time. The columns of these datasets represent match statistics and the rows represent each match the player participated in. These datasets contain the same amount of columns and statistic types, however Player1 has more rows meaning that they played more games than Player2. Each dataset contains 27 columns which is a lot of variables to work with, however a lot of them contain mostly null values so we will only be using a select amount of them. Those variables are: Choice, Result, Eliminations, Deaths, Score, Damage, and TotalXP.

Now for the GameModes dataset, only 3 variables are listed: Mode, ScoreLimit, and TimeLimit. Obviously these variables tell us the score and time limit of each game mode. These variables will also be useful to explore each match with the score and time limit for reason of the match concluding.

An important value that sticks out in the GameMode dataset is that the game mode Domination has no time limit. This will be worth noting when answer our EDA question below.

2.2 Exploratory Data Analysis

We can now perform some exploratory data analysis (EDA) to find relationships with the Player data and game modes data. We will look into which game mode is likely to reach the score limit and what variables are predictors of TotalXP.

To discover which game mode is likely to reach the score limit, some data wrangling must be in order. We first combine the Player1 and Player2 data and add a PlayerID column to the merged dataset. Then, we clean the GameModes set by taking off the game modes that contain HC and combining them with their respective game types. For example, the HC - Kill Confirmed game type will be combined with Kill Confirmed. We also removed rows with NA values. Figure 1 below displays the count of number of matches for each game mode.TDM (Team Deathmatch) has by far the most number of matches at 474, and Domination the least at 14.

