PUBG Finish Placement Prediction

MACHINE LEARNING PROJECT REPORT

Team Name:- BGMI

Team members:- Rittik Panda (MT2022090)

Hardeep Singh Arora (MT2022047)

Under the Guidance of Dr. Dinesh Babu Jayagopi and Neelam Sinha

&

TA :- Preyas Garg

Table of Content

PROBLEM STATEMENT

- We are provided with a large number of anonymized PUBG game stats, formatted so that each row contains one player's post-game stats. The data comes from matches of all types: solos, duos, squads, and custom; there is no guarantee of there being 100 players per match, nor at most 4 player per group.
- This project aims at creating a model which predicts players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place).

DATASET DESCRIPTION

Train.csv	Test.csv
Shape:- 3112876 rows	Shape:- 1334090 rows
29 columns	28 columns

Data fields:

- DBNOs - Number of enemy players knocked.
- assists - Number of enemy players this player damaged that were killed by teammates.
- boosts - Number of boost items used.
- damageDealt - Total damage dealt. Note: Self inflicted damage is subtracted.
- headshotKills - Number of enemy players killed with headshots.
- heals - Number of healing items used.
- Id - Player’s Id
- killPlace - Ranking in match of number of enemy players killed.
- killPoints - Kills-based external ranking of player. (Think of this as an Elo ranking where only kills matter.) If there is a value other than -1 in rankPoints, then any 0 in killPoints should be treated as a “None”.
- killStreaks - Max number of enemy players killed in a short amount of time.
- kills - Number of enemy players killed.
- longestKill - Longest distance between player and player killed at time of death. This may be misleading, as downing a player and driving away may lead to a large longestKill stat.
- matchDuration - Duration of match in seconds.

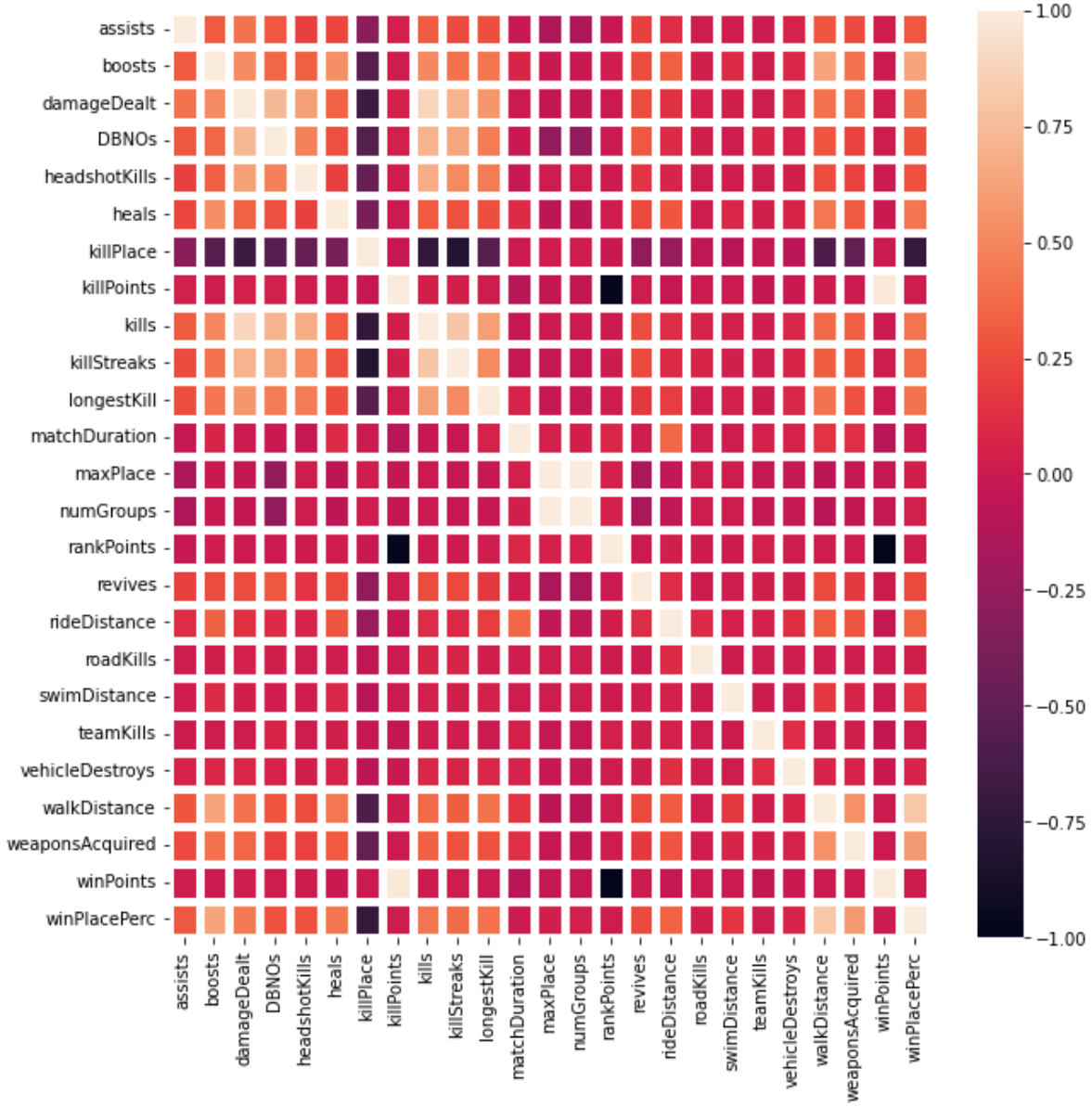
- matchId - ID to identify match. There are no matches that are in both the training and testing set.
- matchType - String identifying the game mode that the data comes from. The standard modes are “solo”, “duo”, “squad”, “solo-fpp”, “duo-fpp”, and “squad-fpp”; other modes are from events or custom matches.
- rankPoints - Elo-like ranking of player. This ranking is inconsistent and is being deprecated in the API’s next version, so use with caution. Value of -1 takes place of “None”.
- revives - Number of times this player revived teammates.
- rideDistance - Total distance traveled in vehicles measured in meters.
- roadKills - Number of kills while in a vehicle.
- swimDistance - Total distance traveled by swimming measured in meters.
- teamKills - Number of times this player killed a teammate.
- vehicleDestroys - Number of vehicles destroyed.
- walkDistance - Total distance traveled on foot measured in meters.
- weaponsAcquired - Number of weapons picked up.
- winPoints - Win-based external ranking of player. (Think of this as an Elo ranking where only winning matters.) If there is a value other than -1 in rankPoints, then any 0 in winPoints should be treated as a “None”.
- groupId - ID to identify a group within a match. If the same group of players plays in different matches, they will have a different groupId each time.
- numGroups - Number of groups we have data for in the match.
- maxPlace - Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
- winPlacePerc - The target of prediction. This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match. It is calculated off of maxPlace, not numGroups, so it is possible to have missing chunks in a match.
- The target field ‘winPlacePerc’ is present in the train.csv only.

