SUMMER TRAINING REPORT

**On**

**PUBG Dataset Prediction**

**(Machine Learning)**

**Submitted by**

**Vineet Rathor**

**(171500382)**

Department of Computer Engineering & Applications

**Institute of Engineering & Technology**



**GLA University**

**Mathura- 281406, INDIA**

**2019**

**CERTIFICATE**

****

** Summer Training Synopsis**

**B.tech. (CSE)-Batch 2017-2021**

**Student Information:**

|  |  |
| --- | --- |
| Name: Vineet Rathor | University Roll. No. 171500382 |
| Mobile: 8171015692 | Email: Vineet.rathor\_cs17@gla.ac.in |

**Information about Industry/Organization:**

|  |  |
| --- | --- |
| Industry/Organization Name with full Address | Inversion Consultancy Llp - RMZ GALLERIA, YELAHANKA, BANGALIRU,KARNATAKA, KARNATAKA,560064 |
| Contact Person | Name : Shubham Sharda  Mobile/email: 9632047066 |

**Project Information:**

|  |  |
| --- | --- |
| Title Of Project/Training/Task | PUBG Dataset Prediction |
| Role & Responsibility |  |
| Technical Details | Software Requirements: Anaconda Software |
| Training Implementation Details | Partial Implemented |
| Training Period | Start Date: 16/05/2019  End Date: 30/06/2019  Duration Of Training (In Weeks): 06 |

**Summary of the Training Work:**

The project titled”PUBG Dataset Prediction” is use to predict the Win Place Percentage (final Placement) from the final in-game stats and initial player ratings of different players using Regression algorithms. First the machine is trained on the training dataset and then the algorithm is tested over the test dataset and the final output is in the form of new csv file which contains IDs with their predicted Final Placement percentage.

**ACKNOWLEDGEMENT**

I thank the almighty for giving me the courage and perseverance in completing the project.

This project itself is acknowledgements for all those people who have given us their heartfelt co-operation in making this project a grand success.

I extend my sincere thanks to ***Mr. Shubham Sharda***,   
Trainer at “Inversion Consultancy Llp” for providing valuable guidance at every stage of this project work. I am profoundly grateful towards the unmatched services rendered by him. I could not have done this work without his help. The work culture in Inversion Consultancy Llp is really motivating. Everybody is such a friendly and cheerful companion here that work stress is never comes in way. He not only advised me in the project, but listened my arguments in our discussion.

Last but not least, we would like to express our deep sense of gratitude and earnest thanks giving to our dear parents for their moral support and heartfelt cooperation in doing the main project.

**Department of computer Engineering and Applications**

**GLA University, Mathura**

**17 km. Stone NH#2, Mathura-Delhi Road, P.O. – Chaumuha,**

**Mathura – 281406**



**Declaration**

I hereby declare that the work which is being presented in the Summer Training “**PUBG Dataset Prediction”,** in partial fulfillment of the requirements for Summer Training viva voce, is an authentic record of my own work carried under the supervision of “Inversion Consultancy”

Signature of Candidate:

Name of Candidate: Vineet Rathor

Roll. No.: 171500382

Course: B.tech (Computer Science and Engineering)

Year: 3rd

Semester: 5th

**PUBG DATASET PREDEICTION**

**ABSTRACT**

The report presents the three tasks completed during summer internship at Persistent System Limited

Which are listed below:

1. Understand of the Problem objective & business implication

2. Understanding the dataset & build the model

3. Evaluation of the model

All these tasks have been completed successfully and results were according to expectations. All the tasks needed very systematic approach, starting from the behavior of the data to the application of the algorithm and till evaluation of the model. The most challenging task was the domain knowledge, to understand the behavior of the data. Once the dataset has been analyzed, we applied different supervised learning algorithm for model building. It is one of the major areas and really need very fundamental and conceptual knowledge of Machine Learning.

