

MACHINE LEARNING PROJECT REPORT

11 DECEMBER 2022

Team: LRForEverything – Anand & Ayushmaan

Mentor: Preyas Garg

Professor: Dinesh J. Babu & Neelam Sinha

Background and Basic Introduction

Player Unknown Battle Ground's (PUBG) is an online multiplayer battle royale game developed by PUBG Corporation. A player vs player action game with less than 100 players in battle royale. Players can choose to play the game alone, in pairs, or in small teams of up to four people.

However, some events and custom games have team sizes greater than 4. The last surviving person or team wins the match. Players must defeat enemies to survive to the end. During the game, players search buildings and ghost towns to find weapons, vehicles, armor and other gear.

This data is part of his Kaggle contest where data was collected from 65,000 games using APIs. The goal is to predict a player's odds of winning based on 28 given traits. For example, in a 100 player solo game if a player reaches rank 80, that player's winPlacePerc is

$$(100-80)/(100-1)=0.204$$

using the formula

$$\text{winPlacePerc}=(\text{maxPlace}-\text{winPlace})/(\text{maxPlace}-1)$$

It is basically a regression problem of predicting a player's winPlacePerc between [0,1]. This kind of problem is solved by extensive exploratory data analysis and feature engineering before applying regression models. We gained insight into features and added new related features, then crafted functions by exploring the semantics of games and attributes.

Aim

To predict the final ranking and placement from the game statistics and the initial player ratings.

The overall steps involved in the project have been listed below:

1. Data Cleaning and Exploratory Data Analysis
2. Visualizing the data and feature engineering
3. Training models on training data and then drawing comparison between them using validation data
4. Choosing the best model and fitting the test data on it
5. Reporting the accuracy of the final chosen model and also reporting other tried models as well as hyperparameters benchmarking

Problem Statement and Dataset

Provided with a large number of anonymized PUBG game stats, formatted so that each row contains one player's post-game stats, we must create a model that predicts players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place).

- The final stats data comes from several types of matches, such as solos, duos, squads and custom.
- There is no guarantee of having 100 players per match, nor at most 4 players per group.

Features in the dataset:

- DBNOs - Number of enemy players knocked.
- assists - Number of enemy players this player damaged that were killed by teammates.
- boosts - Number of boost items used.
- damageDealt - Total damage dealt. Note: Self-inflicted damage is subtracted.
- headshotKills - Number of enemy players killed with headshots.
- heals - Number of healing items used.
- Id - Player's Id
- killPlace - Ranking in match of number of enemy players killed.
- killPoints - Kills-based external ranking of player. (Think of this as an Elo ranking where only kills matter.) If there is a value other than -1 in rankPoints, then any 0 in killPoints should be treated as a "None".
- killStreaks - Max number of enemy players killed in a short amount of time.
- kills - Number of enemy players killed.
- longestKill - Longest distance between player and player killed at time of death. This may be misleading, as downing a player and driving away may lead to a large longestKill stat.
- matchDuration - Duration of match in seconds.
- matchId - ID to identify match. There are no matches that are in both the training and testing set.

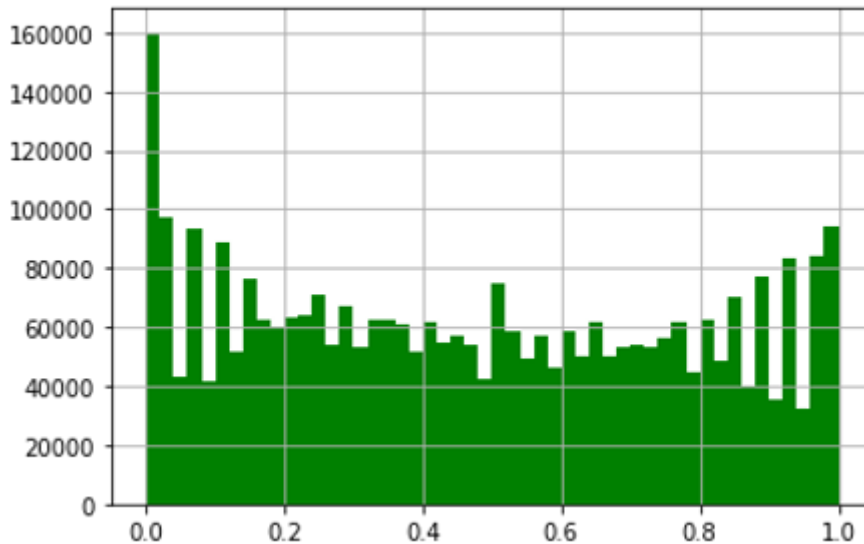
-
- matchType - String identifying the game mode that the data comes from. The standard modes are “solo”, “duo”, “squad”, “solo-fpp”, “duo-fpp”, and “squad-fpp”; other modes are from events or custom matches.
 - rankPoints - Elo-like ranking of player. This ranking is inconsistent and is being deprecated in the API’s next version, so use with caution. Value of -1 takes place of “None”.
 - revives - Number of times this player revived teammates.
 - rideDistance - Total distance traveled in vehicles measured in 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. It is calculated off of maxPlace, not numGroups, so it is possible to have missing chunks in a match.

