1 League of Legends Classification

Author: Eric Wehmueller

1.1 Overview

This project is the third project for Flatiron School's bootcamp program in Data Science. We are being placed into a hypothetical situation as a Data Scientist and hoping to provide value to our business for the scenario we are given.

1.2 Business Problem

I have been hired by the esports organization Cloud9 as a player coach/analyst for the professional League of Legends team. They are competing at the top level and are looking to win every game they possibly can, as there is a lot of money on the line. My job is to help them determine the most important factors in winning League of Legends games. I am to investigate what should I be advising our players to focus on in the first 10 minutes of each game to provide the highest chance to win the game.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import scipy.stats as stats
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, plot_
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import export_graphviz, plot_tree
from IPython.display import Image
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

executed in 1.71s, finished 05:17:01 2021-04-19

1.3 Data Investigation and

Cleaning

```
file1 = "data\high_diamond_ranked_10min.csv"
file2 = "data\Challenger_Ranked_Games_10minute.csv"
df = pd.read_csv(file1)
```

executed in 44ms, finished 05:17:01 2021-04-19

```
df.info()
```

executed in 23ms, finished 05:17:01 2021-04-19

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9879 entries, 0 to 9878
Data columns (total 40 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   gameId                        9879 non-null   int64  
 1   blueWins                      9879 non-null   int64  
 2   blueWardsPlaced               9879 non-null   int64  
 3   blueWardsDestroyed            9879 non-null   int64  
 4   blueFirstBlood                9879 non-null   int64  
 5   blueKills                     9879 non-null   int64  
 6   blueDeaths                    9879 non-null   int64  
 7   blueAssists                   9879 non-null   int64  
 8   blueEliteMonsters             9879 non-null   int64  
 9   blueDragons                   9879 non-null   int64  
 10  blueHeralds                   9879 non-null   int64  
 11  blueTowersDestroyed           9879 non-null   int64  
 12  blueTotalGold                 9879 non-null   int64  
 13  blueAvgLevel                  9879 non-null   float64
 14  blueTotalExperience           9879 non-null   int64  
 15  blueTotalMinionsKilled        9879 non-null   int64  
 16  blueTotalJungleMinionsKilled  9879 non-null   int64  
 17  blueGoldDiff                  9879 non-null   int64  
 18  blueExperienceDiff            9879 non-null   int64  
 19  blueCSPerMin                  9879 non-null   float64
 20  blueGoldPerMin                9879 non-null   float64
 21  redWardsPlaced                9879 non-null   int64  
 22  redWardsDestroyed             9879 non-null   int64  
 23  redFirstBlood                 9879 non-null   int64  
 24  redKills                      9879 non-null   int64  
 25  redDeaths                     9879 non-null   int64  
 26  redAssists                    9879 non-null   int64  
 27  redEliteMonsters              9879 non-null   int64  
 28  redDragons                    9879 non-null   int64  
 29  redHeralds                    9879 non-null   int64  
 30  redTowersDestroyed            9879 non-null   int64  
 31  redTotalGold                  9879 non-null   int64  
 32  redAvgLevel                   9879 non-null   float64
 33  redTotalExperience            9879 non-null   int64  
 34  redTotalMinionsKilled         9879 non-null   int64  
 35  redTotalJungleMinionsKilled   9879 non-null   int64  
 36  redGoldDiff                   9879 non-null   int64  
 37  redExperienceDiff             9879 non-null   int64  
 38  redCSPerMin                   9879 non-null   float64
 39  redGoldPerMin                 9879 non-null   float64
dtypes: float64(6), int64(34)
memory usage: 3.0 MB
```

```
pd.set_option('display.max_columns', None)
df.head(10)
```

executed in 41ms, finished 05:17:01 2021-04-19

	gameId	blueWins	blueWardsPlaced	blueWardsI
0	4519157822	0	28	2
1	4523371949	0	12	1
2	4521474530	0	15	0
3	4524384067	0	43	1
4	4436033771	0	75	4
5	4475365709	1	18	0
6	4493010632	1	18	3
7	4496759358	0	16	2
8	4443048030	0	16	3
9	4509433346	1	13	1

