IMPORTANCE of the FIRST 10 MINUTES to WIN a RANKED LOL MATCH

1) OVERVIEW

Our group contains 2 members: Vural Can Şişman (201611055) and Yaren İPEK (201711038).

Vural's responsibilities: Literature review, preprocessing the data.
Yaren's responsibilities: Submitting the project proposal, preprocessing the data.

The rest of the division of labor will be done after this week since the other steps of the project has not been announced yet.

We will work on the League of Legends Diamond Ranked Games (10 min) dataset on Kaggle. This dataset contains the first 10 minutes statistics of 10.000 ranked games from a high ELO.

1.1) LITERATURE

There are 86 the League of Legends Diamond Ranked Games (10 min) notebooks on Kaggle. We examined some of these works in detail. These are the attributes and the algorithms used for data analysis related to the attributes in our dataset.

Warding totem: An item that a player can put on the map to reveal the nearby area. Very useful for map/objectives control.
Minions: NPC that belong to both teams. They give gold when killed by players.
Jungle minions: NPC that belong to NO TEAM. They give gold and buffs when killed by players.
Elite monsters: Monsters with high hp/damage that give a massive bonus (gold/XP/stats) when killed by a team.
Dragons: Elite monster which gives team bonus when killed. The 4th dragon killed by a team gives a massive stats bonus. The - - 5th dragon (Elder Dragon) offers a huge advantage to the team.
Herald: Elite monster which gives stats bonus when killed by the player. It helps to push a lane and destroys structures.
Towers: Structures you have to destroy to reach the enemy Nexus. They give gold.
Level: Champion level. Start at 1. Max is 18.
First Blood: First blood gives 100 bonus gold to the kill bounty and 50 to the assist bounty.
Kills: A kill is the event of decreasing an enemy champion's health to zero while the enemy has no abilities or Items to prevent death
Deaths: Death occurs when a champion takes sufficient damage to be reduced to zero health.
Assists: An assist is the action of helping an allied champion kill someone.
Gold: It is used to buy items in the shop that provide champions with bonus stats and abilities, which in turn is one of the main ways for champions to increase their power over the course of a game.

1.1.1 Gaussian Naïve Bayes

Naïve Bayes Theorem is based on calculating the probability of a hypothesis. Given training data X, posteriori probability of a hypothesis H, P(H|X), follows the Bayes' Theorem :

$$P(H|X) = (P(X|H) * P(H)) / P(X)$$

Where ,
P(H): initial probability
P(X): the probability that sample data is observed
P(X|H): the probability of observing the sample X, given that the hypothesis holds

For the same dataset, there are 3 different accuracy results which obtained by the Gaussian Naïve Bayes. One of them is **71.56%**[1], the other one is **73.43%**[2] and the last one is **71.76%** [3].

1.1.2 Decision Tree

Decision tree is a popular algorithm which can be used for classification and regression problems. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. A tree with decision and leaf nodes is the final result.

There are many different algorithms used in Decision Trees such as ID3, C4.5, CART, etc. We will be interested in the ID3 algorithm. The algorithm requires the calculation of the Entropy (H) and the Information Gain of the attributes.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i \qquad Gain(T, X) = Entropy(T) - Entropy(T, X)$$

There are 3 different accuracy results which are obtained by the Decision Tree algorithm. They are **63.12%** [4], **69.28%** [3], and **70.95%** [1].

1.1.3 Random Forest

Random Forest is a similar algorithm with Decision Tree. It also can be used in classification and regression problems. But it generates more than one decision trees and takes the average to improve the predictive accuracy of the given dataset.

There are 3 different accuracy results which are obtained by the Random Forest. They are **71.05%** [1], **72.82%** [3], **72.06%** [4].

1.1.4 K-Nearest Neighbors

K-Nearest Neighbors algorithm is based on the similarity and the calculation of the distance between points. Euclidean distance can be preferred to calculate the distance.

$$\Delta(X,Y) = w_X w_Y \sum_{i=1}^{m} d(x_i, y_i)^2 \qquad w_X = \frac{T}{C}$$

Where wx and wy are the weights for X and Y, m is the number of attributes and xi and yi are the values of the i^th attribute for X and Y. T is the total number of times that X is selected as the nearest neighbor, C is the total number of times that X correctly classifies examples.

Accuracy results of KNN algorithm on different projects are: **69.84%** [2], **71.71%** [3], **75.3%** [5].

2) GOALS

The aim of this project is to analyze the LOL Diamond Ranked Games (10 Min) dataset with Python. After preprocessing the data, the proper methods will be applied. As a result of the project, the relations between the features will be specified and explained. Models will be compared and the best model will be chosen.

3) DATA ANALYSIS

-We started with the attributes. There are 40 columns in our dataset. The attribute names and their data types are below:

Input:

```
txt = ""
for x in  data.columns:
        txt = txt + x + ": " + data[x].dtype.name+ ", " + "\n"
print(txt)
```

Output:

```
gameId: int64,
blueWins: int64,
blueWardsPlaced: int64,
blueWardsDestroyed: int64,
blueFirstBlood: int64,
blueKills: int64,
blueDeaths: int64,
blueAssists: int64,
blueEliteMonsters: int64,
blueDragons: int64,
blueHeralds: int64,
blueTowersDestroyed: int64,
blueTotalGold: int64,
```

```
blueAvgLevel: float64,
blueTotalExperience: int64,
blueTotalMinionsKilled: int64,
blueTotalJungleMinionsKilled: int64,
blueGoldDiff: int64,
blueExperienceDiff: int64,
blueCSPerMin: float64,
blueGoldPerMin: float64,
redWardsPlaced: int64,
redWardsDestroyed: int64,
redFirstBlood: int64,
redKills: int64,
redDeaths: int64,
redAssists: int64,
redEliteMonsters: int64,
redDragons: int64,
redHeralds: int64,
redTowersDestroyed: int64,
redTotalGold: int64,
redAvgLevel: float64,
redTotalExperience: int64,
redTotalMinionsKilled: int64,
redTotalJungleMinionsKilled: int64,
redGoldDiff: int64,
redExperienceDiff: int64,
redCSPerMin: float64,
redGoldPerMin: float64
```

In the light of this output we can say that there is no categorical attributes, all of them are numerical.

