```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files unc

import os
for dirname, _, filenames in os.walk('/content/'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the
```

```
/content/high_diamond_ranked_10min.csv
/content/.config/gce
/content/.config/active_config
/content/.config/config_sentinel
/content/.config/.last_opt_in_prompt.yaml
/content/.config/.last_update_check.json
/content/.config/.last_survey_prompt.yaml
/content/.config/.metricsUUID
/content/.config/configurations/config_default
/content/.config/logs/2020.12.02/22.03.37.873126.log
/content/.config/logs/2020.12.02/22.04.38.150307.log
/content/.config/logs/2020.12.02/22.04.21.823807.log
/content/.config/logs/2020.12.02/22.04.13.854338.log
/content/.config/logs/2020.12.02/22.03.59.234441.log
/content/.config/logs/2020.12.02/22.04.37.441505.log
/content/sample_data/anscombe.json
/content/sample_data/README.md
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_train_small.csv
/content/sample_data/california_housing_test.csv
/content/sample_data/mnist_test.csv
```

```
#import libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

Import Data

Data I will be using for this project is "high_diamond_ranked_10min.csv" from
https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min

It contains columns of "blueWins" which is my target variable, and other variables such as "blueKills" or "blueTotalGold", and I will be trying to figure out what components are most important in winning a game of league during first 10 minutes of the game. Also, I will be comparing accuracy of following algorithms. ["Logistic Regression", "Decision Tree Classification", "XGBoost", "Neural Network (Keras)", "Neural Network (Pytorch)"]

```
# import data
url = '/content/high_diamond_ranked_10min.csv'

data = pd.read_csv(url, index_col=0)
data.head()
```

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