```
for colName in df.columns:
    print(f'-{colName}- Value Counts')
    print(df[colName].value_counts())
    print();
```

executed in 61ms, finished 05:17:01 2021-04-19

```
-gameId- Value Counts
4458383359    1
4492870986    1
4447992971    1
4517889362    1
4524077612    1
             ..
4526469771    1
4511142535    1
4503476670    1
4516498052    1
4473786370    1
Name: gameId, Length: 9879, dtype: int64

-blueWins- Value Counts
0    4949
1    4930
Name: blueWins, dtype: int64

-blueWardsPlaced- Value Counts
16     1255
```

Notes on data exploration:

-----Challenger Dataset-----

There are some multiple gameIDs, we are going to want to filter out duplicates.

Inclusion of Dragon TYPE data is nice. We should make binary columns for Air,Earth,Water,Fire for each side (Red/Blue)

Filter out elder dragon data/game, this should not be possible by 10 minutes and is likely a bug.

Engineer a gold diff column (Positive for blue, negative for red)

First blood columns for each team is always zero. This means the first blood data is not in this dataset. This is really unfortunate because I believe (from my own personal experience) that having this data is actually really important if we are talking about action in the first 10 minutes of a game.

-----Diamond+ Dataset-----

No duplicate gameIDs, no missing values.

This dataset is much more ready to use for modeling and classification purposes, although it is a smaller dataset (10k vs 26k entries)

I'm going to continue working with this dataset from here on out, circling back to the challenger dataset if I have time.

With 4949 Red side wins and 4930 blue side wins, this is a fairly balanced dataset.

There is no categorical data, so no values need to be directly changed before modeling.

```
#Todo: Visualization of Gold Diff vs Win?
```

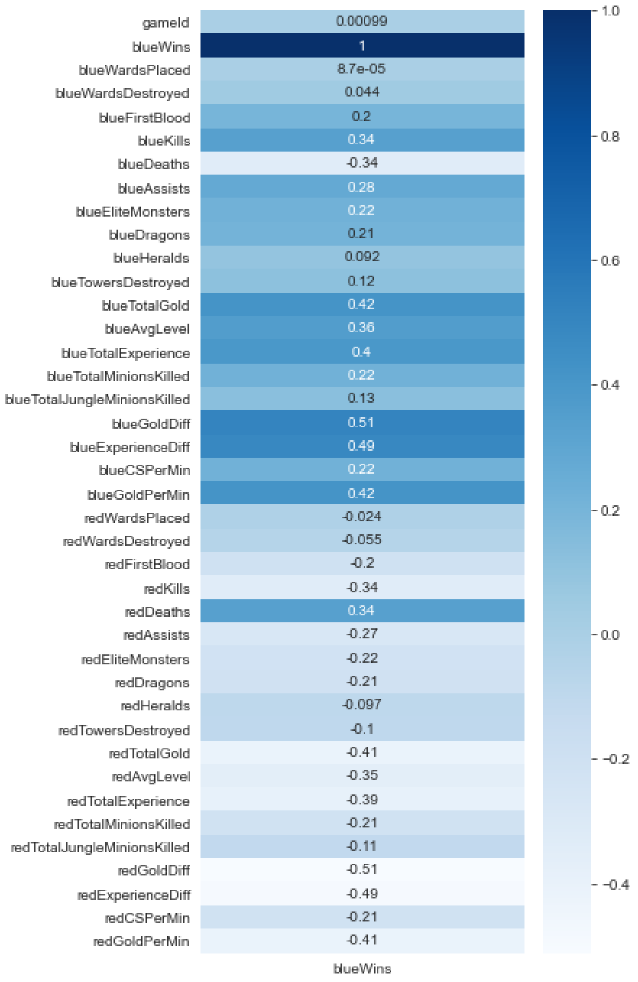
executed in 14ms, finished 05:17:01 2021-04-19

1.4 Visualizations

Let's take an initial look at which features correlate the most to game outcome directly.

