PUBG FINISH PLACEMENT PREDICTION

CHANUKYA LAKAMSANI

Graduate Project DSA 5013

Index

Introduction

Pubg is a Mobile multiplayer game which recently received lot of attention due to the game play and graphics in the application. The game starts with 100 players with each player having a unique ID. The players can be in team and each team is ranked at end of the game. There are various tasks that can player can perform in the game like carry any munitions, drive a car or a boat, run, shoot, jump and all possible actions that we can perform in real life.

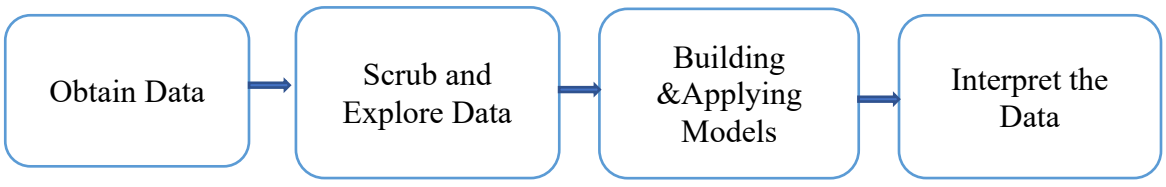
The data given comes from the matches: single, duo, squads and customs.
Objective: To predict the Winner placement if the 100 players play the game.

In order to predict the placement(rank) of the 100 players, we have used and compared two regression models:

- Linear Regression
- Gradient Boosting

I have Used Python language as medium of comparing the two models.

The project can be can majorly segregated into:



Data Exploration[1][2]:

The Dataset is obtained from the Kaggle where competition is held to predict the 100 places for a given set of players based upon the features given.
The data set contains 29 variables where WinPlaceperc is the dependent Variable and rest of them are Independent Variable. There are 10,48,574 observations.

Independent Variables:

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.

Dependent Variable:

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.

Summary of the Data:

Dataset info

Number of variables	40
Number of observations	1048575
Total Missing (%)	0.0%
Total size in memory	320.0 MiB
Average record size in memory	320.0 B

Variables types

Numeric	22
Categorical	0
Boolean	16
Date	0
Text (Unique)	0
Rejected	2
Unsupported	0

Warnings

- `numGroups` is highly correlated with `maxPlace` (ρ = 0.99789) **Rejected**
- `roadKills` is highly skewed (γ1 = 33.721) **Skewed**
- `winPoints` is highly correlated with `killPoints` (ρ = 0.98337) **Rejected**
- Dataset has 168 duplicate rows **Warning**

(screenshot from the code)

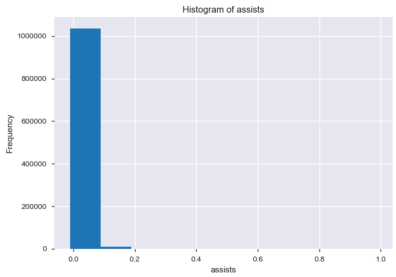
Operations performed on the Dataset:

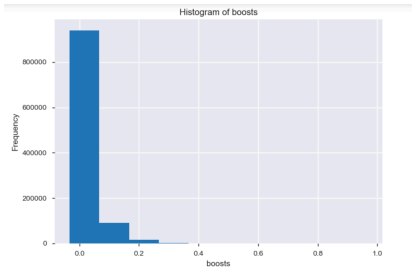
1) Standardization the Data: Each Block in the data set is applied the below formula

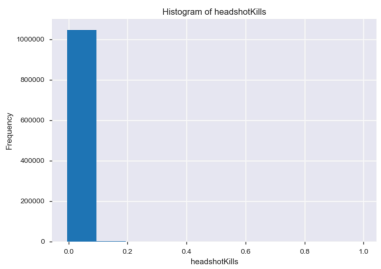
$$x_{new} = \frac{x - \mu}{\sigma}$$

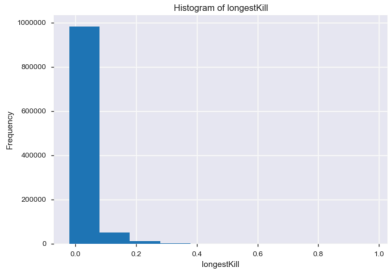
(source: http://www.dataminingblog.com/standardization-vs-normalization/)

