Halo: The Master Chief Collection - The Impact of Game Play Stats on Win Rate

Michael Fredericks

12/12/2020

Introduction

Throughout this project, I will be looking at the impact of seven unique game play statistics on my win rate in the game Halo: The Master Chief Collection. To accomplish this, I will first be presenting some initial exploratory analysis and visualizations. After that, I will be modeling my data using Logistic Regression, measuring the performance of my model, and finally interpreting the coefficient of each variable in relation to probability of win rate.

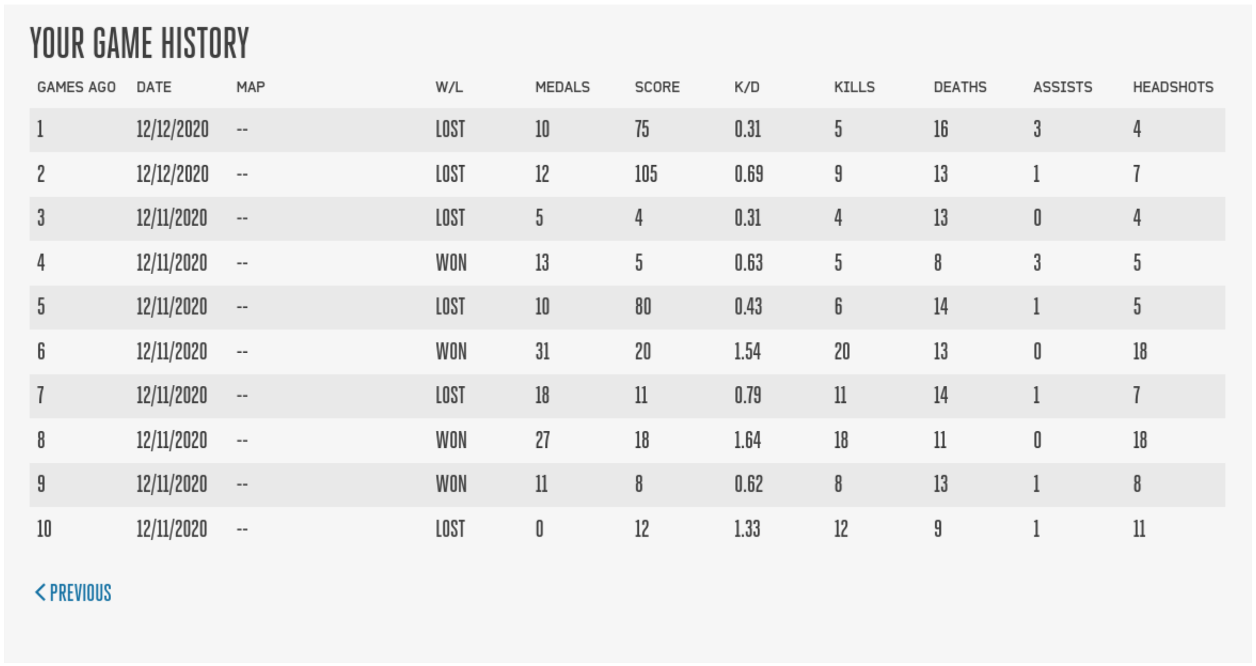
Library Imports

I am using the following libraries for my analysis. Seaborn will be our primary library for visualization and Sklearn will be used for modeling.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

Data

The data I will be using is accessible to me through Halo Waypoint. If you play Halo, and would like to see your own game history you can access your match history by clicking Games > Halo: The Master Chief Collection > Game History. Once there, you will find a table on the bottom of the page that looks like this:


YOUR GAME HISTORY

GAMES AGO	DATE	MAP	W/L	MEDALS	SCORE	K/D	KILLS	DEATHS	ASSISTS	HEADSHOTS
1	12/12/2020	--	LOST	10	75	0.31	5	16	3	4
2	12/12/2020	--	LOST	12	105	0.69	9	13	1	7
3	12/11/2020	--	LOST	5	4	0.31	4	13	0	4
4	12/11/2020	--	WON	13	5	0.63	5	8	3	5
5	12/11/2020	--	LOST	10	80	0.43	6	14	1	5
6	12/11/2020	--	WON	31	20	1.54	20	13	0	18
7	12/11/2020	--	LOST	18	11	0.79	11	14	1	7
8	12/11/2020	--	WON	27	18	1.64	18	11	0	18
9	12/11/2020	--	WON	11	8	0.62	8	13	1	8
10	12/11/2020	--	LOST	0	12	1.33	12	9	1	11

< PREVIOUS

The seven game stats I will be looking at are “MEDALS”, “SCORE”, “K/D”, “KILLS”, “DEATHS”, “ASSISTS”, & “HEADSHOTS”. The column labeled “W/L” is where I can see whether or not I won the game. This is the column we are trying to predict. Below is a brief explanation of each stat:

- Medals - These are recognitions for various types of plays. While getting a kill isn’t going to get you a medal, getting two kills within five seconds of each other gets you a “Double Kill” medal. There are various types of medals for different types of plays. Generally speaking, medals are positive.
- Score - Depending on the format of game you are playing, different things are worth different amounts of points. Again, score is a positive stat.
- K/D - This is the Kill/Death ratio. It is simply the number of kills you get divided by the number of deaths you have. This is an important metric that looks at the relationship between those two other stats.
- Kills - This is the number of players on the opposing team that you kill in the game.
- Deaths - This is the number of times you die in the game. You can die due to players on the other team killing you, a teamate accidentally killing you, or the unfortunate rare event of you falling off the map and killing youself.
- Assists - This is the number of teammate kills that you had a hand in. You may have been shooting an opposing player and causing damage, but the final blow was done by another teammate - This would be considered an asssist.
- Headshots - The number of times the last shot you made on an opposing player was a headshot and resulted in a kill.

In order to use this data for my analysis, I first copied it into a spreadsheet (I used Libre Office Calc). As you will see in my code and outputs below, the format of the data once copied into a spreadsheet was a single vertical column. I want to be looking at this data in a Data Frame format, so there is some manipulation that I have to do once the .csv file is read into my code.

```
raw_data = pd.read_csv('halowaypoint_data.csv', )
raw_data
```

```
##           Data
## 0            1
## 1     12/12/20
## 2           --
## 3         Lost
## 4           10
## ...        ...
## 1007      2.11
## 1008        19
## 1009         9
## 1010         0
## 1011        18
##
## [1012 rows x 1 columns]
```

Before I can get this into the correct format for a Data Frame, I will first need to change the data to a list format. I will then create a list of lists using a for loop, put it all into a dictionary with the column names I want as keys before finally putting back into Data Frame format.

```
raw_data = raw_data['Data'].to_list()

games_ago = []
date = []
game_map = []
win_loss = []
medals = []
score = []
k_d = []
kills = []
deaths = []
assists = []
headshots = []

keys = [games_ago, date, game_map, win_loss, medals, score, k_d, kills, deaths, assists, headshots]

count = 0
for i in keys:
    if len(i) == 0:
        i.append(raw_data[count::11])
    count += 1

data_dict = {
    'games_ago' : games_ago[0],
    'date'      : date[0],
    'game_map'  : game_map[0],
    'win_loss'  : win_loss[0],
    'medals'    : medals[0],
    'score'     : score[0],
    'k_d'       : k_d[0],
    'kills'     : kills[0],
    'deaths'    : deaths[0],
    'assists'   : assists[0],
    'headshots' : headshots[0]
}

data_df = pd.DataFrame.from_dict(data_dict)
data_df = data_df[['win_loss', 'medals', 'score', 'k_d', 'kills', 'deaths', 'assists', 'headshots']]
data_df.head()
```

```
##   win_loss medals score   k_d kills deaths assists headshots
## 0     Lost     10    75  0.31     5     16       3         4
## 1     Lost     12   105  0.69     9     13       1         7
## 2     Lost      5     4  0.31     4     13       0         4
## 3      Won     13     5  0.63     5      8       3         5
## 4     Lost     10    80  0.43     6     14       1         5
```

I am also going to create a Win/Loss column with binary values and reformat all dtypes as either strings, integers, or floats.

```
data_df['win_loss_binary'] = data_df['win_loss'].apply(lambda x: 0 if x == 'Lost' else (1 if x == 'Won'
data_df['win_loss'] = data_df['win_loss'].apply(lambda x: str(x))
data_df['k_d'] = data_df['k_d'].astype(float)
data_df[['medals', 'score', 'kills', 'deaths', 'assists', 'headshots', 'win_loss_binary']] = \
    data_df[['medals', 'score', 'kills', 'deaths', 'assists', 'headshots', 'win_loss_binary']].astype(in
data_df.head()
```

```
##   win_loss  medals  score   k_d  ...  deaths  assists  headshots  win_loss_binary
## 0     Lost      10     75  0.31  ...      16        3          4                0
## 1     Lost      12    105  0.69  ...      13        1          7                0
## 2     Lost       5      4  0.31  ...      13        0          4                0
## 3      Won      13      5  0.63  ...       8        3          5                1
## 4     Lost      10     80  0.43  ...      14        1          5                0
##
## [5 rows x 9 columns]
```

Lastly, I will create a seperate Data Frame object for my first analysis visual - a barplot comparing Wins and Losses.

