Project:
Santander Customer Transaction Prediction

Background -

At Santander , mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals. Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

Problem Statement -

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

1.Introduction

The given dataset contains the bank's customer's financial transactions and other service utilization details. The given dataset does not contain any explicit information about the customers, instead a code identification number and numerical values. The given dataset is a binary classification problem where we are required to predict the outcome of the target variable. The dataset has train and test csv files where the train file has target variable and the outcome form the train results need to be tested on test file.

2. Exploratory Data Analysis

2.1 Pre Processing

We begin by exploring the data, cleaning the data as well as visualizing the data through graphs and plots, which is often called as **Exploratory Data Analysis (EDA).**

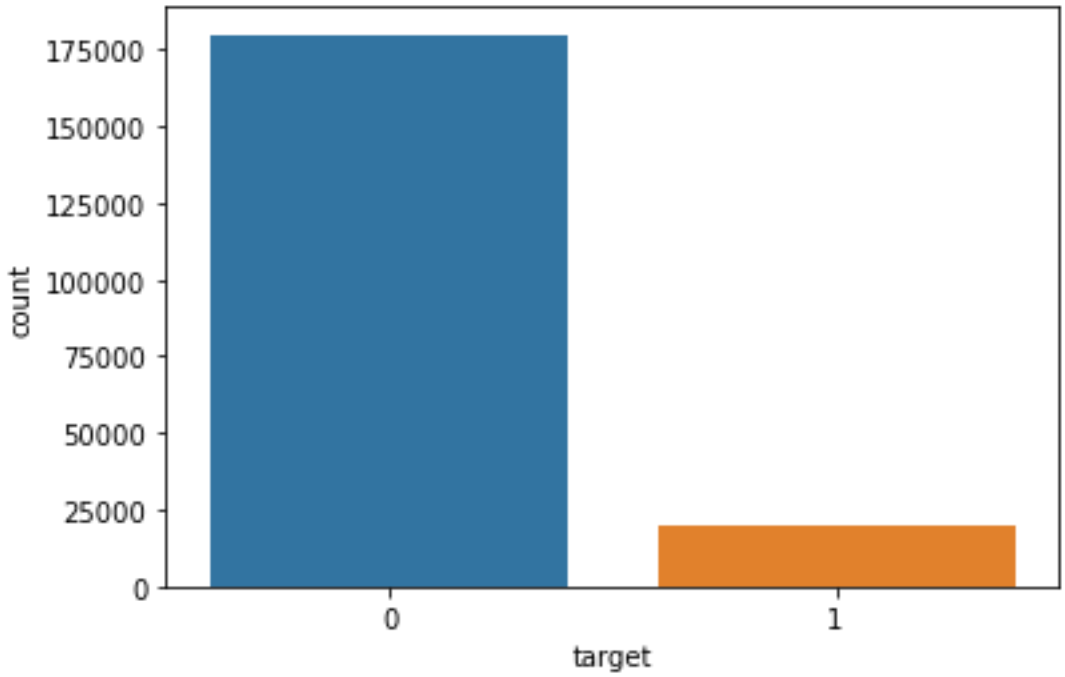
In this process we try to extract all essential details about the given information and analyse it for further processing of the data.

We first check the shape of the dataset .the dataset contains 200000 rows and 202 columns in the train dataset and 200000 rows and 201 columns in test dataset.

We then check if there are any null values or also known as missing values. The given train and test dataset has no missing values.

The describe command provides us insights into important statistical detains regarding the dataset like the count, mean, median, mode etc.

