Final Project Submission

Please fill out:

- Student name:
- Student pace: self paced / part time / full time:
- Scheduled project review date/time:
- Instructor name:
- Blog post URL:
- Video of 5-min Non-Technical Presentation:

TABLE OF CONTENTS

Click to jump to matching Markdown Header.

INTRODUCTION

> Explain the point of your project and what question you are trying to answer with your modeling.

Business Problem

Summary of the business problem you are trying to solve, and the data questions that you plan to answer to solve them.

Questions to consider:

- What are the business's pain points related to this project?
- How did you pick the data analysis question(s) that you did?
- Why are these questions important from a business perspective?

OBTAIN

Data Understanding

Describe the data being used for this project.

Questions to consider:

- Where did the data come from, and how do they relate to the data analysis questions?
- What do the data represent? Who is in the sample and what variables are included?
- What is the target variable?
- What are the properties of the variables you intend to use?

Importing packages for importing data and exploratory visual analysis.

In [500]:

```
import pandas as pd
import seaborn as sns
# sns.set_theme(color_codes=True)
import matplotlib.pyplot as plt
import numpy as np

## Preprocessing tools
from sklearn.model_selection import train_test_split,cross_val_predict,cross
from sklearn.preprocessing import MinMaxScaler,StandardScaler,OneHotEncoder
scaler = StandardScaler()
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from imblearn.over_sampling import SMOTE,SMOTENC
from sklearn import metrics

## Models & Utils
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression,LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from time import time

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings(action='ignore')
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,Baggi
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import plot_confusion_matrix

from sklearn.metrics import accuracy_score, confusion_matrix, classification
```

In [501]:

```python
# Visualize the impact of a few key metrics on Hall of Fame inclusivity
def comparative_graph(s):
    cat, num = 'inducted', s
    fig, ax = plt.subplots(nrows=1, ncols=3,  sharex=False, sharey=False, fi
    fig.suptitle(s + ' vs Inducted', fontsize=20)

    # Create a distribution graph to compare HOF inducted players against th
    ax[0].title.set_text('density')
    for i in df[cat].unique():
        sns.distplot(df[df[cat]==i][num], hist=False, label=i, ax=ax[0])
    ax[0].grid(True)

    # Create a stacked bar graph containing 10 bins to help visualize the di
    ax[1].title.set_text('bins')
    breaks = np.quantile(df[num], q=np.linspace(0,1,11))
    tmp = df.groupby([cat, pd.cut(df[num], breaks, duplicates='drop')]).size
    tmp = tmp[df[cat].unique()]
    tmp["tot"] = tmp.sum(axis=1)
    for col in tmp.drop("tot", axis=1).columns:
         tmp[col] = tmp[col] / tmp["tot"]
    tmp.drop("tot", axis=1).plot(kind='bar', stacked=True, ax=ax[1], legend=

    # Create a boxplot to compare HOF inducted players against those not ind
    ax[2].title.set_text('outliers')
    sns.boxplot(x=cat, y=num, data=df, ax=ax[2])
    ax[2].grid(True)
    plt.savefig(s)
    plt.show();


# Create a new correlated dataframe with absolute value of a number,
def high_corr(df):
    df_highcorr = df.corr().abs().stack().reset_index().sort_values(0, ascen
    df_highcorr['Highly Correlated Pairs'] = list(zip(df_highcorr.level_0, d
    df_highcorr.set_index(['Highly Correlated Pairs'], inplace = True)
    df_highcorr.drop(columns=['level_1', 'level_0'], inplace = True)
    df_highcorr.columns = ['Correlation']
    df_highcorr.drop_duplicates(inplace=True)
    return df_highcorr[(df_highcorr.Correlation>.7) & (df_highcorr.Correlati

# Create function used to find Precision, Recall, Accuracy, and F1 Scores.
def print_metrics(labels, preds):
    print("Precision Score: {}".format(precision_score(labels, preds)))
    print("Recall Score: {}".format(recall_score(labels, preds)))
    print("Accuracy Score: {}".format(accuracy_score(labels, preds)))
    print("F1 Score: {}".format(f1_score(labels, preds)))

# Find the optimal K value for KNN models.
def find_best_k(X_train, y_train, X_test, y_test, min_k=1, max_k=25):
    best_k = 0
    best_score = 0.0
    for k in range(min_k, max_k+1, 2):
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train, y_train)
        preds = knn.predict(X_test)
        f1 = f1_score(y_test, preds)
```

```
        if f1 > best_score:
            best_k = k
            best_score = f1

    print("Best Value for k: {}".format(best_k))
    print("F1-Score: {}".format(best_score))

# Create a function that visualizes the confusion matrix for the model.
def plot_cm(model, normalize='true'):
    fig, ax = plt.subplots(figsize=(8, 8))
    plt.grid(False)
    plot_confusion_matrix(model, X_test, y_test, cmap='Blues', ax=ax, normal

# Create function for performing log transformations.
def log_transform(df,features):
    '''Runs a log transformation on a feature

        @params
        df is a pd.Dataframe
        features is a list of columns to be considered

        @output
        new log-transformed column

    '''
    for feature in features:
        df[feature + '_log'] = np.log(df[feature]+1)
    return df
```

In [502]:

```
df = pd.read_csv('data/high_diamond_ranked_10min.csv')

df.columns
```