Vineet Rathor

**CONTENTS**

**Certificate……………………………………...................**

**Synopsis…………………………………………………...**

**Acknowledgement………………………………................**

**Declaration………………………………………………..**

**Abstract………………………………………………......**

1. **Introduction:**

Introduction to PUBG Prediction……………...………….......08

Pre requisites………………………………………….……....08

Understanding the Problem……....………………….……......09

Objectives………………………………………………...…………..09

1. **Software Requirement**

Introduction toAnaconda………………………….…………10

Algorithms Used and their functionality………….………….10

**3. Testing Data**……………………………...…………..15

**4. Training Data**………………………………………...16

Data Analysis…………………………………………….......16

Features of Dataset……………………………………….….16

Data Insights………………………………………...…….…17

**5. Conclusion**…………………………………………...19

**6. Bibliography**………………………………………....20

**7.** **Appendices**………………………………………......21

**Chapter 1**

**Introduction**

**Introduction to PUBG Prediction**

Player Unknown Battle Ground’s (PUBG) is an online Multiplayer battle royale game developed by PUBG Corporation. It is a player versus player action game in which no more than 100 player fight in a battle royale. Players can choose to enter the match solo, duo or with a small team up to 4 people. We will address this discrepancy during feature engineering. The last person or the last team alive wins the match.

The data is part of a Kaggle competition which has scraped 65000 games worth of data using an API. The goal is to predict the win percentage of the player based on 28 given features. For example, in a solo game of 100 players if a player got rank of 80, then its winPlacePerc will be (100-80)/(100-1)=0.204 using the formula winPlacePerc = (maxPlace-winPlace)/(maxPlace-1). The problem is then essentially a regression problem to predict the winPlacePerc of the player between [0,1].

These types of problems are solved by doing extensive exploratory data analysis and feature engineering before applying a regression model. We were provided with the Training Dataset so we divided it into Training and Validation Dataset in 70:30 ratio respectively. We first used the LGBM (Light Gradient Boosting Machine) (5) Regressor to train our training dataset after doing exploratory analysis on the data. We compared other models with the same dataset. We reported the accuracy of each model that we got on the validation data.

**Pre-requisite**

Hands-on knowledge of Numpy, Scikit learn, Matplottlib, Seaborn and Pandas is essential before working on the Project. Make sure that you have the following packages installed and running before using different algorithms.

**Understanding the Problem**

In a PUBG game, up to 100 players start in each match (matchId). Players can be on teams (groupId) which get ranked at the end of the game (winPlacePerc) based on how many other teams are still alive when they are eliminated. In game, players can pick up different munitions, revive downed-but-not-out (knocked) teammates, drive vehicles, swim, run, shoot, and experience all of the consequences -- such as falling too far or running themselves over and eliminating themselves. 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 players per group.

**Objective**

To predict the final ranking and placement from the game statistics and the initial player ratings.

The overview of the steps involved in the project is:

1. Data Cleaning and Exploratory Data Analysis

2. visualizing the data and feature engineering

3. Training Models on Training data and then drawing comparison between them using validation data

4. Choosing the best model and fitting the Test data on it

5. Reporting the Accuracy of the final chosen model.

**Chapter 2**

**Software Requirement**

**Introduction to Anaconda**

Anaconda is a free and open-source[5] distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.)

The Anaconda distribution is used by over 15 million users and includes more than 1500 popular data-science packages suitable for Windows, Linux, and MacOS.

The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda.

We will be using Jupyter which comes inbuilt in anaconda for our project.

**Jupyter Notebook:**

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

**Algorithms Used and their functionality**

**Light Gradient Boosting Machine(LGBM):**

LGBM is a type of gradient boosting algorithm which uses decision trees as its framework. The most inherent part of this model is that it can handle a very large size of data and takes very low memory to run. Due to its efficiency, accuracy and interpretability it has started to be widely used for very large data sets. Similarly it is not advised to use this model with small data set as it is prone to overfitting.

