Technical Report (League of Legends)

May 15, 2022

0.1 Picking Smarter Games in League of Legends

- Creator: Jacky Dam
- LinkedIn: https://www.linkedin.com/in/jackydam/
- GitHub: https://github.com/jacky3331

0.2 Summary of Findings

0.2.1 Introduction

League of Legends is a popular team-based strategy game where players are able to select champions (characters with unique fighting abilities) and work together to destroy the enemys' base. Before starting a competitive ranked match, players are queued up in a matchmaking lobby in which each of the two teams alternate between picking a unique champion visible to everyone before the game start. A ranked champion selection demo can be found in the later sections.

With the ability to scrape large datasets of League of Legends match history information, I explored the possibility of creating models which would be able to predict the winning team given the champion team compositions. If it was possible to create a model that is able to predict the outcome of a ranked game with a reasonable performance (better than 50%), then it would allow players to make a better informed decision of dodging a less ideal game to preserve champion stats and improve global player ranking.

0.2.2 Data Collection

By learning how to utilize the official Riot Games API library (Cassiopeia), I was able to generate a large dataset of match history information to start my technical analysis. This API library is available to the public with a certain limit restriction of how much player data could be retrieved per minute. The script I used for this project can be found in the same folder with this python notebook. More information about the Cassiopeia API library can be found below: - https://github.com/meraki-analytics/cassiopeia

0.2.3 Cleaning and EDA

Since I utilized a script to collect lots of match history information, I need to create a method to process the raw data in order to ensure that the model isn't working with duplicate/incomplete match information. Additionally, I chose to select certain columns which includes: match id, team color, team position, champion, kills, deaths, assists, and whether they won their game.

In my Exploratory Data Analysis, I drew inference from visualizing the win rates of the champions and illustrated the popularity of the champions by the number of games played. Through my

analysis, I am able to observe interesting trends which I would later use to build the perfect team composition with my predictive model.

0.2.4 Model Selection

0.2.5 Results & Outcome

0.3 Imports

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import ttest_ind

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from category_encoders import  OneHotEncoder, OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn import linear_model
import random

from IPython.display import YouTubeVideo
from tabulate import tabulate
```

0.3.1 Ranked Champion Selection Demo (optional)

[2]:
```
YouTubeVideo("CfXQrfhFRnI")
```

[2]:

0.4 Code

0.4.1 Processing & Cleaning

[3]:
```
def clean_data(match_history):
    # Make a copy of the dataframe
    df = match_history.copy()

    # Select columns we want to work with
    df = df[["match_id", "side", "team_position", "champion", "kills",␣
↪"deaths", "assists", "win"]]
    df = df.drop_duplicates()

    # Drop player information if match is incomplete i.e. match id does not␣
↪contain 10 players
    df = df[df["match_id"].map((df["match_id"].value_counts()==10))]

    return df
```

[4]:
```
match_history = pd.read_csv("match_history.csv")
match_history.head()
```

[4]:
```
      match_id        duration    summoner_name  side team_position  \
0  4306514682  0:00:01.822000  top is weakside  blue      top_lane
1  4306514682  0:00:01.822000        Bellydrum  blue        jungle
2  4306514682  0:00:01.822000      Roh Jungeui  blue      mid_lane
3  4306514682  0:00:01.822000  sun kissed dawn  blue      bot_lane
4  4306514682  0:00:01.822000         Verayson  blue       utility

   summoner_level    win  champion  kills  deaths  assists  damage_to_turrets  \
0              48  False  Malphite      1       8        8                323
1             248  False    Graves     12      10        8                743
2             433  False     Sylas      9       9       10               1954
3              41  False    Lucian      5       9       10               1944
4             109  False     Rakan      3       6       16                  0

   gold_earned  turret_takedowns  vision_score  minions_killed  first_blood  \
0         8539                 0            26             150        False
1        14714                 0            90              81         True
2        12959                 1            13             183        False
3        11379                 0            16             190        False
4         8784                 0            70              34        False

   first_tower
0        False
1        False
2        False
3        False
4        False
```

[5]:
```
# There are 29353 rows of player data, and 18 different column features
match_history.shape
```

[5]:
```
(34361, 18)
```

[6]:
```
# Check for duplicate/incomplete data (match_ids with more or less than 10␣
↪players)
match_history['match_id'].value_counts().sort_values()
```

[6]:
```
4304892065      1
4303458524      3
4305634239      4
4305773030      4
4304402761      4
               …
4305878645     70
4305922045     80
4306420131     90
4306514682    100
```

