Overwatch 2 Win Prediction

Kristopher Pouncy

2024-11-13

Introduction

For this project, I collected data over a course of a couple of weeks from Overwatch 2. The games are predominantly Competitive Mode with a few from Quick Play. The goal of this project is to predict the game `Result`, or game outcome, using only the data present in the leaderboard from a single snapshot.

Dataset Collection

Using screen capturing software and a MobileNetV2 model fine-tuned on character images, the leaderboard and additional information like `Game Mode` and `Time` are recorded when the user checks their stats during the game. The details were saved to a SQL database and then exported to a CSV file. The data was collected from PC games with cross-play compatibility enabled. The dataset contains games from Season 10 and from Season 14, both occurring in 2024 between the times of 12pm-3am. There are a total of 174 games in the dataset. There are 82 losses, 2 draws, and 90 Wins.

Known Limitation: One known limitation is that the text extraction software has issues with the numbers 11 and 0. These are typically NA. It should also be noted that when a character image is blocked or hard to detect, the `Character` value is assigned the name "HIDDEN".

Dataset Structure

The dataset includes the following columns:

- **GameID**: Unique identifier for each game.
- **SnapID**: Identifier for the specific game snapshot.
- **PlayerID**: Represents the player's position on the leaderboard (0-9), with:
  - **0–4**: Ally team
  - **5–9**: Enemy team
- **Mode**: The game mode being played.
- **Map**: Map on which the game is taking place.
- **Time**: Time elapsed in the game.
- **Character**: Character associated with the player.
- **K, A, D, Damage, H, MIT**: Player-specific performance metrics:
  - **K**: Number of kills.
  - **A**: Number of assists.
  - **D**: Number of deaths.
  - **Damage**: Total damage dealt by the player.
  - **H**: Healing provided by the player.
  - **MIT**: Mitigation, or damage prevented by the player.
- **Result**: Final outcome of the game.

Data Preparation

Before beginning the Exploratory Data Analysis (EDA), several preprocessing steps were applied to clean and transform the dataset:

1. **Reading and Transforming the Dataset**:
   - The dataset was loaded using `read.csv()` and cleaned using a series of transformations.
2. **Column Adjustments**:
   - The `Result` column was transformed to a numerical scale where:
     - `win` = 1
     - `draw` = 0
     - `lose` = -1
   - The `Character` column was cleaned by removing the "label_" prefix for consistency.
3. **Time Conversion**:
   - The `Time` string column was split into `Minutes` and `Seconds`, which were then combined into a single numeric `Time` column representing the time in minutes.
   - Rows with non-positive values for `Time` were filtered out to maintain data integrity.
4. **Removal of Unnecessary Columns**:
   - Columns not needed for analysis, such as `Latency`, `Minutes`, and `Seconds`, were removed.
5. **Handling Missing Values**:
   - Remaining `NA` values in the dataset were replaced with 0 to ensure clean data.
6. **Rollover previous Character Status if invalid or Hidden**
   - Due to classification error, characters could be incorrect or unknown. To fix this, the character from the previous snapshot is used instead

Background Information

In Overwatch 2, there are a total of 41 playable heroes. The heroes are categorized into three roles [1]:

- **Tank**: D.Va, Doomfist, Junker Queen, Mauga, Orisa, Ramattra, Reinhardt, Roadhog, Sigma, Winston, Wrecking Ball, Zarya
- **Damage**: Ashe, Bastion, Cassidy, Echo, Genji, Hanzo, Junkrat, Mei, Pharah, Reaper, Sojourn, Soldier: 76, Sombra, Symmetra, Torbjörn, Tracer, Venture, Widowmaker
- **Support**: Ana, Baptiste, Brigitte, Illari, Juno, Kiriko, Lifeweaver, Lúcio, Mercy, Moira, Zenyatta

Each team has 5 players composed of 1 Tank, 2 Damage, and 2 Support characters.

Below is an example of a some data collected from a Snapshot.

```
# show an example snapshot
dataset %>% filter(GameID == 2 & SnapID==0) %>%
  select(Mode, Map, PlayerID, Character, K, A, D, Result, Time)
```

```
## # A tibble: 10 x 9
##    Mode   Map     PlayerID Character     K     A     D Result  Time
##    <chr>  <chr>      <int> <chr>     <dbl> <dbl> <dbl>  <dbl> <dbl>
##  1 HYBRID MIDTOWN        0 Orisa         0     1     1      0  2.87
##  2 HYBRID MIDTOWN        1 Cassidy       7     0     3      0  2.87
##  3 HYBRID MIDTOWN        2 Sojourn       3     0     2      0  2.87
##  4 HYBRID MIDTOWN        3 Illari        0     2     1      0  2.87
##  5 HYBRID MIDTOWN        4 Ana           1     3     0      0  2.87
##  6 HYBRID MIDTOWN        5 DVa           3     3     1      0  2.87
##  7 HYBRID MIDTOWN        6 Sombra        5     2     2      0  2.87
##  8 HYBRID MIDTOWN        7 Bastion       5     0     2      0  2.87
##  9 HYBRID MIDTOWN        8 Mercy         1     3     3      0  2.87
## 10 HYBRID MIDTOWN        9 Juno          3     2     2      0  2.87
```

Additionally, historical data from Overbuff has been downloaded to compared with my dataset. Overbuff is a

website that tracks statistics for Overwatch, enabling players to view their own stats and how they compare against different ranks, game platforms, and timescales [2]. For this dataset, PC statistics for the year (2024) are used for comparisons.

Exploratory Data Analysis

EDA Objective

Before going into the EDA, I have three questions or assumptions that I would like to address and learn more about.

My three hypotheses:

(1) The number of swaps done by the Tank role is positively correlated with winning.

(2) Healing is not correlated with the game outcome.

(3) Damage is positively correlated with winning.

For hypothesis 1, many believe that swapping tanks during game will improve your chances of winning. **For hypothesis 2**, being a person that predominantly plays the support role, I hypothesize that healing isn't as important to winning as other factors. In fact, I feel like the more healing that is done, the more I'm losing. **For hypothesis 3**, the more damage that is done by damage characters, the greater our chances are of winning. However, from experience, it should be noted that this could be confounded by factors like healing. Note: the p-value of 0.05 will be used.

Character Analysis

Pick Rate

Which character is played the most for each player role? (Continued on next page)

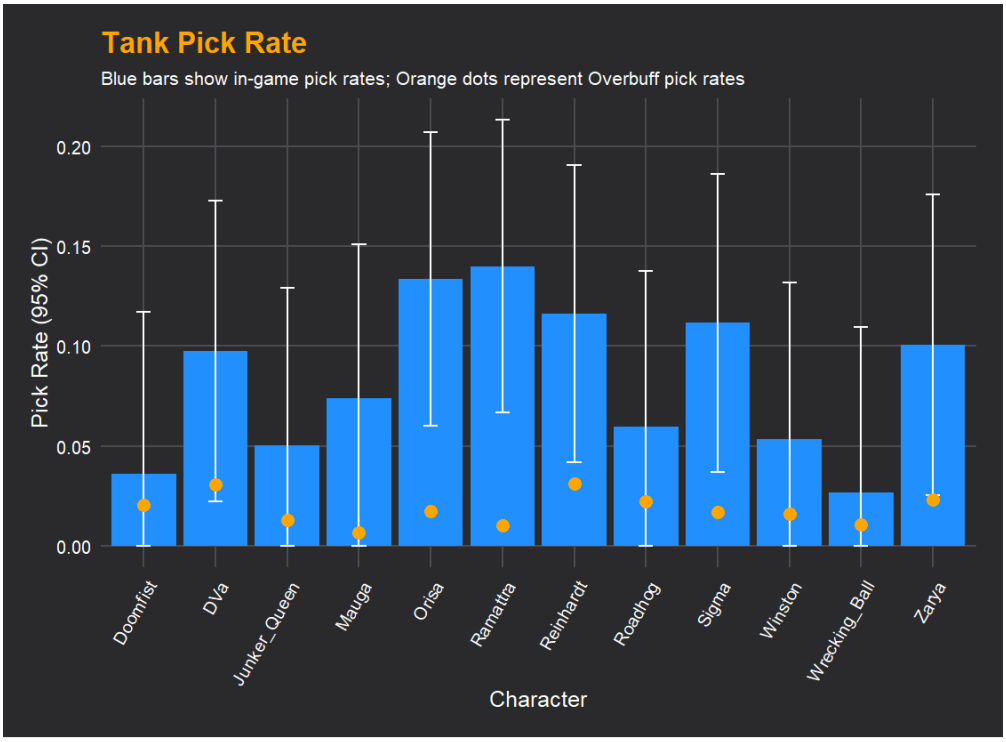


Figure 1: The confidence intervals are quite large due to small sample size. Some intervals don't contain orange dots, showing bias in the dataset.

```
plot_data
```

```
## # A tibble: 12 x 3
##    Character     proportion     me
##    <chr>              <dbl>  <dbl>
##  1 DVa               0.0975 0.0753
##  2 Doomfist          0.0362 0.0807
##  3 Junker_Queen      0.0503 0.0788
##  4 Mauga             0.0739 0.0768
##  5 Orisa             0.134  0.0734
##  6 Ramattra          0.140  0.0731
##  7 Reinhardt         0.116  0.0743
##  8 Roadhog           0.0597 0.0779
##  9 Sigma             0.112  0.0745
## 10 Winston           0.0535 0.0785
## 11 Wrecking_Ball     0.0267 0.0829
## 12 Zarya             0.101  0.0751
```

For tank characters, the pick rate ranges from 0.027 to 0.14. The least picked tank character is Wrecking Ball, and the most picked is Ramattra. Comparing the 95% confidence intervals with the Overleaf data, it appears that the dataset has a bias towards Orisa, Ramattra, Sigma, and Reinhardt.

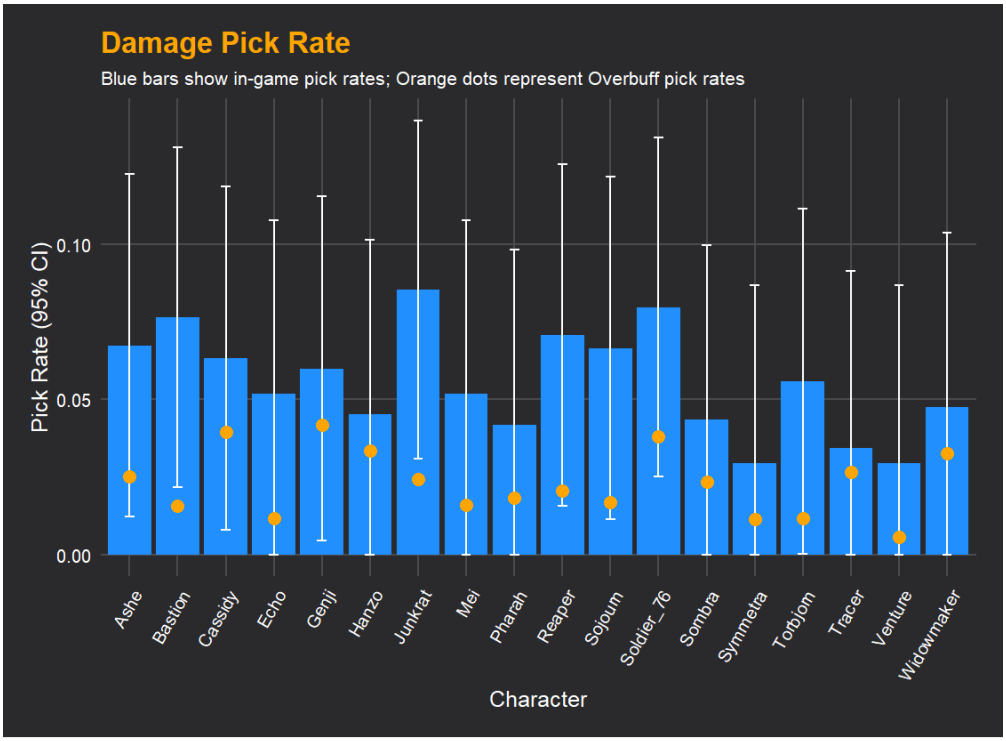


Figure 2: The confidence intervals are quite large due to small sample size. There is a bias for Junkrat and Bastion.

```
plot_data
```

```
## # A tibble: 18 x 3
##    Character  proportion     me
##    <chr>           <dbl>  <dbl>
##  1 Ashe           0.0673 0.0551
##  2 Bastion        0.0764 0.0547
##  3 Cassidy        0.0632 0.0552
##  4 Echo           0.0517 0.0558
##  5 Genji          0.0599 0.0554
##  6 Hanzo          0.0452 0.0561
##  7 Junkrat        0.0854 0.0543
##  8 Mei            0.0517 0.0558
##  9 Pharah         0.0419 0.0563
## 10 Reaper         0.0706 0.0549
## 11 Sojourn        0.0665 0.0551
## 12 Soldier_76     0.0796 0.0546
## 13 Sombra         0.0435 0.0562
## 14 Symmetra       0.0296 0.0573
## 15 Torbjorn       0.0558 0.0556
## 16 Tracer         0.0345 0.0569
## 17 Venture        0.0296 0.0573
## 18 Widowmaker     0.0476 0.0560
```

For damage character, the pick rate ranges from 0.0296 to 0.0854. The least picked damage character is Symmetra, and the most picked is Junkrat. Comparing the 95% confidence intervals with the Overleaf data, it appears that Junkrat and Bastion are played more frequently in my games.

