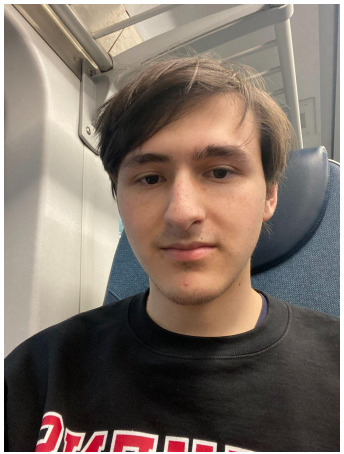
# CS 513-B Final Project



**By:** Matthew Angelakos(20005936) Tri Luu(20007629)
**Section:** B
**Group:** 22
**Project Name:** Super Smash Brothers Melee Win Loss Classification

# What is our project?

- We got our dataset at https://www.kaggle.com/datasets/thomasdubail/super-smash-bros-database/
- This dataset is a collection of sets for the eSport Super Smash Brothers Melee from the years of 2018 to 2023. This game is a 1v1 fighting game where you choose from 25 characters and pick from 6 stages and attempt to beat your opponent.
- This is the original form of the dataset containing over 1156635 data points!

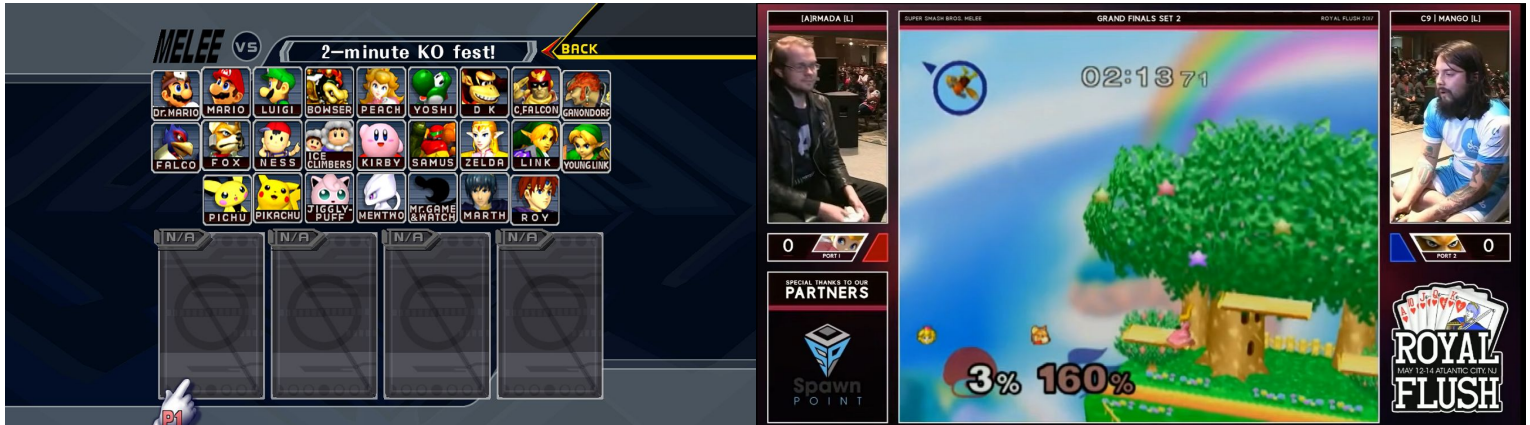
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| index | key | game | tourname | winner_ic | p1_id | p2_id | p1_score | p2_score | location_r | bracket_n | bracket_o | set_order | best_of | game_data |
| 0 | 1.05E+08 | melee | mdva-invi | 5620 | 5620 | Chillin | 3 | 1 | ["R1", "Round 1", "R | | 1 | A | 5 | [] |
| 1 | 1.05E+08 | melee | mdva-invi | Aglet | 15634 | Aglet | 2 | 3 | ["R1", "Round 1", "R | | 1 | B | 5 | [] |
| 2 | 1.05E+08 | melee | mdva-invi | 6126 | 6126 | 1097 | 3 | 0 | ["R1", "Round 1", "R | | 1 | C | 5 | [] |
| 3 | 1.05E+08 | melee | mdva-invi | 1069 | Chu | 1069 | 0 | 3 | ["R1", "Round 1", "R | | 1 | D | 5 | [] |
| 4 | 1.05E+08 | melee | mdva-invi | Rishi | Jerry | Rishi | 1 | 3 | ["R1", "Round 1", "R | | 1 | E | 5 | [] |
| 5 | 1.05E+08 | melee | mdva-invi | Rishi | Rishi | 15634 | 3 | 2 | ["R2", "Round 2", "R | | 1 | F | 5 | [] |
| 6 | 1.05E+08 | melee | mdva-invi | 5620 | 1069 | 5620 | 0 | 3 | ["R2", "Round 2", "R | | 1 | G | 5 | [] |
| 7 | 1.05E+08 | melee | mdva-invi | 1097 | 1097 | Chu | 3 | 1 | ["R2", "Round 2", "R | | 1 | H | 5 | [] |
| 8 | 1.05E+08 | melee | mdva-invi | 6126 | Aglet | 6126 | 1 | 3 | ["R2", "Round 2", "R | | 1 | I | 5 | [] |
| 9 | 1.05E+08 | melee | mdva-invi | Jerry | Chillin | Jerry | 0 | 3 | ["R2", "Round 2", "R | | 1 | J | 5 | [] |
| 10 | 1.05E+08 | melee | mdva-invi | 5620 | 5620 | 1097 | 3 | 2 | ["R3", "Round 3", "R | | 1 | K | 5 | [] |
| 11 | 1.05E+08 | melee | mdva-invi | Rishi | 6126 | Rishi | 2 | 3 | ["R3", "Round 3", "R | | 1 | L | 5 | [] |
| 12 | 1.05E+08 | melee | mdva-invi | Chu | Chu | Aglet | 3 | 1 | ["R3", "Round 3", "R | | 1 | M | 5 | [] |
| 13 | 1.05E+08 | melee | mdva-invi | Jerry | Jerry | 15634 | 3 | 2 | ["R3", "Round 3", "R | | 1 | N | 5 | [] |
| 14 | 1.05E+08 | melee | mdva-invi | 1069 | Chillin | 1069 | 2 | 3 | ["R3", "Round 3", "R | | 1 | O | 5 | [] |
| 15 | 1.05E+08 | melee | mdva-invi | 5620 | Aglet | 5620 | 0 | 3 | ["R4", "Round 4", "R | | 1 | P | 5 | [] |

# What do we wish to accomplish?

- The overall goal of over project is to develop a binary classification model to predict whether player(p1 or player 1) won or lost against their opponent in a given match

- To accomplish this we wish to feed our model all possible pre-game attributes with all values of our dataset in order to develop this model

- To do this however we must first clean our data to be in a useable form to run models on



# Expansion of the game_data column

[{"winner_id": 288832, "loser_id": 1409499, "winner_score": null, "loser_score": null, "winner_char": "melee/fox", "loser_char": "melee/captainfalcon", "stage": "Yoshi's Story"}, {"winner_id": 288832, "loser_id": 1409499, "winner_score": null, "loser_score": null, "winner_char": "melee/fox", "loser_char": "melee/captainfalcon", "stage": "Battlefield"}]

- Each of the {} indicates a different match and the attributes in each match that we can extract as attributes

