PUBG Mobile data analysis and finish placement prediction.

**Nouran Ali Mohammed**

**ID: 100064373**

**Due: 19/11/2020 2:55 PM**

# Abstract

In this assignment, I’ll be analyzing a real-world dataset from PUBG mobile game, which is an online battle game, I will also build a model to predict the winning place percentage of a player. In a PUBG game, up to 100 players start in each match (matchId). Players can be on teams (groupId) that 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. **My goal** in this project is to build a storyline of the perfect winning PUBG match with the help of EDA and ML.

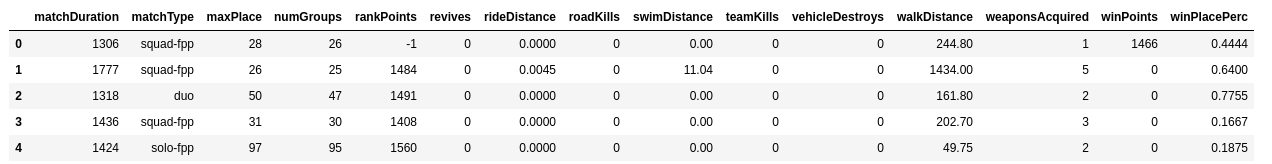
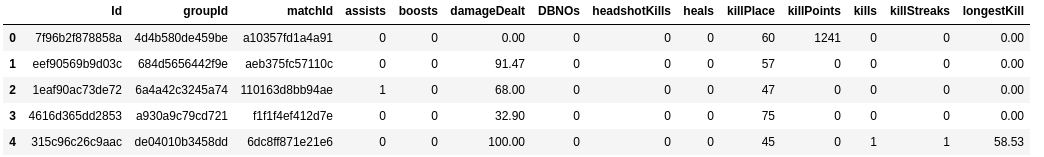
# Data Gathering and cleaning:

1- Dataset description:

This dataset is available for access through Kaggle [here](https://www.kaggle.com/c/pubg-finish-placement-prediction/data), it has 29 column attributes and comes as 2 CSV files one for training and one for testing in a size total of 920.93 MB.

Column fields are described in detail on the dataset page on Kaggle.

Here’s a snapshot of the data



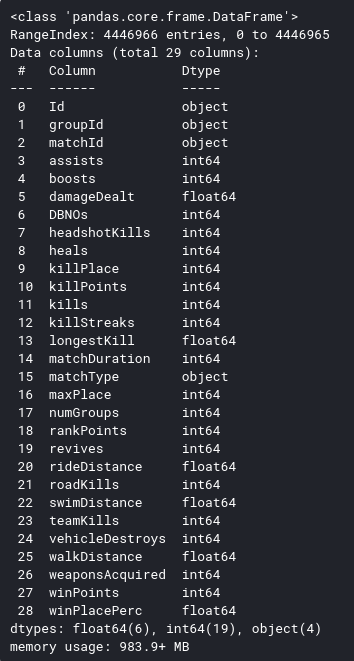
2- Data Cleaning:

I used Pandas and Numpy to assist the cleaning process easier and to investigate the dataset by getting statistical info about the data and here’s what I did with this info. :

* The dataset contained only 1 missing value in the target column, so I dropped it as it was not significant “The dataset contains 4446966 rows.

