ML PROJECT

PUBG Placement Prediction

REPORT ON THE KAGGLE PROJECT

Taught By:

Prof. Dinesh Babu

Prof. Neelam Sinha

TA Name:

Preyas Garg

Submitted By:

Aditya M[MT2022161]

Kiran Kumar B.[MT2022163]

Team Name: Newbies

Description of Project:

In the 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.

Final Standing in the Kaggle Competition

- 5th Position

Objective:

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.

We are expected to create a model which predicts players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place).

Evaluation Metrics:

- Mean Squared Error (MSE)
- Mean Average Error (MAE)

Data Set Description:

1. File descriptions
   a. train.csv - the training set
   b. test.csv - the test set
   c. mocksubmission.csv - a sample submission file in the correct format

2. Data Fields:

There are 29 Data fields brief description is provided below:

- 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 me ters.
- roadKills - Number of kills while in a vehicle.
- swimDistance - Total distance traveled by swimming measured i n meters.
- teamKills - Number of times this player killed a teammate.
- vehicleDestroys - Number of vehicles destroyed.
- walkDistance - Total distance traveled on foot measured in meter s.
- 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 tr eated as a "None".
- groupId - ID to identify a group within a match. If the same grou p 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 ove r placements.
- winPlacePerc - The target of prediction. This is a percentile winn ing placement, where 1 corresponds to 1st place, and 0 correspon ds 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.

Initial observations from the data:

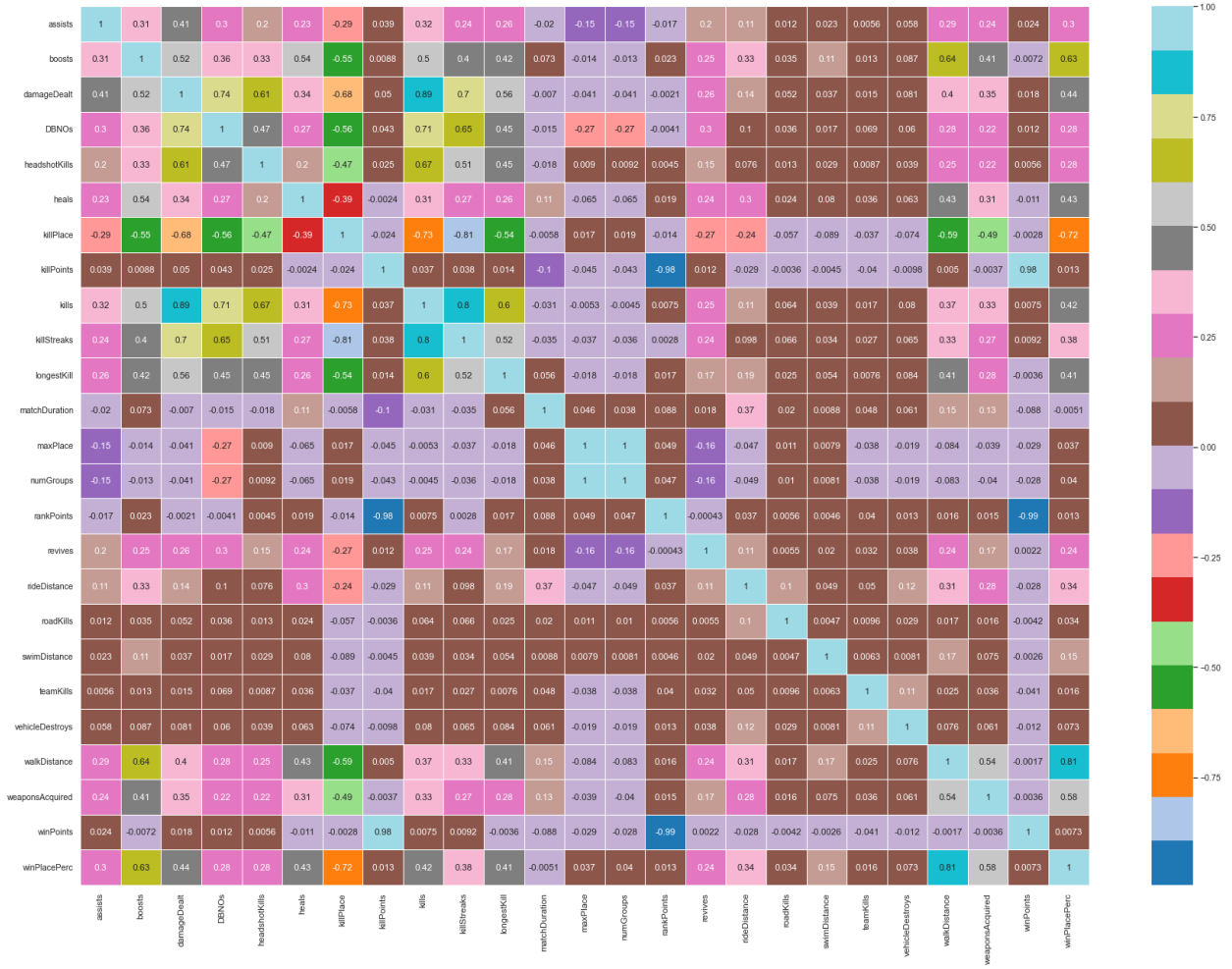
1. We noticed that many features like 'assists', 'DBNOs', 'revives','teamKills' have 75% or more of the values as 0. This is because these features are applicable only for team matches and the dataset is a combination of solo, duo and squad matches.

```
]: db.describe()
]:
```

	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	killPoints	kills	killStreaks	longe
count	3.112876e+06	3.112876e+06	3.112876e+06	3.112876e+06	3.112876e+06	3.112876e+06	3.112876e+06	3.112876e+06	3.112876e+06	3.112876e+06	3.11287
mean	2.339239e-01	1.107759e+00	1.308017e+02	6.587818e-01	2.268834e-01	1.370783e+00	4.758207e+01	5.050118e+02	9.252460e-01	5.442054e-01	2.30241
std	5.887983e-01	1.716308e+00	1.707582e+02	1.146708e+00	6.026371e-01	2.679204e+00	2.746282e+01	6.275414e+02	1.558777e+00	7.111749e-01	5.10003
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
25%	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.400000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
50%	0.000000e+00	0.000000e+00	8.435000e+01	0.000000e+00	0.000000e+00	0.000000e+00	4.700000e+01	0.000000e+00	0.000000e+00	0.000000e+00	0.00000
75%	0.000000e+00	2.000000e+00	1.862000e+02	1.000000e+00	0.000000e+00	2.000000e+00	7.100000e+01	1.171000e+03	1.000000e+00	1.000000e+00	2.13600
max	2.200000e+01	3.300000e+01	6.616000e+03	5.300000e+01	6.400000e+01	8.000000e+01	1.000000e+02	2.170000e+03	7.200000e+01	2.000000e+01	1.07500