Out[502]:

```
Index(['gameId', 'blueWins', 'blueWardsPlaced', 'blueWardsDestroyed',
       'blueFirstBlood', 'blueKills', 'blueDeaths', 'blueAssists',
       'blueEliteMonsters', 'blueDragons', 'blueHeralds',
       'blueTowersDestroyed', 'blueTotalGold', 'blueAvgLevel',
       'blueTotalExperience', 'blueTotalMinionsKilled',
       'blueTotalJungleMinionsKilled', 'blueGoldDiff', 'blueExperienceDiff',
       'blueCSPerMin', 'blueGoldPerMin', 'redWardsPlaced', 'redWardsDestroyed',
       'redFirstBlood', 'redKills', 'redDeaths', 'redAssists',
       'redEliteMonsters', 'redDragons', 'redHeralds', 'redTowersDestroyed',
       'redTotalGold', 'redAvgLevel', 'redTotalExperience',
       'redTotalMinionsKilled', 'redTotalJungleMinionsKilled', 'redGoldDiff',
       'redExperienceDiff', 'redCSPerMin', 'redGoldPerMin'],
      dtype='object')
```

In [503]:
```
df.head()
```

Out[503]:

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueKills	blueDea
0	4519157822	0	28	2	1	9	
1	4523371949	0	12	1	0	5	
2	4521474530	0	15	0	0	7	
3	4524384067	0	43	1	0	4	
4	4436033771	0	75	4	0	6	

5 rows × 40 columns

In [504]:

```
df.info()
```

```
<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
```

In [505]:
```
df.describe()
```

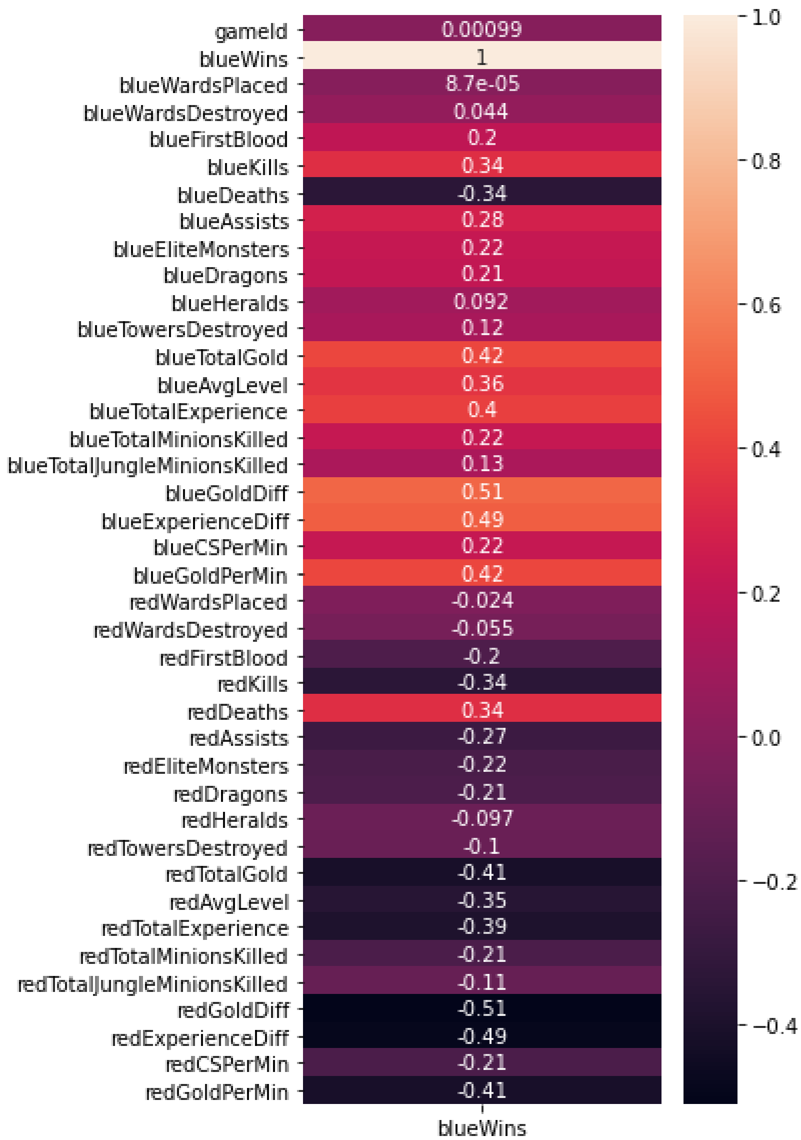
Out[505]:

	gameId	blueWins	blueWardsPlaced	blueWardsDestroyed	blueFirstBlood	blueK
count	9.879000e+03	9879.000000	9879.000000	9879.000000	9879.000000	9879.000
mean	4.500084e+09	0.499038	22.288288	2.824881	0.504808	6.183
std	2.757328e+07	0.500024	18.019177	2.174998	0.500002	3.011
min	4.295358e+09	0.000000	5.000000	0.000000	0.000000	0.000
25%	4.483301e+09	0.000000	14.000000	1.000000	0.000000	4.000
50%	4.510920e+09	0.000000	16.000000	3.000000	1.000000	6.000
75%	4.521733e+09	1.000000	20.000000	4.000000	1.000000	8.000
max	4.527991e+09	1.000000	250.000000	27.000000	1.000000	22.000

