PUBG Finish Placement Prediction

November 19, 2018

DePaul University
DSC 478 Final Project

Letian Xu - 1871304

Table of Contents

I. Introduction

Battle Royale-style video games have taken the world by storm. 100 players are dropped onto an island empty-handed and must explore, scavenge, and eliminate other players until only one is left standing, all while the play zone continues to shrink. Player Unknown's Battlegrounds (PUBG) has enjoyed massive popularity.

The team at PUBG has made official game data available for the public to explore and scavenge outside of "The Blue Circle." Kaggle collected data made possible through the PUBG Developer API. With the provided data, we were trying to predict final placement from final in-game stats and initial player ratings.

II. Data Description

In a PUBG game, up to 100 players start in each match (matchId). Players can be on teams (groupId) which get ranked at the end of the game (winPlacePerc) based on how many other teams are still alive when they are eliminated. In game, players can pick up different munitions, revive downed-but-not-out (knocked) teammates, drive vehicles, swim, run, shoot, and experience all of the consequences -- such as falling too far or running themselves over and eliminating themselves.

In the Kaggle website, the original data was pre-split into training and testing dataset. We will be using the whole training set in the project since the target variable was missing in the testing dataset due the Kaggle rule. After checking the missing value, the dataset was completed. The whole training dataset includes 4357336 cases which refer to every single player's record in every game, and the dataset contains 26 columns including target variable, winning place percentage.

Variable Description:

- DBNOs - Number of enemy players knocked.
- assists - Number of enemy players this player damaged that were killed by teammates.
- boosts - Number of boost items used.

- damageDealt - Total damage dealt. Note: Self-inflicted damage is subtracted.
- headshotKills - Number of enemy players killed with headshots.
- heals - Number of healing items used.
- Id - Player’s Id
- killPlace - Ranking in match of number of enemy players killed.
- killPoints - Kills-based external ranking of player.
- killStreaks - Max number of enemy players killed in a short amount of time.
- kills - Number of enemy players killed.
- longestKill - Longest distance between player and player killed at time of death.
- rankPoints - Elo-like ranking of player. This ranking is inconsistent and is being deprecated in the API’s next version, so use with caution. Value of -1 takes place of “None”.
- revives - Number of times this player revived teammates.
- rideDistance - Total distance traveled in vehicles measured in meters.
- roadKills - Number of kills while in a vehicle.
- swimDistance - Total distance traveled by swimming measured in meters.
- teamKills - Number of times this player killed a teammate.
- vehicleDestroys - Number of vehicles destroyed.
- walkDistance - Total distance traveled on foot measured in meters.
- weaponsAcquired - Number of weapons picked up.
- winPoints - Win-based external ranking of player. (Think of this as an Elo ranking where only winning matters.) If there is a value other than -1 in rankPoints, then any 0 in winPoints should be treated as a “None”.
- groupId - ID to identify a group within a match. If the same group of players plays in different matches, they will have a different groupId each time.
- numGroups - Number of groups we have data for in the match.
- maxPlace - Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
- winPlacePerc - The target of prediction. This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match.

Variable Data Type:

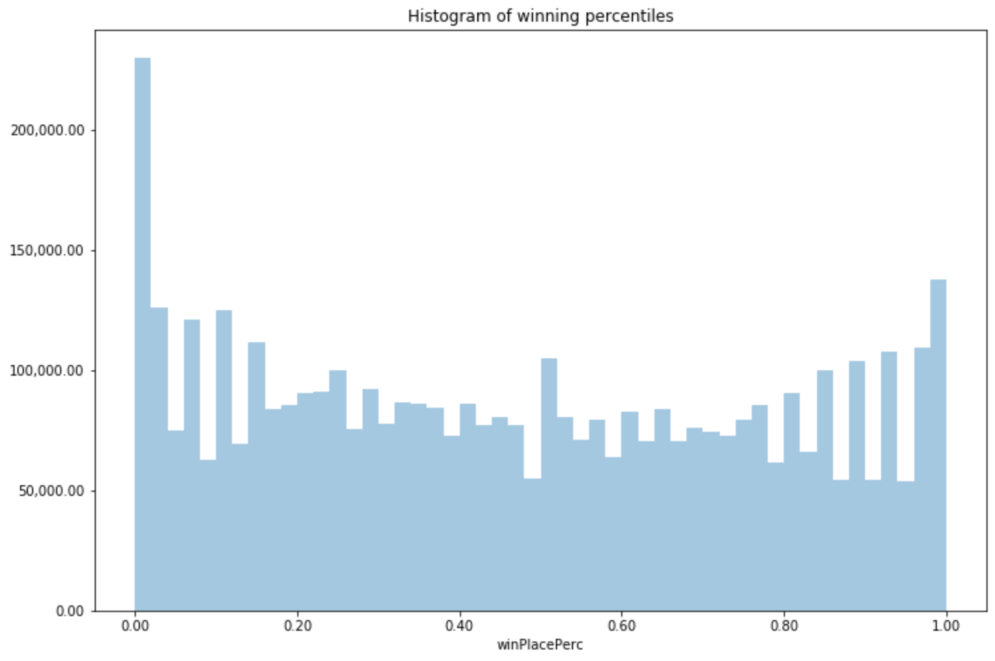
Since all the features were numeric variable, then I choose to conduct regression as the main method to build a prediction model. I will discuss the detail the rest of my project.