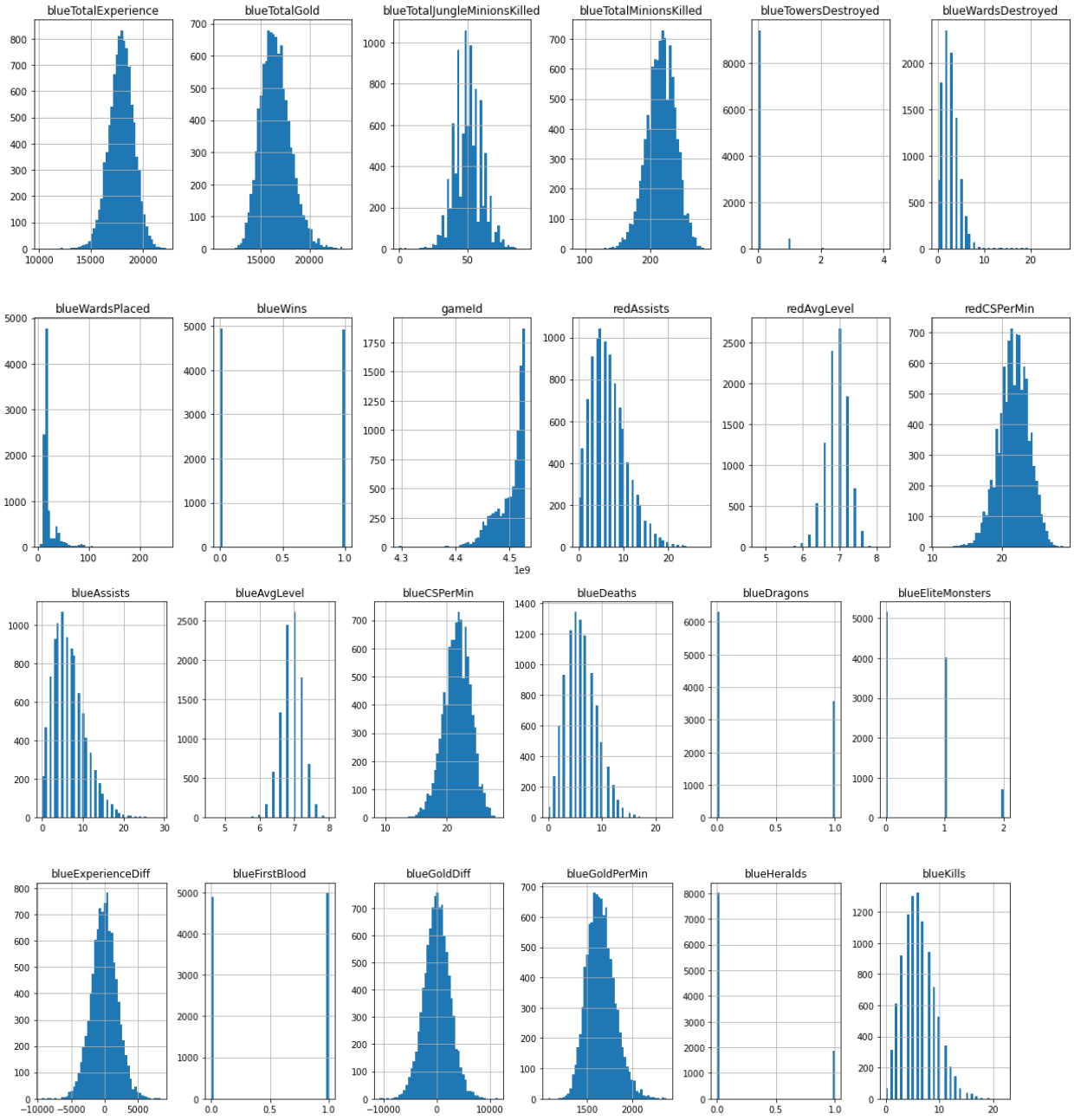
-There is no NA value in our dataset.