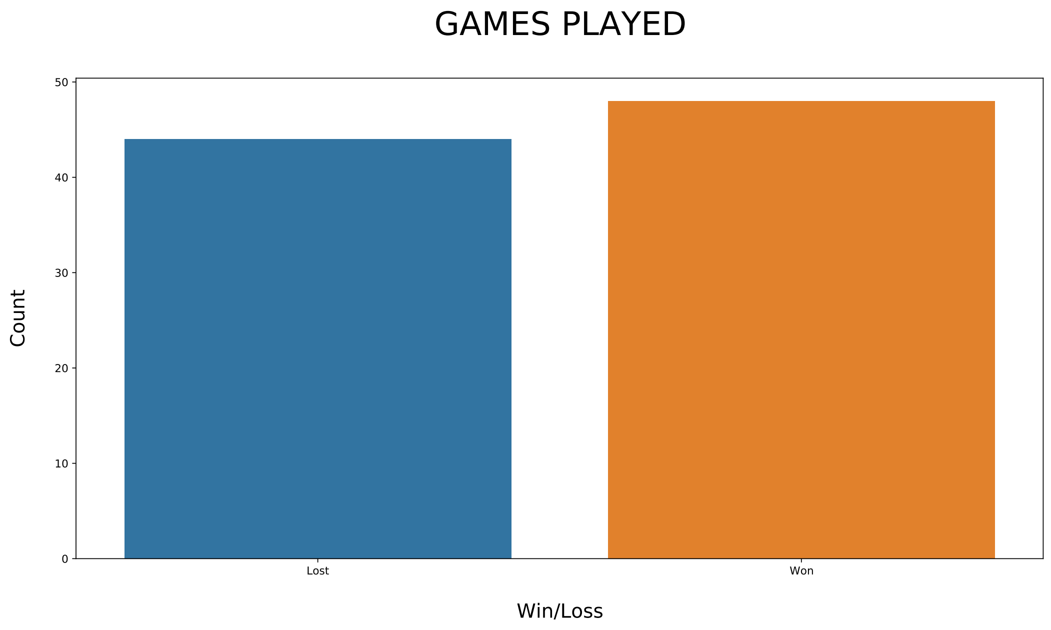
```
barplot_df = pd.DataFrame([['Lost', data_df[data_df['win_loss'] == 'Lost']['win_loss'].count()],
                           ['Won', data_df[data_df['win_loss'] == 'Won']['win_loss'].count()],],
                         columns = ['Win/Loss', 'Count'])
barplot_df.head()
```

```
##   Win/Loss  Count
## 0     Lost     44
## 1      Won     48
```

Visual Analysis

Like previously mentioned, the first visualization I will present is a simple bar graph comparing the total number of wins to total number of losses.

```
plt.figure(figsize=(16,8))
sns.barplot(x = 'Win/Loss', y = 'Count', data = barplot_df)
plt.title('\nGAMES PLAYED\n', fontsize = 30)
plt.xlabel('\nWin/Loss\n', fontsize = 18)
plt.ylabel('\nCount\n', fontsize = 18)
```



As shown in my plot, of the 92 games played, I won 48 and lost 44, for a slightly positive win rate. What I am really interested in, however, is how each of the seven stats recorded impact my win rate, so that I might be able to focus on certain aspects of my game in order to improve my win rate.

Therefore, it is important that I look at each variable in my dataset. I will provide a summary of descriptive statistics as well as a histogram for each of the seven game stats.

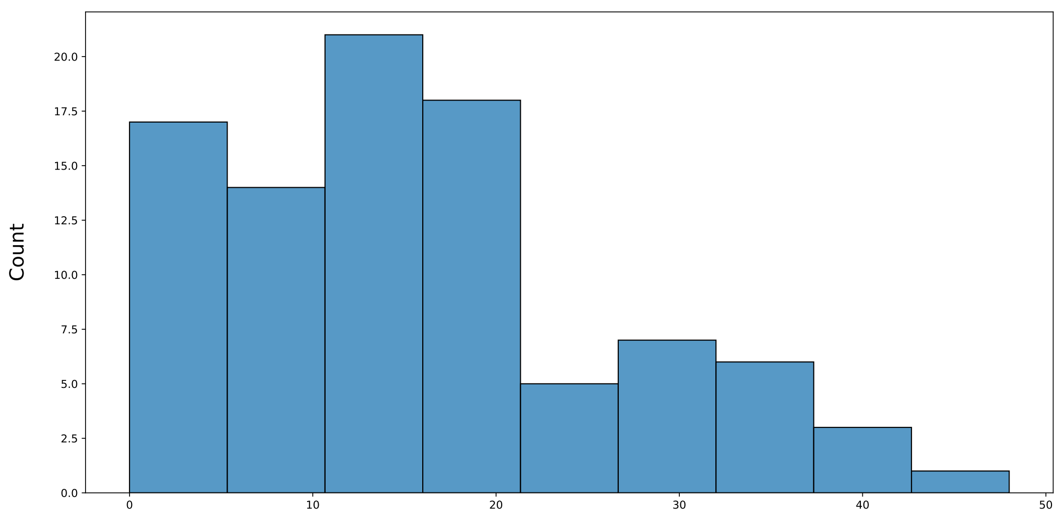
```
plot_cols = ['medals', 'score', 'k_d', 'kills', 'deaths', 'assists', 'headshots']
for col in plot_cols:
    print(col.upper());
    print();
    print(data_df[col].describe());
    print();

    plt.figure(figsize=(16,8))
    sns.histplot(data_df[col], kde = False)
    plt.xlabel('')
    plt.ylabel('\nCount\n', fontsize = 18)
    plt.title('\n' + col.upper() + ' Distribution\n', fontsize = 24)
    plt.show();

    print();
    print('--------------------------------------------------------------------------------------------------------------
    print();
```

```
## MEDALS
##
```

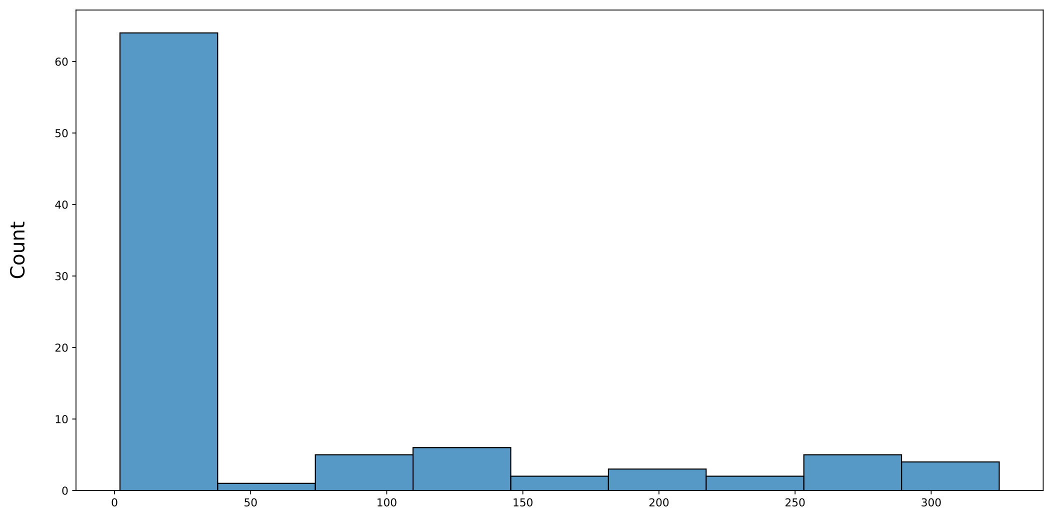
```
## count    92.000000
## mean     15.847826
## std      10.717076
## min       0.000000
## 25%       8.750000
## 50%      14.500000
## 75%      21.000000
## max      48.000000
## Name: medals, dtype: float64
##
##
## ----------------------------------------------------------------------------------------------------
##
## SCORE
##
## count     92.000000
## mean      63.228261
## std       92.211308
## min        2.000000
## 25%        8.000000
## 50%       12.000000
## 75%       97.500000
## max      325.000000
## Name: score, dtype: float64
##
##
## ----------------------------------------------------------------------------------------------------
##
## K_D
##
## count    92.000000
## mean      1.023370
## std       0.886142
## min       0.140000
## 25%       0.492500
## 50%       0.755000
## 75%       1.315000
## max       6.500000
## Name: k_d, dtype: float64
##
##
## ----------------------------------------------------------------------------------------------------
##
## KILLS
##
## count    92.000000
## mean     11.369565
## std       5.737322
## min       2.000000
## 25%       7.000000
## 50%      10.000000
## 75%      15.000000
## max      26.000000
## Name: kills, dtype: float64
```