```
sns.set_style("whitegrid")

fig = plt.figure(figsize=(5, 12))
sns.heatmap(df.corr()[['blueWins']], annot=True, cmap="B
```

executed in 815ms, finished 05:17:02 2021-04-19

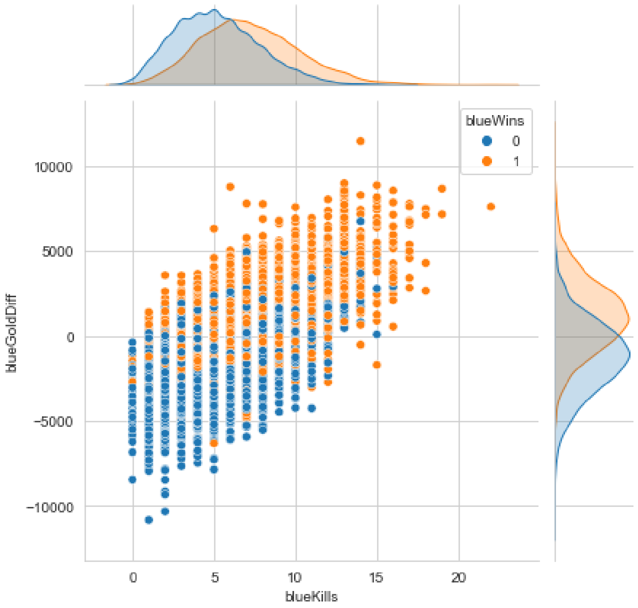


The largest correlation values lie in the Gold and Experience Differences, followed closely by other fields which tell the same story: Total Gold and Average Level (another interpretation of experience). Beyond that, Kills and Assists also seem to influence the outcome a moderate amount.

```
fig = plt.figure(figsize=(20, 20))
sns.jointplot(x='blueKills', y='blueGoldDiff', data=df,
```

executed in 614ms, finished 05:17:02 2021-04-19

```
<Figure size 1440x1440 with 0 Axes>
```



1.5 Feature Engineering

I'm going to quickly feature engineer a stat metric often used in the professional scene: KDA (Kills+Assists / Deaths) for both Blue and Red side.

```
df['blueKillsAndAssists'] = df['blueKills'] + df['blueAs
df['redKillsAndAssists'] = df['redKills'] + df['redAssis
```

executed in 15ms, finished 05:17:02 2021-04-19

```
def calc_KDA(data, isBlue=True):
    prefix = 'blue'
    if not isBlue:
        prefix = 'red'
    #checking for np.inf division by zero, replace these
    df[prefix+'KDA'] = round((df[prefix+'KillsAndAssists
```

executed in 14ms, finished 05:17:02 2021-04-19

```
calc_KDA(df, isBlue=True)
calc_KDA(df, isBlue=False)
```

executed in 13ms, finished 05:17:02 2021-04-19

```
df.head(10)
```

executed in 46ms, finished 05:17:02 2021-04-19

	gameId	blueWins	blueWardsPlaced	blueWardsI
0	4519157822	0	28	2
1	4523371949	0	12	1
2	4521474530	0	15	0
3	4524384067	0	43	1
4	4436033771	0	75	4
5	4475365709	1	18	0
6	4493010632	1	18	3
7	4496759358	0	16	2
8	4443048030	0	16	3
9	4509433346	1	13	1

2 Modeling

2.1 Gaussian Naive-Bayes

To start, I'm going to make a Gaussian Naive-Bayes Model. This model assumes that features are independent of one another, so I will be dropping some features to meet this assumption.

```
df_no_red_kills = df[df['redKills'] == 0]
df_zero_kills = df_no_red_kills[df_no_red_kills['blueKil


df_zero_kills.shape
```

executed in 14ms, finished 05:17:02 2021-04-19

```
(0, 44)
```

There are no games where neither team has a kill at 10 minutes in this dataset. We can assume that if one team does not take first blood, the other team must have taken first blood. Hence, we can remove the 'redFirstBlood' feature. The other features I'm removing below are the inverse of the corresponding blue feature.