8 rows × 40 columns

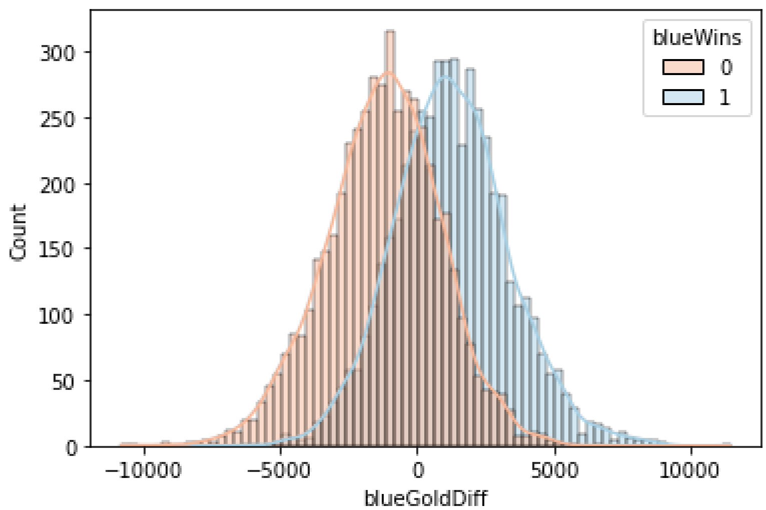
In [506]:
```
fig = plt.figure(figsize=(4, 10))
sns.heatmap(df.corr()[['blueWins']], annot=True)
```

Out[506]: <AxesSubplot:>



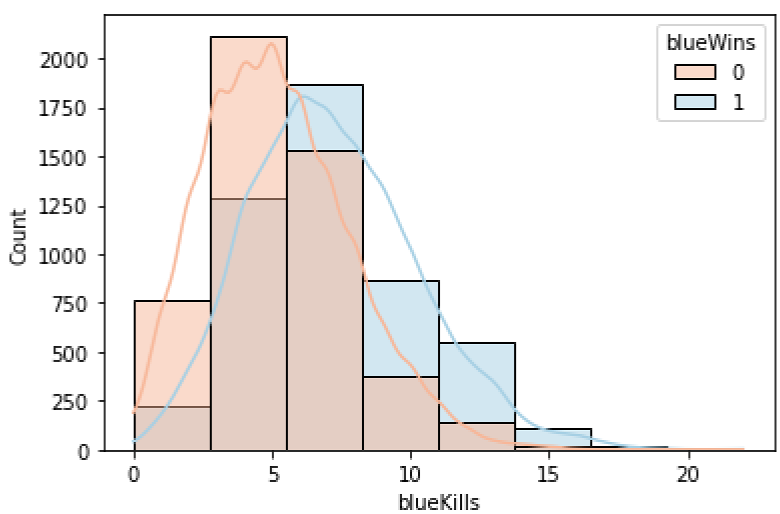
In [507]:
```
1 sns.histplot(x='blueGoldDiff', data=df, hue='blueWins', palette='RdBu', kde=
```

Out[507]: <AxesSubplot:xlabel='blueGoldDiff', ylabel='Count'>



In [508]:
```
1 sns.histplot(x='blueKills', data=df, hue='blueWins', palette='RdBu', kde=Tru
```

Out[508]: <AxesSubplot:xlabel='blueKills', ylabel='Count'>



In [ ]:
```
1
```

In [ ]:

```
 
```

SCRUB

Data Preparation

Describe and justify the process for preparing the data for analysis.

Questions to consider:

- Were there variables you dropped or created?
- How did you address missing values or outliers?
- Why are these choices appropriate given the data and the business problem?

After initial data understanding, we are confident that the data we're using is sound. No NA or missing values were discovered.

Let's take a look at the columns:

In [509]:

```
df.columns
```

Out[509]:

```
Index(['gameId', 'blueWins', 'blueWardsPlaced', 'blueWardsDestroyed',
       'blueFirstBlood', 'blueKills', 'blueDeaths', 'blueAssists',
       'blueEliteMonsters', 'blueDragons', 'blueHeralds',
       'blueTowersDestroyed', 'blueTotalGold', 'blueAvgLevel',
       'blueTotalExperience', 'blueTotalMinionsKilled',
       'blueTotalJungleMinionsKilled', 'blueGoldDiff', 'blueExperienceDiff',
       'blueCSPerMin', 'blueGoldPerMin', 'redWardsPlaced', 'redWardsDestroyed',
       'redFirstBlood', 'redKills', 'redDeaths', 'redAssists',
       'redEliteMonsters', 'redDragons', 'redHeralds', 'redTowersDestroyed',
       'redTotalGold', 'redAvgLevel', 'redTotalExperience',
       'redTotalMinionsKilled', 'redTotalJungleMinionsKilled', 'redGoldDiff',
       'redExperienceDiff', 'redCSPerMin', 'redGoldPerMin'],
      dtype='object')
```

There are a few columns that can be removed entirely and a few that can be combined into categorical variables.

First Blood

'First Blood' is awarded to the team who gets the first kill in the game. Both blueFirstBlood and redFirstBlood are binary and inversely related. If Blue wins First Blood, blueFirstBlood will be recorded as 1 and redFirstBlood will be recorded as 0.

We can merge these columns into one.

In [510]:
```
df['blueFirstBlood']
```

Out[510]:
```
0       1
1       0
2       0
3       0
4       0
       ..
9874    1
9875    0
9876    0
9877    1
9878    1
Name: blueFirstBlood, Length: 9879, dtype: int64
```

In [511]:
```
firstBlood = []

for item in df['blueFirstBlood']:
    if item == 1:
        firstBlood.append('Blue')
    else:
        firstBlood.append('Red')

df['firstBlood'] = firstBlood

df['firstBlood']
```

Out[511]:
```
0       Blue
1        Red
2        Red
3        Red
4        Red
        ...
9874    Blue
9875     Red
9876     Red
9877    Blue
9878    Blue
Name: firstBlood, Length: 9879, dtype: object
```

We can discard blueFirstBlood and redFirstBlood

In [512]:
```
del df['blueFirstBlood']
del df['redFirstBlood']
```

Kills & Deaths

blueKills is inversely related with redDeaths, and redKills is inversely related with blueDeaths since the Blue team can only kill Red players and vice versa. blueDeaths and redDeaths can both be removed, leaving kills intact will preserve this information.