2. One of the row has no 'winPlacePerc' this is not useful hence we deleted the row.

Correlation Matrix:



Observations from correlation Matrix

- Here we can see that the feature 'walkDistance' has high positive correlation with the target variable. This is because in many match types as the play area keeps decreasing player has to move inside the circle

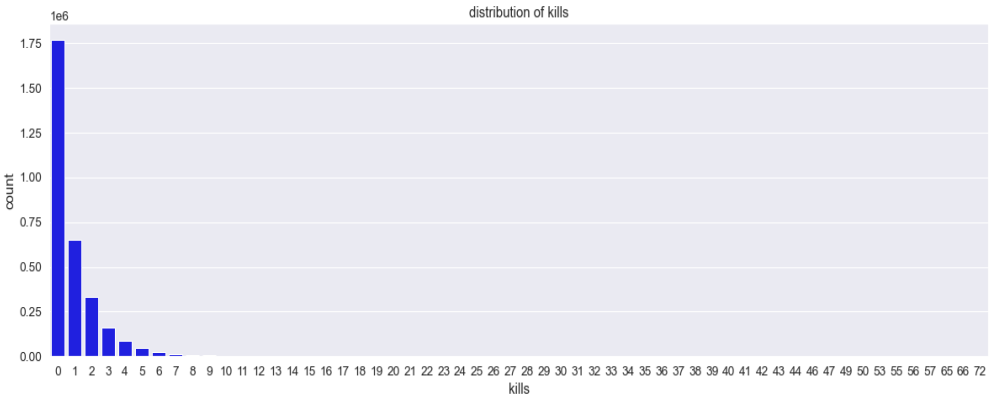
hence any player who wins should walk some distance. 'killPlace' has high negative correlation with the target variable.

- Even though 'kill' should greatly impact the chance of winning here correlation is less, this is because correlation can only show linear dependency between 2 variables.

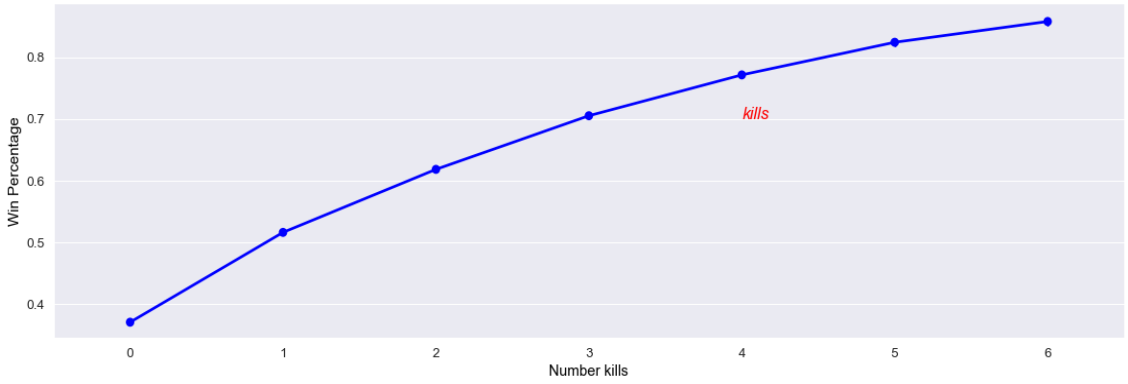
Data Cleaning and Visualization:

We are performing visualization by observing the features one by one and trying to find the relation among the features and impact of the feature on the target variable.

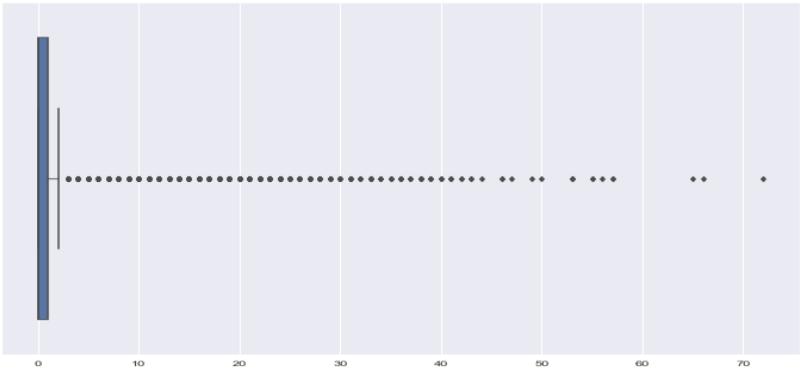
1. killPoints and rankPoints:



- Here we can see that 99% of the players have less than or equal to 7 kills. While on average a player will make 0.92 kills per match. Maximum number of kills recorded is 72.

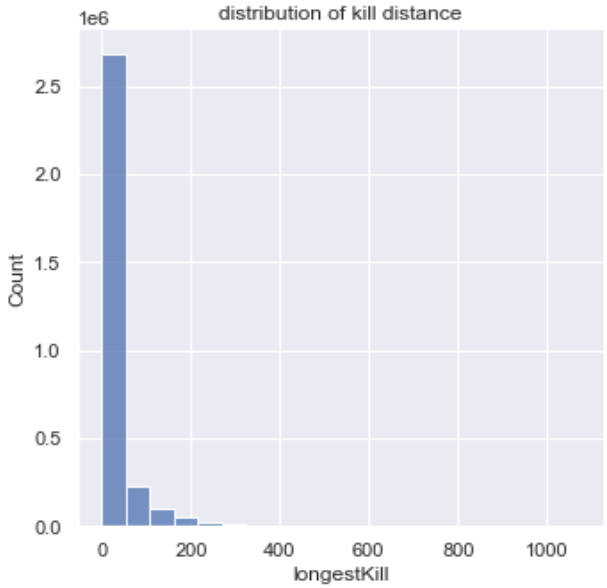


- From the above plot we can observe that the number of kills influence the win Percentage.