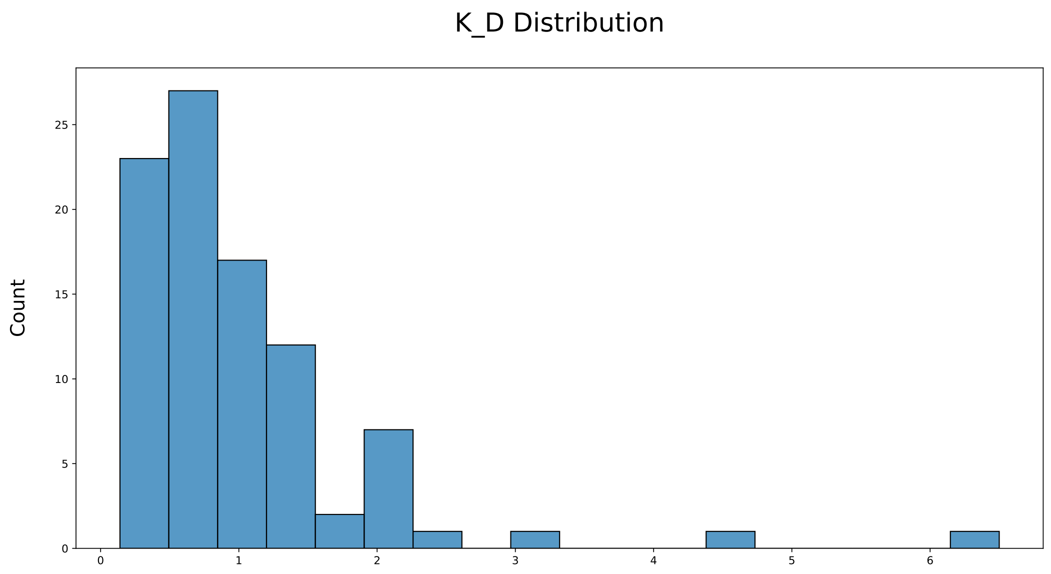
```
##
##
## -----------------------------------------------------------------------------------------------------
##
## DEATHS
##
## count    92.000000
## mean     13.250000
## std       4.386856
## min       4.000000
## 25%      10.000000
## 50%      13.000000
## 75%      16.000000
## max      26.000000
## Name: deaths, dtype: float64
##
##
## -----------------------------------------------------------------------------------------------------
##
## ASSISTS
##
## count    92.000000
## mean      1.010870
## std       1.346502
## min       0.000000
## 25%       0.000000
## 50%       0.000000
## 75%       2.000000
## max       7.000000
## Name: assists, dtype: float64
##
##
## -----------------------------------------------------------------------------------------------------
##
## HEADSHOTS
##
## count    92.000000
## mean     10.119565
## std       5.326365
## min       0.000000
## 25%       7.000000
## 50%       9.000000
## 75%      13.250000
## max      25.000000
## Name: headshots, dtype: float64
##
##
## -----------------------------------------------------------------------------------------------------
```