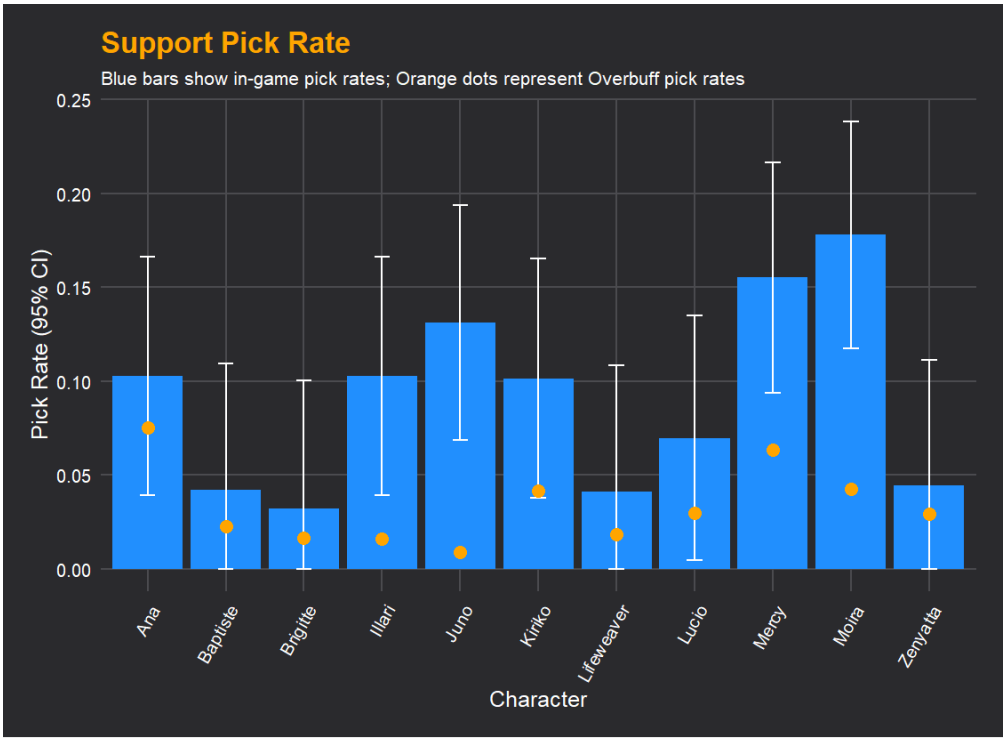


Figure 3: The confidence intervals are quite large due to small sample size. There is a bias towards four characters that I typically play: Illari, Juno, Mercy, and Moira.

```
plot_data
```

```
## # A tibble: 11 x 3
##    Character  proportion     me
##    <chr>           <dbl>  <dbl>
##  1 Ana            0.103  0.0636
##  2 Baptiste       0.0422 0.0670
##  3 Brigitte       0.0319 0.0682
##  4 Illari         0.103  0.0636
##  5 Juno           0.131  0.0624
##  6 Kiriko         0.101  0.0636
##  7 Lifeweaver     0.0410 0.0671
##  8 Lucio          0.0696 0.0652
##  9 Mercy          0.155  0.0614
## 10 Moira          0.178  0.0605
## 11 Zenyatta       0.0445 0.0668
```

For support characters, the pick rate ranges from 0.0319 to 0.178. The least picked support character is Brigitte, and the most picked is Moira. Comparing the 95% confidence intervals with the Overleaf data, there also appears to be a bias towards Mercy, Moira, Illari, and Juno. In fact, I predominantly played support class and cycle through these support characters in my games.

Win Rate

What is the top winning character for each role?

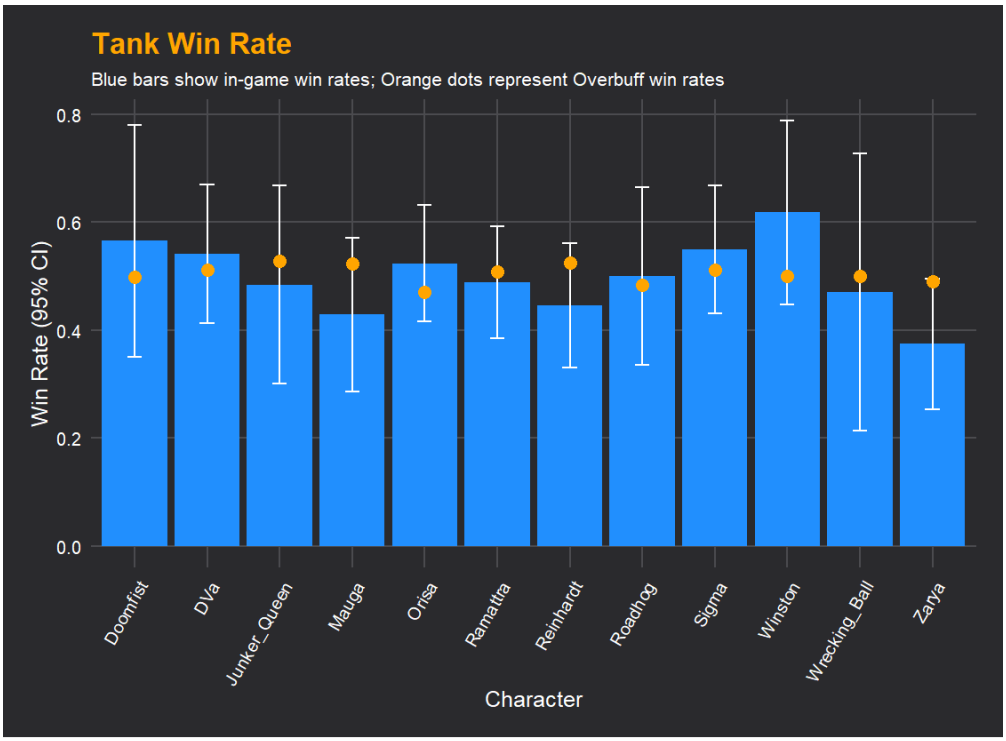


Figure 4: There are no significant differences between win rates for the Tank role.

```
plot_data %>% filter(Character %in% overbuff_plot$Character)
```

```
## # A tibble: 12 x 4
##    Character     win_rate count    me
##    <chr>            <dbl> <int> <dbl>
##  1 DVa              0.541    61 0.128
##  2 Doomfist         0.565    23 0.214
##  3 Junker_Queen     0.484    31 0.183
##  4 Mauga            0.429    49 0.142
##  5 Orisa            0.524    84 0.108
##  6 Ramattra         0.489    92 0.104
##  7 Reinhardt        0.446    74 0.115
##  8 Roadhog          0.5      38 0.164
##  9 Sigma            0.549    71 0.118
## 10 Winston          0.618    34 0.170
## 11 Wrecking_Ball    0.471    17 0.257
## 12 Zarya            0.375    64 0.121
```

For tanks, the win rate ranges from 0.375 to 0.618. Looking at the confidence intervals, there isn't enough data to see which characters are more likely to win a game. However, the data does show that there isn't a difference in win rate between my games and others. The plot above shows that Winston won the most, while Zarya lost the most.

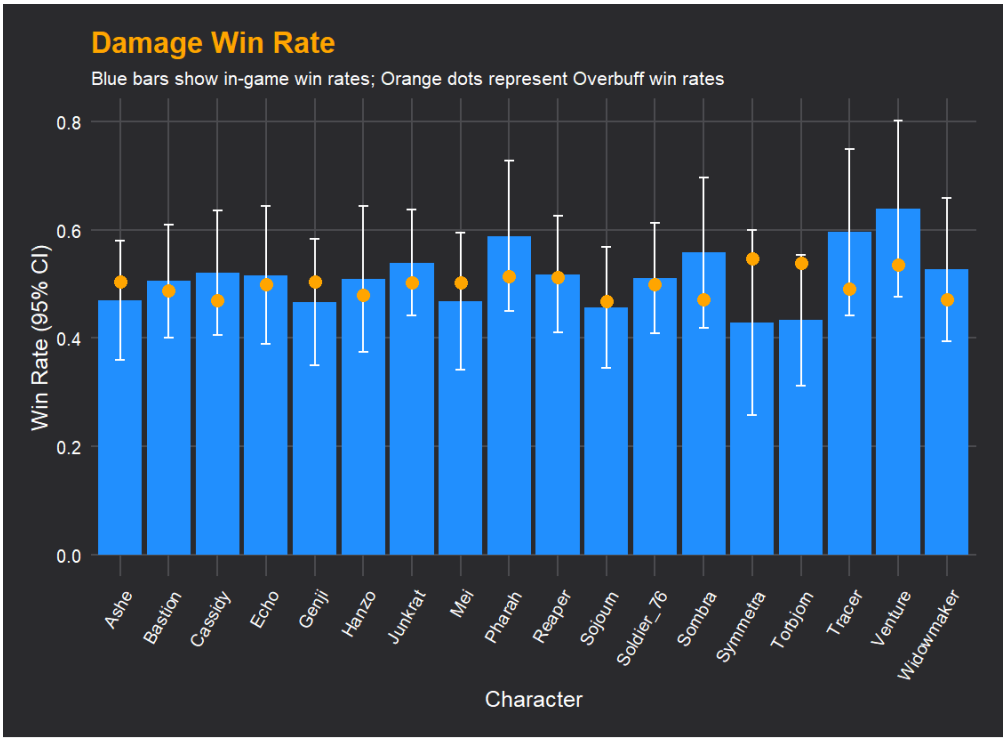


Figure 5: There are no significant differences between win rates for the Damage role.

```
plot_data %>% filter(Character %in% overbuff_plot$Character)
```

```
## # A tibble: 18 x 4
##    Character  win_rate count     me
##    <chr>         <dbl> <int>  <dbl>
##  1 Ashe          0.469    81 0.110
##  2 Bastion       0.505    91 0.104
##  3 Cassidy       0.52     75 0.115
##  4 Echo          0.516    62 0.127
##  5 Genji         0.466    73 0.116
##  6 Hanzo         0.509    55 0.135
##  7 Junkrat       0.539   102 0.0979
##  8 Mei           0.468    62 0.127
##  9 Pharah        0.588    51 0.138
## 10 Reaper        0.518    85 0.108
## 11 Sojourn       0.456    79 0.112
## 12 Soldier_76    0.510    96 0.101
## 13 Sombra        0.558    52 0.138
## 14 Symmetra      0.429    35 0.170
## 15 Torbjorn      0.433    67 0.121
## 16 Tracer        0.595    42 0.153
## 17 Venture       0.639    36 0.163
## 18 Widowmaker    0.526    57 0.132
```

For damage characters, the win rate ranges from 0.429 to 0.639. Looking at the confidence intervals, there isn't enough data to see which characters are more likely to win a game. However, the data does show that there isn't a difference in win rate between my games and others. The plot above shows that Venture won the most, while Symmetra (closely followed by Torbjorn) lost the most.

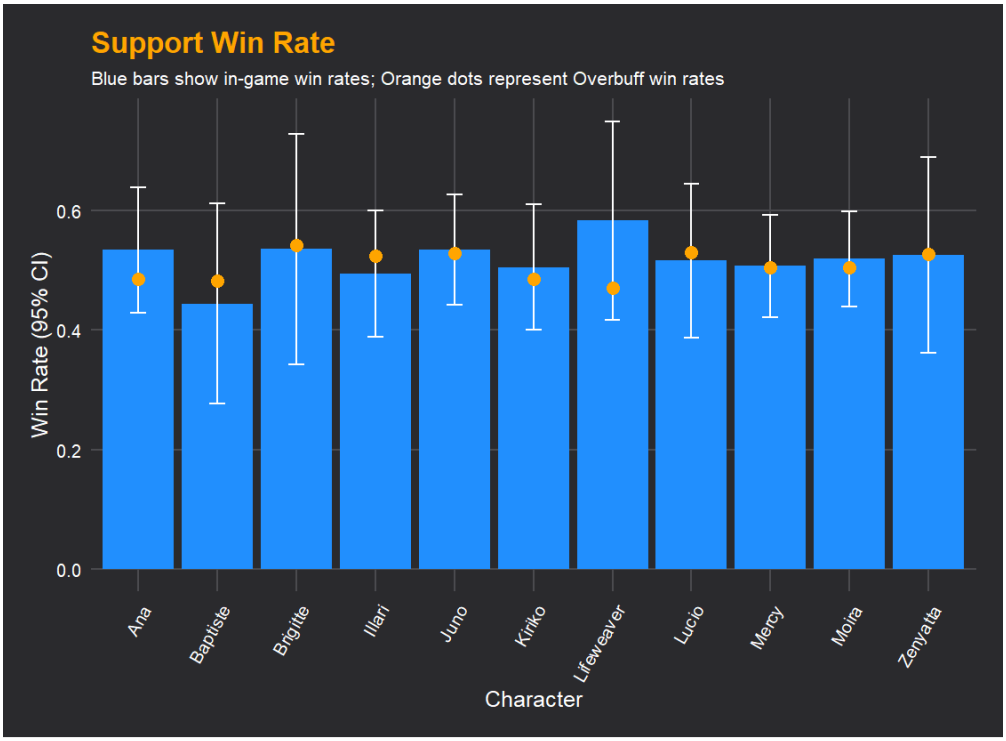


Figure 6: There are no significant differences between win rates for the Support role.

```
plot_data %>% filter(Character %in% overbuff_plot$Character)
```

```
## # A tibble: 11 x 4
##    Character  win_rate count     me
##    <chr>         <dbl> <int>  <dbl>
##  1 Ana           0.534    88 0.106
##  2 Baptiste      0.444    36 0.168
##  3 Brigitte      0.536    28 0.193
##  4 Illari        0.494    89 0.105
##  5 Juno          0.535   114 0.0925
##  6 Kiriko        0.506    89 0.105
##  7 Lifeweaver    0.583    36 0.167
##  8 Lucio         0.517    60 0.129
##  9 Mercy         0.507   134 0.0854
## 10 Moira         0.519   154 0.0795
## 11 Zenyatta      0.526    38 0.164
```

For support characters, the win rate ranges from 0.444 to 0.583. Looking at the confidence intervals, there isn't enough data to see which characters are more likely to win a game. The plot above shows that Lifeweaver won the most, while Baptiste lost the most.

