**CISC7201 Introduction to Data Science Programming**

**Group Project - PUBG Final Placement Prediction**

**Group P**

**M-B9-5531 (Kenneth Stewart)**

**M-B9-5482 (Lindsay Ding)**

**M-B9-5494 (Sammy Cheong)**

**Abstract**

The objective of this project is to predict the final placement of a well-known survival game called “Player Unknown’s Battlegrounds” (PUBG). Based on a large number of anonymized PUBG game stats provided by Kaggle, we would like to find out what are the good strategies or elements can make the player got higher rank (winPlacePerc) in the “Solo” game. We will use Numpy & Pandas to do the data cleaning, then use Matplotlib & Seaborn to visualize the data and analyze it and lastly use Statsmodels.api & Scikit-learn 4 different regression models for machine learning. The result is promising with more than 89% R-square and the error % is just around +-5.6%. Our finding is sitting in one spot and hide your way won’t bring you into the victory, you must keep moving to collect more weapons, equip better armors and get more supplement (heals & boosts), take the first moving advantage and find a good spot to give a sudden strike to other players will lead you to the “Chicken Dinner!”

**Introduction**

**Data Source:** <https://www.kaggle.com/c/pubg-finish-placement-prediction/data>

You are provided with many anonymized PUBG game stats, formatted so that each row contains one player's post-game stats. The data comes from matches of all types: solos, duos, squads, and custom; there is no guarantee of there being 100 players per match, nor at most 4 player per group.

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

**File :** train\_V2.csv (628MB)

**Data fields**

• Id - Player’s Id

• 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.

• matchId - ID to identify match. There are no matches that are in both the training and testing set.

• 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.

• DBNOs - Number of enemy players knocked.

• headshotKills - Number of enemy players killed with headshots.

• heals - Number of healing items used.

• killPlace - Ranking in match of number of enemy players killed.

• killPoints - Kills-based external ranking of player. (Think of this as an Elo ranking where only kills matter.) If there is a value other than -1 in rankPoints, then any 0 in killPoints should be treated as a “None”.

• kills - Number of enemy players killed.

• killStreaks - Max number of enemy players killed in a short amount of time.

• longestKill - Longest distance between player and player killed at time of death. This may be misleading, as downing a player and driving away may lead to a large longestKill stat.

• matchDuration - Duration of match in seconds.

• matchType - String identifying the game mode that the data comes from. The standard modes are “solo”, “duo”, “squad”, “solo-fpp”, “duo-fpp”, and “squad-fpp”; other modes are from events or custom matches.

• maxPlace - Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.

• numGroups - Number of groups we have data for in the match.

• 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.

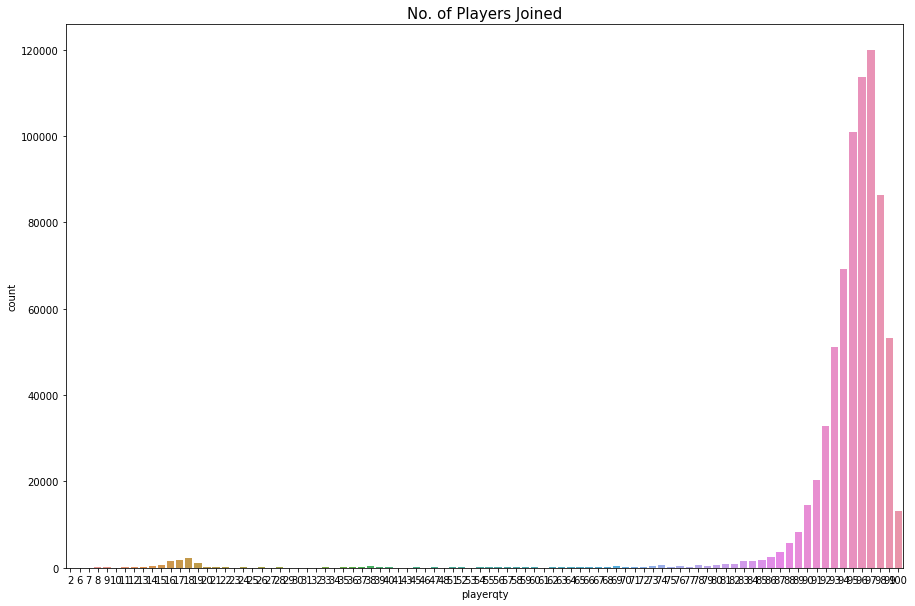
• weaponsAcquired - Number of weapons picked up.