EXPLORATORY DATA ANALYSIS (eda) and data visualization

The heatmap correlation helped us to visualize the strength of relationships between the target data field and various other data fields.

After checking the heatmap we got to know that there were some data fields which were highly correlated with the target(winPlacePerc) :-

- Boosts
- damageDealt
- killPlace
- Kills
- walkDistance
- weaponsAcquired



FEATURE ENGINEERING

- Fortunately in the given dataset there was only 1 row which had target field as null, hence we dropped that row.
- After understanding the game properly we found some useful insights, hence it gave us some light to add some interesting features which became potential characteristics for prediction.
- New features added:-
    - medicKits (heals+boosts) - ‘medickits’ are total no. of heals and boosts picked up by a player.
    - totalDistance (swimDistance+rideDistance+walkDistance) - By adding swimDistance,rideDistance,walkDistance we created a new feature called ‘totalDistance’

- footSpeed (walkDistance/matchDuration) - 'footSpeed' represents an approximation by dividing 'walkDistance' by total 'matchDuration' , which works fine for this project.
- OtherDistances – It is the sum of 'swimDistance' and 'rideDistance'.
- headshotKillrate (headshotKills / kills) - It represents the ratio of 'headshotkills' and 'totalkills' of a player. It gives us the insight about the player's skill level.
- Players_in_match (grouped matchId) - Total no. of players joined in a match.We created it by grouping the players using matchId.
- damageDealtKill (damageDealt/kills) - It gives us the amount of damage dealt by a player to kill an opponent. It gives us the insight about the player's skill level.
- teamHelping (assists+revives) - It is the no. of assists and revives done by a player , which tells us about the how much the player has helped his or her team.
- kills_without_moving – It is a boolean feature , which indicates if the player has killed multiple players without moving at all . By using it we can identify a potential cheater.
- normalizedKills – This feature indicates how many more enemies a player could have killed if total 100 players would have joined in the server.
- NormalizedDamageDealt - This feature indicates amount of probable damage dealt by a player if total 100 players would have joined in the game.
- NormalizedTeamHelping - This feature indicates how many more revives and assists a player could have done if total 100 players would have joined in the server.
- NormalizedKillPlace - This feature indicates probable 'killPlace' of a player if total 100 players would have joined in the server.
- NormalizedLongestKill - This feature indicates probable 'longestKill' of a player if total 100 players would have joined in the server.
- NormalizedKillStreaks - This feature indicates probable 'killStreaks' of a player if total 100 players would have joined in the server.
- NormalizedDBNOs – This feature indicates how many more enemy players a palyer could have knocked if maximum (i.e.100) no. Of players would have joined the game.
- NormalizedVehicleDestroys - This feature indicates how many more vehicle a palyer could have destroyed if total100 players would have joined the game.

As we were going through each feature and visualizing them against target using various plots, we encountered outliers which we removed as a part of cleaning the dataset.