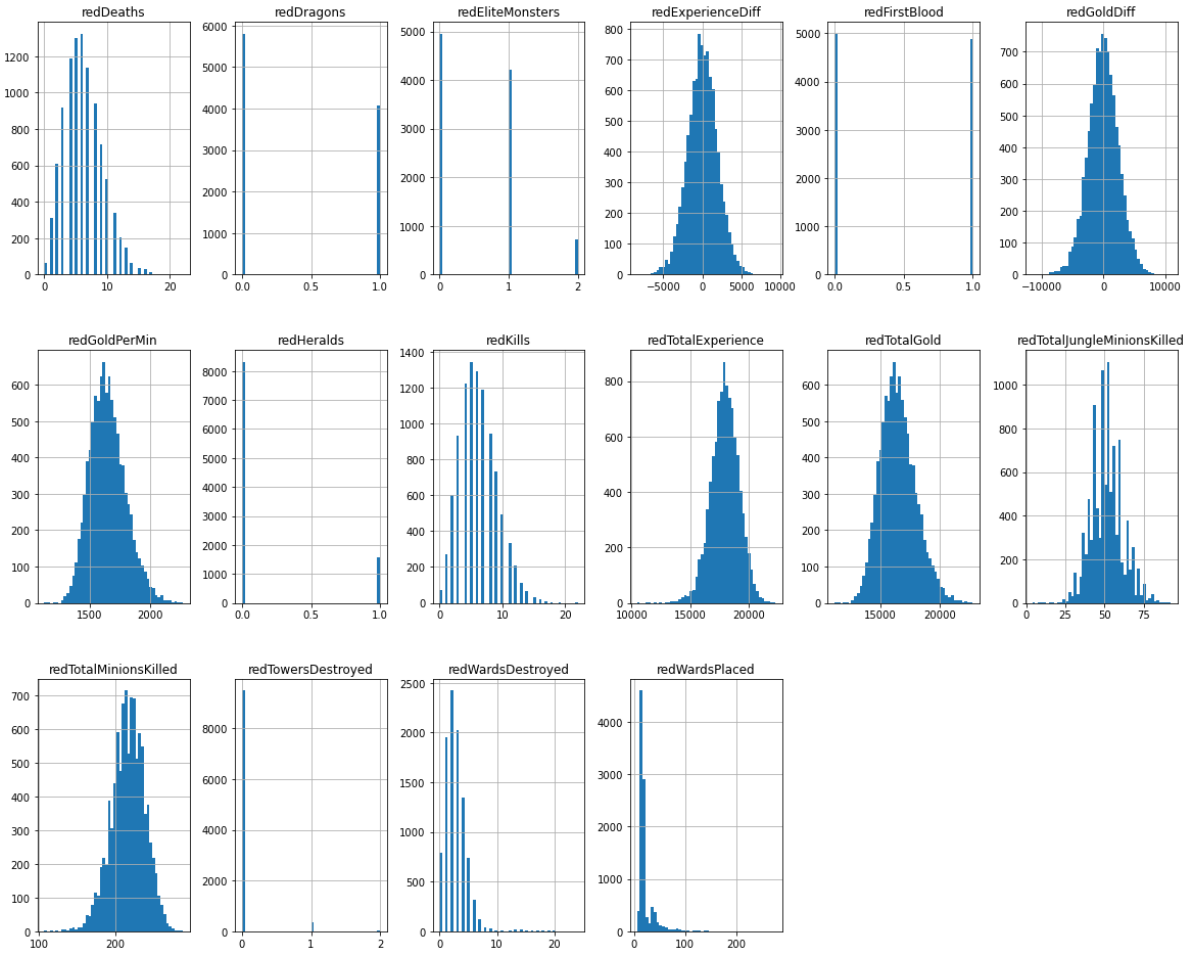
Input: np.where(pd.isnull(data))
Output: `(array([], dtype=int64), array([], dtype=int64))`

-Here is the histograms of the attributes.

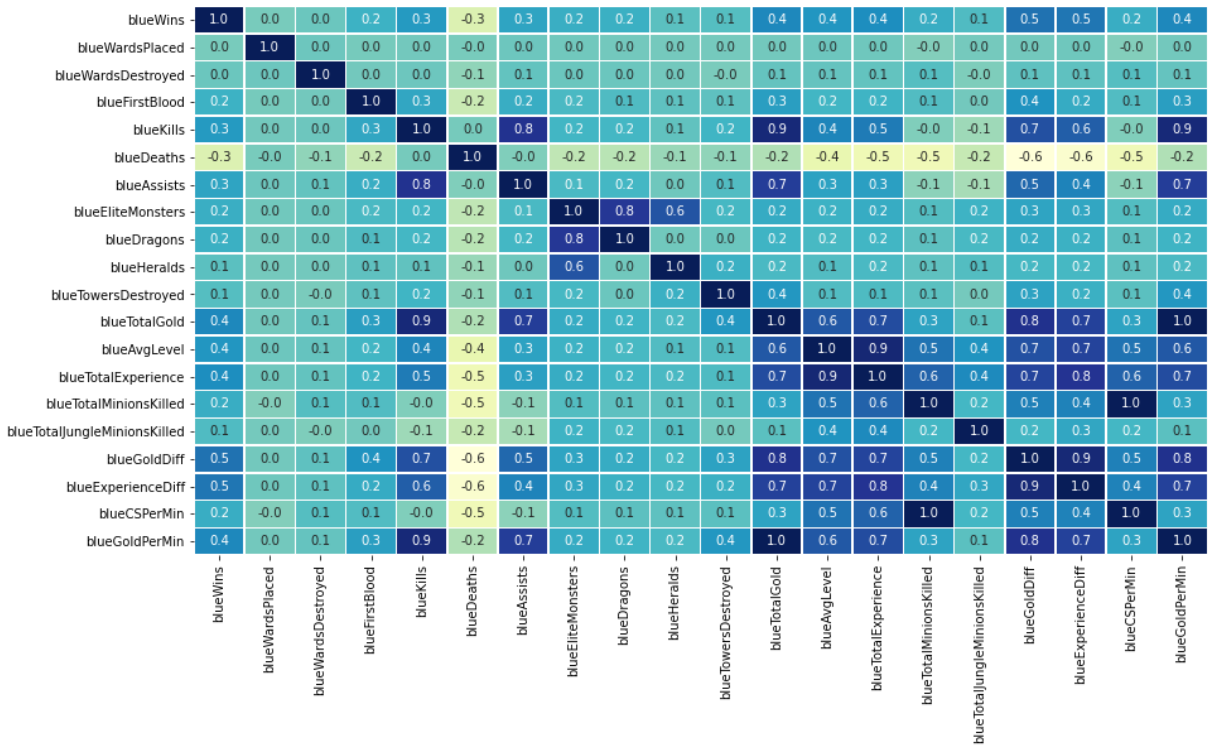
Input: data.hist(bins = 50, figsize = (20,40))
plt.show()