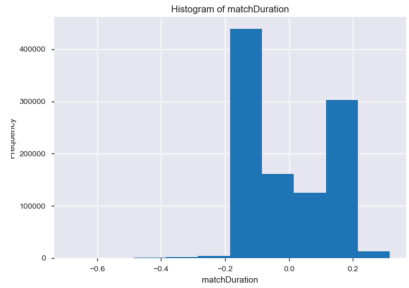
2) Distribution of the Dataset[3]:

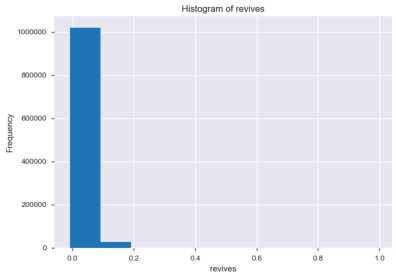


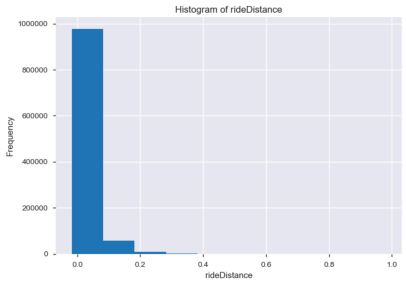


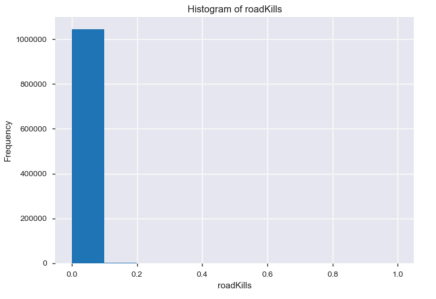


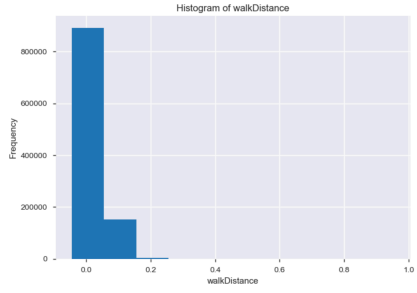






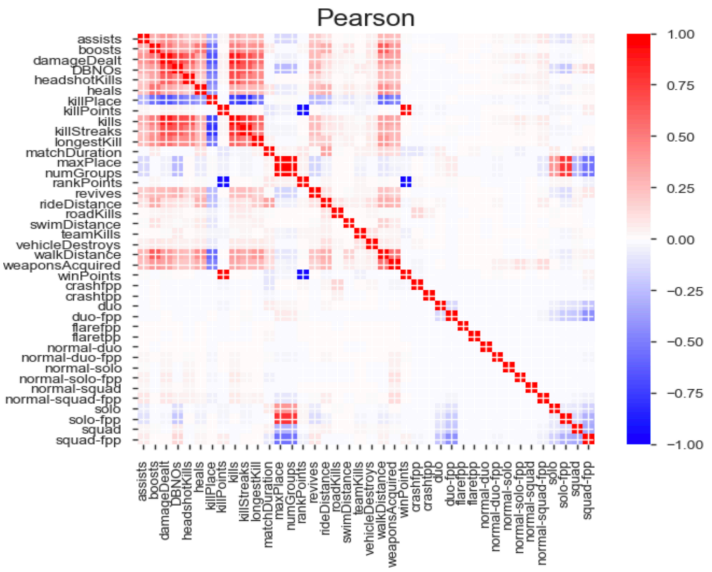




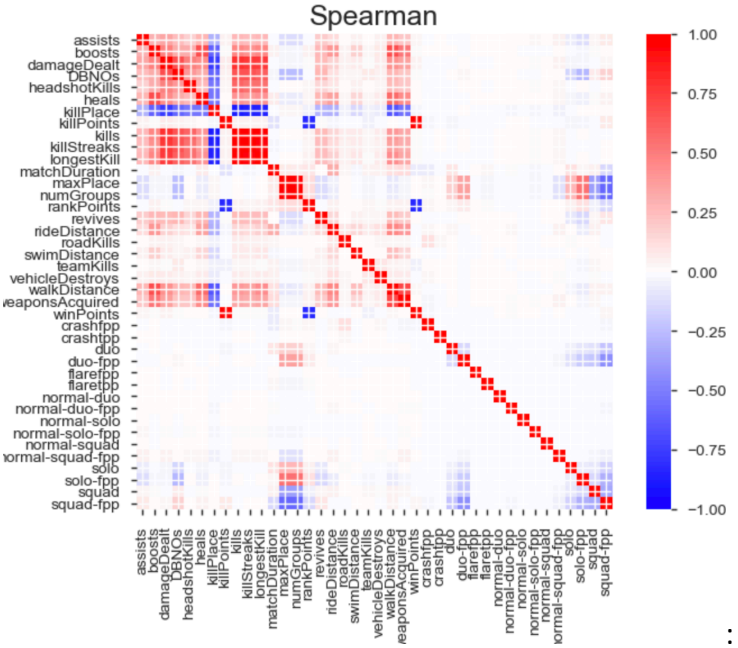


3) **Finding correlation**:

   a) Pearson Correlation[4].



b) Spearman Correlation[5].



:

Removing multicollinearity[6]:

What is multicollinearity?

In mutliregression when one variable can is linearly dependent on another variable or can predicted by another variable, then this phenomenon is said to be multicollinearity.