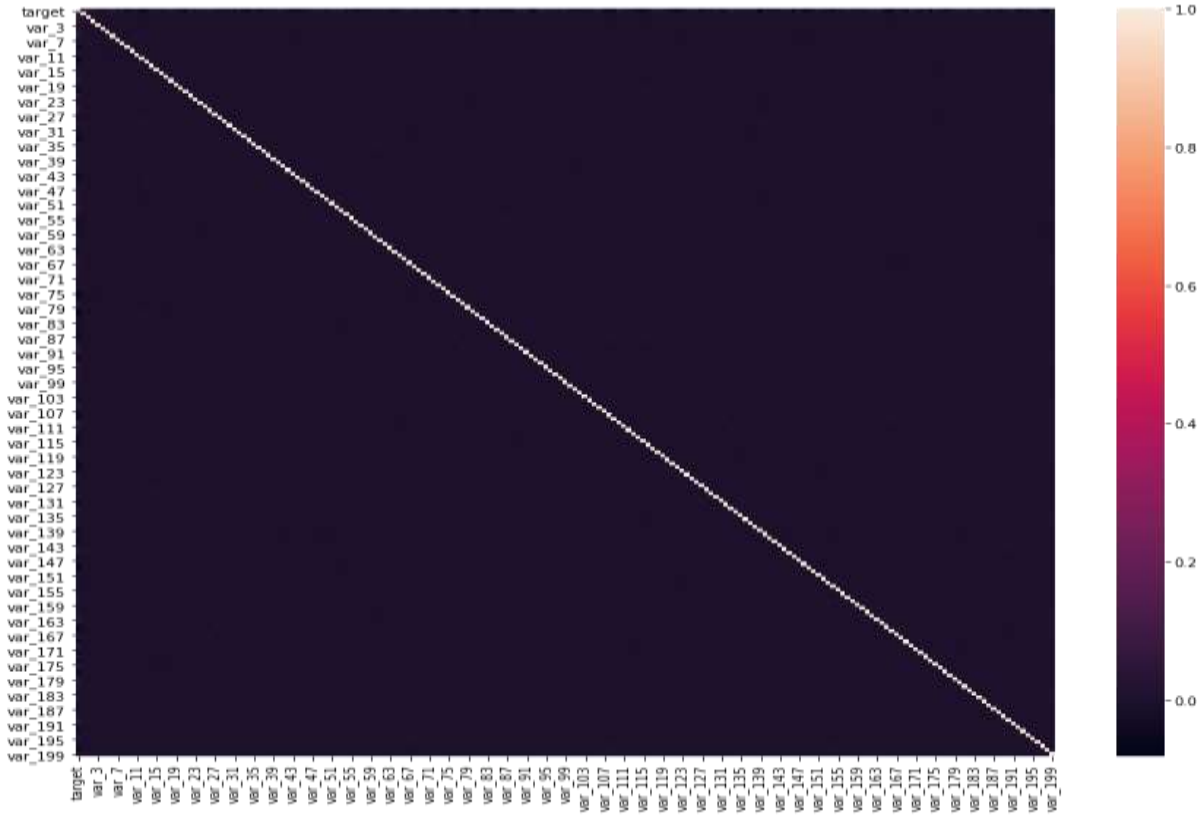
The train dataset has an attribute target which contains 0's and 1's. The value_counts() command provides the count of the values. Form the value counts it is evident that the dataset is highly imbalanced.



The above fig represents the value count of the target variable.

Correlation Analysis

It provides information regarding correlation among variables in the dataset.

