```
#Final Code Submitted on Kaggle

import numpy as np
import pandas as pd

import seaborn as sns
import matplotlib.pyplot as plt

from timeit import default_timer as timer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
import lightgbm as lgb
from sklearn.metrics import mean_absolute_error, r2_score
import gc,sys
gc.enable()

def feature_engineering(is_train=True,debug=True):
    test_idx = None
    if is_train:
        print("processing train.csv")
        if debug == True:
            df = pd.read_csv('../input/train_V2.csv', nrows=10000)
        else:
            df = pd.read_csv('../input/train_V2.csv')

        df = df[df['maxPlace'] > 1]
    else:
        print("processing test.csv")
        df = pd.read_csv('../input/test_V2.csv')
```

```
    test_idx = df.Id

# df = reduce_mem_usage(df)
#df['totalDistance'] = df['rideDistance'] + df["walkDistance"] + df["swimDistance"]

# df = df[:100]

print("remove some columns")
target = 'winPlacePerc'

print("Adding Features")

df['headshotrate'] = df['kills']/df['headshotKills']
df['killStreakrate'] = df['killStreaks']/df['kills']
df['healthitems'] = df['heals'] + df['boosts']
df['totalDistance'] = df['rideDistance'] + df["walkDistance"] + df["swimDistance"]
df['killPlace_over_maxPlace'] = df['killPlace'] / df['maxPlace']
df['headshotKills_over_kills'] = df['headshotKills'] / df['kills']
df['distance_over_weapons'] = df['totalDistance'] / df['weaponsAcquired']
df['walkDistance_over_heals'] = df['walkDistance'] / df['heals']
df['walkDistance_over_kills'] = df['walkDistance'] / df['kills']
df['killsPerWalkDistance'] = df['kills'] / df['walkDistance']
df["skill"] = df["headshotKills"] + df["roadKills"]

#extra
df['playersJoined'] = df.groupby('matchId')['matchId'].transform('count')
df['killsNorm'] = df['kills']*((100-df['playersJoined'])/100 + 1)
df['headshotKillsNorm'] = df['headshotKills']*((100-df['playersJoined'])/100 + 1)
df['killPlaceNorm'] = df['killPlace']*((100-df['playersJoined'])/100 + 1)
```

```
    df['killPointsNorm'] = df['killPoints']*((100-df['playersJoined'])/100 + 1)
    df['killStreaksNorm'] = df['killStreaks']*((100-df['playersJoined'])/100 + 1)
    df['longestKillNorm'] = df['longestKill']*((100-df['playersJoined'])/100 + 1)
    df['roadKillsNorm'] = df['roadKills']*((100-df['playersJoined'])/100 + 1)
    df['teamKillsNorm'] = df['teamKills']*((100-df['playersJoined'])/100 + 1)
    df['damageDealtNorm'] = df['damageDealt']*((100-df['playersJoined'])/100 + 1)
    df['DBNOsNorm'] = df['DBNOs']*((100-df['playersJoined'])/100 + 1)
    df['revivesNorm'] = df['revives']*((100-df['playersJoined'])/100 + 1)


df=df.drop(['kills','headshotKills','killPlace','killPoints','killStreaks','longestKill','roadKills','teamKills','dama
geDealt','DBNOs','revives'],axis=1)
    df[df == np.Inf] = np.NaN
    df[df == np.NINF] = np.NaN

    print("Removing Na's From DF")
    df.fillna(0, inplace=True)
    memory_usage(df)



    features = list(df.columns)
    features.remove("Id")
    features.remove("matchId")
    features.remove("groupId")
    features.remove("matchType")

    # matchType = pd.get_dummies(df['matchType'])
    # df = df.join(matchType)
```

```
    y = None



    if is_train:
        print("get target")
        y = np.array(df.groupby(['matchId','groupId'])[target].agg('mean'), dtype=np.float64)
        features.remove(target)


    print("get group mean feature")
    agg = df.groupby(['matchId','groupId'])[features].agg('mean')
    agg_rank = agg.groupby('matchId')[features].rank(pct=True).reset_index()


    if is_train: df_out = agg.reset_index()[['matchId','groupId']]
    else: df_out = df[['matchId','groupId']]


    df_out = df_out.merge(agg.reset_index(), suffixes=["", ""], how='left', on=['matchId', 'groupId'])