- 99.99% of the players have 21 kills or less, but remaining 0.01% of the players might be very good at the game or cheating. We can drop these outliers.

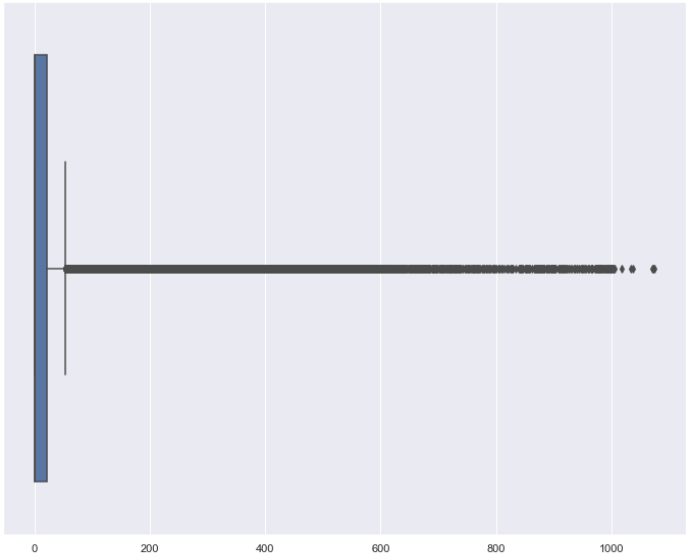
2. Longest Kill:




```
db['longestKill'].quantile(0.999)
```

```
415.4
```

- As we can see 99.9% of the players have the longestKill less than 416. Maximum value is 1075 which is almost impossible. This indicates that either the player is cheating or he knocked down a player and travelled very long distance. Hence we are getting rid of these outliers.

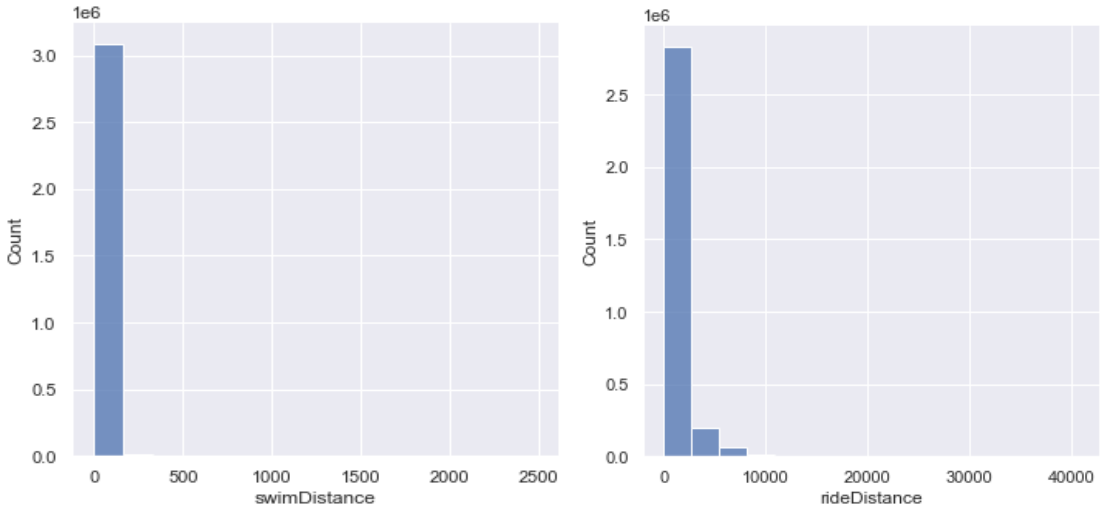


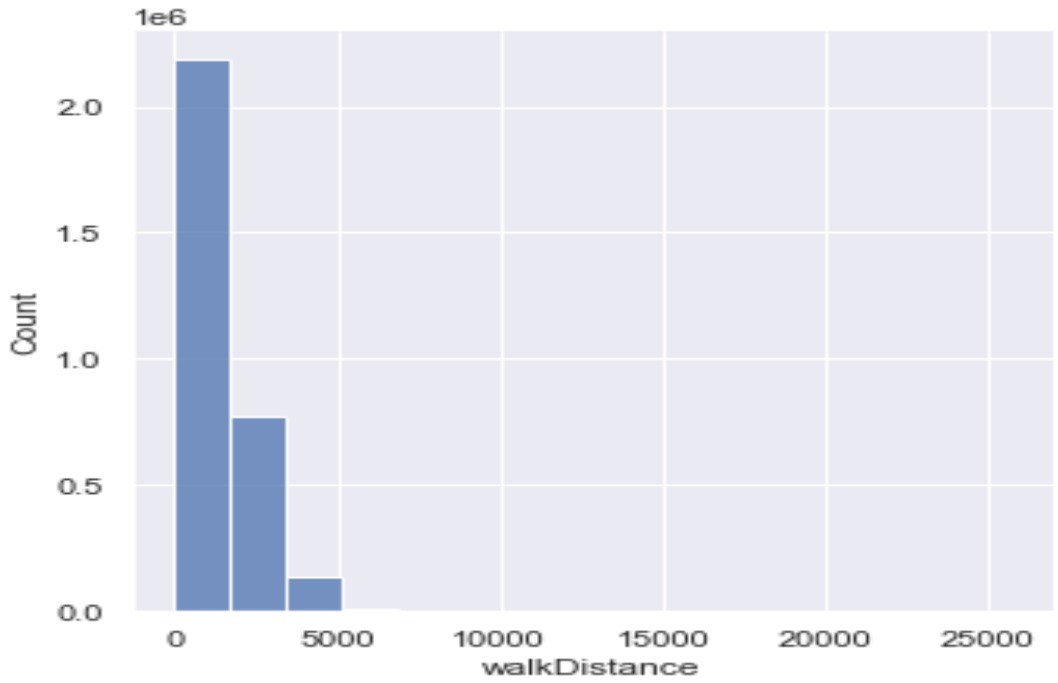
pic. Depicting no. of outliers that can be removed.