Variable	Data Type
Id	Numeric
groupId	Numeric
matchId	Numeric
assists	Numeric
boosts	Numeric
damageDealt	Numeric
DBNOs	Numeric
headshotKills	Numeric
heals	Numeric
killPlace	Numeric
killPoints	Numeric
kills	Numeric
killStreaks	Numeric
longestKill	Numeric
maxPlace	Numeric
numGroups	Numeric
revives	Numeric
rideDistance	Numeric
roadKills	Numeric
swimDistance	Numeric
teamKills	Numeric
vehicleDestroys	Numeric
walkDistance	Numeric
weaponsAcquired	Numeric
winPoints	Numeric
winPlacePerc	Numeric

III. Data Visualization

In the previous chapter, we already know that the target variable was distribute in between 0 to 1, so we need to visualize it to if it is normal distributed or not.

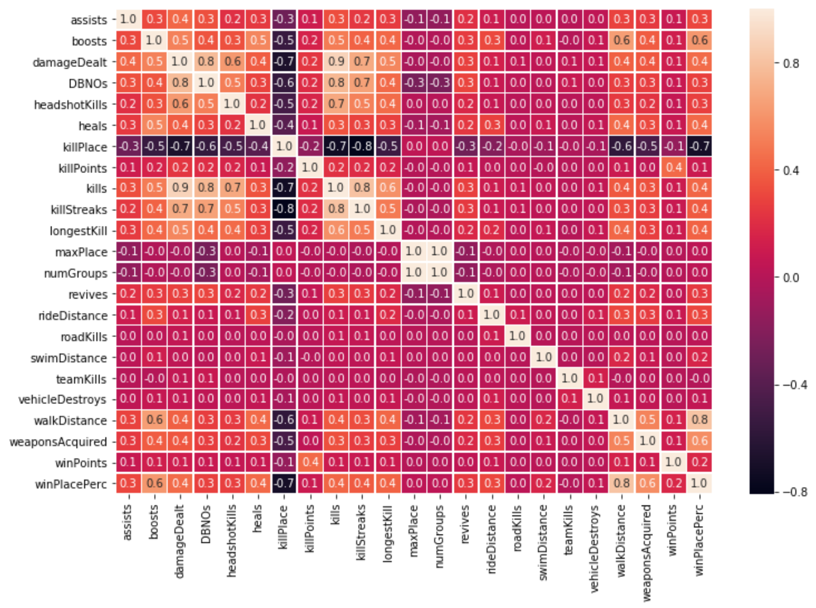
Target Variable



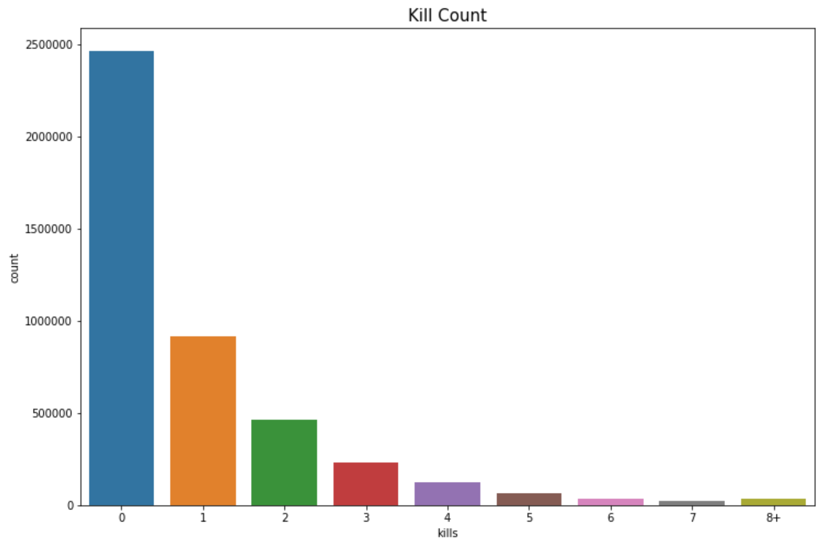
From the above, we can see that our percentile distribution isn't exactly as uniform as we expected. There seems to be more cases occurring at low percentiles. This indicates that overall, we have more losers than winners, which reflect the rule of PUGB game, only one player or team gets victory.

The higher counts at the percentile extremes is to be expected. There is always some someone getting the percentiles scores of zero and 1, the rest of the scores varying across different matches. After calculating this distribution, the average winning percentile is 0.472, the median is 0.458. Hence, majority of players end up in the middle of the game.