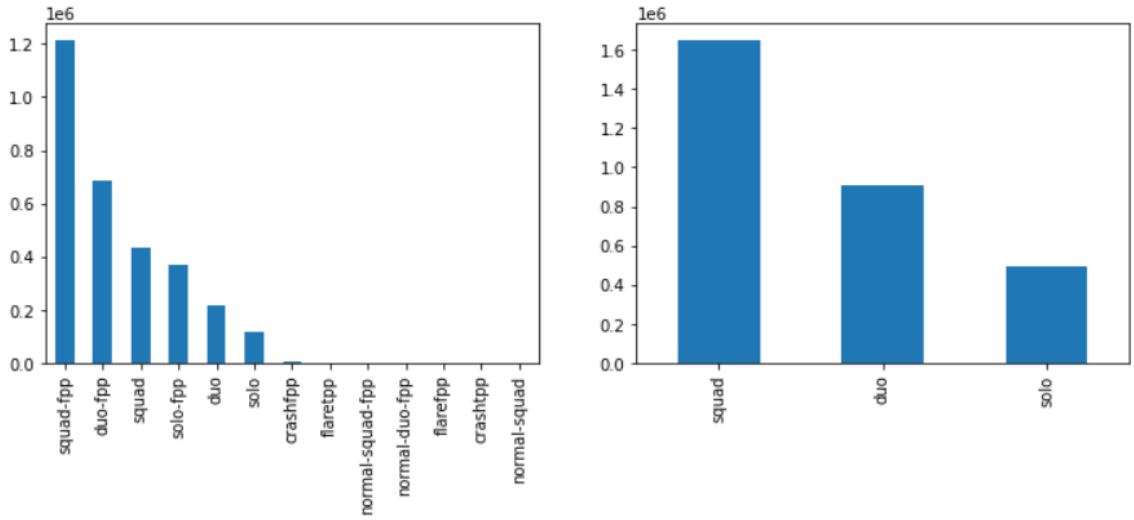
The outliers were removed from the following features:-

- Kills – We removed the rows which got above 15 kills by a player.
- medicKits - We removed the rows in which a player picked up more than 20 medicKits.

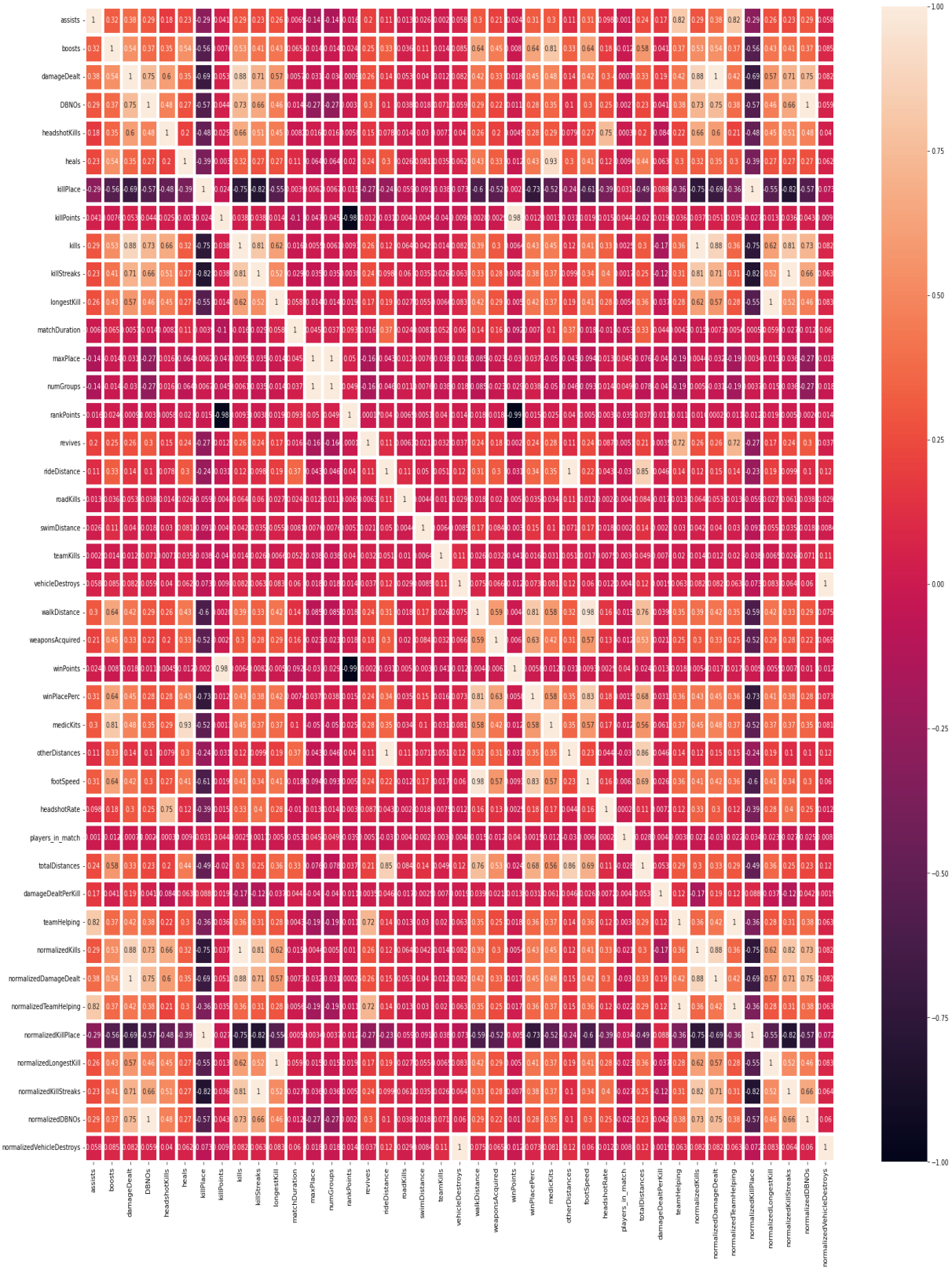
- weaponsAcquired - We removed the rows which contains a player , who picked up more than 10 weapons.
- footSpeed – We deleted those players' informations who walked with a speed more than 2.5 m/s , which indicates cheaters.
- totalDistance – We removed above 7500 , which is not possible in a PUBG match.
- matchDuration – We removed the matches which got match duration less than 300 seconds.
- killStreaks – We removed killStreaks above 3 , which is not possible in a real game.
- roadKills – We removed the rows with more than 3 no. of roadKills which rarely happens in a PUBG match.