```
# data types are mostly int64, and some are float64
data.info()
```

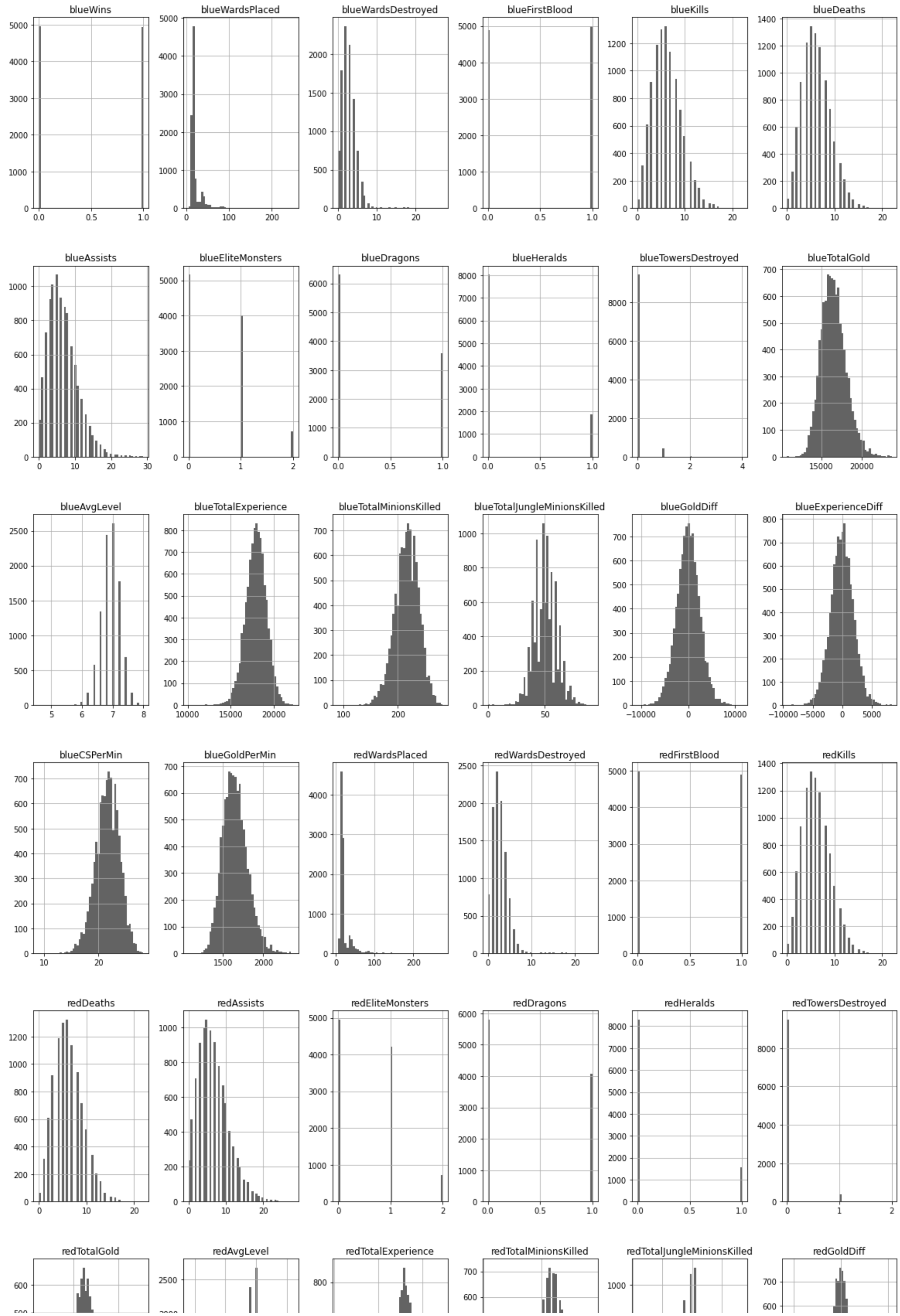
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9879 entries, 4519157822 to 4523772935
Data columns (total 39 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   blueWins                      9879 non-null   int64  
 1   blueWardsPlaced               9879 non-null   int64  
 2   blueWardsDestroyed            9879 non-null   int64  
 3   blueFirstBlood                9879 non-null   int64  
 4   blueKills                     9879 non-null   int64  
 5   blueDeaths                    9879 non-null   int64  
 6   blueAssists                   9879 non-null   int64  
 7   blueEliteMonsters             9879 non-null   int64  
 8   blueDragons                   9879 non-null   int64  
 9   blueHeralds                   9879 non-null   int64  
 10  blueTowersDestroyed           9879 non-null   int64  
 11  blueTotalGold                 9879 non-null   int64  
 12  blueAvgLevel                  9879 non-null   float64
 13  blueTotalExperience           9879 non-null   int64  
 14  blueTotalMinionsKilled        9879 non-null   int64  
 15  blueTotalJungleMinionsKilled  9879 non-null   int64  
```