Swapping Characters

What are the average number of characters played by each player role during a game?

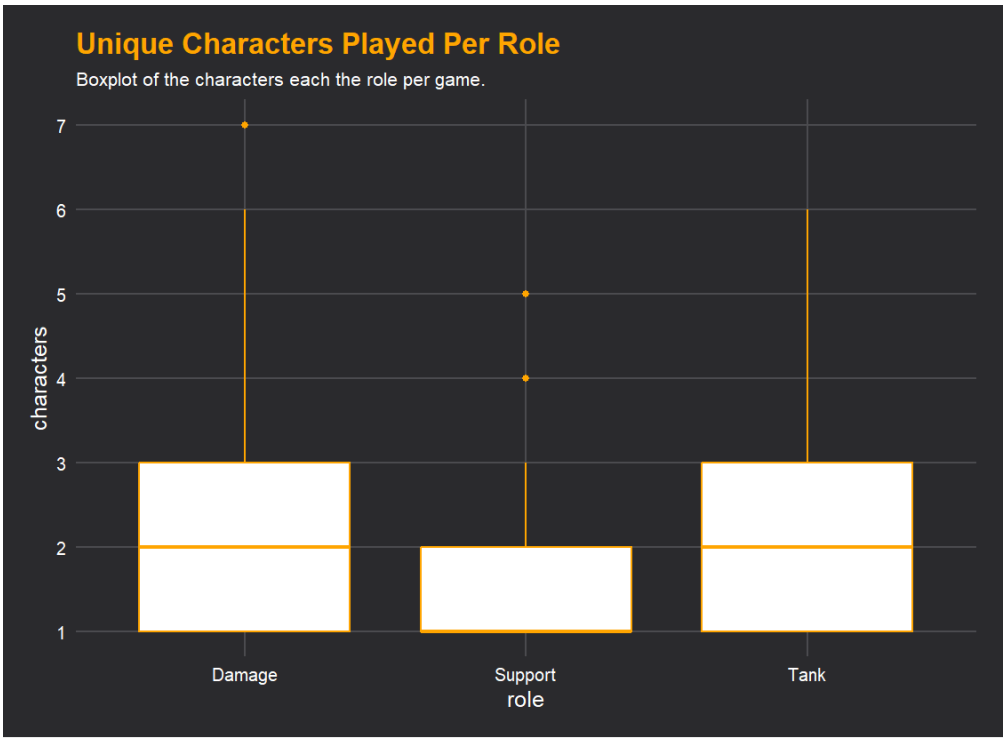


Figure 7: Each role typically plays 1-3 unique characters throughout a game.

```
plot_data %>% group_by(role) %>%
  reframe(avg_characters=mean(characters)) %>%
  arrange(desc(avg_characters))
```

```
## # A tibble: 3 x 2
##   role    avg_characters
##   <chr>            <dbl>
## 1 Damage            2.14
## 2 Tank              2.05
## 3 Support           1.66
```

It looks like Support characters are more likely to stick with 1-2 characters. Damage and Tank players, on the other hand, typically have 1-3 characters to swap to per game.

What are the average number of swaps done by each player role during a game?

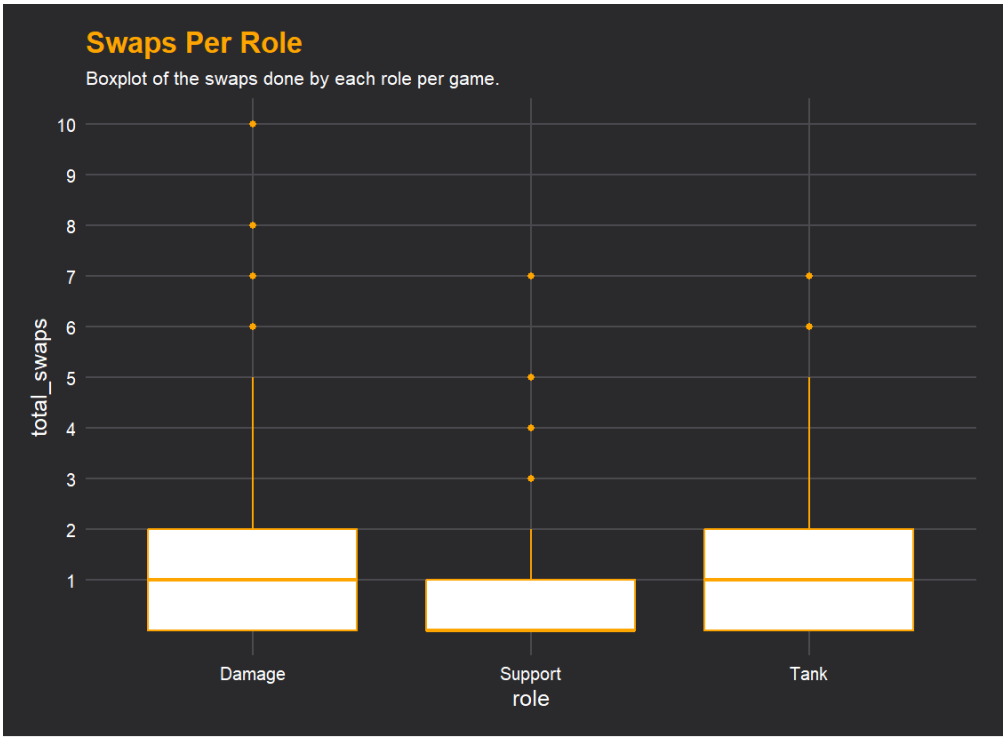


Figure 8: The median number of swaps in a game is 1 for Tank and Damage, and 0 for support.

```
plot_data %>% group_by(role) %>%
  reframe(avg_swaps=mean(total_swaps)) %>%
  arrange(desc(avg_swaps))
```

```
## # A tibble: 3 x 2
##   role    avg_swaps
##   <chr>       <dbl>
## 1 Damage      1.44
## 2 Tank        1.40
## 3 Support     0.813
```

Tank and damage characters swap more frequently compared to the support role. Support characters are least likely to swap 3+ times, whereas damage and tank players are least likely to swap 6+ times.

Analysis of Tank Swaps

(1) The number of swaps done by the Tank role is positively correlated with winning.

Players in the Overwatch community coined the term 'counter-watch' to describe the adaptive strategy of picking a character that counters the enemy team's composition. When one character swaps, a character on the other swaps in response, and it repeats. I'm curious about how often Tanks (arguably an important role in the game) swap and if there are correlations with swap frequency and with game outcome.

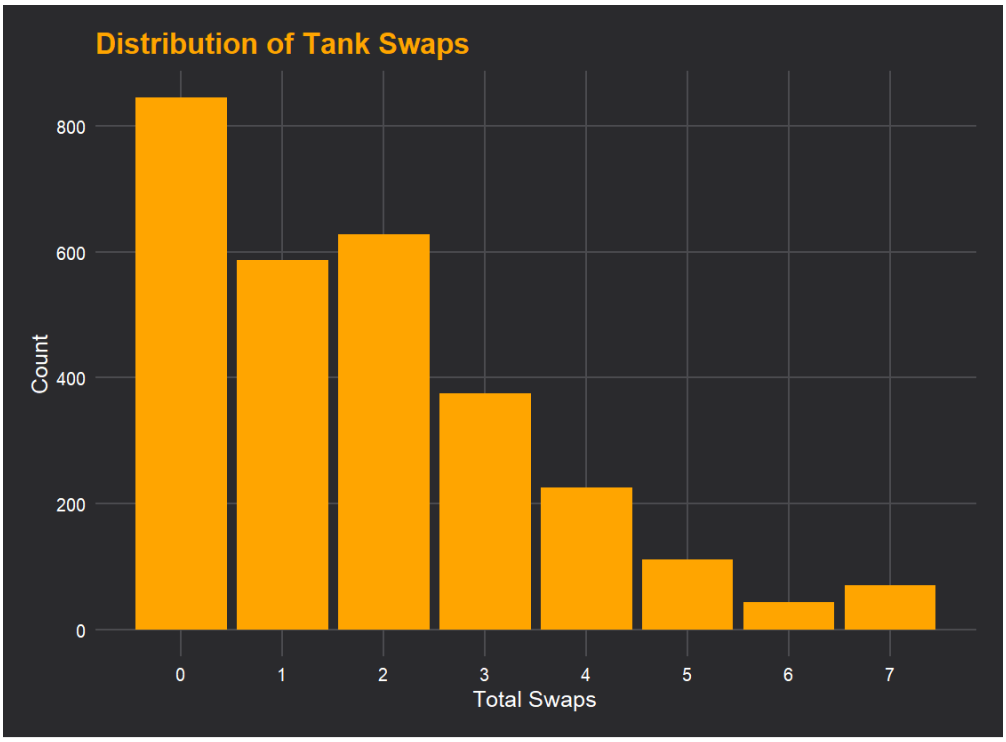


Figure 9: Histogram of the number of swaps a Tank does in a game.

The plot above shows the distribution of tank swapping with the majority of games having tanks not swapping at all.

(Continued on next page)

Is there a correlation between swapping and game outcome?

*given that there are only 2 draws, they have been excluded from data plotted data below

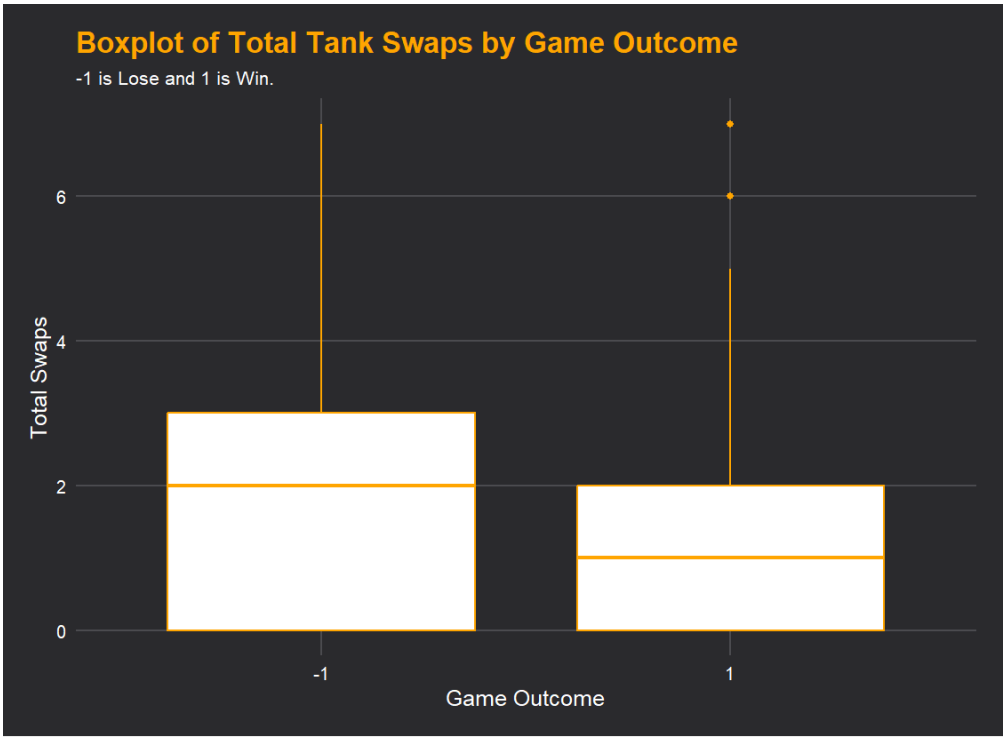


Figure 10: The median number of Tank swaps is higher for a game that resulted in a loss (-1).

Based on the box plot, it looks like tanks are more likely to swap when the game outcome is a Loss. Perhaps they lost because they swap more or they swapped more because they were losing.

```
# The Kendall's rank test is used due to the large amount of ties
cor.test(plot_data$total_swaps, plot_data$result, method="kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  plot_data$total_swaps and plot_data$result
## z = -7.128, p-value = 1.018e-12
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##        tau
## -0.1182669
```

There appears to be a weak negative correlation between tank swap frequency and game outcome with a p-value less than 0.05.

Is there a correlation between ally tank swap count and enemy tank swap count?

```
##
##  Spearman's rank correlation rho
##
```

```
## data:  plot_data$`0` and plot_data$`5`
## S = 605765, p-value = 6.83e-05
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.2980084
```

There appears to be a weak positive correlation between the number of swaps done by the Tanks from each team.

Stats

What is the average win rate?

```
dataset %>% distinct(GameID, .keep_all = TRUE) %>%
  filter(Result != 0) %>%
  summarize(win_rate=mean(Result==1))
```

```
## # A tibble: 1 x 1
##   win_rate
##      <dbl>
## 1    0.523
```

How are metrics distributed for K, A, D, Damage, H, MIT?