Standardizing match type

- (squad, squad-fpp,normal-squad, normal-squad-fpp,flarefpp,flaretpp) to squad
- (duo, duo-fpp,normal-duo, normal-duo-fpp,crashfpp,crashtpp) to duo
- (solo, solo-fpp,normal-solo,normal-solo-fpp) to solo



- We created a new Boolean valued feature named kills_without_moving and used that to find potential cheaters or players with no possibility of winning.
- Then we used that feature to remove those rows, later we dropped that feature.
- We also removed some of the features which were irrelevant for the prediction or which had very low correlation with the target field, such as:- Id, groupId, matchId, winPoints, killPoints, rankPoints etc.



Heat Map With Newly Added Features and After Removing Outliers

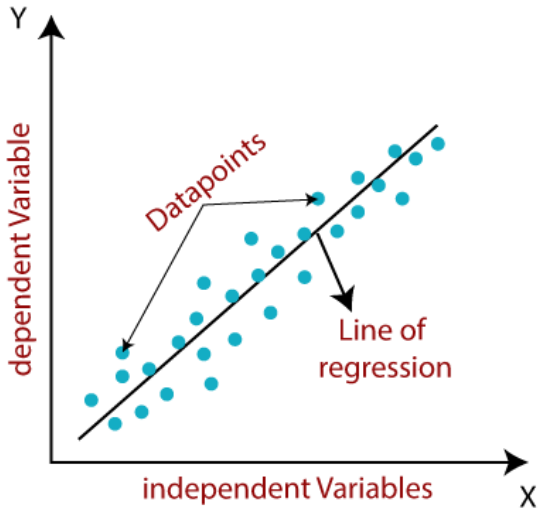
MODELS IMPLEMENTED

- Linear Regression
- XGBoost
- Random Forest
- Fully connected Neural Networks

Linear Regression:-

Linear regression is a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The case of one explanatory variable is called *simple linear regression*; for more than one, the process is called multiple linear regression.[1] This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

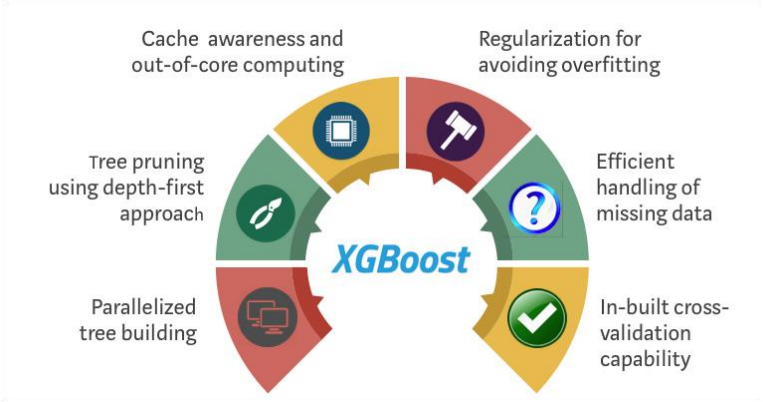
In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.



