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.

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. 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 537, and Domination the least at 14

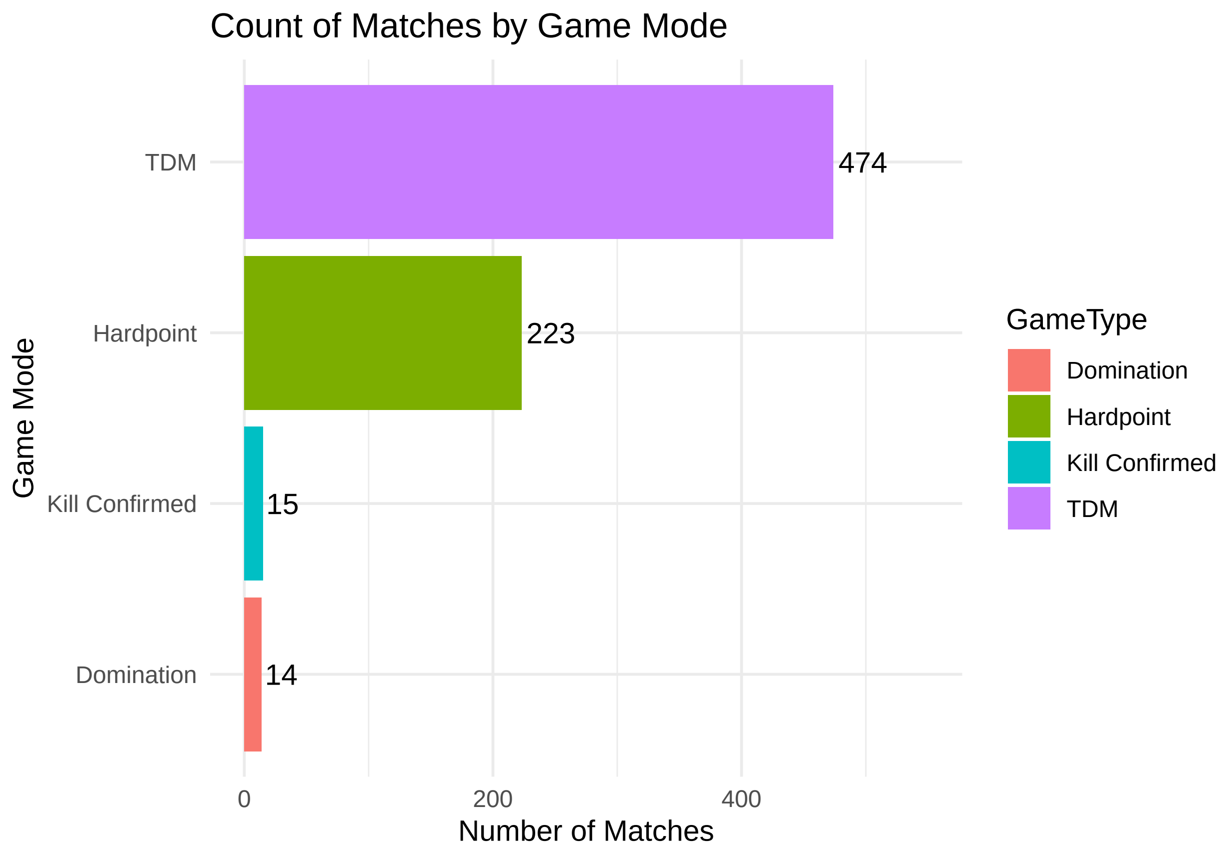


Figure 1: Count of each Game Mode

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 Hardpoint matches are most likely to reach the score limit since every one of their matches have reached the score limit. Kill Confirmed matches are most likely to reach the time limit. It is however important to note that there are only 14 Domination matches in the dataset, so it may not be completely fair to compare this percentage with TDM matches because there are more than 500 additional data points of TDM.