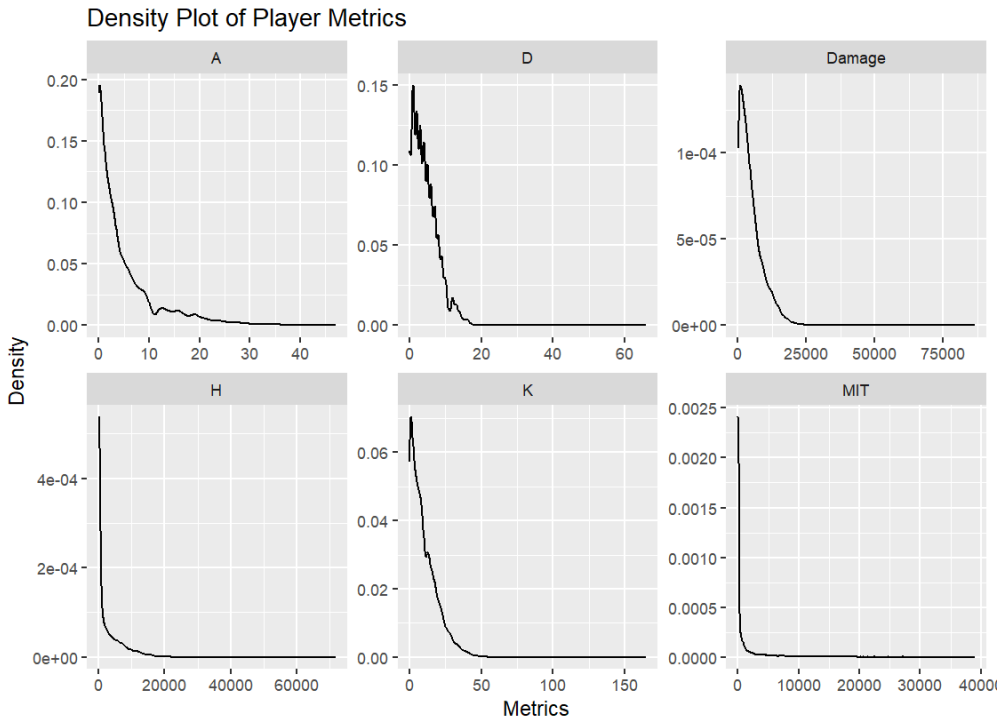


Figure 11: Density plots of player metrics before removing extreme values.

There appears to be some outliers in the data. For example, there is a slight bump in 100 kills, which means

that the OCR reader likely adds an extra zero to `10`. 100 Kills are very unlikely in a 15 minute game. This will be more evidence in the next plot.

How do the metrics change over time?

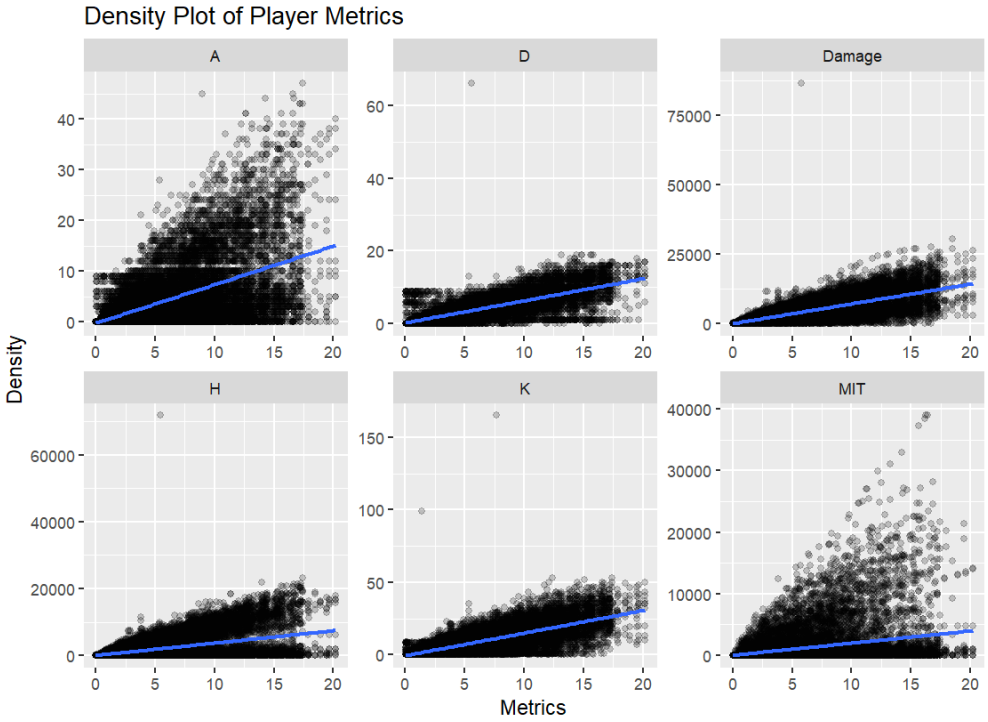


Figure 12: Density plots with regression trend of data without extreme values removed. These outliers are more evident when data is faceted by time.

Viewing player metrics over time, there are some clear outliers. For example, Deaths (`D`) has a value more than 60 at around 5 minutes, `Damage` has more than 75,000 around 5 minutes, and Kills (`K`) has 100 near the start of the game. These will need to be kept in mind when doing predictive modeling.

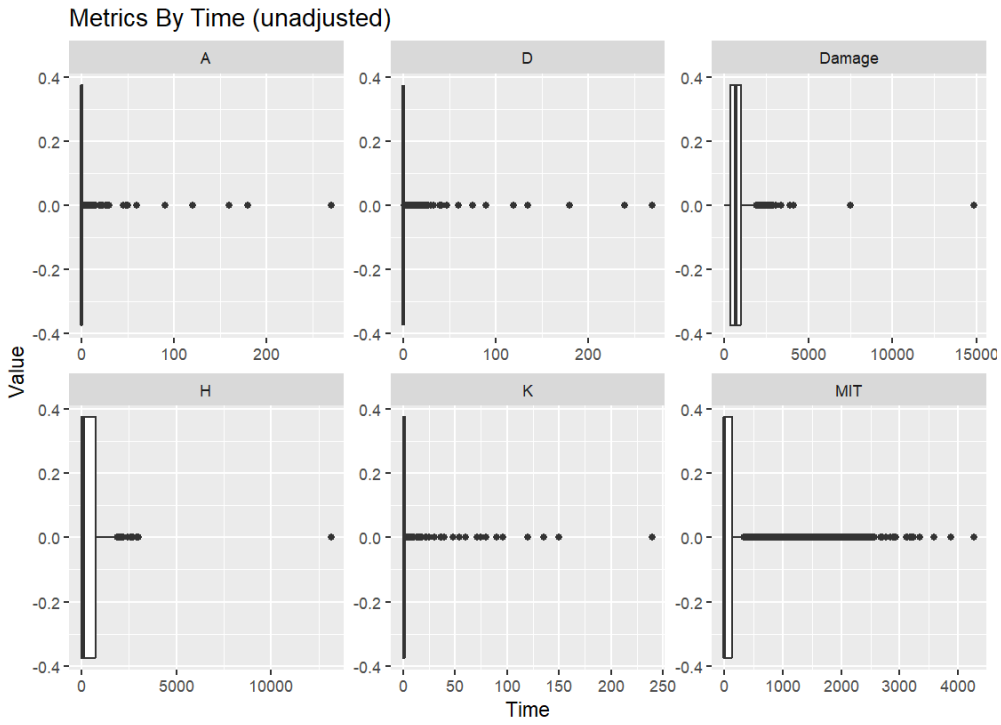


Figure 13: Boxplot of data facted by time, without extreme values removed.

```r
adjusted_dataset <- dataset %>%
  mutate(
    across(c(K, A, D, Damage, H, MIT), ~ .x / Time),  # Normalize metrics by Time
    K = pmin(K, 5),
    A = pmin(A, 4),
    D = pmin(D, 3),
    Damage = pmin(Damage, 2500),
    H = case_when(
      PlayerID %in% c(0, 5) ~ pmin(H, 600),
      PlayerID %in% c(1, 2, 6, 7) ~ pmin(H, 400),
      TRUE ~ pmin(H, 2500)
    ),
    MIT = pmin(MIT, 2500)
  )

adjusted_dataset %>% select(PlayerID, K, A, D, Damage, H, MIT, Time) %>%
  pivot_longer(cols = c(K, A, D, Damage, H, MIT)) %>%
  ggplot(aes(x = value)) +
  geom_boxplot() +
  facet_wrap(~name, scales = "free") +
  labs(
    title = "Metrics By Time (adjusted)",
    y = "Value",
    x = "Time"
  )
```

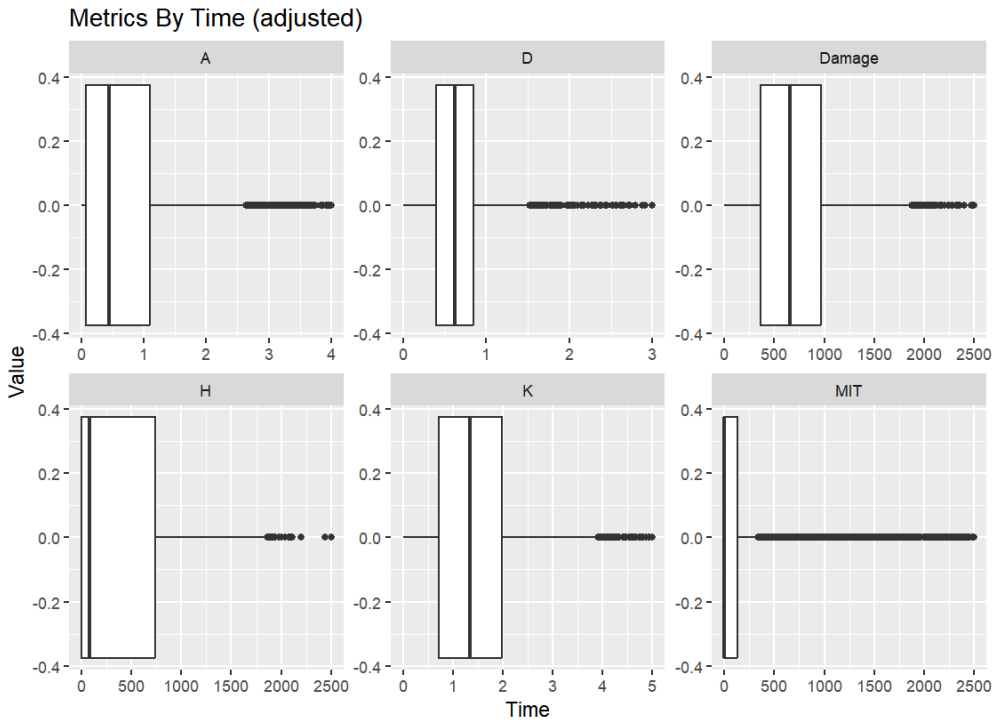


Figure 14: Boxplot of corrected data facted by time.

In the **adjusted version**, clipping is applied to limit extreme outliers; the capped-limits were determined by previous recorded data and industry knowledge (i.e., gaming experience). For example, Kills (`K`) are capped at 5 per unit of time, Assists (`A`) at 4, and `Damage` at 2500. Healing (`H`) is further tailored by role, with lower caps for damage and tank players. This ensures outliers don't disproportionately skew the results while aligning metrics with role-specific expectations. This technique will be used during modeling.

Analysis of Healing and Damage

Time to analyze these hypotheses.

(2) Healing is not correlated with the game outcome.

(3) Damage is positively correlated with winning.

After adjusting the values and removing the outliers, it's time to see if the amount of team healing/damage is positively correlated to winning.

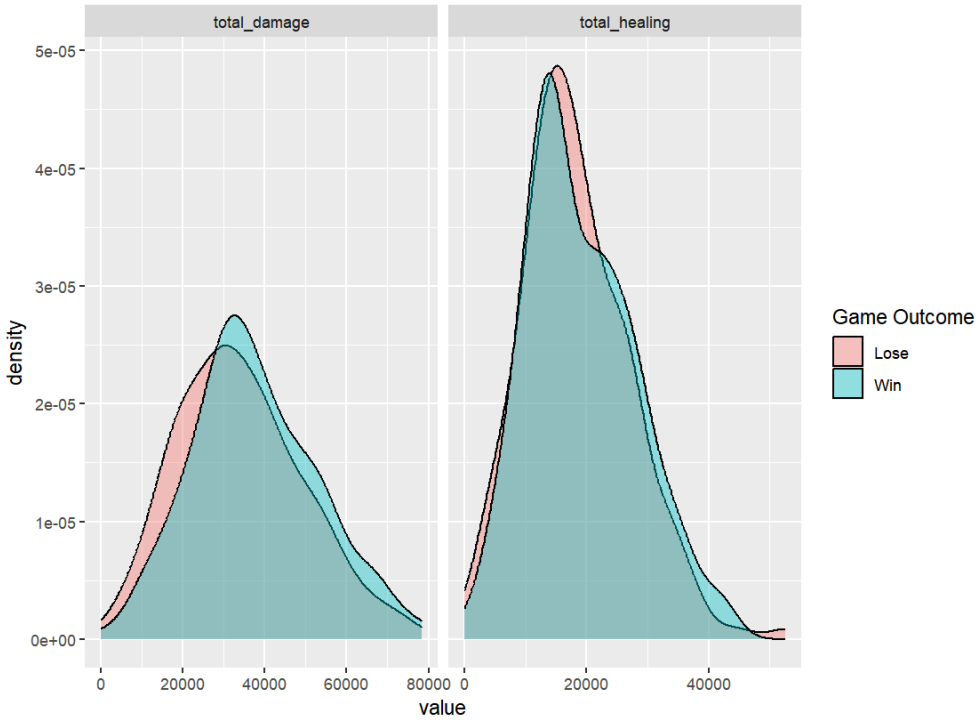


Figure 15: Density plot of total damage and total healing for all teams for each game by game outcome.

The plot above shows that there are only slight differences between the two game outcomes.