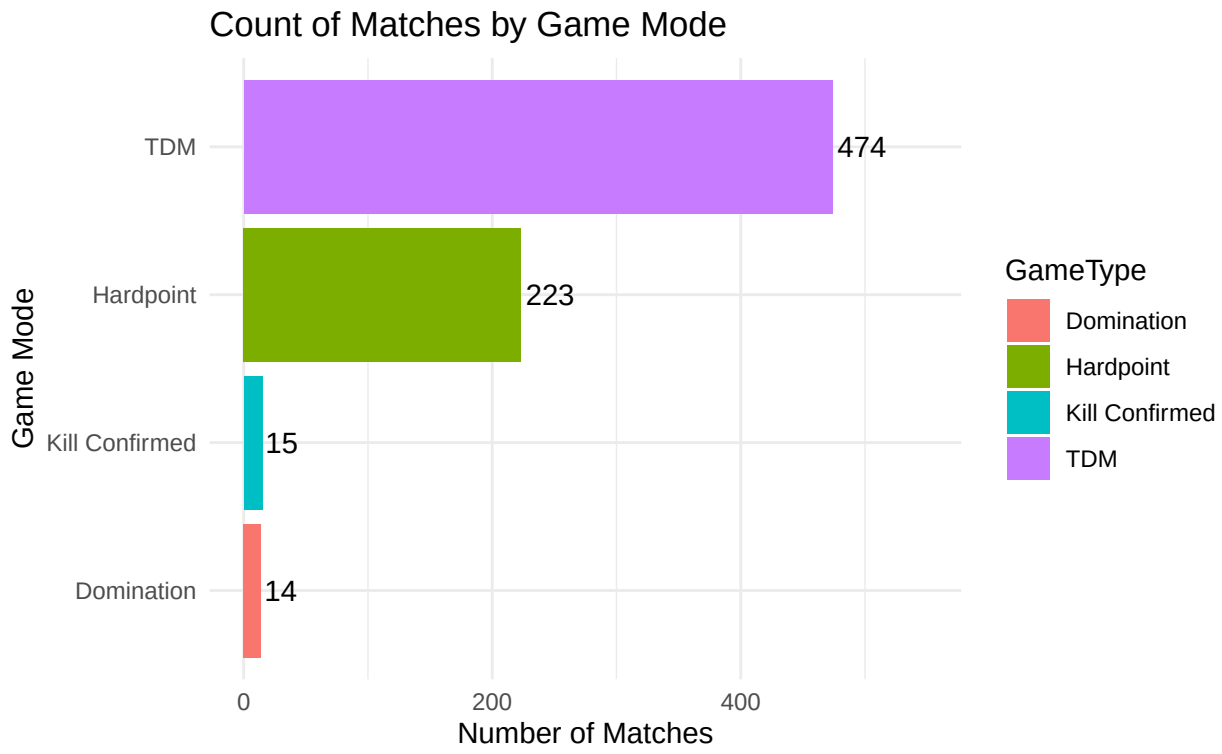


Figure 1: Count of each Game Mode

Next, we perform a join with the Game Modes data with the score and time limit variables. We do this because we want to create a score limit indicator and compare the match score. If these are equal, we yield a 1. Otherwise, 0. We can then calculate the proportions and display the results. The proportion of matches reaching score limit by game mode is shown in Figure 2 below.

Based on this visual, we can infer that Domination matches are most likely to reach the score limit since every one of their matches have reached the score limit. This of course makes sense because we pointed out in section 2.1 that Domination matches have no time limit, so matches have to end via score.

Team Deathmatch (TDM) matches are most likely to end via time limit. However, comparing this percentage with Hardpoint and Kill Confirmed is a bit difficult considering there are so many more TDM matches. If a similar number of matches were played between TDM, Hardpoint, and Kill Confirmed, the comparison in percentage of games ended via score limit might look a bit different.