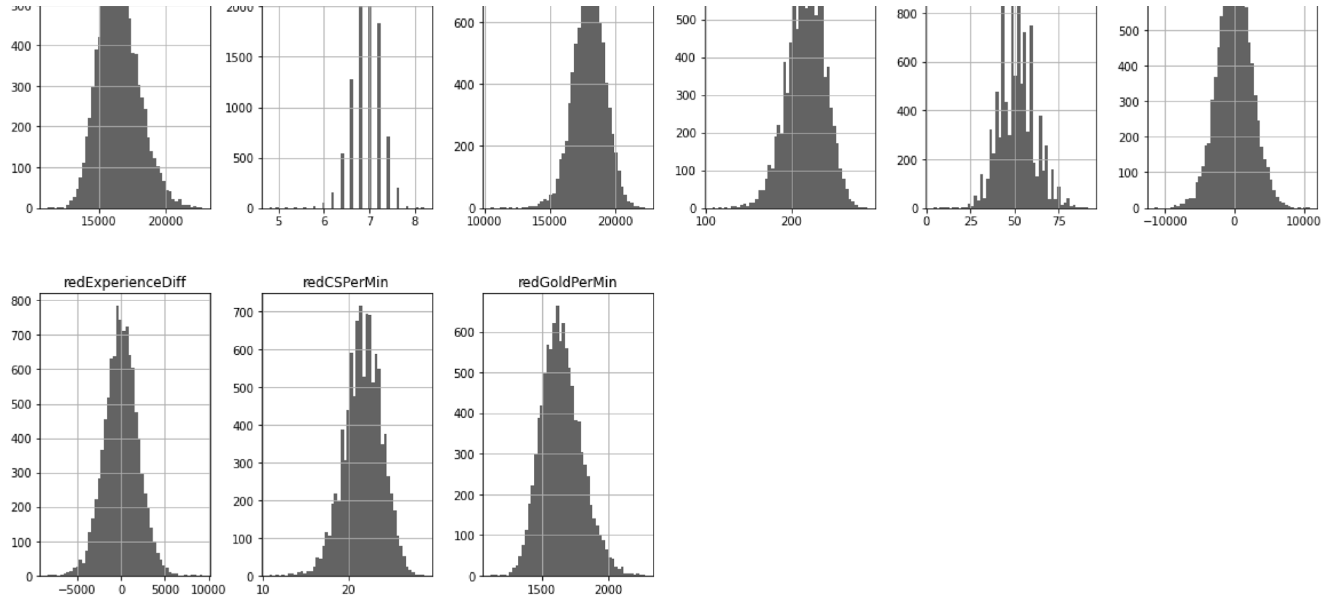
```
 16  blueGoldDiff                  9879 non-null   int64
 17  blueExperienceDiff            9879 non-null   int64
 18  blueCSPerMin                  9879 non-null   float64
 19  blueGoldPerMin                9879 non-null   float64
 20  redWardsPlaced                9879 non-null   int64
 21  redWardsDestroyed             9879 non-null   int64
 22  redFirstBlood                 9879 non-null   int64
 23  redKills                      9879 non-null   int64
 24  redDeaths                     9879 non-null   int64
 25  redAssists                    9879 non-null   int64
 26  redEliteMonsters              9879 non-null   int64
 27  redDragons                    9879 non-null   int64
 28  redHeralds                    9879 non-null   int64
 29  redTowersDestroyed            9879 non-null   int64
 30  redTotalGold                  9879 non-null   int64
 31  redAvgLevel                   9879 non-null   float64
 32  redTotalExperience            9879 non-null   int64
 33  redTotalMinionsKilled         9879 non-null   int64
 34  redTotalJungleMinionsKilled   9879 non-null   int64
 35  redGoldDiff                   9879 non-null   int64
 36  redExperienceDiff             9879 non-null   int64
 37  redCSPerMin                   9879 non-null   float64
 38  redGoldPerMin                 9879 non-null   float64
dtypes: float64(6), int64(33)
memory usage: 3.0 MB
```

```
# no NULL values exist in the dataset
data.isna().values.any()
```

```
False
```