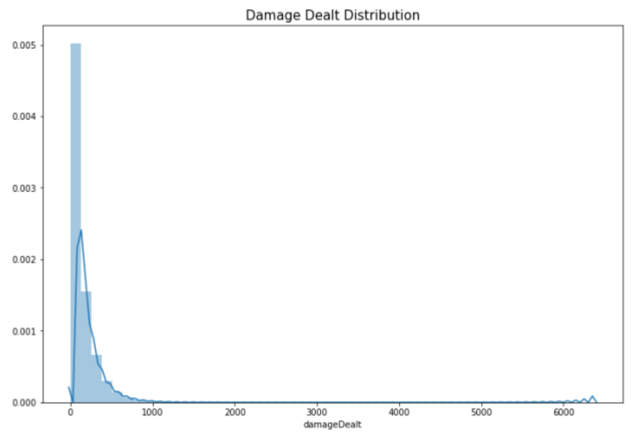
Variables Visualization

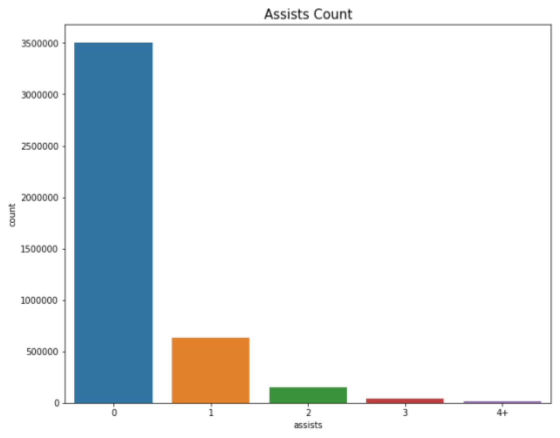


In terms of the target variable (winPlacePerc), there are a few variables high medium to high correlation. The highest positive correlation is walkDistance and the highest negative the killPlace. Also, weaponsAcquired and walkDistance are both strongly and positively correlated, so those may be the two to check out next.



The average person kills 0.9345 players, 99% of people have 7.0 kills or less, while the most kills ever recorded is 60.





The average person had 132.9033 damage, 99% of people can't exceed 775.8 damage dealt or more, while the highest damage dealt ever recorded is 6384.0.

The average person had 0.2656 assist, 99% of people have 3.0 assist or less, while the highest assist ever recorded is 20.

After plot three main variable in the data, kill, damage, and assist, it actually very shocked me since it was maximum 100 players in one match and kill number with 60 means that a player or team kills 60% of other players. Considering that every player was distributed equally to different area in the match, the highest kill number was suspicious and probably a cheating player.