Disadvantage due to Multicollinearity: Due to multicollinearity there will be a drastic change in the coefficients. Change in coefficients lead to change in model. It leads to decrease in precision and Accuracy and F1 score.

How to deal with Multi collinearity?

Calculate Variance Inflation factor. Variance Inflation factor = 1/(1-r_square). Where r_square is the proportion of the variance in the dependent variable that is predictable from independent variable.

VIF value should be less than 5. If VIF is less than 5 then there will not be a drastic change in coefficients.

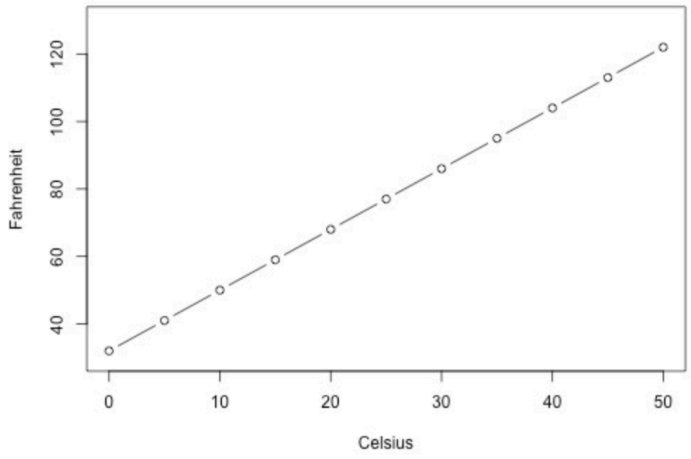
Regression Models:

a) **Linear Regression**:

Statistical Approach to summarize and study relationships between two continuous variables.