```
4305980817    100
Name: match_id, Length: 2734, dtype: int64
```

[7]:
```
df = clean_data(match_history)
df.head()
```

[7]:

	match_id	side	team_position	champion	kills	deaths	assists	win
0	4306514682	blue	top_lane	Malphite	1	8	8	False
1	4306514682	blue	jungle	Graves	12	10	8	False
2	4306514682	blue	mid_lane	Sylas	9	9	10	False
3	4306514682	blue	bot_lane	Lucian	5	9	10	False
4	4306514682	blue	utility	Rakan	3	6	16	False

[8]:
```
# Validate the shape of our cleaned dataframe
df.shape
```

[8]:
```
(27120, 8)
```

[9]:
```
# Validate that our cleaned dataframe doesn't have duplicate/incomplete data␣
↪(match_ids with more or less than 10 players)
df['match_id'].value_counts().sort_values()
```

[9]:
```
4303216652    10
4302149827    10
4307085501    10
4306046994    10
4306190360    10
              ..
4271376223    10
4298264438    10
4305825690    10
4307121969    10
4294744029    10
Name: match_id, Length: 2712, dtype: int64
```

[10]:
```
# Validate that we aren't working with incomplete missing data
df.isna().sum()
```

[10]:
```
match_id         0
side             0
team_position    0
champion         0
kills            0
deaths           0
assists          0
win              0
dtype: int64
```

[11]:
```
# Validate data types of each column feature
df.dtypes
```

[11]:
```
match_id          int64
side             object
team_position    object
champion         object
kills             int64
deaths            int64
assists           int64
win                bool
dtype: object
```

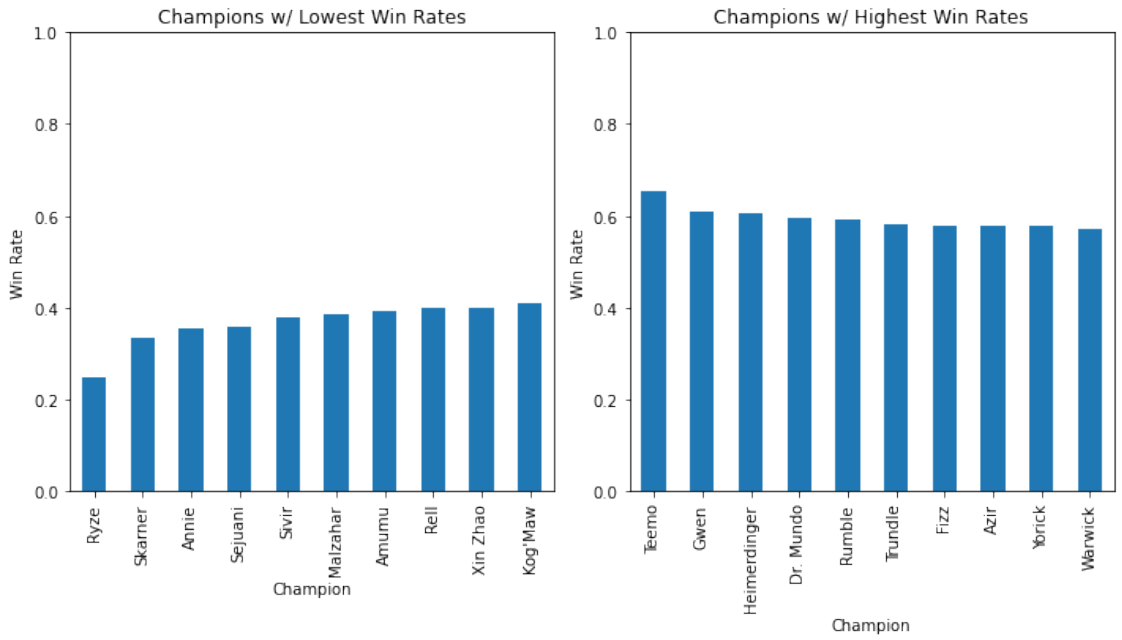
Our method of cleaning the dataframe looks correct, now we can get into our analysis…

0.4.2 Exploratory Data Analysis

In the figures below, we can observe the champions with the lowest and highest win rates. After creating our model, we will see whether we determine the probability of winning with fixed team compositions with the information shown in the bar plots above.

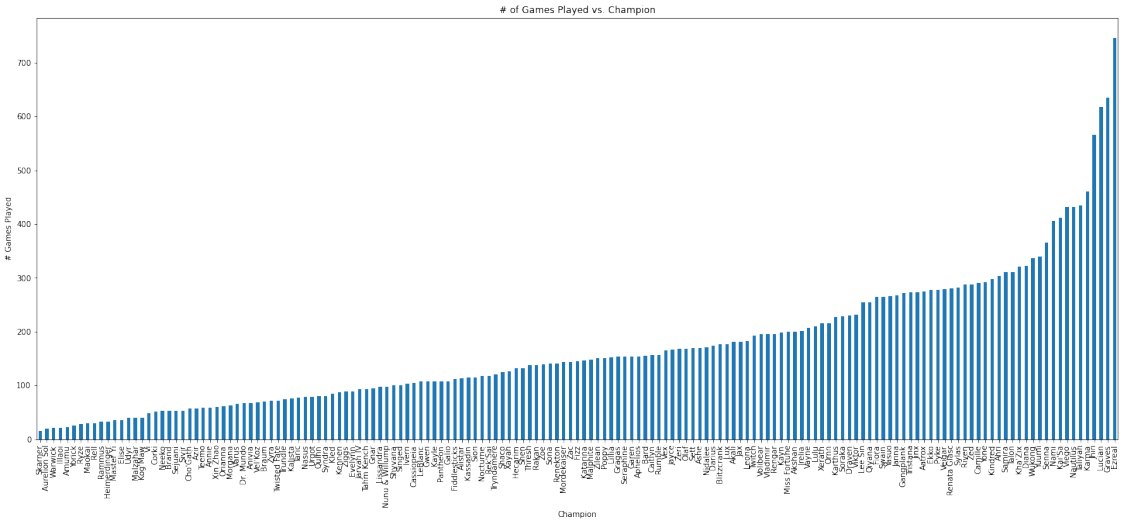
[12]:
```
champion_dist = (
    df.pivot_table(index="win",
                   columns="champion",
                   aggfunc="size")
)