Dataset in terms of data type

ATTRIBUTES	DESCRIPTION	DATATYPE
Id	Player's ID	Category
groupId	ID to identify group in match	Category
DBNOs	Enemy Players Knocked	uint8
Assists	Enemy Player Damaged that were killed By Teammates	uint8
Boosts	Total Boosts item used	uint8
DamageDealt	Total Damage Dealt	float16
HeadShotKills	Total headshots taken by the player	uint8
heals	No. of healing items used	uint8
killPlace	Ranking in match of number of enemy players killed	uint8
killPoints	Kills-based external ranking of player	uint8
killStreaks	Maximum enemy killed in a short duration	uint8
kills	No. of enemy player killed	uint8
longestKill	Longest distance between player and player killed	float16
numGroups	Total groups in a match	uint8
matchDuration	Duration of match in seconds	uint8
matchId	The Match ID to trace match	Category
matchType	Types of match like solo, duo etc.	Category
maxPlace	The max rank a team/player got in that match	uint8
rankPoints	Elo-like ranking of player	uint8
revives	No. of times player revived teammates	uint8
rideDistance	Total distance covered in the form of ride	float16
roadKills	No. of kills while in a vehicle	uint8
swimDistance	Total distance in form of swimming	float16
teamKills	Total teammates killed in a game	uint8
vehicleDestroys	Total vehicle destroyed	uint8
walkDistance	Total distance walked in a game	float16
weaponsAcquired	Total weapons aquired throughout the game	uint8
winPoints	Win-based external ranking of player.(Elo-like)	uint8
winPlacePerc	The winning percentage of a player	float16

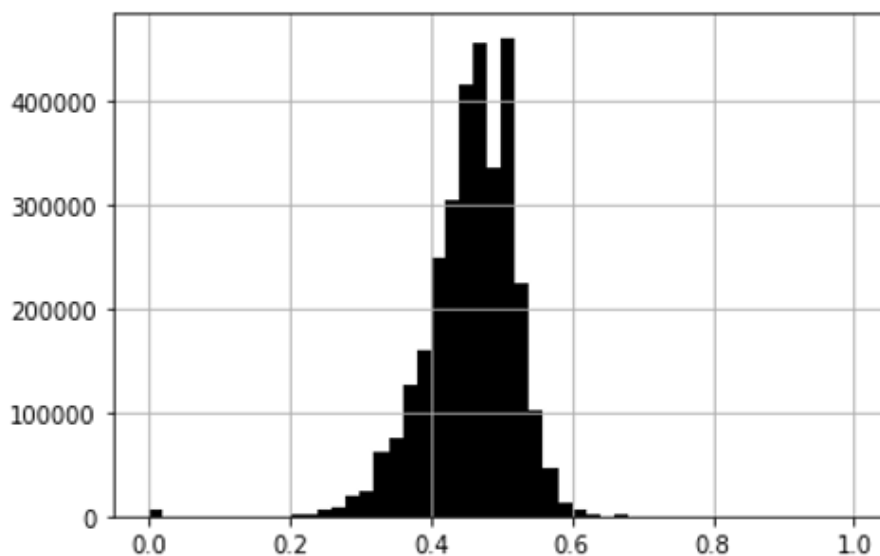
EDA and Visualizing Data

- Histogram for win percentile



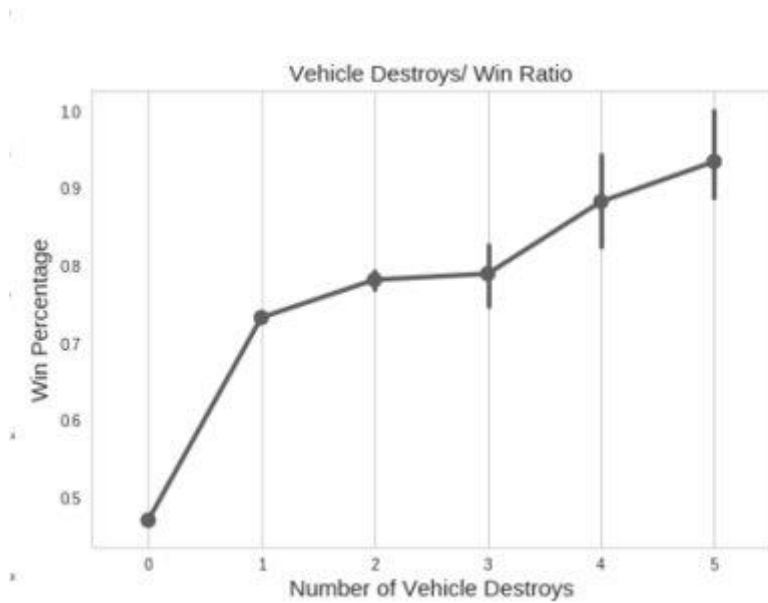
Observation: Win percentile is not uniformly distributed. Higher density at the lower end.