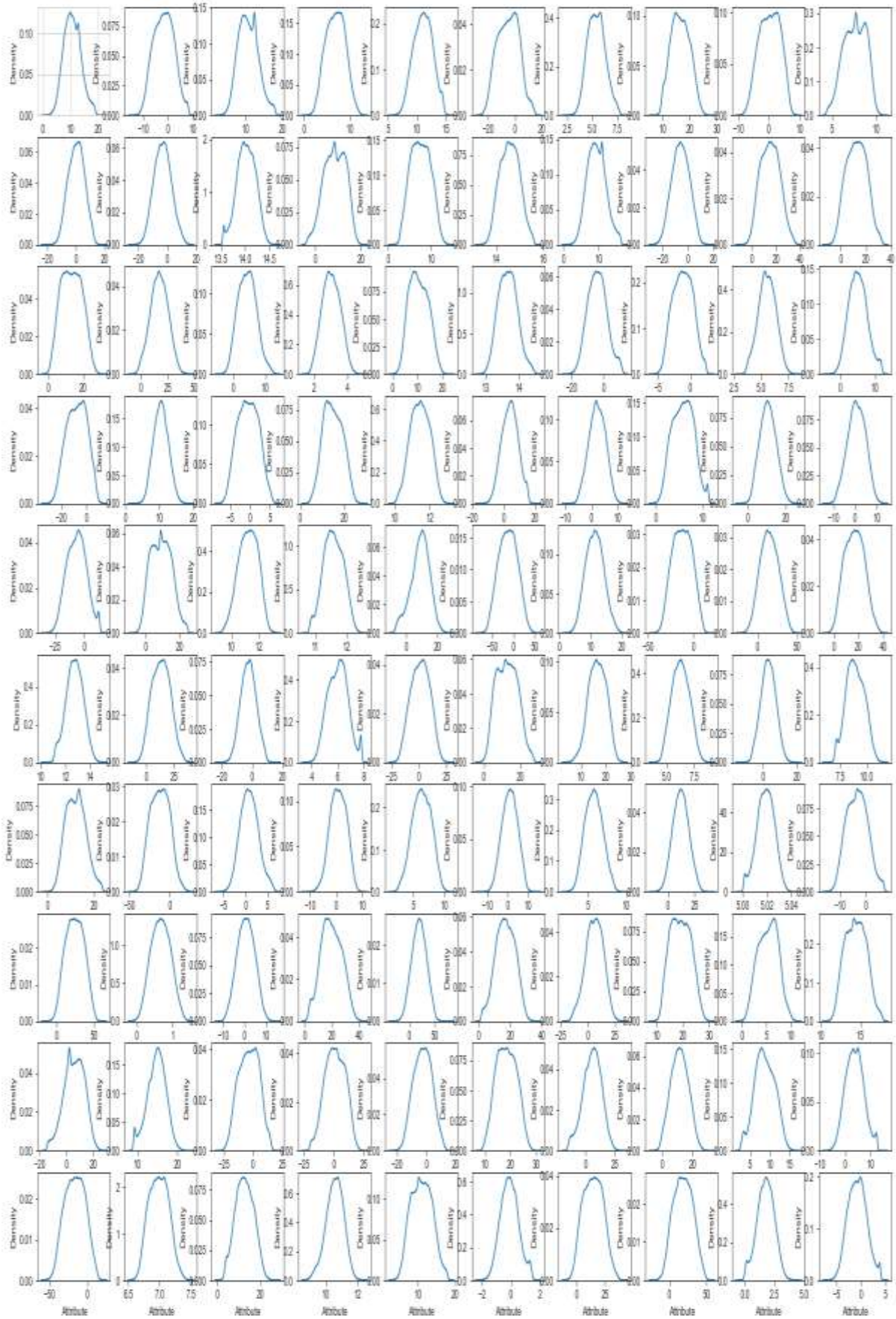


Form the heatmap it is evident that the features are independent and are not correlated.

Distribution of train attributes

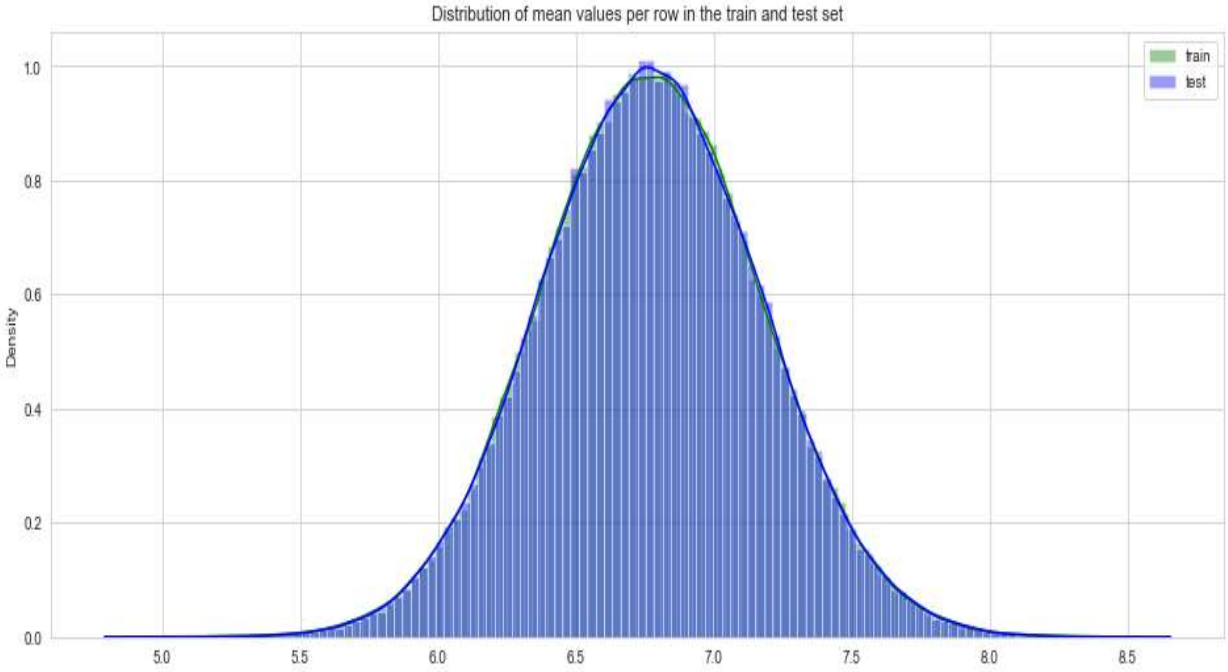
We can observed that their is a considerable number of features which are significantly have different distributions for two target variables. For example like var_0,var_1,var_9,var_198 var_180 etc. We can observed that their is a considerable number of features which are significantly have same distributions for two target variables. For example like var_3,var_7,var_10,var_171,var_185 etc.