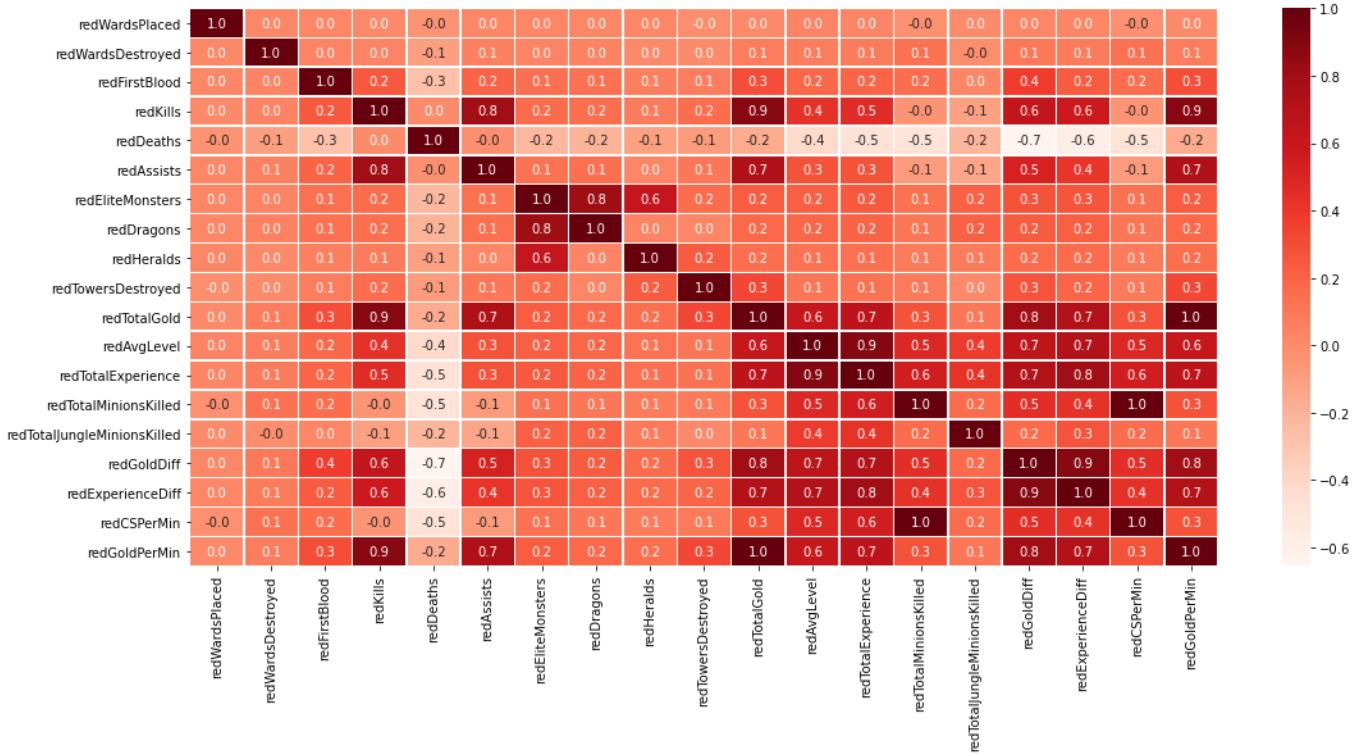
Output:


blueTotalExperience
blueTotalGold
blueTotalJungleMinionsKilled
blueTotalMinionsKilled
blueTowersDestroyed
blueWardsDestroyed
blueWardsPlaced
blueWins
gameId
1e9
redAssists
redAvgLevel
redCSPerMin
blueAssists
blueAvgLevel
blueCSPerMin
blueDeaths
blueDragons
blueEliteMonsters
blueExperienceDiff
blueFirstBlood
blueGoldDiff
blueGoldPerMin
blueHeralds
blueKills



-We also created two correlation matrices. One is for Blue Team and the other one is for Red Team.