Eg: Fah =95Cels+32.
X(cel) = Independent Variable

Y(Fah) axis = Dependent Variable.



source: Pennstate website(regression methods)[7]

For our Problem

Dependent Variable: WinPlacePerc

Independent Variables: 'assists', 'boosts', 'headshotKills', 'heals', 'killPlace', 'longestKill', 'matchDuration', 'rankPoints', 'revives', 'rideDistance', 'roadKills', 'swimDistance', 'teamKills', 'vehicleDestroys', 'walkDistance', 'weaponsAcquired', 'crashfpp', 'crashtpp', 'duo', 'duo-fpp', 'flarefpp', 'flaretpp', 'normal-duo', 'normal-duo-fpp', 'normal-solo', 'normal-solo-fpp', 'normal-squad', 'normal-squad-fpp', 'solo', 'solo-fpp', 'squad', 'squad-fpp'.

RESULT: The most common method for fitting a regression line is the method of lease squares. The method calculates the best fitting line for the observed by minimizing the sum of the vertical deviations from each data point to the line.

Outlier: Anything is far away from the fitted line can be considered as an outlier.

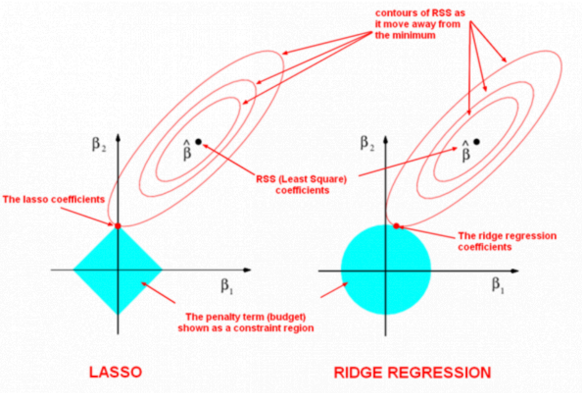
b) **GBM(Gradient Boosting Regression)**[8]:

It is also called MART (Multiple additive Regression Trees). They build on forward stage-wise additive model which means that the predictors are not made independently but sequentially.

This method employs the logic in which the subsequent predictors learn from the mistake. GBM involves three elements:

a) Loss function: Depends on the type of problem to be solved. In our case we used Regression square error.
b) Weak Learner: Decision trees are used as the week learner is gradient boosting.
c) Additive model: Trees are added one at a time.

c) **LASSO and Ridge Regression**: In order to overcome the over fitting, we tend to increase the complexity by adding the hyper parameters. When we try to decrease the prediction errors which becomes subtle, due to that bias turns to over fitting. Lasso and Ridge are popular ways to reduce this regularized statistical approach to overcome this problem. Lasso basically does both variable selection and regularization to improve the accuracy.



Source: Quora written by Balaji[9].

Metrics for Model Evaluation[10]:

Mean Absolute Error[11]: The difference between the two continuous variables. It measures the average magnitude of errors in set of forecasts, without considering the direction.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Source: Towards Data science[11]

RMSE(Root Mean Square Error): It method of calculating the average magnitude of the error.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Source: Towards Data science[11]

LASSO(least absolute shrinkage and selection operator) regression: Adjusted R^2 is used to measure the measure the accuracy.

$$R^2_{adjusted} = 1 - \frac{(1-R^2)(n-1)}{n-k-1}$$

Results & Conclusions:

Metrics

Model	MAE	RMSE
Linear Regression	0.10347155642278862	0.019194943232088358
GBM	0.07301925718909927	0.010533641489882443
LASSO	0.10010192571890992	0.093019257189099275

Conclusions: From the result we can see than GBM(Gradient Boosting Regression) performed the best among the two models. The RMSE for GBM is .0105 which is slightly greater than 1% of the mean value of the Winnerplace.

References:

[1] "PUBG Finish Placement Prediction (Kernels Only) | Kaggle." [Online]. Available: https://www.kaggle.com/c/pubg-finish-placement-prediction/data. [Accessed: 08-May-2019].

[2] S. Idreos, O. Papaemmanouil, and S. Chaudhuri, "Overview of Data Exploration Techniques."

[3] Y. Ioannidis, "The History of Histograms (abridged)."

[4] M. M. Mukaka, "Statistics corner: A guide to appropriate use of correlation coefficient in medical research.," *Malawi Med. J.*, vol. 24, no. 3, pp. 69–71, Sep. 2012.