3. Total Distances:

Newly added feature combining walk, swim and ride distances.

- We observe that winners should travel more. But, travelling more will not guarantee win. Also, as all the 3 features are needed for the model and are similar we can create a new feature 'totalDistance' as summation of all the 3 features.






```
db['walkDistance'].quantile(0.9999)
```

```
9096.390999995172
```

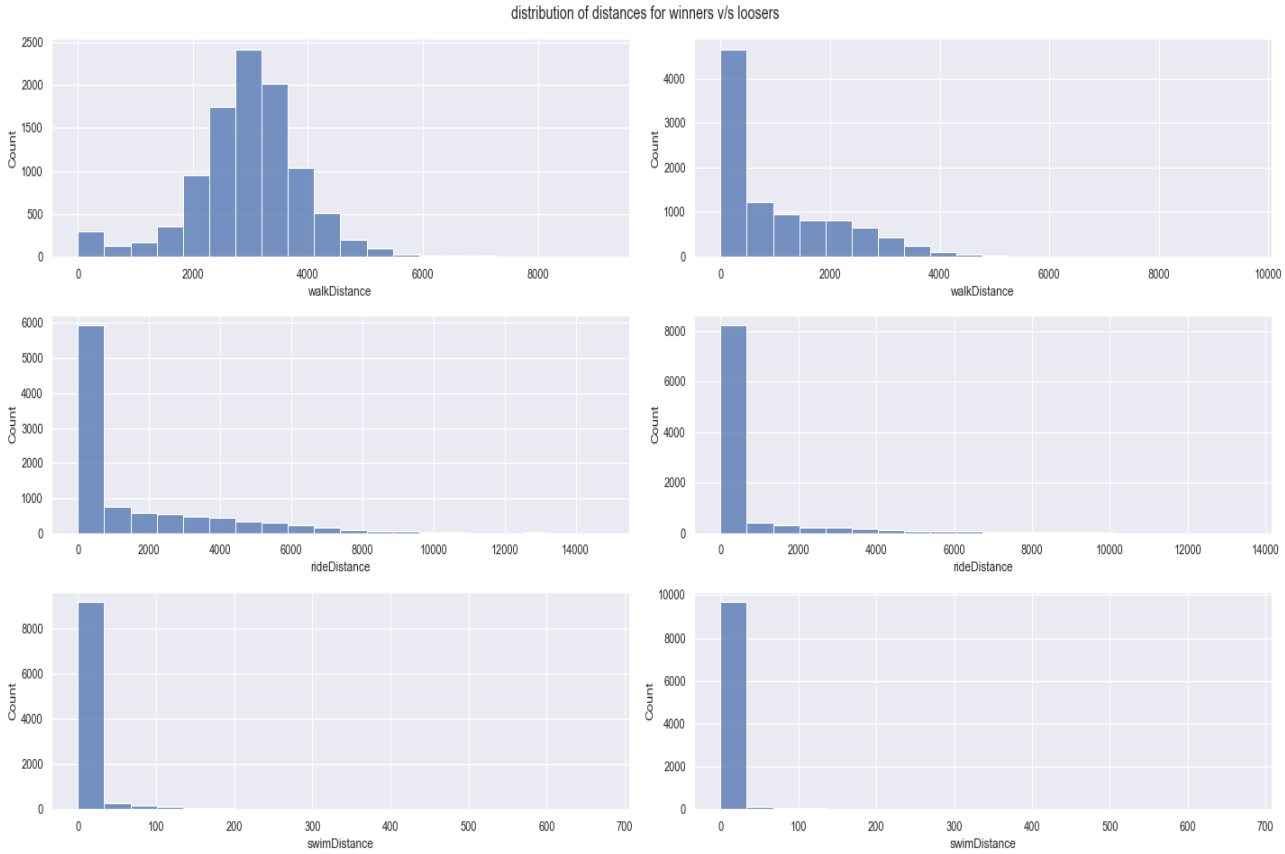
```
db['rideDistance'].quantile(0.9999)
```

```
15353.563999980688
```

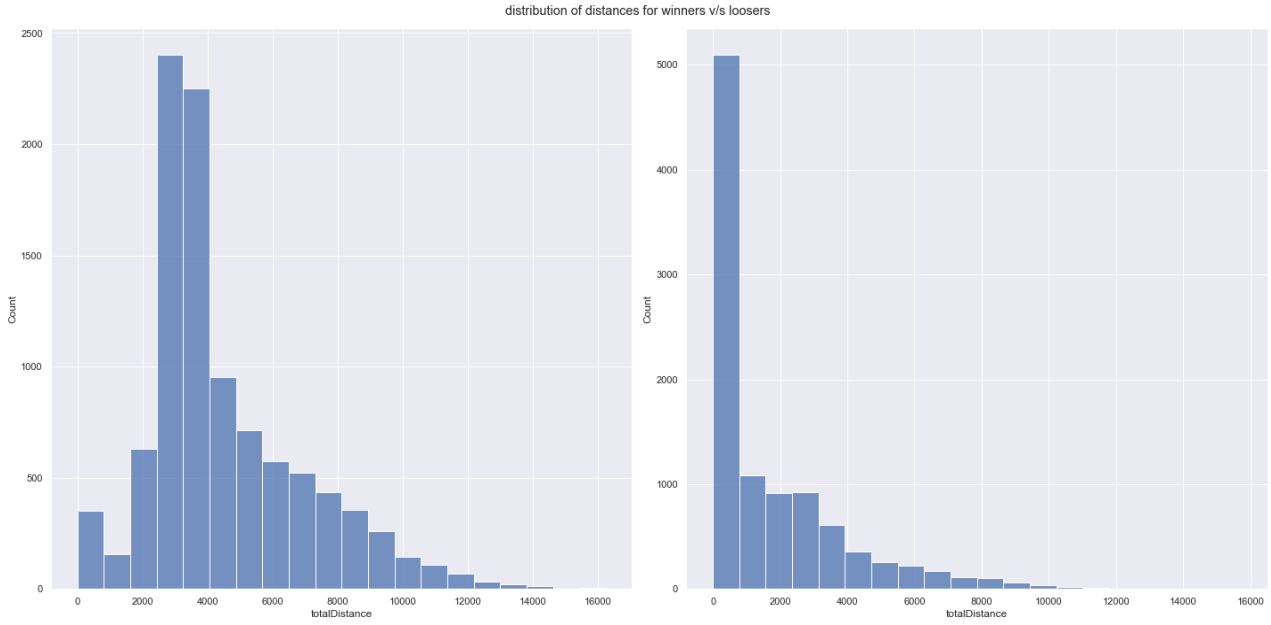
```
db['swimDistance'].quantile(0.9999)
```

```
783.9747399993241
```

Based on the quantile information we're removing the outliers accordingly in the dataset.