champion_dist = champion_dist/champion_dist.sum()
champion_dist = champion_dist.T

fig, axs = plt.subplots(ncols=2, nrows=2, figsize=(10,10))
fig.delaxes(axs[1][1])
fig.delaxes(axs[1][0])

ax = champion_dist[True].sort_values(ascending=True)[:10].plot.
 ↪bar(ax=axs[0][0], title="Champions w/ Lowest Win Rates")
ax.set_xlabel("Champion")
ax.set_ylabel("Win Rate")
ax.set_ylim(0,1)

ax = champion_dist[True].sort_values(ascending=False)[:10].plot.
 ↪bar(ax=axs[0][1], title="Champions w/ Highest Win Rates")
ax.set_xlabel("Champion")
ax.set_ylabel("Win Rate")
ax.set_ylim(0,1)
```

```
fig.tight_layout()
plt.show()
```



In the figure below, we can observe the popularity of all the champions selected by the players in our data set. A quick overview of the plot indicate players are selective of the characters they choose to play League of Legends with.

[13]:
```
plt.figure(figsize=(25,10))
df.groupby("champion").count()["match_id"].rename("# Games Played").
 ↪sort_values(ascending=True).plot.bar(title="# of Games Played vs. Champion")
plt.ylabel("# Games Played")
plt.xlabel("Champion")
plt.show()
```



The tables shown below display the champions with the most and least games played.

[14]:
```
print(df.groupby("champion").count()["match_id"].rename("# Games Played").
↪sort_values(ascending=False)[:10].to_markdown())
```

champion	# Games Played
Ezreal	746
Graves	635
Lucian	617
Jhin	565
Karma	460
Taliyah	434
Nautilus	432
Viego	431
Kai'Sa	412
Nami	405

[15]:
```
print(df.groupby("champion").count()["match_id"].rename("# Games Played").
↪sort_values(ascending=True)[:10].to_markdown())
```

champion	# Games Played
Skarner	15
Aurelion Sol	20
Warwick	21
Illaoi	21
Amumu	23
Yorick	26