Let's perform a statistical test to see the correlation. But first we need to see if the values are normally distributed with p-value = 0.05. To reduce the Type 1 error 0.05 will be divided by the number of normality tests being performed (4) making the **adjusted p-value 0.0125**.

```
# Filter and select data for the ally team
ally_data <- plot_data %>%
  filter(team == 'ally') %>%
  select(total_damage, total_healing, Result)

# Filter and select data for the enemy team
enemy_data <- plot_data %>%
  filter(team == 'enemy') %>%
  select(total_damage, total_healing, Result)

# Perform Shapiro-Wilk tests for the ally team
ally_dmg_test <- shapiro.test(ally_data$total_damage)
ally_heal_test <- shapiro.test(ally_data$total_healing)

# Create a data frame with Shapiro-Wilk results for the ally team
shapiro_results <- data.frame(
  Metric = c("Total Damage", "Total Healing"),
  Team = c("Ally", "Ally"),
  Statistic = c(ally_dmg_test$statistic, ally_heal_test$statistic),
  P_Value = c(ally_dmg_test$p.value, ally_heal_test$p.value)
)
```

```r
# Perform Shapiro-Wilk tests for the enemy team
enemy_dmg_test <- shapiro.test(enemy_data$total_damage)
enemy_heal_test <- shapiro.test(enemy_data$total_healing)

# Append Shapiro-Wilk results for the enemy team
shapiro_results <- shapiro_results %>%
  bind_rows(
    data.frame(
      Metric = c("Total Damage", "Total Healing"),
      Team = c("Enemy", "Enemy"),
      Statistic = c(enemy_dmg_test$statistic, enemy_heal_test$statistic),
      P_Value = c(enemy_dmg_test$p.value, enemy_heal_test$p.value)
    )
  )

# Print the results
shapiro_results <- shapiro_results %>% mutate(reject=P_Value < 0.05/4)
print(shapiro_results)
```

```
##          Metric  Team Statistic     P_Value reject
## 1  Total Damage  Ally 0.9864663 0.095843605  FALSE
## 2 Total Healing  Ally 0.9760597 0.004554882   TRUE
## 3  Total Damage Enemy 0.9875131 0.130972703  FALSE
## 4 Total Healing Enemy 0.9737506 0.002415065   TRUE
```

The table above shows that the `Total Damage` follows a normal distribution, while `Total Healing` does not. Therefore, Pearson's correlation test will be used to assess the relationship between `Damage` and game outcome (`Result`) due to normality. In contrast, Spearman's rank correlation test will be used to assess the relationship between `Total healing` and game outcome, as it is a non-parametric method suitable for non-normal data.

To ensure that data remains independent, metrics from my team and the enemy team were tested separately for correlation with game outcome. Given the multiple tests for correlation, a p-value of $0.025 = 0.05 \ / \ 2$ tests, will be used to see if there are correlations between Healing vs. Outcome and Damage vs. Outcome.

Time to calculate the correlations

```r
cor.test(ally_data$total_damage, ally_data$Result, method='pearson')
```

```
##
##  Pearson's product-moment correlation
##
## data:  ally_data$total_damage and ally_data$Result
## t = 0.61978, df = 170, p-value = 0.5362
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1028843  0.1957249
## sample estimates:
##       cor
## 0.0474811
```

```r
cor.test(enemy_data$total_damage, enemy_data$Result, method='pearson')
```

```
##
##  Pearson's product-moment correlation
##
```

```
## data:  enemy_data$total_damage and enemy_data$Result
## t = 2.5073, df = 170, p-value = 0.01311
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.0403439 0.3291714
## sample estimates:
##       cor
## 0.1888383
```

```
cor.test(ally_data$total_healing, ally_data$Result, method='spearman')
```

```
##
##  Spearman's rank correlation rho
##
## data:  ally_data$total_healing and ally_data$Result
## S = 860174, p-value = 0.8523
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##         rho
## -0.01430117
```

```
cor.test(enemy_data$total_healing, enemy_data$Result, method='spearman')
```

```
##
##  Spearman's rank correlation rho
##
## data:  enemy_data$total_healing and enemy_data$Result
## S = 775477, p-value = 0.2644
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.0855725
```

Given that we are doing two correlations for Healing vs Outcome and two correlations for Damage vs Outcome, the p-value set (based Bonferroni's correction) is 0.025. Out of all four tests, only the correlation between enemy Total Damage and game outcome was statistically significant with a weak positive correlation of 0.1888383 and p-value of 0.01311.

Gamemode

What are the frequency of Game Modes?

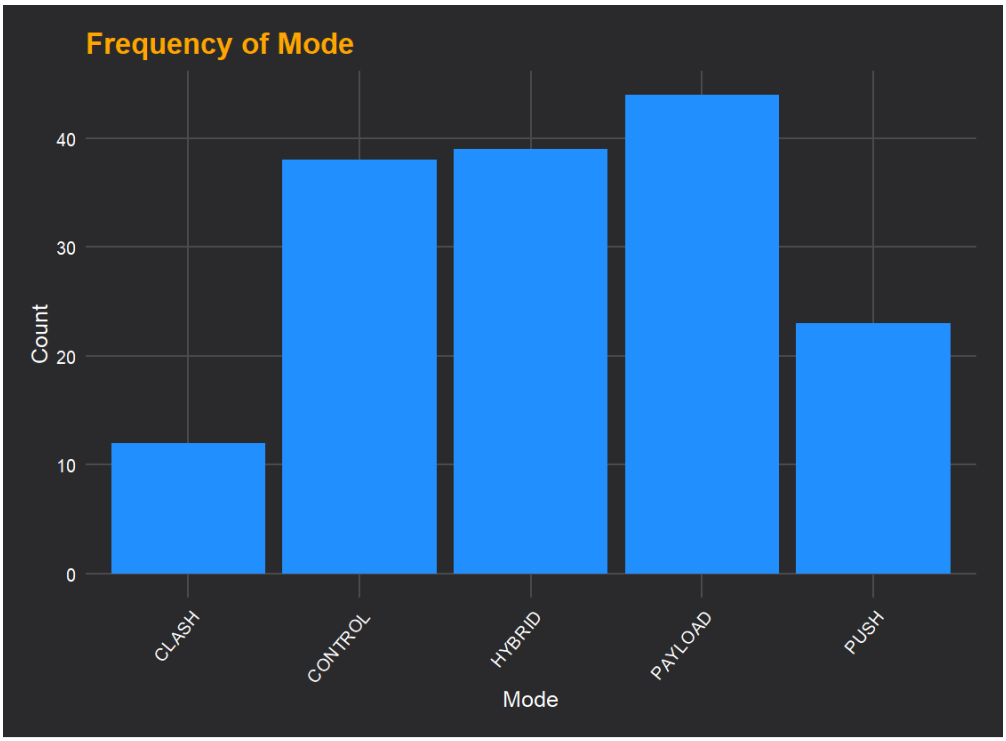


Figure 16: Bar plot of the frequency for each game mode.

There are five times of games modes in the dataset. The most played is Payload and the least played is Clash (the newest game mode).

What are the associated win rates for the game modes? (Continued on next page)

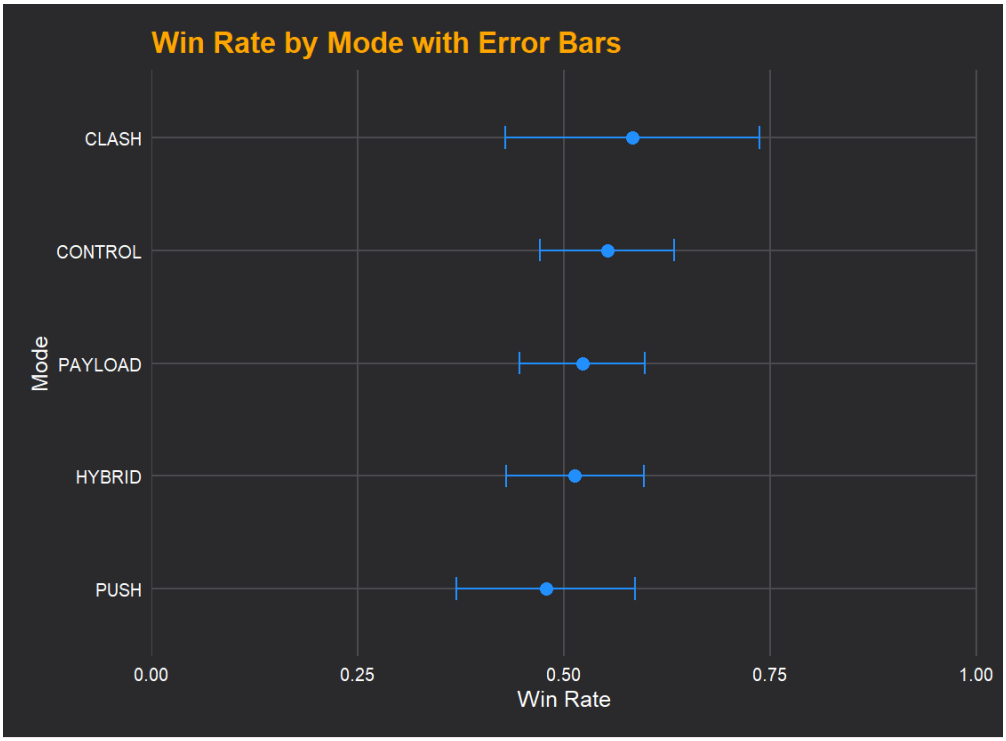


Figure 17: Error plot of win rates of each mode with 95% confidence. There are no clear differences.

```
# Let's do a statistical test to see if they are different from one another
plot_data %>% filter(Result!=0) %>% select(Mode, Result) %>% table
```

```
##          Result
## Mode      -1  1
##   CLASH    5  7
##   CONTROL 17 21
##   HYBRID  18 19
##   PAYLOAD 21 23
##   PUSH    12 11
```

```
test_data <- plot_data %>% filter(Result!=0) %>% select(Mode, Result)

kruskal.test(Mode ~ Result, data=test_data)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Mode by Result
## Kruskal-Wallis chi-squared = 0.39145, df = 1, p-value = 0.5315
```

There doesn't seem to be a statistically significant difference between win rates for Game Modes.

Map

What are the frequency of maps?

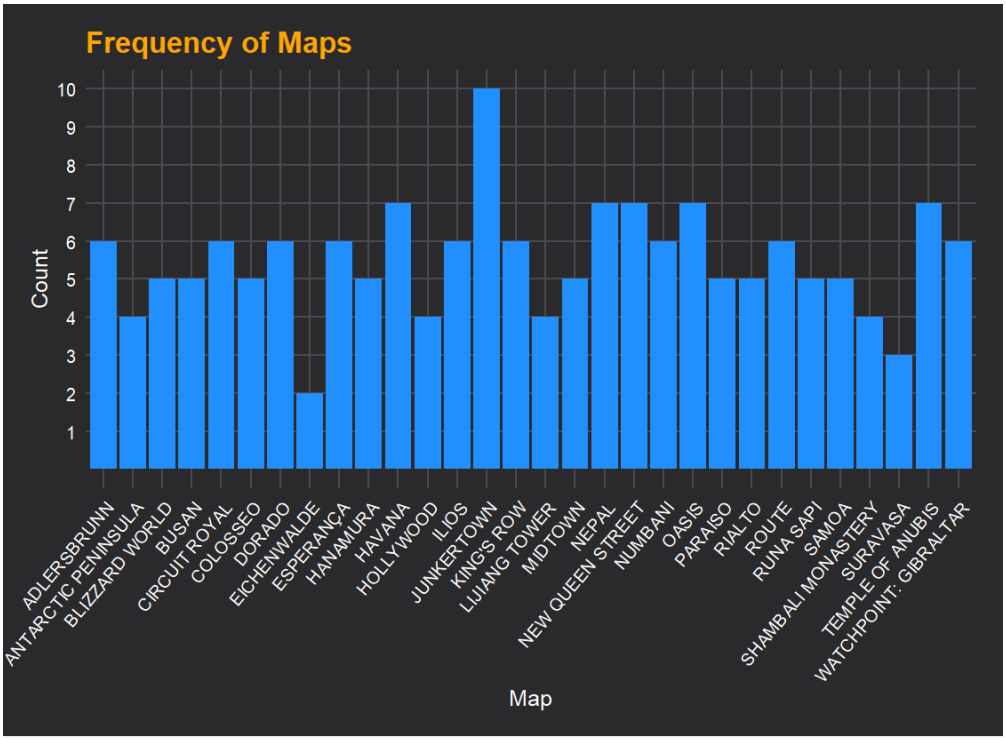


Figure 18: Histogram of the map frequencies. They are all below 30 which means the data should be taken with a grain of salt.

The map selection appears to resemble a uniform distribution. **Junkertown** was the most played with a frequency of 10 games, and **Eichenwalde** is the least played with a frequency of 2 games.

What are the associated win rates for the maps?

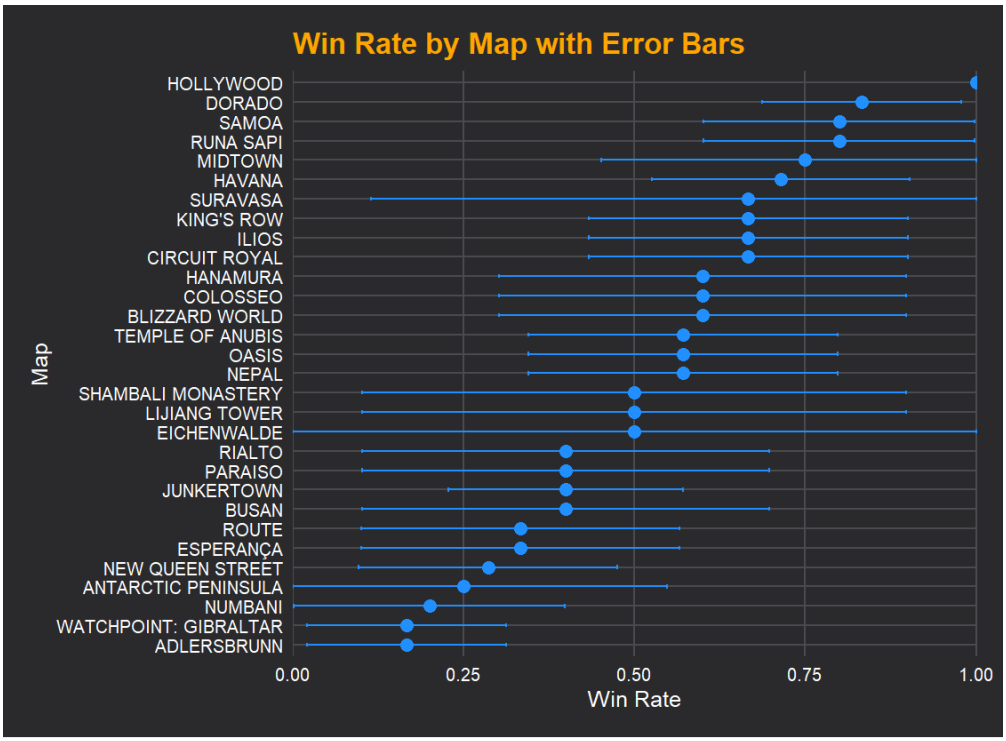


Figure 19: Error bar plot of the win rate for each map with 95% confidence. Hollywood has a 100% win rate while the two maps at the bottoms aren't overlapping with many of the other maps.