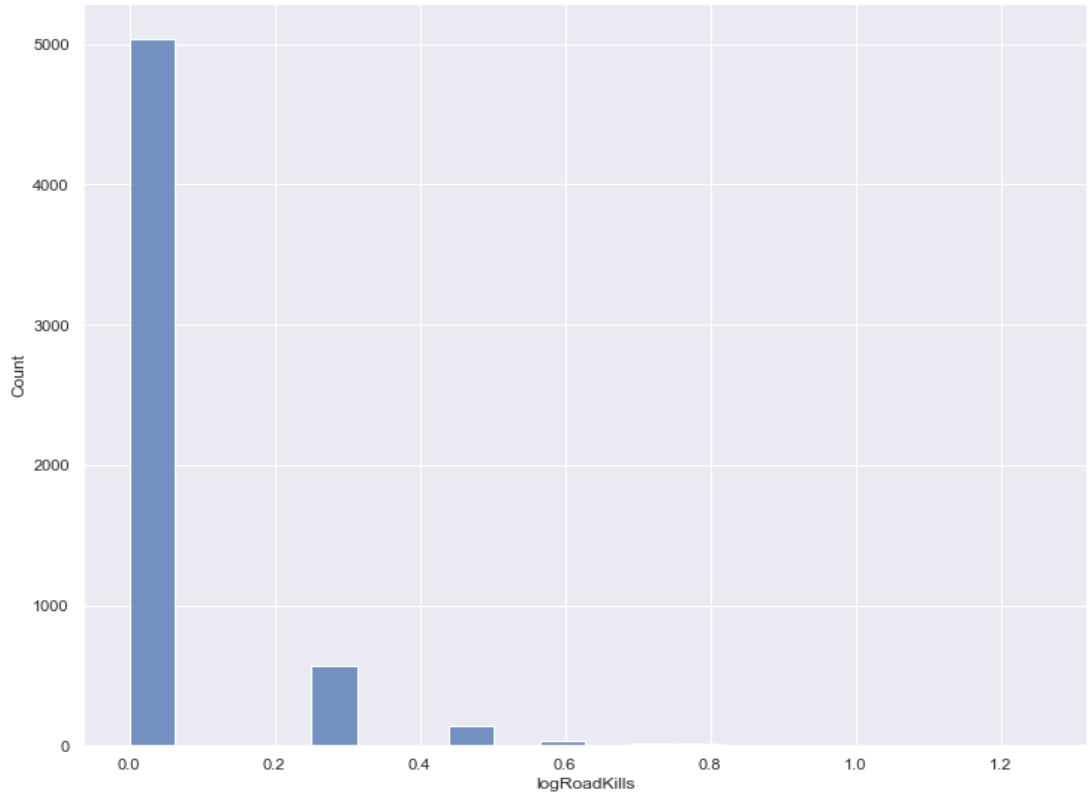


- We observed that winners travel certain distance. Also we can observe that travelling more will not guarantee win.
- Also, as all these 3 features are needed for the model and are similar we can create a new feature 'totalDistance' as combination of all the 3 features.
- Since we observed high correlation of WalkDistance with WinPercentage, hence we've created 4 more features based on percentages, those are namely walkdistance per matchduration, no. of kills per walkdistance, damagedealt per walkdistance and weapons acquired per walkdistance, which are self explanatory as per their names.



4. Roadkill:

- We can see that 99.99% of players have less than or equal to 3 road kills, based on the quantile range.



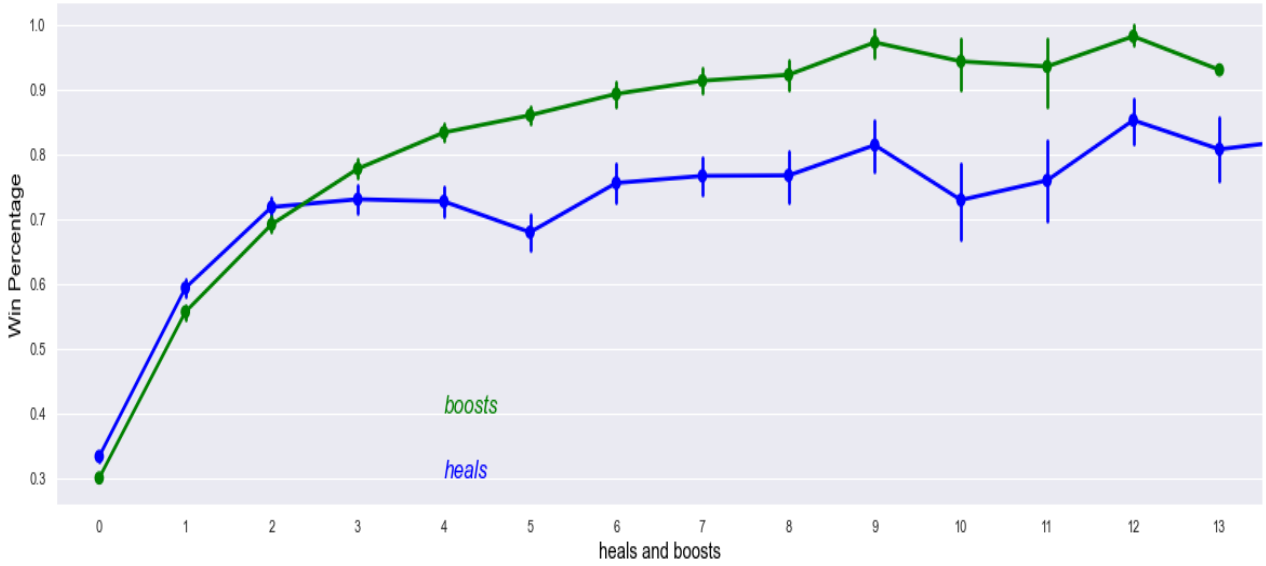
We can see that 99.99% of players have less than or equal to 3 road kills.