```
| Ryze          |               28 |
| Maokai        |               30 |
| Rell          |               30 |
| Rammus        |               33 |
```

0.4.3 Predictive Model (Logistic Regression)

[16]:
```
def feat(datum):
    feat = [1]
    for champion_selected in datum:
        champion = [0]*len(champion_dict)
        champion[champion_dict[champion_selected]] = 1
        feat += champion
    return feat
```

[17]:
```
# Prepare the team composition data with their win label
data = df.groupby(["match_id", "win"])["champion"].apply(list).unstack()

# Set aside a test set of size 298.
reserved_test = data.iloc[:298]

output = [(i, True) for i in data[True].values]
output += [(i, False) for i in data[False].values]
```

[18]:
```
data.head()
```

[18]:
```
win                                                  False  \
match_id
4167993974        [Camille, Graves, Orianna, Jhin, Thresh]
4168019856          [Yone, Jarvan IV, Yasuo, Caitlyn, Lux]
4168150884         [Jayce, Taliyah, Viktor, Caitlyn, Shen]
4168188605  [Tryndamere, Olaf, Akali, Heimerdinger, Rakan]
4168265192            [Wukong, Graves, Yone, Jhin, Xerath]

win                                                   True
match_id
4167993974        [Gwen, Shaco, Yasuo, Kai'Sa, Alistar]
4168019856             [Sett, Shaco, Vex, Draven, Pyke]
4168150884  [Mordekaiser, Lee Sin, Ahri, Alistar, Jhin]
4168188605    [Camille, Rek'Sai, Viktor, Ezreal, Leona]
4168265192            [Camille, Ekko, Corki, Ashe, Vex]
```

[19]:
```
output[:5]
```

[19]:
```
[(['Gwen', 'Shaco', 'Yasuo', "Kai'Sa", 'Alistar'], True),
 (['Sett', 'Shaco', 'Vex', 'Draven', 'Pyke'], True),
 (['Mordekaiser', 'Lee Sin', 'Ahri', 'Alistar', 'Jhin'], True),
```

```
 (['Camille', "Rek'Sai", 'Viktor', 'Ezreal', 'Leona'], True),
 (['Camille', 'Ekko', 'Corki', 'Ashe', 'Vex'], True)]
```

[20]:
```
# Create a dictionary for the unique champions in the game to prep for one-hot-encoding
champion_dict = dict(enumerate(np.unique([i[0] for i in output])))
champion_dict = dict((v,k) for k,v in champion_dict.items())
```

[21]:
```
X = [feat(d[0]) for d in output]
y = [d[1] for d in output]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

[22]:
```
model = sklearn.linear_model.LogisticRegression(penalty="l2", fit_intercept=True, C=1, class_weight="balanced")
model.fit(X_train, y_train)
```

[22]:
```
LogisticRegression(C=1, class_weight='balanced')
```

[23]:
```
print('Training Accuracy:', model.score(X_train, y_train))
print('Validation Accuracy:', model.score(X_test, y_test))
```

```
Training Accuracy: 0.6310209725743259
Validation Accuracy: 0.511520737327189
```

Looking at the results of my Logistic Regression model, the model appears to perform well on the training set but perform poorly on the validation set. Although the results don't appear very impressive, this accuracy is misleading because it doesn't evaluate the accuracy of predicting the winning team out of two different team compositions. I will evaluate the true performance of the model on the test set after implementing other models.

0.4.4 Predictive Model (Random Forest)

[24]:
```
model = RandomForestClassifier(max_depth=10)
model.fit(X_train, y_train)
```

[24]:
```
RandomForestClassifier(max_depth=10)
```

[25]:
```
print('Training Accuracy:', model.score(X_train, y_train))
print('Validation Accuracy:', model.score(X_test, y_test))
```

```
Training Accuracy: 0.708919105784743
Validation Accuracy: 0.48847926267281105
```