- Histogram for median win percentile in each match

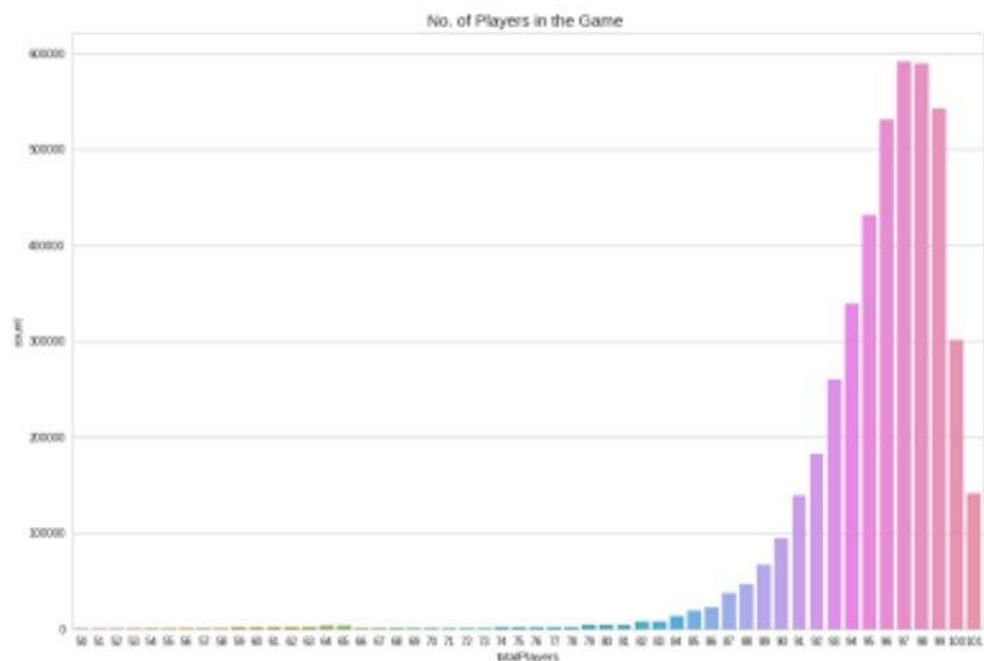


Observation: Similar to the mean distribution, the median distribution also points to the same observation.

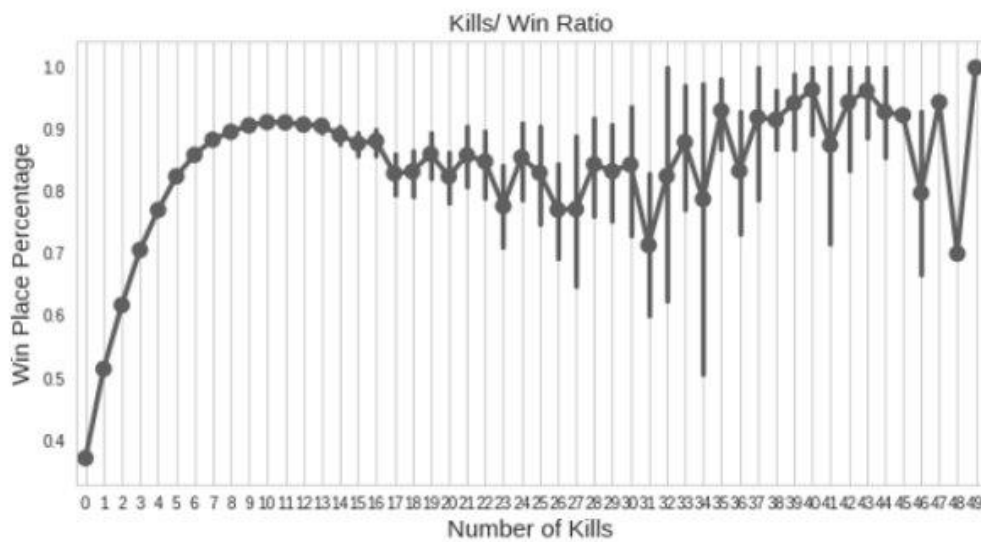
- How vehicle destroys correlate with win percentile