5. Removing Cheaters[killing without moving] :

- We found that players are killing without even moving from a place in the entire match, which is highly impossible. Hence, we created a feature named Cheaters and tried to remove them.
- There are 1067 players who have made more than one kill without movement. All these data points can be removed.

6. Heals and Boosts:

- Both heals and boosts are used by players to restore lost health during a match.

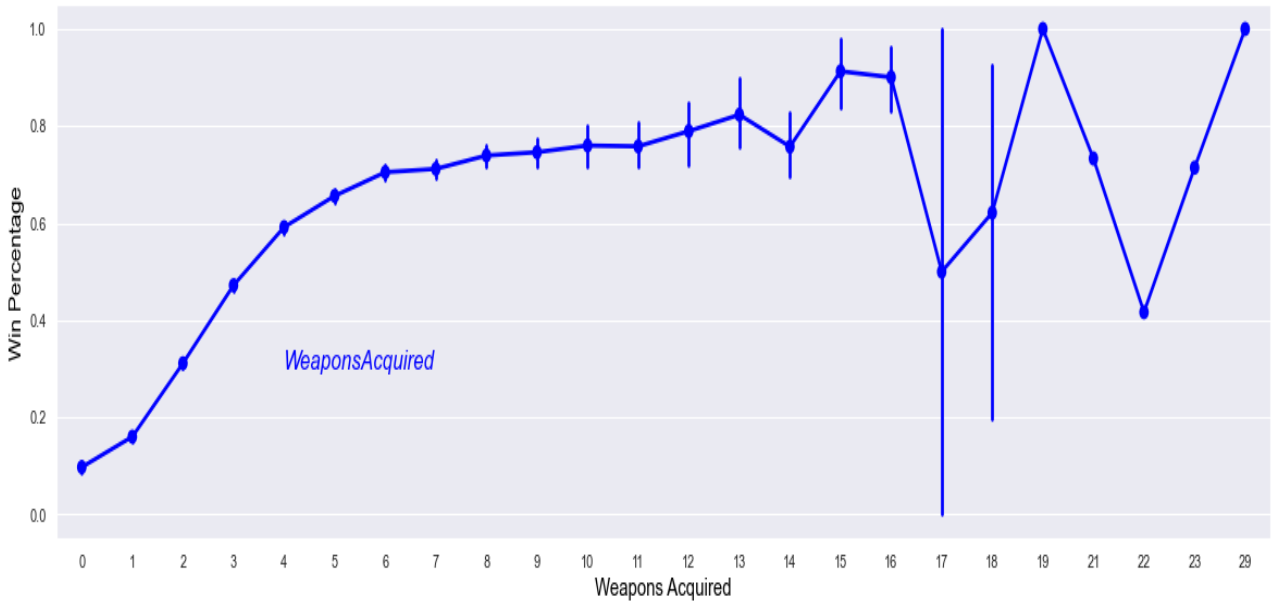


Above pic. depicts the relationship between winPercentage with heals and boosts respectively.

- We have also dropped few rows based on the quantile range of 0.9999 with some extra tolerance, treating them as outliers.
- And we have created a new feature named recover by adding heals and boosts.

7. WeaponsAcquired:

- The below pointplot shows relationship between WinPercentage and WeaponsAcquired.
- The same quantile of 0.9999 is used to remove outliers.



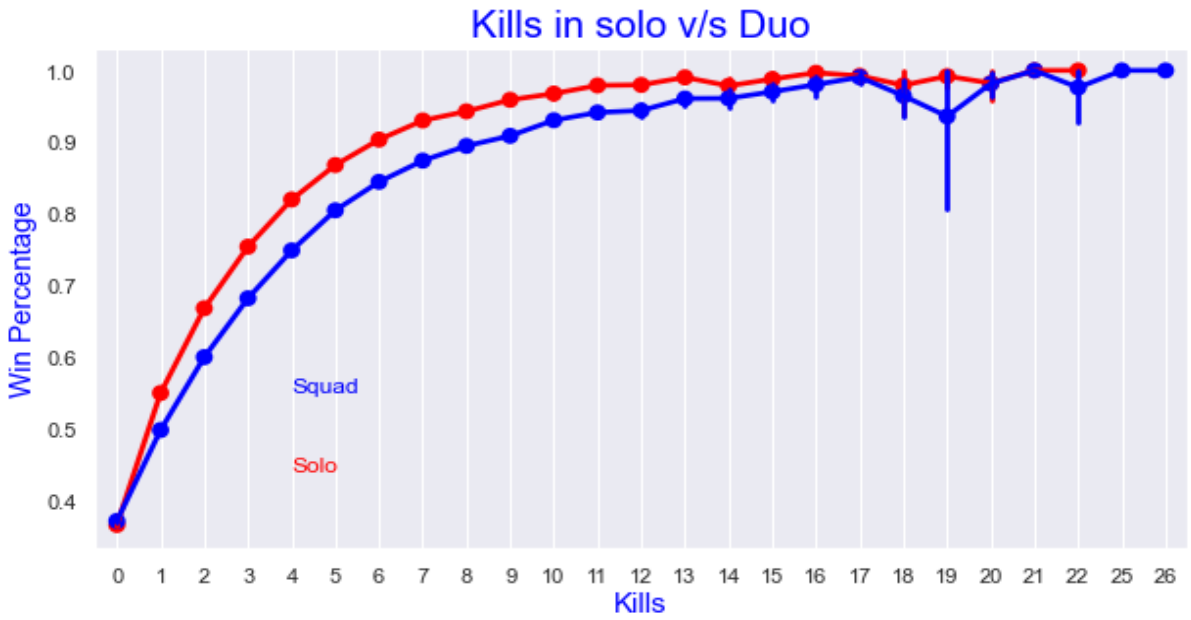
Above pic. depicts the effect of WeaponsAcquired by players on winPercentage.