# Data Cleaning Part 1

- We first need to expand the ‘game_data’ column so that we can get all the attributes of each individual match.
- We will decide to remove all rows where there is no game data as the data there is categorical there is no way to get an average or something of the sort in regards to characters or stages.
- Also there would be no way to tell who won any of the matches

```
# Read the CSV file into a DataFrame
df = pd.read_csv(csv_file_path)
df=df[df['game_data'] != '[]']
df = df.drop(columns=['winner_id'])
df['game_data'] = df['game_data'].apply(json.loads)
# Explode the 'game_data' column and reset the index
df_exploded = df.explode('game_data').reset_index(drop=True)

# Use json_normalize to flatten the 'game_data' column
df_normalized = pd.json_normalize(df_exploded['game_data'])

# Concatenate the normalized DataFrame with the original DataFrame
df = pd.concat([df_exploded, df_normalized], axis=1)
```

# Data Cleaning Part 2

- We then need to remove certain attributes that are not potential predictors for the outcome of the match
- These include:
  - ‘index’: This is not an actual attribute but it's just the index of where it is the database which is not actually part of the matches at all
  - ‘game’: We are only examining Melee so this is irrelevant
  - ‘game_data’: We are gonna expand the data in here eventually so this will be need to be removed
  - ‘key’: Same reason as index
  - ‘winner_score’, ‘loser_score’: These are post game attributes so they cannot be a factor in predicting the outcome
  - ‘winner_id’: This is the winner of the whole set and not an individual match so it is not applicable here

```
# Drop the original 'game_data' column
df = df.drop(columns=['game_data', 'index', 'key', 'game', 'p1_score', 'p2_score', 'winner_score', 'loser_score'])
```

- Next we standardize the naming conventions for the characters, the location name, and also decide to add a standard ‘bracket_name’ to just be the default ‘Bracket’ and we do not have to throw away all data without it

```
# Drop the original columns
df['location_names'] = df['location_names'].apply(lambda x: x.split(', ')[1] if isinstance(x, str) else x)
df['bracket_name'] = df['bracket_name'].fillna('Bracket')
df['winner_char'] = df['winner_char'].str.replace('melee/', '')
df['loser_char'] = df['loser_char'].str.replace('melee/', '')
```

# Data Cleaning Part 3

- We must now alter the dataframe in order to indicate if player 1 won the match
- We do this by locating where the player 1 id is equal to the winner id, if so in the new column ‘win_status’ will be equal to ‘win’ and if not then it is ‘lose’

```
#set default values for the new columns
df['win_status'] = 'win'
df['p1_char'] = ' '
df['p2_char'] = ' '
#remove na values in order to transform data to the proper type
df.dropna(subset=['p1_id'], inplace=True)
df.dropna(subset=['loser_id'], inplace=True)
df.dropna(subset=['winner_id'], inplace=True)
#change columns to the proper data type
df['p1_id'] = df['p1_id'].astype(int)
df['p2_id'] = df['p2_id'].astype(int)
df['bracket_order'] = df['bracket_order'].astype(int)
df['best_of'] = df['best_of'].astype(int)
df['loser_id'] = df['loser_id'].astype(int)
df['winner_id'] = df['winner_id'].astype(int)
```

# Data Cleaning Part 4

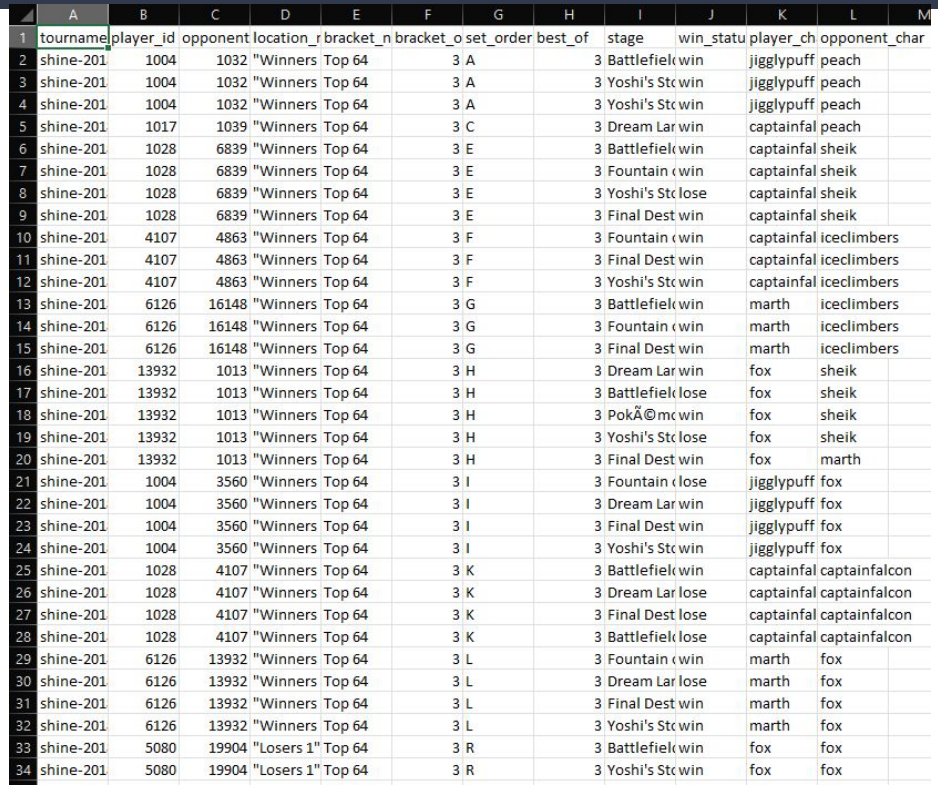
```
#we create a mask to find all the locations where the loser is player 1
#at these locations we set the win_status to lose as well as making the player 1 and 2 character's the correct one
mask = (df['loser_id'] == df['p1_id'])
df.loc[mask, 'win_status'] = 'lose'
df.loc[mask, 'p1_char'] = df.loc[mask, 'loser_char']
df.loc[mask, 'p2_char'] = df.loc[mask, 'winner_char']
#do the same for when player 1 wins
mask2 = (df['winner_id'] == df['p1_id'])
df.loc[mask2, 'win_status'] = 'win'
df.loc[mask2, 'p1_char'] = df.loc[mask2, 'winner_char']
df.loc[mask2, 'p2_char'] = df.loc[mask2, 'loser_char']
#remove the old columns
df = df.drop(columns=['winner_char', 'loser_char', 'winner_id', 'loser_id'])
#update the naming of the columns for readability
df.columns = ['tournament_key', 'player_id', 'opponent_id', 'location_names', 'bracket_name', 'bracket_order', 'set_order', 'best_of', 'stage', 'win_status', 'player_char', 'opponent_char']
```

# Data Cleaning Part 5

- If any NAN values exist now we now wish to remove them
- At this point the 1st cleaned dataset can be saved

```
df.dropna(subset=['stage'], inplace=True)
df.dropna(subset=['player_char'], inplace=True)
df.dropna(subset=['opponent_char'], inplace=True)
df.dropna(subset=['tournament_key'], inplace=True)
df.dropna(subset=['location_names'], inplace=True)
df.dropna(subset=['bracket_name'], inplace=True)
df.dropna(subset=['bracket_order'], inplace=True)
df.dropna(subset=['set_order'], inplace=True)
df.dropna(subset=['best_of'], inplace=True)
df.dropna(subset=['player_id'], inplace=True)
df.dropna(subset=['opponent_id'], inplace=True)
#print(df)
df.to_csv(csv_file_path_clean, index=False)
```