LGBM is an ensemble model of decision tree which learns by fitting negative gradients which are also known as residual errors. Suppose we have n identical and independent distributed (x1, x2,...xn) with dimension s in a Gradient Space. In every iteration of gradient boosting, the residuals of loss function with respect to output of the model are denoted by (g1,g2,...gn). So the model splits each node at the point where the Information Gain is the maximum.

**Parameters:**

1.) “objective”(application): This is the most important parameter and specifies the application of your model, whether it is a regression problem or classification problem. LightGBM will by default consider model as a regression model.

regression: for regression

binary: for binary classification

multiclass: for multiclass classification problem

2.) “metric”: again one of the important parameter as it specifies loss for model building. Below are few general losses for regression and classification.

mae: mean absolute error

mse: mean squared error

binary\_logloss: loss for binary classification

multi\_logloss: loss for multi classification

3.) “n\_estimators” (max\_depth): It describes the maximum depth of tree. This parameter is used to handle model overfitting. Any time you feel that your model is overfitted, my first advice will be to lower max\_depth.

4.) “early\_stopping\_round”: This parameter can help you speed up your analysis. Model will stop training if one metric of one validation data doesn’t improve in last early\_stopping\_round rounds. This will reduce excessive iterations.

5.) “num\_leaves”: number of leaves in full tree, default: 31

6.) “learning\_rate”: This determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates. Typical values: 0.1, 0.001, 0.003…

7.) “bagging\_fraction”: specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.

**Multiple Linear Regression**

In Multiple Linear Regression we tend to learn about multiple uncorrelated independent variables with the dependent variable or the criterion. In our project there were 28 dependent variables and 1 independent variable winPlacePerc (which give the winning percentile of particular player). The equation is given as:



**Ridge Regression**

Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors. It is hoped that the net effect will be to give estimates that are more reliable. Cost function of this regression is –

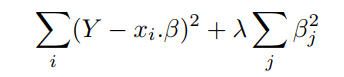


Fig 2.1

**Lasso Regression**

In statistics and machine learning, lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.

**Elastic Net Regression**

Elastic Net is a combination of both Ridge and Lasso Regression. Both penalty terms of Ridge and Lasso are added to the cost function and is shown as following:

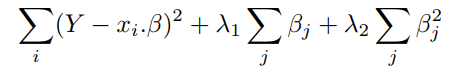


Fig 2.2

**Gradient Boosting Model**

It is a supervised technique where the main objective is to minimize the Loss function i.e., the Mean Squared Error(MSE). It uses Gradient Descent to minimize the loss and uses a learning rate alpha to find the updated value of predicted values. LGBM is a type of gradient boosting algorithm which uses decision trees as its framework. The most inherent part of this model is that it can handle a very large size of data and takes very low memory to run. Due to its efficiency, accuracy and interpretability it has started to be widely used for very large data sets. Similarly it is not advised to use this model with small data set as it is prone to overfitting.

**Importing required libraries**

Let's import the necessary libraries first. Remember, the names of these libraries are self-descriptive so you can put 2 and 2 together.

1.) import pandas as pd -- It's an open source data analysis library for providing easy-to-use data structures and data analysis tools. It supports reading and writing excels spreadsheets, CVS's and a whole lot of manipulation.

2.) import numpy as np -- Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Moreover Numpy forms the foundation of the Machine Learning stack.

3.) import matplotlib.pyplot as plt -- Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

4.) import seaborn as sns -- statistical data visualization. Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

5.) from sklearn.model\_selection import train\_test\_split -- scikit-learn provides a helpful function for partitioning data, train\_test\_split, which splits out your data into a training set and a test set on the basis of train size and test size we provide and it can also shuffle the data by using random\_state so that similar data does not go altogether.

6.) from sklearn.metrics import mean\_absolute\_error, r2\_score – these are used for evaluating the models and will be dicussed in testing data.

7.) from lightgbm import LGBMRegressor -- this will import the LGBM and then only we can use this algorithm.

**Chapter 3**

**Testing Data Set**

We have used **Mean Absolute Error** as our base performance metric. Since the predicting class is a continuous variable. MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It’s the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight. It is negatively oriented which means the lower the value, the better is the result. The Mean absolute error is calculated by the formula 1.