8. MatchType:

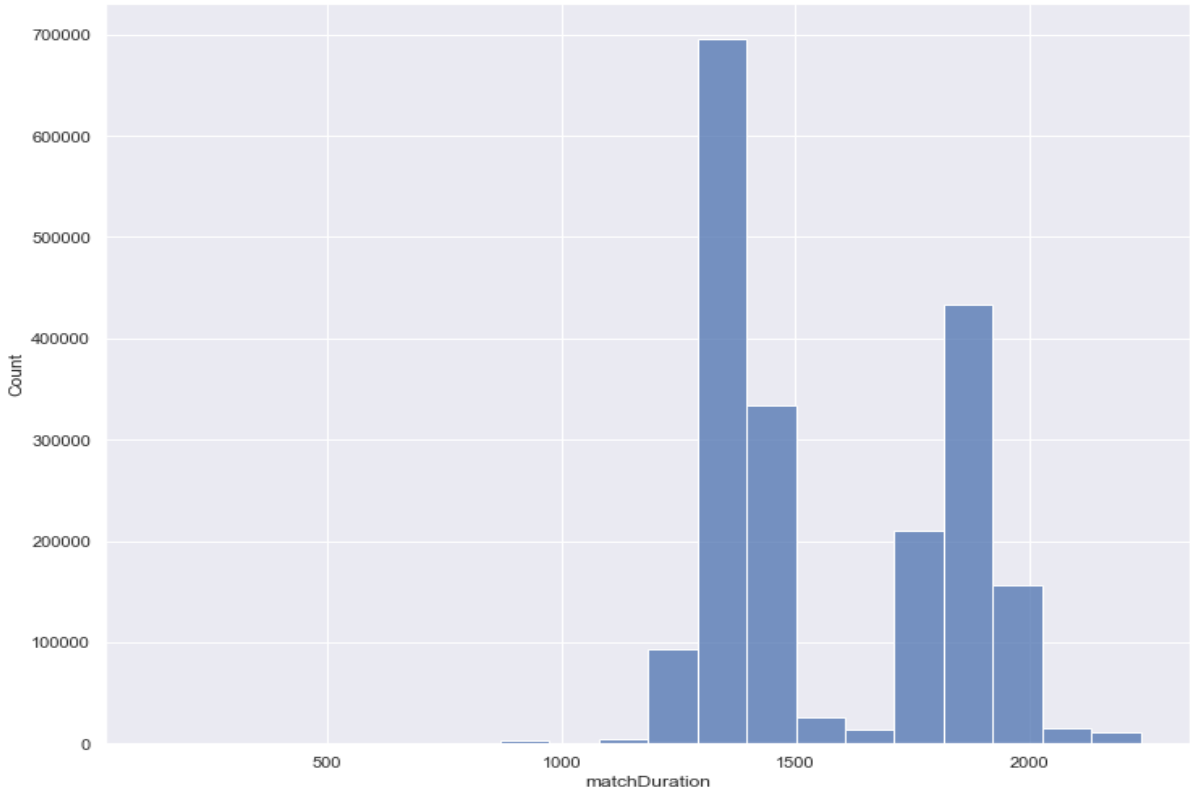
```
solo = ['solo','solo-fpp','normal-solo-fpp','normal-solo']
duo = ['duo','duo-fpp','normal-duo-fpp','normal-duo']
squad = ['squad','squad-fpp','normal-squad-fpp','normal-squad']
other = ['crashfpp','flaretpp','crashtpp','flarefpp']
db['teamSize'] = db.groupby('groupId')['groupId'].transform('count')
```

- Different matchTypes are categorized as shown above to get a clear picture of the matches and players performance.
- This is because we noticed that there are multiple type of matches and the techniques used by players depend on the matchType , like in solo match a player will have more kills compared to a squad match.
- Also in a squad match a player who doesn't play much but wins the match because of the other players in match needs to be handled as well.

Below pic. depicts the affect of kills on winPercentage for solo and squad matchtypes.



9. MatchDuration:



- We can see 2 distinct bell curves in the graph which are nothing but for solo and squad matches.

Handling Groups:

- Here we've grouped entire data based on matchId and groupId. Since, the matches contains groups having players in squad and players in solo matches. So, we're grouped by the individual groups in a match and then handle the problem as discussed above of a player not playing but ends up winning because of good teammates.
- And also each column's aggregations are taken based on the above grouping done.

Information Gain:

```
maxPlace                2.508985
numGroups               1.204068
maxkillPlace            1.116744
summatch_walkPerc       0.656463
sumtotalDistance        0.604835
sumweaponacq_walkPerc   0.470096
sumrecover              0.365554
sumweaponsAcquired      0.320521
sumdamage_walkPrec      0.306770
sumkills_walkPerc       0.288710
teamSize                0.241076
sumDBNOs                0.232125
maxlongestKill          0.187590
sumkills                0.184508
sumdamageDealt          0.177762
maxkillStreaks          0.132056
assists                 0.109888
sumrevives              0.104622
sumheadshotKills        0.082029
meanrankPoints          0.074274
meanwinPoints           0.028242
meankillPoints          0.019848
matchDuration           0.009238
sumteamKills            0.008083
sumvehicleDestroys      0.007533
sumroadKills            0.000000
dtype: float64
```

Feature Engineering:

1. **totalDistance** = sum of the features 'rideDistance','walkDistance' and 'swimDistance'

2. **walkDis/matchDuration** = total walked distance a player during entire match duration.

3. **noKill/walkDis** = No. of kills made by a player per distance the player has walked.

4. **damageDealt/walkDis** = Total damage dealt by player per distance the player has walked.

5. **weaponAcquired/walkDis** = Weapons acquired by player per distance the player has walked.

6. **recover** = sum of 'heals' and 'boosts'.

7. **matchType** = convert to solo,duo and squad and then one hot encode it .

8. **teamSize** = number of players in same group.

9. **groupBy for team** = combining all the data points for the players in same group and then giving prediction for them as a team.

10. **playersInMatch** = No. of Players in that particular match.