# 1st Cleaned Data Set



| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | tourname | player_id | opponent | location_r | bracket_n | bracket_o | set_order | best_of | stage | win_statu | player_ch | opponent_char | |
| 2 | shine-201 | 1004 | 1032 | "Winners | Top 64 | 3 | A | 3 | Battlefiel | win | jigglypuff | peach | |
| 3 | shine-201 | 1004 | 1032 | "Winners | Top 64 | 3 | A | 3 | Yoshi's St | win | jigglypuff | peach | |
| 4 | shine-201 | 1004 | 1032 | "Winners | Top 64 | 3 | A | 3 | Yoshi's St | win | jigglypuff | peach | |
| 5 | shine-201 | 1017 | 1039 | "Winners | Top 64 | 3 | C | 3 | Dream Lar | win | captainfal | peach | |
| 6 | shine-201 | 1028 | 6839 | "Winners | Top 64 | 3 | E | 3 | Battlefiel | win | captainfal | sheik | |
| 7 | shine-201 | 1028 | 6839 | "Winners | Top 64 | 3 | E | 3 | Fountain | win | captainfal | sheik | |
| 8 | shine-201 | 1028 | 6839 | "Winners | Top 64 | 3 | E | 3 | Yoshi's St | lose | captainfal | sheik | |
| 9 | shine-201 | 1028 | 6839 | "Winners | Top 64 | 3 | E | 3 | Final Dest | win | captainfal | sheik | |
| 10 | shine-201 | 4107 | 4863 | "Winners | Top 64 | 3 | F | 3 | Fountain | win | captainfal | iceclimbers | |
| 11 | shine-201 | 4107 | 4863 | "Winners | Top 64 | 3 | F | 3 | Final Dest | win | captainfal | iceclimbers | |
| 12 | shine-201 | 4107 | 4863 | "Winners | Top 64 | 3 | F | 3 | Yoshi's St | win | captainfal | iceclimbers | |
| 13 | shine-201 | 6126 | 16148 | "Winners | Top 64 | 3 | G | 3 | Battlefiel | win | marth | iceclimbers | |
| 14 | shine-201 | 6126 | 16148 | "Winners | Top 64 | 3 | G | 3 | Fountain | win | marth | iceclimbers | |
| 15 | shine-201 | 6126 | 16148 | "Winners | Top 64 | 3 | G | 3 | Final Dest | win | marth | iceclimbers | |
| 16 | shine-201 | 13932 | 1013 | "Winners | Top 64 | 3 | H | 3 | Dream Lar | win | fox | sheik | |
| 17 | shine-201 | 13932 | 1013 | "Winners | Top 64 | 3 | H | 3 | Battlefiel | lose | fox | sheik | |
| 18 | shine-201 | 13932 | 1013 | "Winners | Top 64 | 3 | H | 3 | PokÃ©m | win | fox | sheik | |
| 19 | shine-201 | 13932 | 1013 | "Winners | Top 64 | 3 | H | 3 | Yoshi's St | lose | fox | sheik | |
| 20 | shine-201 | 13932 | 1013 | "Winners | Top 64 | 3 | H | 3 | Final Dest | win | fox | marth | |
| 21 | shine-201 | 1004 | 3560 | "Winners | Top 64 | 3 | I | 3 | Fountain | lose | jigglypuff | fox | |
| 22 | shine-201 | 1004 | 3560 | "Winners | Top 64 | 3 | I | 3 | Dream Lar | win | jigglypuff | fox | |
| 23 | shine-201 | 1004 | 3560 | "Winners | Top 64 | 3 | I | 3 | Final Dest | win | jigglypuff | fox | |
| 24 | shine-201 | 1004 | 3560 | "Winners | Top 64 | 3 | I | 3 | Yoshi's St | win | jigglypuff | fox | |
| 25 | shine-201 | 1028 | 4107 | "Winners | Top 64 | 3 | K | 3 | Battlefiel | win | captainfal | captainfalcon | |
| 26 | shine-201 | 1028 | 4107 | "Winners | Top 64 | 3 | K | 3 | Dream Lar | lose | captainfal | captainfalcon | |
| 27 | shine-201 | 1028 | 4107 | "Winners | Top 64 | 3 | K | 3 | Final Dest | lose | captainfal | captainfalcon | |
| 28 | shine-201 | 1028 | 4107 | "Winners | Top 64 | 3 | K | 3 | Battlefiel | lose | captainfal | captainfalcon | |
| 29 | shine-201 | 6126 | 13932 | "Winners | Top 64 | 3 | L | 3 | Fountain | win | marth | fox | |
| 30 | shine-201 | 6126 | 13932 | "Winners | Top 64 | 3 | L | 3 | Dream Lar | lose | marth | fox | |
| 31 | shine-201 | 6126 | 13932 | "Winners | Top 64 | 3 | L | 3 | Final Dest | win | marth | fox | |
| 32 | shine-201 | 6126 | 13932 | "Winners | Top 64 | 3 | L | 3 | Yoshi's St | win | marth | fox | |
| 33 | shine-201 | 5080 | 19904 | "Losers 1" | Top 64 | 3 | R | 3 | Battlefiel | win | fox | fox | |
| 34 | shine-201 | 5080 | 19904 | "Losers 1" | Top 64 | 3 | R | 3 | Yoshi's St | win | fox | fox | |

# Data Cleaning Part 6

- If we wish to use certain models it will need to be a numerical dataset so we convert the dataframe to be numerical and also save it into a different dataset

```
le = LabelEncoder()
df['tournament_key'] = le.fit_transform(df['tournament_key'])
df['location_names'] = le.fit_transform(df['location_names'])
df['bracket_name'] = le.fit_transform(df['bracket_name'])
df['bracket_order'] = le.fit_transform(df['bracket_order'])
df['set_order'] = le.fit_transform(df['set_order'])
df['best_of'] = le.fit_transform(df['best_of'])
df['player_char'] = le.fit_transform(df['player_char'])
df['opponent_char'] = le.fit_transform(df['opponent_char'])
df['player_id'] = le.fit_transform(df['player_id'])
df['opponent_id'] = le.fit_transform(df['opponent_id'])
df['stage'] = le.fit_transform(df['stage'])
print(df)
df.to_csv(csv_file_path_clean_num, index=False)
# Display the modified dataframe
df = pd.read_csv(csv_file_path_clean)
print(df[df.isna().any(axis=1)]) # shows NaN values in data frame if it exists
print(df)
```

# 2nd Cleaned Data Set

- Each number in this new dataset represents a unique value in the other dataset based on a label encoder.
- 1083300 rows in total!