```
#several of these columns are just diving by 10 to get "
#some are redundant, removing them for modelling to redu
model_df = df.drop(['gameId',
                 'blueGoldPerMin', 'redGoldPerMin',
                 'blueCSPerMin', 'redCSPerMin',
                 'redGoldDiff', 'redExperienceDiff',
                 'redTotalGold', 'redTotalExperience',
                 'redKills', 'redDeaths',
                'redFirstBlood'], axis=1)
```

executed in 14ms, finished 05:17:02 2021-04-19

```
model_df.head(10)
```

executed in 31ms, finished 05:17:02 2021-04-19

	blueWins	blueWardsPlaced	blueWardsDestroyed	b
0	0	28	2	1
1	0	12	1	0
2	0	15	0	0
3	0	43	1	0
4	0	75	4	0
5	1	18	0	0
6	1	18	3	1
7	0	16	2	0
8	0	16	3	0
9	1	13	1	1

```
X1= model_df.drop('blueWins', 1)
y1= model_df['blueWins']

X_train, X_test, y_train, y_test = train_test_split(
    X1,
    y1)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

executed in 14ms, finished 05:17:02 2021-04-19

```
((7409, 31), (2470, 31), (7409,), (2470,))
```

```
nb = GaussianNB()
nb.fit(X_train, y_train)
```

executed in 15ms, finished 05:17:02 2021-04-19

```
GaussianNB()
```

```
y_pred_train = nb.predict(X_train)
y_pred_test = nb.predict(X_test)
```

executed in 14ms, finished 05:17:02 2021-04-19

```
print(classification_report(y_train, y_pred_train))
```

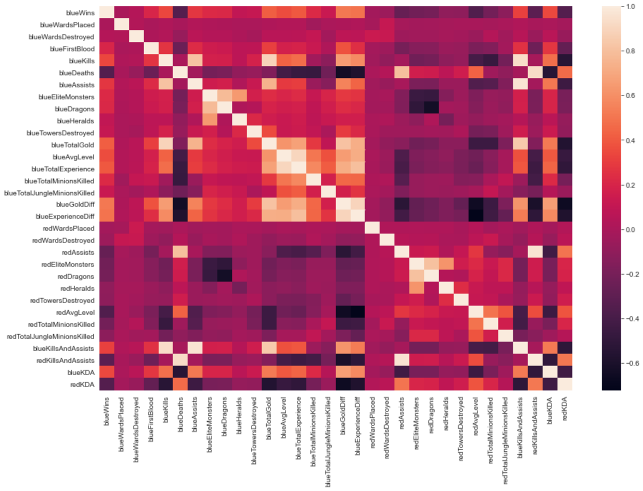
executed in 15ms, finished 05:17:02 2021-04-19

```
              precision    recall  f1-score   support

           0       0.72      0.73      0.72      3687
           1       0.73      0.72      0.72      3722

    accuracy                           0.72      7409
   macro avg       0.72      0.72      0.72      7409
weighted avg       0.72      0.72      0.72      7409
```

I'm now looking to remove features that may be highly correlated that are still remaining. Let's check that.

```
corr = model_df.corr()
plt.figure(figsize=(15,10))
ax= plt.subplot()
sns.heatmap(corr, ax=ax);
```

executed in 1.09s, finished 05:17:04 2021-04-19



In this visualization, I'm looking for values that are completely white or completely black (1.00 or -1.00), meaning that these features are highly correlated.

We can remove 'redAvgLevel' since in-game level IS experience, and it correlates too heavily with 'blueExperienceDiff'. It's essentially the same thing. Same with 'blueAvgLevel' and 'blueTotalExperience', this is just a scaled-down field of the same values.

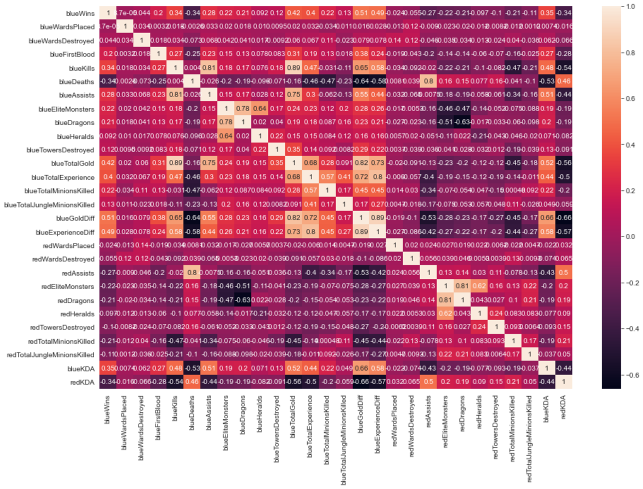
I also realized I need to remove my blue and red Kills+Assists features, as this was just a helpful step to calculate KDA while handling division by zero.

```
model_df = model_df.drop(['redAvgLevel',
                          'blueKillsAndAssists',
                          'redKillsAndAssists',
                          'blueAvgLevel'], axis=1)
```

