Table Of Content

Chapter No.	Title		Page No.
1	Problem Statement		3
2	Implementation		4
	2.1	About Dataset	4
	2.2	Exploratory Data Analysis and Data Pre-processing	6
	2.3	Feature Engineering	19
3	Training Process		20
	3.1	Models Used	15
	3.2	Metric Used	22
	3.3	Parameter Tuning	23
	3.4	Best Parameters	26
4	Conclusion		27
5	References		27

1. Problem Statement:

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 ammunition, 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.

You are provided with a large number of 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 players 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).

2. Implementation:

2.1 About Dataset:

The PUBG Dataset has up to 100 players in each match which are uniquely identified based on their matchId. The players can form a team in a match, for which they will have the same groupId and the same final placement in that particular match.

The data consists of different groupings, hence the data has variety of groups based on the number of members in the team(not more than 4) and matchType can be solo, duo, squad and customs.Also the matchType can be further more classified based on the perspective mode like TPP and FPP.

Approximately there are 3 million training data points and 1.3 million testing data points. There are in total 29 features. They are summarised as follows:

Sr.No.	Feature	Type	Description
1	Id	String	Unique Id for each Player.
2	matchId	String	Id to identify matches.
3	groupId	String	Id to identify the group.
4	assists	Real	Number of enemy players this player damaged that were killed by teammates.
5	boosts	Real	Number of boost items used.
6	damageDealt	Real	Total damage dealt. Note: Self inflicted damage is subtracted.
7	DBNOs	Real	Number of enemy players knocked.
8	headshotKills	Real	Number of enemy players killed with headshots.
9	heals	Real	Number of healing items used.
10	killPlace	Real	Ranking in match of number of enemy players killed.
11	killPoints	Real	Kills-based external ranking of player.
12	kills	Real	Number of enemy players killed.
13	killStreaks	Real	Max number of enemy players killed in a

			short amount of time.
14	longestKill	Real	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.
15	matchDuration	Real	Duration of match in seconds.
16	maxPlace	Real	Worst placement we have data for in the match.
17	numGroups	Real	Number of groups we have data for in the match.
18	rankPoints	Real	Elo-like ranking of players.
19	revives	Real	Number of times this player revived teammates.
20	rideDistance	Real	Total distance travelled in vehicles measured in metres.
21	roadKills	Real	Number of kills while in a vehicle.
22	swimDistance	Real	Total distance travelled by swimming measured in metres.
23	teamKills	Real	Number of times this player killed a teammate.
24	vehicleDestroys	Real	Number of vehicles destroyed.
25	walkDistance	Real	Total distance travelled on foot measured in metres.
26	weaponsAcquired	Real	Number of weapons picked up.
27	winPoints	Real	Win-based external ranking of players.
28	matchType	Categorical	Identifies the matchType.
29	winPlacePerc	Real	This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match.

2.2 Exploratory Data Analysis and Data Pre-Processing:

❖ *Dataset* Size:

The EDA was quite interesting as the training dataset was about 3 million rows in size.The size of the training dataset was about **688.7 MB,** hence the task to handle it would have been somewhat difficult if it would have been involved in any computations.

So by looking at the datatypes of the columns, most of the types were float64 and int64, so we downcasted the datatype of all the numerical columns to as small as possible and reduced the size of the training dataset to **237.5 MB.**

dtypes: float64(6), int64(19), object(4) memory usage: 688.7+ MB CPU times: user 7.49 ms, sys: 1.02 ms, total: 8.51 ms Wall time: 7.98 ms	dtypes: float32(6), int16(5), int8(14), object(4) memory usage: 237.5+ MB CPU times: user 5.5 ms, sys: 975 µs, total: 6.48 ms Wall time: 6.23 ms
Before	**After**

Hence now the computation will be quite fast as compared to the original dataset.

Total number of null values in the dataset was only one, and it was removed. Dropped the **Id** column as it will be of no use in decision making.

❖ *matchType*:

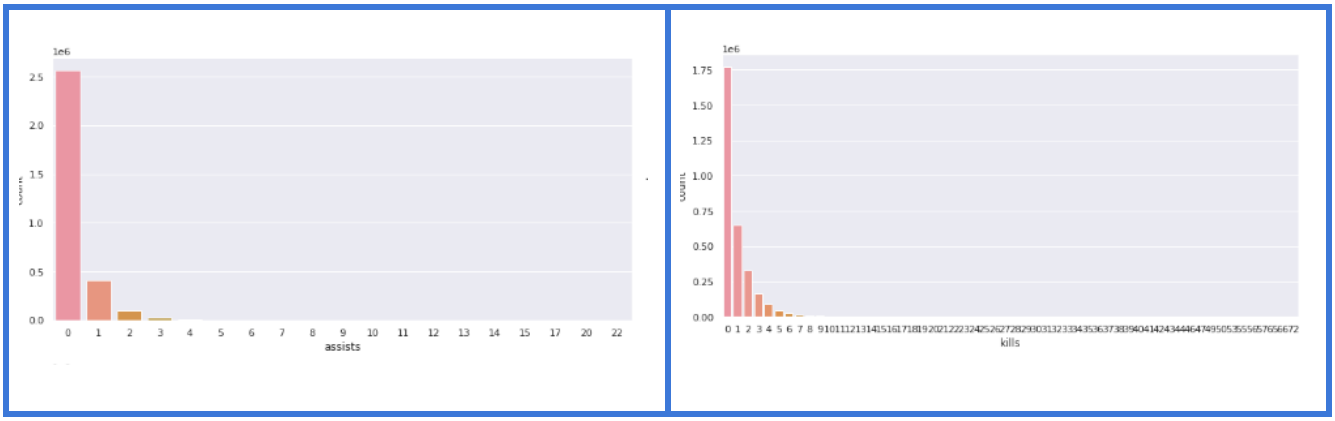
There are 16 match types as shown below with combinations of fpp, tpp, solo, duo, squad,etc.So we are generalising them into only solo, duo and squad.After that applying LabelEncoding to matchType column.

Mapping of Label Encoding: solo - 1 ; duo - 0 ; squad - 2

We will be using these encoding for the rest of our project work from now on.