It looks like Maps like `Watchpoint: Gilbratar` and `Aldersbrunn` have a lower win rate compared to maps like `Nepal` and `Oasis`, which have higher win rate of 50%+. It should be noted that map sample size is small and may not be fully representative of the population. The maps have 5-6 samples on average.

Modeling

Objective

The goal of modeling is to predict whether a player is going to win based on information available from only the current screenshot of the leaderboard.

Some things to consider are:

- The extreme values for player metrics (error due to OCR)
- The unknown values for Map and Mode (error due to OCR)
- That player placement is static and based on the character being played. For example from a damage character's perspective, they will always be playerID 1. From a support player's perspective, they will always be playerID 3. For this reason, it would be ideal to shuffle playerIDs for the same role on the same team to improve generalization and reduce bias. Especially for support characters. (bias due to leaderboard presentation)

Preprocessing

1. Clean the dataset based on the data exploration by clipping the player metrics and dropping Map and Mode with the value of `Unknown`.

```
clean_dataset <- dataset %>%
  filter(Map != 'UNKNOWN' & Mode != 'Unknown') %>%
  select(GameID, SnapID, PlayerID, K, A, D, Damage, H, MIT, Time, Mode, Map, Result) %>%
  mutate(
    across(c(K, A, D, Damage, H, MIT), ~ .x / Time),  # Normalize metrics by Time
    K = pmin(K, 5),
    A = pmin(A, 4),
    D = pmin(D, 3),
    Damage = pmin(Damage, 2500),
    H = case_when(
      PlayerID %in% c(0, 5) ~ pmin(H, 600),
      PlayerID %in% c(1, 2, 6, 7) ~ pmin(H, 400),
      TRUE ~ pmin(H, 2500)
    ),
    MIT = pmin(MIT, 2500)
  )

# Ensure that the categorical variables are factors
clean_dataset <- clean_dataset %>% mutate(
  Mode=as.factor(Mode),
  Map=as.factor(Map)
)

# adjust the game outcome to be between 0 and 1 (drop the drawn games)
clean_dataset <- clean_dataset %>%
  filter(Result!=0) %>%
  mutate(Result=(Result+1)/2,
         Result=as.factor(Result),
         Result=relevel(Result, ref = "1"))

# show example data from a single snapshot
clean_dataset %>% arrange(SnapID) %>% head(n=10)
```

```
## # A tibble: 10 x 13
##    GameID SnapID PlayerID     K     A     D Damage       H    MIT  Time Mode
##     <int>  <int>    <int> <dbl> <dbl> <dbl>  <dbl>   <dbl>  <dbl> <dbl> <fct>
##  1      3      0        0 0.476 0     0.952   294.   0.952  151.    2.1 PUSH
##  2      3      0        1 0.476 0     1.43    386.   0       12.9   2.1 PUSH
##  3      3      0        2 0.952 0.476 0.476   537. 237.       0     2.1 PUSH
##  4      3      0        3 0.952 0.476 0.476   535. 640        0     2.1 PUSH
##  5      3      0        4 0.952 0.476 0.952   253. 710        0     2.1 PUSH
##  6      3      0        5 1.90  0     0.476   643.   0     1024.    2.1 PUSH
##  7      3      0        6 1.90  0     0       980.   0        0     2.1 PUSH
##  8      3      0        7 2.38  0     0.476   928.   0      186.    2.1 PUSH
##  9      3      0        8 1.43  0.952 0.476   348. 589.       0     2.1 PUSH
## 10      3      0        9 2.86  2.86  0       566. 434.       0     2.1 PUSH
## # i 2 more variables: Map <fct>, Result <fct>
```

2. Group by snapshot id (`snapID`) and then shuffle the playerIDs to improve generalization

```r
set.seed(42) # for reproducibility

swap_ids <- function(ids, index1, index2){
  temp_id <- ids[index1]
  ids[index1] <- ids[index2]
  ids[index2] <- temp_id
  return(ids)
}

# row [tank1, dmg1, dmg1, supp1, supp1, tank2, dmg2, dmg2, supp2, supp2]

shuffle_players <- function(ids) {
  # randomly shuffle damage for team 1?
  if (sample(0:1, 1) == 1) {
    ids <- swap_ids(ids, 2, 3)
  }

  # randomly shuffle damage for team 2?
  if (sample(0:1, 1) == 1) {
    ids <- swap_ids(ids, 7, 8)
  }

  # randomly shuffle support for team 1?
  if (sample(0:1, 1) == 1) {
    ids <- swap_ids(ids, 4, 5)
  }

  # randomly shuffle support for team 2?
  if (sample(0:1, 1) == 1) {
    ids <- swap_ids(ids, 9, 10)
  }

  return (ids)
}

shuffled_clean_dataset <- clean_dataset %>%
  group_by(SnapID) %>%
  group_modify(~ {
    # Shuffle the PlayerID vector for the current group
    .x$PlayerID <- shuffle_players(.x$PlayerID)
    return(.x)
  }) %>%
  ungroup()
```

3. Combine the data from each snapshot into a single observation

```r
shuffled_clean_dataset <- shuffled_clean_dataset %>% group_by(GameID, SnapID) %>%
  pivot_wider(
    names_from = PlayerID,
    values_from = c(K, A, D, Damage, H, MIT),
    names_prefix = "player"
  ) %>% ungroup() %>%
  select(-SnapID, -GameID) # remove unnecessary details
```

```r
head(shuffled_clean_dataset)
```

```
## # A tibble: 6 x 64
##    Time Mode    Map     Result K_player0 K_player1 K_player2 K_player3 K_player4
##   <dbl> <fct>   <fct>   <fct>      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
## 1 2.1   PUSH    ESPERA~ 0          0.476     0.476     0.952     0.952     0.952
## 2 0.217 CONTROL ANTARC~ 0          0         0         0         0         0
## 3 1.18  PAYLOAD SHAMBA~ 0          0.845     1.69      0.845     0         0
## 4 1.37  HYBRID  NUMBANI 0          0.732     2.93      2.20      0.732     2.20
## 5 2.17  PAYLOAD ROUTE   0          1.38      0         0.923     0         0
## 6 0.433 HYBRID  EICHEN~ 0          2.31      2.31      2.31      2.31      0
## # i 55 more variables: K_player5 <dbl>, K_player6 <dbl>, K_player7 <dbl>,
## #   K_player8 <dbl>, K_player9 <dbl>, A_player0 <dbl>, A_player1 <dbl>,
## #   A_player2 <dbl>, A_player3 <dbl>, A_player4 <dbl>, A_player5 <dbl>,
## #   A_player6 <dbl>, A_player7 <dbl>, A_player8 <dbl>, A_player9 <dbl>,
## #   D_player0 <dbl>, D_player1 <dbl>, D_player2 <dbl>, D_player3 <dbl>,
## #   D_player4 <dbl>, D_player5 <dbl>, D_player6 <dbl>, D_player7 <dbl>,
## #   D_player8 <dbl>, D_player9 <dbl>, Damage_player0 <dbl>, ...
```

Training

```r
set.seed(42)
# create the train, test, and validation set
# 80% for train 10% for test and 10% for validation
train_ind <- createDataPartition(shuffled_clean_dataset$Result, p=0.8, list=F)
train_set <- shuffled_clean_dataset[train_ind,]
remaining_set <- shuffled_clean_dataset[-train_ind,]

test_ind <- createDataPartition(remaining_set$Result, p=0.5, list=F)
test_set <- remaining_set[test_ind,]
val_set <- remaining_set[-test_ind,]
```

Base Model (Logistic Regression)

```r
# First find the best CV value using the validation set
set.seed(42)
model = train(Result ~.,
              data=train_set,
              method='glmnet',
              trControl=trainControl(method='cv', number=5),
              preProc = c("center", "scale"))

pred <- predict(model, newdata=val_set)
confusionMatrix(pred, val_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 51 13
##          0 14 56
```

```
##
##                Accuracy : 0.7985
##                  95% CI : (0.7205, 0.8628)
##     No Information Rate : 0.5149
##     P-Value [Acc > NIR] : 9.523e-12
##
##                   Kappa : 0.5965
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.7846
##             Specificity : 0.8116
##          Pos Pred Value : 0.7969
##          Neg Pred Value : 0.8000
##              Prevalence : 0.4851
##          Detection Rate : 0.3806
##    Detection Prevalence : 0.4776
##       Balanced Accuracy : 0.7981
##
##        'Positive' Class : 1
##
```

```
model = train(Result ~.,
              data=train_set,
              method='glmnet',
              trControl=trainControl(method='cv', number=10),
              preProc = c("center", "scale"))

pred <- predict(model, newdata=val_set)
confusionMatrix(pred, val_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 52 13
##          0 13 56
##
##                Accuracy : 0.806
##                  95% CI : (0.7288, 0.8692)
##     No Information Rate : 0.5149
##     P-Value [Acc > NIR] : 2.487e-12
##
##                   Kappa : 0.6116
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8000
##             Specificity : 0.8116
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.8116
##              Prevalence : 0.4851
##          Detection Rate : 0.3881
##    Detection Prevalence : 0.4851
##       Balanced Accuracy : 0.8058
```

```
##
##        'Positive' Class : 1
##
```

Based on the different values, a CV value of 10 appears to perform better on the dataset. Now, let's see how it performs on the test_set.

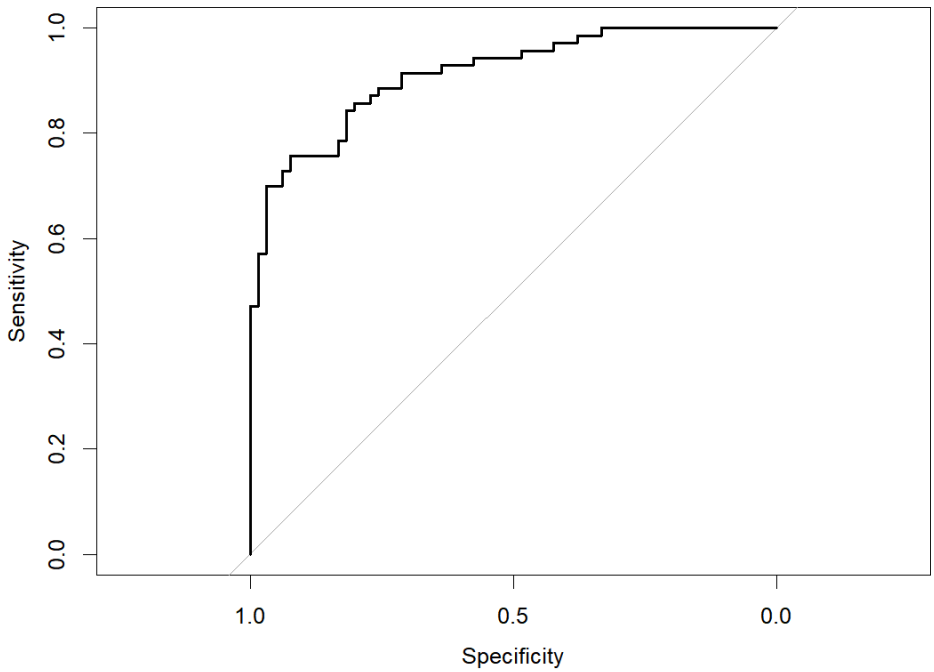
```
# train the model to see the base results
base_model = train(Result ~.,
                   data=train_set,
                   method='glmnet',
                   trControl=trainControl(method='cv', number=10),
                   preProc = c("center", "scale"))

pred <- predict(base_model, newdata=test_set)
confusionMatrix(pred, test_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 49  8
##          0 17 62
##
##                Accuracy : 0.8162
##                  95% CI : (0.7407, 0.8774)
##     No Information Rate : 0.5147
##     P-Value [Acc > NIR] : 2.372e-13
##
##                   Kappa : 0.6306
##
##  Mcnemar's Test P-Value : 0.1096
##
##             Sensitivity : 0.7424
##             Specificity : 0.8857
##          Pos Pred Value : 0.8596
##          Neg Pred Value : 0.7848
##              Prevalence : 0.4853
##          Detection Rate : 0.3603
##    Detection Prevalence : 0.4191
##       Balanced Accuracy : 0.8141
##
##        'Positive' Class : 1
##
```

```
pred_prob <- predict(base_model, newdata = test_set, type = "prob")[,1]
roc_obj <- roc(test_set$Result, pred_prob)

# Plot ROC curve
plot(roc_obj)
```




```
print(auc(roc_obj))
```

```
## Area under the curve: 0.9126
```

Model 2 (with regularization finetune)