• 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”.

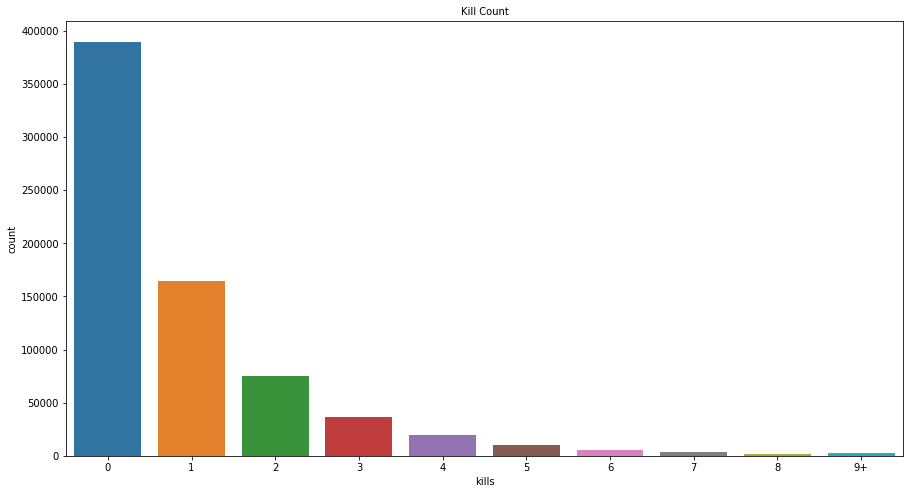
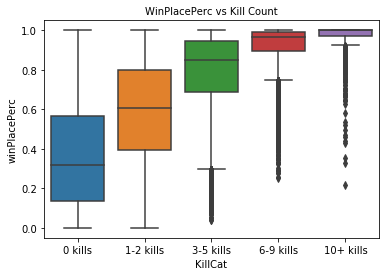
• winPlacePerc - The target of prediction. This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match. It is calculated off of maxPlace, not numGroups, so it is possible to have missing chunks in a match.

**Data Cleaning**

1. Before we get started, we use **train.info()** to preview the data structure, sample size, columns data type and memory usage.
2. After that, **train.dropna(inplace=True)** to remove records with null value.
3. Then use **train.describe(include.np.number).drop(‘count’).T** to have a glance of the basic stats about that columns, it speeds up the process to screen out the weird data and have a basic understanding about the data.
4. Use **train.groupby('matchId')['matchType'].first().value\_counts().plot.bar()** to plot out the sample size of different match type. It included 47,964 unique matches in total, but this project we will just focus on “solo” game only
5. Since “solo” matches are our target, we drop out some variables which are not meaningful to the study and only appear in the ‘duo’ and ‘squad’ games including ‘assists’, ‘DBNOs’, ‘revives’, ‘teamKills’.
6. Not every match has 100 players joined, according to the histogram chart, we decided to keep matches with more than 50 players.

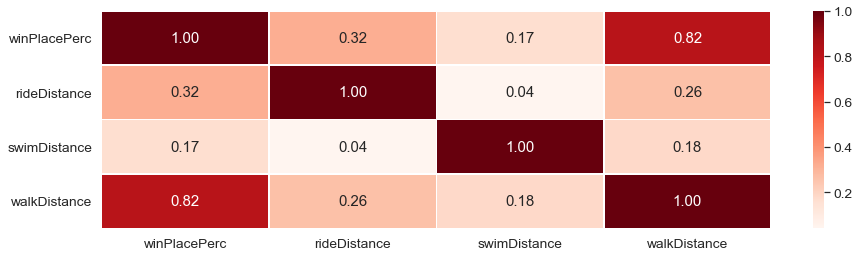


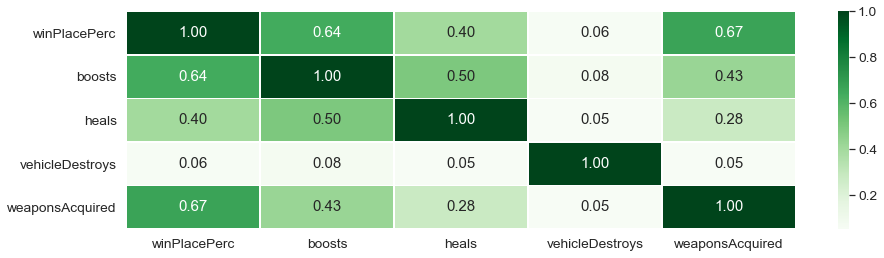
1. After applied above conditions, there is 7,477 matches left with 709,343 players joined and the file size is trimmed to around 100MB
2. We found that around 50% of players got killed before they kill somebody else and players with more Kills count tend to have higher rank in the ‘winPlacePerc’.