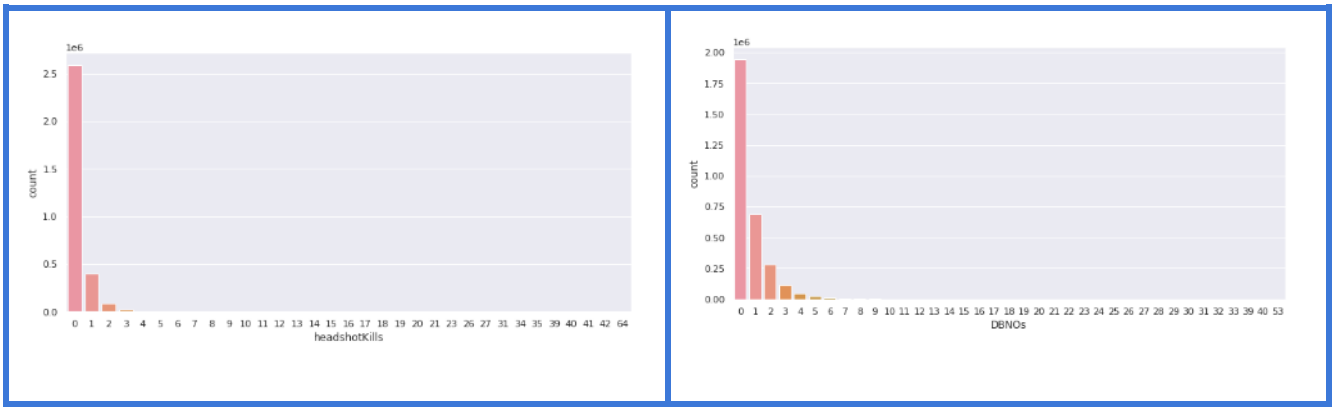
❖ ***Some features and their behaviours:***

1) ***assists and kills***: Number of assists the player has done for the team and the number of kills a player has done.From the below graphs it can be seen the count of zeros is very high but still an important feature while determining the final rank.

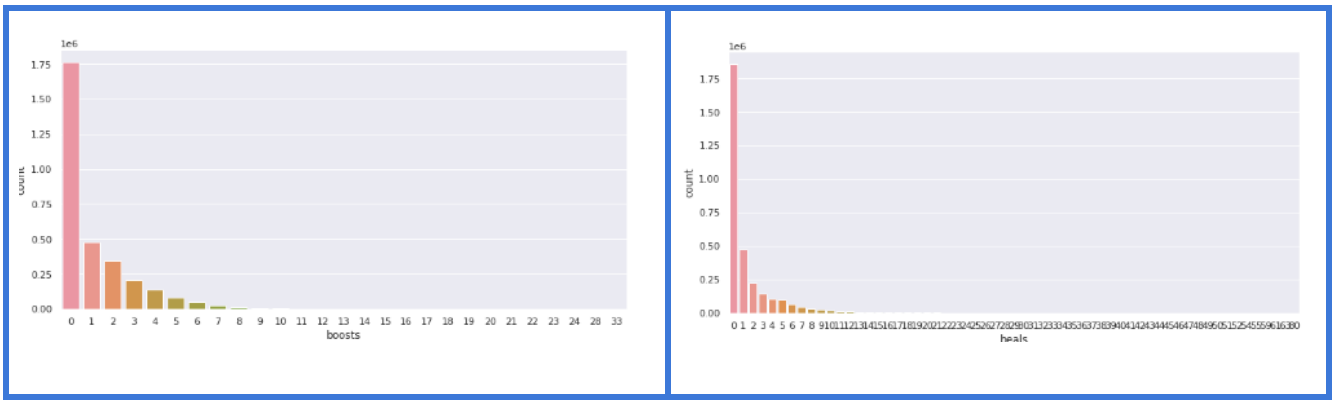


2) ***roadKills and teamKills***: Roadkills indicate the number of people killed while travelling in a vehicle whereas TeamKills indicate the number of people killed by a team member within the same team. These features seem to be useless as it is highly unlikely that this will happen which can be proven from the figures below.

3) ***headshotKills and DBNOs***: Headshot kills indicate the number of kills done by the player with headshot and DBNOs indicate the number of enemies knocked by a player. These features are important as they indicate skill of a player which can be a good metric to judge the final placement prediction of the player.



4) ***boosts and heals***: Boosts and Heals are the items which increase the health of the player in the game, boosts have an immediate effect whereas heals take longer time.However, both can be important features for further decision making.



❖ ***Analysis on Dataset*:**

According to the data provided, in a match, people with the same groupId form a group and that group has the same target placement in that match. This was according to us one of the main challenges the model faced as for the same target value, it had different feature values, leading to confusion for model learning. So, to alleviate that, I decided to group the data points based on groupId and matchId and aggregate their feature values to be represented as one row for each group in the match.

So based on the idea mentioned above we thought of representing all the players in the same team as a one entity/player/team.Hence we reduced the dataset by grouping the rows based on groupId, and now each row will represent a team or an individual in the case of solo mode.

Now what about the aggregation of the other columns, so for that we have used sum, mean and max, for e.g:

- ➔ ***kills*:** We have taken the **sum** of the kills scored by all the teammates.
- ➔ ***killPlace:*** For the *killPlace* , we have taken the **mean** of that of all players.
- ➔ ***rideDistance:*** So for the ride distance we have taken **max** of that of all players in the same team.

So the idea behind the logic of which aggregation is applied to which columns is as follows:

- ➔ ***So basically the feature which describes any teamwork we will take sum of it (e.g kills, assists).***
- ➔ ***If it's a scaling feature we'll be taking the mean of it.***
- ➔ ***If the feature describes the quality of a player in a team we'll take max of it hence his/her team gets affected positively.***

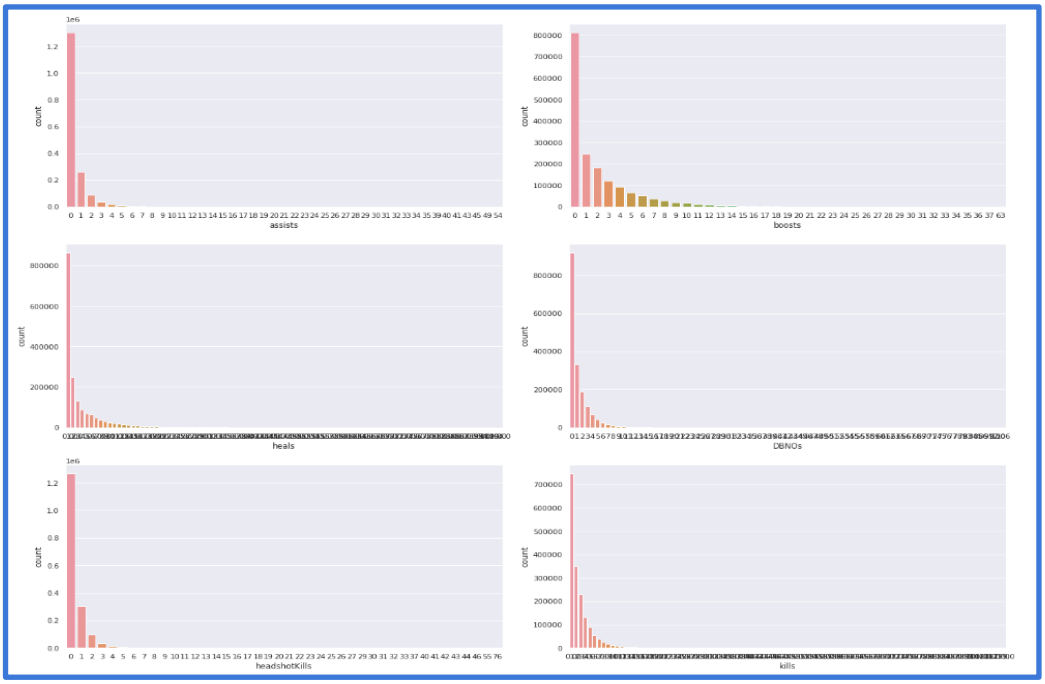
Following table shows the columns and the corresponding aggregation function which is applied to it.