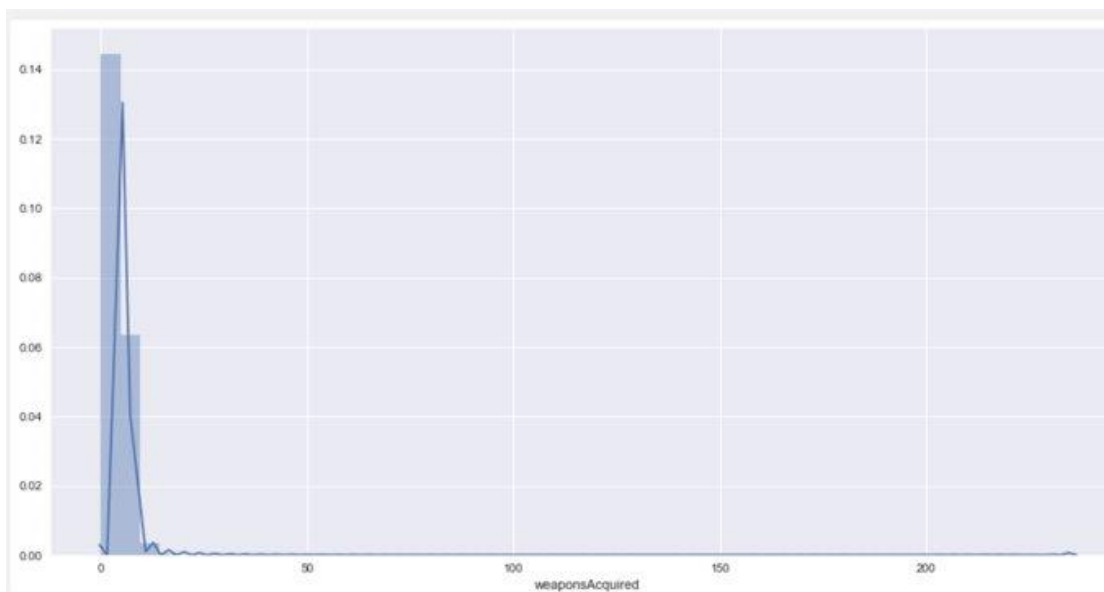
- Number of players in matches



- Kills correlation with win percentiles
-

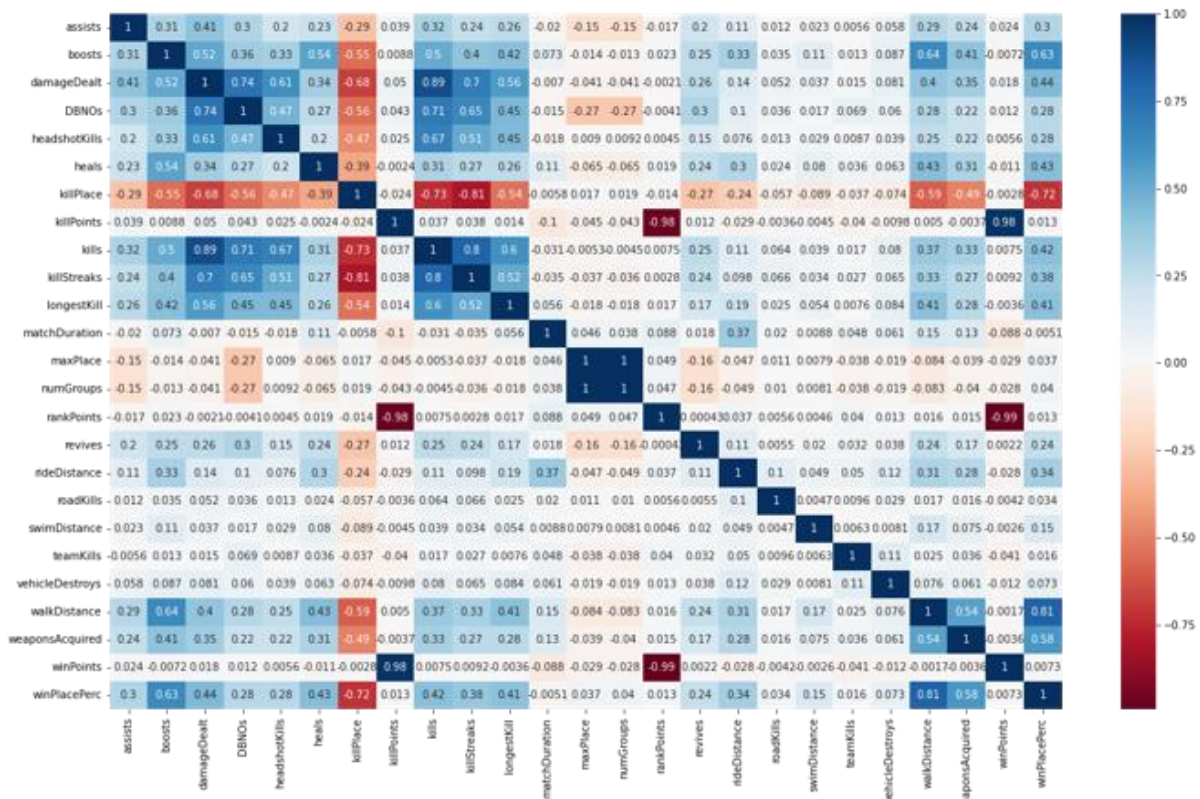


- Detecting bot players using weapons acquired data



Observation: Since there are a small number of players who acquire an abnormal number of weapons, they are considered outliers and removed from the dataset.