Non Tree Based Models Benchmarkings:

1. Linear Regression:

```
LinearRegression()
        Training time: 1.3155s
        Prediction time: 0.0246s
        Mean absolute error: 0.10727304308771986
        Mean squared error: 0.01962831902867452
```

2. Ridge Regression:

```
model = Ridge()
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
param = {
    'alpha': [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
}
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
search = GridSearchCV(model, param, cv=cv, verbose=3)
res = search.fit(lrX_train, lrY_train)
```

Few Recordings:

Alpha Value	Score
1e-5	0.783
1e-4	0.783
1e-3	0.783
1e-2	0.782
1e-1	0.7815
1	0.781
10	0.782
100	0.779

Best scores obtained:

```
Best Score: 0.7815678905167414
Best Hyperparameters: {'alpha': 0.1}

Ridge(alpha=0.01)
        Training time: 0.3993s
        Prediction time: 0.0212s
        Mean absolute error: 0.10727297171859171
        Mean squared error: 0.01962824132056379
```

3. Lasso Regression:

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import GridSearchCV
```

```
model = Lasso()
cv = RepeatedKFold(n_splits=5, n_repeats=3, random_state=1)
param = {
    'alpha': [1e-5, 1e-4, 1e-3, 1e-2],
    'max_iter':[800,1500,2000]
}
```

Few Recordings:

Alpha Value	Max Iterations	Score
1e-5	800	0.782
	1500	0.781
	2000	0.783
1e-4	800	0.776
	1500	0.777
	2000	0.776
1e-3	800	0.739
	1500	0.740
	2000	0.739
1e-2	800	0.584
	1500	0.585
	2000	0.587

Best scores obtained:

```
Best Score: 0.781320363147718
Best Hyperparameters: {'alpha': 1e-05, 'max_iter': 1500}

Lasso(alpha=1e-05, max_iter=2000)
        Training time: 30.7253s
        Prediction time: 0.0442s
        Mean absolute error: 0.10733070486943945
        Mean squared error: 0.01962754288294771
```

4. SVR:

```
model = LinearSVR()
cv = RepeatedKFold(n_splits=5, n_repeats=2, random_state=1)
param = {
    'C': [1e-2, 1e-1, 1],
    'max_iter':[1000,2000]
}
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
search = GridSearchCV(model, param, cv=cv, verbose=3)
res = search.fit(lrX_train, lrY_train)
```

Few Recordings:

C Value	Max_iter	Score
1e-2	1000	0.771
	2000	0.772
1e-1	1000	0.773
	2000	0.771
1	1000	0.771
	2000	0.770

Best scores obtained:

```
Best Score: 0.7725749656216911
Best Hyperparameters: {'C': 1, 'max_iter': 2000}

LinearSVR(C=0.01, max_iter=2000)
        Training time: 3.5880s
        Prediction time: 0.0169s
        Mean absolute error: 0.10528084375005818
        Mean squared error: 0.020449941915131727
```

Tree Based Models Benchmarkings:

1. XGBoostRegressor:

```
from sklearn.model_selection import train_test_split
x_t,x_test,y_t,y_test=train_test_split(x_train,y_train,test_size=0.2,random_state=42)
```

```
lr=[0.3,0.6,0.9] #learning_rate
md=[8,10,12] #max_depth
col=[0.5,0.7,0.9] #colsample_bytree
nest=[750,1500,2000] #n_estimators
```

Few Recordings:

Learning Rate	Max Depth	Colsample bytree	N estimators	Mean Squared Error
0.9	12	0.9	2000	0.0054%
0.6	10	0.5	750	0.0052%
0.3	8	0.7	1500	0.0049%
0.3	12	0.5	750	0.0050%
0.6	10	0.9	1500	0.0051%

Best scores obtained:

```
Mean Absolute Error: 0.0527%
Mean Squared Error: 0.0054%
79.)----------------------------------------------------------
learning_rate=0.9,max_depth=12,colsample_bytree=0.9,n_estimators=750
Test Result:
Mean Absolute Error: 0.0527%
Mean Squared Error: 0.0054%
80.)----------------------------------------------------------
learning_rate=0.9,max_depth=12,colsample_bytree=0.9,n_estimators=1500
Test Result:
Mean Absolute Error: 0.0529%
Mean Squared Error: 0.0054%
81.)----------------------------------------------------------
learning_rate=0.9,max_depth=12,colsample_bytree=0.9,n_estimators=2000

Test Result:
Mean Absolute Error: 0.0529%
Mean Squared Error: 0.0054%
best parameters are: {'learning_rate': 0.3, 'max_depth': 8, 'colsample_bytree': 0.7, 'n_estimators': 1500}
```

```
best score: 0.0049173007113347674
```

2. LightGBM:

```
from lightgbm import LGBMRegressor
lr=[0.03,0.05,0.1] #learning_rate
nl=[10,25,50,75] #num_leaves
col=[0.4,0.7,0.9] #colsample_bytree
nest=[750,1500] #n_estimators
```

Few Recordings:

Learning Rate	Num leaves	Colsample bytree	N estimators	Mean Squared Error
0.03	25	0.4	1500	0.0051%
0.1	25	0.4	750	0.0050%
0.05	75	0.4	1500	0.0049%
0.03	75	0.9	750	0.0050%
0.05	10	0.7	1500	0.0053%

Best scores obtained:

```
70.)------------------------------------------------------------
learning_rate=0.1,num_leaves=75,colsample_bytree=0.7,n_estimators=1500
[LightGBM] [Warning] num_threads is set=4, n_jobs=-1 will be ignored. Current value: num_threads=4
Test Result:
Mean Absolute Error: 0.0506%
Mean Squared Error: 0.0049%
71.)------------------------------------------------------------
learning_rate=0.1,num_leaves=75,colsample_bytree=0.9,n_estimators=750
[LightGBM] [Warning] num_threads is set=4, n_jobs=-1 will be ignored. Current value: num_threads=4
Test Result:
Mean Absolute Error: 0.0507%
Mean Squared Error: 0.0049%
72.)------------------------------------------------------------
learning_rate=0.1,num_leaves=75,colsample_bytree=0.9,n_estimators=1500
[LightGBM] [Warning] num_threads is set=4, n_jobs=-1 will be ignored. Current value: num_threads=4
Test Result:
Mean Absolute Error: 0.0506%
Mean Squared Error: 0.0049%
best parameters are: {'learning_rate': 0.05, 'num_leaves': 75, 'colsample_bytree': 0.4, 'n_estimators': 1500}
```

```
best score: 0.004907050535820301
```