In [513]:
```
del df['blueDeaths']
del df['redDeaths']
```

Dragon & Herald

While this wouldn't hold true for LoL data spanning the entire length of each game, we know that there is only one opportunity to kill both the Dragon and the Harold in the first 10 minutes of each match. Unlike firstBlood where the action always occurs in the first 10 minutes (at least for the matches in our dataset), each dragon or herald can be killed only once or not at all.

Therefore, dragon and herald can be categorized as 'Blue,' 'Red,' or 'None.'

In [514]:
```
dragon_list = []

dragon_kill = df['blueDragons'] - df['redDragons']

for item in dragon_kill:
    if item == 1:
        dragon_list.append('Blue')
    elif item == -1:
        dragon_list.append('Red')
    else:
        dragon_list.append('No Dragon')

df['dragon'] = dragon_list
```

blueDragons and redDragons can be removed:

In [515]:
```
del df['blueDragons']
del df['redDragons']
```

We can reuse this code for the herald feature:

In [516]:
```
herald_list = []

herald_kill = df['blueHeralds'] - df['redHeralds']

for item in herald_kill:
    if item == 1:
        herald_list.append('Blue')
    elif item == -1:
        herald_list.append('Red')
    else:
        herald_list.append('No Herald')

df['herald'] = herald_list
```

In [517]:
```
del df['blueHeralds']
del df['redHeralds']
```

Elite Monsters

In [518]:
```
del df['blueEliteMonsters']
del df['redEliteMonsters']
```

GoldDiff, ExperienceDiff, CSPerMin, and GoldPerMin

Both blue and red teams have these four metrics. While they are useful metrics for other types of analyses, they are essentially duplicative, since they are all calculated in a similar fashion from features already included in our data.

- GoldDiff represents the difference between blueTotalGold and redTotalGold
- ExperienceDiff represents the difference between blueTotalExperience and redTotalExperience
- blue and red CSPerMin represents the minute rate of blue and red TotalMinionsKilled. For our 10 minute data, CSPerMin for each team will always be TotalMinionsKilled divided by 10
- similarly, blue and red GoldPerMin represents blue and red TotalGold divided by 10

These four features from both teams (totaling 8 features) can be removed without losing any information.

In [519]:
```
del df['blueGoldDiff']
del df['blueExperienceDiff']
del df['blueCSPerMin']
del df['blueGoldPerMin']
del df['redGoldDiff']
del df['redExperienceDiff']
del df['redCSPerMin']
del df['redGoldPerMin']
```

gameId

gameId represents a unique identifier for every LoL game, no two gameId's will ever be the same, so this column can be removed.

In [520]:
```
del df['gameId']
```

Reviewing cleaned data

In [521]:
```
print(df.columns)

print(df.shape)
```

```
Index(['blueWins', 'blueWardsPlaced', 'blueWardsDestroyed', 'blueKills',
       'blueAssists', 'blueTowersDestroyed', 'blueTotalGold', 'blueAvgLevel',
       'blueTotalExperience', 'blueTotalMinionsKilled',
       'blueTotalJungleMinionsKilled', 'redWardsPlaced', 'redWardsDestroyed',
       'redKills', 'redAssists', 'redTowersDestroyed', 'redTotalGold',
       'redAvgLevel', 'redTotalExperience', 'redTotalMinionsKilled',
       'redTotalJungleMinionsKilled', 'firstBlood', 'dragon', 'herald'],
      dtype='object')
(9879, 24)
```

We were able to remove 14 columns through this process without losing any information.

Alternative Dataset - Differences

In [522]:
```
df
```

Out[522]:

	blueWins	blueWardsPlaced	blueWardsDestroyed	blueKills	blueAssists	blueTowersDestroyec
0	0	28	2	9	11	(
1	0	12	1	5	5	(
2	0	15	0	7	4	(
3	0	43	1	4	5	(
4	0	75	4	6	6	(
...	...	...	...	...	...	..
9874	1	17	2	7	5	(
9875	1	54	0	6	8	(
9876	0	23	1	6	5	(
9877	0	14	4	2	3	(
9878	1	18	0	6	5	(

9879 rows × 24 columns

In [523]:

```
diff_df = pd.DataFrame()

diff_df['WardsPlaced'] = df['blueWardsPlaced'] - df['redWardsPlaced']
diff_df['WardsDestroyed'] = df['blueWardsDestroyed'] - df['redWardsDestroyed
diff_df['Kills'] = df['blueKills'] - df['redKills']
diff_df['Assists'] = df['blueAssists'] - df['redAssists']
diff_df['TowersDestroyed'] = df['blueTowersDestroyed'] - df['redTowersDestro
diff_df['TotalGold'] = df['blueTotalGold'] - df['redTotalGold']
diff_df['AvgLevel'] = df['blueAvgLevel'] - df['redAvgLevel']
diff_df['TotalExperience'] = df['blueTotalExperience'] - df['redTotalExperie
diff_df['TotalMinionsKilled'] = df['blueTotalMinionsKilled'] - df['redTotalM
diff_df['TotalJungleMinionsKilled'] = df['blueTotalJungleMinionsKilled'] - d

diff_df = pd.concat([diff_df, df[['firstBlood', 'dragon', 'herald', 'blueWin

diff_df.head()
```

Out[523]:

	WardsPlaced	WardsDestroyed	Kills	Assists	TowersDestroyed	TotalGold	AvgLevel	TotalExperi
0	13	-4	3	3	0	643	-0.2	
1	0	0	0	3	-1	-2908	-0.2	
2	0	-3	-4	-10	0	-1172	-0.4	
3	28	-1	-1	-5	0	-1321	0.0	
4	58	2	0	-1	0	-1004	0.0	