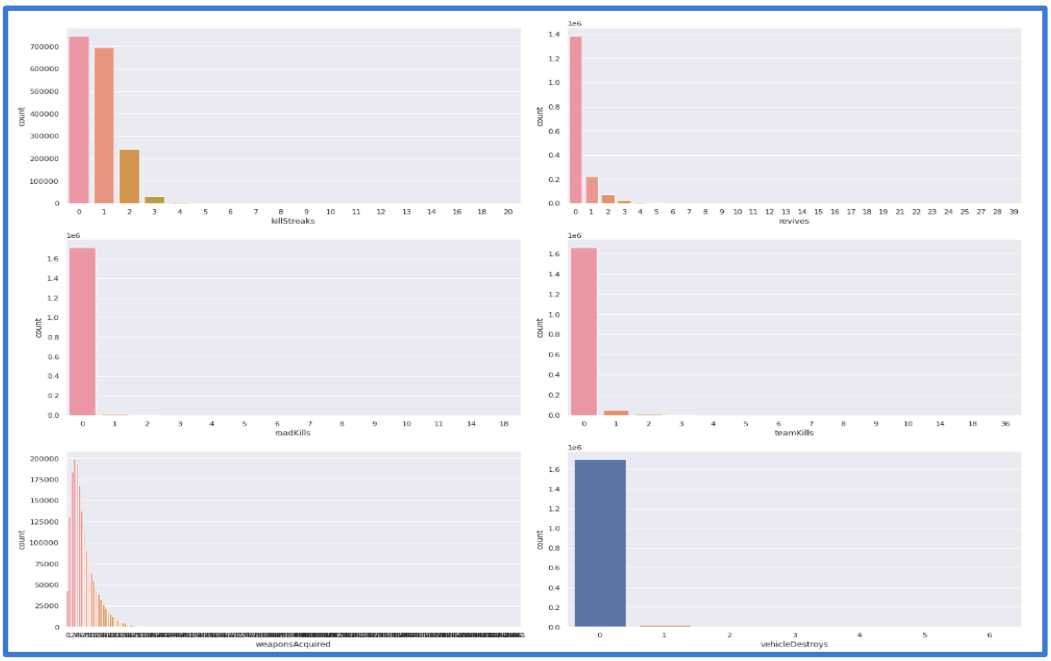
Columns	Functions	Columns	Functions
matchId	max	maxPlace	mean
assists	sum	numGroups	mean
boosts	sum	rankPoints	max
damageDealt	sum	matchType	mean
DBNOs	sum	revives	sum
headshotKills	sum	rideDistance	max

matchId	max	maxPlace	mean
assists	sum	numGroups	mean
boosts	sum	rankPoints	max
damageDealt	sum	matchType	mean
DBNOs	sum	revives	sum
headshotKills	sum	rideDistance	max
heals	sum	roadKills	sum
killPlace	mean	swimDistance	sum
killPoints	max	teamKills	sum
kills	sum	vehicleDestroys	sum
killStreaks	max	walkDistance	max
longestKill	mean	weaponsAcquired	sum
matchDuration	max	winPoints	max

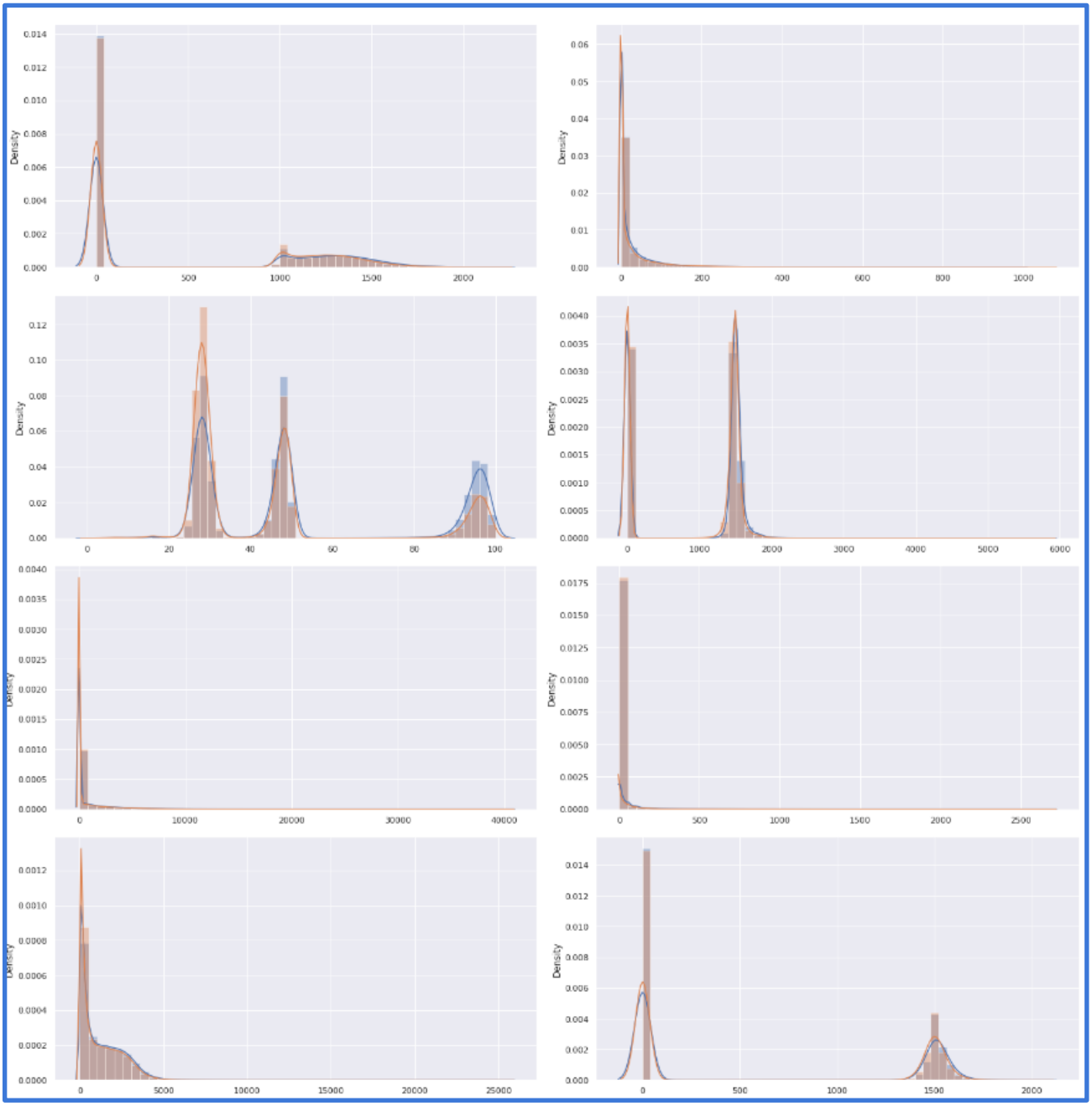
```
Memory Before : 237.49  MB
Memory After :  136.24  MB
```

Here we have significantly reduced the dataset memory, but is it legit reducing the dataset in this way ? Lets see some plots and figure out:So we plotted the discrete features and found that the distribution was similar like the original distribution.