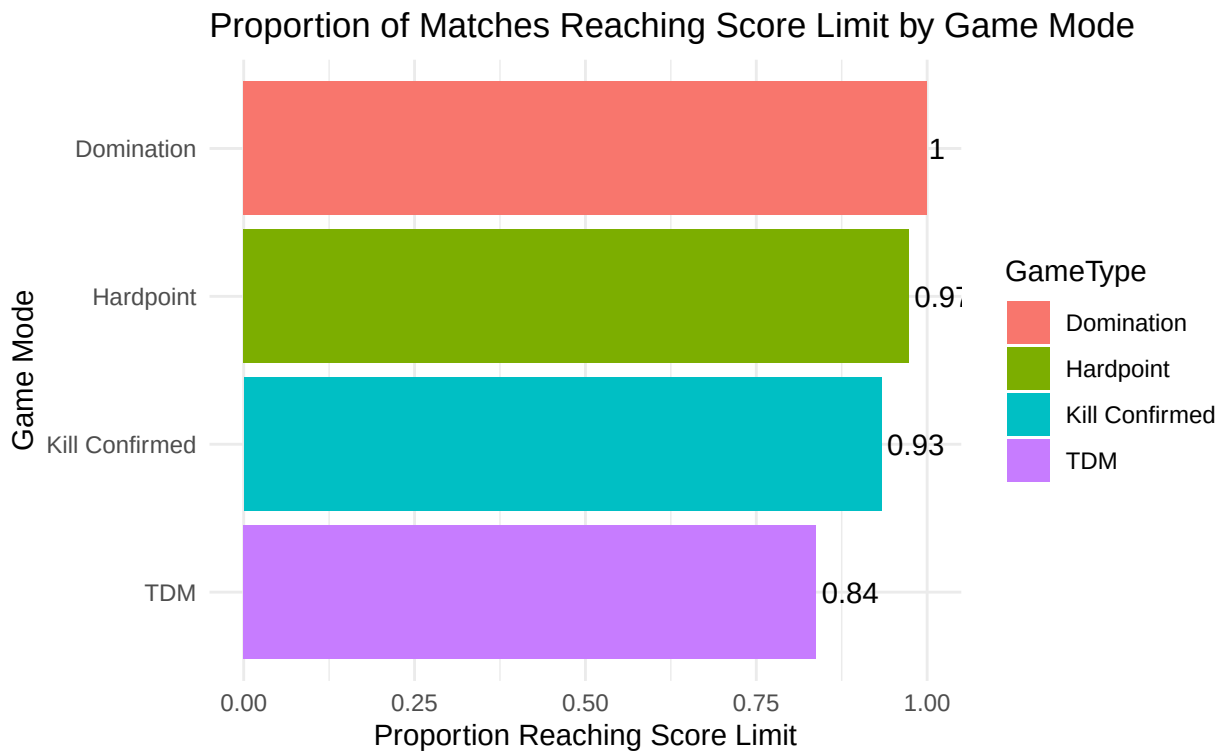


Figure 2: Proportion of Matches Reaching Score Limit by Game Mode

3 Inference Modeling

To accurately identify what variables are predictors of TotalXP, we must once again prepare our data. First, we mutate the TotalXP variable to ScaledXP. The column "XPType" for the Player1 and Player2 datasets indicate the factor a player's raw XP is changed by resulting in their TotalXP. For example, matches are either "10% Boost" or "Double XP + 10% Boost." For 10% boosted matches, the raw XP (what the player actually earned) was multipled by 1.1 to calculate the player's TotalXP. Double XP + 10% Boost matches had raw XP multiplied by 2.1 to find TotalXP. We will calculate our variable ScaledXP by performing the respective inverse operations on TotalXP, so identical game stats for 10% boosted matches and double + boosted matches earn the same XP for the sake of our models all else equal.

After scaling XP, we remove partial games and select only numeric variables from combined Player1 and Player2 dataset which removes map choice, primary weapon, game mode. Next, we remove variables with 50% or more missing values (columns specific to a game mode such as Confirms and Denies). Finally we must see that variables with near-zero variance are removed, but for this case, did not remove any columns. Now that we prepared the data for the model, we can create the model that decides which variables best predict TotalXP.

Our approach to identifying the best predictors involves using an AIC-based stepwise selection process. AIC is goodness-of-fit plus penalty for complexity. This algorithm valuates models by adding/removing predictors and chooses the combination with the lowest AIC, balancing predictive accuracy and model simplicity. Lower AIC values indicate a better model. Each additional predictor increases the penalty, so only variables that significantly improve fit (reduce deviance) are kept. We decided on this process because it provides an objective, numeric criterion for model comparison. For this process, each candidate model's AIC is compared; a drop of 2+ points in AIC generally indicates a meaningfully better model, while increases signal overfitting. When all was said and done, we obtained a final model containing only the most important predictors of ScaledXP.

The final variables we selected all yielded p-values less than 0.05, which we labeled as statistically significant:

- **TeamScore**
- **Eliminations**
- **Deaths**
- **Score**
- **Damage**
- **ReachedLimit**

Let's look deeper into the Eliminations variable as a means of predicting TotalXP. All else equal, for an increase of one elimination, we expect on average for ScaledXP to increase by about 109.01 (~119.911 increase in TotalXP for 10% XP Boosted matches and ~228.921 increase in TotalXP for Double XP + 10% Boosted matches).