Now, I'm going to finetune the lambda and alpha values using the validation set to try to get a better score.

```
# train the model to see the base results
set.seed(1)
model_2 = train(Result ~.,
                data=train_set,
                method='glmnet',
                preProc = c("center", "scale"),
                trControl=trainControl(method='cv', number=10),
                 tuneGrid=expand.grid(alpha = seq(0, 1, length=10),
                 lambda = seq(0.0001, 1, length = 100))
               )
pred <- predict(model_2, newdata=val_set)
print(model_2$bestTune)
```

```
##    alpha lambda
## 11     0 0.1011
```

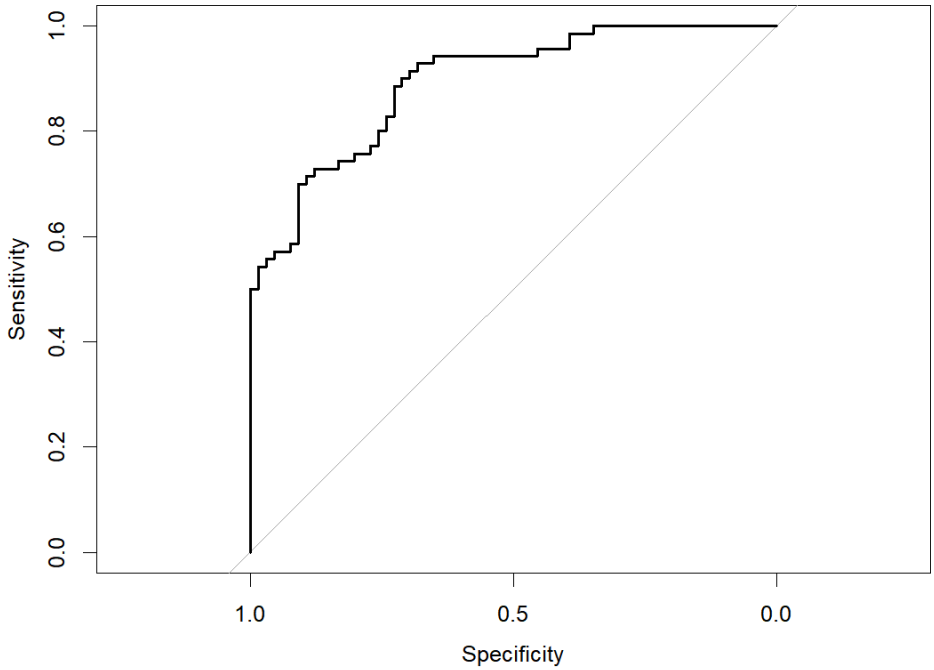
```
# train the model with the fine-tuned reg
set.seed(42)
model_2 = train(Result ~.,
                 data=train_set,
                 method='glmnet',
                 trControl=trainControl(method='cv', number=10),
                 tuneGrid=data.frame(alpha=0, lambda=.1011),
                 preProc = c("center", "scale"))
```

```r
pred <- predict(model_2, newdata = test_set)
confusionMatrix(pred, test_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 47  8
##          0 19 62
##
##                Accuracy : 0.8015
##                  95% CI : (0.7245, 0.8649)
##     No Information Rate : 0.5147
##     P-Value [Acc > NIR] : 3.781e-12
##
##                   Kappa : 0.6007
##
##  Mcnemar's Test P-Value : 0.05429
##
##             Sensitivity : 0.7121
##             Specificity : 0.8857
##          Pos Pred Value : 0.8545
##          Neg Pred Value : 0.7654
##              Prevalence : 0.4853
##          Detection Rate : 0.3456
##    Detection Prevalence : 0.4044
##       Balanced Accuracy : 0.7989
##
##        'Positive' Class : 1
##
```

```r
pred_prob <- predict(model_2, newdata = test_set, type = "prob")[,1]
roc_obj <- roc(test_set$Result, pred_prob)

# Plot ROC curve
plot(roc_obj)
```




```
print(auc(roc_obj))
```

```
## Area under the curve: 0.8929
```

Base Model (RandomForest)

```
set.seed(42)
control <- trainControl(method='cv', number=10, search = 'grid')

rf_model <- train(Result ~ .,
                  data = train_set,
                  method = "rf",
                  metric = "Accuracy",
                  trControl = control,
                  importance = TRUE)

# View the model details
print(rf_model)
```

```
## Random Forest
##
## 1083 samples
##   63 predictor
##    2 classes: '1', '0'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 975, 974, 975, 974, 975, 976, ...
## Resampling results across tuning parameters:
##
```

```
##   mtry  Accuracy   Kappa
##    2    0.9122480  0.8244792
##   47    0.9141341  0.8280981
##   93    0.9141510  0.8281826
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 93.
```

```
pred <- predict(rf_model, newdata = test_set)
confusionMatrix(pred, test_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 62  1
##          0  4 69
##
##                Accuracy : 0.9632
##                  95% CI : (0.9163, 0.988)
##     No Information Rate : 0.5147
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9263
##
##  Mcnemar's Test P-Value : 0.3711
##
##             Sensitivity : 0.9394
##             Specificity : 0.9857
##          Pos Pred Value : 0.9841
##          Neg Pred Value : 0.9452
##              Prevalence : 0.4853
##          Detection Rate : 0.4559
##    Detection Prevalence : 0.4632
##       Balanced Accuracy : 0.9626
##
##        'Positive' Class : 1
##
```

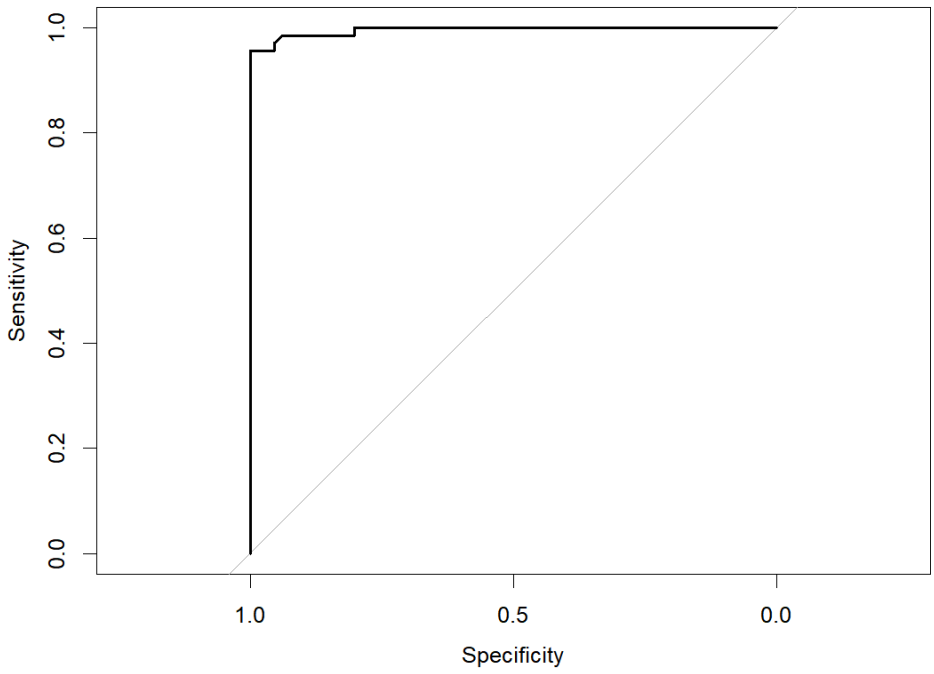
```
# test on validation
pred <- predict(rf_model, newdata = val_set)
confusionMatrix(pred, val_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 63  4
##          0  2 65
##
##                Accuracy : 0.9552
##                  95% CI : (0.9051, 0.9834)
##     No Information Rate : 0.5149
##     P-Value [Acc > NIR] : <2e-16
##
```

```
##                  Kappa : 0.9104
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9692
##             Specificity : 0.9420
##          Pos Pred Value : 0.9403
##          Neg Pred Value : 0.9701
##              Prevalence : 0.4851
##          Detection Rate : 0.4701
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.9556
##
##        'Positive' Class : 1
##
```

```
pred_prob <- predict(rf_model, newdata = test_set, type = "prob")[,1]
roc_obj <- roc(test_set$Result, pred_prob)

# Plot ROC curve
plot(roc_obj)
```




```
print(auc(roc_obj))
```

```
## Area under the curve: 0.9958
```

Model 2 (hyperparameter finetune)

```
set.seed(42)

# Define repeated cross-validation control with random search
```

```r
control <- trainControl(method = "repeatedcv",
                        number = 5,
                        repeats = 3,
                        search = "random")

# Random search will randomly pick `tuneLength` combinations
tune_length <- 5  # Number of random combinations to test

# Train the Random Forest model with random search
rf_tuned <- train(
  Result ~ .,
  data = train_set,
  method = "rf",
  metric = "Accuracy",
  trControl = control,
  tuneLength = tune_length,
  importance = TRUE
)

# View the tuned model
print(rf_tuned)
```

```
## Random Forest
##
## 1083 samples
##   63 predictor
##    2 classes: '1', '0'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 867, 867, 866, 866, 866, 866, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   18    0.9039569  0.8076910
##   25    0.9101070  0.8200089
##   49    0.9085709  0.8169766
##   65    0.9079507  0.8157244
##   74    0.9073363  0.8145016
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 25.
```

```r
pred <- predict(rf_tuned, newdata = test_set)
confusionMatrix(pred, test_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 64  3
##          0  2 67
##
##                Accuracy : 0.9632
```

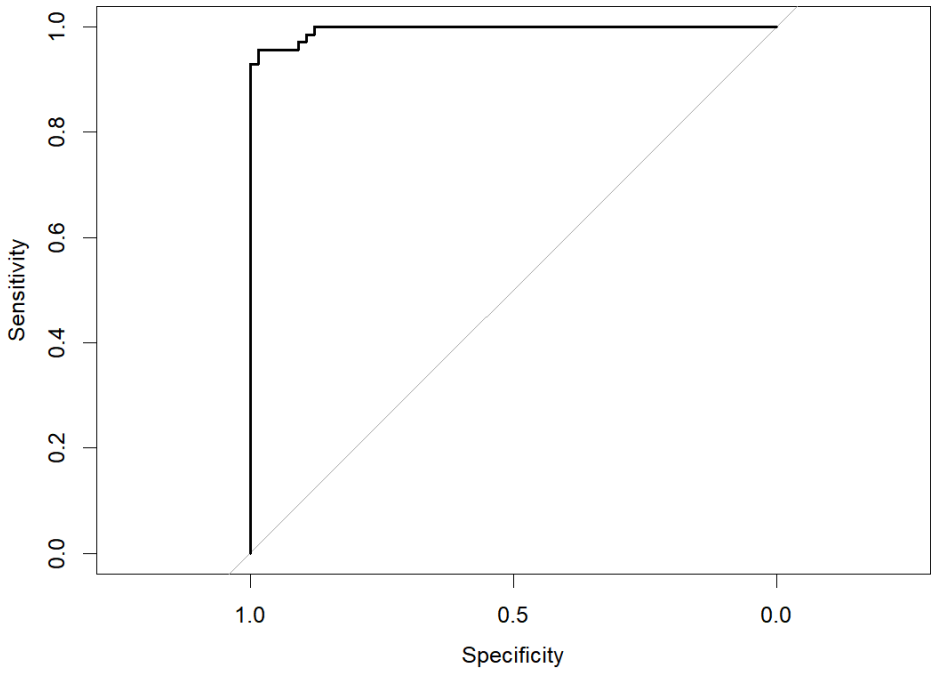
```
##                  95% CI : (0.9163, 0.988)
##     No Information Rate : 0.5147
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9264
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9697
##             Specificity : 0.9571
##          Pos Pred Value : 0.9552
##          Neg Pred Value : 0.9710
##              Prevalence : 0.4853
##          Detection Rate : 0.4706
##    Detection Prevalence : 0.4926
##       Balanced Accuracy : 0.9634
##
##        'Positive' Class : 1
##
```

```
# test on validation
pred <- predict(rf_tuned, newdata = val_set)
confusionMatrix(pred, val_set$Result)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  0
##          1 63  2
##          0  2 67
##
##                Accuracy : 0.9701
##                  95% CI : (0.9253, 0.9918)
##     No Information Rate : 0.5149
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9402
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9692
##             Specificity : 0.9710
##          Pos Pred Value : 0.9692
##          Neg Pred Value : 0.9710
##              Prevalence : 0.4851
##          Detection Rate : 0.4701
##    Detection Prevalence : 0.4851
##       Balanced Accuracy : 0.9701
##
##        'Positive' Class : 1
##
```

```
pred_prob <- predict(rf_tuned, newdata = test_set, type = "prob")[,1]
roc_obj <- roc(test_set$Result, pred_prob)
```

```
# Plot ROC curve
plot(roc_obj)
```




```
print(auc(roc_obj))
```

```
## Area under the curve: 0.995
```

Results and Discussion

Model Performance

Model Name	Accuracy*	F1-Score	AUC
Base LR	0.8162	0.7967	0.9126
Tuned LR	0.7989	0.7768	0.8929
Base RF	0.9626	0.9612	0.9958
Tuned RF	0.9634	0.9624	0.9950

Table 1: Table of the test model performances. *Accuracy is balanced

The data was split into three sets: the train set, validation set, and test set (final holdout). The validation set was used for hyperparameter tuning of the model. Table 1, above, shows that the hyperparameter-tuned model for Logistic Regression performed poorly compared to the base model without tuning. This could likely be due to hypercollinearity in the training set or lack of generalization from the validation set. The best performing models were the RF models with the hyperparameter-tuned RF achieving a balanced accuracy of 96.34% and AUC of 0.995. This is a 14.72% increase in Accuracy and 16.57% increase in F1-score, compared to the best performing LR model.

Insights from Results

Hypothesis Results

(1) The number of swaps done by the Tank role is positively correlated with winning.