In [524]:

```
df = diff_df.copy()
```

In [ ]:

In [ ]:

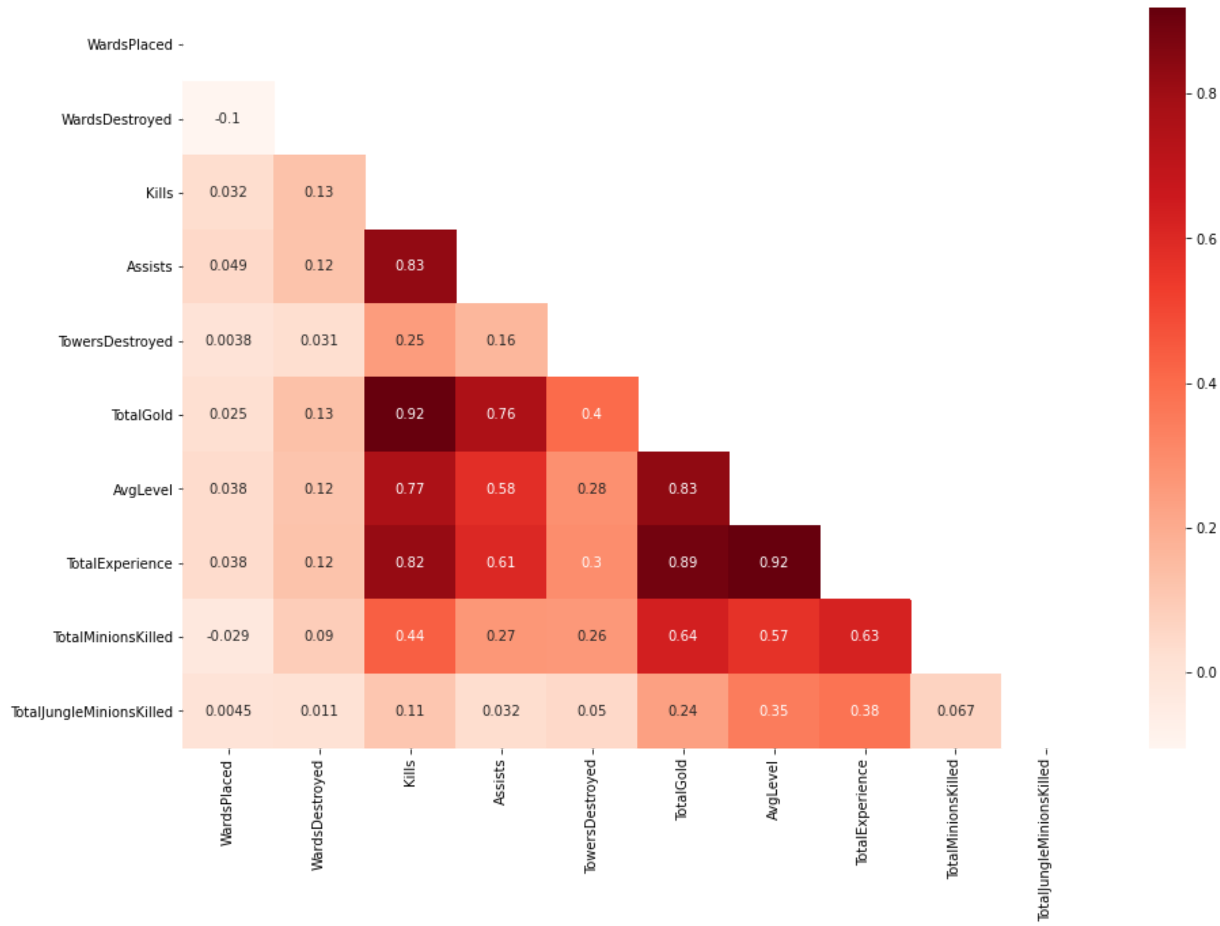
EXPLORE

In [525]:

```
def heatmap(df_name, figsize=(15,10), cmap='Reds'):
    corr = df_name.drop('blueWins',axis=1).corr()
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True
    fig, ax = plt.subplots(figsize=figsize)
    sns.heatmap(corr, annot=True, cmap=cmap, mask=mask)
    return fig, ax

heatmap(df)
```

Out[525]:

```
(<Figure size 1080x720 with 2 Axes>, <AxesSubplot:>)
```



In [526]:

```
# pd.set_option('display.max_rows', df.shape[0]+1)

# corr = df.drop('blueWins',axis=1).corr().abs()

# sort = corr.unstack()
# sort_order = sort.sort_values(kind="quicksort")

# corr_df = sort_order.to_frame()

# corr_features_df = corr_df[(corr_df[0] > 0.6) & (corr_df[0] < 1) ]

# corr_features_df.sort_values(by=0, ascending=False)
```

In [527]:

```
# https://pydatascience.org/2019/07/23/remove-duplicates-from-correlation-ma
def corr_list(df):
    dataCorr = df.drop('blueWins',axis=1).corr()

    dataCorr = dataCorr[abs(dataCorr) >= 0.01].stack().reset_index()
    dataCorr = dataCorr[dataCorr['level_0'].astype(str)!=dataCorr['level_1']]

    # filtering out lower/upper triangular duplicates
    dataCorr['ordered-cols'] = dataCorr.apply(lambda x: '-'.join(sorted([x['
    dataCorr = dataCorr.drop_duplicates(['ordered-cols'])
    dataCorr.drop(['ordered-cols'], axis=1, inplace=True)

    return dataCorr.sort_values(by=[0], ascending=False).head(10) #Get 10 hi

corr_list(df)
```

Out[527]:

	level_0	level_1	0
64	AvgLevel	TotalExperience	0.919161
23	Kills	TotalGold	0.917008
54	TotalGold	TotalExperience	0.894729
53	TotalGold	AvgLevel	0.833493
21	Kills	Assists	0.830751
25	Kills	TotalExperience	0.822845
24	Kills	AvgLevel	0.766222
33	Assists	TotalGold	0.759321
55	TotalGold	TotalMinionsKilled	0.638765
75	TotalExperience	TotalMinionsKilled	0.625556

Our multicollinearity analysis has presented a few variable relationships that need additional consideration.