XGBoost:-

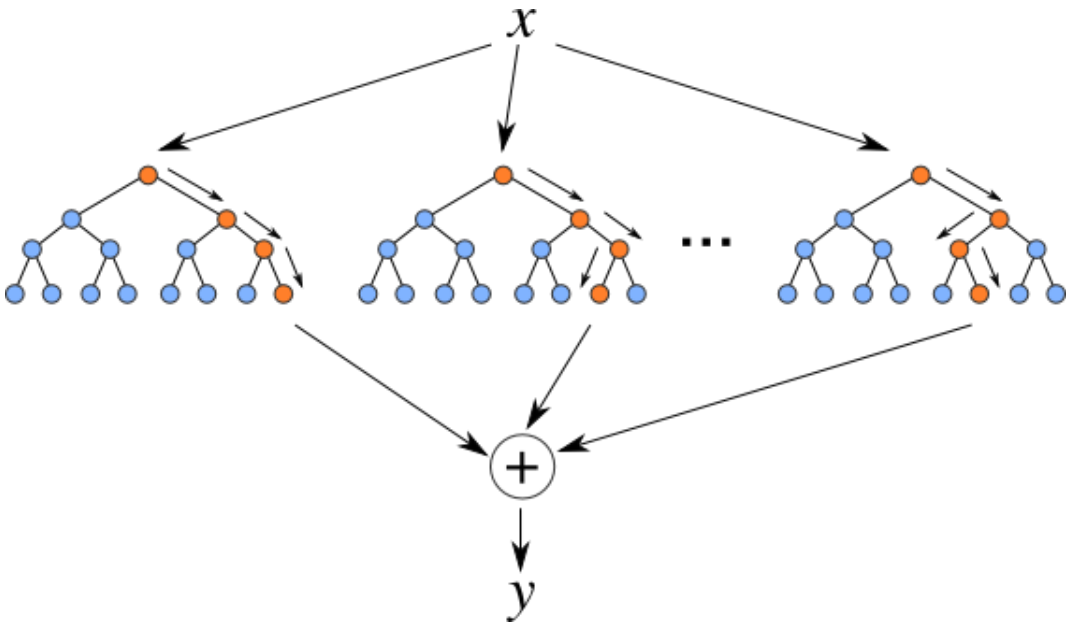
XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.



Random Forest:-

Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.
Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap.



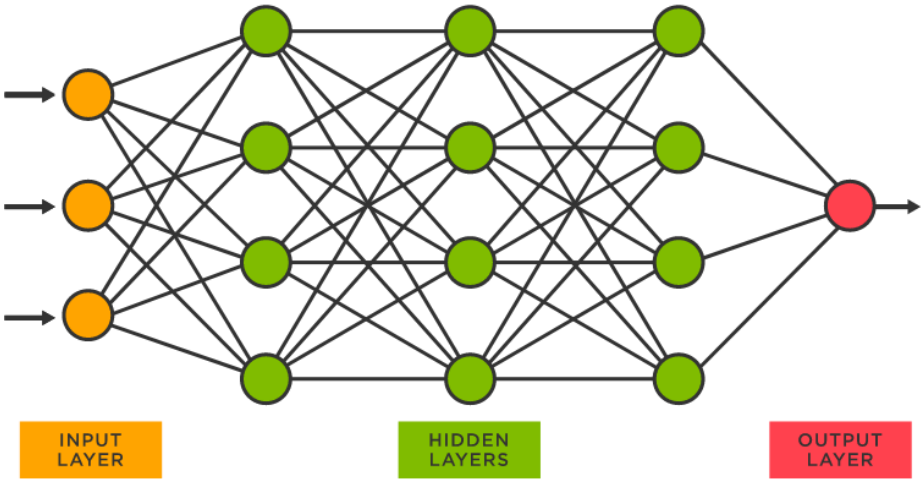
The diagram above shows the structure of a Random Forest. You can notice that the trees run in parallel with no interaction amongst them. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees.

Neural Networks:-

A **neural network** is a network or circuit of biological neurons, or, in a modern sense, an artificial neural network, composed of artificial neurons or nodes. Thus, a neural network is either a biological neural network, made up of biological neurons, or an artificial neural network, used for solving artificial intelligence (AI) problems. The connections of the biological neuron are modeled in artificial neural networks as weights between nodes. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information.

We scaled the data using Sklearn's StandardScaler() before using Neural Networks.



Benchmarking of Models (Hyperparameter Tuning):-

XGBoost

Min_child_weight	Max_depth	Learning_rate	Gamma	Colsample_bytree	Score	MSE
5	140	0.05	0.1	0.7	0.9360	0.00624
5	130	0.008	0.3	0.7	0.7095	0.02966
5	120	0.05	0.2	0.5	0.9311	0.00633
-	-	-	-	-	0.9306	0.00624

Random Forest

N_estimators	Min_samples_leaf	Max_features	N_jobs	MSE
80	4	Sqrt	-1	0.00778
120	3	Sqrt	-1	0.00786
160	4	Sqrt	-1	0.00774
200	4	Log2	-1	0.00775
180	3	Log2	-1	0.007756
150	4	Sqrt	-1	0.007754
-	-	-	-	0.00795

Neural Network

Models	No. of Hidden Layers	No. of Neurons at each Hidden Layer	MSE
Model 1	2	[128,64]	0.007682
Model 2	2	[64,64]	0.007318
Model 3	3	[64,32,16]	0.007695
Model 4	2	[32,32]	0.007933
Model 5	4	[64,32,32,16]	0.0077136
Model 6	4	[128,64,32,16]	0.007697