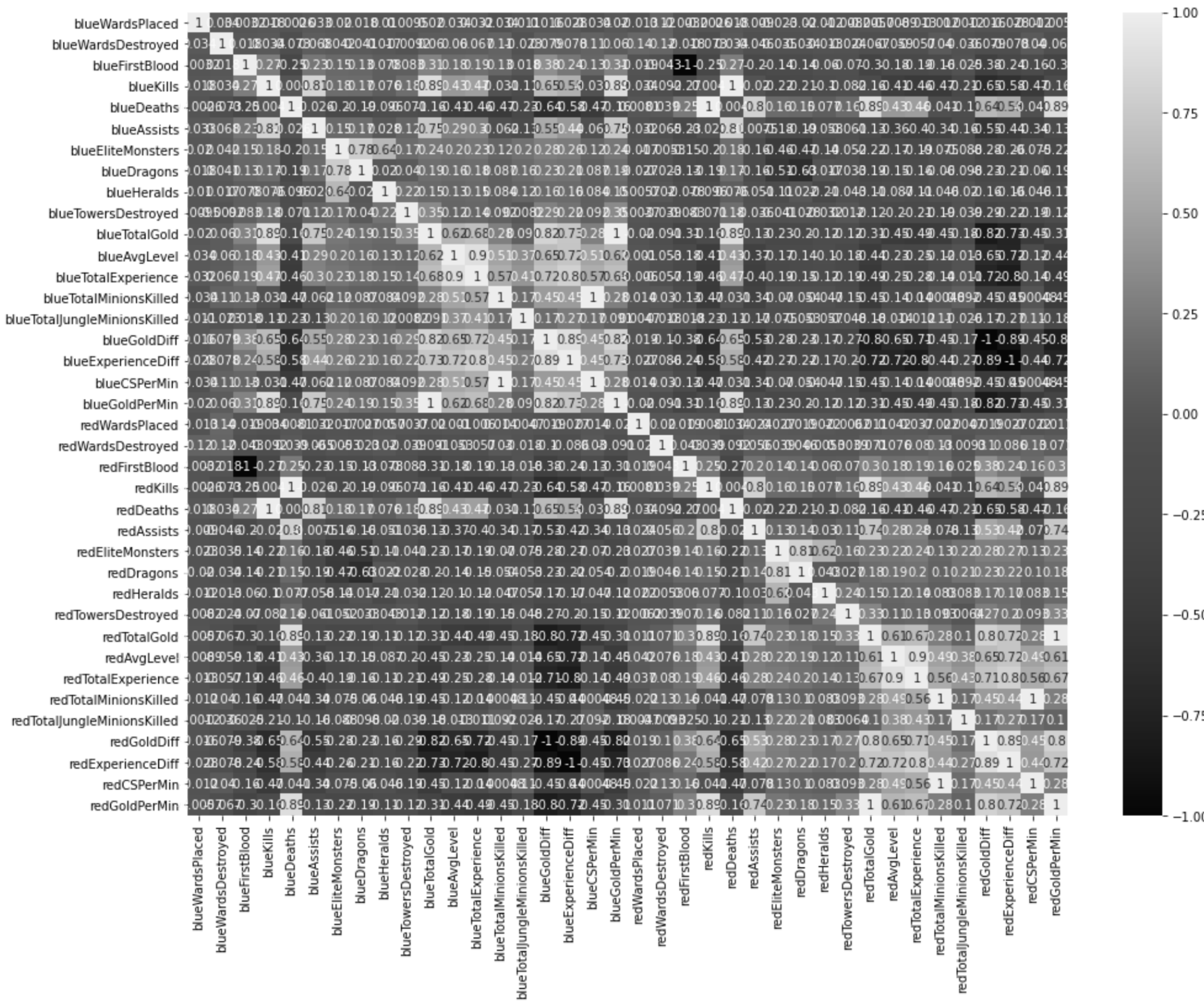
```
# Check histogram to see if data is normally distributed.
data.hist(bins = 50, figsize = (20,40))
plt.show()
```


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



Most of data is normally distributed so now I'm ready to pre process the data.

```
# check heatmap for correlations
import seaborn as sns
plt.figure(figsize=(16, 12))
temp = data.copy()
sns.heatmap(temp.drop('blueWins', axis=1).corr(), annot=True);
```



```
# find highly correlated values with correlation over 0.9
```

```
# Find highly correlated values with correlation over 0.9
corr = pd.DataFrame(temp.corr().unstack().sort_values().drop_duplicates())
corr.columns = ['cr']
corr[(corr['cr'] > 0.9) | (corr['cr'] < -0.9)]
```

		cr
blueExperienceDiff	**redExperienceDiff**	-1.000000
blueTotalExperience	**blueAvgLevel**	0.901297
redAvgLevel	**redTotalExperience**	0.901748
redCSPerMin	**redCSPerMin**	1.000000
blueTotalGold	**blueGoldPerMin**	1.000000
blueTotalMinionsKilled	**blueCSPerMin**	1.000000
redCSPerMin	**redTotalMinionsKilled**	1.000000
redTotalGold	**redGoldPerMin**	1.000000

Preparing Data

```
# Drop highly correlated values and reformat data.
df = data.copy()

df['ExperienceDiff'] = df['blueExperienceDiff']
df['blueWardScore'] = df['blueWardsPlaced'] + df['blueWardsDestroyed']
df['redWardScore'] = df['redWardsPlaced'] + df['redWardsDestroyed']

df = df.drop(columns=['blueWardsPlaced', 'blueWardsDestroyed', 'redWardsPlaced', 'redWardsDes
df = df.drop(columns=['blueExperienceDiff', 'redExperienceDiff', 'blueTotalMinionsKilled', 'r
df = df.drop(columns=['blueFirstBlood', 'redFirstBlood'])
df.head()
```

	blueWins	blueKills	blueDeaths	blueAssists	blueEliteMonsters	blueDrago
gameId						
4519157822	0	9	6	11	0	
4523371949	0	5	5	5	0	
4521474530	0	7	11	4	1	
4524384067	0	4	5	5	1	
4436033771	0	6	6	6	0	