blueWins 1.0 0.0 0.0 0.2 0.3 -0.3 0.3 0.2 0.2 0.1 0.1 0.4 0.4 0.4 0.2 0.1 0.5 0.5 0.2 0.4
blueWardsPlaced 0.0 1.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 -0.0 0.0
blueWardsDestroyed 0.0 0.0 1.0 0.0 0.0 -0.1 0.1 0.0 0.0 0.0 -0.0 0.1 0.1 0.1 0.1 -0.0 0.1 0.1 0.1 0.1
blueFirstBlood 0.2 0.0 0.0 1.0 0.3 -0.2 0.2 0.2 0.1 0.1 0.1 0.3 0.2 0.2 0.1 0.0 0.4 0.2 0.1 0.3
blueKills 0.3 0.0 0.0 0.3 1.0 0.0 0.8 0.2 0.2 0.1 0.2 0.9 0.4 0.5 -0.0 -0.1 0.7 0.6 -0.0 0.9
blueDeaths -0.3 -0.0 -0.1 -0.2 0.0 1.0 -0.0 -0.2 -0.2 -0.1 -0.1 -0.2 -0.4 -0.5 -0.5 -0.2 -0.6 -0.6 -0.5 -0.2
blueAssists 0.3 0.0 0.1 0.2 0.8 -0.0 1.0 0.1 0.2 0.0 0.1 0.7 0.3 0.3 -0.1 -0.1 0.5 0.4 -0.1 0.7
blueEliteMonsters 0.2 0.0 0.0 0.2 0.2 -0.2 0.1 1.0 0.8 0.6 0.2 0.2 0.2 0.2 0.1 0.2 0.3 0.3 0.1 0.2
blueDragons 0.2 0.0 0.0 0.1 0.2 -0.2 0.2 0.8 1.0 0.0 0.0 0.2 0.2 0.2 0.1 0.2 0.2 0.2 0.1 0.2
blueHeralds 0.1 0.0 0.0 0.1 0.1 -0.1 0.0 0.6 0.0 1.0 0.2 0.2 0.1 0.2 0.1 0.1 0.2 0.2 0.1 0.2
blueTowersDestroyed 0.1 0.0 -0.0 0.1 0.2 -0.1 0.1 0.2 0.0 0.2 1.0 0.4 0.1 0.1 0.1 0.0 0.3 0.2 0.1 0.4
blueTotalGold 0.4 0.0 0.1 0.3 0.9 -0.2 0.7 0.2 0.2 0.2 0.4 1.0 0.6 0.7 0.3 0.1 0.8 0.7 0.3 1.0
blueAvgLevel 0.4 0.0 0.1 0.2 0.4 -0.4 0.3 0.2 0.2 0.1 0.1 0.6 1.0 0.9 0.5 0.4 0.7 0.7 0.5 0.6
blueTotalExperience 0.4 0.0 0.1 0.2 0.5 -0.5 0.3 0.2 0.2 0.2 0.1 0.7 0.9 1.0 0.6 0.4 0.7 0.8 0.6 0.7
blueTotalMinionsKilled 0.2 -0.0 0.1 0.1 -0.0 -0.5 -0.1 0.1 0.1 0.1 0.1 0.3 0.5 0.6 1.0 0.2 0.5 0.4 1.0 0.3
blueTotalJungleMinionsKilled 0.1 0.0 -0.0 0.0 -0.1 -0.2 -0.1 0.2 0.2 0.1 0.0 0.1 0.4 0.4 0.2 1.0 0.2 0.3 0.2 0.1
blueGoldDiff 0.5 0.0 0.1 0.4 0.7 -0.6 0.5 0.3 0.2 0.2 0.3 0.8 0.7 0.7 0.5 0.2 1.0 0.9 0.5 0.8
blueExperienceDiff 0.5 0.0 0.1 0.2 0.6 -0.6 0.4 0.3 0.2 0.2 0.2 0.7 0.7 0.8 0.4 0.3 0.9 1.0 0.4 0.7
blueCSPerMin 0.2 -0.0 0.1 0.1 -0.0 -0.5 -0.1 0.1 0.1 0.1 0.1 0.3 0.5 0.6 1.0 0.2 0.5 0.4 1.0 0.3
blueGoldPerMin 0.4 0.0 0.1 0.3 0.9 -0.2 0.7 0.2 0.2 0.2 0.4 1.0 0.6 0.7 0.3 0.1 0.8 0.7 0.3 1.0
blueWins
blueWardsPlaced
blueWardsDestroyed
blueFirstBlood
blueKills
blueDeaths
blueAssists
blueEliteMonsters
blueDragons
blueHeralds
blueTowersDestroyed
blueTotalGold
blueAvgLevel
blueTotalExperience
blueTotalMinionsKilled
blueTotalJungleMinionsKilled
blueGoldDiff
blueExperienceDiff
blueCSPerMin
blueGoldPerMin
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