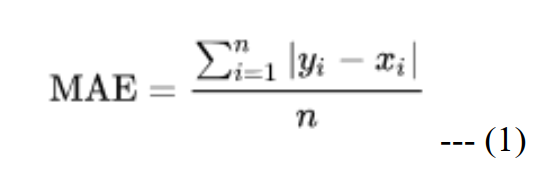
****

Fig 3.1

**R-squared** is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. The definition of R-squared is fairly straight-forward; it is the percentage of the response variable variation that is explained by a linear model. Or:

R-squared = Explained variation / Total variation

R-squared is always between 0 and 100%:

0% indicates that the model explains none of the variability of the response data around its mean. 100% indicates that the model explains all the variability of the response data around its mean. In general, the higher the R-squared, the better the model fits your data.

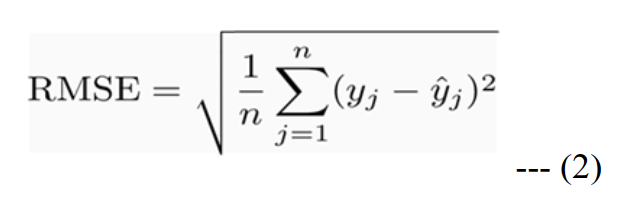


Fig 3.2

**Chapter 4**

**Training Data Set**

**Data Analysis**

Kaggle has 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 and squads. Data set is divided into two parts: training set and testing set. Training set contains 4.5 million instances with 29 attributes, whereas testing set contains 1.9 million instances with 28 attributes (without label).

**Features of Dataset**

1.) DBNOs - Number of enemy players knocked.

2.) Assists - Number of enemy players this player damaged that were killed by teammates.

3.) Boosts - Number of boost items used.

4.) DamageDealt - Total damage dealt. Note: Self inflicted damage is subtracted

5.) HeadshotKills - Number of enemy players killed with headshots.

6.) Heals - Number of healing items used.

7.) Id - Player’s Id

8.) KillPlace - Ranking in match of number of enemy players killed.

9.) KillPoints - Kills-based external ranking of player.

10.) KillStreaks - Max number of enemy players killed in a short amount of time.

11.) Kills - Number of enemy players killed.

12.) 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.

13.) MatchDuration - Duration of match in seconds.

14.) MatchId - ID to identify match. There are no matches that are in both the training and testing set.

15.) 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.

16.) RankPoints - ranking of player.

17.) Revives - Number of times this player revived teammates.

18.) RideDistance - Total distance traveled in vehicles measured in meters.

19.) RoadKills - Number of kills while in a vehicle.

20.) SwimDistance - Total distance traveled by swimming measured in meters.

21.) TeamKills - Number of times this player killed a teammate.

22.) VehicleDestroys - Number of vehicles destroyed.

23.) WalkDistance - Total distance traveled on foot measured in meters.

24.) WeaponsAcquired - Number of weapons picked up.

25.) WinPoints - Win-based external ranking of player.

26.) 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.

27.) NumGroups - Number of groups we have data for in the match.