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

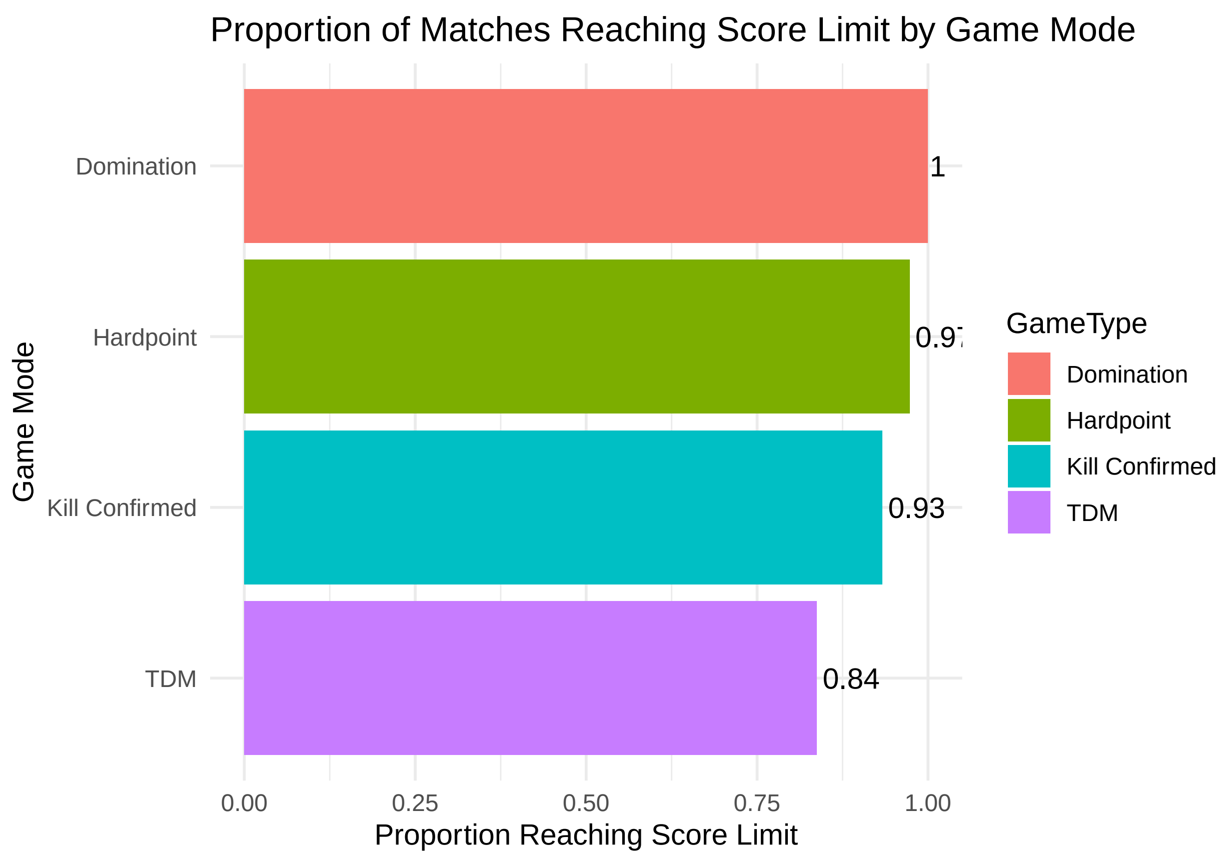


Figure 2: Proportion of Matches Reaching Score Limit by Game Mode

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.

3.1 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 use this k value to predict result of testing data and compare predicted probability of a win to threshold, creating confusion matrix.

