Milestone Report

John Bello

Introduction

The purpose of this project is to analyze professional League of Legends matches from 2014 to 2018 with the intent to create a model capable of determining which team is most likely to win a game, as well as which factors were most likely to lead to a victory.

This analysis can influence what direction Riot Games, the developer of League of Legends, will take regarding the gameplay. By understanding which factors were most responsible for League's popularity and what game mechanics are the most significant, the game developers can choose to target and improve those factors, potentially increasing the popularity of League of Legends and Riot Games' revenue.

The data required to complete this analysis is the League of Legends dataset, which can be found on Kaggle at the following URL: https://www.kaggle.com/chuckephron/leagueoflegends.
This dataset contains information on each game's bans, metrics on player performance, and objective control.

Game Explanation

League of legends is a multiplayer online battle arena (MOBA). There are two teams of 5 players and the purpose of the game is to destroy the opposing team's base structure, also known as a Nexus. In order to reach the Nexus, the opposing team must destroy the towers protecting the Nexus. There are three lanes with 3 towers each. There is a structure in each lane called an inhibitor which buffs the invading team's minions. There are an additional two towers located by the Nexus. The purpose of the towers is to protect their teams. Minions are creatures that appear through each lane and autonomously attempt to invade the enemy team's base and destroy their structures. In competitive games, the roles are as follows:

- Top Laner, located in top lane
- Jungle, resides within the area of the map that is not part of the lanes
- Mid Laner, located in the middle lane
- ADC, located at the bottom lane
- Support, located at the bottom lane

There are multiple objectives throughout the game which empower the team they are taken by. The names of these objectives are the rift herald, dragons, and barons. These are highly contested and are often fought for by each team. There is no time limit within the game, and champions (which the players control) are strengthened through items bought by gold.

Dataset Description

The league of legends dataset contains game metrics regarding competitive league of legends matches from 2014 to 2018. It contains seven csv files, one of which functions as a combination of the other six. The other six cover the following topics:

1. Bans
2. Gold
3. Kills
4. Matchinfo
5. Monsters
6. Structures

For this project, the majority of the processing is done on the summary csv file, titled "LeagueofLegends.csv." This is done to avoid unnecessary merging and joining steps.

Data Wrangling

First, the `.head()`, `.describe()`, and `.info()` methods were called on the LeagueofLegends dataframe. It was observed that some columns contained missing values. Upon further inspection, this was a result of the format of the specific tournament the data was referencing. Values such as a team name, or dragons were missing. Because this dataset was fairly large (~8000 samples) and the number of missing values was quite small (~40), it was decided that the rows containing missing values could be dropped without much consequence. Afterwards, each column of dataframe was inspected in order to determine whether they would need to be processed and how to properly process them. Categorical variables were dummy encoded, the variances of lists were calculated, and columns describing objectives were counted. A summary of how each column was processed is displayed below.

Table I. Summary of data wrangling procedures.

Column(s)	**Procedure**
Address, rResult	Dropped
League, Year, Season, Type, blueTeamTag, redTeamTag, blueTop, blueTopChamp, blueJungle, blueJungleChamp, blueMiddle, blueMiddleChamp, blueADC, blueADCChamp, blueSupport, blueSupportChamp, redTop, redTopChamp, redJungle, redJungleChamp, redMiddle, redMiddleChamp, redADC, redADCChamp, redSupport, redSupportChamp	Dummy encoded
golddiff, goldblue, goldred, goldblueTop, goldblueJungle, goldblueMiddle, goldblueADC, goldblueSupport, goldredTop, goldredJungle, goldredMiddle, goldredADC, goldredSupport	Calculate Variance
bKills, bInhibs, bDragons, bBarons, bHeralds, rKills, rInhibs, rDragons, rDragons, rHeralds	Count
bDragons, rDragons	Pivot

