

Init

Type *Markdown* and LaTeX: α^2

Team : Minus One

members :

Narasimhan N(MT2022062)

Sreenidhi K R (MT2022115)

In []:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import math
5 import seaborn as sns
6 from sklearn.feature_selection import mutual_info_regression
7 from sklearn.model_selection import train_test_split
8 from lightgbm import LGBMRegressor
```

In []:

```
1 dev_mode = 0
```

In []:

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Mounted at /content/drive

In []:

```
1 # traindf = pd.read_csv('/Users/sreenidhikr/Development/pubg-version-3/train_up.csv')
2 traindf = pd.read_csv('/content/drive/MyDrive/pubg/train_up.csv')
3
4 if(dev_mode):
5     df = traindf.head(2100000)
6     testdf = traindf.tail(1012876)
7
8 else :
9     df = traindf.copy()
10    # testdf = pd.read_csv('/Users/sreenidhikr/Development/pubg-version-3/test_up.csv')
11    testdf = pd.read_csv('/content/drive/MyDrive/pubg/test_up.csv')
12
13
```

In []:

```
1 # To Expand columns
2 pd.set_option('display.max_columns', None)
3 pd.set_option('display.expand_frame_repr', False)
4 pd.set_option('max_colwidth', -1)
5
6 #ignore warnings
7 pd.options.mode.chained_assignment = None
8
9 pd.set_option('display.max_columns', None)
10 pd.set_option('display.max_rows', None)
```

<ipython-input-20-a86a97bf760e>:4: FutureWarning: Passing a negative integer is deprecated in version 1.0 and will not be supported in future version. Instead, use None to not limit the column width.
pd.set_option('max_colwidth', -1)

In []:

```
1 print(traindf.shape)
2 print(df.shape)
3 print(testdf.shape)
```

```
(3112876, 29)
(3112876, 29)
(1334090, 28)
```

Presetting the datatypes Python automatically reads the data type, which causes a lot of memory waste. So if we know in advance the memory we will set up, we can use it much more effectively.

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> (<https://www.kaggle.com/gemartin/load-data-reduce-memory-usage>)

In []:

```
1 # def reduceMemory(df):
2 #     import numpy as np
3
4 #     for col in df.columns:
5 #         col_type = df[col].dtype
6
7 #         if col_type != object:
8 #             c_min = df[col].min()
9 #             c_max = df[col].max()
10 #             if str(col_type)[:3] == 'int':
11 #                 if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
12 #                     df[col] = df[col].astype(np.int8)
13 #                 elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
14 #                     df[col] = df[col].astype(np.int16)
15 #                 elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
16 #                     df[col] = df[col].astype(np.int32)
17 #                 elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
18 #                     df[col] = df[col].astype(np.int64)
19 #             else:
20 #                 if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
21 #                     df[col] = df[col].astype(np.float16)
22 #                 elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
23 #                     df[col] = df[col].astype(np.float32)
24 #                 else:
25 #                     df[col] = df[col].astype(np.float64)
26
27 #     return df
28 # df = reduceMemory(df)
```

In []:

```
1 #You can run this oneliner which will build and compile LightGBM with GPU enabled in colab:
2 #! cd LightGBM && rm -rf build && mkdir build && cd build && cmake -DUSE_GPU=1 ../../LightGBM && make -j4 && cd ../python-package
```

```
-- The C compiler identification is GNU 7.5.0
-- The CXX compiler identification is GNU 7.5.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenMP_C: -fopenmp (found version "4.5")
-- Found OpenMP_CXX: -fopenmp (found version "4.5")
-- Found OpenMP: TRUE (found version "4.5")
-- Looking for CL_VERSION_2_2
-- Looking for CL_VERSION_2_2 - found
-- Found OpenCL: /usr/lib/x86_64-linux-gnu/libOpenCL.so (found version "2.2")
-- OpenCL include directory: /usr/include
-- Found Boost: /usr/include (found suitable version "1.65.1", minimum required is "1.56.0") found components: files
```

In []:

```

1 # After running
2 #! git clone --recursive https://github.com/Microsoft/LightGBM

```

```

Cloning into 'LightGBM'...
remote: Enumerating objects: 28146, done.
remote: Counting objects: 100% (184/184), done.
remote: Compressing objects: 100% (111/111), done.
remote: Total 28146 (delta 79), reused 159 (delta 70), pack-reused 27962
Receiving objects: 100% (28146/28146), 19.92 MiB | 11.70 MiB/s, done.
Resolving deltas: 100% (20793/20793), done.
Submodule 'include/boost/compute' (https://github.com/boostorg/compute) registered for path 'external_libs/compute'
Submodule 'eigen' (https://gitlab.com/libeigen/eigen.git) registered for path 'external_libs/eigen'
Submodule 'external_libs/fast_double_parser' (https://github.com/lemire/fast_double_parser.git) registered for path 'external_libs/fast_double_parser'
Submodule 'external_libs/fmt' (https://github.com/fmtlib/fmt.git) registered for path 'external_libs/fmt'
Cloning into '/content/LightGBM/external_libs/compute'...
remote: Enumerating objects: 21733, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 21733 (delta 1), reused 3 (delta 1), pack-reused 21728
Receiving objects: 100% (21733/21733), 8.51 MiB | 14.41 MiB/s, done.
Resolving deltas: 100% (17567/17567), done.
Cloning into '/content/LightGBM/external_libs/eigen'...
remote: Enumerating objects: 117460, done.
remote: Counting objects: 100% (547/547), done.
remote: Compressing objects: 100% (189/189), done.
remote: Total 117460 (delta 358), reused 543 (delta 355), pack-reused 116913
Receiving objects: 100% (117460/117460), 102.96 MiB | 24.38 MiB/s, done.
Resolving deltas: 100% (96939/96939), done.
Cloning into '/content/LightGBM/external_libs/fast_double_parser'...
remote: Enumerating objects: 769, done.
remote: Counting objects: 100% (168/168), done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 769 (delta 119), reused 122 (delta 99), pack-reused 601
Receiving objects: 100% (769/769), 830.75 KiB | 1.85 MiB/s, done.
Resolving deltas: 100% (390/390), done.
Cloning into '/content/LightGBM/external_libs/fmt'...
remote: Enumerating objects: 30960, done.
remote: Counting objects: 100% (152/152), done.
remote: Compressing objects: 100% (90/90), done.
remote: Total 30960 (delta 76), reused 115 (delta 49), pack-reused 30808
Receiving objects: 100% (30960/30960), 13.72 MiB | 17.47 MiB/s, done.
Resolving deltas: 100% (20945/20945), done.
Submodule path 'external_libs/compute': checked out '36350b7de849300bd3d72a05d8bf890ca405a014'
Submodule path 'external_libs/eigen': checked out '3147391d946bb4b6c68edd901f2add6ac1f31f8c'
Submodule path 'external_libs/fast_double_parser': checked out 'ace60646c02dc54c57f19d644e49a61e7e7758ec'
Submodule 'benchmark/dependencies/abseil-cpp' (https://github.com/abseil/abseil-cpp.git) registered for path 'external_libs/fast_double_parser/benchmarks/dependencies/abseil-cpp'
Submodule 'benchmark/dependencies/double-conversion' (https://github.com/google/double-conversion.git) registered for path 'external_libs/fast_double_parser/benchmarks/dependencies/double-conversion'
Cloning into '/content/LightGBM/external_libs/fast_double_parser/benchmarks/dependencies/abseil-cpp'...
remote: Enumerating objects: 19661, done.
remote: Counting objects: 100% (1941/1941), done.
remote: Compressing objects: 100% (640/640), done.
remote: Total 19661 (delta 1342), reused 1359 (delta 1298), pack-reused 17720
Receiving objects: 100% (19661/19661), 11.92 MiB | 16.59 MiB/s, done.
Resolving deltas: 100% (15400/15400), done.
Cloning into '/content/LightGBM/external_libs/fast_double_parser/benchmarks/dependencies/double-conversion'...
remote: Enumerating objects: 1352, done.
remote: Counting objects: 100% (196/196), done.
remote: Compressing objects: 100% (103/103), done.
remote: Total 1352 (delta 108), reused 156 (delta 86), pack-reused 1156
Receiving objects: 100% (1352/1352), 7.15 MiB | 20.74 MiB/s, done.
Resolving deltas: 100% (880/880), done.
Submodule path 'external_libs/fast_double_parser/benchmarks/dependencies/abseil-cpp': checked out 'd936052d32a5b7ca08b0199a6724724aea432309'
Submodule path 'external_libs/fast_double_parser/benchmarks/dependencies/double-conversion': checked out 'f4cb2384efa55dee0e6652f8674b05763441ab09'
Submodule path 'external_libs/fmt': checked out 'b6f4ceaed0a0a24ccf575fab6c56dd50ccf6f1a9'

```

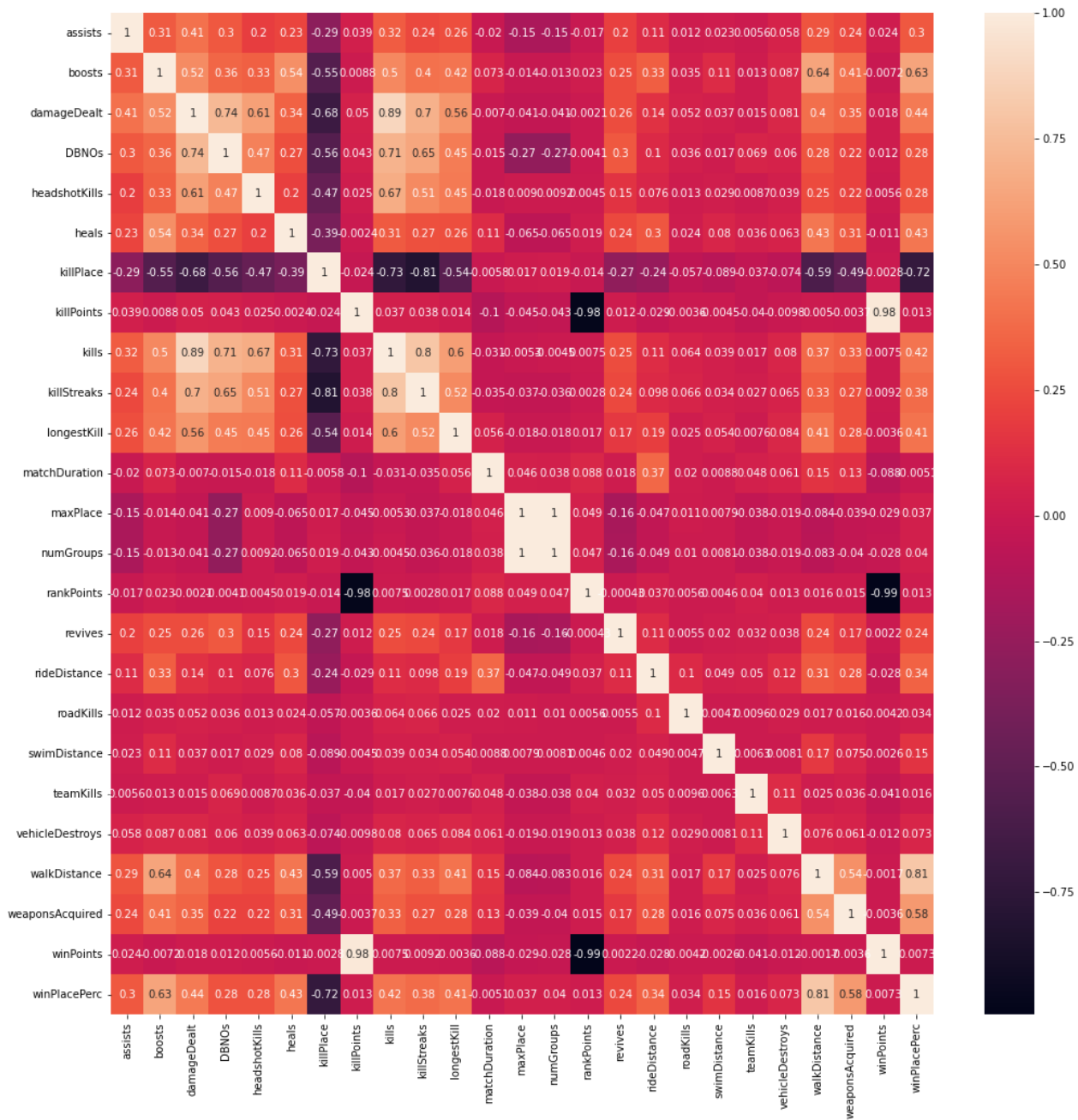
Initial Observations

In []:

```
1 plt.figure(figsize=(17,17))
2 sns.heatmap(df.corr(), annot = True)
```

Out[9]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7eb5042220>



In []:

```
1 df.corr()['winPlacePerc'].sort_values()
```

Out[10]:

```
killPlace      -0.719009
matchDuration  -0.005130
winPoints       0.007265
killPoints      0.013171
rankPoints      0.013321
teamKills       0.015535
roadKills       0.034090
maxPlace        0.037429
numGroups       0.039714
vehicleDestroys 0.073495
swimDistance    0.149715
revives         0.241043
headshotKills  0.277494
DBNOs           0.280254
assists         0.299125
rideDistance    0.342521
killStreaks     0.377600
longestKill     0.409985
kills           0.419801
heals           0.427901
damageDealt     0.440601
weaponsAcquired 0.582801
boosts          0.634121
walkDistance    0.810920
winPlacePerc    1.000000
Name: winPlacePerc, dtype: float64
```

Mutual information (MI) [1] between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.

based on knn

In []:

```
1 samp = df.sample(300000)
2 samp.dropna(inplace = True)
3 mutual_info = mutual_info_regression(samp.drop(columns = ['winPlacePerc', 'matchType', 'matchId', 'Id', 'groupId'], axis=1), samp.winPla
4 mutual_info = pd.Series(mutual_info)
5 mutual_info.index = samp.drop(columns = ['winPlacePerc', 'matchType', 'matchId', 'Id', 'groupId'], axis=1).columns
6 mutual_info = mutual_info.sort_values(ascending=False)
7
8 print(mutual_info)
```

```
maxPlace      2.378594
numGroups     1.181999
killPlace     0.948081
walkDistance  0.733891
weaponsAcquired 0.326114
boosts        0.322689
heals         0.203367
damageDealt   0.162876
longestKill   0.144017
kills         0.128810
rideDistance  0.120961
DBNOs         0.112352
killStreaks   0.099580
assists       0.066635
rankPoints    0.061645
revives       0.053547
headshotKills 0.051120
matchDuration 0.044275
swimDistance  0.032904
winPoints     0.022494
killPoints    0.018551
vehicleDestroys 0.004675
teamKills     0.003511
roadKills     0.000430
dtype: float64
```

Preprocessing on Rows

In []:

```
1 df.dropna(inplace=True)
```

Possibility of Hacker:

- 1) Walk distance = 0 but kills are very high
- 2) Walk distance = 0 but weapons acquired are high
- 3) Walk distance = 0 but damageDealt are high
- 4) Walk distance = 0 but DBNO are high
- 5) Walk distance = 0 but assists are high
- 6) When Walk distance = 0 but Ride distance > 0
- 7) Walk distance = 0 and Heals/Boosts > 0 :
- 8) Very long kills
- 9) Very high Headshots

Think weather to drop the rows or set the win percentage of these to 0 ?????

In []:

```
1 drop_rows = list(df[(df['walkDistance']==0) & (
2     (df['kills']>0) | (df['weaponsAcquired']>0) | (df['damageDealt']>0) | (df['rideDistance']>0) | (df['DBNOs']>0) |
3     (df['assists']>0) | (df['heals']>0) | (df['boosts']>0))].index)
4
5 print("Number of Hackers where walk distance is = 0 but (kills , damanage , dealth ,...etc) > 0 is ",len(drop_rows))
6 df.drop(index=drop_rows,inplace=True)
```

Number of Hackers where walk distance is = 0 but (kills , damanage , dealth ,...etc) > 0 is 14681

In []:

```
1 drop_rows = list(df[(df['winPlacePerc']==0) & ((df['kills']>1) | (df['walkDistance']>1000) | (df['assists']>1))].index)
2 print("Number of Hackers winPlacePerc = 0 but (kills , assists) > 1",len(drop_rows))
3 df.drop(index=drop_rows,inplace=True)
```

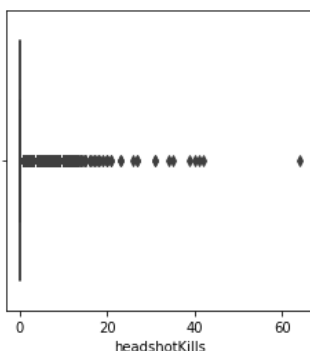
Number of Hackers winPlacePerc = 0 but (kills , assists) > 1 2522

In []:

```
1 fig = plt.figure(figsize=(4,4))
2 sns.boxplot(x=df['headshotKills'])
```

Out[72]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7edf57fbb0>



In []:

```
1 drop_rows = list(df[df['headshotKills']>20].index)
2 print("Number of Hackers where headshots are very high : ",len(drop_rows))
3 df.drop(index=drop_rows,inplace=True)
```

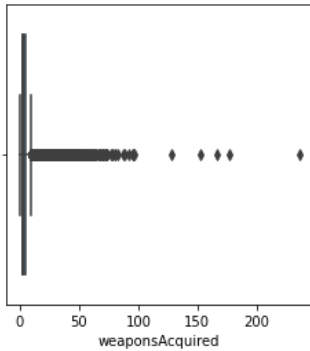
Number of Hackers where headshots are very high : 16

In []:

```
1 fig = plt.figure(figsize=(4,4))
2 sns.boxplot(x=df['weaponsAcquired'])
```

Out[74]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7edf5814c0>



In []:

```
1 drop_rows = list(df[df['weaponsAcquired']>50].index)
2 print("Number of Hackers where weaponsAcquired are very high : ",len(drop_rows))
3 df.drop(index=drop_rows,inplace=True)
```

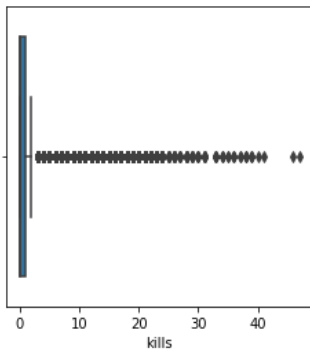
Number of Hackers where weaponsAcquired are very high : 100

In []:

```
1 fig = plt.figure(figsize=(4,4))
2 sns.boxplot(x=df['kills'])
```

Out[76]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7ecffe0220>



In []:

```
1 drop_rows = list(df[df['kills']>30].index)
2 print("Number of Hackers where kills are very high : ",len(drop_rows))
3 df.drop(index=drop_rows,inplace=True)
```

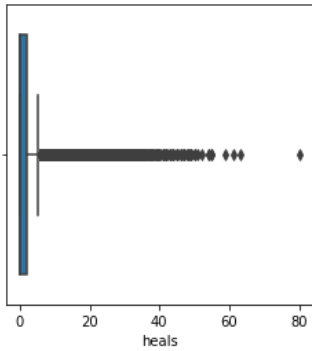
Number of Hackers where kills are very high : 32

In []:

```
1 fig = plt.figure(figsize=(4,4))
2 sns.boxplot(x=df['heals'])
```

Out[78]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7e7c2bdac0>



In []:

```
1 drop_rows = list(df[(df['heals']>35) | (df['boosts']>35)].index)
2 print("boosts / heals are very high : ",len(drop_rows))
3 df.drop(index=drop_rows,inplace=True)
```

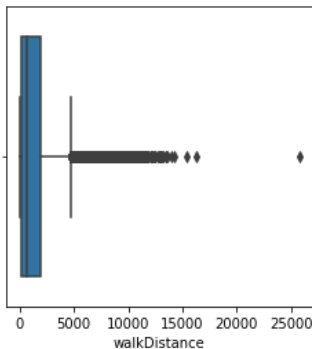
boosts / heals are very high : 195

In []:

```
1 fig = plt.figure(figsize=(4,4))
2 sns.boxplot(x=df['walkDistance'])
```

Out[80]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7ec627ab50>



In []:

```
1 drop_rows = list(df[df['walkDistance']>10000].index)
2 print("Walk distance very high : ",len(drop_rows))
3 df.drop(index=drop_rows,inplace=True)
```

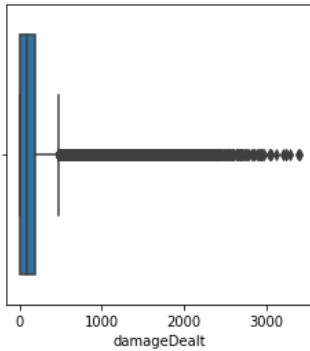
Walk distance very high : 155

In []:

```
1 fig = plt.figure(figsize=(4,4))
2 sns.boxplot(x=df['damageDealt'])
```

Out[82]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7e8f6a6e50>



In []:

```
1 drop_rows = list(df[df['damageDealt']>1500].index)
2 print("Walk distance very high : ",len(drop_rows))
3 df.drop(index=drop_rows,inplace=True)
```

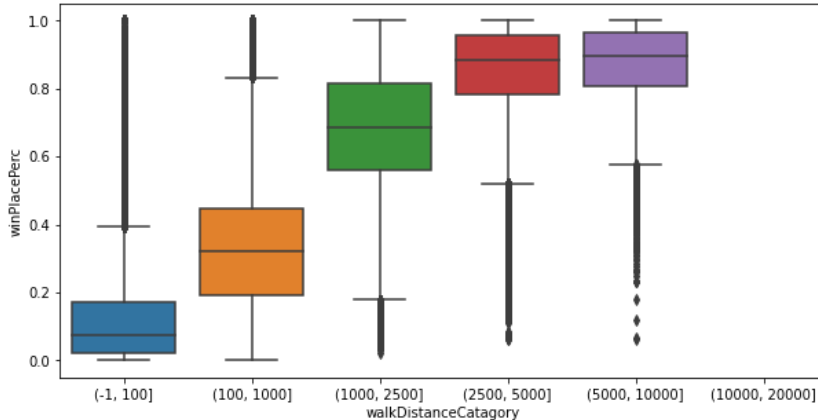
Walk distance very high : 1479

In []:

```
1 fig = plt.figure(figsize=(10,5))
2 df_temp = df.copy()
3 df_temp['walkDistanceCatagory'] = pd.cut(df_temp['walkDistance'],[-1,100,1000,2500,5000,10000,20000])
4 sns.boxplot(x=df_temp['walkDistanceCatagory'],y=df_temp['winPlacePerc'])
```

Out[84]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7edf8dfe20>



In []:

```
1 drop_index = list(df[(df['walkDistance']<=200) & (df['winPlacePerc']>=0.9)].index)
2 print("Dropping Rows where the walkDistance is < 100m but have winPlacePercentage >= 90% : ",len(drop_index))
3 df.drop(index=drop_index,inplace=True)
```

Dropping Rows where the walkDistance is < 100m but have winPlacePercentage >= 90% : 3892

In []:

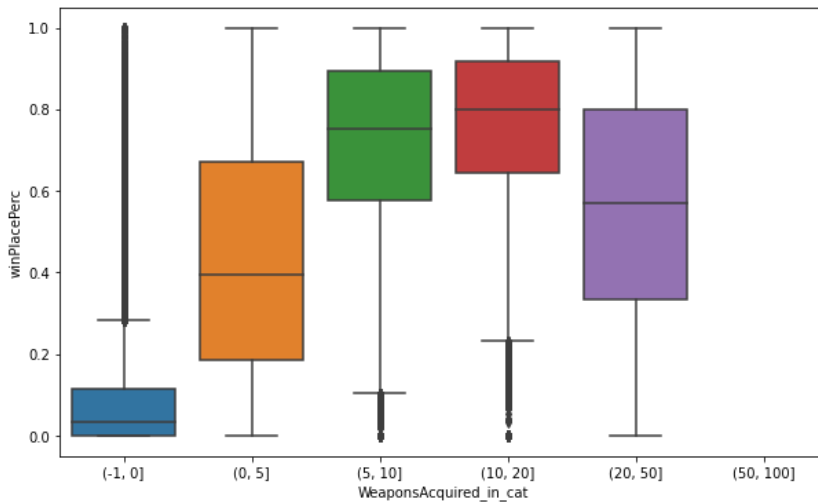
```

1 fig = plt.figure(figsize=(10,6))
2 temp_df = df.copy()
3 temp_df['WeaponsAcquired_in_cat'] = pd.cut(temp_df['weaponsAcquired'],[-1,0,5,10,20,50,100])
4 sns.boxplot(x=temp_df['WeaponsAcquired_in_cat'],y=df['winPlacePerc'])

```

Out[86]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7ec628b370>



In []:

```

1 drop_index = list(df[(df['weaponsAcquired']==0) & (df['winPlacePerc']>=0.5)].index)
2 print("Dropping Rows where the walkDistance is < 100m but have winPlacePercentage >= 50% : ",len(drop_index))
3 df.drop(index=drop_index,inplace=True)

```

Dropping Rows where the walkDistance is < 100m but have winPlacePercentage >= 50% : 5227

In []:

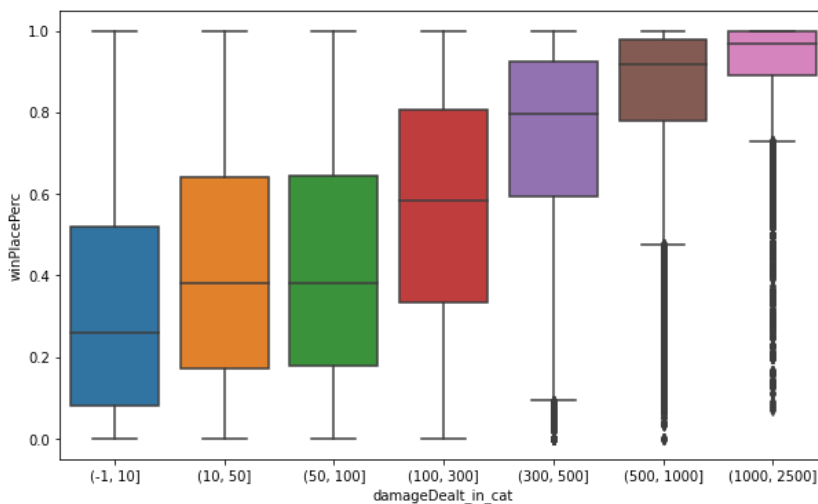
```

1 fig = plt.figure(figsize=(10,6))
2 temp_df = df.copy()
3 temp_df['damageDealt_in_cat'] = pd.cut(temp_df['damageDealt'],[-1,10,50,100,300,500,1000,2500])
4 sns.boxplot(x=temp_df['damageDealt_in_cat'],y=df['winPlacePerc'])

```

Out[88]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f7edf57f6a0>



In []:

```

1 drop_index = list(df[(df['damageDealt']>=500) & (df['winPlacePerc']<=0.3)].index)
2 print("Dropping Rows where the damageDealt is > 500 but have winPlacePercentage <= 30% : ",len(drop_index))
3 df.drop(index=drop_index,inplace=True)

```

Dropping Rows where the damageDealt is > 500 but have winPlacePercentage <= 30% : 1736

Preprocessing Columns and Feature Engineering

In []:

```

1 def preprocess(df,train=1):
2
3     df['walkDistancePerMatchDuration'] = df['walkDistance'] / df['matchDuration'] * 100;
4     df['weaponsAcquiredPerWalkDistance'] = df['weaponsAcquired'] / (df['walkDistance'] + 0.1) * 100;
5     df['killsPerWalkDistance'] = df['kills'] / (df['walkDistance'] + 0.1) * 100;
6     df['damageDeltPerWalkDistance'] = df['damageDealt'] / (df['walkDistance'] + 0.1) * 100;
7     df['boostsPerWalkDistance'] = df['boosts'] / (df['walkDistance'] + 0.1) * 100;
8     df['killPlacePerMaxPlace'] = df['killPlace']/(df['maxPlace']+1) * 100;
9     df['killPlacePerNumGroups'] = df['killPlace'] / (df['numGroups'] + 1)* 100;
10
11     #Tried these new feature not including them since they were not useful
12     #df['healsandboosts'] = df['heals'] + df['boosts']
13     #test['totalDistance'] = test['rideDistance'] + test['walkDistance'] + test['swimDistance']
14
15
16     #deleting not so useful columns
17     drop_columns = ['matchId','rankPoints', 'winPoints','killPoints','teamKills','vehicleDestroys','roadKills','swimDistance']
18     if(train):
19         drop_columns.append('Id')
20     df.drop(columns= drop_columns , inplace = True)
21     return df
22

```

In []:

```
1 df = preprocess(df)
```

In []:

```

1 samp = df.sample(200000)
2 samp.dropna(inplace = True)
3 mutual_info = mutual_info_regression(samp.drop(columns =['winPlacePerc','matchType','groupId'],axis=1),samp.winPlacePerc)
4 mutual_info = pd.Series(mutual_info)
5 mutual_info.index = samp.drop(columns =['winPlacePerc','matchType','groupId'],axis=1).columns
6 mutual_info = mutual_info.sort_values(ascending=False)
7
8 print(mutual_info)
9

```

```

killPlacePerMaxPlace      2.499106
maxPlace                  2.367130
killPlacePerNumGroups     1.493418
numGroups                 1.164521
killPlace                 0.946137
walkDistancePerMatchDuration 0.782216
walkDistance              0.740667
weaponsAcquiredPerWalkDistance 0.495817
weaponsAcquired           0.337588
boosts                    0.325896
boostsPerWalkDistance     0.314652
damageDeltPerWalkDistance 0.266982
killsPerWalkDistance      0.245184
heals                     0.204364
damageDealt               0.164618
longestKill               0.147006
kills                     0.133237
rideDistance              0.121718
DBNOs                     0.116329
killStreaks               0.103425
assists                   0.069702
revives                   0.053941
headshotKills             0.048042
matchDuration             0.035617
dtype: float64

```

In []:

```
1 df.corr()['winPlacePerc'].sort_values()
```

Out[92]:

killPlace	-0.725854
killPlacePerNumGroups	-0.609368
killPlacePerMaxPlace	-0.608327
weaponsAcquiredPerWalkDistance	-0.131156
damageDeltPerWalkDistance	-0.116204
killsPerWalkDistance	-0.060709
matchDuration	-0.004906
maxPlace	0.039723
numGroups	0.041075
boostsPerWalkDistance	0.141908
revives	0.242088
headshotKills	0.285420
DBNOs	0.286784
assists	0.305092
rideDistance	0.344488
killStreaks	0.381216
longestKill	0.413464
heals	0.432353
kills	0.434179
damageDealt	0.456147
weaponsAcquired	0.614288
boosts	0.637994
walkDistance	0.817985
walkDistancePerMatchDuration	0.831719
winPlacePerc	1.000000

Name: winPlacePerc, dtype: float64

Grouping

Intuition behind grouping teammates in squad and duo

Since PUBG is mostly a group game (except in solo) the ranking of a player depends on the ranking of the other players in the group

for example even if a player in a group gets out early, whereas his team-mates go on to win the match, the player who got out early still gets 1st place

so if we make predictions just by seeing a player, it will lead to many inaccuracies, therefore we group the data based on group id and collectively judge the groups performance

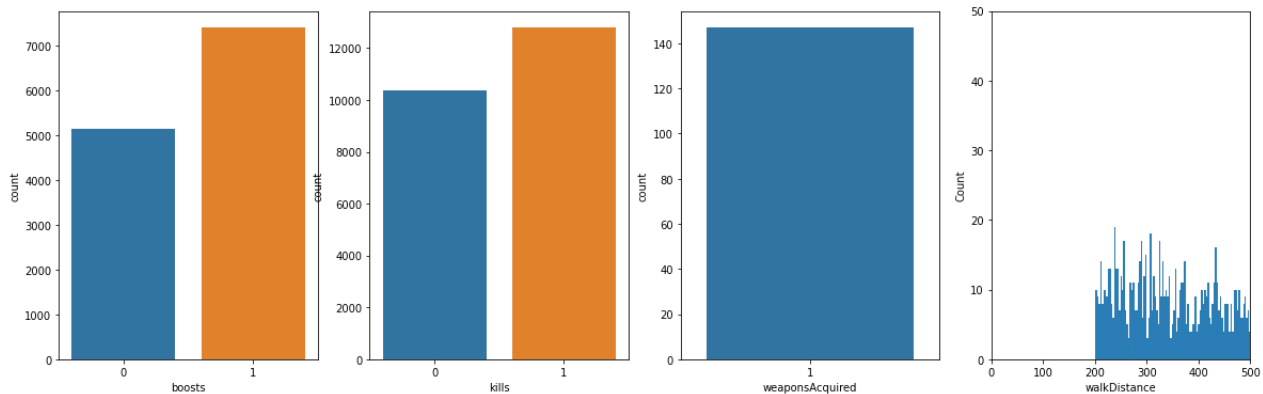
In []:

```

1 fig, ax = plt.subplots(1,4)
2 fig.set_figheight(6)
3 fig.set_figwidth(20)
4
5 fig.suptitle("highly correlated columns when win percent = 100",fontsize=15)
6 sns.countplot(data=df[(df['winPlacePerc'] == 1) & (df['boosts'] < 2)], x="boosts",ax=ax[0])
7
8 sns.countplot(data=df[(df['winPlacePerc'] == 1) & (df['kills'] < 2)], x="kills",ax=ax[1])
9
10 sns.countplot(data=df[(df['winPlacePerc'] == 1) & (df['weaponsAcquired'] < 2)], x="weaponsAcquired",ax=ax[2])
11
12
13 sns.histplot(data=df[df['winPlacePerc'] == 1], x="walkDistance",ax=ax[3],binwidth=3)
14 sns.histplot(data=df[df['winPlacePerc'] == 1], x="walkDistance",ax=ax[3],binwidth=3)
15 ax[3].set_xlim([0, 500])
16 ax[3].set_ylim([0, 50])
17 # plt.tight_layout()
18 fig.show()
19

```

highly correlated columns when win percent = 100



It can be seen that even with 0 boosts, 0 kills, 0 weapons, 0 walk distance the win percentage is 100, this implies that there team mates have performed well and won the match

In []:

```

1 group_by_columns = {
2     'winPlacePerc' : 'max',
3     'killStreaks' : 'max',
4     'longestKill' : 'max',
5     'walkDistancePerMatchDuration': ['sum','mean','min','max'],
6     'weaponsAcquiredPerWalkDistance': ['sum','mean','min','max'],
7     'killsPerWalkDistance': ['sum','mean','min','max'],
8     'damageDeltPerWalkDistance': ['sum','mean','min','max'],
9     'rideDistance': ['sum'],
10    'DBNOs': ['sum'],
11    'assists': ['sum'],
12    'numGroups' : 'first',
13    'maxPlace' : 'first',
14    'damageDealt': ['sum','mean','min','max'],
15    'revives' : ['sum'],
16    'headshotKills' : ['sum'],
17    'kills' : ['sum','mean','min','max'],
18    'weaponsAcquired': ['sum','mean','min','max'],
19    'boosts': ['sum','mean','min','max'],
20    'heals': ['sum','mean','min','max'],
21    'matchDuration': 'first',
22    'walkDistance' : ['sum','mean','min','max'],
23    'killPlace' : ['sum','mean','min','max'],
24    'killPlacePerMaxPlace': ['sum','mean','min','max'],
25    'killPlacePerNumGroups': ['sum','mean','min','max'],
26    }
27 group_by_columns_test = group_by_columns.copy();
28 group_by_columns_test['Id'] = 'first'
29 if(not dev_mode):
30     group_by_columns_test.pop('winPlacePerc')

```

In []:

1

In []:

```
1 squad_df_ug = df[(df['matchType'] == 'squad-fpp') | (df['matchType'] == 'squad') | (df['matchType'] == 'normal-squad-fpp') | (df['matc
2 duo_df_ug = df[(df['matchType'] == 'duo-fpp') | (df['matchType'] == 'duo') | (df['matchType'] == 'normal-duo-fpp') | (df['matchType'] == 'n
3 solo_df_ug = df[(df['matchType'] == 'solo-fpp') | (df['matchType'] == 'solo') | (df['matchType'] == 'normal-solo') | (df['matchType'] == 'n
```

In []:

```
1 squad_df = squad_df_ug.groupby('groupId').agg(group_by_columns)
2 duo_df = duo_df_ug.groupby('groupId').agg(group_by_columns)
3 solo_df = solo_df_ug.drop(columns = ['matchType', 'groupId'])
```

In []:

```
1 squad_df.rename(columns='_'.join, inplace=True)
2 squad_df.columns = squad_df.columns.map('_'.join)
3
4 duo_df.rename(columns='_'.join, inplace=True)
5 duo_df.columns = duo_df.columns.map('_'.join)
6
7 squad_df.columns = squad_df.columns.str.replace('_', '')
8 duo_df.columns = duo_df.columns.str.replace('_', '')
9
10 duo_df.rename(columns = {'winPlacePercmax': 'winPlacePerc'}, inplace = True)
11 squad_df.rename(columns = {'winPlacePercmax': 'winPlacePerc'}, inplace = True)
```

In []:

```
1 squad_df.sort_index(axis=1, inplace=True)
2 duo_df.sort_index(axis=1, inplace=True)
3 solo_df.sort_index(axis=1, inplace=True)
```

In []:

```

1 samp = squad_df.sample(200000)
2 samp.dropna(inplace = True)
3 mutual_info = mutual_info_regression(samp.drop(columns = ['winPlacePerc'],axis=1),samp.winPlacePerc)
4 mutual_info = pd.Series(mutual_info)
5 mutual_info.index = samp.drop(columns = ['winPlacePerc'],axis=1).columns
6 mutual_info = mutual_info.sort_values(ascending=False)
7
8 print(mutual_info)
9

```

```

killPlacePerMaxPlacemax          2.413984
killPlacePerMaxPlacemin          2.040084
maxPlacefirst                    1.687704
killPlacePerMaxPlacemean         1.555259
killPlacePerNumGroupsmax         1.483027
killPlacePerMaxPlacesum          1.413369
killPlacePerNumGroupsmin         1.099005
killPlacemax                     0.993555
killPlacePerNumGroupsmean        0.854838
walkDistancePerMatchDurationmean 0.825030
walkDistancePerMatchDurationmax  0.787785
walkDistancemean                 0.771347
walkDistancemax                  0.733713
walkDistancePerMatchDurationmin  0.669792
numGroupsfirst                   0.661922
killPlacePerNumGroupssum         0.649288
walkDistancemin                  0.638455
killPlacemean                    0.626557
killPlacemin                     0.621639
walkDistancePerMatchDurationsum  0.545394
walkDistancesum                  0.519575
weaponsAcquiredPerWalkDistancemin 0.477664
weaponsAcquiredPerWalkDistancemean 0.475180
weaponsAcquiredPerWalkDistancemax 0.464628
boostsmean                       0.446742
weaponsAcquiredmean              0.410128
boostsmax                        0.390856
weaponsAcquiredPerWalkDistancesum 0.387412
killPlacesum                     0.385758
boostssum                        0.384682
weaponsAcquiredmin               0.324794
damageDeltPerWalkDistancemax     0.297740
killsPerWalkDistancemax          0.291074
healsmean                        0.272916
weaponsAcquiredmax               0.269036
damageDeltPerWalkDistancesum     0.265548
killsPerWalkDistancesum          0.252035
damageDeltPerWalkDistancemean    0.244232
boostsmin                        0.237501
healssum                        0.228891
healsmax                         0.224528
killsPerWalkDistancemean         0.206545
damageDealtmean                  0.202677
killsmean                        0.201012
damageDeltPerWalkDistancemin     0.197262
longestKillmax                   0.187291
damageDealtmax                   0.180829
healsmin                         0.153295
killsPerWalkDistancemin          0.148789
killssum                         0.141990
killsmax                         0.141618
damageDealtsum                   0.138941
damageDealtmin                   0.129020
weaponsAcquiredsum               0.125025
rideDistancesum                  0.117847
assistssum                       0.095838
killsmin                         0.083435
killStreaksmax                   0.078973
DBNOsum                          0.076352
revivessum                       0.073682
headshotKillssum                 0.063241
matchDurationfirst               0.000000
dtype: float64

```

In []:

```

1 samp = solo_df.sample(200000)
2 samp.dropna(inplace = True)
3 mutual_info = mutual_info_regression(samp.drop(columns = ['winPlacePerc'],axis=1),samp.winPlacePerc)
4 mutual_info = pd.Series(mutual_info)
5 mutual_info.index = samp.drop(columns = ['winPlacePerc'],axis=1).columns
6 mutual_info = mutual_info.sort_values(ascending=False)
7
8 print(mutual_info)

```