- avgLevel and TotalExperience are highly correlated, which is not surprising. For now, we will stick with TotalExperience for blue and red teams since it's a bit more precise than AvgLevel.

However, we might want to run our baseline model with AvgLevel instead of TotalExperience to see if AvgLevel is more predictive.

- blue and red TotalGold appear consistently in our list. This is also not surprising since kills and assists award gold. We will remove gold for now.

In [528]:
```
corr_list(diff_df)
```

Out[528]:

	level_0	level_1	0
64	AvgLevel	TotalExperience	0.919161
23	Kills	TotalGold	0.917008
54	TotalGold	TotalExperience	0.894729
53	TotalGold	AvgLevel	0.833493
21	Kills	Assists	0.830751
25	Kills	TotalExperience	0.822845
24	Kills	AvgLevel	0.766222
33	Assists	TotalGold	0.759321
55	TotalGold	TotalMinionsKilled	0.638765
75	TotalExperience	TotalMinionsKilled	0.625556

In [529]:
```
df
```

Out[529]:

	WardsPlaced	WardsDestroyed	Kills	Assists	TowersDestroyed	TotalGold	AvgLevel	TotalEx
0	13	-4	3	3	0	643	-0.2	
1	0	0	0	3	-1	-2908	-0.2	
2	0	-3	-4	-10	0	-1172	-0.4	
3	28	-1	-1	-5	0	-1321	0.0	
4	58	2	0	-1	0	-1004	0.0	
...	...	...	...	...	...	...	...	
9874	-29	-1	3	-2	0	2519	0.4	
9875	42	-21	2	5	0	782	0.2	
9876	9	1	-1	-6	0	-2416	-0.4	
9877	-52	0	-1	2	0	-839	-0.6	
9878	9	-2	0	1	0	927	0.2	

9879 rows × 14 columns

In [530]:
```
df
```

Out[530]:

	WardsPlaced	WardsDestroyed	Kills	Assists	TowersDestroyed	TotalGold	AvgLevel	TotalEx
0	13	-4	3	3	0	643	-0.2	
1	0	0	0	3	-1	-2908	-0.2	
2	0	-3	-4	-10	0	-1172	-0.4	
3	28	-1	-1	-5	0	-1321	0.0	
4	58	2	0	-1	0	-1004	0.0	
...	...	...	...	...	...	...	...	
9874	-29	-1	3	-2	0	2519	0.4	
9875	42	-21	2	5	0	782	0.2	
9876	9	1	-1	-6	0	-2416	-0.4	
9877	-52	0	-1	2	0	-839	-0.6	
9878	9	-2	0	1	0	927	0.2	

9879 rows × 14 columns

In [ ]:

In [ ]:

In [ ]:

MODEL

In [ ]:

Data Modeling

Describe and justify the process for analyzing or modeling the data.

Questions to consider:

- How did you analyze or model the data?
- How did you iterate on your initial approach to make it better?
- Why are these choices appropriate given the data and the business problem?

Train Test Split

In [531]:

```
def tt_split_df(df):

    y = df['blueWins']
    X = df.drop(columns=['blueWins'], axis=1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

    X_train_tf = X_train.copy()
    X_test_tf = X_test.copy()

    categoricals = X.select_dtypes('O').columns
    numericals = X.select_dtypes('number').columns

    encoder = OneHotEncoder(sparse=False,drop='if_binary')
    train_categoricals = encoder.fit_transform(X_train_tf[categoricals])
    test_categoricals = encoder.transform(X_test_tf[categoricals])

    train_categoricals_df = pd.DataFrame(train_categoricals,
                                 columns=encoder.get_feature_names(categ

    test_categoricals_df =  pd.DataFrame(test_categoricals,
                                 columns=encoder.get_feature_names(categ
    train_numericals_df = pd.DataFrame(scaler.fit_transform(X_train_tf[numer
                              columns=numericals)

    test_numericals_df = pd.DataFrame(scaler.transform(X_test_tf[numericals]
                              columns=numericals)

    X_train_tf = pd.concat([train_numericals_df, train_categoricals_df], axi
    X_test_tf = pd.concat([test_numericals_df, test_categoricals_df], axis=1

    return X_train_tf, X_test_tf, y_train, y_test

# tt_split_df(diff_df)

X_train, X_test, y_train, y_test = tt_split_df(diff_df)

X_train_tf, X_test_tf, y_train, y_test = tt_split_df(df)

```

In [532]:

```
X_train_tf

```

Out[532]:

	WardsPlaced	WardsDestroyed	Kills	Assists	TowersDestroyed	TotalGold	AvgLevel
0	-0.791231	-0.380079	-0.245136	-0.337819	-0.019208	-0.163406	-0.398722
1	0.003580	1.043867	0.471110	0.183071	-0.019208	0.306376	1.698298
2	0.154972	-0.024093	0.709858	-0.337819	3.069716	1.276605	1.698298
3	-0.980472	-1.804026	-0.961381	-0.858708	-0.019208	-0.876870	0.020682
4	-0.034269	-1.448039	-0.961381	-0.511449	-0.019208	-0.721911	-0.818126
...	...	...	...	...	...	...	...
6910	0.003580	-0.736066	-0.722633	-1.032338	-0.019208	-1.157349	-0.818126
6911	0.041428	0.331894	0.471110	0.356701	-0.019208	0.727912	0.859490
6912	-0.450598	1.043867	-1.200130	-1.205968	-0.019208	-1.100926	-0.818126
6913	-0.450598	-0.024093	-1.916375	-0.858708	-0.019208	-1.756331	-2.495742
6914	-0.677687	-0.736066	0.471110	0.530331	-0.019208	0.149782	0.020682

6915 rows × 17 columns

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [533]:

```
# >>> X_train, X_test, _, _ = train_test_split(X,y, test_size=0.33, random_s
```

In [534]:
```
# Identify features and target

y = df['blueWins']
X = df.drop(columns=['blueWins'], axis=1)

# Assign train / test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ran

# Confirm split

print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
```

```
X_train shape: (6915, 13)
X_test shape: (2964, 13)
```

In [ ]:

In [ ]:

In [ ]:

Transforming X Train and Test

In [535]:
```
# X_train_tf = X_train.copy()
# X_test_tf = X_test.copy()
```

Categorical Columns

In [536]:
```
categoricals = X.select_dtypes('O').columns
numericals = X.select_dtypes('number').columns
categoricals, numericals
```

Out[536]:
```
(Index(['firstBlood', 'dragon', 'herald'], dtype='object'),
 Index(['WardsPlaced', 'WardsDestroyed', 'Kills', 'Assists', 'TowersDestroyed',
        'TotalGold', 'AvgLevel', 'TotalExperience', 'TotalMinionsKilled',
        'TotalJungleMinionsKilled'],
       dtype='object'))
```

In [537]:
```
## Encode categorical columns, only drop if binary
encoder = OneHotEncoder(sparse=False,drop='if_binary')
train_categoricals = encoder.fit_transform(X_train[categoricals])
test_categoricals = encoder.transform(X_test[categoricals])
train_categoricals
```

Out[537]:
```
array([[0., 0., 1., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 1., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 1., 0.],
       [1., 1., 0., ..., 0., 1., 0.],
       [1., 0., 0., ..., 1., 0., 0.]])
```

In [538]:
```
train_categoricals_df = pd.DataFrame(train_categoricals,
                                     columns=encoder.get_feature_names(categ

test_categoricals_df =  pd.DataFrame(test_categoricals,
                                     columns=encoder.get_feature_names(categ

train_categoricals_df.head()
```

Out[538]:

	firstBlood_Red	dragon_Blue	dragon_No Dragon	dragon_Red	herald_Blue	herald_No Herald	herald_Red
0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	0.0	1.0	0.0
2	1.0	0.0	0.0	1.0	1.0	0.0	0.0
3	0.0	0.0	0.0	1.0	0.0	0.0	1.0
4	1.0	1.0	0.0	0.0	0.0	1.0	0.0

Numerical Columns

In [539]:
```
from sklearn.preprocessing import StandardScaler

scaler = scaler = StandardScaler()

train_numericals_df = pd.DataFrame(scaler.fit_transform(X_train[numericals])
                              columns=numericals)

test_numericals_df = pd.DataFrame(scaler.transform(X_test[numericals]),
                              columns=numericals)

train_numericals_df.head()
```

Out[539]:

	WardsPlaced	WardsDestroyed	Kills	Assists	TowersDestroyed	TotalGold	AvgLevel	Tot
0	-0.312890	1.373334	0.220704	-0.349942	-0.04164	-0.001092	-0.821423	
1	0.156072	-0.741644	0.220704	-0.003060	-0.04164	-0.440888	-0.821423	
2	-0.000249	-0.389148	-0.733301	-0.349942	-0.04164	-0.276167	-0.405661	
3	0.156072	1.373334	2.128716	2.425111	-0.04164	2.063192	1.673149	
4	-0.039329	-0.036651	0.697707	0.343822	-0.04164	0.381093	0.010101	

In [540]:
```
X_train = pd.concat([train_numericals_df, train_categoricals_df], axis=1)
X_test = pd.concat([test_numericals_df, test_categoricals_df], axis=1)

X_train.head()
```

Out[540]:

	WardsPlaced	WardsDestroyed	Kills	Assists	TowersDestroyed	TotalGold	AvgLevel	Tot
0	-0.312890	1.373334	0.220704	-0.349942	-0.04164	-0.001092	-0.821423	
1	0.156072	-0.741644	0.220704	-0.003060	-0.04164	-0.440888	-0.821423	
2	-0.000249	-0.389148	-0.733301	-0.349942	-0.04164	-0.276167	-0.405661	
3	0.156072	1.373334	2.128716	2.425111	-0.04164	2.063192	1.673149	
4	-0.039329	-0.036651	0.697707	0.343822	-0.04164	0.381093	0.010101	

Logistic Regression

In [541]:
```
model_log = LogisticRegression()

model_log.fit(X_train, y_train)

# scores = cross_val_score(model_lr, X_train,y_train, cv=10) # model, train,


```

Out[541]:
```
LogisticRegression()
```

In [542]:
```
y_train.value_counts(1)
```

Out[542]:
```
1    0.500217
0    0.499783
Name: blueWins, dtype: float64
```

In [543]:
```
y_test.value_counts(1)
```

Out[543]:
```
0    0.503711
1    0.496289
Name: blueWins, dtype: float64
```

In [544]:
```
## Get Predictions for training and test data to check metrics functions
y_hat_train = model_log.predict(X_train)
y_hat_test = model_log.predict(X_test)
```