| | A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | tourname | player_id | opponent | location_r | bracket_n | bracket_o | set_order | best_of | stage | win_statu | player_ch | opponent_char | |
| 2 | 5980 | 3 | 23 | 32 | 318 | 2 | 99 | 1 | 1 | win | 8 | 17 | |
| 3 | 5980 | 3 | 23 | 32 | 318 | 2 | 99 | 1 | 29 | win | 8 | 17 | |
| 4 | 5980 | 3 | 23 | 32 | 318 | 2 | 99 | 1 | 29 | win | 8 | 17 | |
| 5 | 5980 | 12 | 28 | 32 | 318 | 2 | 153 | 1 | 6 | win | 1 | 17 | |
| 6 | 5980 | 21 | 779 | 32 | 318 | 2 | 207 | 1 | 1 | win | 1 | 23 | |
| 7 | 5980 | 21 | 779 | 32 | 318 | 2 | 207 | 1 | 9 | win | 1 | 23 | |
| 8 | 5980 | 21 | 779 | 32 | 318 | 2 | 207 | 1 | 29 | lose | 1 | 23 | |
| 9 | 5980 | 21 | 779 | 32 | 318 | 2 | 207 | 1 | 7 | win | 1 | 23 | |
| 10 | 5980 | 262 | 531 | 32 | 318 | 2 | 234 | 1 | 9 | win | 1 | 7 | |
| 11 | 5980 | 262 | 531 | 32 | 318 | 2 | 234 | 1 | 7 | win | 1 | 7 | |
| 12 | 5980 | 262 | 531 | 32 | 318 | 2 | 234 | 1 | 29 | win | 1 | 7 | |
| 13 | 5980 | 707 | 1690 | 32 | 318 | 2 | 261 | 1 | 1 | win | 13 | 7 | |
| 14 | 5980 | 707 | 1690 | 32 | 318 | 2 | 261 | 1 | 9 | win | 13 | 7 | |
| 15 | 5980 | 707 | 1690 | 32 | 318 | 2 | 261 | 1 | 7 | win | 13 | 7 | |
| 16 | 5980 | 1420 | 9 | 32 | 318 | 2 | 288 | 1 | 6 | win | 5 | 23 | |
| 17 | 5980 | 1420 | 9 | 32 | 318 | 2 | 288 | 1 | 1 | lose | 5 | 23 | |
| 18 | 5980 | 1420 | 9 | 32 | 318 | 2 | 288 | 1 | 22 | win | 5 | 23 | |
| 19 | 5980 | 1420 | 9 | 32 | 318 | 2 | 288 | 1 | 29 | lose | 5 | 23 | |
| 20 | 5980 | 1420 | 9 | 32 | 318 | 2 | 288 | 1 | 7 | win | 5 | 13 | |
| 21 | 5980 | 3 | 139 | 37 | 318 | 2 | 315 | 1 | 9 | lose | 8 | 5 | |
| 22 | 5980 | 3 | 139 | 37 | 318 | 2 | 315 | 1 | 6 | win | 8 | 5 | |
| 23 | 5980 | 3 | 139 | 37 | 318 | 2 | 315 | 1 | 7 | win | 8 | 5 | |
| 24 | 5980 | 3 | 139 | 37 | 318 | 2 | 315 | 1 | 29 | win | 8 | 5 | |
| 25 | 5980 | 21 | 266 | 37 | 318 | 2 | 335 | 1 | 1 | win | 1 | 1 | |

# Attributes Explained

**Attribute Values(11 attributes)**

- **tournament_key:** What tournament the match occurred at
- **player_id:** Who the player is
- **opponent_id:** Who the opponent is
- **location_names:** What round of the tournament it occurred in
- **bracket_name:** Where part of bracket it occurred in
- **bracket_order:** When did the bracket happen in relation to other brackets
- **set_order:** When the match happened relative to other ones in the same part of bracket
- **best_of:** How many rounds in order to win the match
- **stage:** Which in game stage the match was played on
- **player_character:** Who the player played
- **opponent_character:** Who the opponent played

**Target Values(1 target)**

- **win_status:** Did the player win or lose the match

**Training and testing sets:**

- 20% for testing and 80% for training

# Data Scaling

## Standard Scaler

- In order to use certain models such as the logistic regression model and ANN we use the standard scaler in order to normalize the data

# Classification Models

# ANN

[59]:
```
# Transform the data into an array for ANN reading
le = LabelEncoder()
y_train_encoded = le.fit_transform(y_train)
y_test_encoded = le.transform(y_test)
```

[49]:
```
# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[50]:
```
# ANN model
ann_model = MLPClassifier(random_state=42)
ann_model.fit(X_train_scaled, y_train_encoded)
```

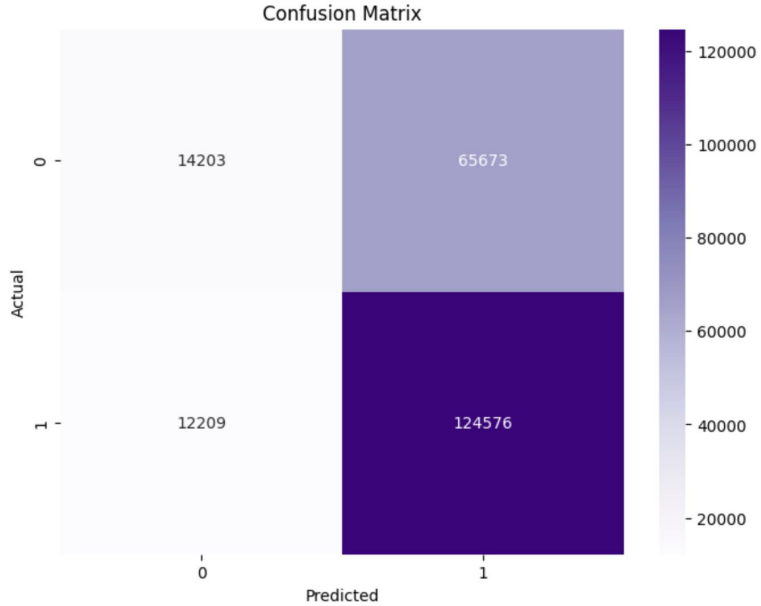
[50]:
```
MLPClassifier(random_state=42)
```

[51]:
```
# Predictions
ann_predictions = ann_model.predict(X_test_scaled)
```

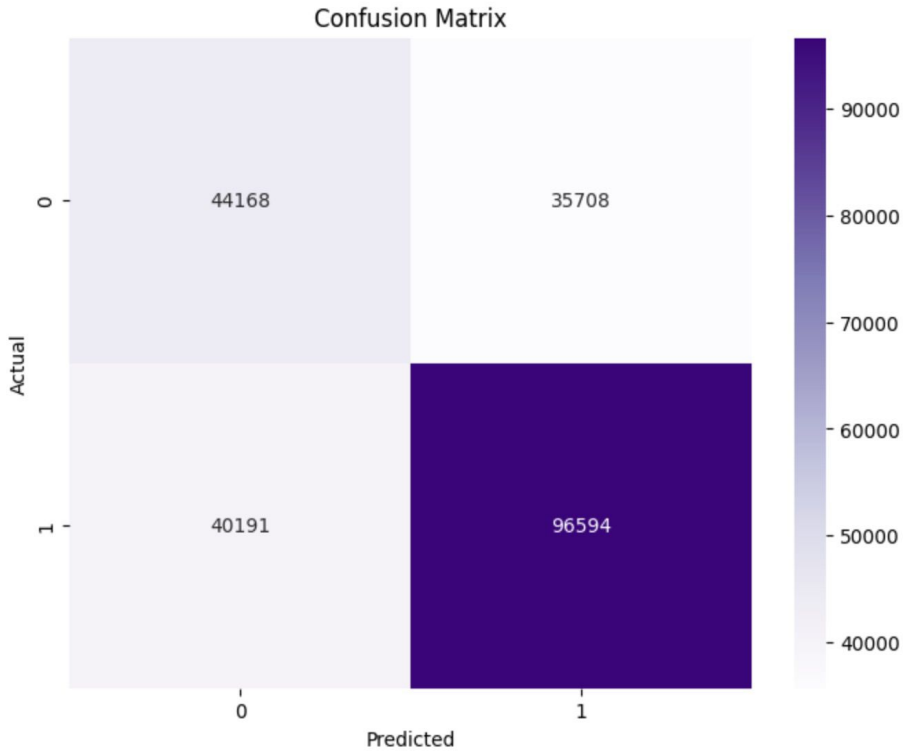
[52]:
```
# Evaluation
ann_accuracy = accuracy_score(y_test_encoded, ann_predictions)
ann_report = classification_report(y_test_encoded, ann_predictions)
ann_accuracy
print( ann_report)
```