Distribution of test attributes

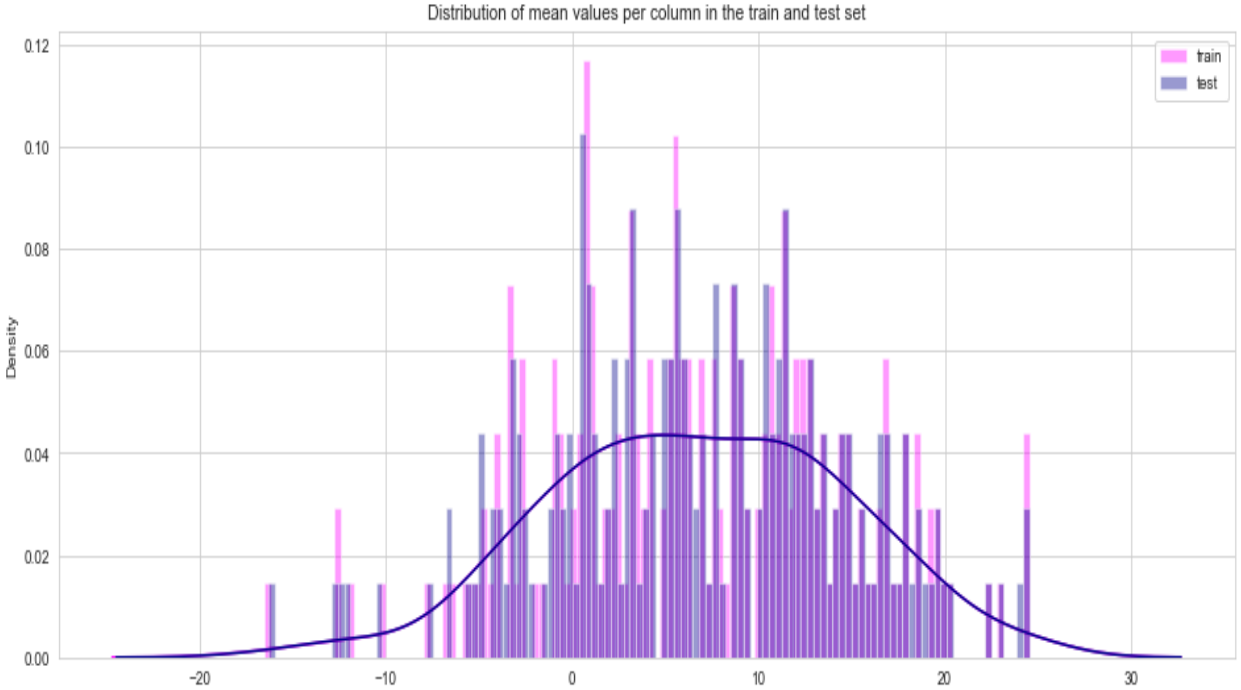


We can observed that their is a considerable number of features which are significantly have different distributions. For example like var_0,var_1,var_9,var_180 var_198 etc. We can observed that their is a considerable number of features which are significantly have same distributions. For example like var_3,var_7,var_10,var_171,var_185,var_192 etc