We also plotted the continuous features for both the original dataset and the reduced one and noticed that they were also similar. Let's have a look at it.



So here both distributions are looking similar, let's have a look on how the correlation of the columns with winPlacePerc is affected.

Let's check the correlation of all the features with winPlacePerc before and after.

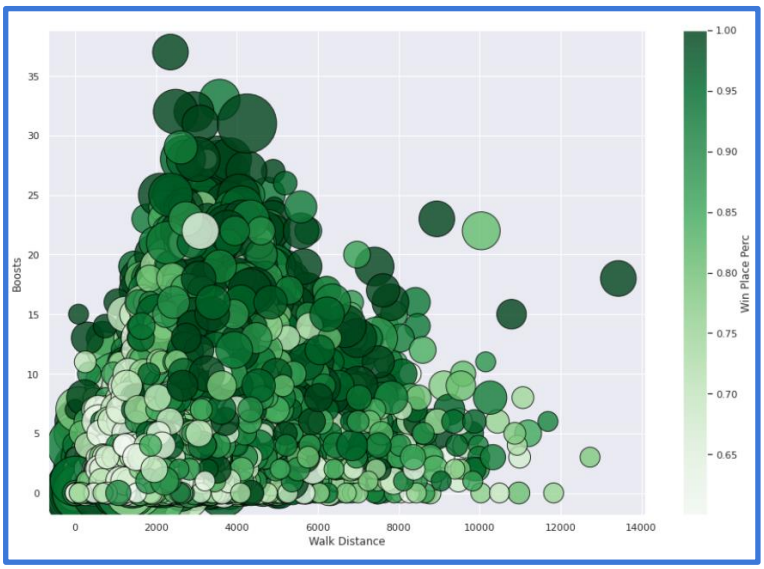
	Orginal Dataset	Reduced Dataset
assists	0.299	0.286
boosts	0.634	0.608
damageDealt	0.441	0.401
DBNOs	0.280	0.248
headshotKills	0.277	0.308
heals	0.428	0.415
killPlace	-0.719	-0.777
killPoints	0.013	0.008
kills	0.420	0.415
killStreaks	0.378	0.369
longestKill	0.410	0.459
matchDuration	-0.005	0.003
matchType	-0.036	0.002
maxPlace	0.037	-0.003
numGroups	0.040	-0.003
rankPoints	0.013	0.009
revives	0.241	0.254
rideDistance	0.343	0.338
roadKills	0.034	0.040
swimDistance	0.150	0.160
teamKills	0.016	0.005
vehicleDestroys	0.073	0.089
walkDistance	0.811	0.814
weaponsAcquired	0.583	0.371
winPoints	0.007	0.006
winPlacePerc	1.000	1.000

So as per the above table we can see there is not much of a difference between the original dataset correlation and the reduced dataset correlation of the features with winPlacePerc.

Hence from the above observation we are taking the Reduced_GroupBy dataset into consideration for the further training purpose.

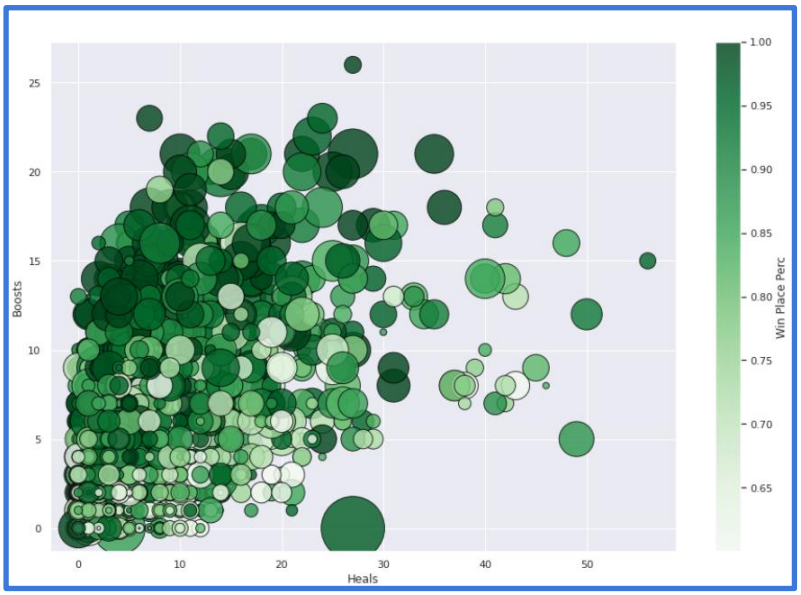
❖ ***Multivariate Analysis*:**

1) *walkDistance | boosts | kills(size of points) | winPlacePerc:*



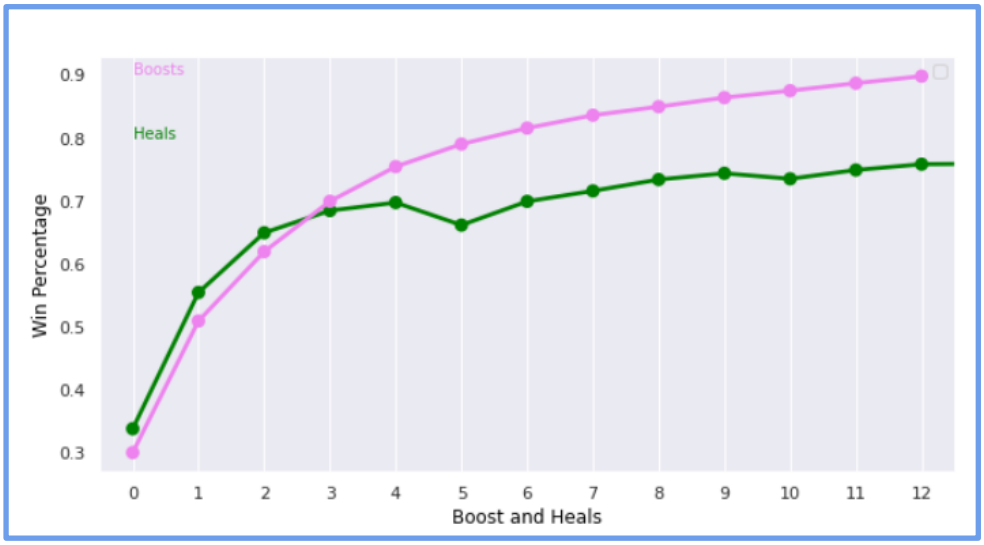
From the above graph, we can observe that as boosts consumption increases players chance of winning the match increases, also logically a player which has high chance of winning tends to be in fight and needs boost, also we can see walkDistance also matters in winning as it will be high for the player/team who has high chances of winning, because to be in the game, players have to be in safe zone for that they need to travel.