executed in 15ms, finished 05:17:04 2021-04-19

```
corr = model_df.corr()
plt.figure(figsize=(15,10))
ax= plt.subplot()
sns.heatmap(corr, ax=ax, annot=True);
```

executed in 5.12s, finished 05:17:09 2021-04-19



2.2 Decision Tree Classifier

```
X_dtc= model_df.drop('blueWins', 1)
y_dtc= model_df['blueWins']

X2_train, X2_test, y2_train, y2_test = train_test_split(
    X_dtc,
    y_dtc)

X2_train.shape, X2_test.shape, y2_train.shape, y2_test.s
```

executed in 14ms, finished 05:17:09 2021-04-19

```
((7409, 27), (2470, 27), (7409,), (2470,))
```

```
dtc = DecisionTreeClassifier() #max_depth= 5
dtc.fit(X2_train, y2_train)
scores = cross_val_score(dtc, X2_train,y2_train)
print('cross-val-scores')
print(scores)
print('mean')
print(round(scores.mean(), 4))
```

executed in 447ms, finished 05:17:09 2021-04-19

```
cross-val-scores
[0.64237517 0.634278   0.63697706 0.63630229 0.65361242]
mean
0.6407
```

```
pred = dtc.predict(X2_test)
print(classification_report(y2_test, pred))
```

executed in 14ms, finished 05:17:09 2021-04-19

```
              precision    recall  f1-score   support

           0       0.63      0.66      0.64      1228
           1       0.65      0.62      0.64      1242

    accuracy                           0.64      2470
   macro avg       0.64      0.64      0.64      2470
weighted avg       0.64      0.64      0.64      2470
```

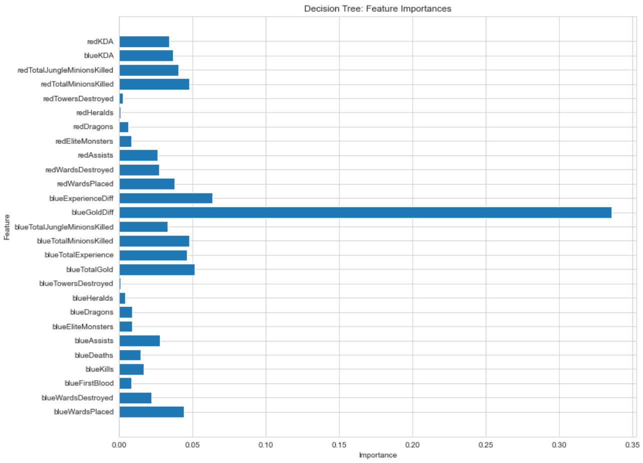
```
plt.figure(figsize=(20,10))
#myplot = plot_tree(dtc, feature_names=X_dtc.columns, cl
#                    rounded =True, proportion=False, pre

#we didn't set a max depth so uncomment this if you REAL
#decision tree, but it takes 2+ minutes to execute
```

executed in 15ms, finished 05:17:09 2021-04-19

```
    <Figure size 1440x720 with 0 Axes>


 <Figure size 1440x720 with 0 Axes>
```

```
sns.set_style("whitegrid")

fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111)
plt.barh(X_dtc.columns, dtc.feature_importances_)
ax.set(
    title="Decision Tree: Feature Importances",
    ylabel="Feature",
    xlabel="Importance"
);
```