Distribution of mean values per row in the train and test set

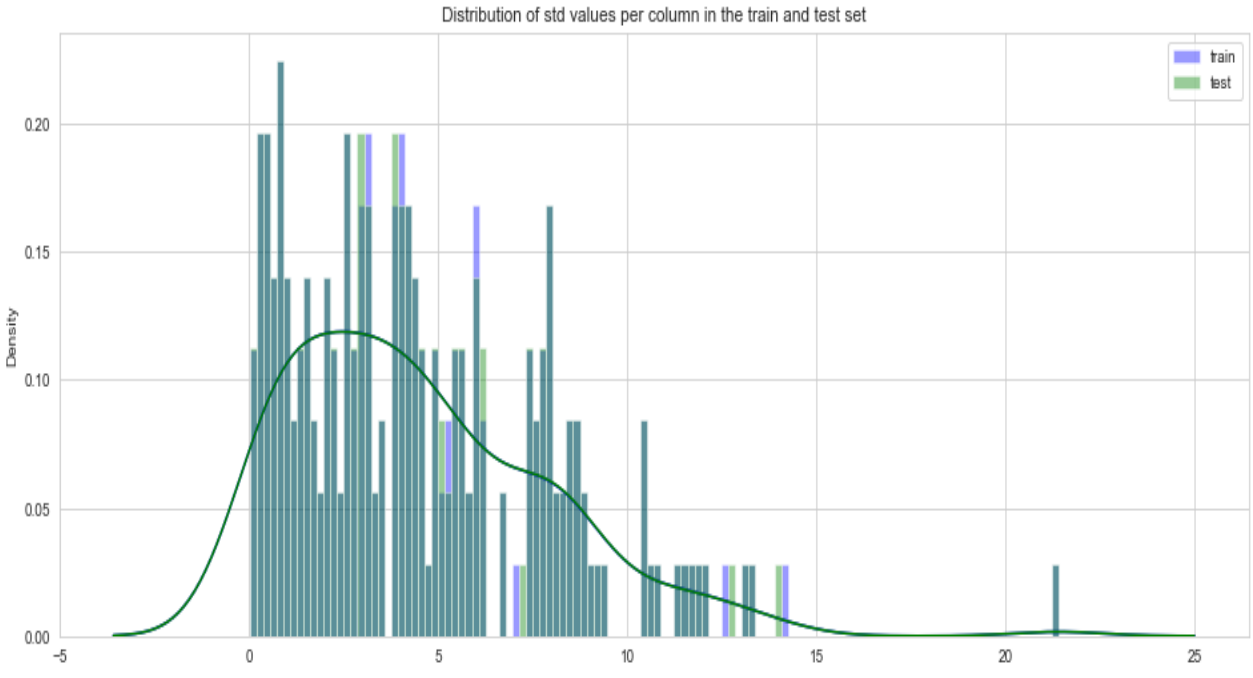


Distribution of mean values per column in the train and test set