2) *heals | boosts | damageDealt(size of points) | winPlacePerc:*



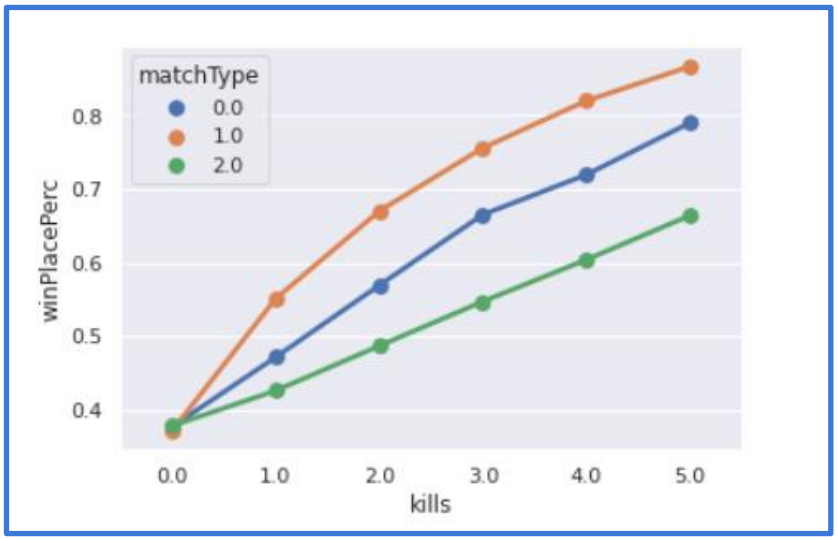
Here the above graph depicts that for high winPlacePerc, along with boosts and heals, the player having high damageDealt also has more tendency to have high winPlacePerc.

3) *boosts and heals | winPlacePerc:*



From the above graph we can see Boosts and Heals show positive relation with winPlacePerc, Boosts shows more than Heal. Maybe we can do some stuff with both of these features later.

4) *kills(matchType wise) | winPlacePerc:*

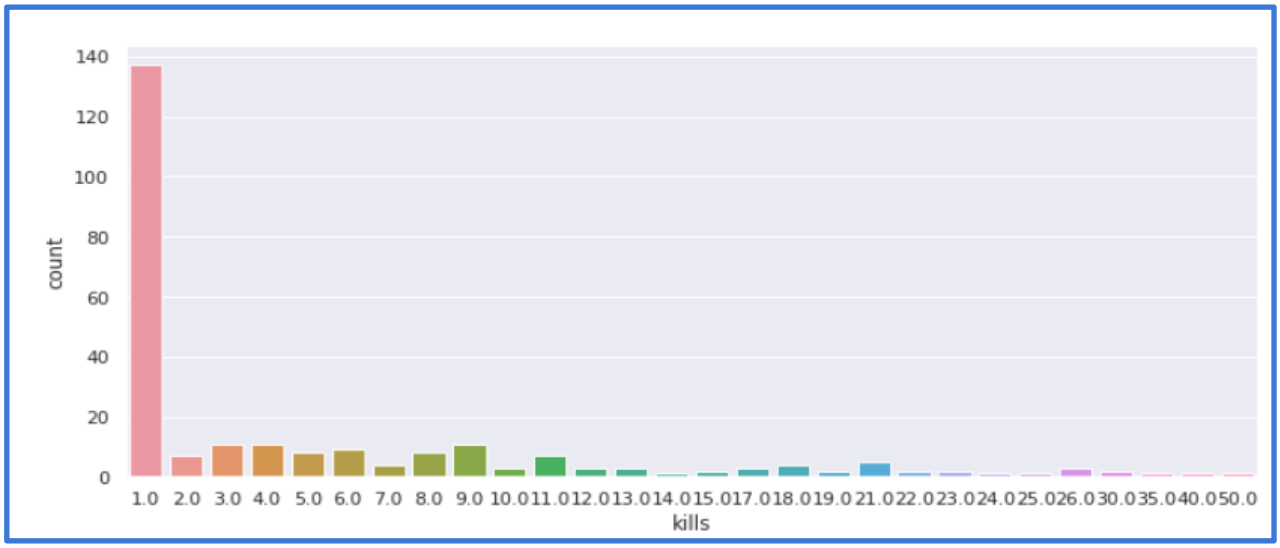


From the above graph we can say that as the number of kills increases chances of winning increases but it does not matter much as we go from match type from solo to squad, because in squad we have to play more strategically and focus is not much on kills in squad.

❖ ***Handling some Anomalies*:**

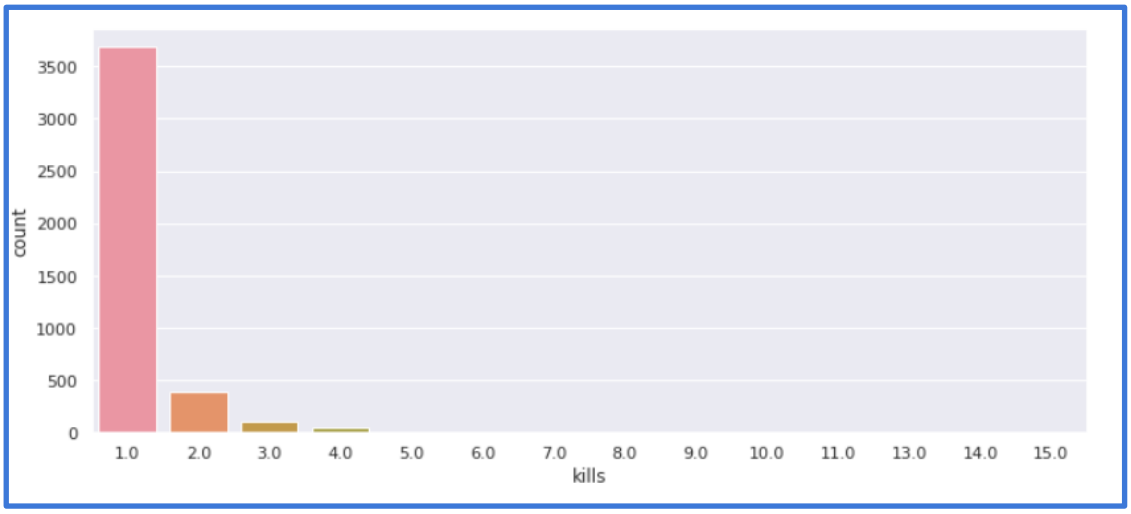
While analysing the dataset we found some irregularities in the dataset itself hence now we'll try to handle those anomalies one by one.