The functions used to process the columns are the following:

```
def bans_to_encodable_cols(df, cols, col_names):
    for col, names in zip(cols, col_names):
        df[col] = df[col].apply(ast.literal_eval)
        encodable_cols = pd.DataFrame(df[col].values.tolist(), index=league_df.index, columns=names)
        df = pd.concat([df.drop(col, axis=1), encodable_cols], axis = 1)

    return df
```

```
def delete_col_get_dummies(df, cols):

    for col in cols:
        if 'blue' in col:
            if 'Top' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bTop')], axis=1) # add prefix or suffix
            elif 'Jungle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bJungle')], axis=1) # add prefix or suffix
            elif 'Middle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bMiddle')], axis=1) # add prefix or suffix
            elif 'ADC' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bADC')], axis=1) # add prefix or suffix
            elif 'Support' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='bSupport')], axis=1) # add prefix or suffi
            else:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='b')], axis=1) # add prefix or suffix
        elif 'red' in col:
            if 'Top' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rTop')], axis=1) # add prefix or suffix
            elif 'Jungle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rJungle')], axis=1) # add prefix or suffix
            elif 'Middle' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rMiddle')], axis=1) # add prefix or suffix
            elif 'ADC' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rADC')], axis=1) # add prefix or suffix
            elif 'Support' in col:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='rSupport')], axis=1) # add prefix or suffi
            else:
                df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col], prefix='r')], axis=1) # add prefix or suffix
        else:
            df = pd.concat([df.drop(col, axis=1), pd.get_dummies(df[col])], axis=1) # add prefix or suffix

    return df
```

```
def get_drag_type(drag_lst):
    return [drag[1] for drag in drag_lst]

def process_dragons(df, cols, col_names):

    drag_df_lists = []
    for col, names in zip(cols, col_names):
        df[col] = df[col].apply(ast.literal_eval).apply(get_drag_type)
        df['num_of_drags'] = df[col].apply(len)

        dict_list = df[col].apply(Counter).tolist()
        drag_df = pd.DataFrame.from_dict(dict_list)
        drag_df.fillna(0, inplace=True)
        drag_df.columns = names
        drag_df.reset_index(drop=True, inplace=True)
        drag_df_lists.append(drag_df)

    df.reset_index(drop=True, inplace=True)
    df = pd.concat([df.drop(cols, axis=1)]+drag_df_lists, axis=1)

    return df
```

```
def var_of_column(df, cols):
    for col in cols:
        df[col] = df[col].apply(ast.literal_eval).apply(np.var)

    return df
```

```
def count_num(df, cols):

    for col in cols:
        df[col] = df[col].apply(ast.literal_eval).apply(len)

    return df
```

```
tower_dict = {'TOP_LANE': {'OUTER_TURRET': 0, 'INNER_TURRET': 1, 'BASE_TURRET': 2},
              'MID_LANE': {'OUTER_TURRET': 3, 'INNER_TURRET': 4, 'BASE_TURRET': 5},
              'BOT_LANE': {'OUTER_TURRET': 6, 'INNER_TURRET': 7, 'BASE_TURRET': 8}}

def convert_to_tower_list(tower_str):

    tower_list = [0,0,0,0,0,0,0,0,0]
    for towers_destroyed in ast.literal_eval(tower_str):
        if len(towers_destroyed) > 0:
            try:
                index = tower_dict[towers_destroyed[1]][towers_destroyed[2]]
                tower_list[index] = towers_destroyed[0]
            except:
                pass

    return tower_list


def process_towers(df, cols, col_names):

    tower_df_lists = []
    for col, names in zip(cols, col_names):
        df[col] = df[col].apply(convert_to_tower_list)
        towers_df = pd.DataFrame(df[col].values.tolist(), columns=names)
        towers_df.head()
        towers_df.reset_index(drop=True, inplace=True)
        tower_df_lists.append(towers_df)

    df.reset_index(drop=True, inplace=True)
    df=pd.concat([df.drop(cols, axis=1)] + tower_df_lists, axis=1)

    return df
```

Figure 1. Functions used to process and clean data.

Data Storytelling

Upon completion of the data wrangling, the data was explored in order to find possible trends within the data. One of notable data visualizations was a heatmap showing where most of the in-game kills occurred during each year. The years 2016 and 2017 were the most insightful, showing that much of the in-game kills occurred around the center of the map, particularly near the baron and dragon pit. This makes sense as these are two highly contested objectives. The plots are shown in the figure below.

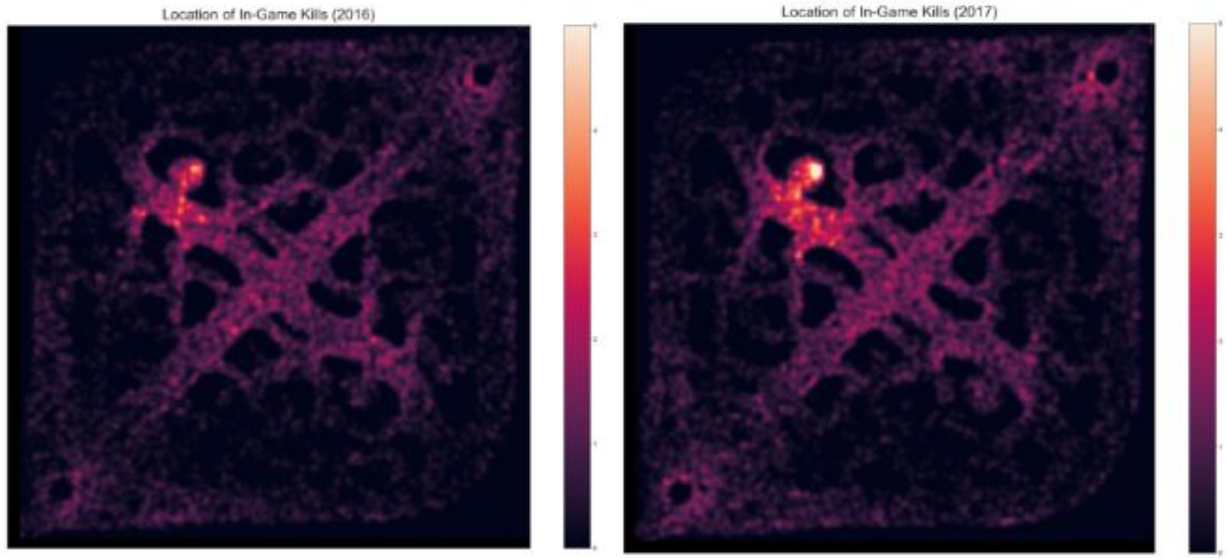

Figure 2. Heatmaps of competitive in-game kills during 2016 and 2017.