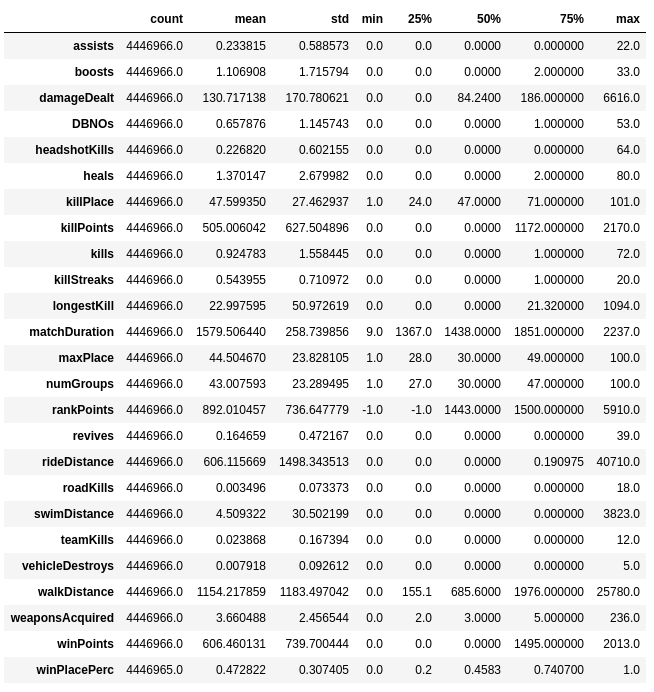


* Some data types needed correction.



* There are outliers in the data but they’re important as they’re not a result of a wrong entry, but they indicate extremely skilled players which might help us in the modeling phase.
* Some of the columns contained similar data so I gathered them in 1 category like the match type.

Here’s a statistical description of the data



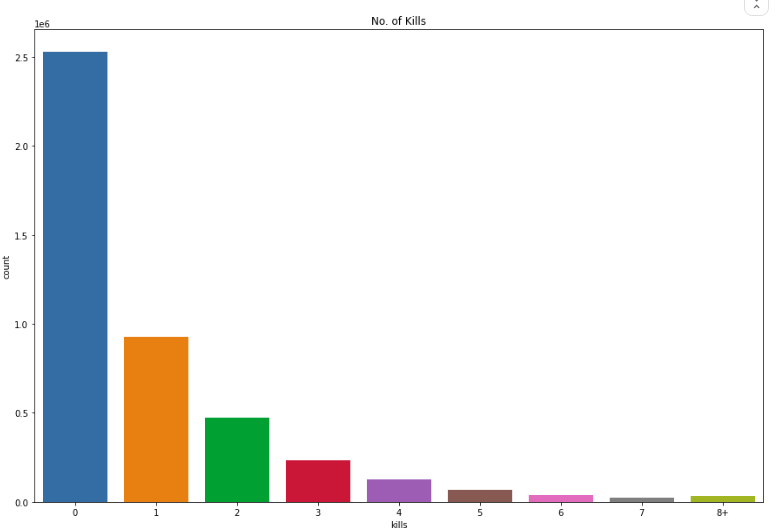
# EDA and Plotting:

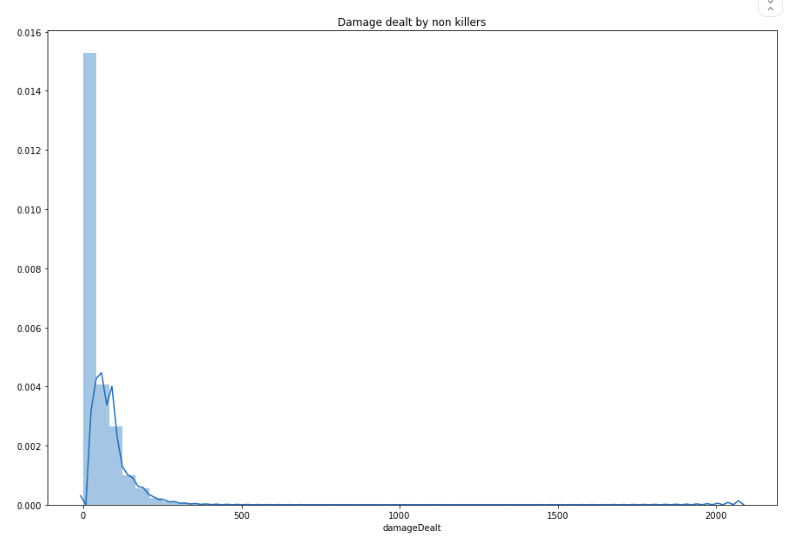
In this stage, I use plots to investigate the data and build the outline of my story and answer some questions that lead me to build a hypothesis.

I used Seaborn and Matplotlib to produce the plots.