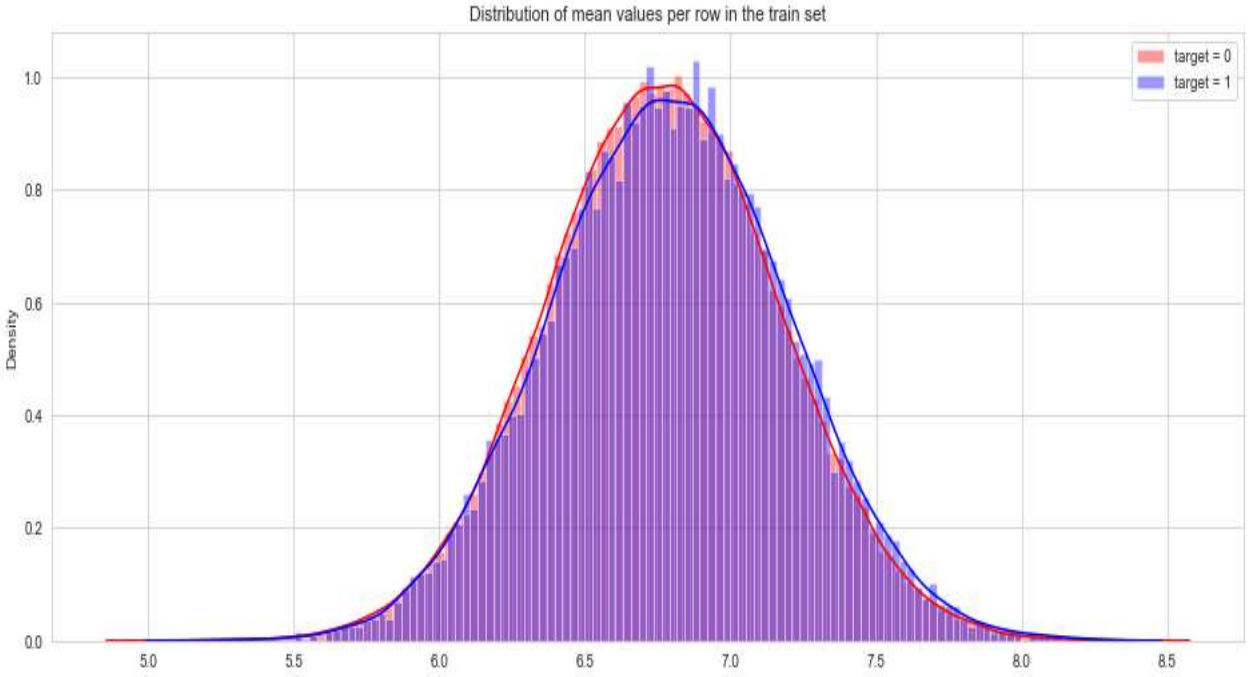


Distribution of std values per row in the train and test set

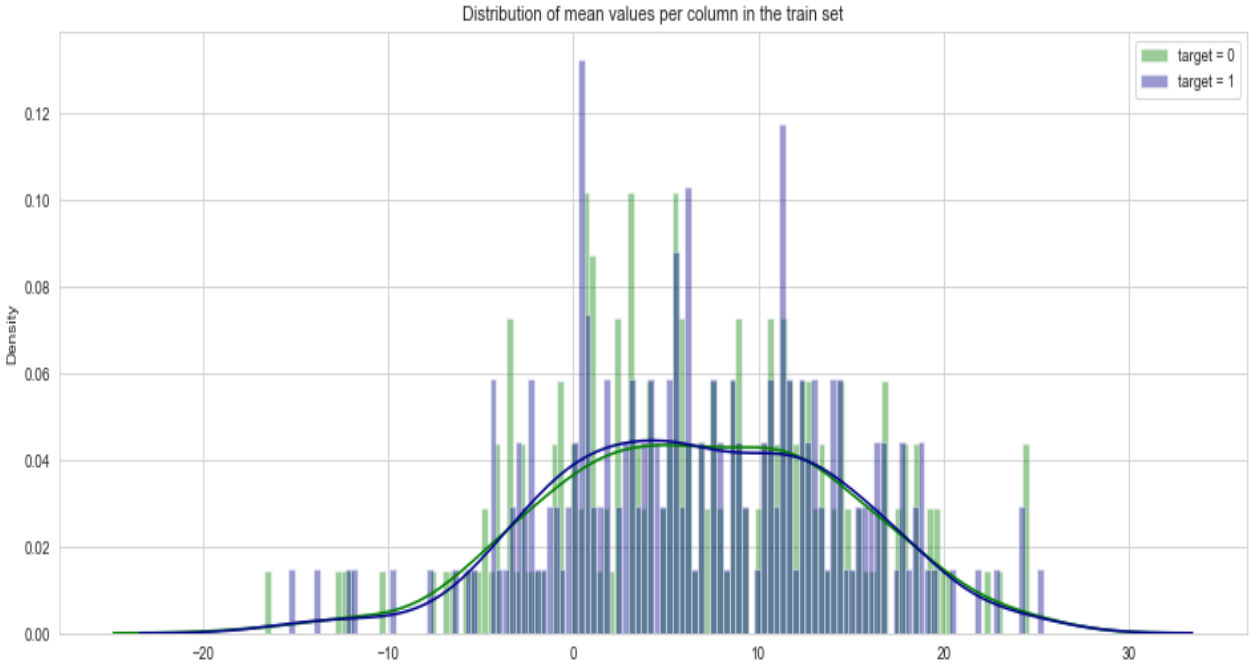
Distribution of std values per column in the train and test set