Neural Networks

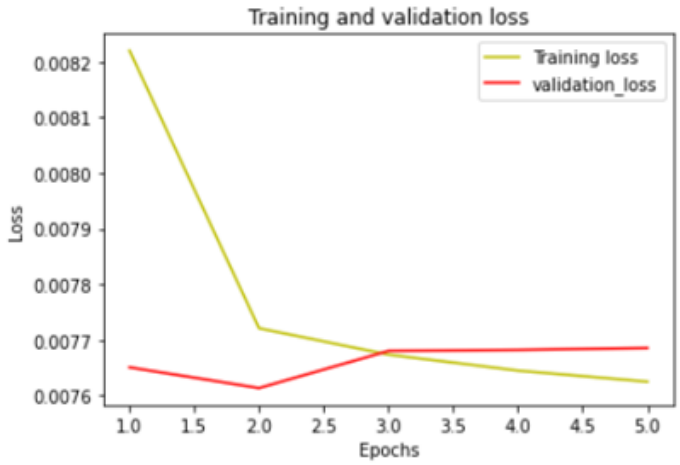
Model 1:-

```
model = Sequential()
model.add(Dense(128,input_dim=13,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='linear')) #output layer
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mae'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1792
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 10,113
Trainable params: 10,113
Non-trainable params: 0
```

```
history = model.fit(X_train_scaled,y_train,validation_split=0.2,epochs=5)
```

```
2022-12-09 04:44:58.882233: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)
Epoch 1/5
55954/55954 [==============================] - 102s 2ms/step - loss: 0.0083 - mae: 0.0673 - val_loss: 0.0077 - val_mae: 0.0653
Epoch 2/5
55954/55954 [==============================] - 94s 2ms/step - loss: 0.0077 - mae: 0.0654 - val_loss: 0.0076 - val_mae: 0.0647
Epoch 3/5
55954/55954 [==============================] - 94s 2ms/step - loss: 0.0077 - mae: 0.0652 - val_loss: 0.0075 - val_mae: 0.0647
Epoch 4/5
55954/55954 [==============================] - 93s 2ms/step - loss: 0.0077 - mae: 0.0650 - val_loss: 0.0076 - val_mae: 0.0654
Epoch 5/5
55954/55954 [==============================] - 91s 2ms/step - loss: 0.0076 - mae: 0.0649 - val_loss: 0.0076 - val_mae: 0.0651
```



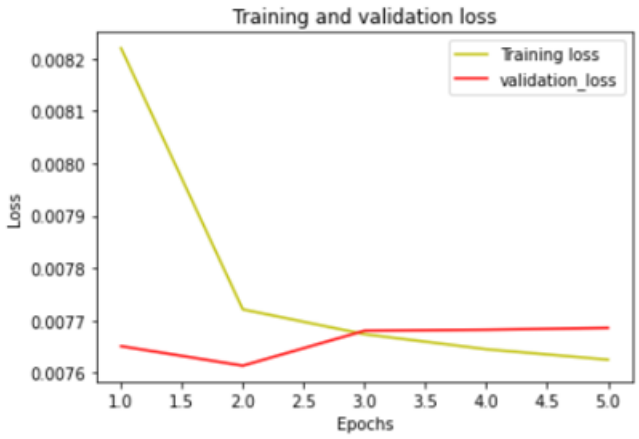
Model 2:-

```
model = Sequential()
model.add(Dense(64,input_dim=13,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='linear')) #output layer
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mae'])
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 64)                896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 5,121
Trainable params: 5,121
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(X_train_scaled,y_train,validation_split=0.2,epochs=5)
```

```
Epoch 1/5
55954/55954 [==============================] - 87s 2ms/step - loss: 0.0082 - mae: 0.0673 - val_loss: 0.0077 - val_mae: 0.0651
Epoch 2/5
55954/55954 [==============================] - 85s 2ms/step - loss: 0.0077 - mae: 0.0654 - val_loss: 0.0076 - val_mae: 0.0648
Epoch 3/5
55954/55954 [==============================] - 82s 1ms/step - loss: 0.0077 - mae: 0.0651 - val_loss: 0.0077 - val_mae: 0.0651
Epoch 4/5
55954/55954 [==============================] - 83s 1ms/step - loss: 0.0076 - mae: 0.0650 - val_loss: 0.0077 - val_mae: 0.0649
Epoch 5/5
55954/55954 [==============================] - 84s 1ms/step - loss: 0.0076 - mae: 0.0649 - val_loss: 0.0077 - val_mae: 0.0657
```