[5] "Spearman's Correlation in Stata - Procedure, output and interpretation of the output using a relevant example." [Online]. Available: https://statistics.laerd.com/stata-tutorials/spearmans-correlation-using-stata.php. [Accessed: 07-May-2019].

[6] K. P. Vatcheva, M. Lee, J. B. McCormick, and M. H. Rahbar, "Multicollinearity in Regression Analyses Conducted in Epidemiologic Studies.," *Epidemiol. (Sunnyvale, Calif.)*, vol. 6, no. 2, Apr. 2016.

[7] "1.7 - Some Examples | STAT 501." [Online]. Available: https://newonlinecourses.science.psu.edu/stat501/node/257/. [Accessed: 08-May-2019].

[8] "Boosting algorithm: GBM – Towards Data Science." [Online]. Available: https://towardsdatascience.com/boosting-algorithm-gbm-97737c63daa3. [Accessed: 07-May-2019].

[9] "(3) How would you describe the difference between linear regression, lasso regression, and ridge regression? - Quora." [Online]. Available: https://www.quora.com/How-would-you-describe-the-difference-between-linear-regression-lasso-regression-and-ridge-regression. [Accessed: 08-May-2019].

[10] Georgios Drakos, "How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics." [Online]. Available: https://towardsdatascience.com/how-to-select-the-right-evaluation-metric-for-machine-

learning-models-part-1-regrression-metrics-3606e25beae0. [Accessed: 07-May-2019].

[11] T. Chai and R. R. Draxler, “Root mean square error (RMSE) or mean absolute error (MAE)?-Arguments against avoiding RMSE in the literature,” *Geosci. Model Dev*, vol. 7, pp. 1247–1250, 2014.

Appendix:

In [54]:
```
import tensorflow as tf
import keras
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Flatten
from keras.layers import Dense, Input, Dropout, LSTM, Activation
import numpy as np
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
from patsy import dmatrices
from sklearn.linear_model import LinearRegression
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

In [2]:
```
import tensorflow as tf
import keras
from sklearn import ensemble
```

Part 1

In [3]:
```
import pandas as pd
```

In [4]:
```
datset = pd.read_csv("/Users/chanukya/Desktop/statistics/Pubg-Finish/train_V2.csv")
```

In [5]:
```
df = datset
df.head()
df.shape
```

Out[5]: (1048575, 29)

In [8]:
```
ID = df["Id"]
groupID = df["groupId"]
matchID = df["matchId"]
```

In [9]:
```
del df["Id"]
del df["groupId"]
del df["matchId"]
```

In [10]:
```
df.head()
```

Out[10]:

ts	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	killPoints	kills	killStreaks	...	revives	rideDistance	roadKills	swimDistance	teamKills	vehic
0	0	0.00	0	0	0	60	1241	0	0	...	0	0.0000	0	0.00	0	
0	0	91.47	0	0	0	57	0	0	0	...	0	0.0045	0	11.04	0	
1	0	68.00	0	0	0	47	0	0	0	...	0	0.0000	0	0.00	0	
0	0	32.90	0	0	0	75	0	0	0	...	0	0.0000	0	0.00	0	
0	0	100.00	0	0	0	45	0	1	1	...	0	0.0000	0	0.00	0	

26 columns

In [11]:
```
ouput = df["winPlacePerc"]
```

In [12]:
```
del df["winPlacePerc"]
```

In [13]:
```
df["matchType"] = df["matchType"].astype('category')
```

In [14]:
```
matchtype = df["matchType"]
```

In [15]:
```
matchtype = pd.get_dummies(matchtype)
```

In [18]:
```
df = df.apply(lambda x: pd.to_numeric(x), axis=0)
```

In [19]:
```
df = df.astype('float64')
```

In [20]:
```
df.dtypes
```

Out[20]:
```
assists             float64
boosts              float64
damageDealt         float64
DBNOs               float64
headshotKills       float64
heals               float64
killPlace           float64
killPoints          float64
kills               float64
killStreaks         float64
longestKill         float64
matchDuration       float64
maxPlace            float64
numGroups           float64
rankPoints          float64
revives             float64
rideDistance        float64
roadKills           float64
swimDistance        float64
teamKills           float64
vehicleDestroys     float64
walkDistance        float64
weaponsAcquired     float64
winPoints           float64
crashfpp            float64
crashtpp            float64
duo                 float64
duo-fpp             float64
flarefpp            float64
flaretpp            float64
normal-duo          float64
normal-duo-fpp      float64
normal-solo         float64
dtype: object
```

In [21]:
```
#Normalizing the Data
df = (df-df.mean())/(df.max()-df.min())
```

In [23]:
```
list(df.columns)
```

Out[23]:
```
['assists',
 'boosts',
 'damageDealt',
 'DBNOs',
 'headshotKills',
 'heals',
 'killPlace',
 'killPoints',
 'kills',
 'killStreaks',
 'longestKill',
 'matchDuration',
 'maxPlace',
 'numGroups',
 'rankPoints',
 'revives',
 'rideDistance',
 'roadKills',
 'swimDistance',
 'teamKills',
 'vehicleDestroys',
 'walkDistance',
 'weaponsAcquired',
 'winPoints',
 'crashfpp',
 'crashtpp',
 'duo',
 'duo-fpp',
 'flarefpp',
 'flaretpp',
 'normal-duo',
 'normal-duo-fpp',
 'normal-solo',
 'normal-solo-fpp',
 'normal-squad'
```