Machine Learning Algorithms

```
# create trainset and test set
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score

# X is variables, y is the target "blueWins"
X = df.drop('blueWins', axis = 1)
y = df['blueWins']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state = 42)
X_scaled = preprocessing.scale(X_train)
```

Logistic Regression

```
# train
from sklearn.linear_model import LogisticRegression
Logreg = LogisticRegression(random_state = 1)
scores = cross_val_score(Logreg, X_scaled, y_train, scoring = 'accuracy', cv = 10)
meanScore = scores.mean()

print("Logistic Regression model has Accuracy of", meanScore)
```

```
Logistic Regression model has Accuracy of 0.7316215653955097
```

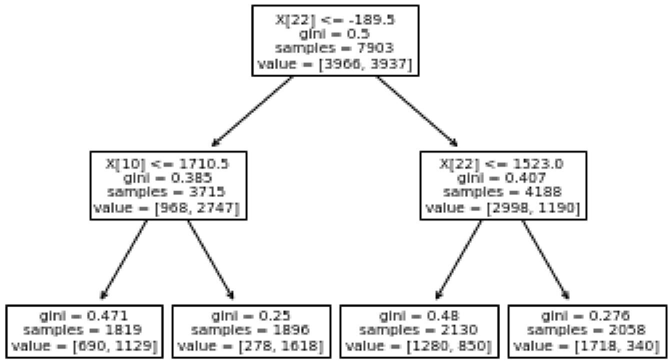
Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
decisionTree = DecisionTreeClassifier(max_depth = 2, random_state = 3)

scores = cross_val_score(decisionTree, X_scaled, y_train, scoring = 'accuracy', cv = 10)
meanScore = scores.mean()
print("Decision Tree model has Accuracy of", meanScore)
```

```
Decision Tree model has Accuracy of 0.7258005408951976
```

```
#  plot decision tree
from sklearn import tree
decisionTree.fit(X_train, y_train)
decisionTree.predict(X_test)
tree.plot_tree(decisionTree);
```



XGBoost

```
from xgboost import XGBClassifier

xgb = XGBClassifier(n_estimators = 100 ,learning_rate = 0.1)

scores = cross_val_score(xgb, X_scaled, y_train, scoring = 'accuracy', cv = 10)
meanScore = scores.mean()
print("XGBoost model has Accuracy of", meanScore)
```

```
XGBoost model has Accuracy of 0.728458448686969
```

```
# Displaying Feature Importance
from matplotlib import pyplot

xgb.fit(X_train, y_train)

fi = xgb.feature_importances_
pyplot.bar(range(len(fi)), fi)
for i, item in enumerate(X_train.columns.values):
  print(f'[{i}] {item} has feature importance of {fi[i]}')
pyplot.show()
```

```
[0] blueKills has feature importance of 0.023184478282928467
[1] blueDeaths has feature importance of 0.024283558130264282
[2] blueAssists has feature importance of 0.016227567568421364
[3] blueEliteMonsters has feature importance of 0.030031917616724968
[4] blueDragons has feature importance of 0.036023467779159546
[5] blueHeralds has feature importance of 0.006249427329748869
[6] blueTowersDestroyed has feature importance of 0.0
[7] blueTotalGold has feature importance of 0.024360589683055878
[8] blueTotalExperience has feature importance of 0.02345031499862671
[9] blueTotalJungleMinionsKilled has feature importance of 0.02545071765780449
[10] blueGoldDiff has feature importance of 0.40110206604003906
[11] blueCSPerMin has feature importance of 0.01830883137881756
[12] redKills has feature importance of 0.0
[13] redDeaths has feature importance of 0.0
[14] redAssists has feature importance of 0.023921972140669823
[15] redEliteMonsters has feature importance of 0.038663946092128754
[16] redDragons has feature importance of 0.05770396068692207
[17] redHeralds has feature importance of 0.0
[18] redTowersDestroyed has feature importance of 0.006170305423438549
[19] redTotalGold has feature importance of 0.03477129340171814
[20] redTotalExperience has feature importance of 0.023670561611652374
[21] redTotalJungleMinionsKilled has feature importance of 0.017487380653619766
[22] redGoldDiff has feature importance of 0.0
[23] redCSPerMin has feature importance of 0.014581427909433842
[24] ExperienceDiff has feature importance of 0.11639149487018585
[25] blueWardScore has feature importance of 0.02030184306204319
[26] redWardScore has feature importance of 0.017662758007645607
```

Neural Network (Keras)