redWardsPlaced 1.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 -0.0 0.0 0.0 0.0 -0.0 0.0
redWardsDestroyed 0.0 1.0 0.0 0.0 -0.1 0.1 0.0 0.0 0.0 0.0 0.1 0.1 0.1 0.1 -0.0 0.1 0.1 0.1 0.1
redFirstBlood 0.0 0.0 1.0 0.2 -0.3 0.2 0.1 0.1 0.1 0.1 0.3 0.2 0.2 0.2 0.0 0.4 0.2 0.2 0.3
redKills 0.0 0.0 0.2 1.0 0.0 0.8 0.2 0.2 0.1 0.2 0.9 0.4 0.5 -0.0 -0.1 0.6 0.6 -0.0 0.9
redDeaths -0.0 -0.1 -0.3 0.0 1.0 -0.0 -0.2 -0.2 -0.1 -0.1 -0.2 -0.4 -0.5 -0.5 -0.2 -0.7 -0.6 -0.5 -0.2
redAssists 0.0 0.1 0.2 0.8 -0.0 1.0 0.1 0.1 0.0 0.1 0.7 0.3 0.3 -0.1 -0.1 0.5 0.4 -0.1 0.7
redEliteMonsters 0.0 0.0 0.1 0.2 -0.2 0.1 1.0 0.8 0.6 0.2 0.2 0.2 0.2 0.1 0.2 0.3 0.3 0.1 0.2
redDragons 0.0 0.0 0.1 0.2 -0.2 0.1 0.8 1.0 0.0 0.0 0.2 0.2 0.2 0.1 0.2 0.2 0.2 0.1 0.2
redHeralds 0.0 0.0 0.1 0.1 -0.1 0.0 0.6 0.0 1.0 0.2 0.2 0.1 0.1 0.1 0.1 0.2 0.2 0.1 0.2
redTowersDestroyed -0.0 0.0 0.1 0.2 -0.1 0.1 0.2 0.0 0.2 1.0 0.3 0.1 0.1 0.1 0.0 0.3 0.2 0.1 0.3
redTotalGold 0.0 0.1 0.3 0.9 -0.2 0.7 0.2 0.2 0.2 0.3 1.0 0.6 0.7 0.3 0.1 0.8 0.7 0.3 1.0
redAvgLevel 0.0 0.1 0.2 0.4 -0.4 0.3 0.2 0.2 0.1 0.1 0.6 1.0 0.9 0.5 0.4 0.7 0.7 0.5 0.6
redTotalExperience 0.0 0.1 0.2 0.5 -0.5 0.3 0.2 0.2 0.1 0.1 0.7 0.9 1.0 0.6 0.4 0.7 0.8 0.6 0.7
redTotalMinionsKilled -0.0 0.1 0.2 -0.0 -0.5 -0.1 0.1 0.1 0.1 0.1 0.3 0.5 0.6 1.0 0.2 0.5 0.4 1.0 0.3
redTotalJungleMinionsKilled 0.0 -0.0 0.0 -0.1 -0.2 -0.1 0.2 0.2 0.1 0.0 0.1 0.4 0.4 0.2 1.0 0.2 0.3 0.2 0.1
redGoldDiff 0.0 0.1 0.4 0.6 -0.7 0.5 0.3 0.2 0.2 0.3 0.8 0.7 0.7 0.5 0.2 1.0 0.9 0.5 0.8
redExperienceDiff 0.0 0.1 0.2 0.6 -0.6 0.4 0.3 0.2 0.2 0.2 0.7 0.7 0.8 0.4 0.3 0.9 1.0 0.4 0.7
redCSPerMin -0.0 0.1 0.2 -0.0 -0.5 -0.1 0.1 0.1 0.1 0.1 0.3 0.5 0.6 1.0 0.2 0.5 0.4 1.0 0.3
redGoldPerMin 0.0 0.1 0.3 0.9 -0.2 0.7 0.2 0.2 0.2 0.3 1.0 0.6 0.7 0.3 0.1 0.8 0.7 0.3 1.0
redWardsPlaced
redWardsDestroyed
redFirstBlood
redKills
redDeaths
redAssists
redEliteMonsters
redDragons
redHeralds
redTowersDestroyed
redTotalGold
redAvgLevel
redTotalExperience
redTotalMinionsKilled
redTotalJungleMinionsKilled
redGoldDiff
redExperienceDiff
redCSPerMin
redGoldPerMin
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

In the light of these correlation matrices, we dropped 8 columns:

Input: columns = np.full((corr.shape[0],), True, dtype=bool)

```
for i in range(corr.shape[0]):
    for j in range(i+1, corr.shape[0]):
        if corr.iloc[i,j] >= 0.9:
            if columns[j]:
                columns[j] = False
selected_columns = data.columns[columns]
data = data[selected_columns]
```

Output:

```
gameId: int64,
blueWins: int64,
blueWardsPlaced: int64,
blueWardsDestroyed: int64,
blueFirstBlood: int64,
blueKills: int64,
blueDeaths: int64,
blueAssists: int64,
blueEliteMonsters: int64,
blueDragons: int64,
blueHeralds: int64,
blueTowersDestroyed: int64,
blueTotalGold: int64,
blueAvgLevel: float64,
blueTotalMinionsKilled: int64,
blueTotalJungleMinionsKilled: int64,
blueGoldDiff: int64,
blueExperienceDiff: int64,
redWardsPlaced: int64,
redWardsDestroyed: int64,
redFirstBlood: int64,
redAssists: int64,
redEliteMonsters: int64,
redDragons: int64,
redHeralds: int64,
redTowersDestroyed: int64,
```

```
redTotalGold: int64,
redAvgLevel: float64,
redTotalMinionsKilled: int64,
redTotalJungleMinionsKilled: int64,
redGoldDiff: int64,
redExperienceDiff: int64,

Dataframe column number: 32
```

REVISION: In preprocess phase, we eliminated 8 columns which have over 0.9 similarity. But when we were applying classification & clustering algorithms, we noticed that eliminating attributes which have over 0.75 similarity gives better results. So we changed it. The columns are left after this process:

Input:

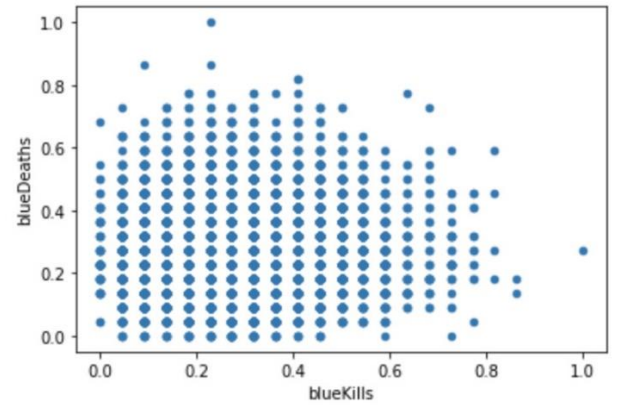
```
columns = np.full((corr.shape[0],), True, dtype=bool)
for i in range(corr.shape[0]):
    for j in range(i+1, corr.shape[0]):
        if corr.iloc[i,j] >= 0.75:
            if columns[j]:
                columns[j] = False
selected_columns = data.columns[columns]
data = data[selected_columns]
```