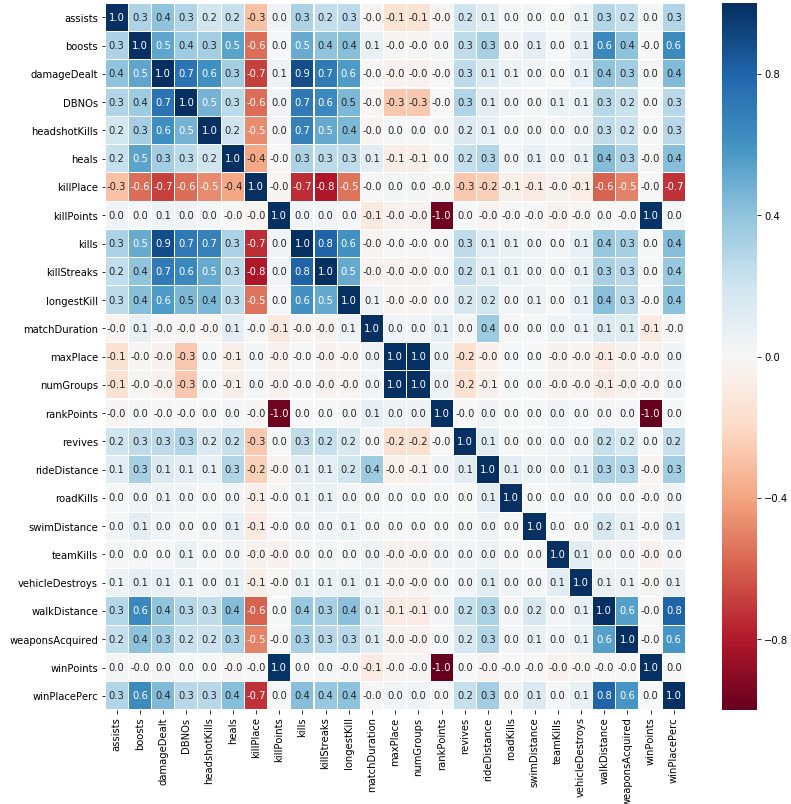
28.) MaxPlace - Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.

29.) 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.

**Data Insights**

In order to find out patterns in the data and determine what features may be useful, we decided to perform Exploratory Data Analysis on the data. This would also help us create features which better depict the skill level of the players. We start by visualizing different features given to us and then we removed the outliers from our dataset, we also looked for the missing values in the dataset and removed that value also as there was only one row which had missing value, we also converted the column ‘matchType’ by creating dummies so as to encode the different values of that feature and thus we will be able to use it. In the end we created the correlation matrix to see how different features are correlated with each other. We got the following correlation –

Fig 4.1



After using the Light GBM model to train on the dataset and to predict on the test dataset , we used it to find the feature importance as to which feature were most important and affected the final output and which were less important ones.

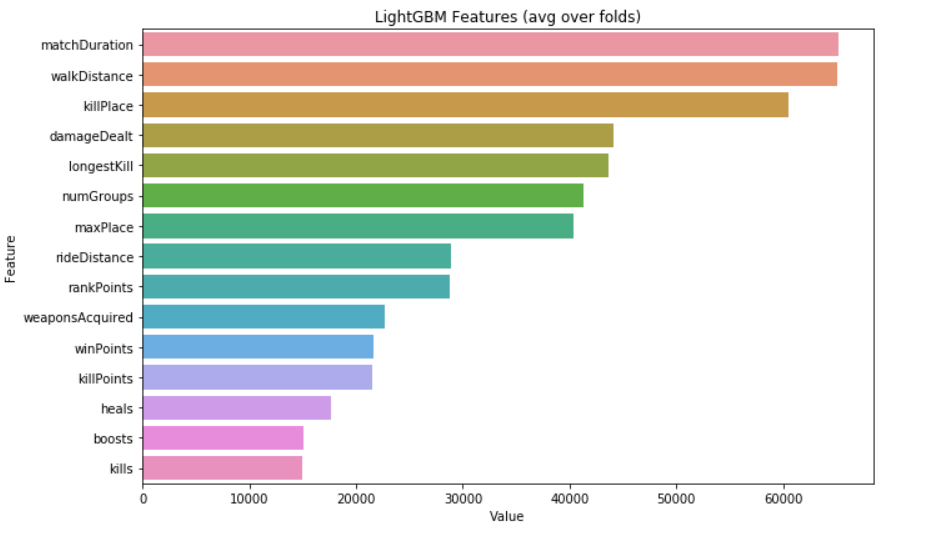


Fig 4.2

**Chapter 5**

**Conclusion**

We got the following Mean Absolute Error(MAE) and R-squared error for the algorithms we used –