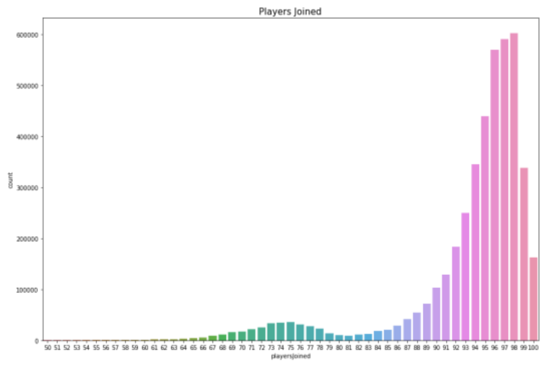
Then, in order to remove those hypothesis cheating players, I did a whole dataset description. When I plot some variables' distributions out, many interesting but strange cases were found. So, I will remove those highly doubtful cheating players in the next step. The discovery process will also be present in the next step.

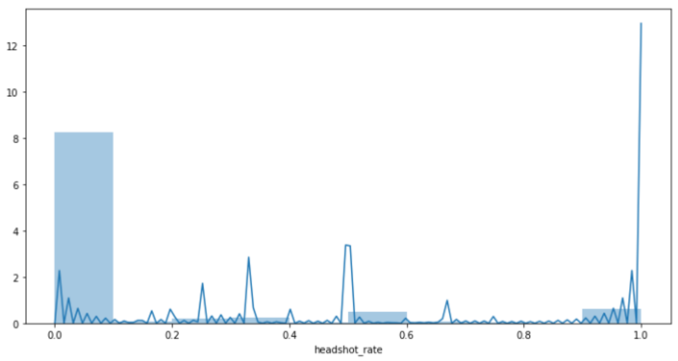
IV. Data Preprocessing

Feature Engineering

Besides the original features, there were some feature that combine together to help us in different aspect. Such as the number of total players per match, since less players in one match, more chance to standing at last. As we talked before, there were some possible cheating players existed in the data. Then instead only use original features to detect, I generate some new feature based on original features to help with us.

The generated features and visualization plot were below:
playersJoined: this variable calculates how many players per match;



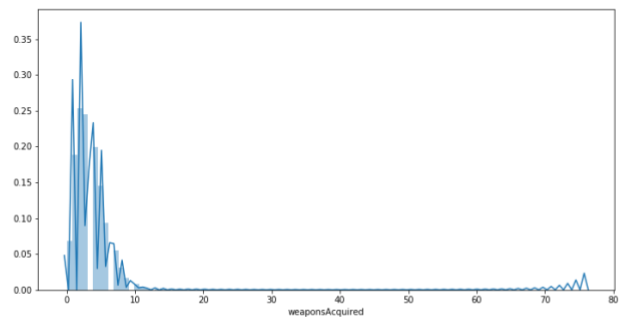


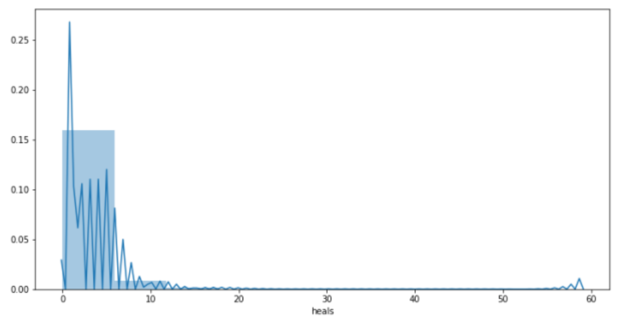
totalDistance: the total distance that a player moved in game;
killsWithoutMoving: I try to identify cheaters by checking if people are getting kills without moving. We first identify the totalDistance travelled by a player and then set a Boolean value to True if someone got kills without moving a single inch. We will remove cheaters in our outlier detection section.
headshot_rate: we first take a look at the whole dataset and create a new feature 'headshot_rate'. We see that the most players score in the 0 to 10% region. However, there are a few players that have a headshot_rate of 100% percent with more than 7 kills.

Remove Outliners

As we use new features, possible cheating players were detected out in dataset.
The most obvious cheating player will be the player that kill people without moving. It is already fishy if a player didn't move an inch during the whole game, but the player could be away from keyboard or got killed instantly. However, if the player can kill another player without moving then it is most likely a cheater.