    df_out = df_out.merge(agg_rank, suffixes=["_mean", "_mean_rank"], how='left', on=['matchId',
'groupId'])


    # print("get group sum feature")
    # agg = df.groupby(['matchId','groupId'])[features].agg('sum')
    # agg_rank = agg.groupby('matchId')[features].rank(pct=True).reset_index()
    # df_out = df_out.merge(agg.reset_index(), suffixes=["", ""], how='left', on=['matchId', 'groupId'])
    # df_out = df_out.merge(agg_rank, suffixes=["_sum", "_sum_rank"], how='left', on=['matchId',
'groupId'])


    # print("get group sum feature")
    # agg = df.groupby(['matchId','groupId'])[features].agg('sum')
    # agg_rank = agg.groupby('matchId')[features].agg('sum')
```

```
    # df_out = df_out.merge(agg.reset_index(), suffixes=["", ""], how='left', on=['matchId', 'groupId'])
    # df_out = df_out.merge(agg_rank.reset_index(), suffixes=["_sum", "_sum_pct"], how='left', on=['matchId', 'groupId'])

    print("get group max feature")
    agg = df.groupby(['matchId','groupId'])[features].agg('max')
    agg_rank = agg.groupby('matchId')[features].rank(pct=True).reset_index()
    df_out = df_out.merge(agg.reset_index(), suffixes=["", ""], how='left', on=['matchId', 'groupId'])
    df_out = df_out.merge(agg_rank, suffixes=["_max", "_max_rank"], how='left', on=['matchId', 'groupId'])
    memory_usage(df_out)

    print("get group median feature")
    agg = df.groupby(['matchId','groupId'])[features].agg('median')
    agg_rank = agg.groupby('matchId')[features].rank(pct=True).reset_index()
    df_out = df_out.merge(agg.reset_index(), suffixes=["", ""], how='left', on=['matchId', 'groupId'])
    df_out = df_out.merge(agg_rank, suffixes=["_median", "_median_rank"], how='left', on=['matchId', 'groupId'])
    memory_usage(df_out)

    print("get group min feature")
    agg = df.groupby(['matchId','groupId'])[features].agg('min')
    agg_rank = agg.groupby('matchId')[features].rank(pct=True).reset_index()
    df_out = df_out.merge(agg.reset_index(), suffixes=["", ""], how='left', on=['matchId', 'groupId'])
    df_out = df_out.merge(agg_rank, suffixes=["_min", "_min_rank"], how='left', on=['matchId', 'groupId'])
    memory_usage(df_out)

    print("get group size feature")
    agg = df.groupby(['matchId','groupId']).size().reset_index(name='group_size')
```

```
    df_out = df_out.merge(agg, how='left', on=['matchId', 'groupId'])
    memory_usage(df_out)

    print("get match mean feature")
    agg = df.groupby(['matchId'])[features].agg('mean').reset_index()
    df_out = df_out.merge(agg, suffixes=["", "_match_mean"], how='left', on=['matchId'])
    memory_usage(df_out)
    # print("get match type feature")
    # agg = df.groupby(['matchId'])[matchType.columns].agg('mean').reset_index()
    # df_out = df_out.merge(agg, suffixes=["", "_match_type"], how='left', on=['matchId'])

    print("get match size feature")
    agg = df.groupby(['matchId']).size().reset_index(name='match_size')
    df_out = df_out.merge(agg, how='left', on=['matchId'])

    df_out.drop(["matchId", "groupId"], axis=1, inplace=True)
    memory_usage(df_out)

    X = df_out

    feature_names = list(df_out.columns)

    del df, df_out, agg, agg_rank
    gc.collect()

    return X, y, feature_names, test_idx

def memory_usage(train):
    for col in train.columns:
```

```
        col_type = train[col].dtype

        if col_type != object:
            c_min = train[col].min()
            c_max = train[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    train[col] = train[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    train[col] = train[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    train[col] = train[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    train[col] = train[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    train[col] = train[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    train[col] = train[col].astype(np.float32)