executed in 323ms, finished 05:17:10 2021-04-19

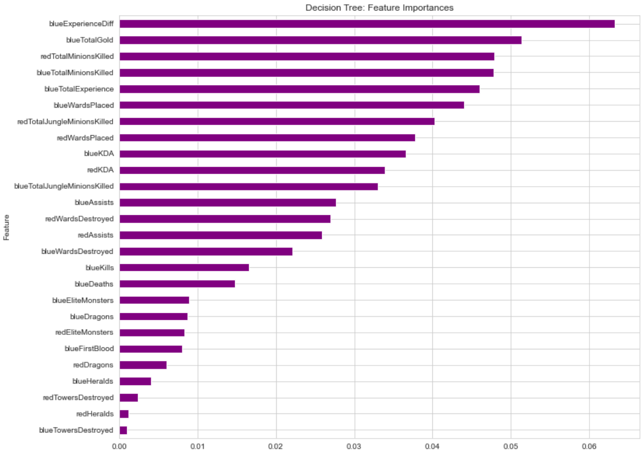


As you can see, our model has a clear winner here- 'blueGoldDiff'. No wonder they show the current gold value for both teams at all times on professional game broadcasts. This model agrees; this is a pretty great indicator for which team is winning. However, it doesn't give us that much insight into the game. Everything you do gives you gold, so how do you create a gold difference? We have to look at the other features.

```
sorted_series = pd.Series(dtc.feature_importances_,index

sns.set_style("whitegrid")
plt.figure(figsize=(12, 10))
sorted_series.iloc[:-1].plot(kind='barh',
                   color='purple',
                   title="Decision Tree: Feature Import
                   xlabel='Feature',
                   ylabel='Importance')
```

executed in 292ms, finished 05:17:10 2021-04-19

```
<AxesSubplot:title={'center':'Decision Tree: Feature Impor
tances'}, ylabel='Feature'>
```



This model favors the Experience difference the most for predicting an end-game result, followed by how many Jungle and Regular minions are killed by red team.

It is worth noting that our model honestly could care less if your team is taking towers and Heralds early on in the game. These are the least impactful by far.

To Summarize: (subject to change on different train-test-splits but should generally be about the same)

Top Features:

- Gold Differential (the obvious answer)
- Experience Differential
- Jungle/Regular Minion total
- Wards Placed

Bottom Features:

- Towers (least impactful)
- Heralds

- Elite Monsters/Dragons

Let's be more picky with our correlation heatmap and remove several more features before our third classification model. While interpreting these results, I realized that the only "Elite Monsters" on the map before 10 minutes are dragons, and only 1 will be able to spawn. Dragons spawn every 5 minutes starting at 5 minutes into the game. Therefore, the Elite Monsters features are essentially useless- providing the same level of information as the dragons column.

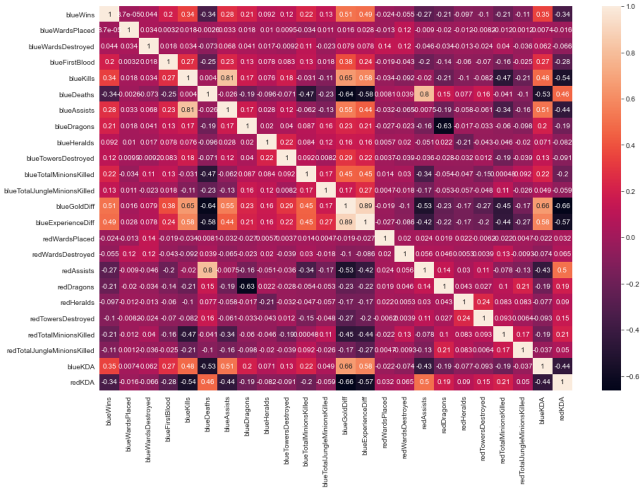
I'm also removing "Total Experience", as it correlates heavily with Exp Differential. We only need one of these. I'm also going to remove the "Total Gold" for the same reasoning.

```
model_df = model_df.drop(['redEliteMonsters',
                          'blueEliteMonsters',
                          'blueTotalExperience',
                           'blueTotalGold'], axis=1)
```

executed in 15ms, finished 05:17:10 2021-04-19

```
corr = model_df.corr()
plt.figure(figsize=(15,10))
ax= plt.subplot()
sns.heatmap(corr, ax=ax, annot=True);
```

executed in 3.92s, finished 05:17:14 2021-04-19



Looking much better than before. No values here greater than 0.81 or less than -0.66 on our correlation heatmap aside from Experience and Gold. However, I would like the keep the two for interpretable results and being able to compare the two in our final model.