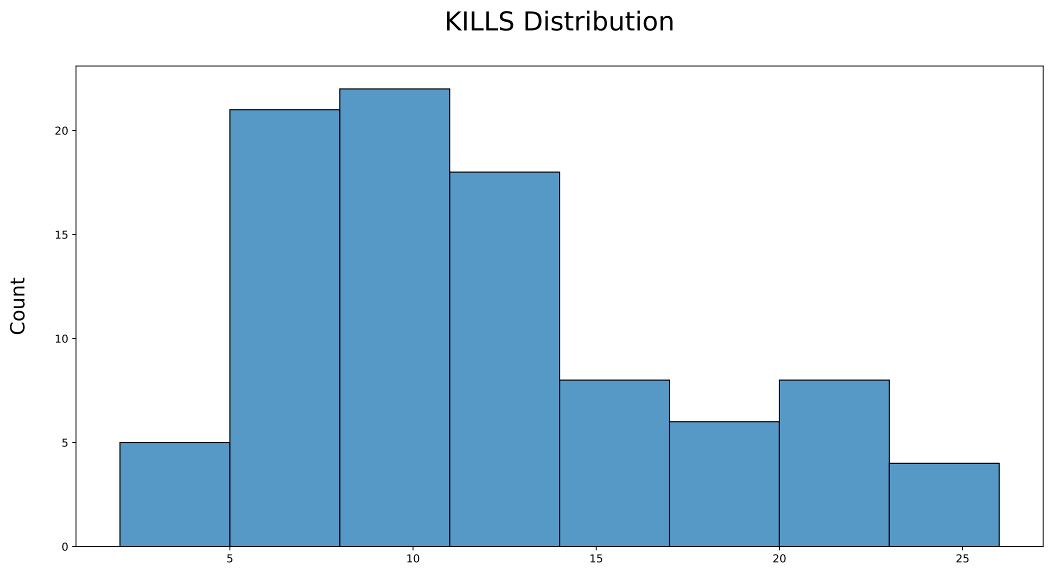
MEDALS Distribution

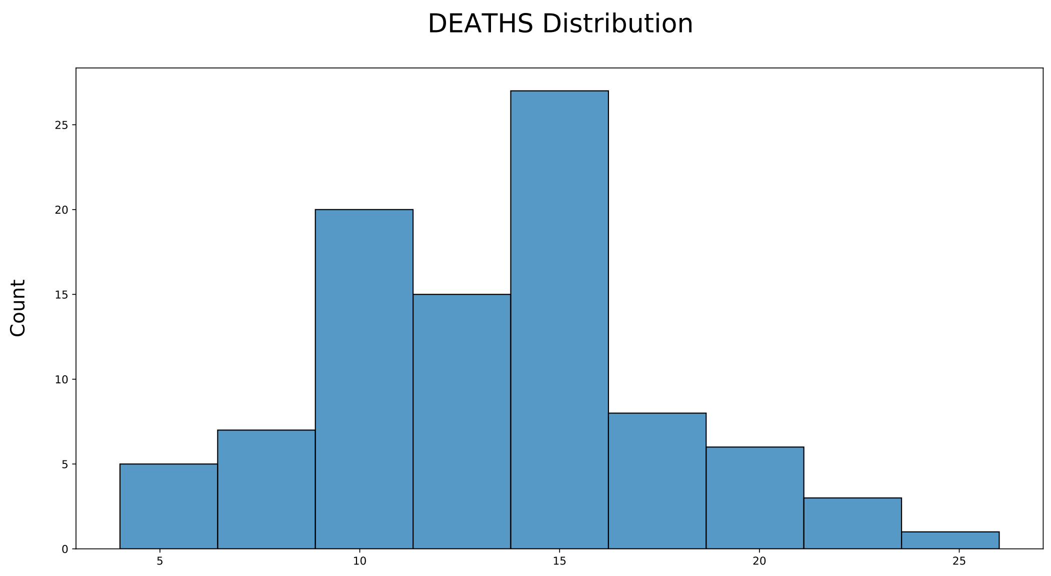


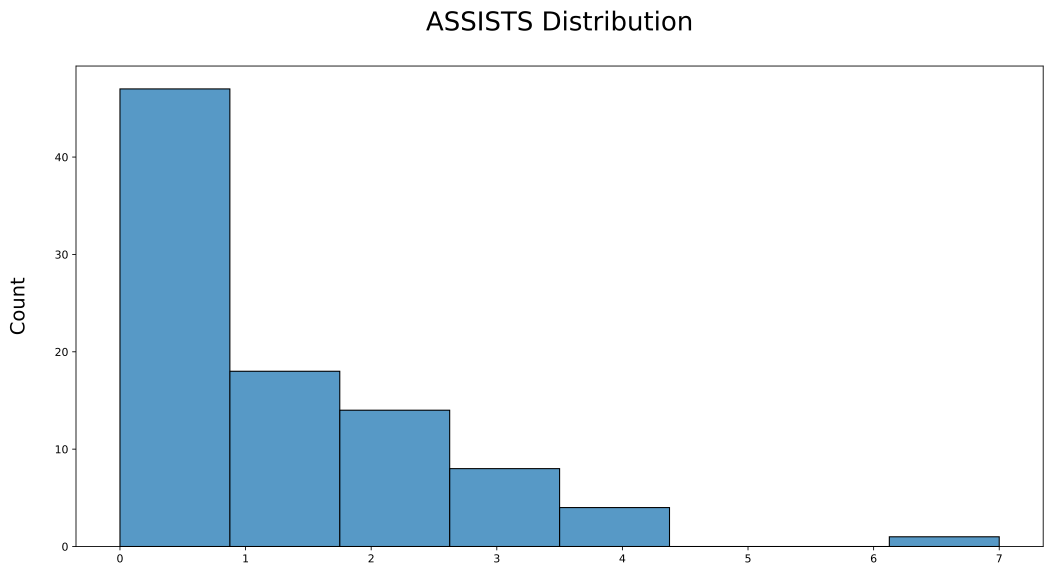
SCORE Distribution

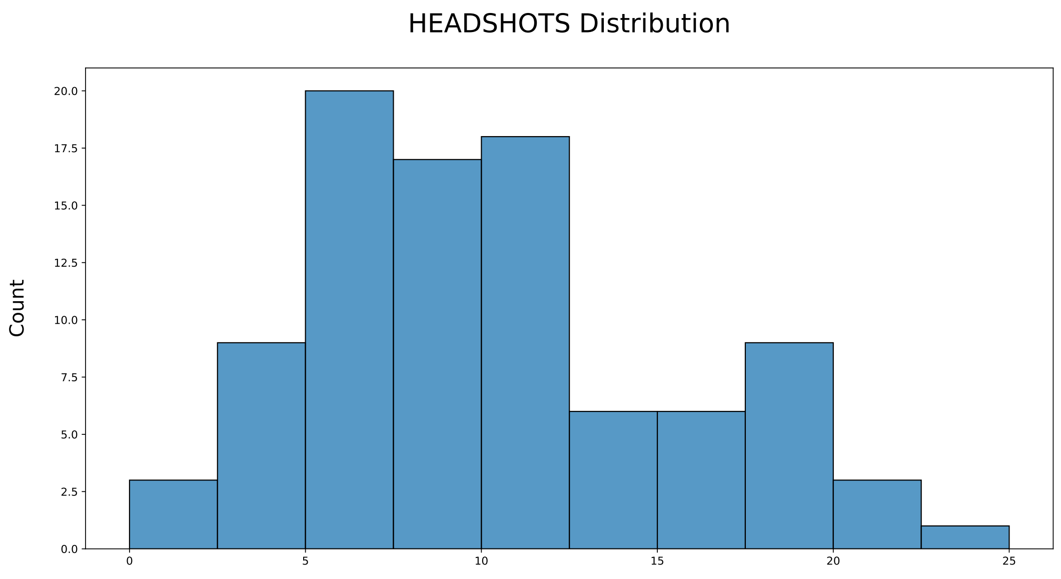



K_D Distribution
Count
0
5
10
15
20
25
0
1
2
3
4
5
6


KILLS Distribution
Count
0
5
10
15
20
5
10
15
20
25


DEATHS Distribution
Count
0
5
10
15
20
25
5
10
15
20
25

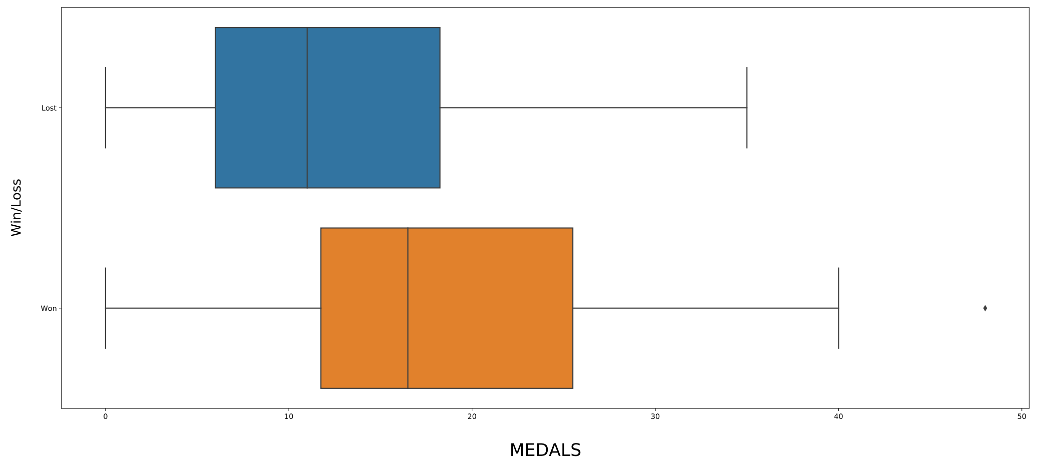

ASSISTS Distribution
Count
0
10
20
30
40
0
1
2
3
4
5
6
7

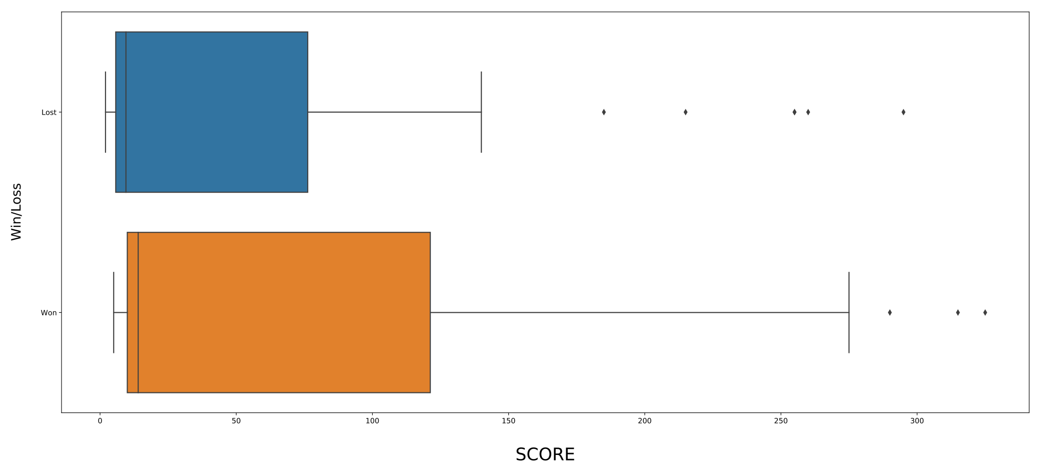


The descriptive statistics and histograms for each game stat make sense. As you can see in the plots, many of the distributions crowd around the lower values, indicating my consistently average game play. For each stat, though, there are some higher values that show up now and then, mostly indicating when I'm having a good game. The only instance where this is not the case is when there are high values in the 'Deaths' stat. I like to think that in games when I am dying a lot, it's because the players on the other team must be semi-professional and very good, rather than myself being just plain horrible at the game.

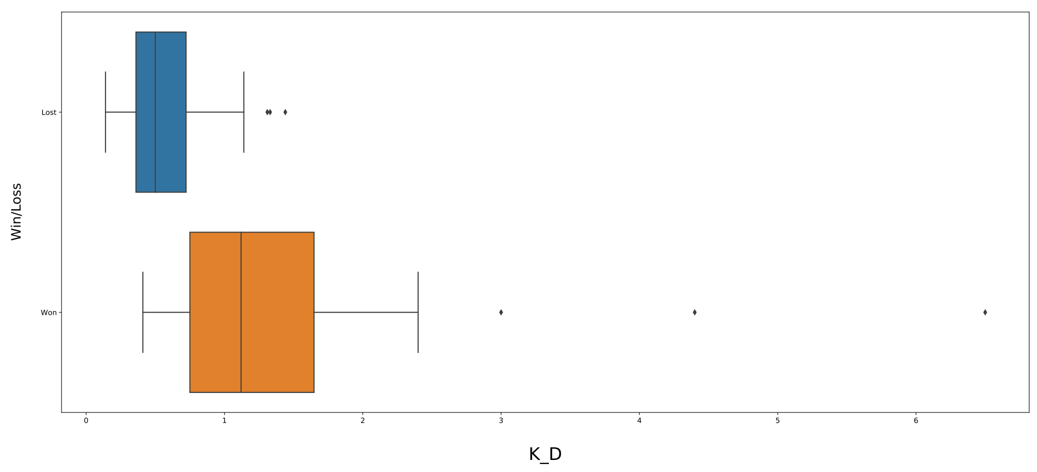
I want to look at the distribution of these stats again. However, this time I will separate the games I won from the games I lost. To show this in a visually pleasing way, I will use boxplots rather histograms.

```
for col in plot_cols:
    plt.figure(figsize=(24,10))
    sns.boxplot(x = col, y = 'win_loss', data = data_df)
    plt.xlabel('\n' + col.upper() + '\n', fontsize = 24)
    plt.ylabel('\nWin/Loss\n', fontsize = 18)
    plt.show();
```