Model Accuracy

In [545]:
```
def model_accuracy(model_string_name, model, X_train=X_train, y_train=y_trai
    print(f'{model_string_name} accuracy score:')
    print(f'Training Score:\t{model_log.score(X_train,y_train):.2f}')
    print(f'Test Score:\t{model_log.score(X_test,y_test):.2f}')

model_accuracy('Logistic Regression', model_log)
```

```
Logistic Regression accuracy score:
Training Score: 0.74
Test Score:     0.72
```

Cross Validation Check

In [546]:
```
def cross_val_check(model_string_name, model, X_train=X_train, y_train=y_tra
    scores = cross_val_score(model, X_train, y_train, cv=10) # model, train,
    print(f'{model_string_name} Cross Validation Scores:\n')
    print(scores)
    print(f'\nCross validation mean: \t{scores.mean():.3f}')

cross_val_check('Logistic Regression', model_log)
```

```
Logistic Regression Cross Validation Scores:

[0.72976879 0.73265896 0.75867052 0.73410405 0.73988439 0.74384949
 0.72503618 0.75253256 0.72648336 0.73950796]

Cross validation mean:  0.738
```

Confusion Matrix

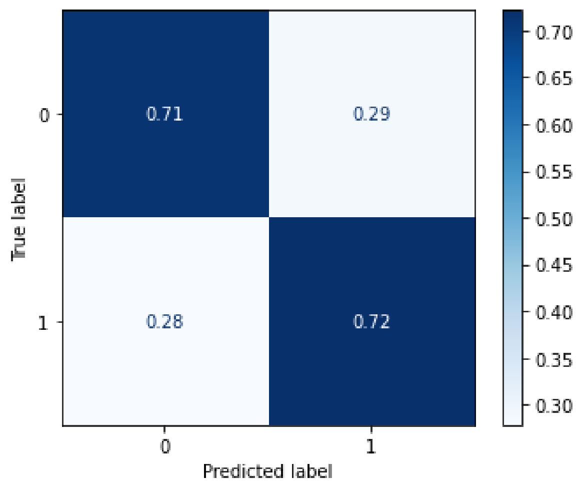
In [ ]:
```

```

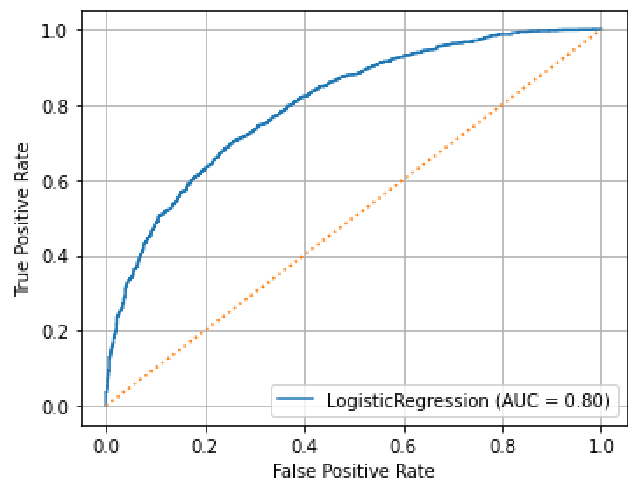
In [547]:

```
## Modified version of our simple eval function from Topic 25 Part 2 Study G
# - Added X_train and y_train for if we want scores for both train and test
def evaluate_classification(model, X_test_tf,y_test,cmap='Blues',
                            normalize='true',classes=None,figsize=(10,4),
                            X_train = None, y_train = None,):
    """Evaluates a scikit-learn binary classification model.

    Args:
        model ([type]): [description]
        X_test_tf ([type]): [description]
        y_test ([type]): [description]
        cmap (str, optional): [description]. Defaults to 'Reds'.
        normalize (str, optional): [description]. Defaults to 'true'.
        classes ([type], optional): [description]. Defaults to None.
        figsize (tuple, optional): [description]. Defaults to (8,4).
        X_train ([type], optional): [description]. Defaults to None.
        y_train ([type], optional): [description]. Defaults to None.
    """


    y_hat_test = model.predict(X_test_tf)
    print(metrics.classification_report(y_test, y_hat_test,target_names=clas

    fig,ax = plt.subplots(ncols=2,figsize=figsize)
    metrics.plot_confusion_matrix(model, X_test_tf,y_test,cmap=cmap,
                                  normalize=normalize,display_labels=classes
                                 ax=ax[0])

    curve = metrics.plot_roc_curve(model,X_test_tf,y_test,ax=ax[1])
    curve.ax_.grid()
    curve.ax_.plot([0,1],[0,1],ls=':')
    fig.tight_layout()
    plt.show()

    ## Add comparing Scores if X_train and y_train provided.
    if (X_train is not None) & (y_train is not None):
        print(f"Training Score = {model.score(X_train,y_train):.2f}")
        print(f"Test Score = {model.score(X_test_tf,y_test):.2f}")



def evaluate_grid(grid,X_test,y_test,X_train=None,y_train=None):
    print('The best parameters were:')
    print("\t",grid.best_params_)

    model = grid.best_estimator_

    print('\n[i] Classification Report')
    evaluate_classification(model, X_test,y_test,X_train=X_train,y_train=y_t
```

In [548]:
```
evaluate_classification(model_log, X_test, y_test)
```

```
              precision    recall  f1-score   support

           0       0.72      0.71      0.72      1493
           1       0.71      0.72      0.72      1471

    accuracy                           0.72      2964
   macro avg       0.72      0.72      0.72      2964
weighted avg       0.72      0.72      0.72      2964
```





Dummy Check