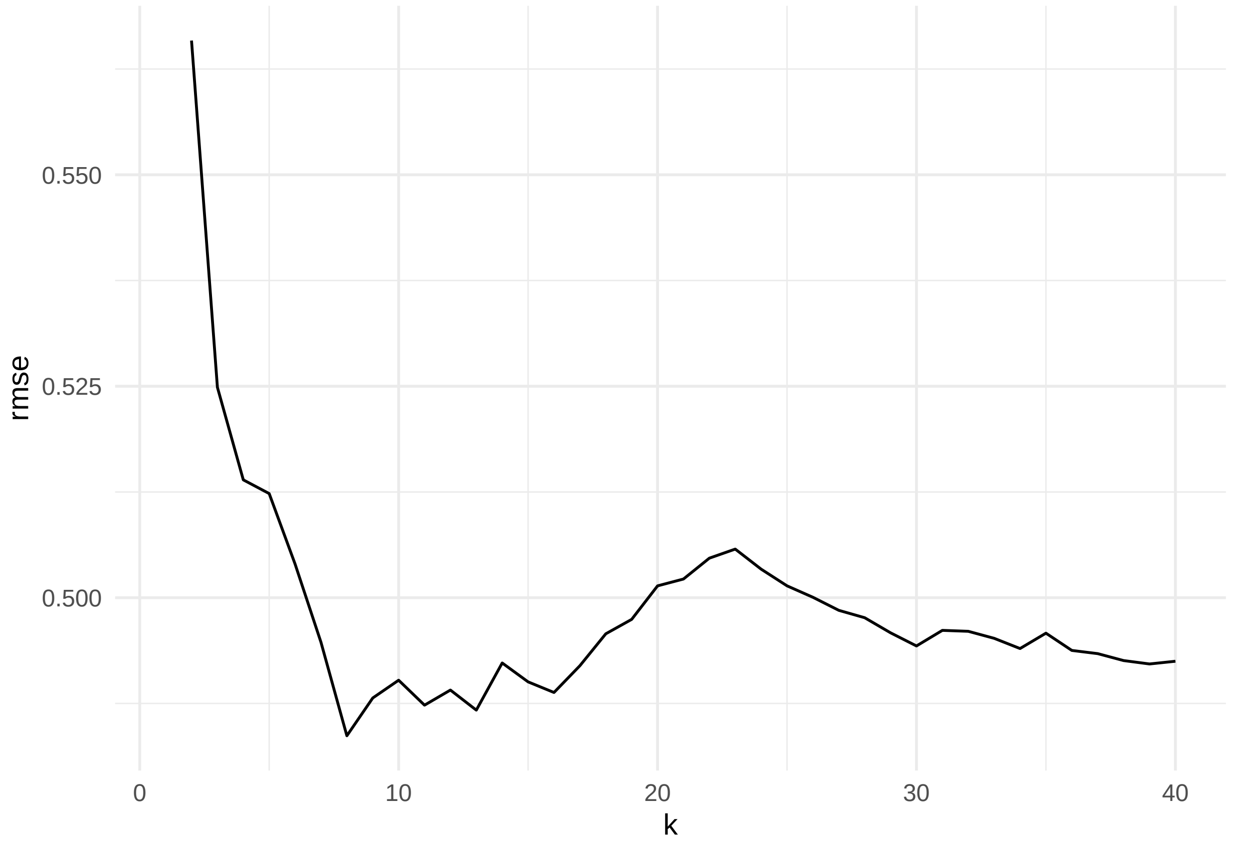


Figure 3: RMSEs by k Nearest Neighbors

```
knn_res8 <- knn.reg(train = Train[ , xvars, drop = FALSE],
                   test = Test[ , xvars, drop = FALSE],
                   y = Train$Result,
                   k = 8)

pred_prob <- knn_res8$pred

#Establish threshold
threshold <- 0.555

pred_win <- ifelse(pred_prob > threshold, 1, 0)

# Create confusion matrix
conf_mat <- table(Predicted = pred_win, Actual = Test$Result)
conf_mat
```

```
##            Actual
## Predicted  0  1
##         0 42 34
##         1 20 50
```

```
accuracy = (42+50) / (42+50+34+20)

accuracy
```

```
## [1] 0.630137
```

```
precision = 50 / (50 + 20)
precision
```

```
## [1] 0.7142857
```