Output:

```
blueWins: int64,
blueWardsPlaced: int64,
blueWardsDestroyed: int64,
blueFirstBlood: int64,
blueKills: int64,
blueDeaths: int64,
blueEliteMonsters: int64,
blueHeralds: int64,
blueTowersDestroyed: int64,
blueAvgLevel: float64,
blueTotalMinionsKilled: int64,
blueTotalJungleMinionsKilled: int64,
redWardsPlaced: int64,
redWardsDestroyed: int64,
redFirstBlood: int64,
redEliteMonsters: int64,
redHeralds: int64,
redTowersDestroyed: int64,
redAvgLevel: float64,
redTotalMinionsKilled: int64,
redTotalJungleMinionsKilled: int64,
```

We removed the outliers with the code below:

Before:



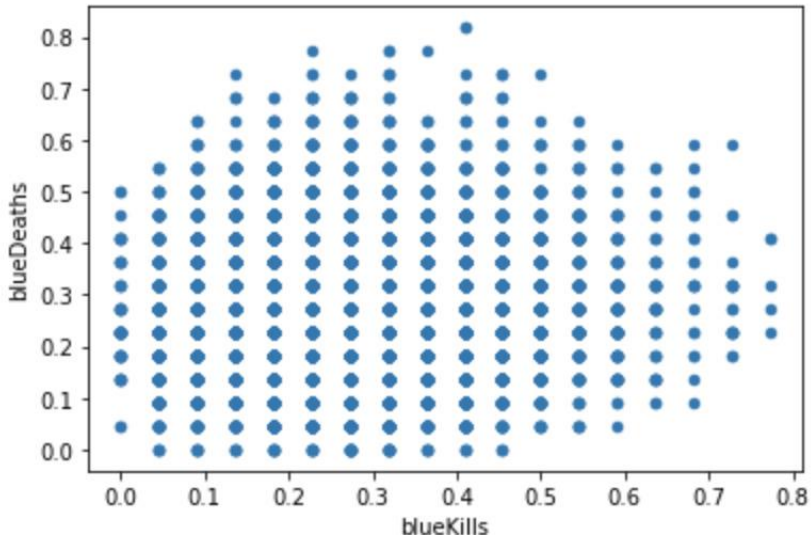
Input:

```
clean_df = pd.DataFrame(normalized_df,columns=['blueKills', 'blueDeaths'])
clean_df.info()

Q1 = data.quantile(q=.25)
Q3 = data.quantile(q=.75)
IQR = data.apply(stats.iqr)
clean_df = clean_df[~((data < (Q1-1.5*IQR)) | (clean_df > (Q3+1.5*IQR))).any(axis=1)]

clean_df.plot(x ='blueKills', y='blueDeaths', kind = 'scatter')
plt.show()
```

Output:



We splitted our data into two parts which are called B and W. B contains all attributes except the blueWins attribute. W only contains blueWins attribute.

Input:

```
B = data.drop(columns=['blueWins'])
W = data['blueWins']
```

We normalized the df B with min-max normalization.

Input:

```
B=(B-B.min())/(B.max()-B.min())
```

Finally, we splitted our data set to train and test sets with the code below:

Input:

```
B_train, B_test, W_train, W_test = train_test_split(B, W, test_size=0.2, random_state=20)
```

4) METHODS

As we discussed in literature, there are 4 methods used which are Gaussian Naïve Bayes, Decision Tree, Random Forest, and KNN. Since the Decision Tree algorithm has the least accuracy, we eliminated this and worked with Gaussian Naïve Bayes, Random Forest and K-Nearest Neighbor algorithms.

4.1 Gaussian Naïve Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

We used the following libraries:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
```

4.2 Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

class `sklearn.ensemble.RandomForestClassifier`(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*)

We used the library: from sklearn.ensemble import RandomForestClassifier

4.3 K-Nearest Neighbor

sklearn.neighbors provides functionality for unsupervised and supervised neighbors-based learning methods. Unsupervised nearest neighbors is the foundation of many other learning methods, notably manifold learning and spectral clustering. Supervised neighbors-based learning comes in two flavors: classification for data with discrete labels, and regression for data with continuous labels.

class `sklearn.neighbors.KNeighborsClassifier`(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs*)

We used the library: from sklearn.neighbors import KNeighborsClassifier

5)RESULTS

We got 3 different accuracy results with 3 different classification algorithms. With Gaussian Naïve Bayes, we got 70.24291% accuracy.

Input:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score


clf_nb = GaussianNB()
clf_nb.fit(B_train, W_train)

pred_nb = clf_nb.predict(B_test)

acc_nb = accuracy_score(pred_nb, W_test)
print(acc_nb)
```

Output: `0.7024291497975709`

With Random Forest, we got 71.65991% accuracy.