1. **Players have done kills without travelling any type of distance:**



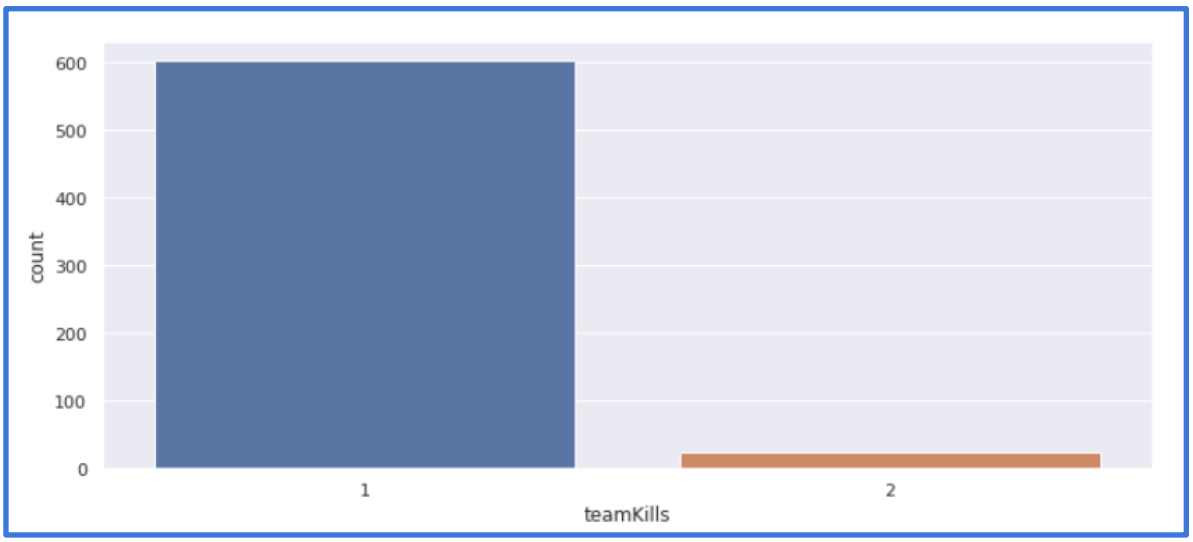
So the above graph is of the players who travel zero distance (distance = walk + ride + swim) yet they have killed enemies seems suspicious, hence removing those rows!!

2. **Longest Kill =0 metre, kill >0:**



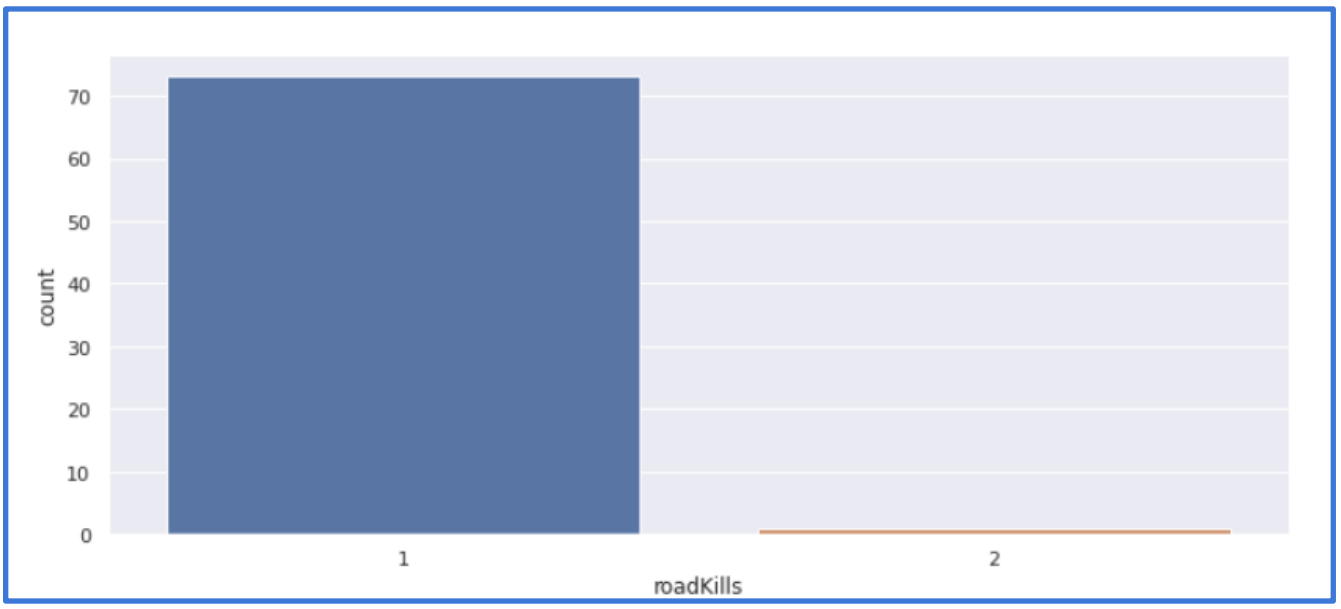
So here we can see the longest kill is zero metre yet there are some non-zero kills which is not possible logically, hence dropping those rows too!

3. **TeamKills and rideDistance:**



In pubg, a player can kill his/her team-mate only if he has a grenade(weapon) or he/she has driven a vehicle over his/her team-mate. But from the above graph there are some players who have killed teamplayer yet they have not acquired any weapon or drove a car/vehicle!

4. **roadKills and rideDistance:**



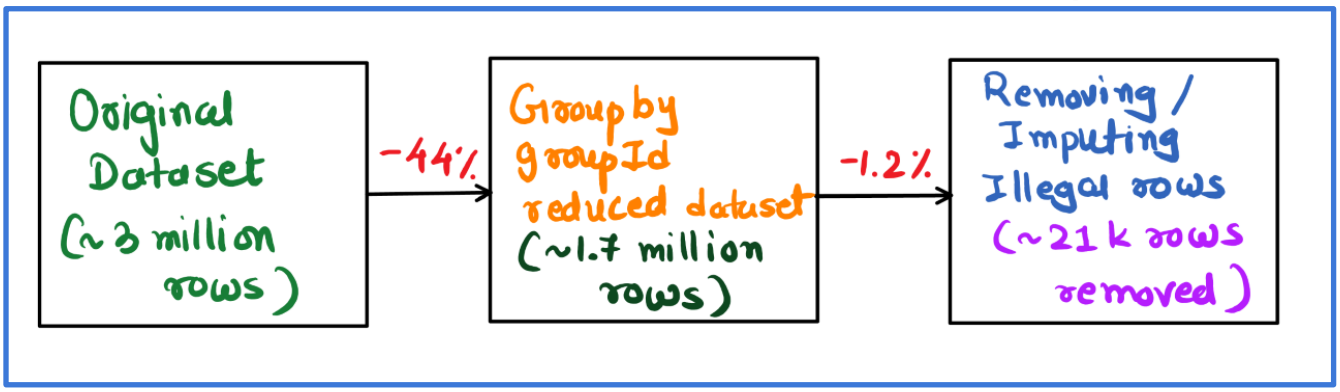
From the above graph, there are some players who have killed enemies while riding a car i.e roadKills, but those players have not rode any vehicles, hence dropping those rows too!

Similarly we have observed some more anomalies stated in the next page.