2.3 XGB Classifier

I've been hearing a lot of things about how powerful XGB is and I'd really like to put it to use here as a final model for this project.

```
model_df.head()
```

executed in 13ms, finished 05:17:14 2021-04-19

	blueWins	blueWardsPlaced	blueWardsDestroyed	b
0	0	28	2	1
1	0	12	1	0
2	0	15	0	0
3	0	43	1	0
4	0	75	4	0

```
X_xgb= model_df.drop('blueWins', 1)
y_xgb= model_df['blueWins']

X3_train, X3_test, y3_train, y3_test = train_test_split(
    X_xgb,
    y_xgb)

X3_train.shape, X3_test.shape, y3_train.shape, y3_test.s
```

executed in 14ms, finished 05:17:14 2021-04-19

```
((7409, 23), (2470, 23), (7409,), (2470,))
```

```
model_xgb = XGBClassifier()
model_xgb.fit(X3_train, y3_train)
scores = cross_val_score(model_xgb, X3_train,y3_train)
print('cross-val-scores')
print(scores)
print('mean')
print(round(scores.mean(), 5))
```

executed in 1.74s, finished 05:17:16 2021-04-19

```
cross-val-scores
[0.73144399 0.69230769 0.69635628 0.70175439 0.71505739]
mean
0.70738
```

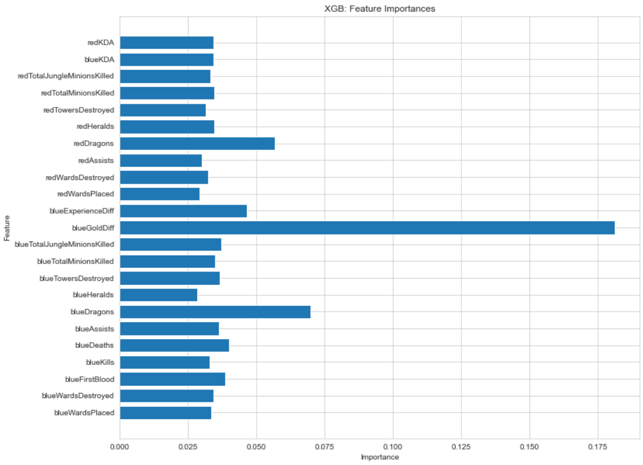
```
pred_xgb = model_xgb.predict(X3_test)
print(classification_report(y3_test, pred_xgb))
```

executed in 29ms, finished 05:17:16 2021-04-19

```
              precision    recall  f1-score   support

           0       0.72      0.71      0.72      1260
           1       0.70      0.71      0.71      1210

    accuracy                           0.71      2470
   macro avg       0.71      0.71      0.71      2470
weighted avg       0.71      0.71      0.71      2470
```

```
sns.set_style("whitegrid")

fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111)
plt.barh(X_xgb.columns, model_xgb.feature_importances_)
ax.set(
    title="XGB: Feature Importances",
    ylabel="Feature",
    xlabel="Importance"
);
```

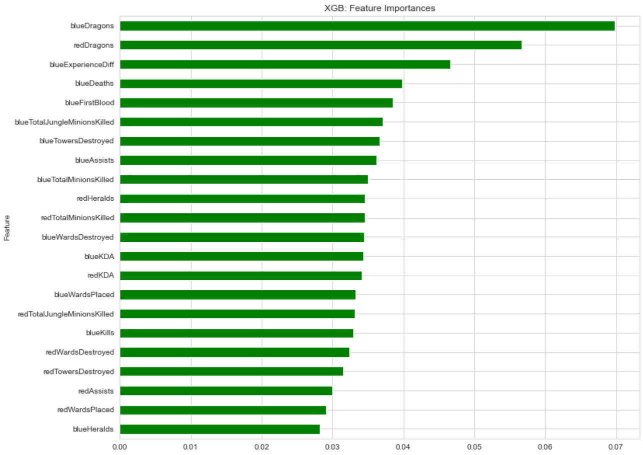
executed in 369ms, finished 05:17:16 2021-04-19



```
sorted_series_xgb = pd.Series(model_xgb.feature_importan

sns.set_style("whitegrid")
plt.figure(figsize=(12, 10))
sorted_series_xgb.iloc[:-1].plot(kind='barh',
                    color='green',
                    title="XGB: Feature Importances",
                    xlabel='Feature',
                    ylabel='Importance')
```

executed in 306ms, finished 05:17:16 2021-04-19

```
<AxesSubplot:title={'center':'XGB: Feature Importances'},
ylabel='Feature'>
```



We have mostly similar results from our XGB classification as well. With a 70.5% accuracy we are able to predict the outcome of a game based on these 23 (mostly) independent features.