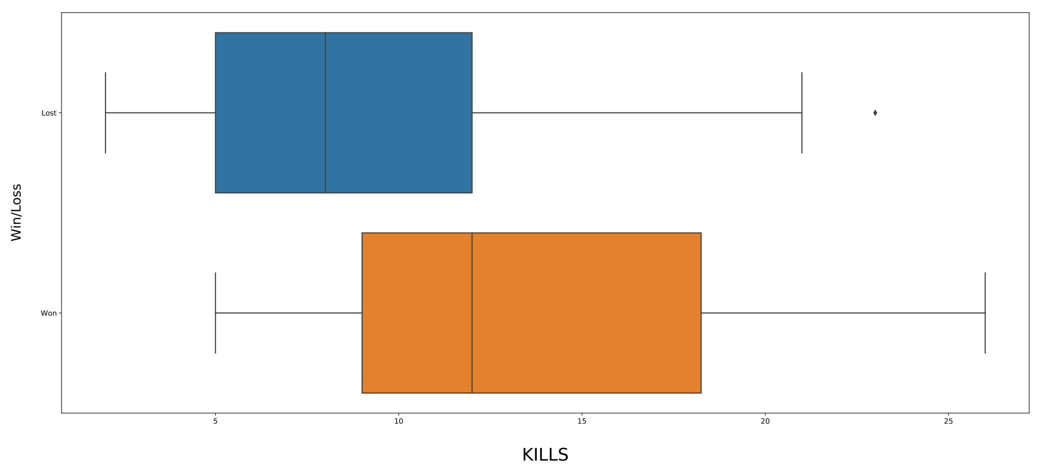




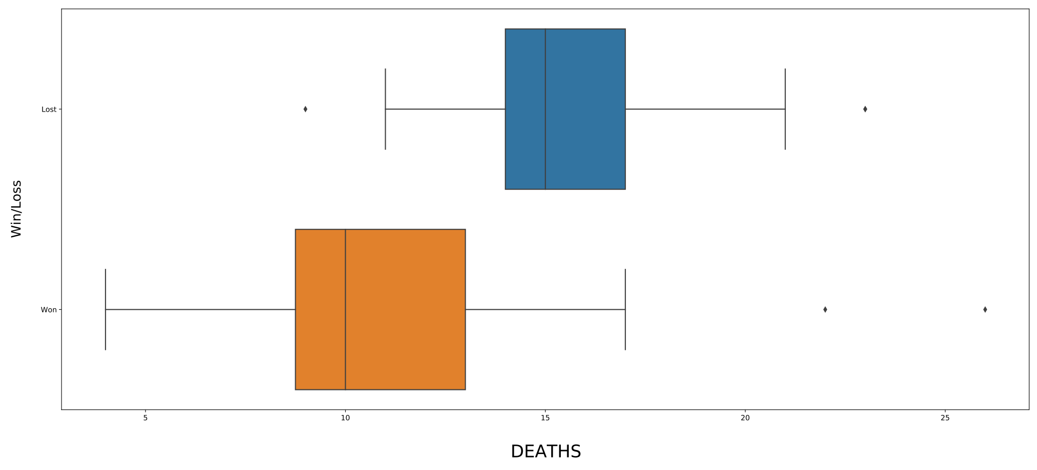
Win/Loss
Lost
Won
0
50
100
150
200
250
300
SCORE

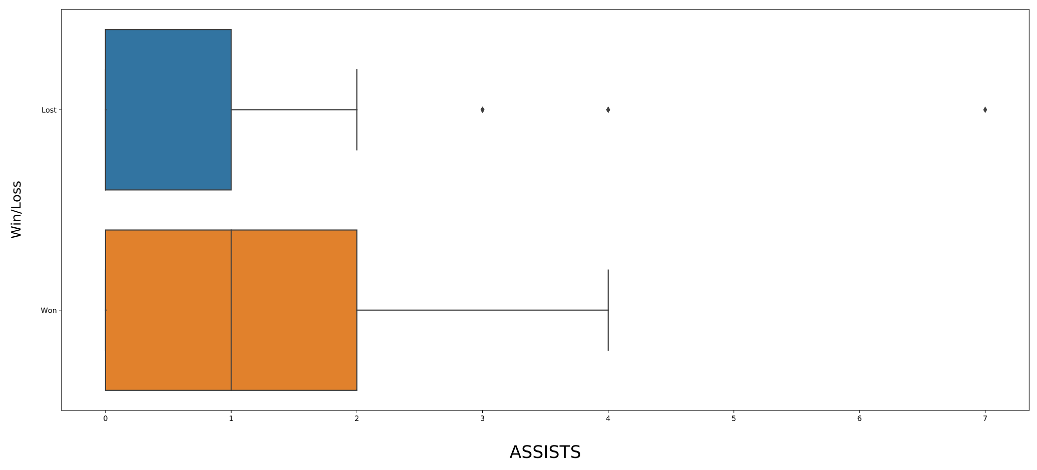


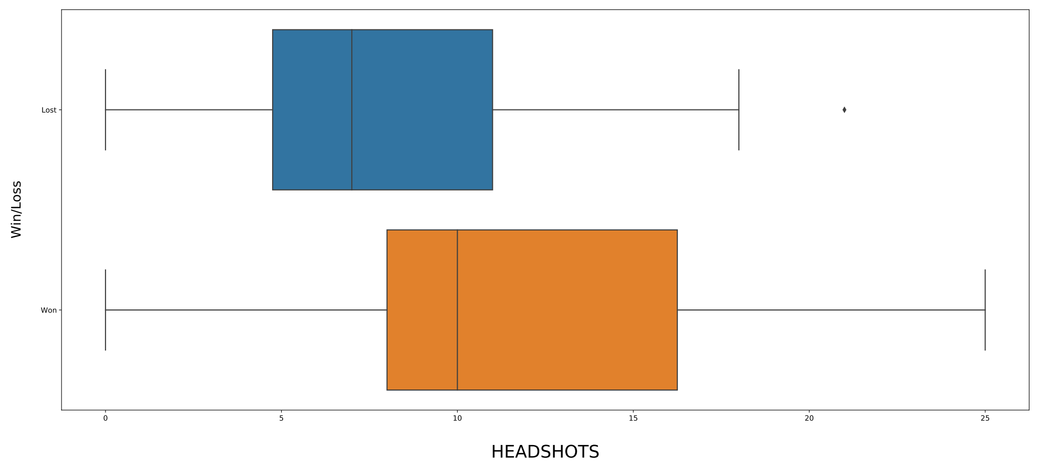
Win/Loss
Lost
Won
0
1
2
3
4
5
6
K_D



Win/Loss
Lost
Won
5
10
15
20
25
KILLS







This is where I start to see some of the stats having more of an impact on win rate than others. Take a look at the “K_D” and “Deaths” plots.

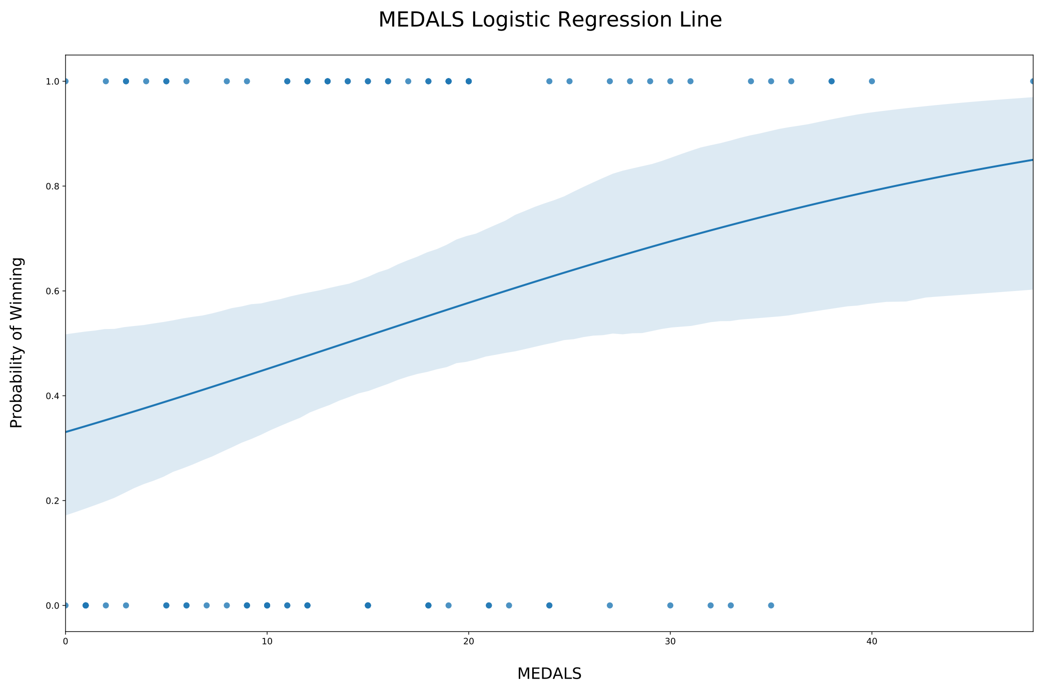
For Kill/Death Rate, the IQR for games I won is almost completely outside the IQR for games I lost. There were obviously some cross-overs in the games that fell within the 1st quartile (for Games Won) and 4th quartile (for Games Lost), but the IQR is a good representation of the data set as a whole, so I believe this is one area that will have a large impact on the Win Rate.