Then, the headshot percentage will be the second way to find out cheating players. There were 28 players that have a headshot_rate of 100% percent with more than 7 kills. Since 99% of people have 7.0 kills or less, it is very strange to have 100% headshot kill when you have more than 7 kills. Then, we consider those players as cheating players also.





Also, the weapon and heal acquired number also will be an indicator for us to detect cheating players. I remove the player that have more than 60 weapons or 50 healing items in game. After removing those possible cheating players, original dataset shrinks from 4357336 cases to 4355887 cases, almost 1500 cases were removed.

V. Dimensional Reduction

Before we heading to build the prediction model, we need to do feature reduction or feature extraction to generate different dataset. Then, we can compare different model's results together to discuss if remove some comparable irrelevant feature will improve model performance or not.

Feature selection

In order to do feature selection, I extract different subsets with corresponding percentage of original features with the computed correlation between each regressor and the target. Use the F-score and p-value to testify the different feature combinations' subsets between the original dataset. Then, use the simple linear regression, we can generate different the coefficient of determination R^2 of the prediction.

ID	%	R^2
1	1	0.14243584720273017
2	6	0.12082565421758706
3	11	0.12035321870618106
4	16	0.12025276581328062
5	21	0.1148372948094489
6	26	0.11481988136070861
7	31	0.11172755808643589
8	36	0.11171994222121706
9	41	0.10425419935581925
10	46	0.10416476615137234
11	51	0.1038273622008418
12	56	0.10382363878254867
13	61	0.10289663545904046
14	66	0.10277423957263643
15	71	0.10277614906086403
16	76	0.1027462165225097

17	81	0.1170030929830116471
18	86	0.09972186692034407
19	91	0.09972233984970455
20	96	0.09553846728429213

With relevant lowest R^2 number was generated from 96% subset. Hence, I will use the 96% subset, which removed the teamKills feature. It is interesting since the 96% was the best optimum number but also the last test number. Maybe the raw data actually is better for regression prediction, hence I will be including raw data on the prediction model step to compare our results. Also, some ID number related variables were removed, since those were irrelevant with our prediction.

Principle Component Analysis

The result was not satisfying in feature selection, then I use the function in sklearn to conduct the PCA process. The purpose of PCA is trying to use the PCA dataset to generate different prediction model and compare the results with each other.

So, I use the 95% explained variance as my boundary, then I got five principle components, and the total explained variance with 0.950. In the end, I apply the min max scale to all features in different dataset, and head to our model build and result analysis.

VI. Model Description and Evaluation

Model Selection

In this project, we keep using the regression as out main method. The simple linear regression and random forest regression were used to compare with each other. I split the data into two datasets, training and testing with holdout partitioning. I will use training dataset to fit model and use testing dataset to do prediction. Since I keep the random seed as 478, so the split method maintains the same way on different datasets.

The Random Forest is one of the most effective machine learning models for predictive analytics, and it is very good at handling tabular data with numerical features with fewer than hundreds of categories. Unlike linear models, random forests are able to capture non-linear interaction between the features and the target.

For random forest regression, each base classifier is a simple decision tree. More formally we can write this class of models as:

$$\mathrm{g}(x) = f_0(x) + f_1(x) + f_2(x) + \cdots$$

where the final model g is the sum of simple base models fi. Here, each base classifier is a simple decision tree.This broad technique of using multiple models to obtain better predictive performance is called model ensemble. In random forests, all the base models are constructed independently using a different subsample of the data.

Evaluation

We will use three measurement to compare different results:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors;

Mean Squared Error (MSE) is the mean of the squared errors;

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors.

	Simple Linear Regression			Random Forest Regression		
	Raw Data	Feature Selection	PCA	Raw Data	Feature Selection	PCA
MAE	0.0953	0.0954	0.13847	0.06134	0.062	0.0736
MSE	0.017	0.017	0.03035	0.0075	0.0077	0.0118
RMSE	0.13046	0.13055	0.17421	0.0866	0.0875	0.1088

VII. Conclusion

Among all 18 measurement results, the Random Forest Regression on raw data give us the best result. It is expected to have best result on raw data since we already find out on our feature selection step. It might indicate that those original variables are significantly important to predict the final winning result.

Overall, the Random Forest Regression provided better results than Simple Linear Regression, and Feature Selection were slightly not patch on Raw Data, and PCA not quite good preprocess to conduct to PUBG prediction.