linear

Mean Absolute Error is 0.08977 R2 score is 84.04%

ridge

Mean Absolute Error is 0.08977 R2 score is 84.04%

lasso

Mean Absolute Error is 0.12083 R2 score is 74.53%

elastic

Mean Absolute Error is 0.11297 R2 score is 77.17%

Adaboost

Mean Absolute Error is 0.09976 R2 score is 81.58%

GBR

Mean Absolute Error is 0.06176 R2 score is 92.21%

forest

Mean Absolute Error is 0.06063 R2 score is 92.26%

tree

Mean Absolute Error is 0.08118 R2 score is 85.64%

LGBM

Mean Absolute Error is 0.05455 R2 score is 93.75%

For this dataset, we have observed that Boosting models like LGBM, AdaBoost and Random Forest perform better than traditional Multiple Regression models. This is perhaps because boosting algorithms are very robust to noise and outliers. They optimize the model via gradient descent using generic differentiable loss functions and also has various benefits like automatic null-handling, in-built feature selection and scale invariance. Since our data is not necessarily linearly separable, multiple linear regression models performed worse. There are multiple features with non-linear relationships between them, which boosting algorithms can recognize better using the ensemble of multiple weak learners and tuning the parameters to control model complexity (avoid overfitting).

**Chapter 6**

**Bibliography**

www.google.com

www.kaggle.com/c/pubg-finish-placement-prediction

www.numpy.org

www.matplotlib.org

lightgbm.readthedocs.io/en/latest/

medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

**Chapter 7**

**Appendices**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

import seaborn as sns

import random

random.seed(42)

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_absolute\_error, r2\_score

from lightgbm import LGBMRegressor

import gc

from sklearn.linear\_model import LinearRegression, Ridge, Lasso, ElasticNet

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor

def reduce\_mem\_usage(df):

""" iterate through all the columns of a dataframe and modify the data type

to reduce memory usage.

# Memory saving function credit to https://www.kaggle.com/gemartin/load-data-reduce-memory-usage

"""

import numpy as np

for col in df.columns:

col\_type = df[col].dtype

if col\_type != object:

c\_min = df[col].min()

c\_max = df[col].max()

if str(col\_type)[:3] == 'int':

if c\_min > np.iinfo(np.int8).min and c\_max < np.iinfo(np.int8).max:

df[col] = df[col].astype(np.int8)

elif c\_min > np.iinfo(np.int16).min and c\_max < np.iinfo(np.int16).max:

df[col] = df[col].astype(np.int16)

elif c\_min > np.iinfo(np.int32).min and c\_max < np.iinfo(np.int32).max:

df[col] = df[col].astype(np.int32)

elif c\_min > np.iinfo(np.int64).min and c\_max < np.iinfo(np.int64).max:

df[col] = df[col].astype(np.int64)

else:

if c\_min > np.finfo(np.float16).min and c\_max < np.finfo(np.float16).max:

df[col] = df[col].astype(np.float16)

elif c\_min > np.finfo(np.float32).min and c\_max < np.finfo(np.float32).max:

df[col] = df[col].astype(np.float32)

else:

df[col] = df[col].astype(np.float64)

return df

df\_train = pd.read\_csv('train\_V2.csv')

df\_train.info()

df\_train = reduce\_mem\_usage(df\_train)

df\_train.info()

pd.set\_option('display.max\_columns', 500)

df\_train.head()

def plot\_hist(x, title, noOfBins=40, col='#AA3939'):

fig, ax = plt.subplots(figsize=(8,8))

plt.hist(df\_train.winPlacePerc,bins=noOfBins,color=col)

plt.xlabel('WinPlacePercentile',fontsize = 15,color='black')

plt.ylabel('Frequency',fontsize = 15,color='black')

plt.grid(axis='y', alpha=0.5)

plt.title(title,fontsize = 20,color='black')

plot\_hist(df\_train.winPlacePerc, title='Histogram of winning percentiles')

df\_test = pd.read\_csv('test\_V2.csv')

df\_test = reduce\_mem\_usage(df\_test)