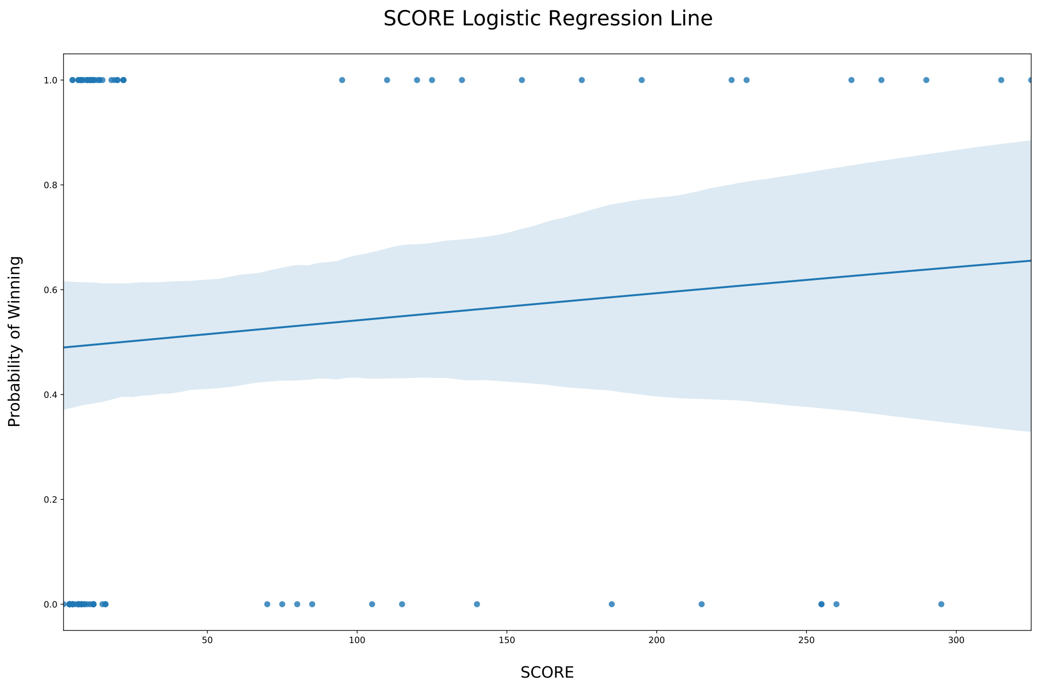
The same distinction can also be seen in the "Deaths" distribution, however the readings are opposite. The IQR for each outcome's distribution are completely separate from each other. In this case, there were much higher values for deaths in the games lost than in the distribution of games won.

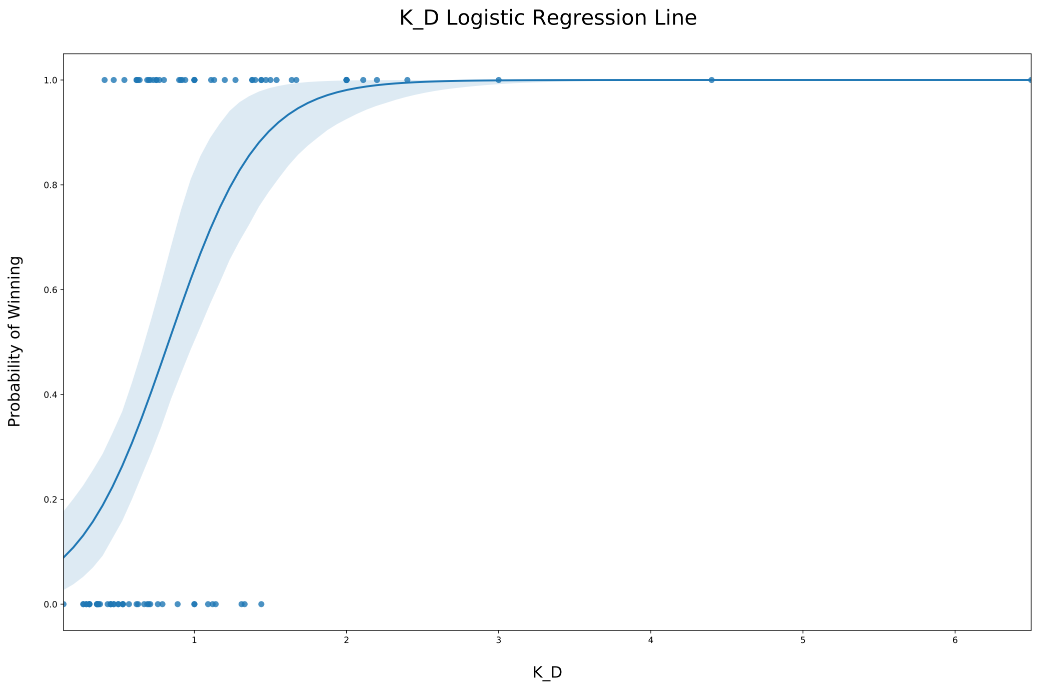
The last visualization exercise I will present before beginning the modeling portion of my project is graphing the logistic regression lines of each variable using seaborn's regplot feature.

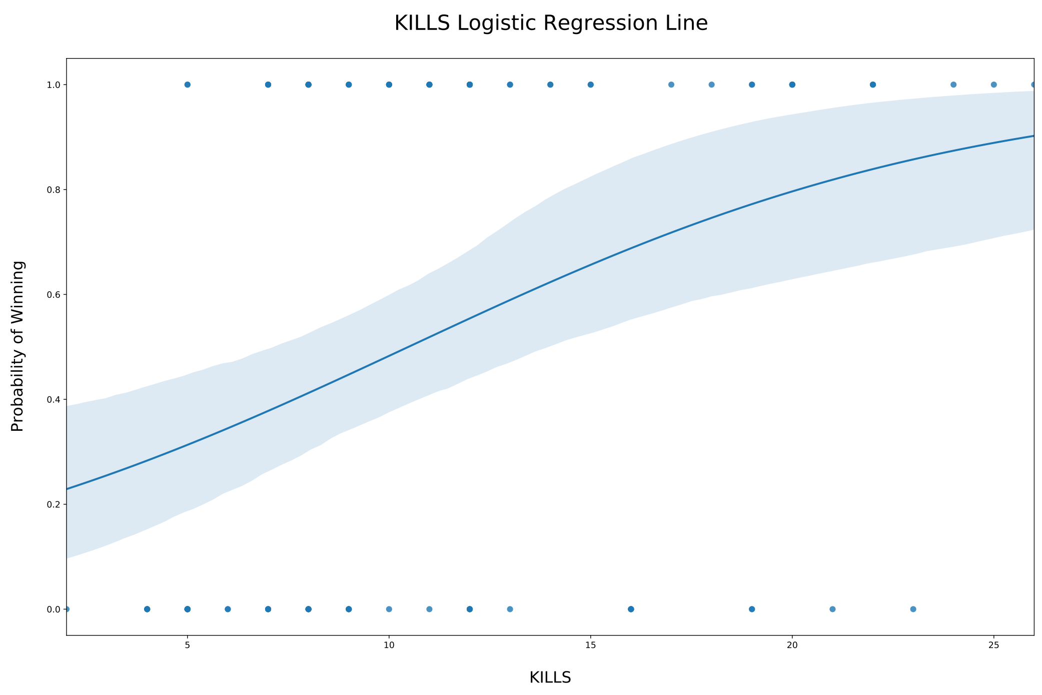
```
for col in plot_cols:
    plt.figure(figsize=(20,12))
    sns.regplot(x = col, y = 'win_loss_binary', data = data_df, logistic = True)
    plt.title('\n' + col.upper() + ' Logistic Regression Line\n', fontsize = 24)
    plt.xlabel('\n' + col.upper() + '\n', fontsize = 18)
    plt.ylabel('\nProbability of Winning\n', fontsize = 18)
    plt.show();
```

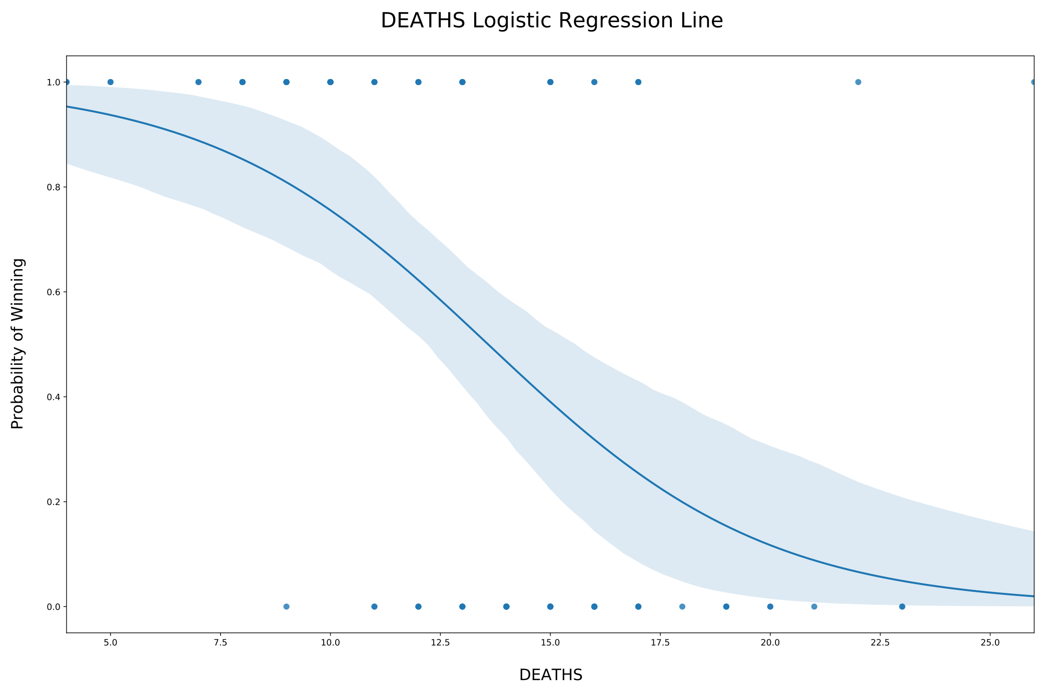
```
## /home/michaelpfredericks/.local/share/r-miniconda/envs/r-reticulate/lib/python3.6/site-packages/stats
##   n_endog_mu = self._clean((1. - endog) / (1. - mu))
## /home/michaelpfredericks/.local/share/r-miniconda/envs/r-reticulate/bin/python:2: RuntimeWarning: Mo
```