After checking the Regression Coefficients, there were some high coefficients number reveal some useful rules in PUBG, and it might be helpful for a player to get higher winning place:

- You shall assist your teammates to eliminate your enemies;
- You shall boost yourself to get a quicker move;
- You shall deal a lot of damage;
- You shall heal yourself to make sure longer existence;

- You shall be able to kill enemy from far distance;
- You shall walk a lot to manage to survive at last.

VIII. Reference

Random Forest Regression: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Evaluation Measurement: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics; https://www.ritchieng.com/machine-learning-evaluate-linear-regression-model/

Visualization Tools: https://seaborn.pydata.org/

IX. Appendix

```
#Data Preprocessing and Visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

train = pd.read_csv('train.csv')


# In[34]:


train.isnull().values.any()


# In[35]:


test.isnull().values.any()


# In[4]:


train.info()


# In[5]:


train.head(3)


# In[6]:


# Drop the irrelevant attributes
train = train.drop(train.columns[[0, 1, 2]], axis=1)
train.head(3)
```

```
# In[56]:


train.describe(include = "all")


# In[12]:


#Assign Target labels to a new table
y_train=train['winPlacePerc']
y_train.head(3)


# In[13]:


#Assign Predictor variables to X table
x_train=train.loc[:, 'assists':'winPoints']
x_train.head(3)


# In[19]:


print('The average winning percentile is {:.3f}, the median is {:.3f}'.format(y_train.mean(), y_train.median()))


# In[31]:


fig, ax = plt.subplots(figsize=(12,8))
formatter = plt.FuncFormatter(lambda y_train, y: '{:,.2f}'.format(y_train))

ax.yaxis.set_major_formatter(formatter=formatter)
ax.xaxis.set_major_formatter(formatter=formatter)

ax.set_title('Histogram of winning percentiles')
sns.distplot(y_train, bins=50, kde=False, ax=ax)


# In[14]:


print("The average person kills {:.4f} players, 99% of people have {} kills or less, while the most kills ever recorded is {}."
      .format(train['kills'].mean(),train['kills'].quantile(0.99), train['kills'].max()))


# In[32]:


data = train.copy()
data.loc[data['kills'] > data['kills'].quantile(0.99)] = '8+'
plt.figure(figsize=(12,8))
sns.countplot(data['kills'].astype('str').sort_values())
plt.title("Kill Count",fontsize=15)
plt.show()


# In[38]:


print("The average person had {:.4f} damage, 99% of people can't excced {} damage dealt or more, while the highest damage dealt ever recorded is {}."
      .format(train['damageDealt'].mean(),train['damageDealt'].quantile(0.99), train['damageDealt'].max()))


# In[37]:
```

```
data = train.copy()
plt.figure(figsize=(12,8))
plt.title("Damage Dealt Distribution",fontsize=15)
sns.distplot(data['damageDealt'])
plt.show()


# In[40]:


print("The average person had {:.4f} assist, 99% of people have {} assist or less, while the highest assist ever recorded is {}."
      .format(train['assists'].mean(),train['assists'].quantile(0.99), train['assists'].max()))


# In[45]:


data = train.copy()
data.loc[data['assists'] > data['assists'].quantile(0.99)] = '4+'
plt.figure(figsize=(10,8))
sns.countplot(data['assists'].astype('str').sort_values())
plt.title("Assists Count",fontsize=15)
plt.show()


# In[58]:


f,ax = plt.subplots(figsize=(12, 8))
sns.heatmap(train.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
plt.show()

# Data Regression Model and Prediction

# In[159]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import feature_selection


# In[160]:


df = pd.read_csv('train.csv')


# In[161]:


df.shape


# ## Feature Engineering

# In[62]:


# PlayersJoin
df['playersJoined'] = df.groupby('matchId')['matchId'].transform('count')