1. We have divided into 3 different categories, which are ‘Damage & Kills’, ‘Moving’, and ‘Items’, to see the correlation between ‘winPlacePerc’ and related attributes. For those correlation visualizations, we use Seaborn.heatmap to show the degree of coefficient of each attribute with others.
   1. **Damage & Kills** included 'killPlace', 'damageDealt', 'headshotKills', 'kills', 'killStreaks', 'longestKill', 'roadKills'.
   2. **Moving** included 'rideDistance', 'swimDistance', 'walkDistance'
   3. **Items** included 'boosts', 'heals', 'vehicleDestroys', 'weaponsAcquired'

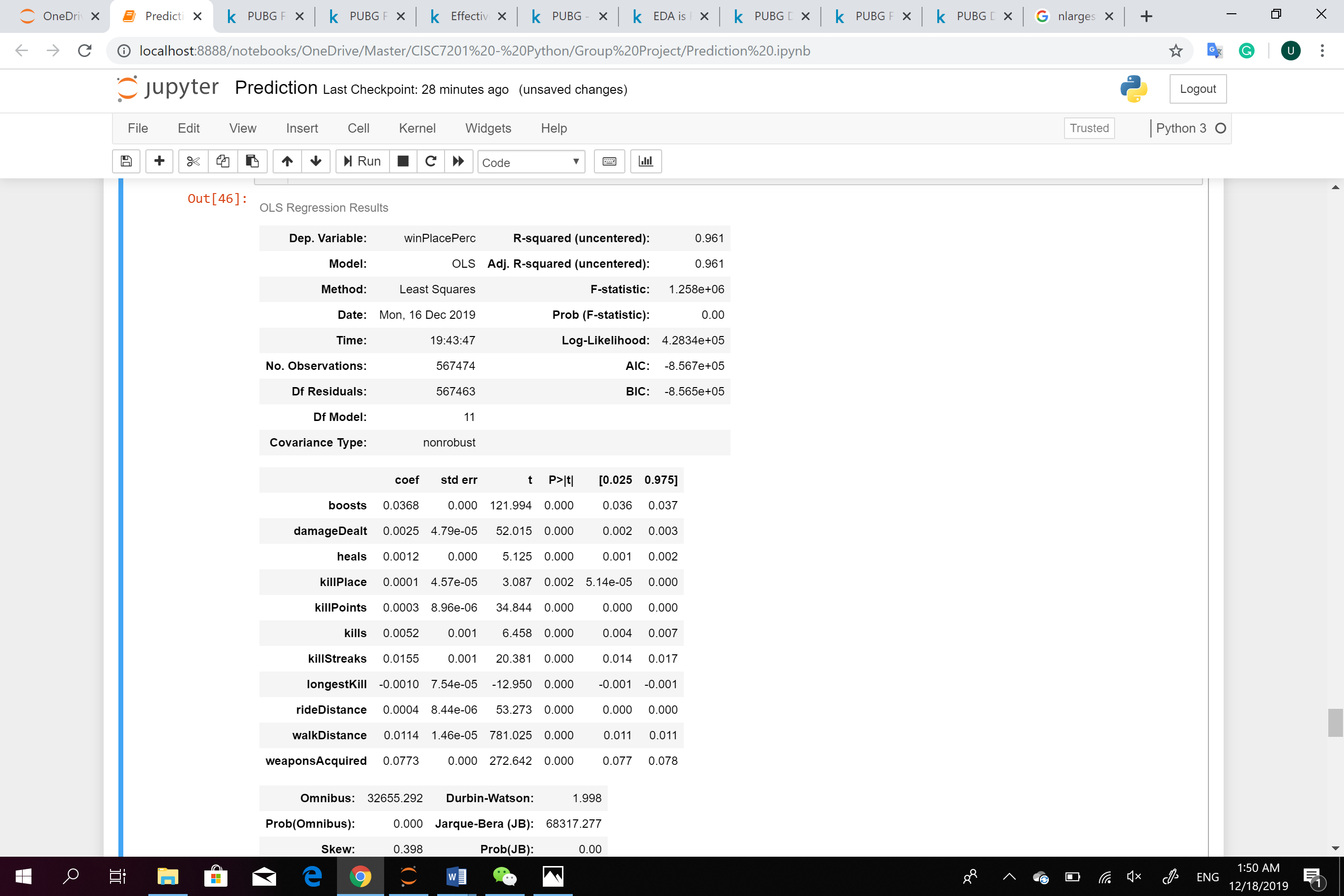






We found that below attributes are highly positive or negative correlated to ‘winPlacePerc’. The ‘walkingDistance’ got the highest coefficient 0.82 in the heatmap, meaning people might walk longer distance might help him to survive longer in the game. Followed by ‘killPlace’ with highest negative correlation to ‘winPlacePerc’, it is due to the nature of the variable ‘killPlace’ which more players you killed, you will get ‘less’ place but ‘winPlacePerc’ close to 1 means that player is the winner or finals. Other than that, ‘weaponsAcquired’, ‘boosts’, and ‘damageDealt’ also got high score in the chart. But some attributes like ‘swimDistance’, ‘vehicleDestroys’ and ‘roadKills’ are very close to zero, meaning less related to the final placement.

**Prediction Model**

1. Based on above data analysis, we have selected 10 attributes for machine learning. Those variables with less correlation are get rid of the model. It will improve the accuracy and R2 of the regression models that we will mention in the following.
2. Since some of attributes contains zero, we have to add 0.0000001 to make sure they only have some values. Besides that, we use square root to apply to all attributes, it is kind of transformation process applied to skewed data in order to make it more suitable for model training.
3. We split 20% of data for test the model and the other 80% of data is used into training
4. Use **statsmodels.api OLS()** function to check that dependent variables have significant contribution to the model and check for multicollinearity. The results were also used to pre-judge the accuracy of our model using different combinations of variables.
5. Then we applied 4 different regression models from Scikit-learn:
   1. Linear Regression
   2. RidgeCV
   3. AdaBoostRegressor
   4. GradientBoostingRegressor
6. Because scikit-learn outputs only a normal R2 value, we created a function that would compute the adjusted R2 value using the shape of the test set. This allowed us to more accurately measure the accuracy of our models.