Additionally, in order to determine which champions were the most impactful within the game, which also might indicate a higher chance to win the game, the most frequently picked champs in each role per season by side was shown through a table. An example of the table is shown below.

Three Most Picked Top Champs for Blue Side

Year	Season		
2014	Summer	Maokai	21
		Ryze	18
		Rumble	13
2015	Spring	Maokai	123
		Gnar	104
		Rumble	73
	Summer	Maokai	172
		Rumble	123
		Gnar	94
2016	Spring	Poppy	200
		Nautilus	119
		Fiora	106
	Summer	Trundle	213
		Gnar	169
		Shen	166
2017	Spring	Nautilus	238
		Maokai	234
		Shen	195
	Summer	JarvanIV	220
		Renekton	204
		Shen	183
2018	Spring	Gnar	51
		Ornn	34
		Gangplank	34

Figure 3. Table depicting most frequently picked champs on the blue side for the top lane.

From these plots, it was easy to determine which champions were considered strong within their respective roles. The most picked champions were not consistent per side, which is likely a result of how the pick phase within the game occurs. The blue side picks first, which means that the most frequently chosen champions for the red side are often responses to the most frequently picked champions for the blue side.

In order to accomplish the same task of determining which champions were the most impactful within the game, the champions who had the highest ban rate were plotted. An example is shown below.

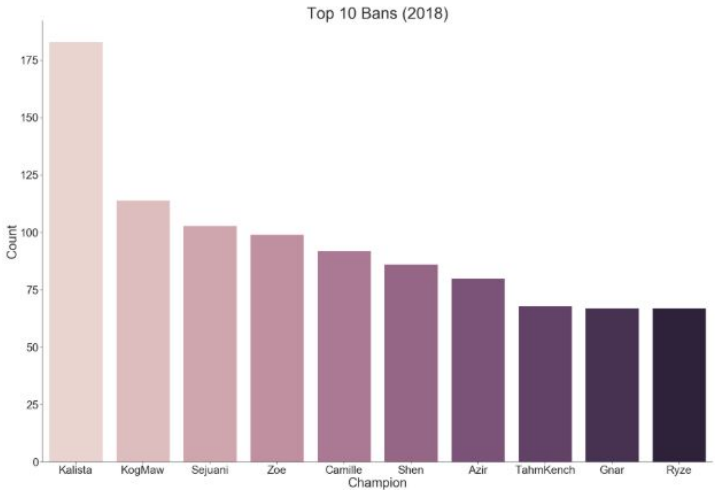

Figure 4. Countplot the 10 most frequently banned champions for 2018.

From these plots, the most frequently banned champions for each year were as follows:

1. 2014 - Alistar
2. 2015 - Kalista
3. 2016 - Nidalee
4. 2017 - Leblanc
5. 2018 - Kalista

Statistical Data Analysis

In order to determine whether some variables were indicators of whether a team would win, the p-value of the following variables were calculated using bootstrapping:

1. Gamelength
2. TIme of first tower taken
3. Number of each type of dragon
    a. Earth
    b. Ocean
    c. Air
    d. Fire
4. Mean gold of winning team by role
    a. Top
    b. Jungle
    c. Mid
    d. ADC
    e. Support

A function was defined that generalized the bootstrapping procedure. It is shown in Figure 5.

```
def calculate_p(blue, red, feature, N=10000):
    bs_mean_diff = np.empty(N)

    mean_total   = np.mean(feature)
    blue_shifted = blue - np.mean(blue) + mean_total
    red_shifted  = red - np.mean(red) + mean_total

    for i in range(N):
        bs_blue = np.random.choice(blue_shifted, size=len(league))
        bs_red = np.random.choice(red_shifted, size=len(league))
        bs_mean_diff[i] = np.mean(bs_blue) - np.mean(bs_red)

    emperical_mean_diff = np.mean(blue) - np.mean(red)
    p = np.sum(bs_mean_diff >= emperical_mean_diff) / len(gamelength_mean_diff)
    return p
```

Figure 5. Function to calculate p-value through bootstrapping.

Of the variables tested, only the null hypothesis of the difference in infernal dragons taken between the blue team and red team could be rejected using an alpha of 0.05. There was a clear difference in the number of infernal dragons the blue team retrieved when they won versus the number of infernal dragons the red team retrieved when they won.