data = df.copy()
data = data[data['playersJoined']>49]
plt.figure(figsize=(15,10))
sns.countplot(data['playersJoined'])
plt.title("Players Joined",fontsize=15)
plt.show()


# In[64]:


# remove cheating player
# Kills without movement
df['totalDistance'] = df['rideDistance'] + df['walkDistance'] + df['swimDistance']
df['killsWithoutMoving'] = ((df['kills'] > 0) & (df['totalDistance'] == 0))

# 100% headshot players
df['headshot_rate'] = df['headshotKills'] / df['kills']
df['headshot_rate'] = df['headshot_rate'].fillna(0)

display(df[df['killsWithoutMoving'] == True].shape)
df[df['killsWithoutMoving'] == True].head()


# ## Preprocessing

# In[65]:


# Remove outliners that kill players without moving
df.drop(df[df['killsWithoutMoving'] == True].index, inplace=True)


# In[66]:


# visualize headshot rate
plt.figure(figsize=(12,6))
sns.distplot(df['headshot_rate'], bins=10)
plt.show()


# In[67]:


# Players who made a minimum of 7 kills and have a headshot_rate of 100%
display(df[(df['headshot_rate'] == 1) & (df['kills'] > 7)].shape)


# In[68]:


# Weapons acquired cheaters
plt.figure(figsize=(12,6))
sns.distplot(df['weaponsAcquired'], bins=100)
plt.show()


# In[69]:


df[df['weaponsAcquired'] >= 60].head()


# In[70]:


# Remove outliers
df.drop(df[df['weaponsAcquired'] >= 60].index, inplace=True)


# In[71]:


# Supply cheaters
```

```
plt.figure(figsize=(12,6))
sns.distplot(df['heals'], bins=10)
plt.show()


# In[72]:


df[df['heals'] >= 50]


# In[73]:


# Remove outliers
df.drop(df[df['heals'] >= 50].index, inplace=True)


# In[74]:


df.shape


# In[75]:


y=df['winPlacePerc']


# In[76]:


df = df.drop(df.columns[[0, 1, 2]], axis=1)


# In[79]:


x = df.drop(['winPlacePerc'],axis=1)


# In[80]:


x.head()


# ## Feature Selection

# In[95]:


from sklearn import model_selection
from sklearn.model_selection import train_test_split


# In[82]:


from sklearn.linear_model import LinearRegression


# In[93]:


from sklearn import feature_selection


# In[104]:
```

```
import warnings
warnings.filterwarnings('ignore')


# In[105]:


# Feature selection
LR = LinearRegression()

percentiles = range(1, 100, 5)
results = []
for i in range(1, 101, 5):
    fs = feature_selection.SelectPercentile(feature_selection.f_regression, percentile=i)
    cm_fs = fs.fit_transform(x, y)
    scores = abs(model_selection.cross_val_score(LR, cm_fs, y, cv=5, scoring='neg_mean_absolute_error'))
    print(i,scores.mean())
    results = np.append(results, scores.mean())


# In[106]:


cm_feature = x.columns


# In[107]:


optimal_percentile = int(np.where(results == results.min())[0])
print("Optimal percentile of features:{0}".format(percentiles[optimal_percentile]), "\n")
optimal_num_features = int(percentiles[optimal_percentile]*len(cm_feature)/100)
print("Optimal number of features:{0}".format(optimal_num_features), "\n")


# In[108]:


import pylab as pl
pl.figure()
pl.xlabel("Percentage of features selected")
pl.ylabel("Cross validation accuracy")
pl.plot(percentiles, results)


# In[114]:


fs = feature_selection.SelectPercentile(feature_selection.f_regression, percentile=96)
x_fs = fs.fit_transform(x, y)
for i in range(len(cm_feature)):
    if fs.get_support()[i]:
        print(cm_feature[i],'\t', fs.scores_[i] )


# In[122]:


from sklearn import preprocessing


# In[123]:


min_max_scaler = preprocessing.MinMaxScaler(copy=True, feature_range=(0,1)).fit(x_fs)
data_norm = min_max_scaler.transform(x_fs)
data_norm[0]
```

```
# In[124]:


x_train_fs, x_test_fs, y_train_fs, y_test_fs = train_test_split(data_norm, y, test_size=0.33, random_state=478)


# In[125]:


LR = LinearRegression()
LR.fit(x_train_fs, y_train_fs)