4 Machine Learning Methods

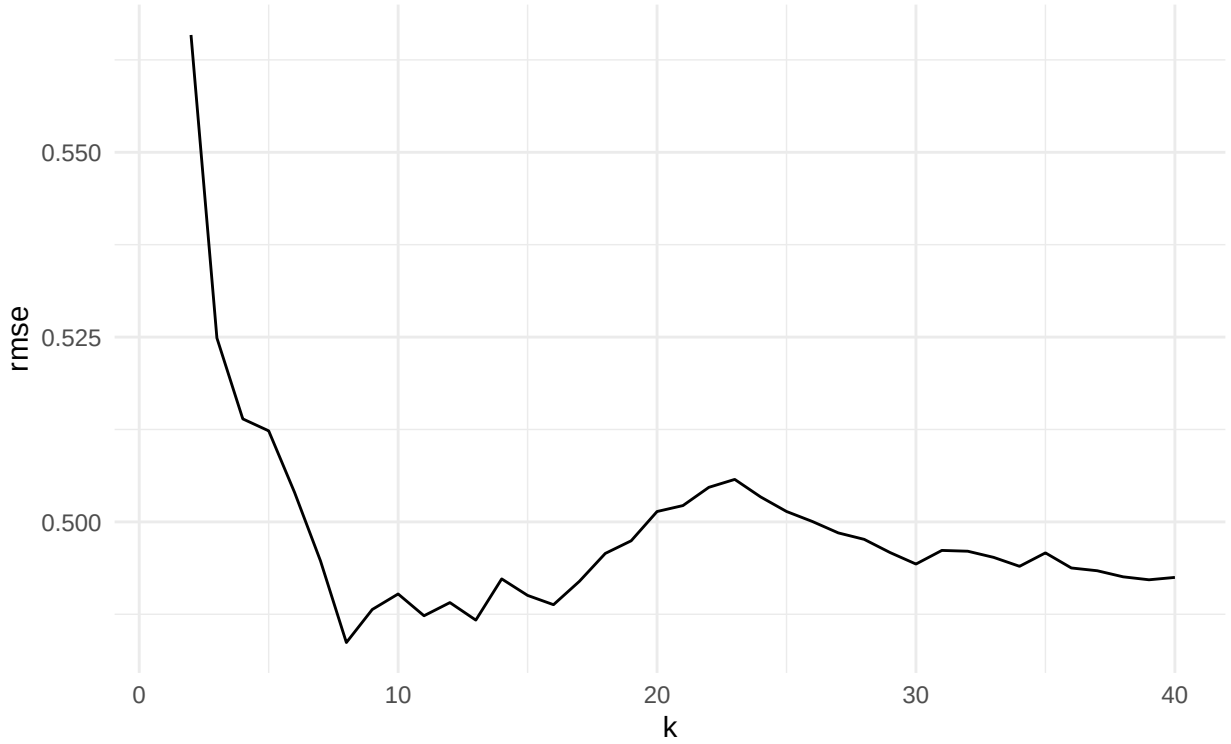
Before diving into our predictions for game outcome (win/loss), we must first establish some preliminary steps for a clean machine learning enviornment. The general process for each model (kNN, Random Forest, and XGBoost) is as follows:

- **Include only full matches**
- **Consider Draws as Losses**
  - Indicator Variable as 1 for Wins, 0 for losses and draws
  - Focus on winning percentage
- **Train/Test split of 80% to 20%**
- **Set Random Seed of 1103**
- **Set threshold for confusion matrix as 0.555**
  - Equal to the true winning percentage of all matches between players

The variables included in each model were Eliminations, Deaths, and Damage. We decided not to include TotalXP or Score in this model, because our intuition thought that these variables would be affected by winning or losing, meaning that winning increases score or XP, so it is not appropriate to consider these potential confounding variables.

4.1 k-Nearest Neighbors (kNN)

kNN was used to classify and predict wins and losses based on the variables we decided on (Eliminations, Deaths, and Damage). We created a nearest neighbors model from k values of 1 through 40. We iterated through each value of nearest neighbors and chose the one that yielded the lowest RMSE value. In the end, indicated by Figure 3, k=8 yielded the lowest RMSE at 0.4836806.



We can now create our final kNN model using k = 8 to predict the result of testing data to the threshold. As a reminder, predicting testing data will return a value on a scale of 0 to 1, and the true winning percentage of the matches is about 55%. Therefore, predicted results above 0.55 will be considered a win (equivalent to 1 in matrix), and anything below will be considered a loss (equivalent to 0 in matrix). For the confusion matrix below (Table 1- Actual results as columns and Predicted as Rows) we will also calculate the accuracy, precision, specificity, and sensitivity metrics in Table 2.