```
              precision    recall  f1-score   support

           0       0.54      0.18      0.27     79876
           1       0.65      0.91      0.76    136785

    accuracy                           0.64    216661
   macro avg       0.60      0.54      0.51    216661
weighted avg       0.61      0.64      0.58    216661
```

Since we are predicting based on non-linear features, ANN is considered a good option for an extensive resource model



# Decision Trees




```
#Initialize decision tree model
dt_classifier_data = DecisionTreeClassifier(random_state=42)
```

```
# Train the model on 'data'
dt_classifier_data.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

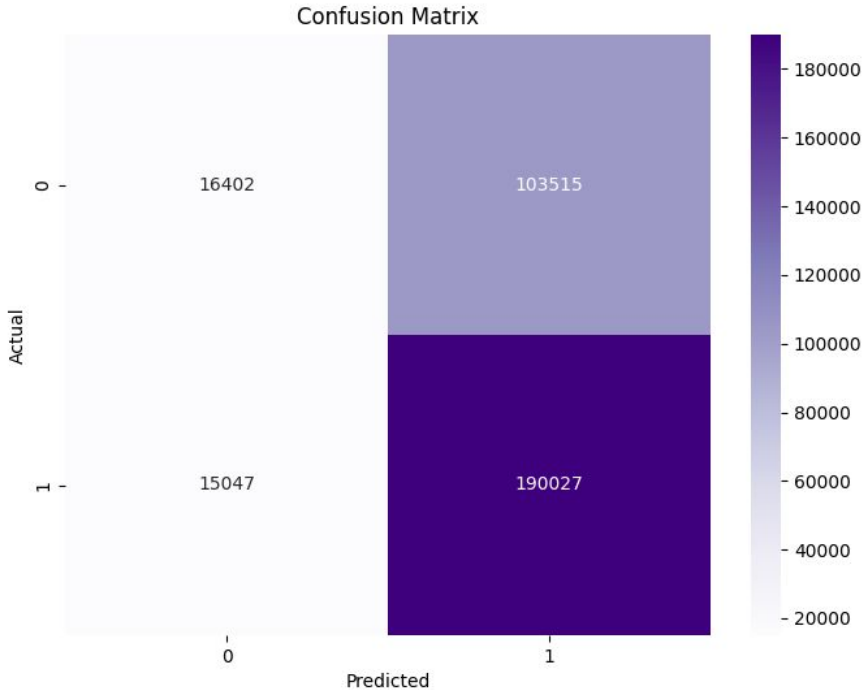
```
# Predictions for 'data'
y_pred_data = dt_classifier_data.predict(X_test)

# Evaluating the model on 'data'
accuracy_dt_data = accuracy_score(y_test, y_pred_data)
classification_rep_dt_data = classification_report(y_test, y_pred_data)
#Accuracy for decision tree
accuracy_dt_data
```

```
0.6496877610645202
```

```
print(classification_rep_dt_data)
```

```
              precision    recall  f1-score   support

        lose       0.52      0.55      0.54     79876
         win       0.73      0.71      0.72    136785

    accuracy                           0.65    216661
   macro avg       0.63      0.63      0.63    216661
weighted avg       0.65      0.65      0.65    216661
```

# Logistic Regression Model

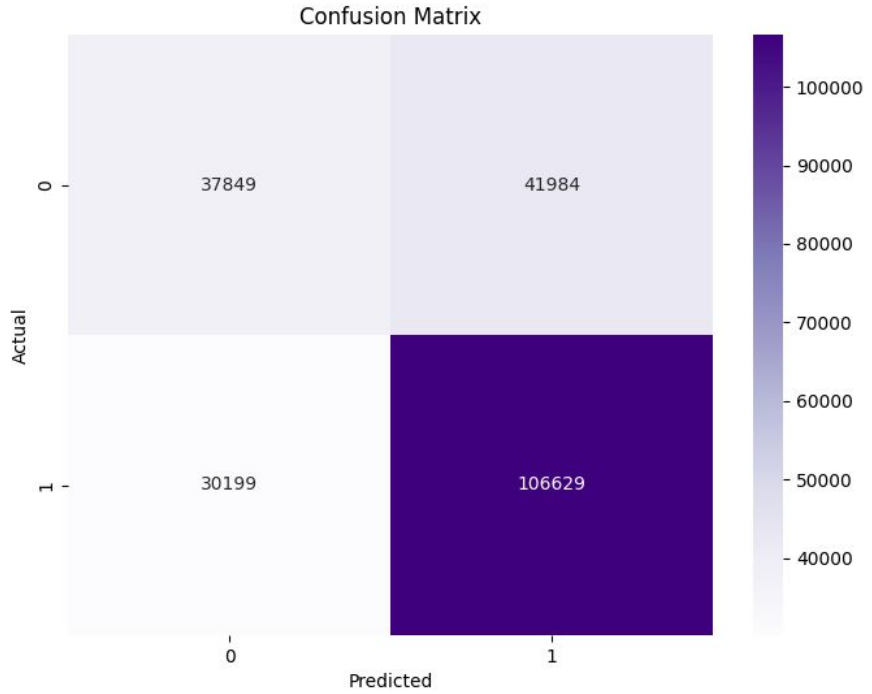



Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| lose | 0.52 | 0.14 | 0.22 | 119917 |
| win | 0.65 | 0.93 | 0.76 | 205074 |
| accuracy | | | 0.64 | 324991 |
| macro avg | 0.58 | 0.53 | 0.49 | 324991 |
| weighted avg | 0.60 | 0.64 | 0.56 | 324991 |

- Uses Standard Scaler normalization
- Very poor result for loss data as based on the results of logistic regression it overpredicts when it wins