```
'squad',
'squad-fpp']
```

In [ ]:
```
#histograms
for i in list(df.columns):
    plt.figure()
    n, bins, patches = plt.hist(df[i],10 )#, 50, density=True, alpha=0.75)

    max_col_value = max(df[i])
    min_col_value = min(df[i])

    plt.xlabel(i)
    plt.ylabel('Frequency')
    plt.title(f'Histogram of {i}')
    #plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
    #plt.axis([40, 160, 0, 0.03])
    plt.grid(True)
    plt.show()

    print(f"Max and min values for {i}",max_col_value, min_col_value)


#Boxplot
for i in list(df.columns):
    plt.figure()
    n, bins, patches = plt.Boxplot(df[i],10 )#, 50, density=True, alpha=0.75)

    max_col_value = max(df[i])
    min_col_value = min(df[i])

    plt.xlabel(i)
    plt.ylabel('Frequency')
    plt.title(f'Histogram of {i}')
    #plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
    #plt.axis([40, 160, 0, 0.03])
    #plt.grid(True)
    #plt.show()

    #print(f"Max and min values for {i}",max_col_value, min_col_value)
    '''
```

In [25]:
```
import pandas_profiling
```

In [26]:
```
pandas_profiling.ProfileReport(df)
```

Overview

Dataset info

Number of variables	40
Number of observations	1048575
Total Missing (%)	0.0%
Total size in memory	320.0 MiB
Average record size in memory	320.0 B

Variables types

Numeric	22
Categorical	0
Boolean	16
Date	0
Text (Unique)	0
Rejected	2
Unsupported	0

Warnings

- numGroups is highly correlated with maxPlace (ρ = 0.99789) Rejected
- roadKills is highly skewed (γ1 = 33.721) Skewed

In [27]:
```
#Now we are going to going to find correlation inorder to find the relationship between the features.
#IF the dataset has exact pos and neg attributes then there is chance that model will be effected by multicollinearity.
#When one variable can be exactly predicte by other we can the both variables are multicollinear.