                else:
                    train[col] = train[col].astype(np.float64)
    return train

X, Y, train_columns,__ = feature_engineering(True,False)

testsample= pd.read_csv('../input/test_V2.csv')

Xtest, _, _ , test_idx = feature_engineering(False,True)

params = {"objective" : "regression", "metric" : "mae", 'n_estimators':20000,"num_leaves" : 45,
"learning_rate" : 0.1, "bagging_fraction" : 0.7,"bagging_freq": 10, "bagging_seed" : 3, "num_threads" :
4,"colsample_bytree" : 0.7}
```

```
params['metric'] = 'auc'
lgtrain = lgb.Dataset(X, label=Y)
#lgval = lgb.Dataset(test_X, label=test_Y)
model = lgb.train(params, lgtrain)
pred_test = model.predict(Xtest, num_iteration=model.best_iteration)
test_sub = testsample.iloc[:,0]
test_sub = pd.concat([test_sub,pd.DataFrame(pred_test,columns=['winPlacePerc'])],axis=1)
test_sub.to_csv("submission_das.csv", index=False)



#Code tried on model selection
## Random Forest

train_data=pd.read_csv(r'D:\EE660\Project\train_V2.csv\train_V2.csv')
df = train_data.drop(['matchId','groupId','Id','matchType'],axis=1)
for col in df.columns:
    col_type = df[col].dtype

    if col_type != object:
        c_min = df[col].min()
        c_max = df[col].max()
        if str(col_type)[:3] == 'int':
            if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
```

```
                df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
df=df.fillna(value={'winPlacePerc': np.random.random()})
Xtrain,Xval,Ytrain,Yval = train_test_split(df.iloc[:,:24],df.iloc[:,24],test_size=0.3)
regr = RandomForestRegressor()
regr = regr.fit(Xtrain,Ytrain)
Ypred = regr.predict(Xval)
mse= mean_squared_error(Ypred[:,1],Yval)
r2score = r2_score(Ypred[:,1],Yval)
print(mse)
print(r2score)

## XGBR

train_data=pd.read_csv(r'D:\EE660\Project\train_V2.csv\train_V2.csv')
train = train_data[train_data['winPlacePerc'].isna() != True]
train['playersJoined'] = train.groupby('matchId')['matchId'].transform('count')
train['killsNorm'] = train['kills']*((100-train['playersJoined'])/100 + 1)
train['headshotKillsNorm'] = train['headshotKills']*((100-train['playersJoined'])/100 + 1)
train['killPlaceNorm'] = train['killPlace']*((100-train['playersJoined'])/100 + 1)
train['killPointsNorm'] = train['killPoints']*((100-train['playersJoined'])/100 + 1)
train['killStreaksNorm'] = train['killStreaks']*((100-train['playersJoined'])/100 + 1)
train['longestKillNorm'] = train['longestKill']*((100-train['playersJoined'])/100 + 1)
```

```
train['roadKillsNorm'] = train['roadKills']*((100-train['playersJoined'])/100 + 1)

train['teamKillsNorm'] = train['teamKills']*((100-train['playersJoined'])/100 + 1)

train['damageDealtNorm'] = train['damageDealt']*((100-train['playersJoined'])/100 + 1)

train['DBNOsNorm'] = train['DBNOs']*((100-train['playersJoined'])/100 + 1)

train['revivesNorm'] = train['revives']*((100-train['playersJoined'])/100 + 1)


win = train['winPlacePerc']

train = train.drop([ 'kills', 'headshotKills', 'killPlace', 'killPoints', 'killStreaks',

 'longestKill', 'roadKills', 'teamKills', 'damageDealt', 'DBNOs',
'revives','matchId','groupId','Id','matchType','winPlacePerc'],axis=1)

train['winPlacePerc'] = win


trainX,testX,trainY,testY = train_test_split(train.iloc[:,:25],train.iloc[:,25],test_size=0.3)

clf = XGBRegressor(n_estimators=1000, learning_rate=0.05)

clf.fit(trainX,trainY)

Ypred = clf.predict(testX)

mae = mean_absolute_error(Ypred,testY)

print(mae)


Ypred = clf.predict(testX)

mae = mean_absolute_error(Ypred,testY)

print(mae)
```