# ADAboost

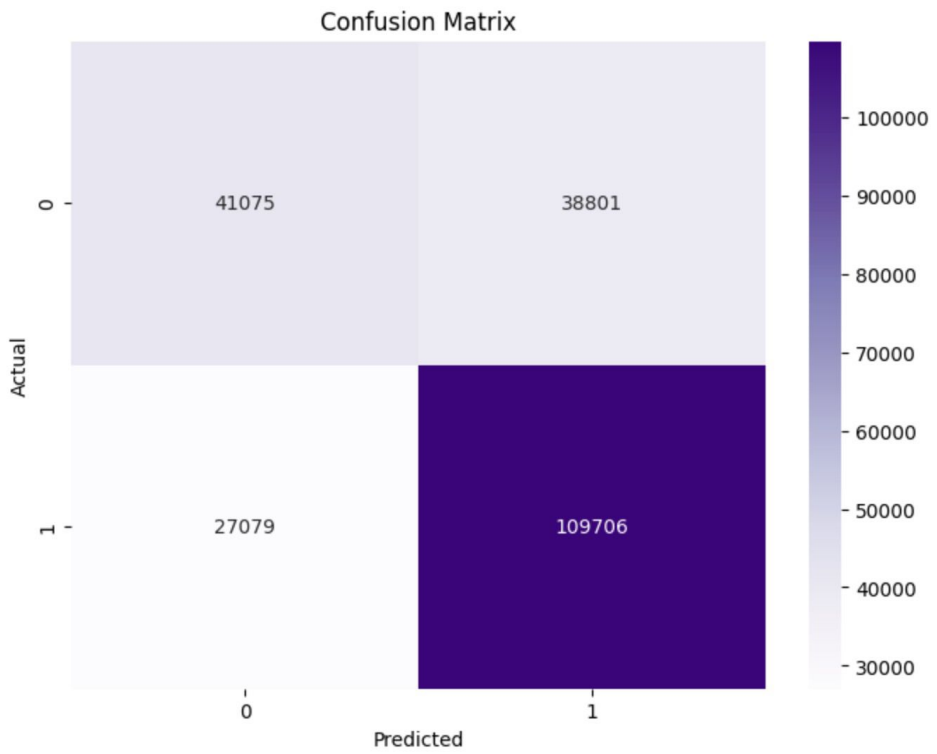



Classification Report

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| lose | 0.56 | 0.47 | 0.51 | 79833 |
| win | 0.72 | 0.78 | 0.75 | 136828 |
| accuracy | | | 0.67 | 216661 |
| macro avg | 0.64 | 0.63 | 0.63 | 216661 |
| weighted avg | 0.66 | 0.67 | 0.66 | 216661 |

max depth: 11
max nodes: 110
accuracy: 67%

# Random Forest




```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=42)
```

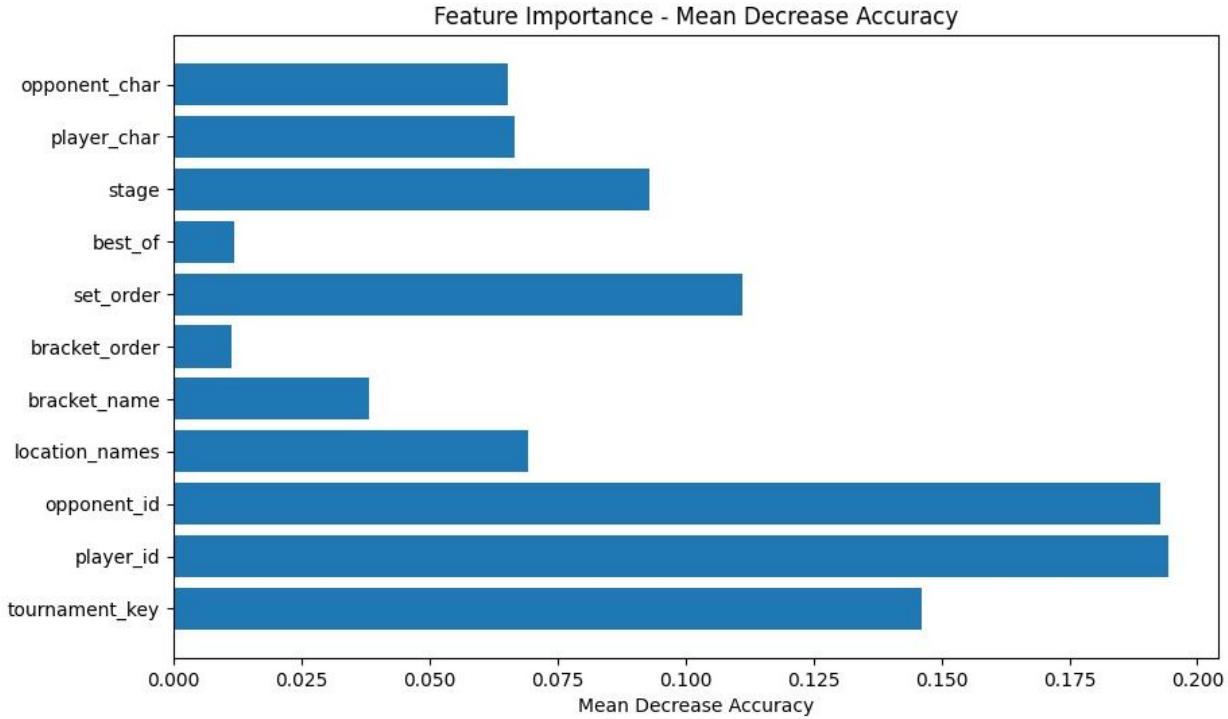
```
# Predictions
y_pred = rf_classifier.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

```
#Accuracy for random Forest
print(classification_rep)
```

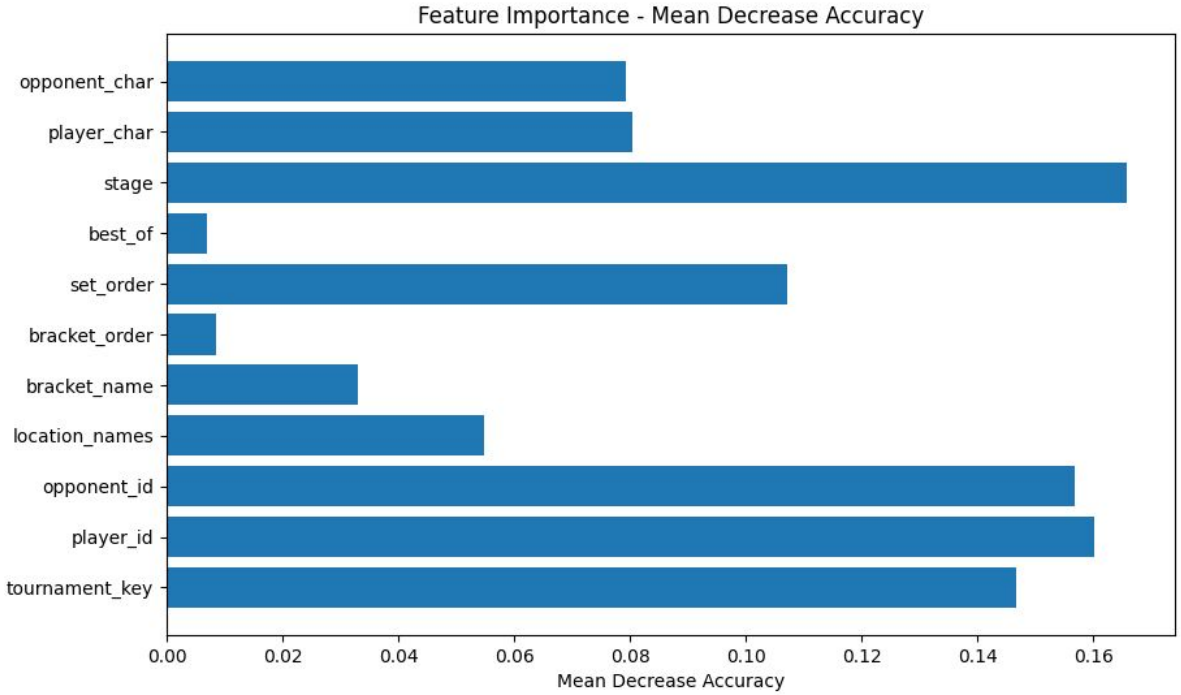
```
              precision    recall  f1-score   support