- Heatmap of correlation values



Observation: Highest correlation values are observed in:

- walkDistance
- killPlace
- boosts
- WeaponsAcquired

Feature Engineering

We analyzed the data and engineered more features from the already existing features which we believe, could help us achieve a better fit for our models. We create a **totalPlayers** feature by grouping data on **matchId** giving us the total number of players who played in that match. This is useful for normalizing data for different number of players in different matches. Similarly, we also group the data by **groupId** to get actual **teamSize** feature of that player, since it is not necessary that a squad game will have all teams of 4. Since distance travelled has a high correlation with the **winPlacePerc**, we add a new feature **totalDistance** which is a sum over the **swimDistance**, **rideDistance** and **walkDistance** of the player. Next, we create **maxPossibleKills** which calculates the max number of kills the team can score. For example, in a match of 100 players, if the player is in a team of 4 players, then maximum players they can kill is $100 - 4 = 96$. **itemsUsed** is the combination of all the items used like boosts, heals and weapons, since number of items used might indicate the player stayed longer in the game and hence might contribute to **winPlace**. We also get the rate/frequency of **kills**, **damageDealt** and **itemsUsed** by dividing them by **totalDistance** to get **itemsPerDistance**, **killsPerDistance** and **damageDealtPerDistance**. Next, we find team statistics like **maxTeamKills** - the maximum kills a teammate of the player has performed, **totalTeamKills** - total kills the player's team has performed combined, **itemsUsedPerTeam** - total items the team has used combined, **percTeamKills** - ratio of kills the team performed to the maximum possible kills, **meanTeamKillPlace** - mean of all teammates rank for killing which essentially finds the overall performance of the whole team in general. Similarly, we also find **percKill** - the ratio of kills performed by player divided by maximum possible kills and **headshotKillRate** which gives the skill of the player to kill an opponent player by a single headshot. There are some more features such as group strength, match_group rank, etc. that have been made, which are the features needed to assign the power of a group and its comparison to other groups which are playing in that match because the power of a team is not decided by a single player. That is why we have taken the mean strength of all the players in the group on the basis of parameters such as **killPlace**, **walkDistance**, **boosts** and **heals**, etc. There are some more features which are particularly helpful in determining winning percentile of a player such as average speed of killing, group team kills,

and more. Further, all of the features that has been made by us in due journey have been listed below.

Initial ideas:

- **total_distance** = all types of distances covered by the player. (walkDistance + rideDistance + swimDistance)
- **totalPlayers** = grouping all matches and counting number of players.
- **groupSize** = finding number of players in each group identified by the groupId.
- **headshotKills_over_kills** = since consistent headshot kills need a significant skill level, it was considered a good feature to predict win rate. (headshotKills/kills)
- **boost_heals** = number of boosts used per heal item. (boosts/heals)
- **group_strength** = using mean of killPlace stats for each group, a new feature is created that approximates every group's strength.
- **itemsUsed** = all offensive items used, with defensive items (like heals) reducing this feature's value. (boosts/heals + weaponsAcquired)
- **killPlace_over_kills_rank** = how each player in a match relates his kill rank to the number of kills scored by the player.
- **match_group_rank** = for each matchId, sort every group by their group_strength defined above, hence achieving a ranking of each group in a match.

Advanced features (obtained from further trials):