[26]:
```
data.head()
```

[26]:
```
win                                                  False  \
match_id
4167993974        [Camille, Graves, Orianna, Jhin, Thresh]
4168019856          [Yone, Jarvan IV, Yasuo, Caitlyn, Lux]
4168150884         [Jayce, Taliyah, Viktor, Caitlyn, Shen]
4168188605  [Tryndamere, Olaf, Akali, Heimerdinger, Rakan]
4168265192            [Wukong, Graves, Yone, Jhin, Xerath]

win                                                   True
match_id
4167993974        [Gwen, Shaco, Yasuo, Kai'Sa, Alistar]
4168019856             [Sett, Shaco, Vex, Draven, Pyke]
4168150884  [Mordekaiser, Lee Sin, Ahri, Alistar, Jhin]
4168188605    [Camille, Rek'Sai, Viktor, Ezreal, Leona]
4168265192            [Camille, Ekko, Corki, Ashe, Vex]
```

[27]:
```
reserved_test.head()
```

[27]:
```
win                                                  False  \
match_id
4167993974        [Camille, Graves, Orianna, Jhin, Thresh]
4168019856          [Yone, Jarvan IV, Yasuo, Caitlyn, Lux]
4168150884         [Jayce, Taliyah, Viktor, Caitlyn, Shen]
4168188605  [Tryndamere, Olaf, Akali, Heimerdinger, Rakan]
4168265192            [Wukong, Graves, Yone, Jhin, Xerath]

win                                                   True
match_id
4167993974        [Gwen, Shaco, Yasuo, Kai'Sa, Alistar]
4168019856             [Sett, Shaco, Vex, Draven, Pyke]
4168150884  [Mordekaiser, Lee Sin, Ahri, Alistar, Jhin]
4168188605    [Camille, Rek'Sai, Viktor, Ezreal, Leona]
4168265192            [Camille, Ekko, Corki, Ashe, Vex]
```

[28]:
```
reserved_test["Probability of Winning (False)"] = [model.
 ↪predict_proba([feat(i)])[0][1] for i in reserved_test[False].values]
reserved_test["Probability of Winning (True)"] = [model.
 ↪predict_proba([feat(i)])[0][1] for i in reserved_test[True].values]
reserved_test = reserved_test[[False, "Probability of Winning (False)", True,␣
 ↪"Probability of Winning (True)"]]
```

```
<ipython-input-28-a4e630d7e56f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  reserved_test["Probability of Winning (False)"] =
[model.predict_proba([feat(i)])[0][1] for i in reserved_test[False].values]
<ipython-input-28-a4e630d7e56f>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  reserved_test["Probability of Winning (True)"] =
[model.predict_proba([feat(i)])[0][1] for i in reserved_test[True].values]
```

In the cell above, I used the model to predict the probability of winning for both team compositions in a match.