There was a weak negative correlation between tank swap frequency and game outcome (Kendall's tau = -0.1183, $p < 0.001$), suggesting that swapping more as tank signals a losing game outcome. Figure 10, shows that losing games do have an increase in swaps with a median of 2, compared to a median of 1 for winning games. Given the results, we can reject the hypothesis that tank swapping is positively associated with winning.

(2) Healing is not correlated with the game outcome.

Very weak correlations were found, however they were not statistically significant. Healing was found to have a correlation of -0.14 for ally total healing vs game outcome (p=0.8523), and a correlation of 0.086 for enemy total healing vs game outcome (p=0.2644). This shows that healing isn't a good predictor for game performance.

(3) Damage is positively correlated with the winning.

The results shows that, interestingly, only enemy total damage was found to have a statistically significant correlation. Enemy total damage has a weak positive correlation with their likelihood of winning ($r = 0.189$, p=0.01311). The 95% confidence interval for the correlation coefficient ranges from 0.04 to 0.329. Ally total damage, on the other hand, had a Pearson's r coefficient of 0.0475 and p-value of 0.5362.

Feature Importances

Below are the top 20 feature importances of both the Base RF model and the Hyperparameter-Tuned RF Model. What is interesting is that the Tuned RF Model has more features that are given high importance, compared to the base results. For example, there are 14 features in the right plot that have an importance of 70 or more. On the left, there are only 3 features with an importance above 70. **Given that the top 20 features in the best performing model are only player metrics, the following analysis will focus on those**.

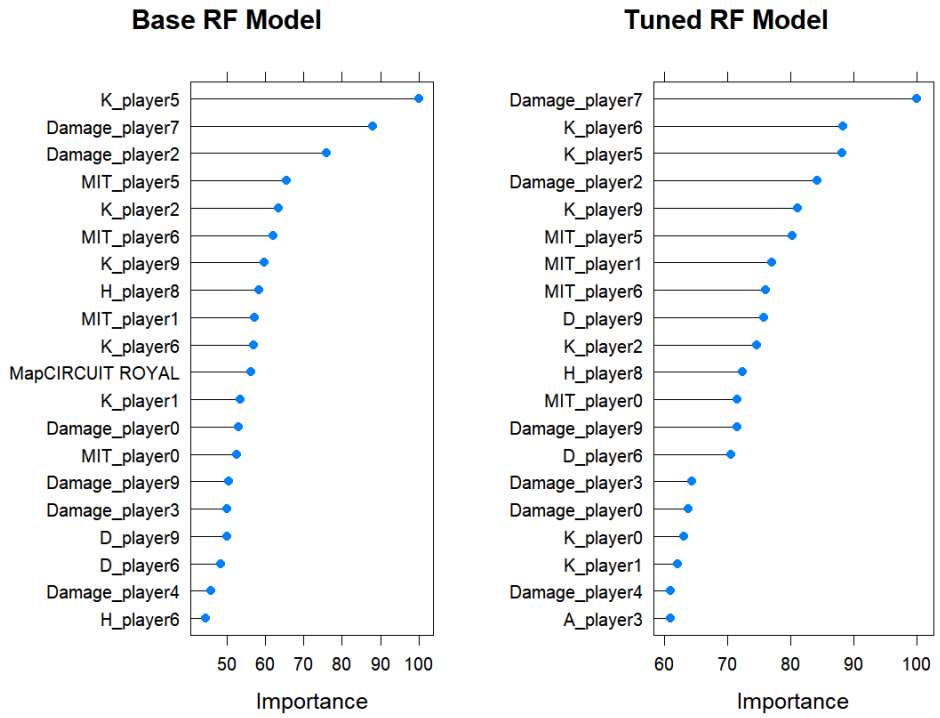


Figure 20: Feature importances of both RF models.

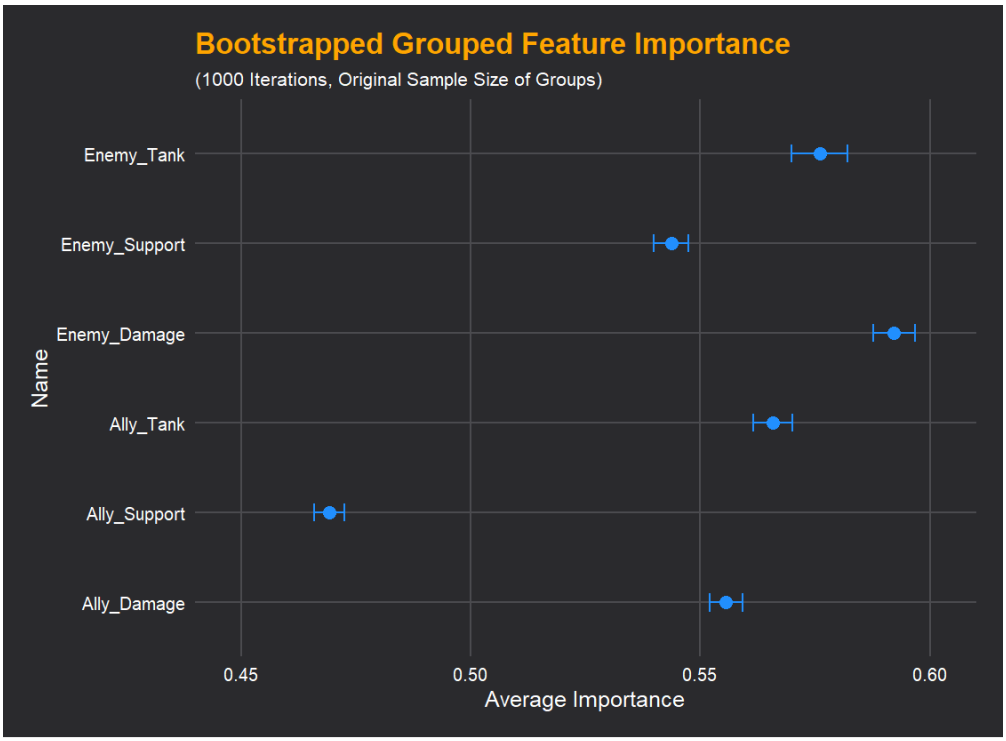


Figure 21: Error plot of the average confidence for each group combination of Team and Role. A 99% confidence interval is used.

The feature importances were grouped by team and player role and a bootstrapped average importance was calculated for all six groups. The plot shows that damage player metrics are the most important. I believe that this is because of two reasons: (1) enemy damage is weakly correlated with game outcome, and (2) regularization caused the model to favor Damage and Kills of the enemy team due to them being a decent aggregate of total players killed on my team.

Lastly, the confidence intervals for the tank role are overlapping and are of high importance, compared to the other groups. This underscores the importance of the tank role. Of the top 20 features for Tuned RF, the tank metrics included were damage mitigation (`MIT`) and kills (`K`).

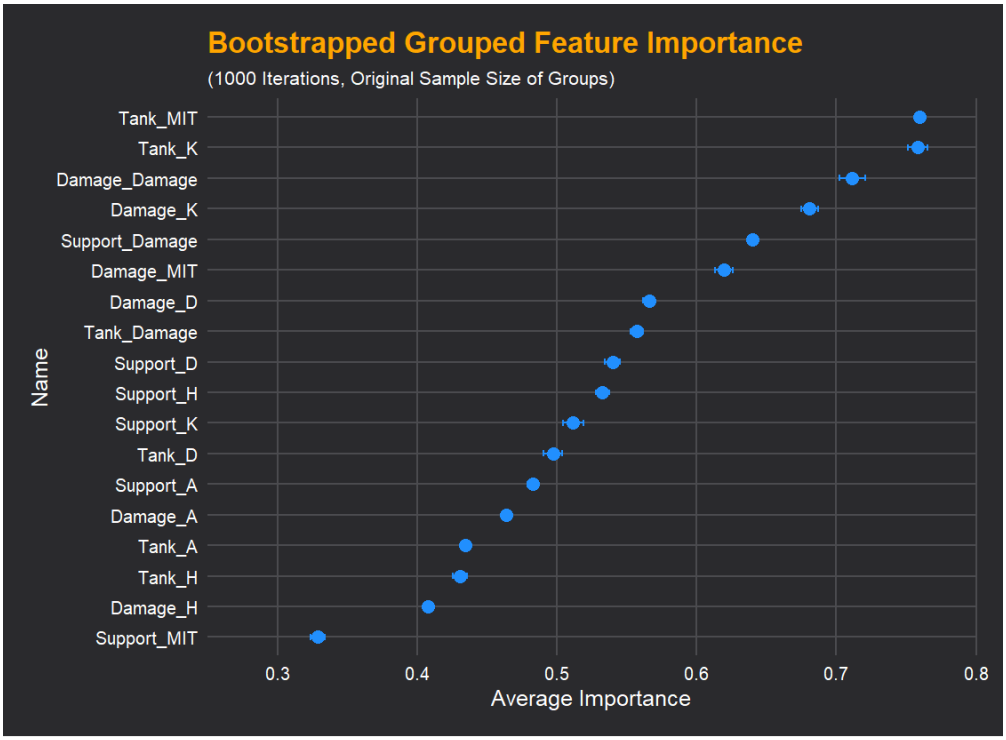


Figure 22: Error plot of the average confidence for each group combination of Player Metric (i.e., K, A, D, Damage, H, MIT) and Role. A 99% confidence interval is used.

Grouping the feature importance by `role_playermetric` it can be seen that MIT and K for tanks are, in fact, the most important player metric in predicting game outcome. This is followed by Damage done by damage players and Damage done by support players. Interestingly, although tanks are important, their deaths aren't as important. Perhaps, this is due to the fact that revives occur consistently which can keep the momentum of a winning game.

```
plot_data
```

```
## # A tibble: 18 x 4
##    name            mean     sd      me
##    <chr>          <dbl>  <dbl>   <dbl>
##  1 Damage_A       0.464 0.0190 0.00155
##  2 Damage_D       0.566 0.0491 0.00400
##  3 Damage_Damage  0.712 0.111  0.00906
##  4 Damage_H       0.408 0.0281 0.00229
##  5 Damage_K       0.681 0.0755 0.00616
```

```
##  6 Damage_MIT     0.620 0.0773 0.00631
##  7 Support_A      0.483 0.0435 0.00355
##  8 Support_D      0.540 0.0661 0.00540
##  9 Support_Damage 0.640 0.0218 0.00178
## 10 Support_H      0.533 0.0581 0.00474
## 11 Support_K      0.512 0.0896 0.00732
## 12 Support_MIT    0.329 0.0627 0.00512
## 13 Tank_A         0.434 0.0223 0.00182
## 14 Tank_D         0.497 0.0798 0.00651
## 15 Tank_Damage    0.557 0.0547 0.00447
## 16 Tank_H         0.430 0.0619 0.00505
## 17 Tank_K         0.758 0.0867 0.00707
## 18 Tank_MIT       0.760 0.0314 0.00256
```

Conclusion

Key Takeaways

Tanks are Important.

- It was found that tanks (on both teams) are an important role in the game. They are the only groups that have overlapping importance intervals. Their damage mitigation and kills are key predictors in game outcome.
- The number of tank swaps is weakly negatively correlated with winning. Given that there is also a weak correlation between ally and enemy swaps, this shows that a team that is winning is more likely to stick with what is working, and that when a tank swaps, it is signalling a change in confidence.

Considering the two points above, tanks have the important role of keeping the game balanced by reducing the damage that their teammates receive, and by increasing the number of kills.

Interestingly, tank deaths are not that important. It should be noted that, like with enemy damage, model regularization might be the reason why MIT was selected, as it can also explain the variance in deaths and damage received.

Support characters can shift the balance of the game.

- Total healing is not correlated with game outcome.
- While the ally total damage isn't statistically significantly correlated with game outcome, the enemy total damage is.
- Viewing bootstrapped average importance of each group of Team/Role, the enemy support group is much more important than ally support group.

Considering the three points above about healing and damage, it shows that damage received is much more important than the amount of healing received. It is plausible to say that, during a game, healing might equally counter damage resulting in a net zero gain, in terms of overall impact on game outcome.

It could be thought that if tanks are balancing each other out and not swapping, and enemy damage is countered by ally healing, then enemy healers can shift the game based on their actions. This is supported by the fact that Kills, Deaths, Heals, and Damage of the enemy support group are among the top 13 most important features (show in plot 20).

Potential Impact

The analysis shows that tanks should work on reducing team damage received by increasing MIT, and that damage players should work on increasing damage done. The analysis also shows that if a game is stagnant or tied, then support character damage can be what tilts the balance of the game. Lastly, this research shows

that healing isn't as important as the damage dealt. It also shows that damage deaths and support deaths are more important compared to tank deaths.

Known Limitations

- Some known limitations are that the OCR software used is estimated to be ~97% accurate and has trouble with registering 0's and 11's.
- The dataset is likely too small to provide enough statistical power to detect significant correlations.
- There are only about 174 games with the majority of map counts being below 7.
- The data is also biased towards games from a support character. It could be likely that the deaths aren't as important due to revives occurring frequency during games.

Future Work

In future research, more advanced methods like oversampling could be used to account for small sample size, and under-sampling could be used for biased characters,

The lack of performance of the logistic regression model when regularized could hint to issues with collinearity. Future models with more data could consider selecting features based on this study to account for the dependencies between ally and enemy players stats.

Lastly, given the insights from the data, more research can be done to further analyze interactions between healing done and enemy damage dealt. Additionally, interactions between kills and MIT for the tank could be analyzed to possibly determine an ideal or target ratio for high performance.

References

Blizzard Entertainment. (n.d.). *Overwatch 2 Heroes*. Retrieved November 13, 2024, from https://overwatch.blizzard.com/en-us/heroes/

Overbuff. (n.d.). *Overwatch Hero Stats*. Retrieved November 13, 2024, from https://www.overbuff.com/heroes?platform=pc&timeWindow=year

Fun Fact Corner

Fun fact: the seed used matches the number of pages!