```
# import libraries
from tensorflow.keras.layers import Activation, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import optimizers
from tensorflow import keras
from keras.models import Model
```

```
k_df = df;
y=data['blueWins']
k_df.drop(['blueWins'],1,inplace=True)
k_df
```

	blueKills	blueDeaths	blueAssists	blueEliteMonsters	blueDragons	blueHe
gameId						
4519157822	9	6	11	0	0	
4523371949	5	5	5	0	0	
4521474530	7	11	4	1	1	
4524384067	4	5	5	1	0	
4436033771	6	6	6	0	0	
...	...	...	...	...	...	
4527873286	7	4	5	1	1	
4527797466	6	4	8	1	1	
4527713716	6	7	5	0	0	

```
# Normalize Z-Score
mean = k_df.mean(axis=0)
std = k_df.std(axis=0)
n_df = (k_df-mean)/std
n_df.head()
```

	blueKills	blueDeaths	blueAssists	blueEliteMonsters	blueDragons	blueHe
gameId						
4519157822	0.935254	-0.046924	1.071441	-0.879186	-0.753188	-0.4
4523371949	-0.393196	-0.387777	-0.404748	-0.879186	-0.753188	-0.4
4521474530	0.271029	1.657340	-0.650779	0.719467	1.327556	-0.4
4524384067	-0.725309	-0.387777	-0.404748	0.719467	-0.753188	2.0
4436033771	-0.061084	-0.046924	-0.158716	-0.879186	-0.753188	-0.4

```
model = Sequential()
model.add(Dense(32, input_shape=(27,)))
model.add(Dense(16, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

opt = optimizers.Adam(learning_rate=0.005)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
```

```
history = model.fit(k_df, y, batch_size=100, epochs=4, validation_split=0.2)
```

```
Epoch 1/4
```

```
80/80 [==============================] - 0s 3ms/step - loss: 207.7623 - accuracy: 0.640
Epoch 2/4
80/80 [==============================] - 0s 2ms/step - loss: 81.6577 - accuracy: 0.6411
Epoch 3/4
80/80 [==============================] - 0s 2ms/step - loss: 54.2966 - accuracy: 0.6429
Epoch 4/4
80/80 [==============================] - 0s 2ms/step - loss: 57.8162 - accuracy: 0.6370
```

Neural Network (Pytorch)

```
# import libraries
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader, TensorDataset, random_split


# get data
targets = data[['blueWins']].values
features = df.values

test_size = int(.10 * 9879)
val_size = test_size
train_size = 9879 - test_size*2
train_size , val_size, test_size


dataset = TensorDataset(torch.tensor(features).float(), torch.from_numpy(targets).float())
pt_train, val_df, test_df = random_split(dataset, [train_size, val_size, test_size])


# train
input_size = 27
output_size = 1
threshold = 0.5
batch_size = 128

train_loader = DataLoader(pt_train, batch_size, shuffle=True)
val_loader = DataLoader(val_df, batch_size)
test_loader = DataLoader(test_df, batch_size)



class PTModel(nn.Module):
    def __init__(self):
        # initiate the model
        super().__init__()
        self.linear = nn.Linear(input_size, output_size)
        self.sigmoid = nn.Sigmoid()
```