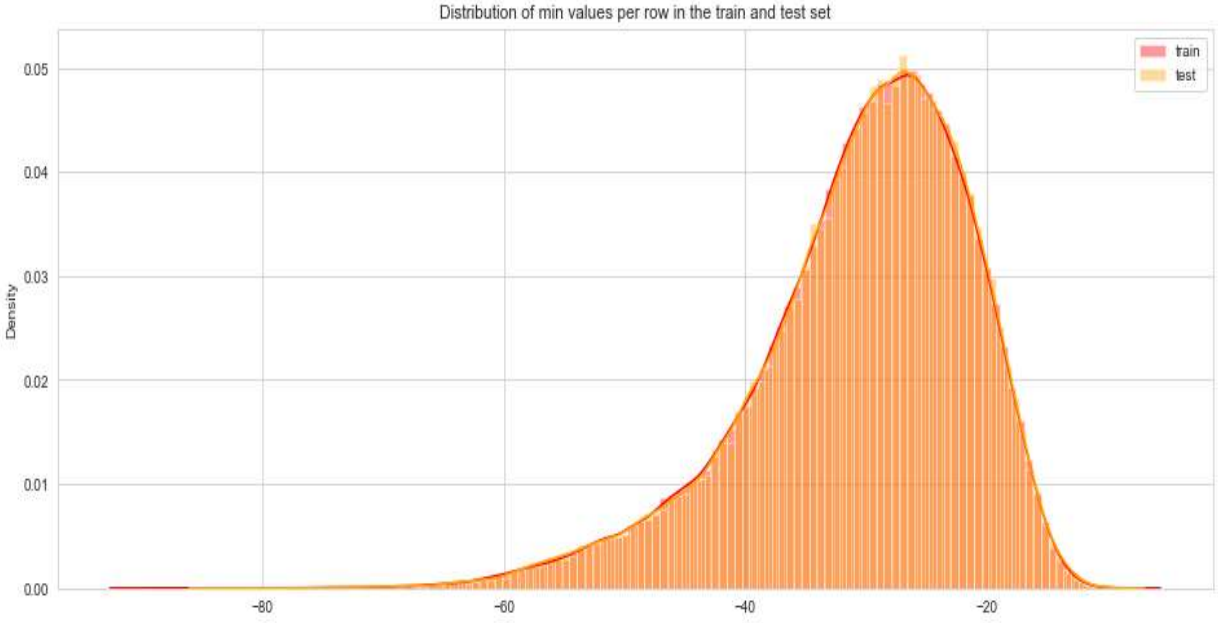
Distribution of mean values per row in the train set



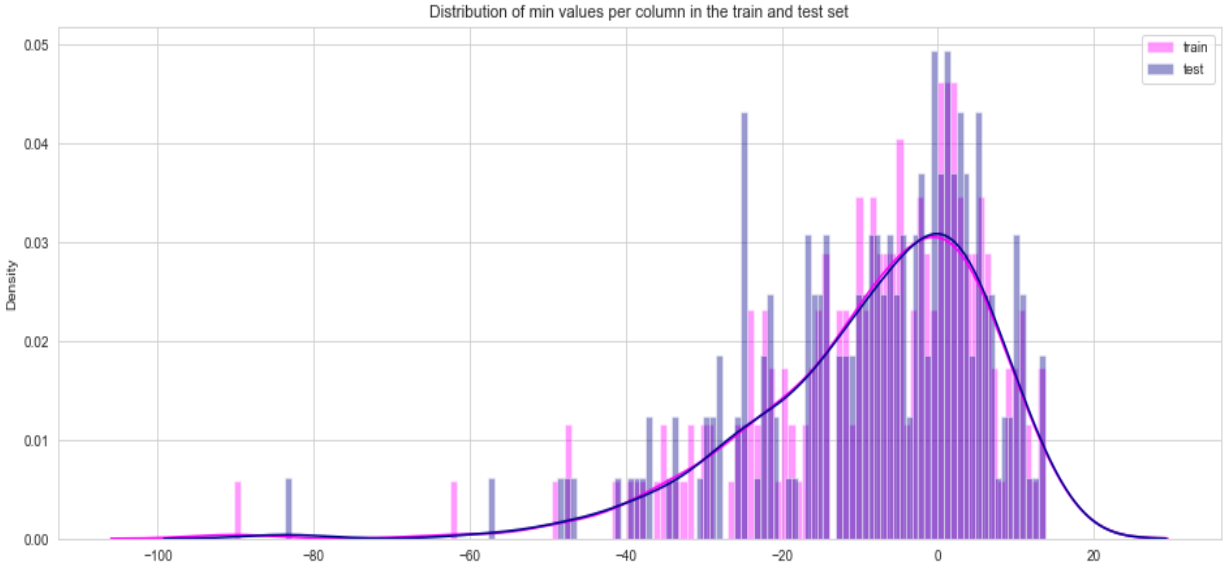
Distribution of mean values per column in the train set