corr_df = df.corr(method='pearson')
print(corr_df)
```

```
                    assists    boosts  damageDealt     DBNOs  headshotKills  \
assists            1.000000  0.306184     0.406947  0.299507       0.197002
boosts             0.306184  1.000000     0.520833  0.360520       0.334814
damageDealt        0.406947  0.520833     1.000000  0.737337       0.613477
DBNOs              0.299507  0.360520     0.737337  1.000000       0.473397
headshotKills      0.197002  0.334814     0.613477  0.473397       1.000000
heals              0.227228  0.535368     0.342923  0.266306       0.201254
killPlace         -0.288667 -0.555363    -0.677056 -0.556120      -0.472402
killPoints         0.037573  0.008314     0.049836  0.042593       0.025761
kills              0.319412  0.501078     0.889202  0.710054       0.673677
killStreaks        0.241822  0.405323     0.702824  0.647578       0.514084
longestKill        0.260553  0.422571     0.563267  0.452262       0.449400
matchDuration     -0.019633  0.072693    -0.005337 -0.013946      -0.017509
maxPlace          -0.147993 -0.014157    -0.041226 -0.267389       0.008637
numGroups         -0.146951 -0.013441    -0.040697 -0.265576       0.008784
rankPoints        -0.014788  0.023453    -0.001155 -0.003104       0.003359
revives            0.198519  0.253580     0.258844  0.299736       0.152455
rideDistance       0.109959  0.327894     0.139854  0.102450       0.075627
roadKills          0.011784  0.035301     0.052527  0.037749       0.012583
```

In [28]:
```
df = pd.DataFrame(df)
print(ouput)
```

```
0          0.4444
1          0.6400
2          0.7755
3          0.1667
4          0.1875
5          0.0370
6          0.0000
7          0.7368
8          0.3704
9          0.2143
10         0.3929
11         0.4043
12         0.9286
13         0.8750
14         0.9000
15         0.2766
16         0.7308
17         0.8211
```

In [29]:
```
#dealing with Multicollinearity with VIF(variance Inflation Factor)
#VIF >=5 remove the feature
#Running a multiple regression
multi_linreg = LinearRegression()
```

In [30]:
```
multi_linreg.fit(df,ouput)
```

Out[30]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [31]:
```
r_square = multi_linreg.score(df,ouput)
r_square
```

Out[31]:
```
0.8404766631314233
```

In [32]:
```
VIF = 1/(1-r_square)
print(VIF)
```

```
6.268675289959927
```

In [33]:
```
ce Inflation factor is more than 5, we can assume there is multicollinearity.
olinearity if two or more than two are highly correlated then they try to influence the model and lead to wrong predict
ng highly correlated variables.
.columns
.tolist()
```

In [34]:
```
#printing out columns which are highly correlated.
```

In [34]:
```
#printing out columns which are highly correlated.
col_cor = []
def col_corr(inp):
    for i in range(len(ls)):
        if i ==len(ls):
            break
        else:

            for j in range(1,len(ls)):
                if (df[ls[i]].corr(df[ls[j]])) > .7:
                    col_cor.append((ls[i],ls[j]))
col_corr(ls)
```

In [35]:
```
df["DBNOs"].corr(df["damageDealt"])
```

Out[35]:
```
0.7373367445607778
```

In [36]:
```
def del_rep(list_in):
    for i in range(len(list_in)):
        if(list_in[i][0] == list_in[i][1]):
            list_in.pop(i)
            return(del_rep(list_in))
del_rep(col_cor)
```

In [37]:
```
print(col_cor)
```

```
[('damageDealt', 'DBNOs'), ('damageDealt', 'kills'), ('damageDealt', 'killStreaks'), ('DBNOs', 'damageDealt'), ('DBNO
s', 'kills'), ('killPoints', 'winPoints'), ('kills', 'damageDealt'), ('kills', 'DBNOs'), ('kills', 'killStreaks'),
('killStreaks', 'damageDealt'), ('killStreaks', 'kills'), ('maxPlace', 'numGroups'), ('maxPlace', 'solo-fpp'), ('numG
roups', 'maxPlace'), ('numGroups', 'solo-fpp'), ('winPoints', 'killPoints'), ('solo-fpp', 'maxPlace'), ('solo-fpp',
'numGroups')]
```

In [38]:
```
df["maxPlace"].corr(df["numGroups"])
```

Out[38]:
```
0.9978921632617327
```

In [39]:
```
print(list(set(sorted(col_cor))))
```