```

killPlacePerMaxPlace      2.972752
maxPlace                  2.201047
killPlacePerNumGroups     1.677426
killPlace                 1.296133
walkDistancePerMatchDuration 0.981246
walkDistance              0.910917
weaponsAcquiredPerWalkDistance 0.608609
numGroups                 0.556885
weaponsAcquired           0.421503
boosts                   0.360736
boostsPerWalkDistance     0.337862
killsPerWalkDistance      0.320804
damageDeltPerWalkDistance 0.319609
damageDealt               0.214065
kills                    0.196304
heals                    0.189696
longestKill               0.189211
killStreaks              0.129265
rideDistance              0.109174
headshotKills             0.086962
assists                   0.016790
DBNOs                    0.001607
matchDuration             0.000000
revives                   0.000000
dtype: float64

```


EDA

In []:

```

1 def plotLine(attribute):
2     f,ax = plt.subplots(figsize = (10,4))
3     sns.lineplot(x = attribute , y = 'winPlacePerc',data = solo_df,color='red',alpha=0.8,ci=None,label='solo');
4     sns.lineplot(x = (attribute+'min') , y = 'winPlacePerc',data = duo_df,color='blue',alpha=0.8,ci=None,label='duo');
5     sns.lineplot(x = (attribute+'min') , y = 'winPlacePerc',data = squad_df,color='green',alpha=0.8,ci=None,label='squad');
6     plt.xlabel(attribute);
7     plt.ylabel('Win Percentage');
8     plt.title(attribute,fontsize = 15,color = 'blue')
9     plt.grid()

```

 todo - bugfix

In []:

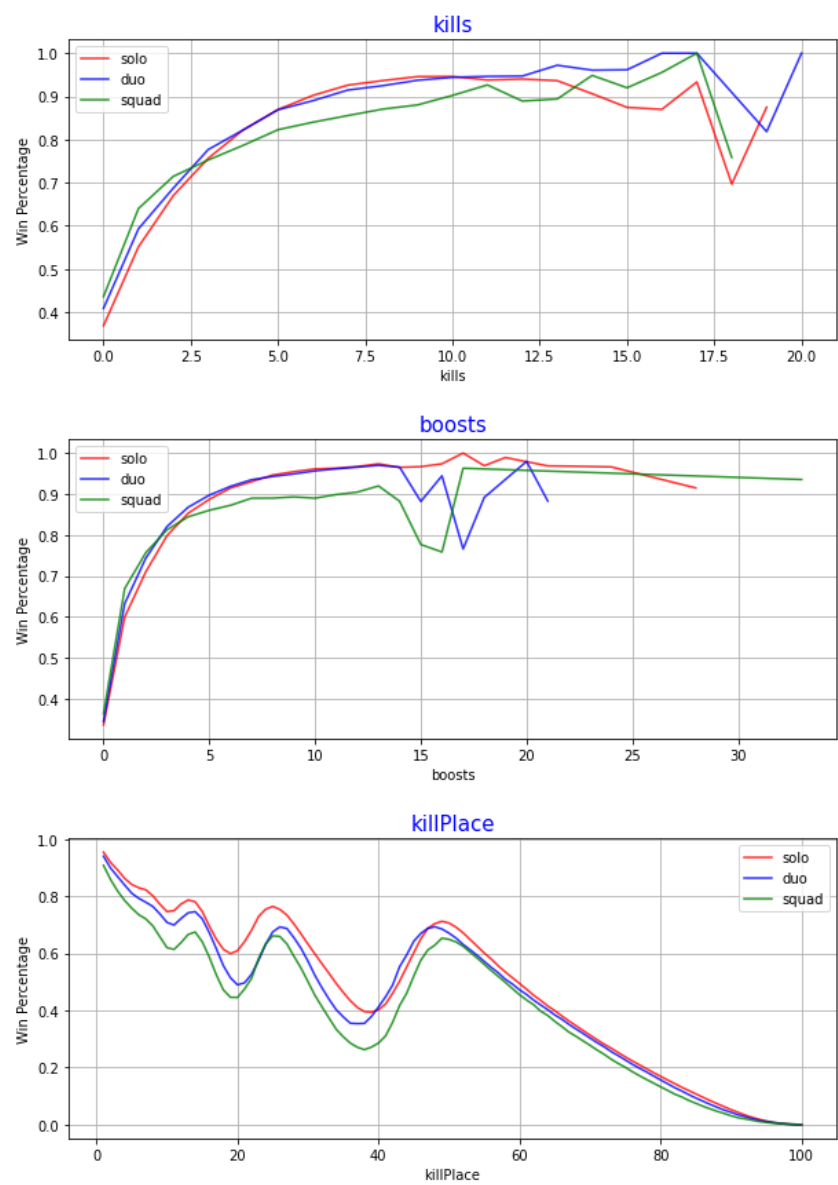
```

1 def plotBox(attribute,intervals):
2     fig, ax =plt.subplots(1,3)
3     fig.set_figheight(5)
4     fig.set_figwidth(15)
5     fig.suptitle(attribute,fontsize=15)
6
7     temp_df = solo_df.ug.copy()
8     temp_df[attribute] = pd.cut(temp_df[attribute],intervals)
9     sns.boxplot(x=temp_df[attribute],y=df['winPlacePerc'],ax=ax[2])
10
11     temp_df = duo_df.copy()
12     temp_df[attribute] = pd.cut(temp_df[attribute],intervals)
13     sns.boxplot(x=temp_df[attribute],y=df['winPlacePerc'],ax=ax[1])
14
15     temp_df = squad_df.ug.copy()
16     temp_df[attribute] = pd.cut(temp_df[attribute],intervals)
17     sns.boxplot(x=temp_df[attribute],y=df['winPlacePerc'],ax=ax[0])
18
19
20
21 plt.tight_layout()
22 fig.show()

```


In []:

```
1 line_plot_columns = ['kills', 'boosts', 'killPlace']
2 for c in line_plot_columns:
3     plotLine(c)
4
```



TRAINING

In []:

```
1
```

In []:

```
1 def get_model_lgbm(df):
2     y_train = pd.DataFrame(df, columns = ['winPlacePerc'])
3     X_train = df.drop(['winPlacePerc'], axis = 1)
4     y_train = y_train.fillna(0)
5     params = {"objective" : "regression", "metric" : "mae", 'n_estimators':2000,
6               "num_leaves" : 85, "learning_rate" : 0.05,
7               "bagging_fraction" : 0.7, "bagging_seed" : 0, "num_threads" : 4,
8               "colsample_bytree" : 0.8, 'device' : 'gpu',
9               }
10
11     model = LGBMRegressor(**params)
12     model.fit(
13         X_train, y_train
14     )
15     return model
```

In []:

```
1 import xgboost as xgb
```

In []:

```
1 def get_model_xg(df):
2     y_train = pd.DataFrame(df, columns = ['winPlacePerc'])
3     X_train = df.drop(['winPlacePerc'], axis = 1)
4     y_train = y_train.fillna(0)
5     model = xgb.XGBRegressor(tree_method='gpu_hist', objective='reg:squarederror',
6                              colsample_bytree = 0.6, learning_rate = 0.07, max_depth = 9,
7                              alpha = 10, n_estimators = 700)
8     model.fit(
9         X_train, y_train
10    )
11    return model
```

In []:

```
1 from sklearn.linear_model import LinearRegression
```

In []:

```
1 def get_model_lr(df):
2     y_train = pd.DataFrame(df, columns = ['winPlacePerc'])
3     X_train = df.drop(['winPlacePerc'], axis = 1)
4     y_train = y_train.fillna(0)
5     model = LinearRegression(n_jobs=4, normalize=True)
6     model.fit(X_train, y_train)
7     return model
```

In []:

```
1 from sklearn.svm import SVR
```

In []:

```
1 def get_model_svm(df, k):
2     y_train = pd.DataFrame(df, columns = ['winPlacePerc'])
3     X_train = df.drop(['winPlacePerc'], axis = 1)
4     y_train = y_train.fillna(0)
5     model = regressor = SVR(kernel=k)
6     model.fit(X_train, y_train)
7     return model
```

In []:

```
1 model_solo = get_model_xg(solo_df)
2 model_duo = get_model_xg(duo_df)
3 model_squad = get_model_xg(squad_df)
4
```

In []:

```
1 print('Feature importances:', model_squad.feature_importances_)
```

```
Feature importances: [0.00053961 0.00141083 0.00197113 0.00524637 0.00065432 0.00085186
0.00065256 0.0006027 0.00049115 0.00060464 0.00040042 0.00051437
0.00060862 0.00052966 0.00041061 0.00050358 0.00085942 0.00058049
0.00060988 0.03102492 0.00108974 0.00442001 0.00452598 0.06808681
0.01120003 0.00610733 0.00721758 0.04447981 0.01510926 0.00356595
0.00277125 0.00125418 0.00138706 0.00435267 0.00700686 0.00175779
0.00462859 0.00459271 0.0186839 0.00303203 0.00076405 0.0020919
0.00157242 0.00112445 0.00101201 0.0031137 0.2739069 0.38685542
0.00179703 0.00076741 0.01223019 0.04354714 0.00088498 0.0007335
0.00055847 0.00065439 0.00075062 0.00057232 0.00067441 0.00087236
0.00068089 0.00049889]
```

In []:

```
1
```

TESTING

In []:

```
1 testdf_grouped = testdf.copy()
2
3 testdf_grouped = preprocess(testdf_grouped,0)
4
5 squad_df_test = testdf_grouped[(testdf_grouped['matchType'] == 'squad-fpp') | (testdf_grouped['matchType'] == 'squad') | (testdf_gro
6 duo_df_test = testdf_grouped[(testdf_grouped['matchType'] == 'duo-fpp') | (testdf_grouped['matchType'] == 'duo') | (testdf_grouped['ma
7 solo_df_test = testdf_grouped[(testdf_grouped['matchType'] != 'squad-fpp') & (testdf_grouped['matchType'] != 'squad') & (testdf_groupe
8
9
10
11 squad_df_test_grouped = squad_df_test.groupby('groupId', as_index=False).agg(group_by_columns_test)
12 duo_df_test_grouped = duo_df_test.groupby('groupId', as_index=False).agg(group_by_columns_test)
13 solo_df_test_grouped = solo_df_test.drop(columns = ['matchType'])
14
15 #
16 sq_g = squad_df_test.drop(columns = ['matchType'])
17
18
19
20 squad_df_test.sort_index(axis=1, inplace=True)
21 duo_df_test.sort_index(axis=1, inplace=True)
22 solo_df_test.sort_index(axis=1, inplace=True)
23 squad_df_test_grouped.sort_index(axis=1, inplace=True)
24 duo_df_test_grouped.sort_index(axis=1, inplace=True)
25 solo_df_test_grouped.sort_index(axis=1, inplace=True)
26
27 #
28 sq_g.sort_index(axis=1, inplace=True)
29
30
31 squad_df_test_grouped.rename(columns='_'.join, inplace=True)
32 squad_df_test_grouped.columns = squad_df_test_grouped.columns.map('_'.join)
33
34 duo_df_test_grouped.rename(columns='_'.join, inplace=True)
35 duo_df_test_grouped.columns = duo_df_test_grouped.columns.map('_'.join)
36
37 squad_df_test_grouped.columns = squad_df_test_grouped.columns.str.replace('_', '')
38 duo_df_test_grouped.columns = duo_df_test_grouped.columns.str.replace('_', '')
39
40 duo_df_test_grouped.rename(columns = {'winPlacePercmax': 'winPlacePerc' , 'Idfirst': 'Id'}, inplace = True)
41 squad_df_test_grouped.rename(columns = {'winPlacePercmax': 'winPlacePerc', 'Idfirst': 'Id'}, inplace = True)
42
```

In []:

```
1 from sklearn.metrics import mean_absolute_error
2 import sklearn.metrics as metrics
```

In []:

```

1 def predict_model(odf ,df, model , s = 1):
2     id_gid = pd.DataFrame(odf,columns = ['Id','groupId'])
3     if(dev_mode):
4         y_test = pd.DataFrame(df,columns = ['winPlacePerc'])
5         xtest = df.drop(columns = ['groupId' , 'Id', 'winPlacePerc'])
6     else :
7         xtest = df.drop(columns = ['groupId', 'Id'])
8     xtest.sort_index(axis=1, inplace=True)
9     predictions = model.predict(xtest)
10    y_pred = pd.DataFrame(predictions, columns = ['winPlacePerc'])
11
12    for i in range(len(y_pred)):
13        if y_pred['winPlacePerc'].iloc[i] < 0 :
14            y_pred['winPlacePerc'].iloc[i] = 0
15        if y_pred['winPlacePerc'].iloc[i] > 1 :
16            y_pred['winPlacePerc'].iloc[i] = 1
17
18    if(dev_mode):
19        print('The mean absolute error of prediction is:', metrics.mean_absolute_error(y_test, y_pred))
20    if(s == 0):
21        odf_id = pd.DataFrame(odf,columns = ['Id'])
22        odf_id.reset_index(drop=True, inplace=True)
23        res = pd.concat([odf_id, y_pred], axis=1, join='inner' ,ignore_index=True, sort=False)
24        res.rename(columns={0: 'Id', 1: 'winPlacePerc'}, inplace=True)
25        return res
26
27    gid_target = pd.concat([df['groupId'], y_pred], axis=1, join='inner' ,ignore_index=True, sort=False)
28    gid_target.rename(columns={0: 'groupId', 1: 'winPlacePerc'}, inplace=True)
29    result = id_gid.merge(gid_target, how='inner', on='groupId')
30    result.drop(columns=['groupId'],inplace = True)
31    return result
32
33

```

In []:

```
1 result_solo = predict_model(solo_df_test_grouped ,solo_df_test_grouped, model_solo,0)
```

The mean absolute error of prediction is: 0.0423610555433123

In []:

```

1 duo_df_test_grouped.replace([np.inf, -np.inf], np.nan, inplace=True)
2 squad_df_test_grouped.replace([np.inf, -np.inf], np.nan, inplace=True)
3 duo_df_test_grouped.fillna(0,inplace=True)
4 squad_df_test_grouped.fillna(0,inplace=True)
5
6

```

In []:

```
1 result_duo = predict_model(duo_df_test ,duo_df_test_grouped, model_duo)
```

The mean absolute error of prediction is: 0.04693529689057275

In []:

```
1 result_squad = predict_model(squad_df_test,squad_df_test_grouped , model_squad)
```

The mean absolute error of prediction is: 0.06341065474921555

In []:

```
1 #0.6330
```

In []:

```

1 if(not dev_mode):
2     result = pd.concat([result_squad, result_duo, result_solo])
3     result.fillna(0.5,inplace = True)
4     result.to_csv('submissionV94.csv', header=True , index = False)
5
6

```

In []:

```
1
```

Ungrouped Benchmark

In []:

```
1 ungrouped_df = df.drop(columns = ['matchType', 'groupId'])
2 ungrouped_df.sort_index(axis=1, inplace=True)
```

In []:

```
1 model_ungrouped = get_model_lgbm(ungrouped_df)
```

```
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
[LightGBM] [Info] This is the GPU trainer!!
[LightGBM] [Info] Total Bins 3527
[LightGBM] [Info] Number of data points in the train set: 2079544, number of used features: 24
[LightGBM] [Info] Using GPU Device: Tesla T4, Vendor: NVIDIA Corporation
[LightGBM] [Info] Compiling OpenCL Kernel with 256 bins...
[LightGBM] [Info] GPU programs have been built
[LightGBM] [Info] Size of histogram bin entry: 8
[LightGBM] [Info] 19 dense feature groups (39.66 MB) transferred to GPU in 0.068726 secs. 1 sparse feature groups
[LightGBM] [Info] Start training from score 0.474267
```

In []:

```
1 test_ungrouped = testdf_grouped.drop(columns = ['matchType'])
2 test_ungrouped.sort_index(axis=1, inplace=True)
```

In []:

```
1 result_ungrouped = predict_model(test_ungrouped ,test_ungrouped, model_ungrouped,0)
```

```
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
The mean absolute error of prediction is: 0.05738620868328145
```

In []:

```
1 r1 = predict_model(sq_g ,sq_g, model_ungrouped,0)
```

```
[LightGBM] [Warning] bagging_fraction is set=0.7, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7
The mean absolute error of prediction is: 0.06657632586317018
```

Benchmarking

GRID SEARCH - LGBM

Result

```
gridParams = {
    'learning_rate'      : [0.1 , 0.05 , 0.75],
    'n_estimators '      : [ 1200, 1700,2000,2400],
    'bagging_fraction'   : [0.7 , 0.8],
    'feature_fraction'   : [0.8,0.7],
    'num_leaves'         : [60,85, 110,140],
    "colsample_bytree"    : [0.7,0.8],
    "objective" : ["regression"],
    "metric" : ["mae"],
    "num_threads" : [4],
    'device' : ['gpu'],
}
```

SOLO

```
Best parameters: {'bagging_fraction': 0.8, 'colsample_bytree': 0.8, 'device': 'gpu', 'feature_fraction': 0.7,
'learning_rate': 0.1, 'metric': 'mae', 'n_estimators ': 2000, 'num_leaves': 140, 'num_threads': 4, 'objective': 'regression'}
```

Accuracy: 0.96

In []:

```
1 from sklearn.model_selection import GridSearchCV
```

In []:

```

1 def find_best_hyperparameters(df,model):
2     # Grid parameters for using in Gridsearch while tuning
3     y_train = pd.DataFrame(df,columns = ['winPlacePerc'])
4     X_train = df.drop(['winPlacePerc'], axis = 1)
5     y_train = y_train.fillna(0)
6     gridParams = {
7         'learning_rate'      : [0.1 , 0.05 , 0.75],
8         'n_estimators '      : [ 1200, 1700,2000,2400],
9         'bagging_fraction'    : [0.7 , 0.8],
10        'feature_fraction'    : [0.8,0.7],
11        'num_leaves'          : [60,85, 110,140],
12        "colsample_bytree"     : [0.7,0.8],
13        "objective" : ["regression"],
14        "metric" : ["mae"],
15        "num_threads" : [4],
16        'device' : ['gpu'],
17    }
18    # Create the grid
19    grid = GridSearchCV(model,
20                        gridParams,
21                        verbose=1,
22                        cv=3)
23    # Run the grid
24    grid.fit(X_train, y_train)
25    print('Best parameters: %s' % grid.best_params_)
26    print('Accuracy: %.2f' % grid.best_score_)
27    return

```

In []:

```
1 model = LGBMRegressor()
```

In []:

```
1 find_best_hyperparameters(squad_df,model)
2 find_best_hyperparameters(solo_df,model)
```

In []:

```
1
```

Grid Search for XG Boost :

In []:

```

1 import xgboost as xgb
2 from sklearn.metrics import mean_absolute_error
3 from sklearn.metrics import r2_score
4 from sklearn.model_selection import RandomizedSearchCV
5 from sklearn.metrics import mean_absolute_error
6 from sklearn.metrics import fbeta_score, make_scorer
7 from xgboost.sklearn import XGBRegressor
8 from xgboost.sklearn import XGBRegressor
9 import datetime
10 from sklearn.model_selection import GridSearchCV

```

In []:

```

1 # Grid Search for Solo Data on XG Boost
2 xgb1 = XGBRegressor()
3 parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
4               'objective':['reg:linear'],
5               'learning_rate': [.03, 0.05, .07], #so called `eta` value
6               'max_depth': [5, 6, 7],
7               'min_child_weight': [4],
8               'silent': [1],
9               'subsample': [0.7],
10              'colsample_bytree': [0.7],
11              'n_estimators': [500],
12              'tree_method':['gpu_hist']}
13
14 xgb_grid = GridSearchCV(xgb1,
15                          parameters,
16                          cv = 2,
17                          n_jobs = 5,
18                          verbose=True)
19
20 xgb_grid.fit(solo_df.drop(columns='winPlacePerc',axis=1), solo_df['winPlacePerc'])
21
22 print(xgb_grid.best_score_)
23 print(xgb_grid.best_params_)

```

Fitting 2 folds for each of 9 candidates, totalling 18 fits

[01:30:13] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.

[01:30:13] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb oost-ci-windows/src/learner.cc:767:

Parameters: { "silent" } are not used.

0.9620177534505304

{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread': 4, 'objective': 'reg:linear', 'silent': 1, 'subsample': 0.7, 'tree_method': 'gpu_hist'}

In []:

```

1 # Grid Search for Squad Data on XG Boost
2 xgb1 = XGBRegressor()
3 parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
4               'objective':['reg:linear'],
5               'learning_rate': [.03, 0.05, .07], #so called `eta` value
6               'max_depth': [5, 6, 7],
7               'min_child_weight': [4],
8               'silent': [1],
9               'subsample': [0.7],
10              'colsample_bytree': [0.7],
11              'n_estimators': [500],
12              'tree_method':['gpu_hist']}
13
14 xgb_grid = GridSearchCV(xgb1,
15                          parameters,
16                          cv = 2,
17                          n_jobs = 5,
18                          verbose=True)
19
20 xgb_grid.fit(squad_df.drop(columns='winPlacePerc',axis=1), squad_df['winPlacePerc'])
21
22 print(xgb_grid.best_score_)
23 print(xgb_grid.best_params_)

```

Fitting 2 folds for each of 9 candidates, totalling 18 fits

[09:47:28] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.

[09:47:28] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb oost-ci-windows/src/learner.cc:767:

Parameters: { "silent" } are not used.

0.9315941745787972

{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread': 4, 'objective': 'reg:linear', 'silent': 1, 'subsample': 0.7, 'tree_method': 'gpu_hist'}

In []:

```

1 # Grid Search for Duo grouped Data on XG Boost
2 xgb1 = XGBRegressor()
3 parameters = {'nthread':[4], #when use hyperthread, xgboost may become slower
4               'objective':['reg:linear'],
5               'learning_rate': [.03, 0.05, .07], #so called `eta` value
6               'max_depth': [5, 6, 7],
7               'min_child_weight': [4],
8               'silent': [1],
9               'subsample': [0.7],
10              'colsample_bytree': [0.7],
11              'n_estimators': [500],
12              'tree_method':['gpu_hist']}
13
14 xgb_grid = GridSearchCV(xgb1,
15                         parameters,
16                         cv = 2,
17                         n_jobs = 5,
18                         verbose=True)
19
20 xgb_grid.fit(duo_df.drop(columns='winPlacePerc',axis=1), duo_df['winPlacePerc'])
21
22 print(xgb_grid.best_score_)
23 print(xgb_grid.best_params_)

```

Fitting 2 folds for each of 9 candidates, totalling 18 fits

[11:59:40] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.

[11:59:41] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb oost-ci-windows/src/learner.cc:767:

Parameters: { "silent" } are not used.

0.9546865985514996

{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread': 4, 'objective': 'reg:linear', 'silent': 1, 'subsample': 0.7, 'tree_method': 'gpu_hist'}

Benchmark Testing :

In []:

```
1 # Benchmarking with Different Parameters for Solo Grouped Data for XG Boost
2
3 df = solo_df
4 y_train = pd.DataFrame(df, columns = ['winPlacePerc'])
5 X_train = df.drop(['winPlacePerc'], axis = 1)
6 y_train = y_train.fillna(0)
7 bm_solo_xg = pd.DataFrame(columns=['Parameters', 'Error'])
8
9 for lr in [0.05, 0.04, 0.06]:
10     for md in [7, 9]:
11         for ne in [500, 600]:
12             params = {'colsample_bytree': 0.7, 'learning_rate': lr, 'max_depth': md, 'min_child_weight': 4, 'n_estimators': ne, '
13             model = xgb.XGBRegressor(**params)
14             model.fit(
15                 X_train, y_train
16             )
17             print("-----\n Model Parameters : ")
18             print(params)
19             predict_model(solo_df_test, solo_df_test_grouped, model)
20
```

```
[22:02:50] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.04277823164065137  
[22:02:58] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.042693481510916204  
[22:03:07] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.04253511252141473  
[22:03:18] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.04250890869370812  
[22:03:31] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.04281594789905563  
[22:03:39] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.042718450311204303  
[22:03:47] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.04258836995790318  
[22:03:59] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.042534955041867294  
[22:04:12] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.042708741885590595  
[22:04:20] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.042653739169740836  
[22:04:28] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.04250346078504994  
[22:04:40] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb  
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----  
Model Paramaters :  
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':  
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}  
The mean absolute error of prediction is: 0.04246925276328048
```

In []:

```
1 # Benchmarking with Differnt Paramaters for Dual Grouped Data for XG Boost
2
3 df = duo_df
4 y_train = pd.DataFrame(df, columns = ['winPlacePerc'])
5 X_train = df.drop(['winPlacePerc'], axis = 1)
6 y_train = y_train.fillna(0)
7
8 for lr in [0.05,0.04,0.06]:
9     for md in [7,9]:
10         for ne in [500,600]:
11             params = {'colsample_bytree': 0.7, 'learning_rate': lr, 'max_depth': md, 'min_child_weight': 4, 'n_estimators': ne, '
12             model = xgb.XGBRegressor(**params)
13             model.fit(
14                 X_train, y_train
15             )
16             print("-----\n Model Paramaters : ")
17             print(params)
18             predict_model(duo_df_test ,duo_df_test_grouped, model)
19
20
```

```
[22:04:54] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04712668620785963
[22:05:10] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04707425384270283
[22:05:26] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.046984259960902275
[22:05:45] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.0469556356273892
[22:06:07] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.047186249716432434
[22:06:22] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04712572601281344
[22:06:37] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04694967451225951
[22:06:59] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.046923191138998045
[22:07:23] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04710603799135232
[22:07:38] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04705759494241881
[22:07:54] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04703563942424392
[22:08:14] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.04702919172339137
```

In []:

```
1 # Benchmarking with Differnt Paramaters for Squad Grouped Data for XG Boost
2
3 df = squad_df
4 y_train = pd.DataFrame(df, columns = ['winPlacePerc'])
5 X_train = df.drop(['winPlacePerc'], axis = 1)
6 y_train = y_train.fillna(0)
7
8 for lr in [0.05,0.04,0.06]:
9     for md in [7,9]:
10         for ne in [500,600]:
11             params = {'colsample_bytree': 0.7, 'learning_rate': lr, 'max_depth': md, 'min_child_weight': 4, 'n_estimators': ne, '
12             model = xgb.XGBRegressor(**params)
13             model.fit(
14                 X_train, y_train
15             )
16             print("-----\n Model Paramaters : ")
17             print(params)
18             predict_model(squad_df_test ,squad_df_test_grouped, model)
19
20
```

```
[22:35:36] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06361005108499965
[22:35:54] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06354192555817892
[22:36:15] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06346587313767461
[22:36:42] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.05, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06345825547122182
[22:37:11] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06368341112786136
[22:37:31] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06358561493414053
[22:37:51] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06341990514865593
[22:38:17] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.04, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06338337728980284
[22:38:46] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06359596562743011
[22:39:04] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 7, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06354468636892242
[22:39:24] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 500, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06350351184681771
[22:39:49] WARNING: C:/buildkite-agent/builds/buildkite-windows-cpu-autoscaling-group-i-03de431ba26204c4d-1/xgboost/xgb
oost-ci-windows/src/objective/regression_obj.cu:213: reg:linear is now deprecated in favor of reg:squarederror.
```

```
-----
Model Paramaters :
{'colsample_bytree': 0.7, 'learning_rate': 0.06, 'max_depth': 9, 'min_child_weight': 4, 'n_estimators': 600, 'nthread':
4, 'objective': 'reg:linear', 'subsample': 0.7, 'tree_method': 'gpu_hist'}
The mean absolute error of prediction is: 0.06351258369358434
```

Grid Search for Linear Regression :

In []:

```

1 from sklearn.linear_model import Ridge
2 from sklearn.model_selection import RepeatedKFold
3 from sklearn.model_selection import GridSearchCV
4 from sklearn.metrics import mean_squared_error, mean_absolute_error

```

In []:

```

1 model = Ridge() # For Linear Regression
2
3 cv = RepeatedKFold(n_splits=5, n_repeats=10, random_state=10)
4 param = {
5     'alpha': [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]
6 }
7
8 # Grid Search : Linear Regression for Solo Data
9 search = GridSearchCV(model, param, cv=cv, verbose=3)
10 resSolo = search.fit(solo_df.drop(['winPlacePerc'], axis = 1), solo_df['winPlacePerc'])
11
12 # Result : Linear Regression for Solo Data
13 print('Best Score: %s' % resSolo.best_score_)
14 print('Best Hyperparameters: %s' % resSolo.best_params_)

```

Fitting 50 folds for each of 8 candidates, totalling 400 fits

C:\Users\nsimh\anaconda3\lib\site-packages\sklearn\linear_model_ridge.py:157: LinAlgWarning: Ill-conditioned matrix (rcond=1.49576e-17): result may not be accurate.

```
return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

[CV 1/50] ENDalpha=1e-05;; score=0.907 total time= 0.4s

C:\Users\nsimh\anaconda3\lib\site-packages\sklearn\linear_model_ridge.py:157: LinAlgWarning: Ill-conditioned matrix (rcond=1.47753e-17): result may not be accurate.

```
return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

[CV 2/50] ENDalpha=1e-05;; score=0.902 total time= 0.2s

C:\Users\nsimh\anaconda3\lib\site-packages\sklearn\linear_model_ridge.py:157: LinAlgWarning: Ill-conditioned matrix (rcond=1.48885e-17): result may not be accurate.