#Removing NAN values from the dataset

df\_train.isnull().sum()

df\_train.dropna(inplace=True)

df\_train.isnull().sum()

print(len(df\_train))

print(len(df\_test))

# visualizations and Removing Outliers

def visualization (col, num\_bin=10):

title = col[0].upper() + col[1:]

f,axes=plt.subplots()

plt.xlabel(title)

plt.ylabel('Log Count')

axes.set\_yscale('log')

df\_train.hist(column=col,ax=axes,bins=num\_bin)

plt.title('Histogram of ' + title)

plt.show()

tmp = df\_train[col].value\_counts().sort\_values(ascending=False)

print('Min value of ' + title + ' is: ',min(tmp.index))

print('Max value of ' + title + ' is: ',max(tmp.index))

visualization('assists')

visualization('roadKills')

# # since, most of the players have kills from 0 to 10

# so to remove the outliers from my data, we drop all the players who have more than 10 roadkills.

# drop all the road kills above 10.

#test

df\_train.drop(df\_train[df\_train['roadKills']>=10].index,inplace=True)

#test

df\_test.drop(df\_test[df\_test['roadKills']>=10].index,inplace=True)

visualization('kills')

# dropping the outliers.

#train

df\_train.drop(df\_train[df\_train['kills']>=35].index,inplace=True)

#test

df\_test.drop(df\_test[df\_test['kills']>=35].index,inplace=True)

visualization('killStreaks')

visualization('teamKills')

visualization('headshotKills', num\_bin=40)

visualization('vehicleDestroys',num\_bin=5)

visualization('revives',num\_bin=50)

visualization('damageDealt', num\_bin=1000)

visualization('weaponsAcquired',num\_bin=30)

# removing the outliers.

#train

df\_train.drop(df\_train[df\_train.weaponsAcquired>=50].index,inplace=True)

#test

df\_test.drop(df\_test[df\_test.weaponsAcquired>=50].index,inplace=True)

visualization('boosts',num\_bin=30)

visualization('heals', num\_bin=100)

# removing the outliers.

#train

df\_train.drop(df\_train[df\_train.heals>=40].index,inplace=True)

#test

df\_test.drop(df\_test[df\_test.heals>=40].index,inplace=True)

visualization('walkDistance',num\_bin=250)

#Removing the outliers

#train

df\_train.drop(df\_train[df\_train['walkDistance']>=10000].index,inplace=True)

#test

df\_test.drop(df\_test[df\_test['walkDistance']>=10000].index,inplace=True)

visualization('rideDistance',num\_bin=500)

#Removing the outliers.

#test

df\_train.drop(df\_train[df\_train.rideDistance >=15000].index, inplace=True)

#test

df\_test.drop(df\_test[df\_test.rideDistance >=15000].index, inplace=True)

visualization('longestKill', num\_bin=100)

Most kills are made from a distance of 100 meters or closer. There are however some outliers who make a kill from more than 1km away. This is probably done by cheaters or game crackers.

# drop outliers.

#train

df\_train.drop(df\_train[df\_train['longestKill']>=1000].index,inplace=True)

#test

df\_test.drop(df\_test[df\_test['longestKill']>=1000].index,inplace=True)

df\_train.shape

So the initial shape is (4446965, 29)And After removing the outliers the new shape is (4445866, -)

Something around 1100 rows have been removed until now. Which is nothing compared to the number of rows we have.

#So the initial shape is (4446965, 29)And After removing the outliers the new shape is (4445866, -)

#Something around 1100 rows have been removed until now. Which is nothing compared to the number of rows we have.