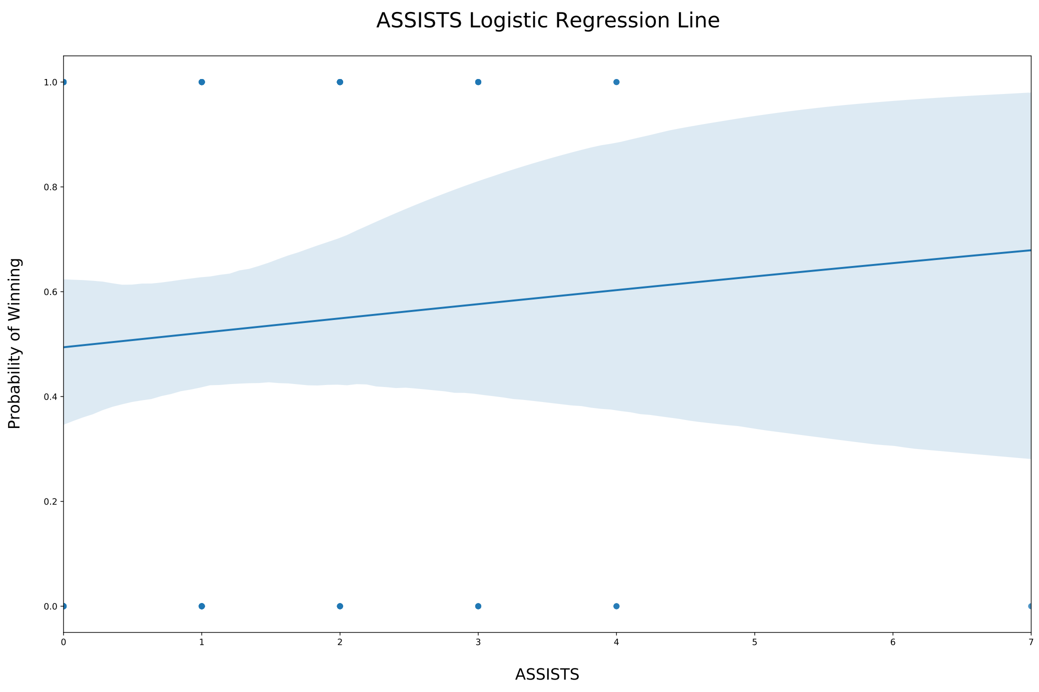


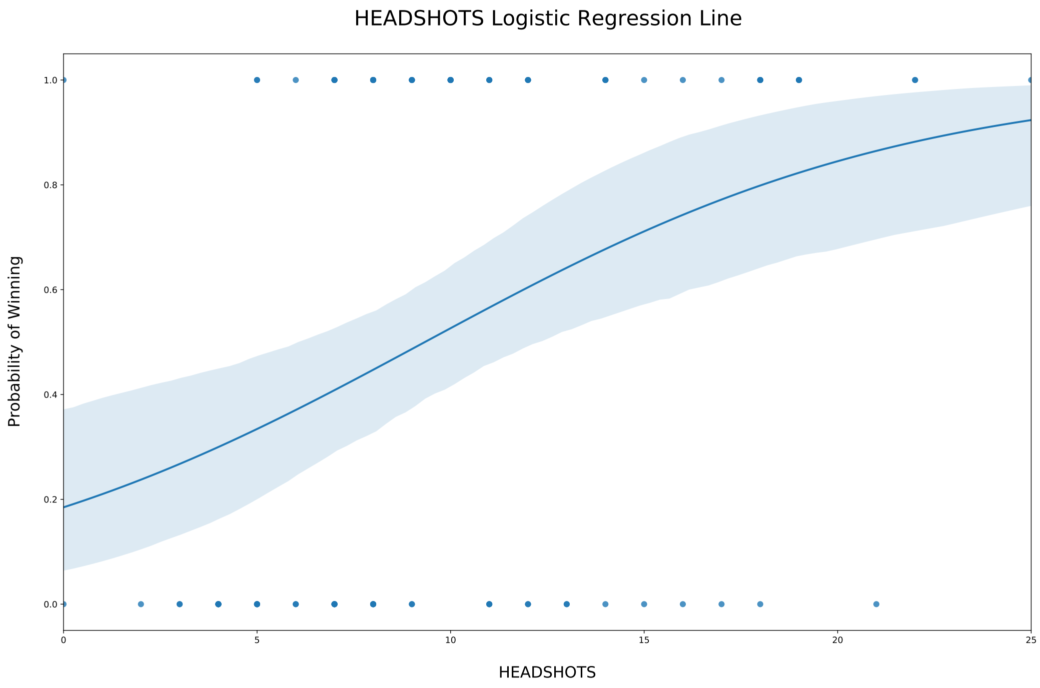

SCORE Logistic Regression Line
Probability of Winning
SCORE
1.0
0.8
0.6
0.4
0.2
0.0
50
100
150
200
250
300


K_D Logistic Regression Line
Probability of Winning
K_D
1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6


KILLS Logistic Regression Line
Probability of Winning
KILLS
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
20
25


DEATHS Logistic Regression Line
Probability of Winning
DEATHS
0.0
0.2
0.4
0.6
0.8
1.0
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0





In case this is the first time you are seeing a graph like this, I will briefly summarize how a Logistic Regression Plot works.

Along the X axis is the independent variable that you are using to determine the likelihood of the dependent variable occurring, which is located on the Y axis. This likelihood is measured as a probability between 0 and 1, or 0% to 100%. The plotted samples will either lie on Y = 0 or Y = 1, due to the dependent variable being binary. In this case, 0 is a Loss and 1 is a Win. There is no in between, so all points must like on these two lines. The "S" shaped line you see going through the center of the plot is modeled regression line between the two points. This is where you are able to read your dataset with claims like:

Referencing the "Headshots Logistic Regression Line" plot, I can see that if I play a game and can score 10 headshots, I have a 50% probability of winning the game.

The shaded area surrounding the line represents the confidence interval, meaning that in the same example as referenced above, that 50% probability is a target but in reality with 10 headshots I have anywhere between a 40% - 60% chance of winning the game.

As you can see, some variables are harder to measure accurately - hence you will see my wider confidence intervals. The "Assists" plot is a good example of this.

Like I had assumed earlier, we can see the most distinct models for the "K_D" and "Death" stats. For Kill/Death ratio, I can start to see probabilities of winning the game over 80% once I hit a 1.25 K/D ratio. For Deaths, I can start to see probabilities of losing the games around 80% once I hit 18 deaths.

The models shown here are all independent of each other, so the plots you see above only model the relationship between the single independent and dependent variables. For the modeling portion of the this project, I will be looking at the relationship between all the dependent variables and the independent variable. This is known as multinomial logistic regression modeling.

Modeling

For this part of my project, I will be creating a multinomial logistic regression model. The process is relatively simple.

First, I take my dataset and split into into two subsets - a training subset and a testing subset. The testing subset I'm using will be 10% of the total dataset. This ensures that I have enough data to train the model.

Then I will be taking the training subset and passing it through the machine learning algorithm along with it's actual outcomes. By doing this, the model will create a rule to determine whether new data passed in will be categorized as a win or a loss. This rule is basically an equation, which gives coefficients to each variable I am including in my training dataset.

Finally, I will use the model created to run our test subset of data and measure the accuracy of my model against the actual results. At this time, I will also do a brief analysis of the different coefficients my model picked for each variable.

Below is the code used to create the model and predict the results of my test dataset.

```
X = data_df.drop(['win_loss', 'win_loss_binary'], axis = 1)
y = data_df['win_loss_binary']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=11)

log_model = LogisticRegression(solver = 'lbfgs', multi_class = 'multinomial', C = 10.0, max_iter = 1000
log_model.fit(X_train, y_train)
```

```
## LogisticRegression(C=10.0, max_iter=1000, multi_class='multinomial')
```

```
predictions = log_model.predict(X_test)
```

Now I will measure the models performance. To do this I will plot a confusion matrix, and include four metrics for performance: Accuracy, Precision, Recall, and F1 Score.