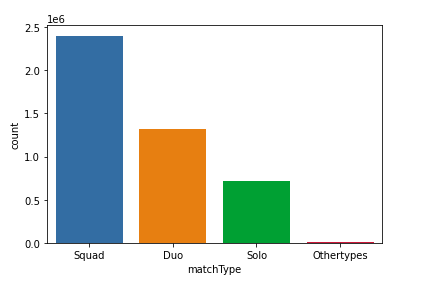
1- Investigate the behavior of the killer players by using a histogram

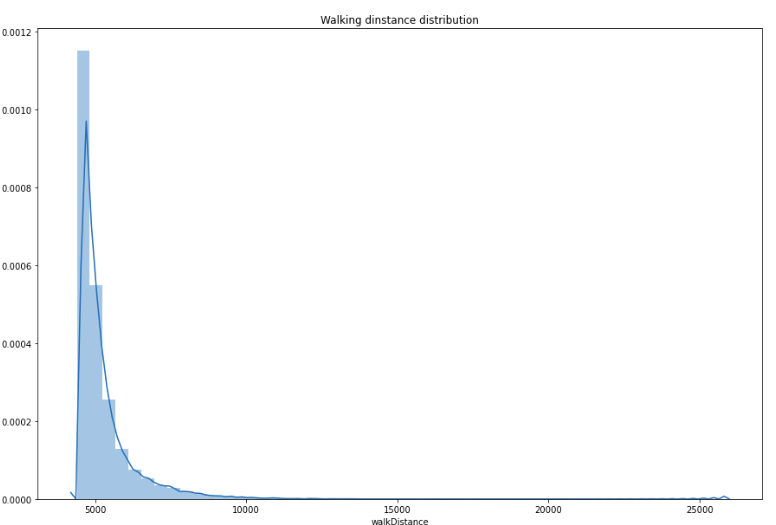
* For a clear plot, I replaced all the data that’s greater than the 99% quantile ( which is equal to 7) with an “8+” label, with this plot I concluded that most players don’t kill any of their enemies, so I moved on to another investigation.



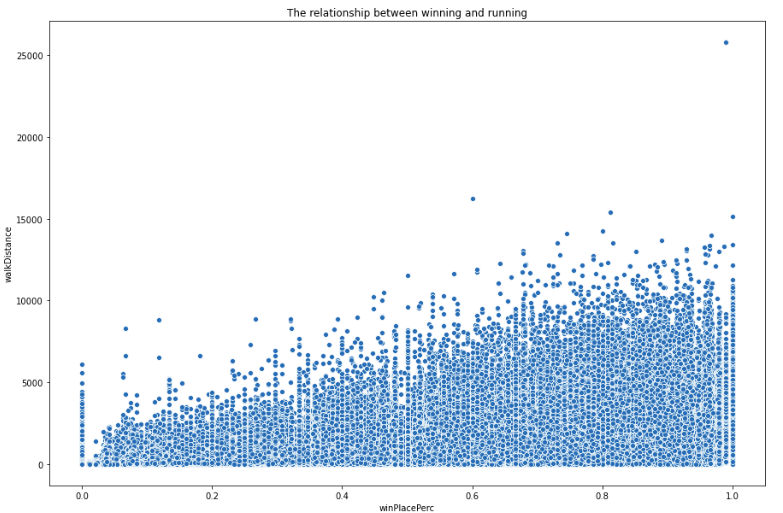
* Does nonkillers at least damage their opponents? The damageDealt column is of type float so I needed to show the distribution of it using a distribution plot which shows that most nonkillers don’t deal any damage.
* 

2- Then checked who plays more Solos, Duos, or Squad players “Squad=4 players”, I did here some more optimization to combine all players of the 3 types into just 3 categories rather than 16, the plot showed that 54% of players play as squads.