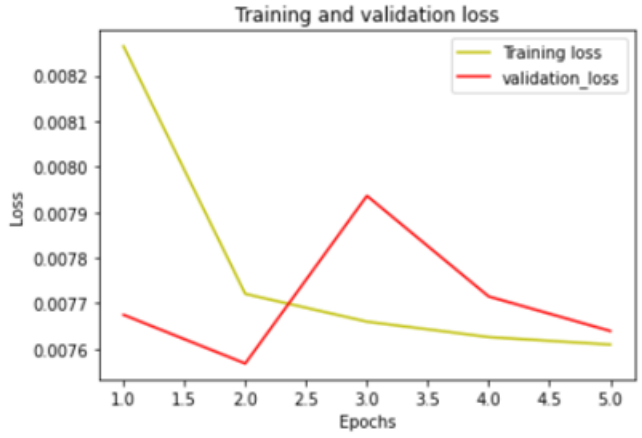
Model 3:-

```
model = Sequential()
model.add(Dense(64,input_dim=13,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(1,activation='linear')) #output layer
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mae'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 64)                896
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_8 (Dense)              (None, 16)                528
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 17
=================================================================
Total params: 3,521
Trainable params: 3,521
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(X_train_scaled,y_train,validation_split=0.2,epochs=5)
```

```
Epoch 1/5
55954/55954 [==============================] - 103s 2ms/step - loss: 0.0083 - mae: 0.0674 - val_loss: 0.0077 - val_mae: 0.0654
Epoch 2/5
55954/55954 [==============================] - 101s 2ms/step - loss: 0.0077 - mae: 0.0653 - val_loss: 0.0076 - val_mae: 0.0645
Epoch 3/5
55954/55954 [==============================] - 102s 2ms/step - loss: 0.0077 - mae: 0.0651 - val_loss: 0.0079 - val_mae: 0.0666
Epoch 4/5
55954/55954 [==============================] - 102s 2ms/step - loss: 0.0076 - mae: 0.0649 - val_loss: 0.0077 - val_mae: 0.0647
Epoch 5/5
55954/55954 [==============================] - 101s 2ms/step - loss: 0.0076 - mae: 0.0648 - val_loss: 0.0076 - val_mae: 0.0644
```



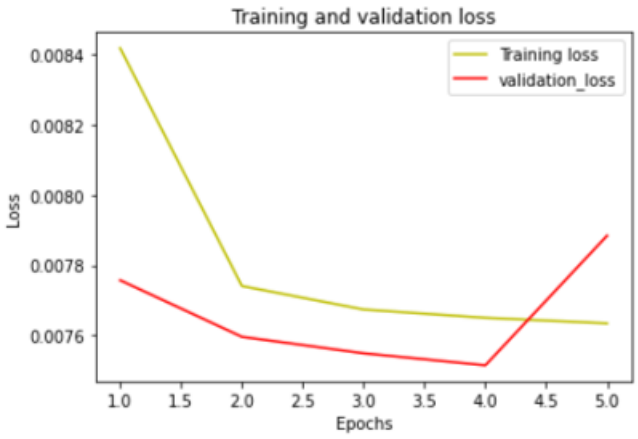
Model 4:-

```
model = Sequential()
model.add(Dense(32,input_dim=13,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(1,activation='linear')) #output layer
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mae'])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 32)                448
_________________________________________________________________
dense_11 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 33
=================================================================
Total params: 1,537
Trainable params: 1,537
Non-trainable params: 0
```

```
history = model.fit(X_train_scaled,y_train,validation_split=0.2,epochs=5)
```

```
Epoch 1/5
55954/55954 [==============================] - 76s 1ms/step - loss: 0.0084 - mae: 0.0679 - val_loss: 0.0078 - val_mae: 0.0659
Epoch 2/5
55954/55954 [==============================] - 75s 1ms/step - loss: 0.0077 - mae: 0.0655 - val_loss: 0.0076 - val_mae: 0.0647
Epoch 3/5
55954/55954 [==============================] - 75s 1ms/step - loss: 0.0077 - mae: 0.0651 - val_loss: 0.0075 - val_mae: 0.0643
Epoch 4/5
55954/55954 [==============================] - 74s 1ms/step - loss: 0.0076 - mae: 0.0650 - val_loss: 0.0075 - val_mae: 0.0644
Epoch 5/5
55954/55954 [==============================] - 73s 1ms/step - loss: 0.0076 - mae: 0.0650 - val_loss: 0.0079 - val_mae: 0.0662
```



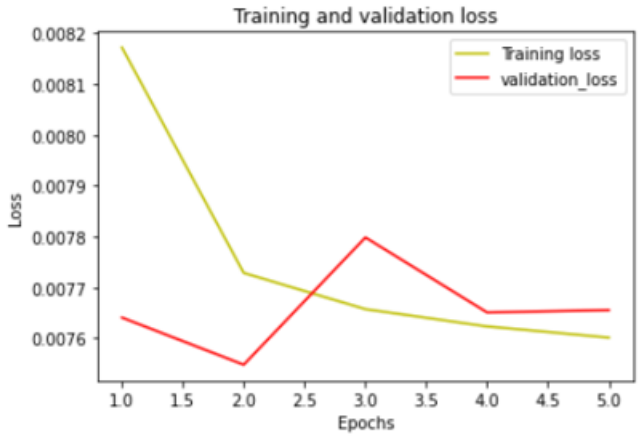
Model 5:-

```
model = Sequential()
model.add(Dense(64,input_dim=13,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(1,activation='linear')) #output layer
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mae'])
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 64)                896
_________________________________________________________________
dense_14 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_15 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_16 (Dense)             (None, 16)                528
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 17
=================================================================
Total params: 4,577
Trainable params: 4,577
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(X_train_scaled,y_train,validation_split=0.2,epochs=5)
```

```
Epoch 1/5
55954/55954 [==============================] - 112s 2ms/step - loss: 0.0082 - mae: 0.0672 - val_loss: 0.0076 - val_mae: 0.0647
Epoch 2/5
55954/55954 [==============================] - 112s 2ms/step - loss: 0.0077 - mae: 0.0654 - val_loss: 0.0075 - val_mae: 0.0644
Epoch 3/5
55954/55954 [==============================] - 117s 2ms/step - loss: 0.0077 - mae: 0.0651 - val_loss: 0.0078 - val_mae: 0.0668
Epoch 4/5
55954/55954 [==============================] - 112s 2ms/step - loss: 0.0076 - mae: 0.0649 - val_loss: 0.0077 - val_mae: 0.0656
Epoch 5/5
55954/55954 [==============================] - 111s 2ms/step - loss: 0.0076 - mae: 0.0648 - val_loss: 0.0077 - val_mae: 0.0651
```