```
return linalg.solve(A, Xy, sym_pos=True, overwrite_a=True).T
```

[CV 3/50] ENDalpha=1e-05;; score=0.902 total time= 0.2s

C:\Users\nsimh\anaconda3\lib\site-packages\sklearn\linear_model_ridge.py:157: LinAlgWarning: Ill-conditioned matrix

In []:

```

1 # Grid Search : Linear Regression for Dual
2 search = GridSearchCV(model, param, cv=cv, verbose=3)
3 resDuo = search.fit(duo_df.drop(['winPlacePerc'], axis = 1), duo_df['winPlacePerc'])
4
5 # Result : Linear Regression for Dual Data
6 print('Best Score: %s' % resDuo.best_score_)
7 print('Best Hyperparameters: %s' % resDuo.best_params_)

```

[CV 8/50] ENDalpha=0.0001;; score=0.894 total time= 0.9s

[CV 9/50] ENDalpha=0.0001;; score=0.883 total time= 0.9s

[CV 10/50] ENDalpha=0.0001;; score=0.893 total time= 0.9s

[CV 11/50] ENDalpha=0.0001;; score=0.894 total time= 1.2s

[CV 12/50] ENDalpha=0.0001;; score=0.893 total time= 1.1s

[CV 13/50] ENDalpha=0.0001;; score=0.891 total time= 1.1s

[CV 14/50] ENDalpha=0.0001;; score=0.888 total time= 1.0s

[CV 15/50] ENDalpha=0.0001;; score=0.895 total time= 1.0s

[CV 16/50] ENDalpha=0.0001;; score=0.893 total time= 0.9s

[CV 17/50] ENDalpha=0.0001;; score=0.892 total time= 1.1s

[CV 18/50] ENDalpha=0.0001;; score=0.893 total time= 1.0s

[CV 19/50] ENDalpha=0.0001;; score=0.888 total time= 1.0s

[CV 20/50] ENDalpha=0.0001;; score=0.894 total time= 1.0s

[CV 21/50] ENDalpha=0.0001;; score=0.894 total time= 1.0s

[CV 22/50] ENDalpha=0.0001;; score=0.891 total time= 1.0s

[CV 23/50] ENDalpha=0.0001;; score=0.893 total time= 0.9s

[CV 24/50] ENDalpha=0.0001;; score=0.894 total time= 0.9s

[CV 25/50] ENDalpha=0.0001;; score=0.889 total time= 0.9s

[CV 26/50] ENDalpha=0.0001;; score=0.895 total time= 0.9s

[CV 27/50] ENDalpha=0.0001;; score=0.893 total time= 0.9s

In []:

```

1 # Grid Search : Linear Regression for Squad
2 search = GridSearchCV(model, param, cv=cv, verbose=3)
3 resSquad = search.fit(squad_df.drop(['winPlacePerc'], axis = 1), squad_df['winPlacePerc'])
4
5 # Result : Linear Regression for Squad Data
6 print('Best Score: %s' % resSquad.best_score_)
7 print('Best Hyperparameters: %s' % resSquad.best_params_)

```

Fitting 50 folds for each of 8 candidates, totalling 400 fits

```

[CV 1/50] END .....alpha=1e-05; score=0.866 total time= 1.3s
[CV 2/50] END .....alpha=1e-05; score=0.866 total time= 1.3s
[CV 3/50] END .....alpha=1e-05; score=0.869 total time= 1.2s
[CV 4/50] END .....alpha=1e-05; score=0.868 total time= 1.2s
[CV 5/50] END .....alpha=1e-05; score=0.867 total time= 1.3s
[CV 6/50] END .....alpha=1e-05; score=0.867 total time= 1.2s
[CV 7/50] END .....alpha=1e-05; score=0.868 total time= 1.4s
[CV 8/50] END .....alpha=1e-05; score=0.868 total time= 1.4s
[CV 9/50] END .....alpha=1e-05; score=0.868 total time= 1.3s
[CV 10/50] END .....alpha=1e-05; score=0.866 total time= 1.4s
[CV 11/50] END .....alpha=1e-05; score=0.867 total time= 1.2s
[CV 12/50] END .....alpha=1e-05; score=0.867 total time= 1.2s
[CV 13/50] END .....alpha=1e-05; score=0.866 total time= 1.2s
[CV 14/50] END .....alpha=1e-05; score=0.867 total time= 1.2s
[CV 15/50] END .....alpha=1e-05; score=0.869 total time= 1.2s
[CV 16/50] END .....alpha=1e-05; score=0.866 total time= 1.3s
[CV 17/50] END .....alpha=1e-05; score=0.867 total time= 1.2s
[CV 18/50] END .....alpha=1e-05; score=0.868 total time= 1.3s

```

Benchmarking :

In []:

```

1 import warnings
2 warnings.filterwarnings('ignore')

```


In []:

```

1  # For Linear Regression : Solo Group model
2  df = solo_df
3  cv = RepeatedKFold(n_splits=5, n_repeats=10, random_state=10)
4
5  for alp in [1e-7,1e-6,1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]:
6      param = {'alpha': [alp]}
7      model = Ridge()
8      search = GridSearchCV(model, param, cv=cv, verbose=0)
9      resSolo = search.fit(df.drop(['winPlacePerc'], axis = 1), df['winPlacePerc'])
10
11  # Result : Linear Regression for Solo Data
12  print('-----\n Paramater : ')
13  print("Alpha : ",alp)
14  print('Best Score: %s' % resSolo.best_score_)
15  print('Best Hyperparameters: %s' % resSolo.best_params_)
16

```

```

-----
Paramater :
Alpha : 1e-07
Best Score: 0.9030150226209629
Best Hyperparameters: {'alpha': 1e-07}
-----

```

```

Paramater :
Alpha : 1e-06
Best Score: 0.9030150226209591
Best Hyperparameters: {'alpha': 1e-06}
-----

```

```

Paramater :
Alpha : 1e-05
Best Score: 0.9030150226209237
Best Hyperparameters: {'alpha': 1e-05}
-----

```

```

Paramater :
Alpha : 0.0001
Best Score: 0.9030150226205675
Best Hyperparameters: {'alpha': 0.0001}
-----

```

```

Paramater :
Alpha : 0.001
Best Score: 0.9030150226170067
Best Hyperparameters: {'alpha': 0.001}
-----

```

```

Paramater :
Alpha : 0.01
Best Score: 0.9030150225813918
Best Hyperparameters: {'alpha': 0.01}
-----

```

```

Paramater :
Alpha : 0.1
Best Score: 0.903015022244786
Best Hyperparameters: {'alpha': 0.1}
-----

```

```

Paramater :
Alpha : 1
Best Score: 0.9030150185791057
Best Hyperparameters: {'alpha': 1}
-----

```

```

Paramater :
Alpha : 10
Best Score: 0.9030149745093616
Best Hyperparameters: {'alpha': 10}
-----

```

```

Paramater :
Alpha : 100
Best Score: 0.9030137804241819
Best Hyperparameters: {'alpha': 100}
-----

```

In []:

```

1  # For Linear Regression Dual data
2  df = duo_df
3  cv = RepeatedKFold(n_splits=5, n_repeats=10, random_state=10)
4
5  for alp in [1e-7,1e-6,1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]:
6      param = {'alpha': [alp]}
7      model = Ridge()
8      search = GridSearchCV(model, param, cv=cv, verbose=0)
9      resSolo = search.fit(df.drop(['winPlacePerc'], axis = 1), df['winPlacePerc'])
10
11  # Result : Linear Regression for Solo Data
12  print('-----\n Paramater : ')
13  print("Alpha : ",alp)
14  print('Best Score: %s' % resSolo.best_score_)
15  print('Best Hyperparameters: %s' % resSolo.best_params_)
16

```

```

-----
Paramater :
Alpha : 1e-07
Best Score: 0.8918107597551744
Best Hyperparameters: {'alpha': 1e-07}
-----

```

```

Paramater :
Alpha : 1e-06
Best Score: 0.8918107597553199
Best Hyperparameters: {'alpha': 1e-06}
-----

```

```

Paramater :
Alpha : 1e-05
Best Score: 0.8918107597567841
Best Hyperparameters: {'alpha': 1e-05}
-----

```

```

Paramater :
Alpha : 0.0001
Best Score: 0.891810759771423
Best Hyperparameters: {'alpha': 0.0001}
-----

```

```

Paramater :
Alpha : 0.001
Best Score: 0.8918107599178035
Best Hyperparameters: {'alpha': 0.001}
-----

```

```

Paramater :
Alpha : 0.01
Best Score: 0.891810761381029
Best Hyperparameters: {'alpha': 0.01}
-----

```

```

Paramater :
Alpha : 0.1
Best Score: 0.891810775954864
Best Hyperparameters: {'alpha': 0.1}
-----

```

```

Paramater :
Alpha : 1
Best Score: 0.8918109160510385
Best Hyperparameters: {'alpha': 1}
-----

```

```

Paramater :
Alpha : 10
Best Score: 0.8918119034357299
Best Hyperparameters: {'alpha': 10}
-----

```

```

Paramater :
Alpha : 100
Best Score: 0.8918133584503248
Best Hyperparameters: {'alpha': 100}
-----

```

In []:

```

1  # For Linear Regression Squad Data
2  df = squad_df
3  cv = RepeatedKFold(n_splits=5, n_repeats=10, random_state=10)
4
5  for alp in [1e-7,1e-6,1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100]:
6      param = {'alpha': [alp]}
7      model = Ridge()
8      search = GridSearchCV(model, param, cv=cv, verbose=0)
9      resSolo = search.fit(df.drop(['winPlacePerc'], axis = 1), df['winPlacePerc'])
10
11  # Result : Linear Regression for Solo Data
12  print('-----\n Paramater : ')
13  print("Alpha : ",alp)
14  print('Best Score: %s' % resSolo.best_score_)
15  print('Best Hyperparameters: %s' % resSolo.best_params_)
16

```

```

-----
Paramater :
Alpha : 1e-07
Best Score: 0.8670887324596849
Best Hyperparameters: {'alpha': 1e-07}
-----

```

```

Paramater :
Alpha : 1e-06
Best Score: 0.8670887324596939
Best Hyperparameters: {'alpha': 1e-06}
-----

```

```

Paramater :
Alpha : 1e-05
Best Score: 0.8670887324597838
Best Hyperparameters: {'alpha': 1e-05}
-----

```

```

Paramater :
Alpha : 0.0001
Best Score: 0.8670887324606825
Best Hyperparameters: {'alpha': 0.0001}
-----

```

```

Paramater :
Alpha : 0.001
Best Score: 0.8670887324696693
Best Hyperparameters: {'alpha': 0.001}
-----

```

```

Paramater :
Alpha : 0.01
Best Score: 0.8670887325595352
Best Hyperparameters: {'alpha': 0.01}
-----

```

```

Paramater :
Alpha : 0.1
Best Score: 0.8670887334581046
Best Hyperparameters: {'alpha': 0.1}
-----

```

```

Paramater :
Alpha : 1
Best Score: 0.8670887424348155
Best Hyperparameters: {'alpha': 1}
-----

```

```

Paramater :
Alpha : 10
Best Score: 0.8670888313072785
Best Hyperparameters: {'alpha': 10}
-----

```

```

Paramater :
Alpha : 100
Best Score: 0.8670896342291694
Best Hyperparameters: {'alpha': 100}
-----

```

In []:

1