Table 1: Confusion Matrix for kNN Model

	0	1
0	42	34
1	20	50

Table 2: Metric Table for kNN Model

Metric	Value
Accuracy	0.630
Precision	0.714
Sensitivity	0.595
Specificity	0.677

The kNN model produces an accuracy of about 63%. This certainly is not great, but better than flipping a coin to predict wins and losses. The model is much better at predicting losses than wins, evident in only 20 False Positives (out of 62 total losses) to 34 False Negatives (out of 84 total wins). A potential conclusion we can draw from this is that a poor player performance is more likely to cause a team loss than a good player performance causing a team win, however more information will be needed to support this claim.

Another metric computed for evaluating the model was the Area Under the Curve (AUC) for the model's Receiver Operating Characteristics (ROC) Curve, which connects True Positive Rate and True Negative Rate pairings at each threshold from 0 to 1. The line y = x is plotted to show a curve with an AUC of 0.50, which would represent a 50/50 prediction of win or loss. For this curve, our AUC was 0.659.

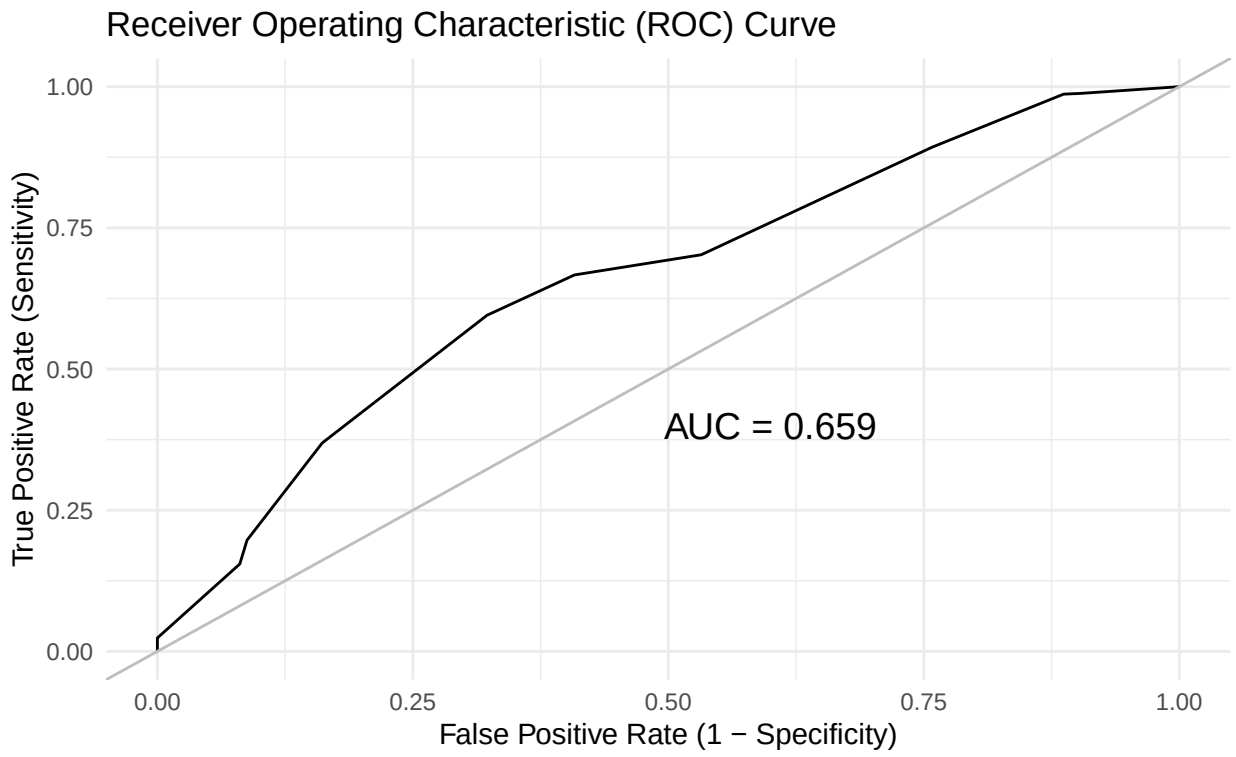


Figure 3: ROC Curve for kNN

4.2 Random Forest

Next, Random Forest was used to predict wins based on the same predictors (Eliminations, Deaths, and Damage). For this model, we created a single decision tree using the same random seed as before. Only a single decision tree was made because there were no methods used in the other two models to account for potential overfitting, such as k-fold cross validation. Before that model can be made, however, we must convert the Result variable into a factor with two levels. This ensures that the randomForest function creates a classification tree, rather than a regression tree.

Table 3: Confusion Matrix for Random Forest Model

	0	1
0	33	19
1	29	65

Table 4: Metric Table for Random Forest Model

Metric	Value
Accuracy	0.671
Precision	0.691

Metric	Value
Sensitivity	0.774
Specificity	0.532

The random forest model produced an accuracy of 67.1%, which is slightly better than the kNN model. This model correctly predicts whether the player won or lost approximately 2/3 of the time. It was very effective at correctly identifying the games in which the player won (hence the high precision), however, it failed to identify true negatives, with a specificity (or negative predictive value) of 53.2%, essentially a coin flip.