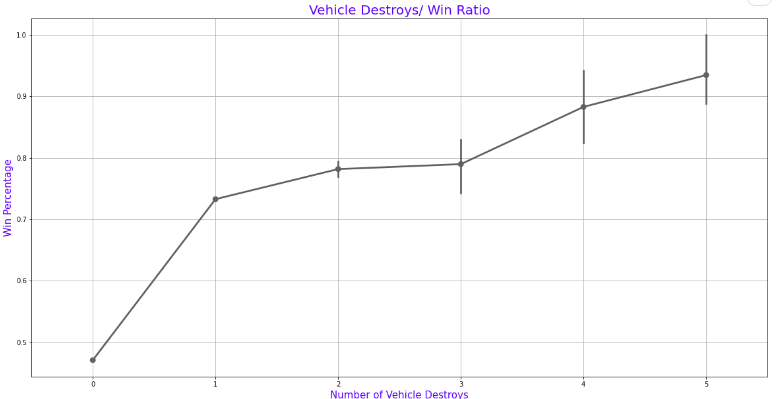
3- Then I went ahead to check movement behavior and if it affects winning, I plotted the distribution of walking distance first and here most players walk 5000 KM

* What about the relationship between walking and winning? Most walking players win according to the scatter plot with a correlation coefficient of 0.81

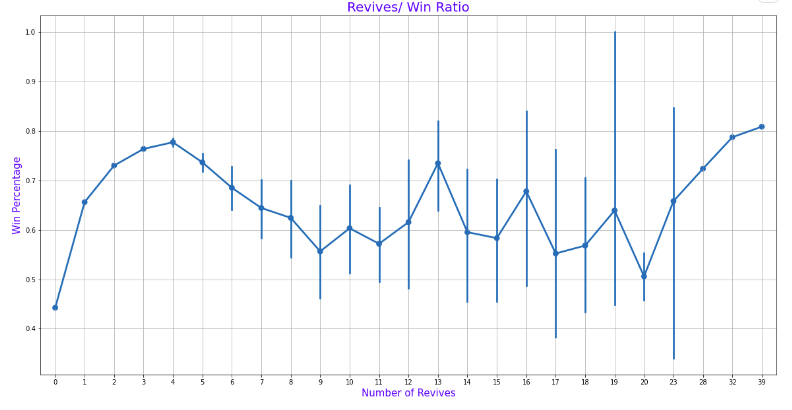


* Riding vehicles and swimming on the other hand doesn’t have that much of an effect...

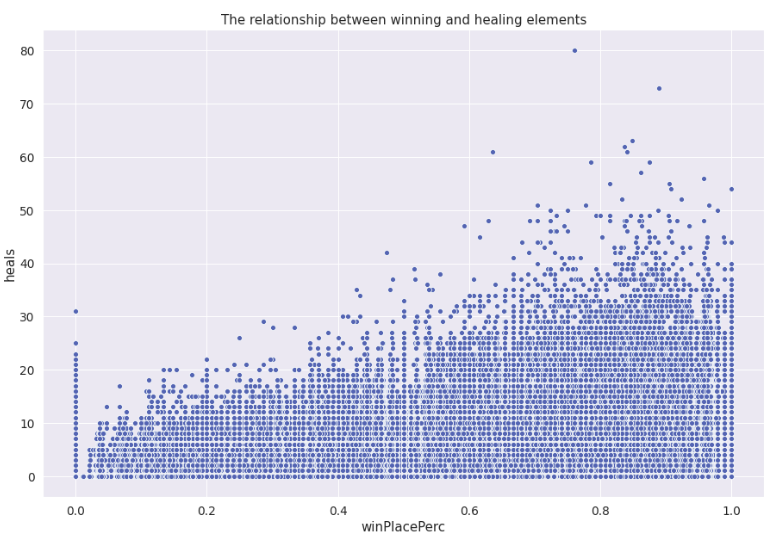
4- An interesting PUBG playing strategy is that you can destroy your enemies cars while they’re riding it which leads to their death, let’s check if that has a relation with winning by using a point plot, it shows that destroying at least on vehicle increases the chance of winning by ~35%, AWESOME!!



5- PUBG is a team-based game, when a member of your team is knocked down, you can revive him and bring him back in the game as long as he's not dead..let's check if that affects the winning. I also used a line plot here, but it shows that my hypothesis was not true as it’s kinda random



6- Final thing I was thinking about does the usage of healing elements and health boosts such as pain killers and adrenaline shots relate to winning, and it does as shown in the plots below with a correlation coefficient of 0.42 for heals, and 0.634 for boosts



7- Then I visualized the top 5 correlated features with the target with a heat map showing the correlation coefficients.