5. **Players have not walked but have consumed heals and boosts which is not possible hence dropping those rows!**
6. **It's not possible to acquire weapons if a player has not walked a distance.**
7. **If matchType is solo then there cannot be any assists value, because to assist we need teammates which we don't have, as the numbers are somewhat high, so instead of dropping the rows, we imputed that feature with 0.**
8. **A player cannot assist a teammate if the walkDistance is 0.**
9. **A player cannot deal damage if he/she has not walked a single metre.**

Hence after performing the Data Pre-processing we reduced the original dataset's size by a significant amount.

Summary of dataset transition uptill now:



2.3 *Feature Engineering:*

We tried adding new features in the system based on our knowledge of the game, those new features are as follows :

1. **killsPerMeter = kills / walkDistance**
2. **healsPerMeter = heals / walkDistance**
3. **totalHeals = heals + boosts**
4. **totalHealsPerMeter = totalHeals / walkDistance**
5. **totalDistance = walkDistance + rideDistance + swimDistance**
6. **headshotRate = headshotKills / kills**
7. **assistsAndRevives = assists + revives**
8. **itemsAcquired = heals + boosts + weaponsAcquired**
9. **healsOverBoosts = heals / boosts**
10. **walkDistanceOverHeals = walkDistance / heals**
11. **walkDistanceAndHeals = walkDistance * heals**
12. **walkDistanceOverKills = walkDistance / kills**
13. **walkDistanceAndKills = walkDistance * kills**
14. **boostsOverTotalDistance = boosts / totalDistance**
15. **boostsAndTotalDistance = boosts * totalDistance**

After finding the correlation of these features with the target, they had a high correlation indicating these will be good features for learning.

3. Training Process:

3.1 Models Used:

We tried various models to train on the dataset which are the following:

1) Linear Regression:

As it is a simple model, comparisons can be made with respect to this model. Linear Regression is a statistical method to predict the relationship between an independent variable and a dependent variable. This problem dealt with the prediction of a predictor variable. In Linear Regression, the unknown function that maps the dependent variable to the independent variable has its model parameters estimated from the data. After fitting the linear Regression model, if additional data is provided to the model, it predicts the predictor variable automatically.

The model assumes to have a linear relationship in the following way,

$$y_i = \beta_0 1 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathsf{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

This is then solved using an ordinary least square solution wherein the parameters of the model are chosen to minimise the least square values between the predicted and the actual value of the predictor variable which is given as follows:

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} \left| y_i - \sum_{j=1}^{n} X_{ij}\beta_j \right|^2 = \left\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \right\|^2.$$

2) Ridge Regression:

Ridge is an extension of the Ordinary Linear Regression wherein a regularizer term is added. The regularizer term is used to penalise the higher order weights and to increase the sparsity of weights in the model. Regularizer is used in the case of overfitting and the amount of regularisation to be added can be decided. A prior term is added when using Ridge Regression wherein the prior term for Ridge is Gaussian. In the given dataset, chances of overfitting were very low as the number of data points were extremely high compared to the number of features, but we still wanted to see if that MSE value changes after using a regularizer. The MAE values were exactly the same as that of the Ordinary Least Square solution indicating that the use of regularizer is not needed.

3) Random Forest:

Random Forest is one of the main models used for predictive modelling as it uses the ensemble model approach. As it is a non-linear model, I wanted to try this on my dataset and as expected the loss reduced after using Random forest. Random Forest as an ensemble model as multiple decision trees are built during training. During testing, the average of the decisions from multiple trees is taken and assigned to be the final predicted value. Random Forest is a strong learner which combines multiple Decision Trees i.e. weak learners to build the system. Random forest works by randomly sampling multiple subsets from the whole dataset with replacement.

This is called bagging. Due to this, the variance of the final model is reduced in turn leading to a consistent estimator.

4) LightGBM

Light Gradient Boosting Method (LightGBM) is a gradient boosting method that uses a tree- based algorithm. Gradient Boosting is a method where weak learners are added to build a strong learner using gradient based approaches. The specialty of LightGBM is that it is a leaf-based algorithm compared to all other approaches which are level-based. In this method, the tree is grown on leaves and hence as the depth of the tree increases, the complexity of the model increases.

However, for large datasets LightGBM is extremely popular as it runs on high speed with large datasets and also requires lower memory to run. It focuses on decreasing the final accuracy thereby growing the tree on the leaf with maximum delta loss. It also supports GPU learning. For

smaller datasets, it might lead to overfitting but as the dataset I have used is very large, it works the best. However, as a lot of parameters are present, hyperparameter tuning is a bit cumbersome.

5) XGBoost

XGBoost is the abbreviation for eXtreme Boosting. This also uses the Gradient Boosting Decision Tree algorithm. Gradient Boosting is an approach where new models are added to the existing models to decrease the loss and the combined result from all these models is used as the final prediction. It uses the gradient descent algorithm to minimize the loss when adding new models. The execution time of the XGboost model is extremely small and it also uses the leaf-based tree growing. XGBoost is a very popular model used in Kaggle competitions due to it's ability to handle large datasets.

3.2 Metric Used:

As we had multiple models, to identify the best model's performance, we used Mean Squared Error (MSE) metric.

Mean Squared Error is the measure of the square of the difference between actual value and the predicted value, average over all the datapoints.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

3.3 Parameter Tuning:

- Random Forest Parameter Tuning :

	Sr.no	max_depth	max_features	min_samples_split	n_estimators	n_jobs	oob_score	warm_start	criterion	mse
0	1	17	None	1300	60	-1	True	True	NaN	0.00824000
1	2	21	None	1500	65	-1	True	True	squared_error	0.00728000
2	3	23	None	700	100	-1	True	True	squared_error	0.00666860
3	4	21	None	300	75	-1	True	True	squared_error	0.00615927
4	5	23	None	300	45	-1	True	True	squared_error	0.00615918
5	6	23	None	500	45	-1	True	True	squared_error	0.00643476
6	7	24	None	200	75	-1	True	True	squared_error	0.00598467
7	8	25	None	90	110	-1	True	True	squared_error	0.00573305
8	9	17	None	50	75	-1	True	True	squared_error	0.00571524
9	10	21	None	50	85	-1	True	True	squared_error	0.00565401
10	11	19	None	40	70	-1	True	True	squared_error	0.00565962
11	12	25	None	40	75	-1	True	True	squared_error	0.00563619
12	13	39	None	45	95	-1	True	True	squared_error	0.00563110
13	14	29	None	30	80	-1	True	True	squared_error	0.00561994
14	15	45	None	20	95	-1	True	True	squared_error	0.00561125
15	16	40	None	15	65	-1	True	True	squared_error	0.00562889
16	17	35	None	20	95	-1	True	True	squared_error	0.00554200

- XGBoost Hyper-parameter tuning:

	Sr.no	n_estimators	max_depth	eta	subsample	colsample_bytree	use_rmm	gamma	tree_method	updater	predictor	max_leaves	reg_alpha	colsample_bylevel	num_parallel_tree	mse
0	1	60	17	0.10000000	0.80000000	0.80000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00539123
1	2	70	17	0.20000000	0.90000000	0.90000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00554078
2	3	60	21	0.30000000	0.70000000	0.90000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00657381
3	4	60	15	0.01000000	0.70000000	0.70000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.03217359
4	5	60	19	0.15000000	0.85000000	0.85000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00561539
5	6	75	17	0.10000000	0.80000000	0.80000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00536817
6	7	80	18	0.09900000	0.80000000	0.80000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00542747
7	8	80	18	0.10000000	0.80000000	0.80000000	True	3.00000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.00579892
8	9	80	18	0.10000000	0.80000000	0.80000000	NaN	2.00000000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00584288
9	10	80	18	0.10000000	0.80000000	0.80000000	NaN	5.00000000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00630422
10	11	70	15	0.09900000	0.90000000	0.90000000	NaN	0.10000000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00521426
11	12	70	17	0.09900000	1.00000000	1.00000000	NaN	0.10000000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00524112
12	13	70	15	0.09900000	0.90000000	0.90000000	NaN	0.09900000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00521241
13	14	70	15	0.09900000	0.90000000	0.90000000	NaN	0.00100000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00523921
14	15	70	17	0.09500000	0.90000000	0.90000000	NaN	0.09900000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00521779
15	16	70	15	0.09900000	0.90000000	0.90000000	NaN	0.09900000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00520739
16	17	70	14	0.10000000	0.80000000	0.80000000	NaN	0.09900000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00519724
17	18	70	15	0.10000000	0.80000000	0.80000000	NaN	0.05000000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00517688
18	19	70	15	0.10000000	0.80000000	0.80000000	NaN	0.05000000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00518126
19	20	100	14	0.09700000	0.80000000	0.80000000	NaN	0.06500000	gpu_hist	grow_gpu_hist	gpu_predictor	NaN	NaN	NaN	NaN	0.00511575
20	21	130	15	0.12000000	0.80000000	0.80000000	NaN	0.03000000	gpu_hist	NaN	NaN	NaN	NaN	NaN	NaN	0.00508642
21	22	130	15	0.12000000	0.80000000	0.80000000	NaN	0.03000000	gpu_hist	NaN	NaN	NaN	NaN	NaN	NaN	0.00507646
22	23	125	15	0.10800000	0.80000000	0.80000000	NaN	0.02950000	gpu_hist	NaN	NaN	1,250.00000000	0.09900000	0.80000000	10.00000000	0.00499091
23	24	125	15	0.10800000	0.80000000	0.80000000	NaN	0.02950000	gpu_hist	NaN	NaN	1,250.00000000	0.09900000	0.80000000	15.00000000	0.00498714
24	25	125	15	0.11300000	0.80000000	0.80000000	NaN	0.02950000	gpu_hist	NaN	NaN	1,250.00000000	0.09950000	0.80000000	20.00000000	0.00497300

- LightGBM Hyper-parameter tuning:

	colsample_bytree	learning_rate	max_depth	min_split_gain	n_estimators	num_leaves	reg_alpha	reg_lambda	subsample	subsample_for_bin	n_jobs	max_bin	num_iterations	min_data_in_bin	mse
0	0.82000000	0.09000000	23	0.00011100	210	2100	0.09500000	0.00100000	0.82000000	45000	-1	700	600	15	0.00495200
1	0.82000000	0.05000000	25	0.00011100	230	2100	0.09500000	0.00100000	0.82000000	45000	-1	700	600	15	0.00488300
2	0.82000000	0.05000000	27	0.00011100	230	2100	0.09500000	0.00100000	0.82000000	45000	-1	700	750	15	0.00488000
3	0.82000000	0.05000000	27	0.00011100	230	2100	0.09500000	0.00100000	0.82000000	45000	-1	700	1200	15	0.00487600
4	0.80000000	0.02500000	29	0.00015000	250	2200	0.50000000	0.00100000	0.80000000	45000	-1	710	2100	12	0.00485100
5	0.82000000	0.04000000	27	0.00011100	250	2100	0.09500000	0.00100000	0.82000000	45000	-1	700	2000	15	0.00485000
6	0.82000000	0.03000000	27	0.00011100	250	2000	0.09500000	0.00100000	0.82000000	35000	-1	600	2100	15	0.00484100
7	0.80000000	0.03000000	28	0.00015000	250	2200	0.12000000	0.01000000	0.80000000	43000	-1	715	2100	15	0.00483300
8	0.80000000	0.03000000	29	0.00015000	250	2200	0.12000000	0.00500000	0.80000000	43000	-1	715	2100	12	0.00483100
9	0.82000000	0.03000000	29	0.00011100	250	2200	0.09500000	0.00100000	0.82000000	45000	-1	700	2100	10	0.00482900
10	0.82000000	0.03000000	29	0.00011100	250	2200	0.09500000	0.00100000	0.82000000	45000	-1	700	2100	20	0.00482800
11	0.80000000	0.03000000	28	0.00015000	250	2200	0.12000000	0.00500000	0.80000000	43000	-1	715	2100	15	0.00482600
12	0.82000000	0.03000000	29	0.00011100	250	2200	0.09500000	0.00100000	0.82000000	45000	-1	700	2100	12	0.00482500
13	0.82000000	0.02000000	29	0.00011100	350	2200	0.09500000	0.00100000	0.82000000	45000	-1	700	3500	20	0.00482400
14	0.82000000	0.02000000	29	0.00011100	350	2200	0.09500000	0.00100000	0.82000000	45000	-1	700	2700	20	0.00482300
15	0.82000000	0.02000000	29	0.00011100	350	2200	0.09500000	0.00100000	0.82000000	44000	-1	700	2800	20	0.00482100
16	0.82000000	0.02000000	29	NaN	350	2200	0.09500000	0.00100000	0.82000000	45000	-1	700	4000	20	0.00480800

3.4 Best Parameters:

Models	Parameters	MSE
Linear Regression	n_jobs=-1	0.012892
Ridge Regression	alpha=10, max_iter=1000, solver='svd'	0.012892
Random Forest	max_depth=35, max_features=None, min_samples_split=20,n_estimators=95, n_jobs=-1,oob_score=True, warm_start=True,criterion="squared_error"	0.005542
XGBoost	gamma=0.0295,n_estimators=125, max_depth=15, eta=0.113, subsample=0.8, colsample_bytree=0.8, tree_method='gpu_hist',max_leaves = 1250,reg_alpha =0.0995,colsample_bylevel = 0.8,num_parallel_tree =20	0.004973
LightGBM	colsample_bytree=0.8, learning_rate=0.03, max_depth=30, min_split_gain=0.00015, n_estimators=250, num_leaves=2200,reg_alpha=0.1, reg_lambda=0.001, subsample=0.8, subsample_for_bin=45000, n_jobs =-1, max_bin =700, num_iterations=5200, min_data_in_bin = 12	0.004829

4. Conclusion:

In this project, a variety of machine learning algorithms and models were experimented. As we have mentioned earlier, we found that the algorithm which works best for this dataset is where grouping of data points is done, and feature dimensions is increased by adding more features from this grouping and also some manual features. Also, LightGBM being fast and efficient for large datasets works the best.