To futher look into this model, we will now evaluate the AUC of the ROC curve, just like we did for kNN. For this model, the AUC is 0.700, which is slightly better than kNN.

4.3 XGBoost

XGBoost (Extreme Gradient Boosting) is a boosting-based machine learning method that builds an ensemble of decision trees sequentially, where each new tree is trained to correct the errors of the previous ones. Because boosting can capture nonlinear relationships and complex interactions between predictors, we applied this method to classify match outcome (win vs. loss) using the same variables as the previous models: Eliminations, Deaths, and Damage. Before fitting the model, we converted the binary outcome into numeric form (1 = Win, 0 = Loss). We then created model matrices so that XGBoost could work with the numeric predictor set. The training/testing split and threshold setup remained consistent with kNN and Random Forest: Train/Test split = 80/20 Random Seed = 1103 Threshold = 0.555, equal to the true winning percentage Values above 0.555 were classified as wins, values below as losses We trained a binary logistic XGBoost model with 200 boosting rounds, allowing it to iteratively refine predictions by reducing classification error at each stage. Below is the confusion matrix for the XGBoost model, using the same threshold approach as the other methods:

Table 5: Corrected Confusion Matrix for XGBoost

	Loss	Win
Loss	51	28
Win	32	51

Table 6: Corrected XGBoost Evaluation Metrics

Metric	Value
Accuracy	0.6296
Precision	0.6145
Sensitivity	0.6456
Specificity	0.6456

The XGBoost model achieved an accuracy of 62.96%, indicating that it correctly predicted match outcomes slightly better than chance. Both precision (61.45%) and specificity (61.45%) suggest that the model performs similarly when predicting losses, correctly identifying negative outcomes at a moderate rate. Meanwhile, the sensitivity (64.56%) shows that the model is somewhat better at identifying wins than losses and AUC score at 0.663. Overall, XGBoost provides a balanced but modest level of predictive performance, capturing some meaningful patterns in player statistics but not reaching the higher accuracy and sensitivity achieved by the Random Forest model.A likely reason XGBoost underperformed is the low dimensionality of the feature set (only Eliminations, Deaths, Damage), which limits the algorithm's ability to capture complex interactions.