# ML modeling technical discussion:

**Feature engineering:** ML is about quality data and relevant features so first in this section, I’ll create some new features and select the features that I believe to be most important as well as doing some normalization and correcting.

1- Creating new feature playersJoined that indicate the number of players in a game:

A PUBG game typically has 100 players but sometimes not all the 100 players join, by grouping the matchID to get the number of players in each game session.

2- Combine boosts and heals elements in one column to create a stronger feature by summing the two columns,

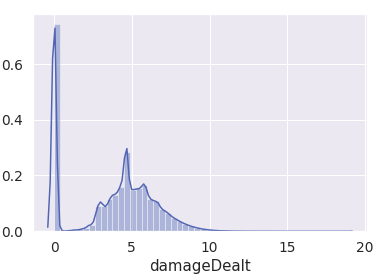
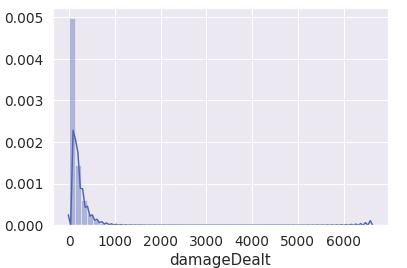
3- Get the number of players in a team corresponding to solos, duos, and squads in team columns

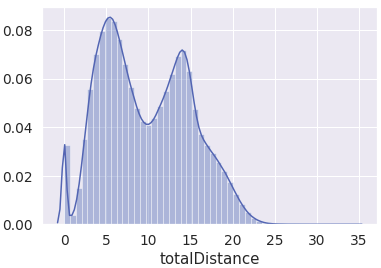
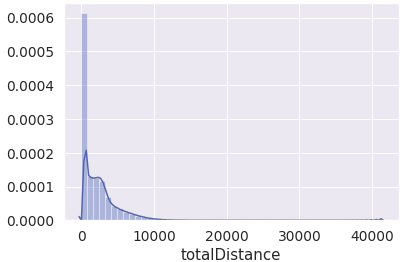
4- Selecting the most promising features:

['assists','healsAndBoosts','damageDealt','DBNOs','kills','team','revives','playersJoined','totalDistance','weaponsAcquired','winPlacePerc']

5- An important finding here is that the columns: winPlacePerc and damageDealt were severely +ve skewed so I normalized them using cube root transformation in order to make them normally distributed to achieve the regression assumptions.

Before skewness and after corrections:





6- Applying standard scaler scaling to data using built-in StandardScaler in SKlearn.

**Technical Discussion of models used:**

I used random forest and Multilayer perceptron regression and compared the results of each model, both using 67% of data for train,33% for test. I also didn’t have enough time to optimize the hyperparameters so I used the default values which gave quite modest results.

* **Random forest Regressor technical discussion:**

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

The features are always randomly permuted at each split. Therefore, the best found split may vary, even with the same training data,

*Default Parameters values are available in the sklearn docs*

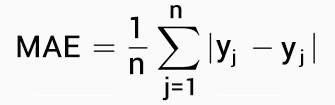
* **MLPRegressor Technical Discussion:**

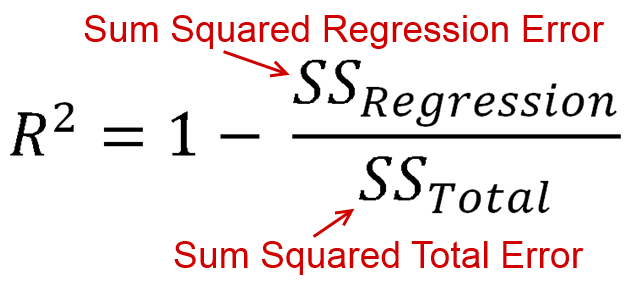
A simple neural network regressor, This model optimizes the squared-loss using stochastic gradient descent, MLPRegressor trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters.