7. The first two models were chosen because they use standard least-squares regression algorithms, and are a good benchmark for prediction performance. We used Ridge Regression specifically because we suspected that our model might be affected by multicollinearity issues (which Ridge regression minimizes using regularization), however we found that the R2 of the regular OLS model and Ridge Regression model were almost identical. The Ridge Regression class we used in Scikit-Learn included multi-candidate regularization coefficient (hyperparameter) selection.
8. In order to increase our prediction accuracy, we implemented two ensemble regression models: Adaboost and Gradient Boosting. These types of regression models are popular in Kaggle competitions, though they are often not practical for real-world applications since they are often not computationally efficient. Surprisingly, the adjusted R2 value for Adaboost was actually lower than regular OLS. However, the adjusted R2 from our Gradient Boosting model ended up being the highest.

OLS: 0.9121269081135818

Ridge: 0.9121268759232596

AdaBoost: 0.893025385071399

Gradient Tree: 0.9323599300593886

1. Then we use this model and apply to the test dataset to predict ‘winPlacePrec’. After we cross-check with the actual placement and calculate the error of prediction. The Mean Absolute Error of test dataset prediction is +-5.6%.

**Conclusion & Discussion**

The winning formula does exist! According to our study, the best strategy of playing PUBG is kept moving in the game, it is because this game require players to explore in the map and find out the good weapons, high protection armor and equipment like heals, boosts, grenades, etc. Besides that, if you hide in one spot and do not move to the next safety zone in advance, once the blue zone starts, you might expose in the danger either being killed by blue zone or killed by snipers who move much earlier than you and found a good sniping place. Boosts and heals did help the players to run faster and prolong their play time in the game. Finally, strengthen your killing skill is important as well because the data tells us the winner is always the best killer in the game.

You might ask did we spot out the cheaters and did we exclude them for the model. We did find some players are highly suspicious like killing people without any movement, crazy driver killed more than 10 players in one game, killing people from >1000m away, really good swimmer swim 5000m in the game, or change weapons more than 80 times, etc… however, it does hurt the R-square at all since the sample size of those cheaters are less than 0.05%.