```
    def forward(self, xb):
        # forward function of the model
        out = self.sigmoid(self.linear(xb))
        return out

    def training_step(self, batch):
        # used for training per batch in an eopch
        inputs, labels = batch
        out = self(inputs)
        loss = F.binary_cross_entropy(out, labels)
        return loss

    def validation_step(self, batch):
        # used on function `evaluate` to iterate model through a batch
        inputs, labels = batch
        out = self(inputs)
        loss = F.binary_cross_entropy(out, labels)
        acc = accuracy(out, labels)
        # `.detach()` makes sure gradient is not tracked
        return {'val_loss': loss.detach(), 'val_acc' : acc.detach()}

    def validation_epoch_end(self, outputs):
        # calculate mean loss and accuracy for batch called w/ `evaluate`
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()
        return {'val_loss': epoch_loss.item(), 'val_acc' : epoch_acc.item()}

    def epoch_end(self, epoch, result, num_epochs):
        # print function to see what's going on
        if ((epoch+1) % 10 == 0) or (epoch == (num_epochs-1)):
            # print for every 5 epochs
            print("Epoch [{}], val_loss: {:.4f}, val_acc {:.4f}".format(epoch+1, result['val_


def accuracy(out, labels):
    return torch.tensor(torch.sum(abs(out-labels) < threshold).item() / len(out))


def evaluate(model, val_loader):
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.Adam):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
        # Training Phase
        for batch in train_loader:
            loss = model.training_step(batch)
            loss.backward()
```

```
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = evaluate(model, val_loader)
        model.epoch_end(epoch, result, epochs)
        history.append(result)
    return history
```

```
model = PTModel()
```

```
evaluate(model, val_loader)
```

```
{'val_acc': 0.5196707844734192, 'val_loss': 47.962921142578125}
```

```
history = fit(750, .0001, model, train_loader, val_loader)
```

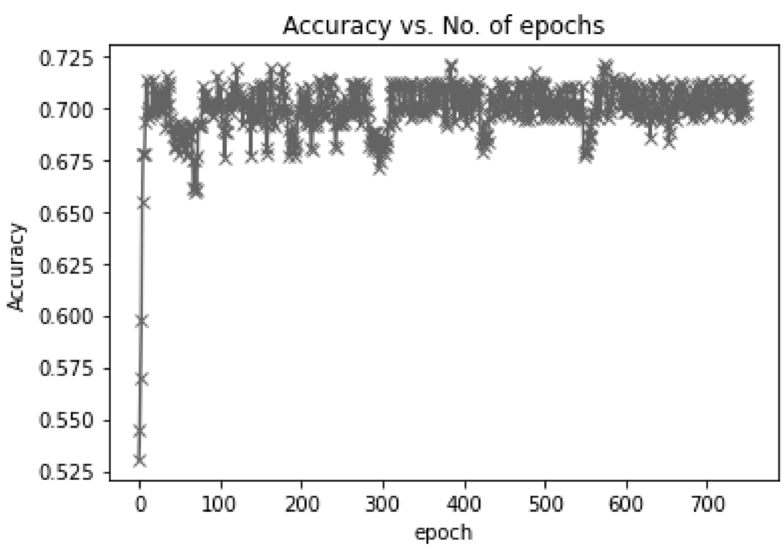
```
Epoch [170], val_loss: 29.5756, val_acc 0.6975
Epoch [180], val_loss: 29.7912, val_acc 0.6965
Epoch [190], val_loss: 30.3845, val_acc 0.6838
Epoch [200], val_loss: 29.3401, val_acc 0.6990
Epoch [210], val_loss: 28.5654, val_acc 0.7053
Epoch [220], val_loss: 29.7157, val_acc 0.6926
Epoch [230], val_loss: 29.5699, val_acc 0.6955
Epoch [240], val_loss: 28.5225, val_acc 0.7068
Epoch [250], val_loss: 29.5266, val_acc 0.6994
Epoch [260], val_loss: 29.4678, val_acc 0.6965
Epoch [270], val_loss: 29.4470, val_acc 0.6984
Epoch [280], val_loss: 28.5136, val_acc 0.7088
Epoch [290], val_loss: 30.9558, val_acc 0.6818
Epoch [300], val_loss: 30.6221, val_acc 0.6858
Epoch [310], val_loss: 28.6539, val_acc 0.7063
Epoch [320], val_loss: 29.4313, val_acc 0.6984
Epoch [330], val_loss: 28.9348, val_acc 0.7033
Epoch [340], val_loss: 27.9253, val_acc 0.7111
Epoch [350], val_loss: 29.5502, val_acc 0.6975
Epoch [360], val_loss: 29.4030, val_acc 0.6994
Epoch [370], val_loss: 28.7855, val_acc 0.7072