# Creating a dummy variable for categorical variable present in our data set.

#matchType

#train

df\_train=pd.get\_dummies(df\_train,columns=['matchType'])

#test

df\_test=pd.get\_dummies(df\_test,columns=['matchType'])

#Correlation Analysis

cols\_to\_drop = ['Id','matchId','groupId','matchType']

cols\_to\_fit = [col for col in df\_train.columns if col not in cols\_to\_drop]

corr = df\_train[cols\_to\_fit].corr()

plt.figure(figsize=(9,7))

sns.heatmap(corr,xticklabels=corr.columns.values,yticklabels=corr.columns.values,linecolor='white',linewidths=0.1,cmap='RdBu')

plt.show()

t = df\_train

t =t.drop(['Id','groupId','matchId',],axis=1)

y = t['winPlacePerc']

X = t.drop(['winPlacePerc'],axis=1)

X\_test = df\_test.drop(['Id','groupId','matchId'],axis=1)

#splitting the data into training and testing by using train\_test\_split

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, train\_size=0.7)

len(X\_train)

del train,test,X,y

#X\_train = X

#y\_train = y

gc.collect()

# Model LightGBM

import lightgbm as lgbm

def calculate\_error(cl,name):

print(name)

y\_pre = cl.predict(X\_val)

print('Mean Absolute Error is {:.5f}'.format(mean\_absolute\_error(y\_val,y\_pre)))

print('R2 score is {:.2%}'.format(r2\_score(y\_val, cl.predict(X\_val))))

# Create parameters to search

params = {"objective" : "regression", "metric" : "mae", 'n\_estimators':20000,

'early\_stopping\_rounds':200, "num\_leaves" : 31, "learning\_rate" : 0.05,

"bagging\_fraction" : 0.7, "bagging\_seed" : 0, "num\_threads" : 4,

"colsample\_bytree" : 0.7

}

lgbTrain = lgbm.Dataset(X\_train, label=y\_train)

lgbVal = lgbm.Dataset(X\_val, label=y\_val)

model = lgbm.train(params,lgbTrain,valid\_sets=[lgbTrain, lgbVal],

early\_stopping\_rounds=200, verbose\_eval=1000)

calculate\_error(model,"LGBM")

y\_predict = model.predict(X\_test)

y\_predict[y\_predict > 1] = 1

y\_predict[y\_predict < 0] = 0

df\_test['winPlacePerc'] = y\_predict

submission = df\_test[['Id', 'winPlacePerc']]

submission.to\_csv('Finalsubmission.csv', index=False)

#find which feature importance

cols\_to\_drop = ['Id', 'groupId', 'matchId', 'winPlacePerc']

cols\_to\_fit = [col for col in X\_train.columns if col not in cols\_to\_drop]

feature\_importance = pd.DataFrame(sorted(zip(model.feature\_importance(), cols\_to\_fit)), columns=['Value','Feature'])

feature\_importance = feature\_importance.tail(15)

plt.figure(figsize=(10, 6))

sns.barplot(x="Value", y="Feature", data=feature\_importance.sort\_values(by="Value", ascending=False))

plt.title('LightGBM Features (avg over folds)')

plt.tight\_layout()

plt.savefig("lgbmfeatures.png",dpi=500)

# Regression Models

def runAllModels(X\_train, Y\_train):

linear = LinearRegression(copy\_X=True)

linear.fit(X\_train,Y\_train)

calculate\_error(linear,"linear")

ridge = Ridge(copy\_X=True)

ridge.fit(X\_train,Y\_train)

calculate\_error(ridge,"ridge")

lasso = Lasso(copy\_X=True)

lasso.fit(X\_train,Y\_train)

calculate\_error(lasso,"lasso")

elastic = ElasticNet(copy\_X=True)

elastic.fit(X\_train,Y\_train)

calculate\_error(elastic,"elastic")

ada = AdaBoostRegressor(learning\_rate=0.8)

ada.fit(X\_train,Y\_train)

calculate\_error(ada,"Adaboost")

GBR = GradientBoostingRegressor(learning\_rate=0.8)

GBR.fit(X\_train,Y\_train)

calculate\_error(GBR,"GBR")

forest = RandomForestRegressor(n\_estimators=10)

forest.fit(X\_train,Y\_train)

calculate\_error(forest,"forest")

tree = DecisionTreeRegressor()

tree.fit(X\_train,Y\_train)

calculate\_error(tree,"tree")

runAllModels(X\_train,y\_train)