# Discussion of results and project objectives:

**Results obtained:**

I used MAE and r2 score to measure performance





* **Random forest Regressor results:**

It took too much memory and time to run this model (as it ensembles many model estimators) so I had to split the data into smaller batches to complete the training process.

MAE= 0.1157361894830011

R2 score=0.743

* **MLPRegressor:**

This model took also much time to finish but not as much memory as the above one, It also gave better results.

MAE=0.10920233436658276 and R2 score= 0.77

**Problems and future enhancements:**

1- The dataset was very large so it took a long time in the training, I split the data into smaller batches to train the random forest regressor.

2- The metrics indicate that the model is not that good so I can try more optimization methods and feature engineering techniques to get a better result.

3- I intend to use a DL approach and compare against SVR and see how it turns out “I didn’t have enough time to train them, unfortunately”

**The winning strategy(My objective):**

So I’ve said in the beginning that I’m trying to find out the best strategy to win a PUBG game by using analytics, so here’s what I figured out from the analysis:

1- Play in a team.

2- Use healings and health-boosting elements.

3- Destroy your enemies vehicles.

4- Kill as many enemies as you can.

5- Move a lot and collect powerful weapons.

# 

# Appendix:

|  |
| --- |
| import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.preprocessing import StandardScaler from sklearn.ensemble import RandomForestRegressor from sklearn.model\_selection import train\_test\_split from sklearn.metrics import mean\_absolute\_error,r2\_score #reading training data train = pd.read\_csv('/kaggle/input/pubg-finish-placement-prediction/train\_V2.csv') #data exploration  train.info()  Train.shape  train.head()  train.isnull().sum()  train.describe().T def data\_cleaning(df):  df=df.dropna()  df.matchType.replace(['squad-fpp','squad','normal-squad-fpp','normal-squad'],'Squad',inplace=True)  df.matchType.replace(['duo-fpp','duo','normal-duo-fpp','normal-duo'],'Duo',inplace=True)  df.matchType.replace(['solo-fpp','solo','normal-solo-fpp','normal-solo'],'Solo',inplace=True) df.matchType.replace(['crashfpp','flaretpp','flarefpp','crashtpp'],'Othertypes',inplace=True)  df['playersJoined'] = df.groupby('matchId')['matchId'].transform('count')  df['healsAndBoosts'] = df['heals']+df['boosts']  df['totalDistance'] = df['walkDistance']+df['rideDistance']+train['swimDistance']  df['team'] = [1 if i>50 else 2 if (i>25 & i<=50) else 4 for i in df['numGroups']]  df=df[['assists','healsAndBoosts','damageDealt','DBNOs','kills','team','playersJoined','totalDistance','weaponsAcquired','winPlacePerc']]  df['damageDealt']=df['damageDealt']\*\*(1/3)  df['totalDistance']=df['totalDistance']\*\*(1/3)  return df  #plots and EDA train.info() train.isnull().sum() train.describe().T train['kills'].quantile(0.99) #replace any no of kills greater than 0.99 of data with 8 kills for better visuals temp= train.copy() temp.loc[temp['kills'] > temp['kills'].quantile(0.99)] = '8+' plt.figure(figsize=(15,10)) sns.countplot(temp['kills'].astype('str').sort\_values()) plt.title('No. of Kills') plt.show() #plot damage of nonkillers temp= train.copy() temp =temp[temp['kills']==0] plt.figure(figsize=(15,10)) sns.distplot(temp['damageDealt']) plt.title('Damage dealt by non killers');  #working on matchtype train['matchType'].value\_counts() plt.figure(figsize=(15,10)) plt.xticks(rotation=45) sns.countplot(train['matchType'].astype('str')) plt.show() train.matchType.replace(['squad-fpp','squad','normal-squad-fpp','normal-squad'],'Squad',inplace=True) train.matchType.replace(['duo-fpp','duo','normal-duo-fpp','normal-duo'],'Duo',inplace=True) train.matchType.replace(['solo-fpp','solo','normal-solo-fpp','normal-solo'],'Solo',inplace=True) train.matchType.replace(['crashfpp','flaretpp','flarefpp','crashtpp'],'Othertypes',inplace=True) sns.countplot(train.matchType); print('{}% of