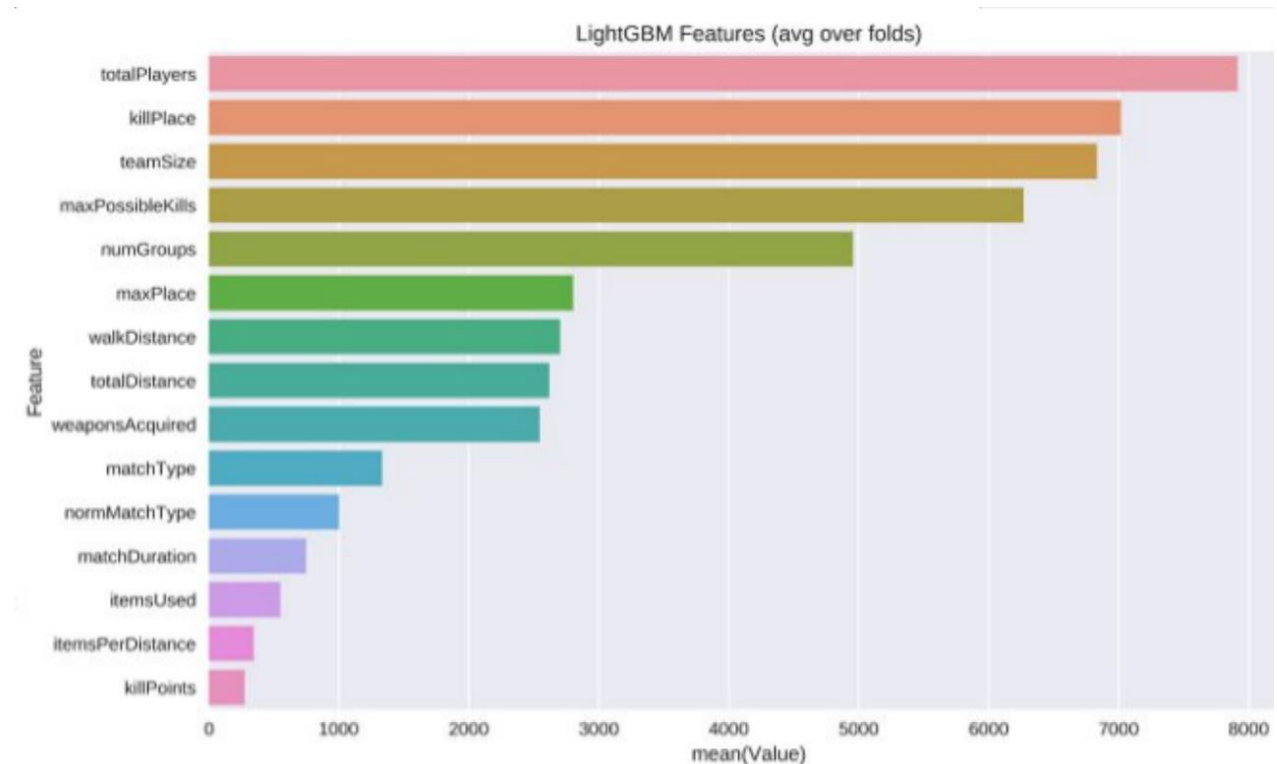
- **killRate** = number of kills obtained in the match duration.
(kills/matchDuration)
- **DBNOrate** = number of DBNOs obtained in match duration.
(DBNOs/matchDuration)
- **killPlacePerc** = how a player's killPlace compares to the match lobby.
(killPlace/totalPlayers)
- **ratioMatchKillPoints** = ratio of every player's kill points compared to the best kill points score of a player in the match.
- **ratioGroupKillPoints** = ratio of every player's kill points compared to the best kill points score of a teammate in the group.
- **ratioMatchWinPoints** = ratio of every player's win points compared to the best win points score of a player in the match.
- **ratioGroupWinPoints** = ratio of every player's win points compared to the best win points score of a teammate in the group.
- **killPointsSumMatch** = sum of all players' kill points in a match.
- **killPointsSumGroup** = sum of all players' kill points in a group.
- **ratioKillPointsGroupAndMatch** = comparison of group kill scores sum and match kill scores sum. (killPointsSumGroup/killPointsSumMatch for each player)
- **avgKillPointsGroup** = average of the kill points of the players in a group.
- **groupRevived** = number of revives used by the players in a group (for each other).
- **groupTeamKills** = number of teamKills scored by the players in a group.
- **avgSpeed** = average approximation of a player's movement speed.
(totalDistance/matchDuration)

Kaggle Scores vs Features considered (for XGBoost)

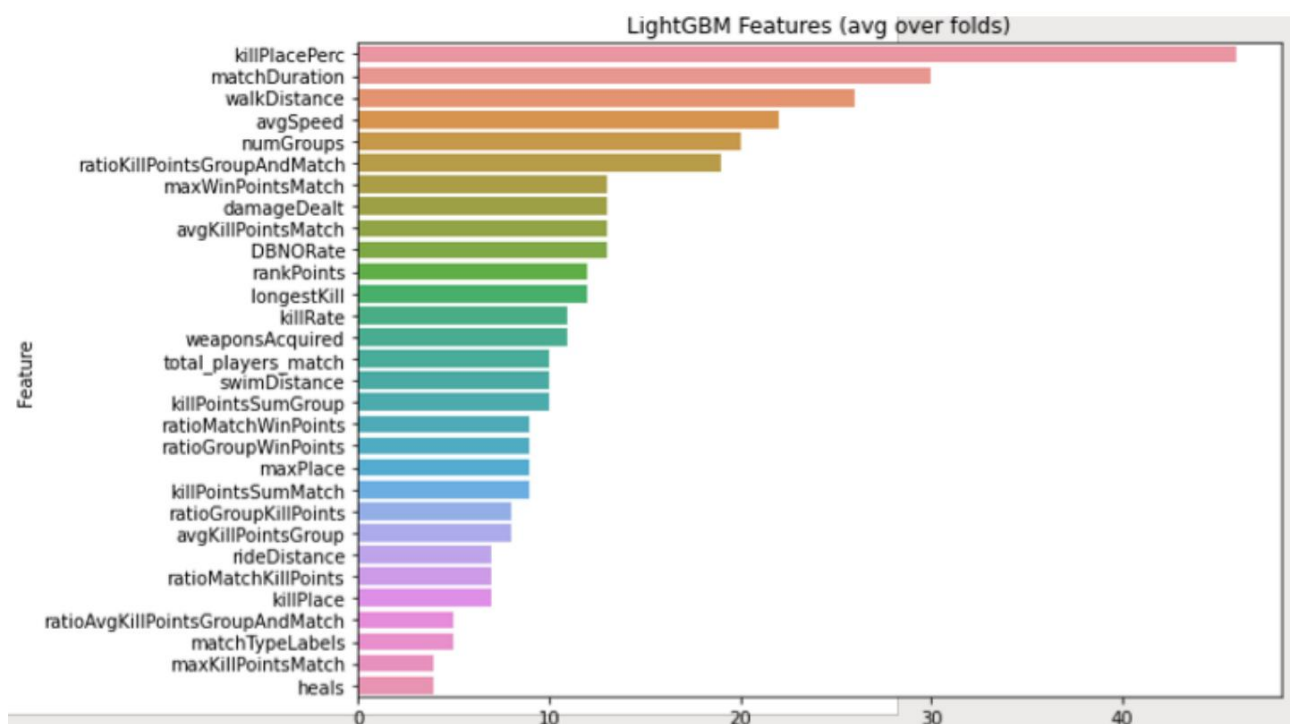
Features (added)	Kaggle Score
Without feature engineering	0.00692
TotalDistance, totalPlayers, groupSize, matchSize added	0.00679
Headshotkills_over_kills, boosts_heals, itemsUsed added	0.00668
groupStrength, groupRank added	0.00645
MatchGroupRank added	0.00608
killsRate, DBNORate added	0.00606
MatchType one-hot encoded	0.00592
KillPlacePercentage, ratioMatchKillPoints, ratioGroupKillPoints added	0.00568
ratioMatchWinPoints added	0.00562
GroupTeamKills, avgSpeed (group), groupRevives added	0.00558
avgKillPointsGroup, ratioKillPointsMatch added	0.00540 (best score)