[29]:
```
reserved_test.head(20)
```

[29]:
```
win                                                    False  \
match_id
4167993974          [Camille, Graves, Orianna, Jhin, Thresh]
4168019856            [Yone, Jarvan IV, Yasuo, Caitlyn, Lux]
4168150884           [Jayce, Taliyah, Viktor, Caitlyn, Shen]
4168188605    [Tryndamere, Olaf, Akali, Heimerdinger, Rakan]
4168265192              [Wukong, Graves, Yone, Jhin, Xerath]
4168337089              [Riven, Graves, Ryze, Kai'Sa, Karma]
4168350291       [Kled, Graves, Twisted Fate, Kog'Maw, Bard]
4172216681        [Camille, Viego, Malzahar, Jinx, Nautilus]
4181062332            [Aatrox, Kayn, Sylas, Swain, Nautilus]
4181088781              [Akali, Graves, Xerath, Jhin, Yuumi]
4184918527                 [Yone, Qiyana, Sylas, Jinx, Nami]
4186521278            [Gragas, Diana, Yone, Aphelios, Janna]
4190767290              [Gwen, Lee Sin, Zeri, Ezreal, Karma]
4195567583                  [Gragas, Shaco, Zed, Jhin, Nami]
4195635632        [Yone, Fiddlesticks, Qiyana, Jhin, Thresh]
4195673110           [Irelia, Diana, Viktor, Caitlyn, Janna]
4221448508        [Gangplank, Graves, Akali, Caitlyn, Karma]
4221870027  [Dr. Mundo, Shaco, Twisted Fate, Jinx, Nautilus]
4222092866         [Tryndamere, Kayn, Viktor, Caitlyn, Nami]
4222137808              [Gragas, Kha'Zix, Viego, Jhin, Nami]

win         Probability of Winning (False)  \
match_id
4167993974                        0.484968
4168019856                        0.495983
4168150884                        0.491679
4168188605                        0.500558
4168265192                        0.482397
4168337089                        0.457119
4168350291                        0.508978
```

```
4172216681                       0.467169
4181062332                       0.492366
4181088781                       0.472815
4184918527                       0.498817
4186521278                       0.472877
4190767290                       0.514673
4195567583                       0.476255
4195635632                       0.496142
4195673110                       0.485398
4221448508                       0.491790
4221870027                       0.494241
4222092866                       0.481687
4222137808                       0.439047

win                                                    True  \
match_id
4167993974            [Gwen, Shaco, Yasuo, Kai'Sa, Alistar]
4168019856                 [Sett, Shaco, Vex, Draven, Pyke]
4168150884      [Mordekaiser, Lee Sin, Ahri, Alistar, Jhin]
4168188605        [Camille, Rek'Sai, Viktor, Ezreal, Leona]
4168265192                [Camille, Ekko, Corki, Ashe, Vex]
4168337089              [Zed, Viego, Malzahar, Jhin, Pyke]
4168350291           [Nasus, Lee Sin, Malzahar, Jinx, Lulu]
4172216681   [Tryndamere, Zed, Twisted Fate, Ezreal, Rakan]
4181062332                [Yone, Diana, Yasuo, Jinx, Lulu]
4181088781     [Wukong, Xin Zhao, Malzahar, Aphelios, Lulu]
4184918527    [Pantheon, Fiddlesticks, Akali, Jhin, Soraka]
4186521278             [Urgot, Viego, Kassadin, Jhin, Sona]
4190767290               [Irelia, Diana, Zed, Jinx, Thresh]
4195567583            [Gwen, Lee Sin, Viktor, Sivir, Janna]
4195635632          [Akshan, Karthus, Rumble, Ezreal, Pyke]
4195673110                [Jax, Karthus, Corki, Jhin, Pyke]
4221448508  [Irelia, Talon, Katarina, Kai'Sa, Renata Glasc]
4221870027                [Gwen, Lillia, Sylas, Jhin, Lux]
4222092866             [Garen, Hecarim, Yasuo, Ekko, Yuumi]
4222137808         [Tryndamere, Kayn, Viktor, Kai'Sa, Pyke]

win         Probability of Winning (True)
match_id
4167993974                       0.518139
4168019856                       0.542935
4168150884                       0.498324
4168188605                       0.488743
4168265192                       0.495669
4168337089                       0.525712
4168350291                       0.488092
4172216681                       0.479087
```

4181062332	0.497475
4181088781	0.488739
4184918527	0.503693
4186521278	0.509582
4190767290	0.490222
4195567583	0.496862
4195635632	0.507303
4195673110	0.525685
4221448508	0.494780
4221870027	0.521007
4222092866	0.535639
4222137808	0.526863

[30]:
```
# Calculate the accuracy of the model when comparing the probability of winning␣
 ↪for two opposing team compositions
result = (reserved_test["Probability of Winning (True)"] >␣
 ↪reserved_test["Probability of Winning (False)"]).mean()
print("Accuracy (Test Set): " + str(result))
```

```
Accuracy (Test Set): 0.7550335570469798
```

In the previous section, I reserved about 298 samples from the data set which was never used to train the models. The flaw of the accuracy that we evaluated for the training set and test set was that it was evaluating whether a single team composition would win the game; however a specific team composition doesn't necessarily imply an automatic win. The way that I evaluated the performance of my model is by calculating the probablity of winning for both team compositions in a single match, and having the model determine the winning team by choosing the team composition with the highest probability of winning. We can observe that the accuracy is quite high compared to my previous iterations of this notebook, and make a better informed decision by comparing the probability of winning for the two team compositions.

0.4.5 Summary of Results

Although the results of the training set and validation was decieving, I was able to significantly improve my model compared to previous iteration and earn an accuracy of about 77% on the test set. By comparing the probability of winning of the two team compositions during champion selection, players will be able to have an edge in climbing global rankings by having the ability to make a better informed decision and dodge a potentially devasting game of League of Legends.

There are definely improvements that could be made on this model performance. In this project, I primarily focused on the population of high-ranking players which introduced bias into this model. It would make to sense to explore other model options such as boosting algorithms, decision trees, etc., look into collecting more sample data, and create a nicer user interface where players can input team composition information to get the probablity of winning.

[ ]: