League of Legends Ranked Analysis

Conrad Cruz

League of Legends is a massive online MOBA game with over 180 million monthly active players. It hosts a big competitive scene with Esports and professional teams around the world, hosting tournaments with approximately 6 million USD in prize pools. Outside of Esports, it's a game that is played recreationally by many casual and competitive players daily. A League of Legends match is two teams of 5 trying to destroy each other's base (aka Nexus). To accomplish this, they must destroy towers to reach the opposing base. Throughout the match the players are trying to collect gold and experience to level up their character; the more gold and experience a player collects compared to another player, the more powerful they are and makes it much easier to win. A steady source of this income comes from NPC's called "Minions" that are slain for gold and experience.

Each team member fulfills a unique role for their team. Perhaps one of the most difficult roles that exists in League is the Jungler role: They must essentially juggle between many tasks to perform well for their team. Examples include traveling to Jungle Minions (NPCs specific to the Jungler) to get gold and experience, performing flanks on the enemy to secure kills, slaying elite monsters for team-wide benefits, keep steady vision in their territory (aka warding), and destroying enemy wards to deny the enemy vision of team movement. This is all while making sure they stay on par with their team in experience and gold so they do not get behind. Mistakes can make the Jungler behind, and thus more likely the enemy team able to win.

The main question this data analysis seeks to answer is the following: Suppose we have someone looking to get into competitive League of Legends as a Jungler. What should they aim for stat wise to have the most success? The following Dataset from Kaggle (https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min) examines nearly ten-thousand ranked games from Diamond players (Diamond is the 3rd highest rank out of 8) within the first 10 minutes of the match. It contains several variables such as the winning team, amount of gold earned per team, number of minions killed per team, wards placed and destroyed per team, and much more. The fact that the data examines these points within 10 minutes of the game is crucial since the data tends to vary more the longer an average League of Legends game goes on (due to a team getting an advantage or not over the other). Starting from the beginning puts the teams on even footing and allows more clearer conclusions to be made.

To begin examining the dataset we will look at the distribution of amount of Jungle Minions slain. Due to the large dataset, we will create a smaller sample by randomly selecting 50 games out of the ten thousand. **Figure 1** and **Figure 2** show histograms of the Jungle Minions slain from both teams in the 50 random sample.

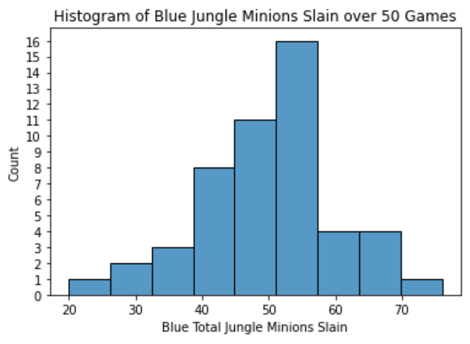


Figure 1.

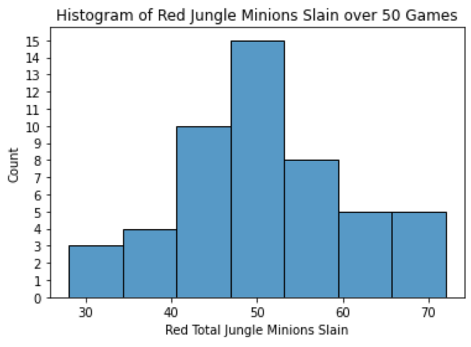
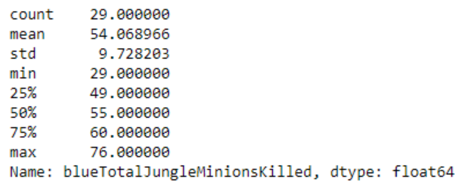


Figure 2.

We can see that with both teams the most counts lies around the ~50 mark. The lower and higher ends of minion kills can be attributed to the "snowball" nature of the game, that is, one team that gets the advantage over the other is easily able to increase their advantage more as the other team cannot contest them as easily.

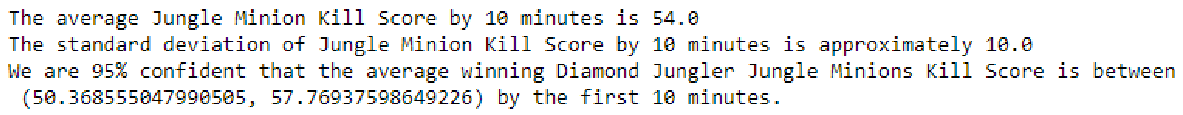
This can expand on this more by figuring out a 95% Confidence Interval overall. That is, we can figure out what the population mean is roughly at for amount of Jungle Minions Slain within 10 minutes in Diamond Games.

This would be helpful as a number for a new Jungler player to shoot for when trying to get into competitive games. First, we will take our same 50 sample dataset and divide up between games that Blue team won vs lost. This is done so we can focus on data that comes from winning games; we don't want to include data where the team lost. Next, we gather a 5-figure summary of the data, which is seen in **Figure 3.** We find that out of 50 games, 29 of them were won by blue team. Finally, we perform a confidence interval using Scipy library using the t-statistic. The result calculated is in **Figure 4.**



```
count    29.000000
mean     54.068966
std       9.728203
min      29.000000
25%      49.000000
50%      55.000000
75%      60.000000
max      76.000000
Name: blueTotalJungleMinionsKilled, dtype: float64
```

Figure 3.



```
The average Jungle Minion Kill Score by 10 minutes is 54.0
The standard deviation of Jungle Minion Kill Score by 10 minutes is approximately 10.0
We are 95% confident that the average winning Diamond Jungler Jungle Minions Kill Score is between
 (50.368555047990505, 57.76937598649226) by the first 10 minutes.
```

Figure 4.

Now since we cannot have a fraction for a Jungle Minion score, it rounds to an interval between 50-58. This might seem like a small range, but the difference of having one more minion slain can mean having enough gold to buy one more item (and therefore have an advantage over your opponent). However, is this accurate for the other team? Can we see this average in both teams in a match? We can plot a scatter plot of both blue and red team minion kills and see the winners, as seen in **Figure 5**. Each plot represents one game and the Blue Team vs Red Team's Jungler minion count, as well as who won.

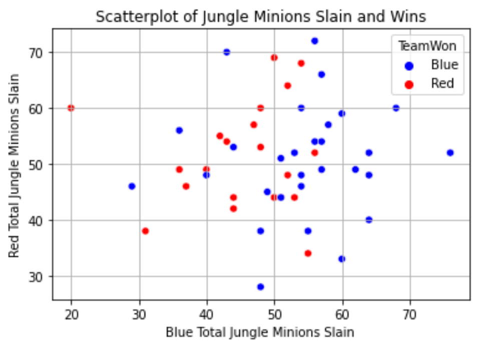


Figure 5.

This scatter plot helps to better visualize games where it was clearly sided towards one team (Example: Blue Team's 20 Score vs Red Team's 60 Score with Red winning.) More importantly, we can see that a fair number of games have Jungler Minions slain scores around 50 for both teams. Of course, there are cases of one team having higher amounts of Minion's slain, but the trend seems to be around having ~50 for both teams.

We can tighten our 95% Confidence Interval by including the rest of the dataset. Before we do that however, we will look at the quartile summary and boxplots on the entire dataset to better confirm the spread of the data for Jungle Minions killed. Setting a standard Outlier range of Q1 – 1.5(IQR) and Q3 + 1.5(IQR) can allow us to detect outliers and remove them before analyzing in our final confidence interval. This can be seen in **Figure 6** below.

```
Q1= 44.0
Median= 52.0
Q3= 59.0
IQR = 15.0
Lower Bound= 21.5
Upper Bound= 81.5
```

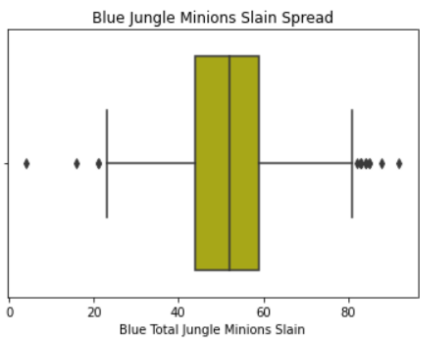
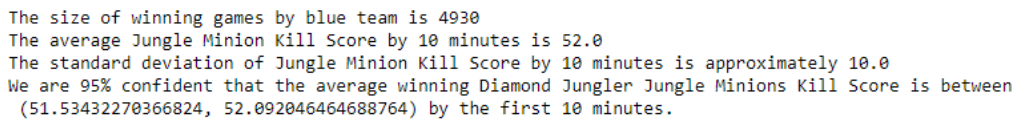


Figure 6.

With our detected outliers being points below 21.5, and above 81.5, we can remove those from our dataset. Now we can apply the larger dataset to our 95% confidence interval calculation. The output is detailed below in **Figure 7.** When we apply the larger dataset, we tighten the interval to be around 52 Jungle Minions as opposed to 50-58 Jungle Minions from before. This essentially means to try and hit around 5 minions a minute to be around the same population mean for Diamond Junglers.



```
The size of winning games by blue team is 4930
The average Jungle Minion Kill Score by 10 minutes is 52.0
The standard deviation of Jungle Minion Kill Score by 10 minutes is approximately 10.0
We are 95% confident that the average winning Diamond Jungler Jungle Minions Kill Score is between
 (51.53432270366824, 52.092046464688764) by the first 10 minutes.
```

Figure 7.

The next variable to look at is Elite Monster’s slain. In a match, there exists two special Monsters the Jungler can slay for a helpful bonus to their team: The Dragon and Herald. These are very contested objectives that require a bit of effort to slay. Ideally the Jungler should get both, but that is not always an option. If the Jungler had to choose between the two, what should a Jungler go for to better increase their chances at winning the match?

For background, each Elite Monster can be slain only once before the 10 minute marker. Thus, we know that if one team gets Dragon/Herald, the other team cannot get Dragon/Herald it until much later in the match. We will examine the entire dataset for this to get a proportion of games and examine both teams. We will exclude games where one team gets both Elite Monsters since this does not help answer our question.

We will first examine games where a team won or lost based on getting the Dragon first within 10 minutes. The Win counts of blue team getting dragon first (and thus red team did not get the dragon) will be tallied. The same tally will be applied to red team getting dragon first (and thus blue team did not get the dragon). The combined tallies will be displayed and graphed in **Figure 8.** We will repeat the same procedure for the Herald in **Figure 9.**

```
Total Games 8439
Wins based off of blue getting dragon first, red did not
 Blue    1770
Red     1096
Name: TeamWon, dtype: int64
Wins based off of red getting dragon first, blue did not
 Red     2033
Blue    1318
Name: TeamWon, dtype: int64
```

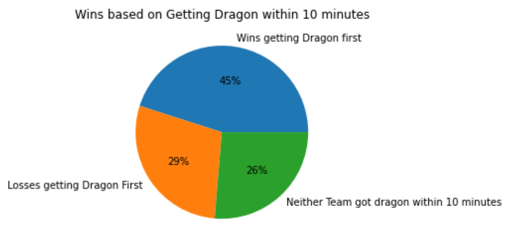


Figure 8.

```
Total Games 8439
Wins based off of blue getting Herald first, red did not
 Blue    583
Red     564
Name: TeamWon, dtype: int64
Wins based off of red getting Herald first, blue did not
 Red     447
Blue    404
Name: TeamWon, dtype: int64
```

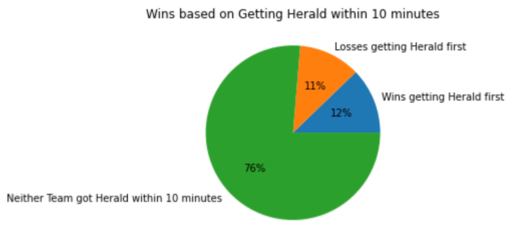


Figure 9.

The first striking feature of these two graphs is 74% (45%+29%) of the games a team acquired Dragon first compared to 23% (11%+12%) of games a team got Herald first, regardless of resulting in wins or losses. Furthermore, look at how close the win and lose ratio is between getting Herald vs the Dragon which is detailed in **Figure10**. Here we look at the win vs lose percentages when a team got Dragon or Herald within the first 10 minutes.

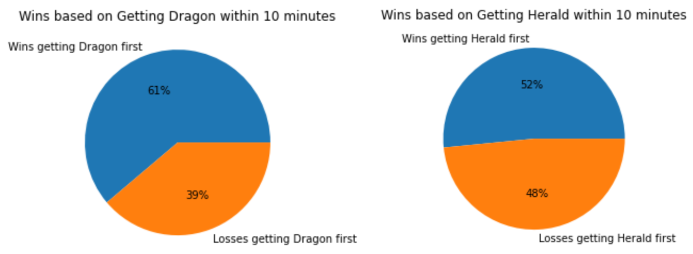


Figure 10.

There is nearly a 50-50 win or lose ratio for a team that gets the Herald first within 10 minutes compared to the 60-40 ratio for getting Dragon. Does this mean getting Herald is bad? Not necessarily. That said, seeing the large percentage of teams going for Dragon first compared to Herald, combined with the win-loss ratio for both Elite Monsters, points a trend towards getting the Dragon first is better.

This makes sense from a gameplay perspective, as the Dragon provides a team with gold upon slaying, while Herald provides a one-time use ability in the game. Therefore, we can probably conclude that a Diamond Jungler will most likely target the Dragon first within 10 minutes of a match, and so a new Jungler should probably prioritize the Dragon over the Herald.

A new Jungler would probably want to hit around 52 minions slain by 10 minutes they want to be at the same competitive level as a Diamond Player. Furthermore, they should prioritize taking the Dragon Elite Monster objective over Herald if they were to choose between the two, as the trend seems to indicate that more Diamond players go for Dragon and win percentages seem to be better than going for Herald.

These trends can also be useful for game developers as well; a game developer of League of Legends making a new Jungler character can measure how well their character meets, exceeds, or misses the population mean of minions killed, which can help indicate whether their new character needs to be made stronger or weaker. Historic trending of the data can be helpful to see changes in population mean overtime with new major updates to the game. A developer can also view the win percentages of going after either Elite Monster and possibly tweak the ratios to try and better balance the choice of getting Dragon vs Herald.

References

Largest Overall Prize Pools in Esports. (2012). Esport Earnings. Retrieved March 8, 2022, from https://www.esportsearnings.com/tournaments

League of Legends Diamond Ranked Games (10 min). (2020). Kaggle. Retrieved March 2, 2022, from https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min

Appendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as st


#CSV File was uploaded to google drive to be directly read by python code. See references page for original
location of data.
url = 'https://drive.google.com/file/d/1t1F5JVKm8I8aoEhMKRr4HMZO9thiJrCG/view?usp=sharing'
path = 'https://drive.google.com/uc?export=download&id='+url.split('/')[-2]
df = pd.read_csv(path)
print(df.columns)
print(df.shape)


teamwon = []
for i in df['blueWins']:
  if i == 1:
    teamwon.append('Blue')
  else:
    teamwon.append('Red')
df['TeamWon'] = teamwon


#Made separate datasets based on wins vs losses
dfwins = df.loc[df['blueWins']==1]
dflosses = df.loc[df['blueWins']==0]



#Made a smaller random sample dataset of 50 out of ~10k
dfsmaller=df.sample(n=50, replace=False, random_state=1)

#Split the random sample dataset
dfsmallerwins = dfsmaller.loc[dfsmaller['blueWins']==1]
dfsmallerlosses = dfsmaller.loc[dfsmaller['blueWins']==0]

#Made Index a column to better call individual games
dfsmaller.reset_index(inplace=True)
dfsmaller.head()

dfsmallerwins.reset_index(inplace=True)




#First lets drop samples that feature getting both Dragon and Herald. Exclude the index with more than 1 Elite killed
dfOneElite = df
dfMoreElite= dfOneElite.loc[(dfOneElite['blueEliteMonsters'] > 1)|(dfOneElite['redEliteMonsters']>1)]
indexlist=dfMoreElite.index
dfOneElite=dfOneElite.drop(indexlist, axis=0)
```

```
plt.xlabel(xlabel= 'Blue Total Jungle Minions Slain')
plt.title("Histogram of Blue Jungle Minions Slain over 50 Games")
plt.xticks(range(0,100,10))
plt.yticks(range(0,20,1))
sns.histplot(data=dfsmaller, x='blueTotalJungleMinionsKilled')
plt.show()
```

```
plt.xlabel(xlabel= 'Red Total Jungle Minions Slain')
plt.title("Histogram of Red Jungle Minions Slain over 50 Games")
plt.xticks(range(0,100,10))
plt.yticks(range(0,20,1))
sns.histplot(data=dfsmaller, x='redTotalJungleMinionsKilled')
plt.show()
```

```
#Examining Team v Team Jungle Minion Kills
plt.title("Scatterplot of Jungle Minions Slain and Wins")
plt.xlabel(xlabel= 'Blue Total Jungle Minions Slain')
plt.ylabel(ylabel= 'Red Total Jungle Minions Slain')
plt.grid()
sns.scatterplot(y='redTotalJungleMinionsKilled', x='blueTotalJungleMinionsKilled',data=dfsmaller,
hue='TeamWon',palette=['#0000FF','#FF0000'])
plt.show()
#We can see a lot of games are centered around ~50 minions killed for both teams.
```

```
#Examining winning team's distribution of total Jungle Minions killed with 10 minutes.
print(dfsmallerwins['blueTotalJungleMinionsKilled'].describe())
print(dfsmallerlosses['redTotalJungleMinionsKilled'].describe())
```

```
#Creating a confidence interval that the population mean of winning team Jungle Diamond Players Jungle Minion
Kill Score.
samples, col= dfsmallerwins.shape
mean =dfsmallerwins['blueTotalJungleMinionsKilled'].mean()
std =dfsmallerwins['blueTotalJungleMinionsKilled'].std()
stderr = std/(samples**0.5)
CI = st.t.interval(0.95,samples-1,mean,stderr)
print('The average Jungle Minion Kill Score by 10 minutes is', round(mean,0))
print('The standard deviation of Jungle Minion Kill Score by 10 minutes is approximately', round(std,0))
print(f'We are 95% confident that the average winning Diamond Jungler Jungle Minions Kill Score is between \n
{CI} by the first 10 minutes.')
```

```
#Examine Outliers and Spread
sns.boxplot(data=dfwins, x='blueTotalJungleMinionsKilled', color='y')
plt.xlabel(xlabel= 'Blue Total Jungle Minions Slain')
plt.title('Blue Jungle Minions Slain Spread')
plt.show()
count,mean,std, mini,q1,q2,q3,maxi= dfwins['blueTotalJungleMinionsKilled'].describe()
iqr=q3-q1
print('Q1=',q1)
print('Median=',q2)
print('Q3=',q3)
print('IQR =',iqr)
lower = q1-1.5*iqr
upper = q3+1.5*iqr
print('Lower Bound=',lower)
print('Upper Bound=',upper)
```

```
#Create Dataset to remove Outliers
dfwinsOutliers = dfwins.loc[(dfwins['blueTotalJungleMinionsKilled'] > upper)
|(dfwins['blueTotalJungleMinionsKilled'] < lower)]
indexlist = dfwinsOutliers.index
dfwinsNoOutliers = dfwins.drop(indexlist)
```

```
#Creating same confidence interval but with the entire blue team winning dataset.
samples, col= dfwinsNoOutliers.shape
mean =dfwinsNoOutliers['blueTotalJungleMinionsKilled'].mean()
std =dfwinsNoOutliers['blueTotalJungleMinionsKilled'].std()
stderr = std/(samples**0.5)
CI = st.t.interval(0.95,samples-1,mean,stderr)
print('The size of winning games by blue team is', samples)
print('The average Jungle Minion Kill Score by 10 minutes is', round(mean,0))
print('The standard deviation of Jungle Minion Kill Score by 10 minutes is approximately', round(std,0))
print(f'We are 95% confident that the average winning Diamond Jungler Jungle Minions Kill Score is between \n
{CI} by the first 10 minutes.')
```

```
#Focus on Dragons killed by both teams out of the entire sample.
blueonedragon= dfOneElite.loc[dfOneElite['blueDragons'] == 1]
blueonedragoncount= blueonedragon['TeamWon'].value_counts()
```

```
redonedragon= dfOneElite.loc[(dfOneElite['redDragons'] == 1)]
redonedragoncount= redonedragon['TeamWon'].value_counts()
samples, col= dfOneElite.shape
print('Total Games', samples )
print('Wins based off of blue getting dragon first, red did not \n', blueonedragoncount)
print('Wins based off of red getting dragon first, blue did not \n',redonedragoncount)

data = [3803/8439,2414/8439,2222/8439]

labels = ['Wins getting Dragon first','Losses getting Dragon First','Neither Team got dragon within 10 minutes']
plt.title('Wins based on Getting Dragon within 10 minutes')
plt.pie(x=data, labels=labels,autopct ='%0.0f%%')
plt.show()




data = [3803/6217,2414/6217]

labels = ['Wins getting Dragon first','Losses getting Dragon first']
plt.title('Wins based on Getting Dragon within 10 minutes')
plt.pie(x=data, labels=labels,autopct ='%0.0f%%')
plt.show()




#Focus on Heralds killed by both teams out of the entire sample.
blueoneherald= dfOneElite.loc[dfOneElite['blueHeralds'] == 1]
blueoneheraldcount= blueoneherald['TeamWon'].value_counts()

redoneherald= dfOneElite.loc[(dfOneElite['redHeralds'] == 1)]
redoneheraldcount= redoneherald['TeamWon'].value_counts()

samples, col= dfOneElite.shape
print('Total Games', samples )
print('Wins based off of blue getting Herald first, red did not \n', blueoneheraldcount)
print('Wins based off of red getting Herald first, blue did not \n',redoneheraldcount)

data = [1030/8439,968/8439,6441/8439]

labels = ['Wins getting Herald first','Losses getting Herald first','Neither Team got Herald within 10 minutes']
plt.title('Wins based on Getting Herald within 10 minutes')
plt.pie(x=data, labels=labels,autopct ='%0.0f%%')
plt.show()




data = [1030/1998,968/1998]

labels = ['Wins getting Herald first','Losses getting Herald first']
```

```
plt.title('Wins based on Getting Herald within 10 minutes')
plt.pie(x=data, labels=labels,autopct ='%0.0f%%')
plt.show()
```