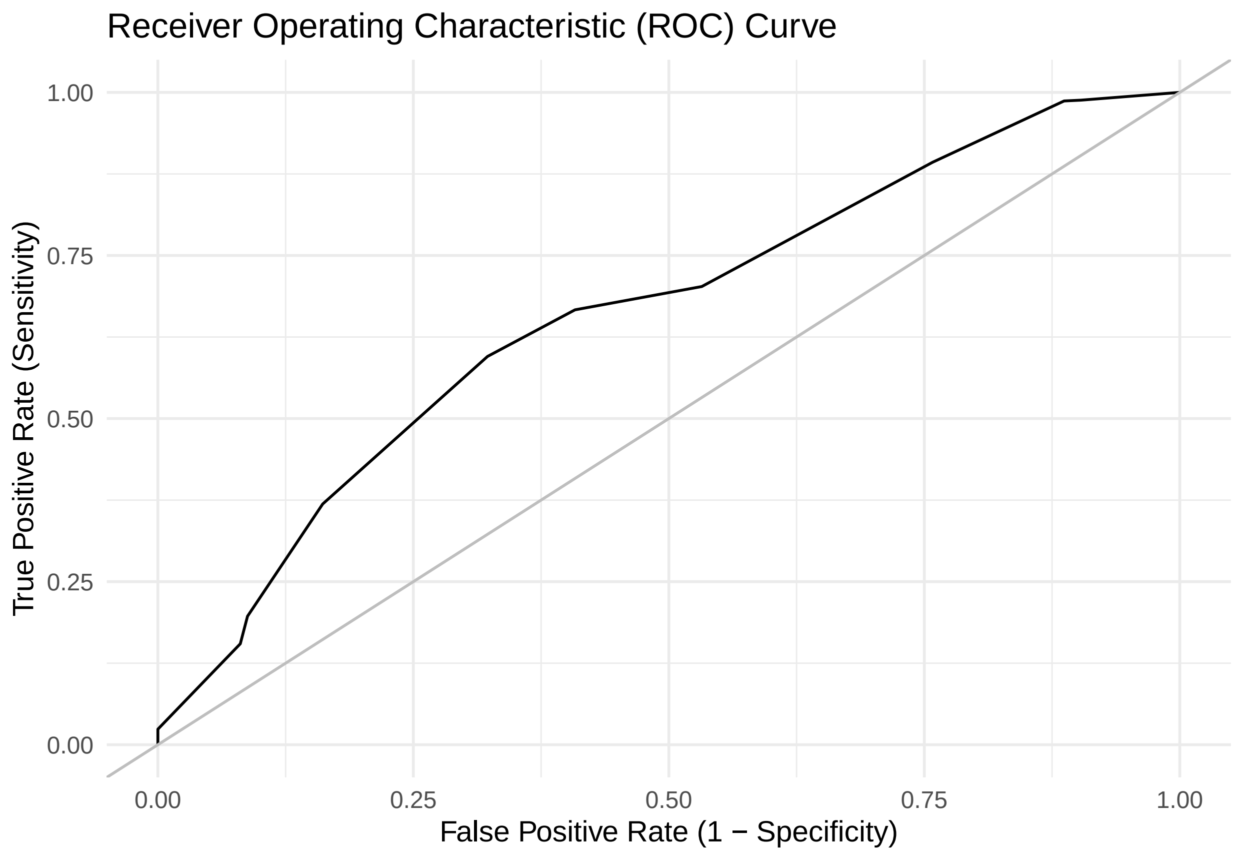
```
sens <- 50 / (50 + 34)
sens
```

```
## [1] 0.5952381
```

```
spec <- 42 / (42 + 20)
spec
```

```
## [1] 0.6774194
```

```
thresholds <- seq(1, 0, -0.01)
TPR_vec <- rep(NA, length(thresholds))
FPR_vec <- rep(NA, length(thresholds))

for(i in 0:length(thresholds)){

  t = thresholds[i]

  #find values in conf matrix
  TP <- sum(pred_prob > t & Test$Result == 1)
  FN <- sum(pred_prob < t & Test$Result == 1)
  TN <- sum(pred_prob < t & Test$Result == 0)
  FP <- sum(pred_prob > t & Test$Result == 0)

  #Find sensitivity
  TPR_vec[i] <- TP / (TP + FN)

  #Find 1-specificity
  FPR_vec[i] <- 1-(TN / (TN + FP))
}

roc_df <- data.frame(threshold = thresholds, TPR = TPR_vec, FPR = FPR_vec)

ggplot(data = roc_df, mapping = aes(x = FPR, y = TPR)) +
  geom_line(color = "black") +
  labs(x = "False Positive Rate (1 - Specificity)", y = "True Positive Rate (Sensitivity)", title = "Re
  geom_abline(intercept=0,slope=1, color = "grey") +
  theme_minimal()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (Sensitivity)
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
False Positive Rate (1 − Specificity)