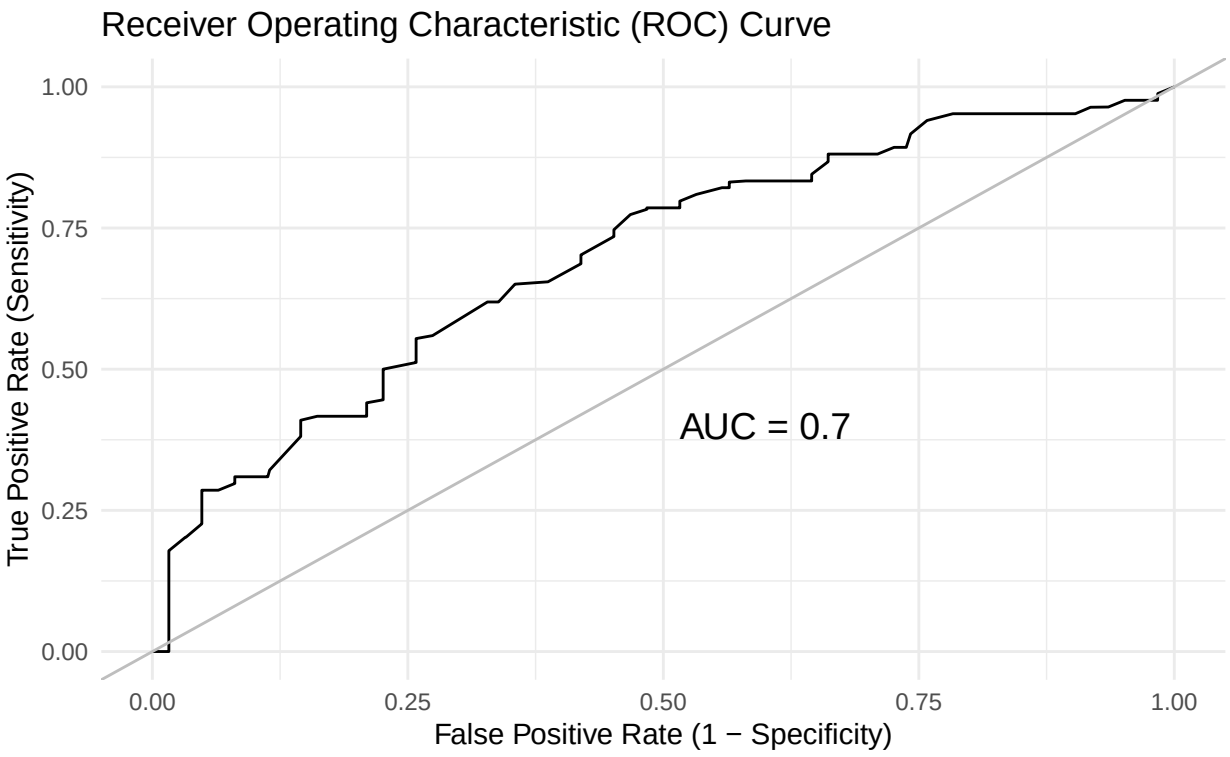


Figure 4: ROC Curve for Random Forest

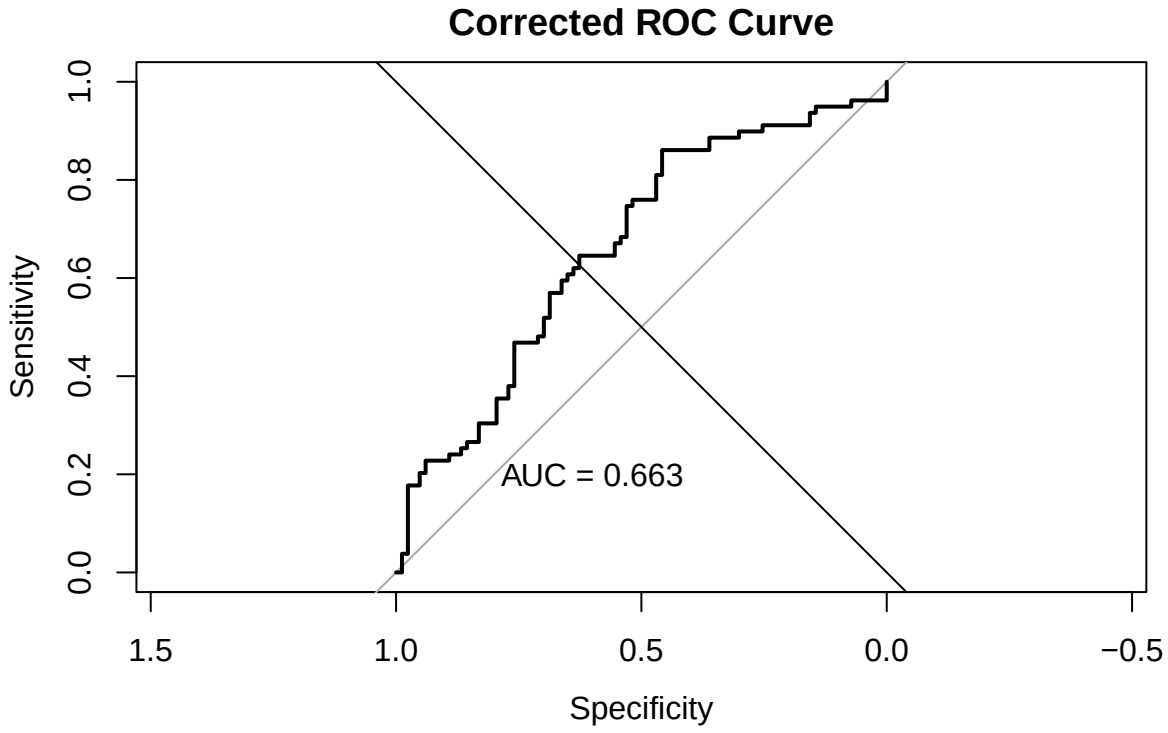


Figure 5: ROC Curve for XGBoost

5 Conclusion

In this project, we ran 3 different machine learning methods with differing results. Random Forest performed the best in terms of predicting match outcome (win/loss) and XGBoost performed the worst. Random Forest yielded an accuracy of 67.12% which is far better than simply a 50/50 chance at winning a match. Specificity was not as high however with a result of 53.23%. What really stood out in Random Forest was the sensitivity which yielded a 77.38%. This shows that the model does a really good job (relatively speaking) in identifying positive cases, in this case wins. The next best performing model was kNN. The accuracy was a little lower than the Random Forest at 63.014%, but performed better in terms of precision and specificity at 71.43% and 67.74% respectively. Sensitivity was far behind Random Forest's 77.38% though with a 59.52%. Finally, XGBoost was the worst performing model in terms of accuracy across all 3 with a 62.96%. However, specificity was higher than Random Forest at 64.56%. However, with relatively high precision across the rest of the models, XGBoost had the lowest at 61.45%.

Method	Accuracy	Precision	Sensitivity	Specificity
Random Forest	0.6712	0.6915	0.7738	0.5323
kNN	0.6301	0.7143	0.5952	0.6774
XGBoost	0.6296	0.6145	0.6456	0.6456

6 AI Statement

All group mates attest to utilizing generative AI (ChatGPT) to assist work throughout the project. AI usage was limited to debugging and not to create entire chunks of code. Each code chunk was created entirely by group members, but certain lines of code were fed into ChatGPT in case code was not outputting as expected, and then the code was modified with AI suggestions. Gen AI was used to assist group work and not to replace it.

7 Packages Used

tidyverse — https://cran.r-project.org/package=tidyverse

caret — https://cran.r-project.org/package=caret

pROC — https://cran.r-project.org/package=pROC

ggplot2 — https://cran.r-project.org/package=ggplot2

dplyr — https://cran.r-project.org/package=dplyr

readxl — https://cran.r-project.org/package=readxl

lubridate — https://cran.r-project.org/package=lubridate

FNN — https://cran.r-project.org/package=FNN

xgboost — https://cran.r-project.org/package=xgboost

kableExtra — https://cran.r-project.org/package=kableExtra

randomForest — https://cran.r-project.org/package=randomForest