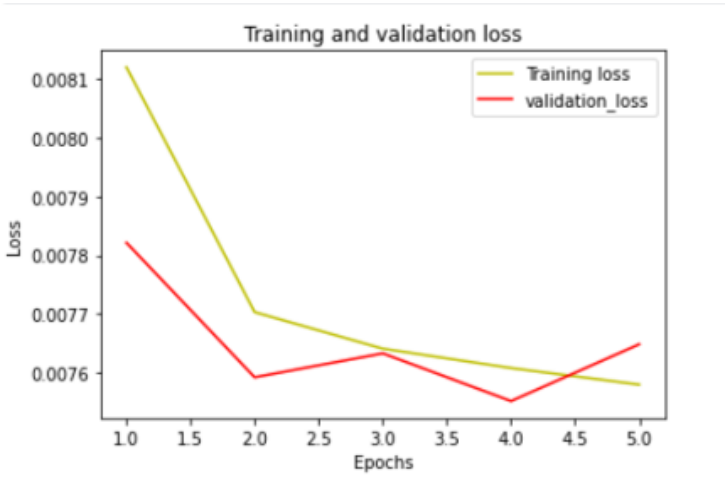
Model 6:-

```
model = Sequential()
model.add(Dense(128,input_dim=13,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(32,activation='relu'))
model.add(Dense(16,activation='relu'))
model.add(Dense(1,activation='linear')) #output layer
model.compile(loss='mean_squared_error',optimizer='adam',metrics=['mae'])
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 128)               1792
_________________________________________________________________
dense_19 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_20 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_21 (Dense)             (None, 16)                528
_________________________________________________________________
dense_22 (Dense)             (None, 1)                 17
=================================================================
Total params: 12,673
Trainable params: 12,673
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(X_train_scaled,y_train,validation_split=0.2,epochs=5)
```

```
Epoch 1/5
55954/55954 [==============================] - 114s 2ms/step - loss: 0.0081 - mae: 0.0670 - val_loss: 0.0078 - val_mae: 0.0657
Epoch 2/5
55954/55954 [==============================] - 114s 2ms/step - loss: 0.0077 - mae: 0.0653 - val_loss: 0.0076 - val_mae: 0.0653
Epoch 3/5
55954/55954 [==============================] - 116s 2ms/step - loss: 0.0076 - mae: 0.0650 - val_loss: 0.0076 - val_mae: 0.0649
Epoch 4/5
55954/55954 [==============================] - 116s 2ms/step - loss: 0.0076 - mae: 0.0648 - val_loss: 0.0076 - val_mae: 0.0641
Epoch 5/5
55954/55954 [==============================] - 115s 2ms/step - loss: 0.0076 - mae: 0.0647 - val_loss: 0.0076 - val_mae: 0.0654
```



Best Model Outcomes:

Model	Score	Train_MSE	Test_MSE
Linear Regression	0.8492	0.01341	0.01794
XGBoost	0.9360	0.00624	0.00621
Random Forest	-	0.00774	0.00889
Neural Networks	-	0.007318	0.00885

Conclusion: The project aims to evaluate the ranking for the PUBG players based on machine learning & deep learning algorithms along with doing EDA for analysis of the dataset in a much better way. After doing extensive feature engineering and detailed outlier removing the algorithms used in this research are Linear Regression, Random Forest ,XGBoost & Deep Neural Network, MSE (Mean Squared Error) was calculated to check which particular algorithm was fitted best for the huge dataset. It was observed that deep neural network performed well with the MSE value of 0.007318 for the training dataset , 0.0077 for the validation dataset and 0.00885 for the testing dataset respectively. The highest MSE value was obtained for Linear Regression which was 0.01341 with a regression score of 0.8492. for the training dataset and for the testing data the value of MSE was 0.01794. For Random Forest with n_estimators = 160, min_samples_leaf = 4, max_features = Sqrt MSE on train data was 0.00774 and on test data it was 0.00889 . Best result was obtained by XGBoost with min_child_weight = 5 , max_depth = 140 , learning_rate = 0.05 , Gamma = 0.1 and colsample_bytree = 0.7 . With the above hyperparameter values we obtained regression score of 0.9360 , MSE on training data was 0.00624 and on testing data it was 0.00621.