Input:

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()

grid = {'n_estimators':[100,200,300,400,500], 'max_depth': [2, 5, 10]}

clf_rf = GridSearchCV(rf, grid, cv=5)
clf_rf.fit(B_train, W_train)

pred_rf = clf_rf.predict(B_test)
```

```
acc_rf = accuracy_score(pred_rf, W_test)
print(acc_rf)
```

Output: `0.7165991902834008`

With KNN, we got 71.356275% accuracy.

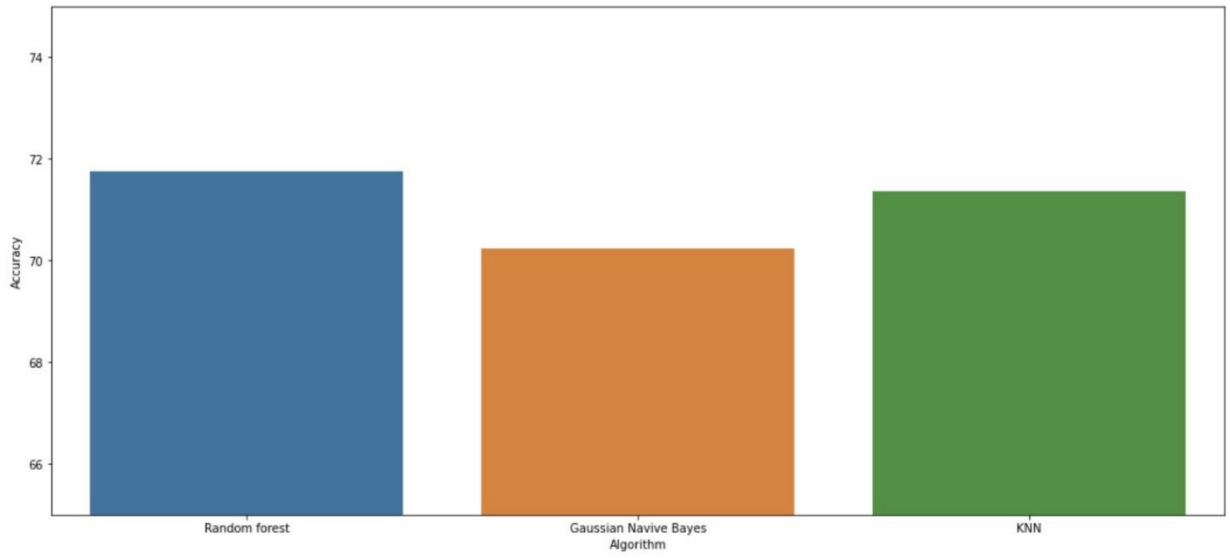
Input:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()

grid = {"n_neighbors":np.arange(1,100)}
clf_knn = GridSearchCV(knn, grid, cv=5)
clf_knn.fit(B_train,W_train)


pred_knn = clf_knn.predict(B_test)
acc_knn = accuracy_score(pred_knn, W_test)
print(acc_knn)
```

Output: `0.7135627530364372`

If we compare the accuracy rates:
Random Forest > KNN > Gaussian Naïve Bayes.



6)CONCLUSION

First of all, we made a literature review to see the previous works with the League of Legends Diamond Ranked Games (10 min) dataset. There were 80+ notebooks on Kaggle but we picked 5 of them and went over them in depth.

There were 4 different algorithms and many different results. The algorithms were Gaussian Naïve Bayes, Decision Tree, Random Forest, and KNN. We decided to work with Gaussian Naïve Bayes, Random Forest, and KNN.

We preprocessed our data. There were 40 columns first. Then we eliminate columns by correlation matrix and there were 20 columns left. There were no NA values. We removed outliers. We split our data into train and test data. B refers to the df that includes attributes and W refers to the target column. After that, we normalized data. And finally, we applied 3 algorithms to our data.

We got 70.24% accuracy with Gaussian Naïve Bayes, 71.65% with Random Forest, 71.35% with K-Nearest Neighbor. Random Forest gave the highest accuracy.

When we compared our results with the notebooks on Kaggle we specified in the literature part, the results look closer to each other. There are no big differences. This differences may be caused by the preprocessing phase. Some data scientists created new attributes by using the actual attributes instead of using both of them. For example;

Actual attributes:

```
blueTotalJungleMinionsKilled: int64,
redTotalJungleMinionsKilled: int64,
```

The new attribute created:

```
JungleMinionsKilledDiff = |blueTotalJungleMinionsKilled –
redTotalJungleMinionsKilled|
```

Scaling method is an another issue which may effect the result. It depends on the data scientists preference.

GAUSSIAN NAIVE BAYES	***ACCURACY***
Our Result	70.24%
https://www.kaggle.com/hrideshkohli/advanced-feature-selection-analysis-classification	71.56%

https://www.kaggle.com/c0011006cyc/game-result-prediction-with-gnb-knn-dt#05	73.43%
https://www.kaggle.com/xiyuewang/lol-how-to-win	71.76%

RANDOM FOREST	***ACCURACY***
Our Result	71.65 %
https://www.kaggle.com/hrideshkohli/advanced-feature-selection-analysis-classification	71.05%
https://www.kaggle.com/xiyuewang/lol-how-to-win	72.82%
https://www.kaggle.com/kevinnolasco/predicting-wins-in-league-of-legends	72.06%

KNN	***ACCURACY***
Our Result	71.35%
https://www.kaggle.com/c0011006cyc/game-result-prediction-with-gnb-knn-dt#05	69.84%
https://www.kaggle.com/xiyuewang/lol-how-to-win	71.71%
https://www.kaggle.com/wallacecorrea/knn-modeling-for-match-result-prediction-75-3	75.3%

7)REFERENCES

[1] https://www.kaggle.com/hrideshkohli/advanced-feature-selection-analysis-classification

[2] https://www.kaggle.com/c0011006cyc/game-result-prediction-with-gnb-knn-dt#05

[3] https://www.kaggle.com/xiyuewang/lol-how-to-win

[4] https://www.kaggle.com/kevinnolasco/predicting-wins-in-league-of-legends

[5] https://www.kaggle.com/wallacecorrea/knn-modeling-for-match-result-prediction-75-3