```
[('numGroups', 'maxPlace'), ('solo-fpp', 'numGroups'), ('kills', 'damageDealt'), ('solo-fpp', 'maxPlace'), ('numGroup
```

In [40]:
```
col_tobedeleted = []
for i in col_cor:
    col_tobedeleted.append(i[0])
print(list(set(col_tobedeleted)))
```

```
['killPoints', 'kills', 'numGroups', 'damageDealt', 'solo-fpp', 'winPoints', 'DBNOs', 'killStreaks', 'maxPlace']
```

In [41]:
```
killStreaks = df["killStreaks"]
killpoints  = df["killPoints"]
DBNOs       = df["DBNOs"]
del df["killStreaks"]
del df["killPoints"]
del df["DBNOs"]
```

In [42]:
```
multi_linreg.fit(df,ouput)
r_square = multi_linreg.score(df,ouput)
VIF = 1/(1-r_square)
print(VIF)
```

```
5.397670292528334
```

In [43]:
```
damaDealt = df["damageDealt"]

del df["damageDealt"]
```

In [44]:
```
multi_linreg.fit(df,ouput)
r_square = multi_linreg.score(df,ouput)
VIF = 1/(1-r_square)
print(VIF)
```

```
5.389203263195229
```

In [45]:
```
kill = df['kills']
maxplac = df['maxPlace']
winpoint = df['winPoints']
numgroup = df['numGroups']
del df['kills']
```

In [47]:
```
df.columns
```

Out[47]:
```
Index(['assists', 'boosts', 'headshotKills', 'heals', 'killPlace',
       'longestKill', 'matchDuration', 'rankPoints', 'revives', 'rideDistance',
       'roadKills', 'swimDistance', 'teamKills', 'vehicleDestroys',
       'walkDistance', 'weaponsAcquired', 'crashfpp', 'crashtpp', 'duo',
       'duo-fpp', 'flarefpp', 'flaretpp', 'normal-duo', 'normal-duo-fpp',
       'normal-solo', 'normal-solo-fpp', 'normal-squad', 'normal-squad-fpp',
       'solo', 'solo-fpp', 'squad', 'squad-fpp'],
      dtype='object')
```

In [48]:
```
#we should use regression if we are predicting a continous quantity.
#Linear Regression
X_train,X_test, y_train, y_test = train_test_split(df,ouput,test_size=.2,random_state=101)
```

In [49]:
```
lm= LinearRegression()
lm.fit(X_train,y_train)
```

Out[49]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [50]:
```
predictions = lm.predict(X_test)
```

In [51]:
```
print(predictions)
```

```
[0.06854714 0.79523171 0.24828041 ... 0.37992943 0.39949501 0.40457238]
```

In [52]:
```
print(len(predictions))
print(len(y_test))
print(len(X_train))
```

```
209715
209715
838860
```

```
838860
```

In [56]:
```
#Evaluation of regression model
#mean_absolute_error(y_test,predictions)
mean_squared_error(y_test, predictions)
```

Out[56]:
```
0.019194943232088358
```

In [ ]:
```
#lasso
```

In [57]:
```
#GradientBoosting regression
#n_estimators: the number of boosting stages to perform. This method is roboust towards overfitting
#Gradient Boosting a method of converting weak learners into a strong learners.
params = {'n_estimators': 500, 'max_depth': 4, 'min_samples_split': 2,
          'learning_rate': 0.01, 'loss': 'ls'}
clf = ensemble.GradientBoostingRegressor(**params)

clf.fit(X_train, y_train)
output_GBM = clf.predict(X_test)
```

In [175]:
```
mean_absolute_error(y_test,output_GBM)
```

Out[175]:
```
0.07301925718909927
```

In [58]:
```
mean_squared_error(y_test, output_GBM)
```

Out[58]:
```
0.010533641489882443
```