It is worth noting that our priorities have changed on this model compared to the Decision Tree classification model.

Ignoring the obvious Gold Difference feature, we have a very different leader for feature importance- **dragons**

Our model thinks that securing dragons is the most impactful early things your team can take before 10 minutes. To summarize:

Top Features:

- Gold Differential (the obvious answer)
- Dragons
- Experience

Bottom Features:

- Towers (least impactful yet again)
- Heralds (again)
- Blue side vision control (wards placed/destroyed)

2.4 K-Nearest Neighbors

```python
best_K = 0
best_score = 0
k_range = range(1,50)
error = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X3_train, y3_train)
    pred = knn.predict(X3_test)
    score = round(knn.score(X3_test, y3_test)*100, 3)
    error.append(np.mean(pred != y3_test))
    if score > best_score:
        best_score = round(score, 3)
        best_K = k

print(f"Best K: {best_K}")
print(f"Best Accuracy: {best_score}%")
```

executed in 10.4s, finished 05:17:27 2021-04-19

```
Best K: 19
Best Accuracy: 72.308%
```

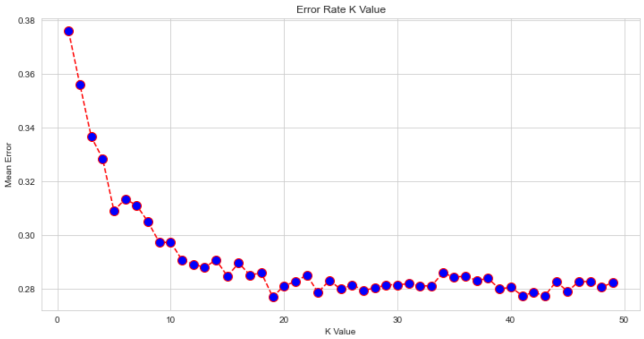
```
model_knn = KNeighborsClassifier(n_neighbors=best_K)
model_knn.fit(X3_train, y3_train)
pred = model_knn.predict(X3_test)
print(classification_report(y3_test, pred))
```

executed in 138ms, finished 05:17:27 2021-04-19

```
              precision    recall  f1-score   support

           0       0.73      0.73      0.73      1260
           1       0.72      0.72      0.72      1210

    accuracy                           0.72      2470
   macro avg       0.72      0.72      0.72      2470
weighted avg       0.72      0.72      0.72      2470
```

```
plt.figure(figsize=(12, 6))
plt.plot(k_range, error, color='red', linestyle='dashed'
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error');
```

executed in 168ms, finished 05:17:27 2021-04-19



3 Conclusions

In conclusion, I believe that our XGBoost model with a ~71% F1-score and feature importances information available is giving us the best possible insights for what our team can be doing in the first 10 minutes of a game to give us the best chance to win.

We want to:

Prioritize: Experience and Dragons

Ignore: Rift Heralds and Towers

4 Future Work

As you can see, there is another dataset included in this notebook that I have not loaded and run tests on. It is very similar in many ways but includes only games from the top 300 players (much more selective than the top 1%) over a longer period of time, with 26k entries vs. 10k entries in my current notebook.

It would also be worth investigating data beyond the 10 minute mark. 15 minutes would be excellent, as games are still guaranteed to last for this amount of time. Anything beyond that and your game lengths being to vary.

There is so much more to this game than just "what do we want to do in the first 10 minutes". For example, a lot of champions are not meant to be strong in the first 10 minutes. So teams will intentionally give objectives to the enemy team just to buy themselves time to get to that point in the game: stalling for "late game" and getting to the 30 minute mark, for example. Looking at data and analysis per champion(character) would be very exciting to see.

Dragon Type data would also be quite interesting. I've only listed "dragons" in this project without mentioning that there are 4 different elemental types that will spawn. The type that spawns is random and gives different character stat buffs depending on which one spawns. Determining the value of those/how many of those you get and finding out how that might impact our model would also be valuable and exciting to discover.