# In[128]:


from sklearn import metrics


# ## Feature Selection with Linear Regression Result

# In[126]:


y_pred_fs = LR.predict(x_test_fs)


# In[129]:


# MAE
print(metrics.mean_absolute_error(y_test_fs, y_pred_fs))
# MSE
print(metrics.mean_squared_error(y_test_fs, y_pred_fs))
# RMSE
print(np.sqrt(metrics.mean_squared_error(y_test_fs, y_pred_fs)))


# In[127]:


print ('Regression Coefficients: \n', LR.coef_)


# ## Feature Selection with Random Forest Regression Result

# In[164]:


RFR = RandomForestRegressor(n_estimators=40, min_samples_leaf=3, max_features='sqrt',
                           n_jobs=-1)
RFR.fit(x_train_fs, y_train_fs)
y_pred = RFR.predict(x_test_fs)


# In[165]:


# MAE
print(metrics.mean_absolute_error(y_test_fs,y_pred))
# MSE
print(metrics.mean_squared_error(y_test_fs,y_pred))
# RMSE
print(np.sqrt(metrics.mean_squared_error(y_test_fs,y_pred)))


# # PCA

# In[5]:
```

```
from sklearn import decomposition


# In[133]:


# PCA
pca = decomposition.PCA(n_components=5)
data_trans = pca.fit(data_norm).transform(data_norm)
print(data_trans)


# In[134]:


print(pca.explained_variance_ratio_)
print(sum(pca.explained_variance_ratio_))


# In[135]:


# pca = decomposition.PCA(n_components=6)
# data_trans = pca.fit(data_norm).transform(data_norm)

# print(pca.explained_variance_ratio_)
# print(sum(pca.explained_variance_ratio_))


# In[136]:


min_max_scaler = preprocessing.MinMaxScaler(copy=True, feature_range=(0,1)).fit(data_trans)
data_norm = min_max_scaler.transform(data_trans)
data_norm[0]


# In[137]:


x_train_pca, x_test_pca, y_train_pca, y_test_pca = train_test_split(data_norm, y, test_size=0.33, random_state=478)


# In[138]:


LR = LinearRegression()
LR.fit(x_train_pca, y_train_pca)
y_pred_pca = LR.predict(x_test_pca)
print ('Regression Coefficients: \n', LR.coef_)


# In[162]:


RFR = RandomForestRegressor(n_estimators=40, min_samples_leaf=3, max_features='sqrt',
                        n_jobs=-1)
RFR.fit(x_train_pca, y_train_pca)
y_pred = RFR.predict(x_test_pca)


# ## PCA with Linear Regression Results

# In[139]:


# MAE
print(metrics.mean_absolute_error(y_test_pca, y_pred_pca))
```

```
# MSE
print(metrics.mean_squared_error(y_test_pca, y_pred_pca))
# RMSE
print(np.sqrt(metrics.mean_squared_error(y_test_pca, y_pred_pca)))


# ## PCA with Random Forest Regression Results

# In[163]:


# MAE
print(metrics.mean_absolute_error(y_test, y_pred))
# MSE
print(metrics.mean_squared_error(y_test, y_pred))
# RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))


# ## Raw Data Results

# In[144]:


x.shape


# In[145]:


min_max_scaler = preprocessing.MinMaxScaler(copy=True, feature_range=(0,1)).fit(x)
data_norm = min_max_scaler.transform(x)
data_norm[0]


# In[146]:


x_train, x_test, y_train, y_test = train_test_split(data_norm, y, test_size=0.33, random_state=478)


# In[147]:


LR = LinearRegression()
LR.fit(x_train, y_train)
y_pred = LR.predict(x_test)
print ('Regression Coefficients: \n', LR.coef_)


# ## Raw Data with Linear Regression Results

# In[148]:


# MAE
print(metrics.mean_absolute_error(y_test, y_pred))
# MSE
print(metrics.mean_squared_error(y_test, y_pred))
# RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))


# In[149]:


from sklearn.ensemble import RandomForestRegressor


# ## Raw Data with Random Forest Regression Results
# In[166]:


RFR = RandomForestRegressor(n_estimators=40, min_samples_leaf=3, max_features='sqrt',
                            n_jobs=-1)
RFR.fit(x_train, y_train)
y_pred = RFR.predict(x_test)


# In[153]:


# MAE
print(metrics.mean_absolute_error(y_test, y_pred))
# MSE
print(metrics.mean_squared_error(y_test, y_pred))
# RMSE
print(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```