Feature Importance

Before feature engineering:








After feature engineering:



Kaggle Submission Status

Final Standings:

#	△	Team	Members	Score	Entries	Last	Code
1	—	minus_one	 	0.00533	43	3d	
2	—	coldplay		0.00540	14	3d	
3	—	LRforEverything	 	0.00542	26	3d	

PS: 2nd position belongs to our team as well.

Past standings:

- **1st Evaluation: Rank 13**
- **2nd Evaluation: Rank 3**
- **3rd Evaluation: Rank 2**

Models Used

Non-tree models:

- **Multiple linear regression** - a linear approach for modelling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables). The mathematical expression for this is shown as follows:

Assuming that the model is

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_k X_k + \varepsilon,$$

the n -tuples of observations are also assumed to follow the same model. Thus they satisfy

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \dots + \beta_k x_{1k} + \varepsilon_1 \\ y_2 &= \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \dots + \beta_k x_{2k} + \varepsilon_2 \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \dots + \beta_k x_{nk} + \varepsilon_n. \end{aligned}$$

These n equations can be written as

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nk} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

$$\text{or } y = X\beta + \varepsilon.$$

- **Ridge regression** - a method of estimating the coefficients of multiple-regression models in scenarios where the independent variables are highly correlated.
- **Lasso regression** - a method similar to ridge regression, forcing the sum of the absolute value of the regression coefficients to be less than a fixed value, which forces certain coefficients to zero, excluding them from impacting prediction.
- **Support Vector regression** - a version of the SVM for regression, where the model produced by SVR depends only on a subset of the training data, because the cost function for building the model ignores any training data close to the model prediction.

Tree models:

We have used 3 Boosting models namely AdaBoost, Gradient Boosting and Light Gradient Boosting Machine (LGBM). Firstly, Boosting is an ensemble technique to create a strong classifier from a combination of weak classifiers. This occurs by creating a model from the train data. Then creating a second model that attempts to correct the errors made in the first model. Models are added till the training set is predicted accurately. Gradient Boosting Model is a supervised technique where the main objective is to minimize the Loss function i.e., the Mean Squared Error (MSE). It uses Gradient Descent to minimize the loss and uses a learning rate alpha to find the updated value of predicted values.

- **AdaBoost** - retrains the algorithm iteratively by choosing the training set based on accuracy of previous training set. Each training data is given a particular weight according to the accuracy achieved and the algorithm runs again.
- **LightGBM** - Light Gradient Boosted Machine is a library developed at Microsoft that provides an efficient implementation of the gradient boosting algorithm. The primary benefit of the LightGBM is the changes to the training algorithm that make the process dramatically faster, and in many cases, results in a more effective model.

LGBM is a type of gradient boosting algorithm which uses decision trees as its framework. The most inherent part of this model is that it can handle a very large size of data and takes very low memory to run. Due to its efficiency, accuracy and interpretability it has started to be widely used for very large data sets. Similarly it is not advised to use this model with small data set as it is prone to overfitting.

LGBM is an ensemble model of decision tree which learns by fitting negative gradients which are also known as residual errors. Suppose we have n identical and independent distributed (x_1, x_2, \dots, x_n) with dimension s in a Gradient Space. In every iteration of gradient boosting, the residuals of loss function with respect to output of the model are denoted by (g_1, g_2, \dots, g_n) . So the model splits each node at the point where the Information Gain is the maximum .

- **XGBoost** - Extreme Gradient Boosting is a library that provides an efficient implementation of the gradient boosting algorithm. The main benefit of the

XGBoost implementation is computational efficiency and often better model performance.

It became popular in the recent days and is dominating applied machine learning and Kaggle competition for structured data because of its scalability. XGBoost is an extension to gradient boosted decision trees (GBM) and specially designed to improve speed and performance.