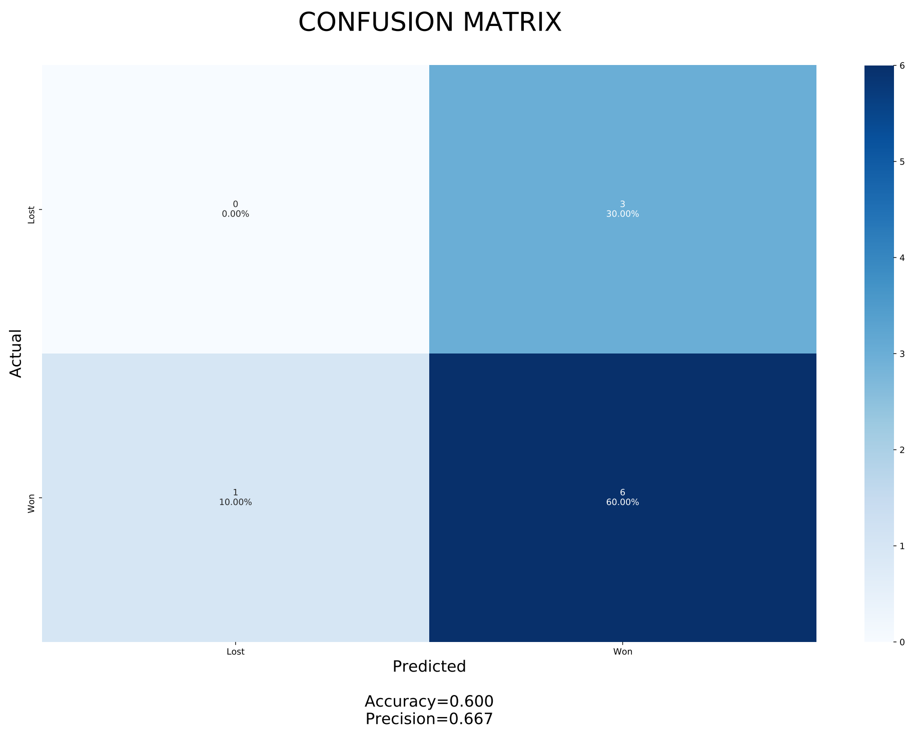
Accuracy is the most intuitive performance measure. It is the ratio of correctly predicted observation to the total observations.

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate.

Recall is the ratio of correctly predicted positive observations to the all observations in actual class - Wins.

F1 Score is the weighted average of Precision and Recall.

```
cf_matrix = confusion_matrix(y_test, predictions)

categories = ['Lost', 'Won']

group_counts = [f'{value:0.0f}' for value in
                cf_matrix.flatten()]
group_percentages = [f'{value:.2%}' for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}'.strip() for v1, v2 in
          zip(group_counts, group_percentages)]
labels = np.asarray(labels).reshape(2,2)

accuracy = np.trace(cf_matrix) / float(np.sum(cf_matrix))
precision = cf_matrix[1,1] / sum(cf_matrix[:,1])
recall = cf_matrix[1,1] / sum(cf_matrix[1,:])
f1_score = 2*precision*recall / (precision + recall)
stats_text = (f'\n\nAccuracy={accuracy:0.3f}\nPrecision={precision:0.3f}\nRecall={recall:0.3f}\nF1 Score

plt.figure(figsize=(20,12))
sns.heatmap(cf_matrix, annot = labels, fmt = '', xticklabels = categories, yticklabels = categories, cma
plt.ylabel('Actual', fontsize = 18)
plt.xlabel('Predicted' + stats_text, fontsize = 18)
plt.title('\nCONFUSION MATRIX\n', fontsize = 30)
plt.show();
```



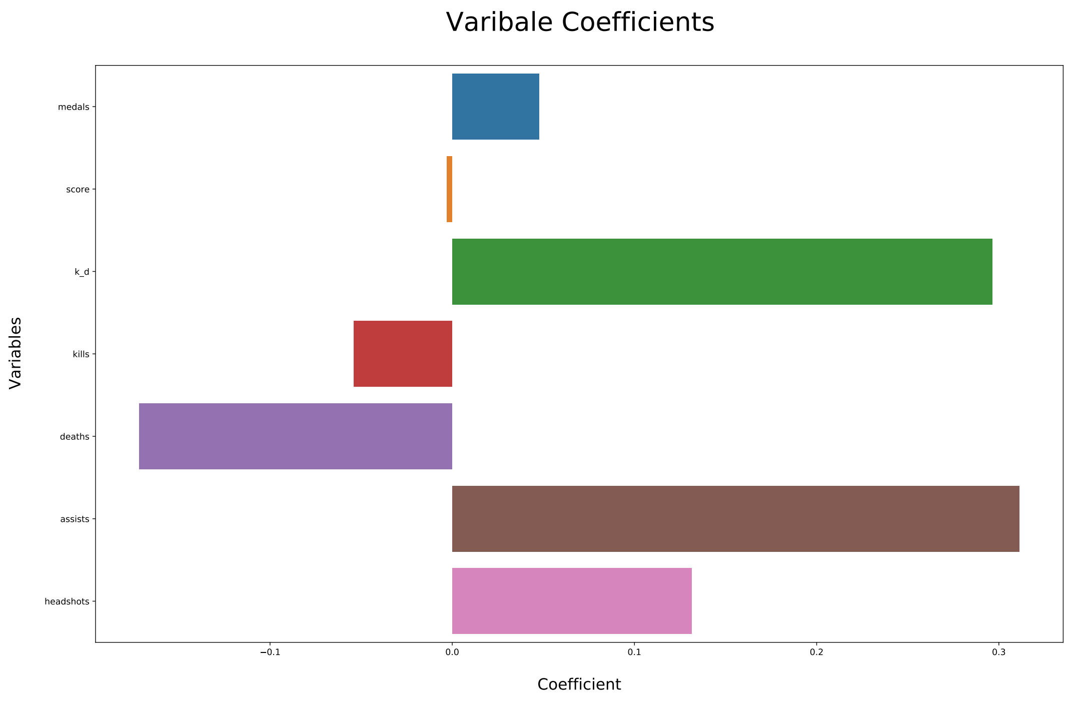
As you can see from the confusion matrix, of the ten different test games seven were wins and three were losses. The model predicted nine wins and one loss. Of the nine wins predicted, only six were actually wins. For the one loss predicted, that game was actually a win.

The accuracy of my model was 60%, predicting six of the ten games correctly. The precision of the model was 67%, predicting only six of the nine games classified as wins correctly. Those three incorrectly classified wins were false positives. The recall was 86%, correctly predicting six of the seven actual wins correctly. And finally, the F1 Score was 75%, a weighted average of the precision's 67% and recall's 86%.

Overall, the model performed alright. I would have liked higher accuracy and lower false positives. That being said, it is not a bad model for my use case. With more data, I can begin to optimize this and create a better model, but for now it will do.

One of the most important things I was looking for by creating this model was actually the coefficients for each variable. I can easily call these with the code shown below. Note, these coefficients and their interpretations are only as accurate as the model, so while it will guide me towards discerning high level conclusions, it does not give a guarantee that each variable is as impactful as the coefficients make it seem.

```
coef_list = log_model.coef_[0]
coef_series = pd.Series(coef_list, index = X_train.columns)

plt.figure(figsize=(20,12))
sns.barplot(x = coef_series, y = coef_series.index)
plt.xlabel('\nCoefficient\n', fontsize = 18)
plt.ylabel('\nVariables\n', fontsize = 18)
plt.title('\nVaribale Coefficients\n', fontsize = 30)
plt.show();
```



From these coefficients we are able to verify a few assumptions from earlier, but also get some new insights into how other variables impact the win rate.

The coefficient for "K_D" was about .3, making it one of the highest POSITIVE impacts on the model. From this, I can confirm that a higher K/D ratio leads to a higher probability of winning the game.

The coefficient for "Deaths" was -.17, making it the largest NEGATIVE impacts on the model. Here, I can confirm that my chance of winning the game increases as the number of deaths I have decreases.

Some new insights I am seeing are related to the "Kills" and "Assists" variables.

For "Kills", there was actually a negative coefficient. This can be interpreted as my chances of winning increase as the number of kills I get decreases, to a lesser extent than "Deaths". While at first this seems counter-intuitive, it might actually make sense if you look at the actual data. A significant portion of the games where I had a high amount of kills were the same games were I had a lot of deaths. And the fact that the coefficient is less negative than "deaths" accounts for the rest of the games I had a lot of kills and not as many deaths.

The coefficient for "Assists" was the highest positive value. This also makes sense. Since Halo is a team game, if I can assist teammates in getting kills I will be helping my team win. One thing to note here is that above, while plotting the single variable relationship between assist and win rate, there was a large confidence interval range. When looking at the descriptive statistics for assist, the standard deviation was even greater than the mean. So while the model has made assists the variable with the largest impact, I'd be interested to see if this changes once more data is acquired.

Conclusion

To conclude, by analyzing game stats on win rate for Halo: The Master Chief Collection, I have been able to determine key areas of play that will improve my win rate.

Until my model changes, Assists seem to be the biggest opportunity for improvement. The more assists I can get will help my team overall win the game. The second takeaway is that my Kill/Death ratio is extremely important to winning the game. The key here isn't necessarily to get more kills, but to not die as much (much harder said than done). This is even more apparent when I look at my third analysis point. If I don't want to lose as many games, I need to not die as much.

Using this analysis I will hopefully improve my game play and have a much better time playing the game.