        lose       0.60      0.51      0.55     79876
         win       0.74      0.80      0.77    136785

    accuracy                           0.70    216661
   macro avg       0.67      0.66      0.66    216661
weighted avg       0.69      0.70      0.69    216661
```

# Conclusion

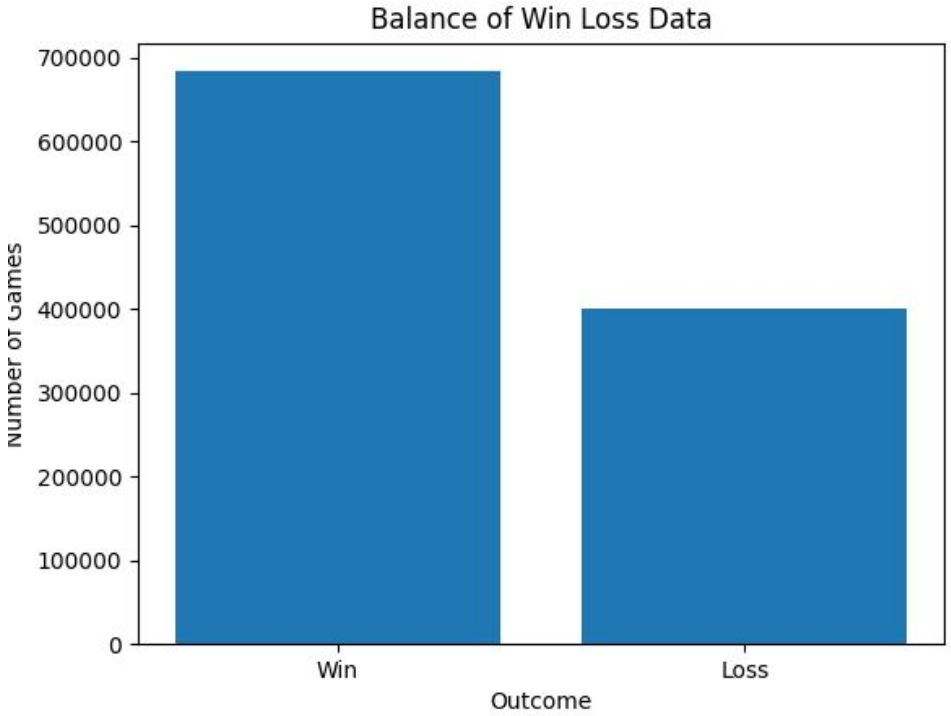
# Attribute Analysis for ADAboost



# Attribute Analysis for Random Forest



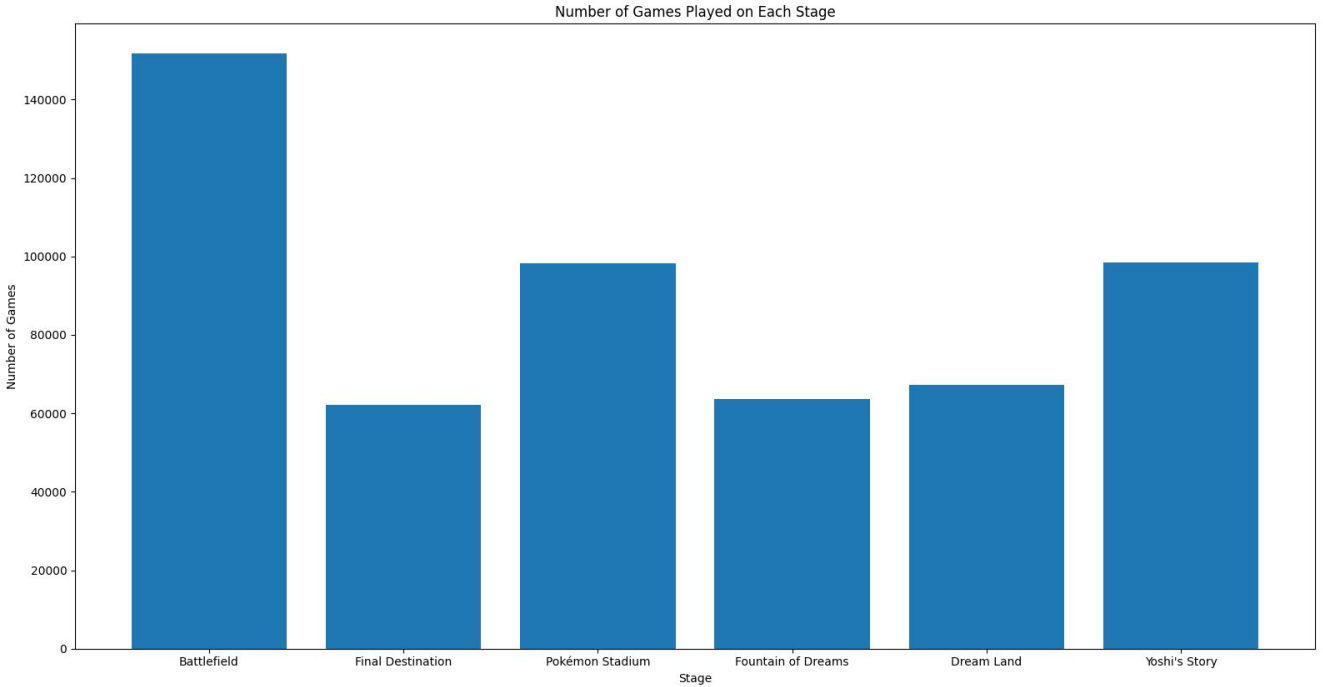
# Win Loss Analysis



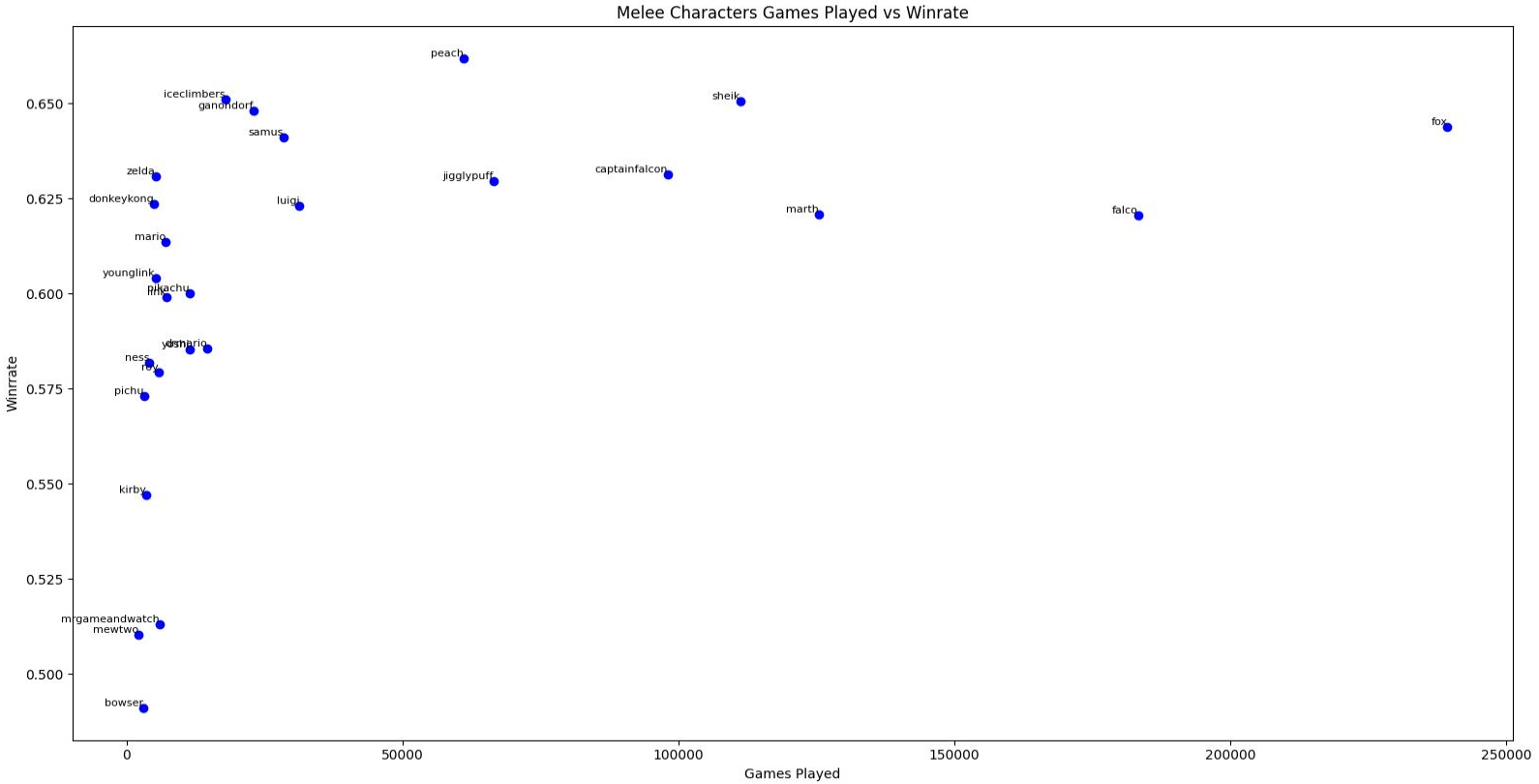

- The person who is player 1 seems to win a majority of the time
- Leads to around a 65-35 split in the data which may explain low f1 scores for loss

# Stage Analysis

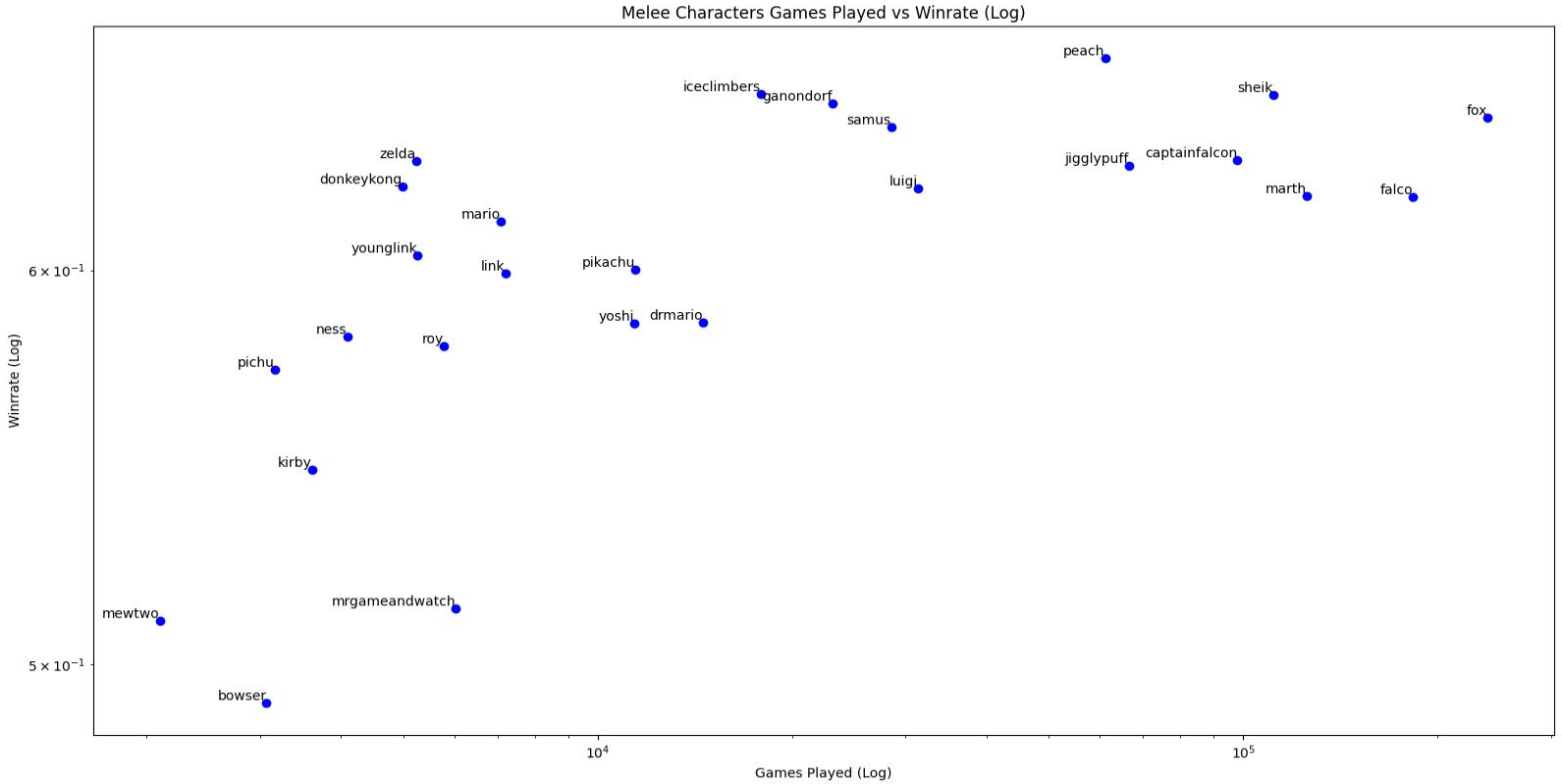
Note we just do games played and not win rate as both players play on the same stage



# Character Analysis (1)



# Character Analysis (2)



# Potential Applications

1. **Game Developer Decisions:**
    a. As seen from the analysis the character that you play and the character that your opponent plays has a high impact on the outcome of the game. As shown many of these characters are stronger and have stronger play rates than others. Game Developers can use this data for a potential sequel or remaster in order to balance the roster more.
    b. Stage also plays a significant role in the outcome of a game. Game Developers can similarly act on this data in order to either make the stages more balanced and potentially redesign the stages in order for more people to want to play on them.
2. **eSport Teams:**
    a. Some of the strongest predictors on if player 1 wins is who they are, who their opponent is, and what tournament it happened at. All of which suggests player skill is the most important factor in determining a winner.
    b. From which eSport Teams can make more informed decisions on who to sponsor safely based on simply who are the strongest players with the best win rates over everyone else.
3. **Tournament Organizers(TOs):**
    a. Similarly to eSport Teams, TOs can use this data to better accurately predict the winners and therefore seed the brackets more fairly for all entrants based on win rates of each player
    b. As shown in the attribute analysis the amount of games in a set does not seem to impact the outcome much. Because tournaments take a long time to run TOs can safely change most sets to be best of 3 in order to decrease the length of the tournament
    c. Similarly to the Game Developer perspective, TOs can potentially opt to ban certain characters or stages if they deem them to be too strong and unfair to match integrity