Some of the unique features of XGBoost are:

- **Regularized Learning:** Regularization term helps to smooth the final learnt weights to avoid over-fitting. The regularized objective will tend to select a model employing simple and predictive functions.
- **Gradient Tree Boosting:** The tree ensemble model cannot be optimized using traditional optimization methods in Euclidean space. Instead, the model is trained in an additive manner.
- **Shrinkage and Column Subsampling:** Besides the regularized objective, two additional techniques are used to further prevent overfitting. The first technique is shrinkage introduced by Friedman. Shrinkage scales newly added weights by a factor η after each step of tree boosting. Similar to a learning rate in stochastic optimization, shrinkage reduces the influence of each tree and leaves space for future trees to improve the model.

So at last we can say that XGBoost is a faster algorithm when compared to other algorithms because of its parallel and distributed computing. XGBoost is developed with both deep considerations in terms of systems optimization and principles in machine learning. The goal of this library is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate library.

Hyperparameters Benchmarking

- Linear Regression

Default Parameters	
Mean Squared Error	0.0126278
Mean Absolute Error	0.0838617

- Ridge Regression

	alpha=0.5, tol=0.0001	alpha=0.1, tol=0.0001	alpha=0.5, tol=0.1
Mean Squared Error	0.0126268	0.0126268	0.0126268
Mean Absolute Error	0.0838634	0.0838634	0.0838634

- Lasso Regression

		alpha=0.1, tol=0.001	alpha=0.5, tol=0.001	alpha=1, tol=0.001	alpha=0.5, tol=0.1	alpha=0.1, tol=0.5	alpha=0.5, tol=1
Mean Squared Error	Poor performance		0.0188105	0.0201275	0.0188152	0.0175518	0.0210605
Mean Absolute Error	Poor Performance		0.1055864	0.1095797	0.1055617	0.1008325	0.110434

- **LGBM Regression**

LR	0.3	0.1	0.3	0.3	0.01
n_estimators	250	250	250	500	500
n_leaves	200	200	500	200	1000
Mean Squared Error	0.0940065	0.093657	0.093595	0.093990	0.091910
Mean Absolute Error	0.263166	0.263566	0.262234	0.263314	0.259448

- **XGBoost Regression**

ETA	0.3	0.1	0.001	0.1	0.3
n_estimators	250	300	300	300	650
Depth	5	7	10	5	5
RMSE	0.30679	0.30489	0.30369	0.30117	0.29563

Conclusion

For this dataset, we have observed that Boosting models like LGBM, AdaBoost and Random Forest, XG boost performed better than traditional Multiple Regression models and Neural Networks. This is perhaps because boosting algorithms are very robust to noise and outliers. They optimize the model via gradient descent using generic differentiable loss functions and also has various benefits like automatic null-handling, in-built feature selection and scale invariance. Since our data is not necessarily linearly separable, multiple linear regression models performed worse. There are multiple features with non-linear relationships between them, which boosting algorithms can recognize better using the ensemble of multiple weak learners and tuning the parameters to control model complexity (avoid overfitting). Also, while neural networks are good for learning complex features at a higher level by doing automatic feature engineering at each layer which would be helpful for data involving image, text or audio where we don't have many features (just the data itself), our problem has tabular data with multiple features already given. With tabular data, often the relationship between features is shallow and there really is no need to explore complex features. As a result, we find that the boosting algorithms can learn non-linear relationships better than linear regression and work better on tabular data than neural networks. This makes them a must use approach for regression problems of this nature. We also understood the process and importance of hyperparameter optimization and achieving it through grid search. We also learned the importance of Exploratory Data Analysis and Feature engineering in Machine Learning. EDA helps us find the trends in the data and understand what type of pre-processing is needed. We can also use these insights to add new features to enhance the robustness of our model. The deep learning model used could have performed better if it was given more time to train. We could also have improved it by hyper tuning other relevant parameters to achieve better results. Even for the best performing model LGBM, a randomized grid search could have given better results than providing grid parameters. We restricted ourselves to the Kaggle problem statement of predicting the winPlacePerc for all the players in the dataset. We could have predicted other statistics like number of kills, walkDistance, headshots given previous data. This prediction can be done for the whole dataset or at a finer scale such as per team or per match.

Summary of Kaggle Score for different models and parameters

Model used and parameters	Status	Kaggle Score
Linear Regression	Before feature engineering	0.2356
Linear Regression	After feature engineering	0.1897
Ridge Regression	Before feature engineering	0.2985
Ridge Regression	After feature engineering	0.2217
Lasso Regression	Before feature engineering	0.2344
Lasso Regression	After feature engineering	0.19.1
Light GBM lr=0.01, n_est=500, n_leaves=1000	Before feature engineering	0.00705
Light GBM lr=0.01, n_est=500, n_leaves=1000	After feature engineering	0.00616
XGBoost eta=0.3, n_est=650, depth=5	Before feature engineering	0.00692
XGBoost eta=0.3, n_est=650, depth=5	After feature engineering	0.00541

Thank You.