Epoch [380], val_loss: 30.2449, val_acc 0.6926
Epoch [390], val_loss: 28.7986, val_acc 0.7014
Epoch [400], val_loss: 28.6612, val_acc 0.7063
Epoch [410], val_loss: 28.6477, val_acc 0.7053
Epoch [420], val_loss: 28.2484, val_acc 0.7121
Epoch [430], val_loss: 30.6520, val_acc 0.6867
Epoch [440], val_loss: 29.4583, val_acc 0.6965
Epoch [450], val_loss: 29.3331, val_acc 0.6984
Epoch [460], val_loss: 28.6547, val_acc 0.7063
Epoch [470], val_loss: 28.4228, val_acc 0.7102
Epoch [480], val_loss: 28.6666, val_acc 0.7072
Epoch [490], val_loss: 28.0429, val_acc 0.7125
Epoch [500], val_loss: 28.2432, val_acc 0.7111
Epoch [510], val_loss: 28.9489, val_acc 0.7014
Epoch [520], val_loss: 28.1857, val_acc 0.7092
Epoch [530], val_loss: 28.8941, val_acc 0.7014
```

```
Epoch [540], val_loss: 29.3458, val_acc 0.6975
Epoch [550], val_loss: 31.4287, val_acc 0.6770
Epoch [560], val_loss: 29.7187, val_acc 0.6936
Epoch [570], val_loss: 29.3972, val_acc 0.6975
Epoch [580], val_loss: 29.1011, val_acc 0.7004
Epoch [590], val_loss: 28.2096, val_acc 0.7111
Epoch [600], val_loss: 29.3266, val_acc 0.6975
Epoch [610], val_loss: 28.7067, val_acc 0.7024
Epoch [620], val_loss: 29.9221, val_acc 0.6936
Epoch [630], val_loss: 29.9238, val_acc 0.6936
Epoch [640], val_loss: 29.2544, val_acc 0.6994
Epoch [650], val_loss: 28.5419, val_acc 0.7102
Epoch [660], val_loss: 28.8631, val_acc 0.7014
Epoch [670], val_loss: 28.1938, val_acc 0.7111
Epoch [680], val_loss: 28.6216, val_acc 0.7033
Epoch [690], val_loss: 28.1122, val_acc 0.7111
Epoch [700], val_loss: 28.6225, val_acc 0.7043
Epoch [710], val_loss: 28.6370, val_acc 0.7053
Epoch [720], val_loss: 29.0957, val_acc 0.7004
Epoch [730], val_loss: 28.6441, val_acc 0.7072
Epoch [740], val_loss: 28.1553, val_acc 0.7131
Epoch [750], val_loss: 29.3391, val_acc 0.6975
```

```
accuracies = [r['val_acc'] for r in history]
plt.plot(accuracies, '-x')
plt.xlabel('epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy vs. No. of epochs')
```

```
Text(0.5, 1.0, 'Accuracy vs. No. of epochs')
```




```
evaluate(model, test_loader)
```

```
{'val_acc': 0.6933808326721191, 'val_loss': 29.978294372558594}
```

Conclusion

```
After trying 5 different Algorithms
we got the results of


*   Logistic Regression : 0.7316215653955097
*   Decision Tree : 0.7258005408951976
*   XGBoost : 0.728458448686969
*   Neural Network (Keras) : 0.6430
*   Neural Network (Pytorch) : 0.6933808326721
```

After trying 5 different Algorithms we got the results of

- Logistic Regression : 0.7316215653955097
- Decision Tree : 0.7258005408951976
- XGBoost : 0.728458448686969
- Neural Network (Keras) : 0.6430
- Neural Network (Pytorch) : 0.6933808326721191