players play as Squads'.format(train.matchType.value\_counts()['Squad']/len(train.matchType) \*100 )) ## The running players print('A player travels an avg distance of {} meters'.format(train['walkDistance'].mean())) temp= train.copy() temp=temp[temp['walkDistance'] > temp['walkDistance'].quantile(0.99)]  plt.figure(figsize=(15,10)) sns.distplot(temp['walkDistance']) plt.title('Walking dinstance distribution'); plt.figure(figsize=(15,10)) sns.scatterplot(x='winPlacePerc',y='walkDistance',data=train) plt.title('The relationship between winning and running') train[['winPlacePerc','walkDistance']].corr() plt.figure(figsize=(15,10)) sns.scatterplot(x='winPlacePerc',y='rideDistance',data=train) plt.title('The relationship between winning and driving') train[['winPlacePerc','rideDistance']].corr() del temp #line plot of destroying vehicles f,ax1 = plt.subplots(figsize =(20,10)) sns.pointplot(x='vehicleDestroys',y='winPlacePerc',data=train,color='#606060',alpha=0.8) plt.xlabel('Number of Vehicle Destroys',fontsize = 15,color='blue') plt.ylabel('Win Percentage',fontsize = 15,color='blue') plt.title('Vehicle Destroys/ Win Ratio',fontsize = 20,color='blue') plt.grid() plt.show() #line plot of revives f,ax1 = plt.subplots(figsize =(20,10)) sns.pointplot(x='revives',y='winPlacePerc',data=train,alpha=0.8) plt.xlabel('Number of Revives',fontsize = 15,color='blue') plt.ylabel('Win Percentage',fontsize = 15,color='blue') plt.title('Revives/ Win Ratio',fontsize = 20,color='blue') plt.grid() plt.show() #healing and boosts plt.figure(figsize=(15,10)) sns.scatterplot(x='winPlacePerc',y='heals',data=train) plt.title('The relationship between winning and healing elements') plt.show() train[['winPlacePerc','heals']].corr() train[['winPlacePerc','boosts']].corr() #heatmap of top 5 related features f,ax = plt.subplots(figsize=(11, 11)) cols = train.corr().nlargest(5, 'winPlacePerc')['winPlacePerc'].index cm = np.corrcoef(train[cols].values.T) sns.set(font\_scale=1.25) hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot\_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values) plt.show()  #feature engineering train['playersJoined'] = train.groupby('matchId')['matchId'].transform('count') train['healsAndBoosts'] = train['heals']+train['boosts'] train['totalDistance'] = train['walkDistance']+train['rideDistance']+train['swimDistance'] train['team'] = [1 if i>50 else 2 if (i>25 & i<=50) else 4 for i in train['numGroups']] train=train[['assists','healsAndBoosts','damageDealt','DBNOs','kills','team','playersJoined','totalDistance','weaponsAcquired','winPlacePerc']] #target and features X=train.drop('winPlacePerc',axis=1) y=train['winPlacePerc'] #skewed columns sns.distplot(X['damageDealt']) plt.show() sns.distplot(X['totalDistance']) plt.show() X['damageDealt']=X['damageDealt']\*\*(1/3) sns.distplot(X['damageDealt']) plt.show() X['totalDistance']=X['totalDistance']\*\*(1/3) sns.distplot(X['totalDistance']) plt.show() #scaling x\_cols=X.columns sc=StandardScaler() #modeling X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42) rf=RandomForestRegressor() rf\_m=rf.fit(X\_train,y\_train) y\_pred=rf\_m.predict(X\_test) print("MAE={}".format(mean\_absolute\_error(y\_test, y\_pred))) print("R2 score={}".format(r2\_score(y\_test, y\_pred))) # MLP regressor mlp=MLPRegressor(random\_state=1, max\_iter=500).fit(X\_train, y\_train) ml=mlp.predict(X\_test) print("MAE={}".format(mean\_absolute\_error(y\_test, ml))) print("R2 score={}".format(r2\_score(y\_test, ml))) |
|  |
|  |