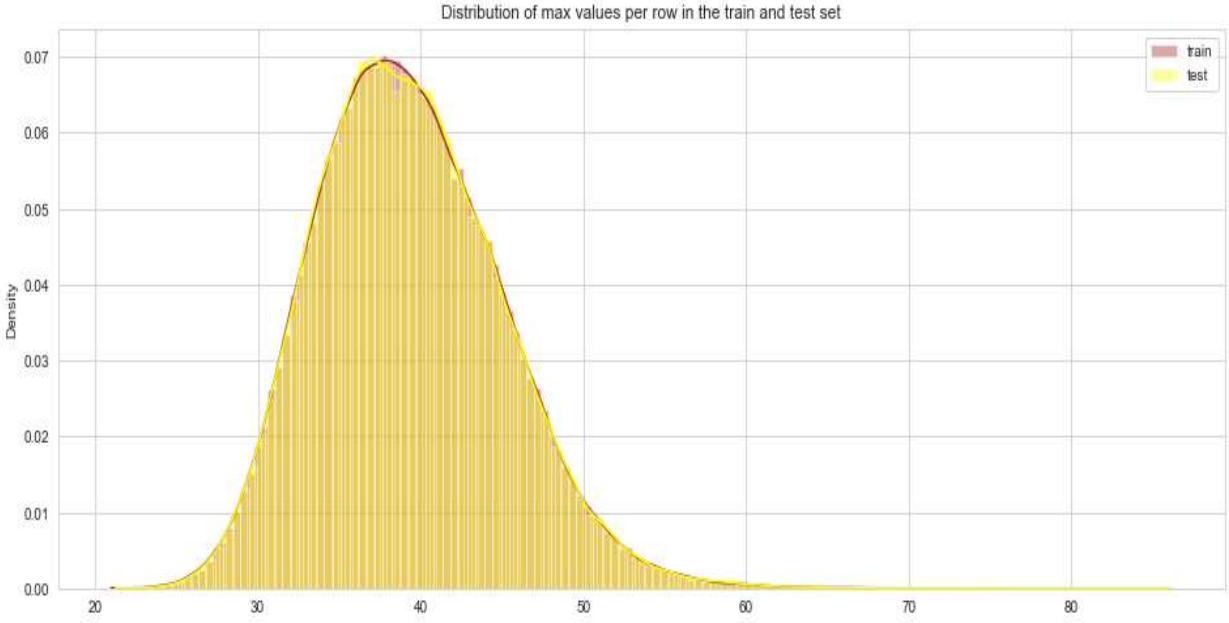
Distribution of min values per row in the train and test set



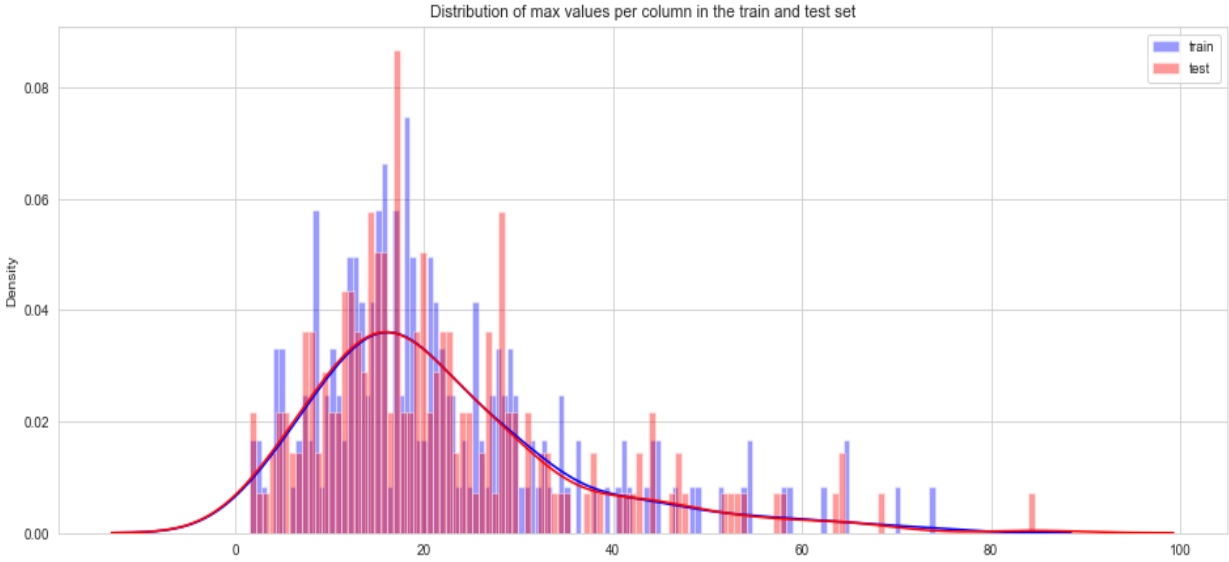
Distribution of min values per column in the train and test set



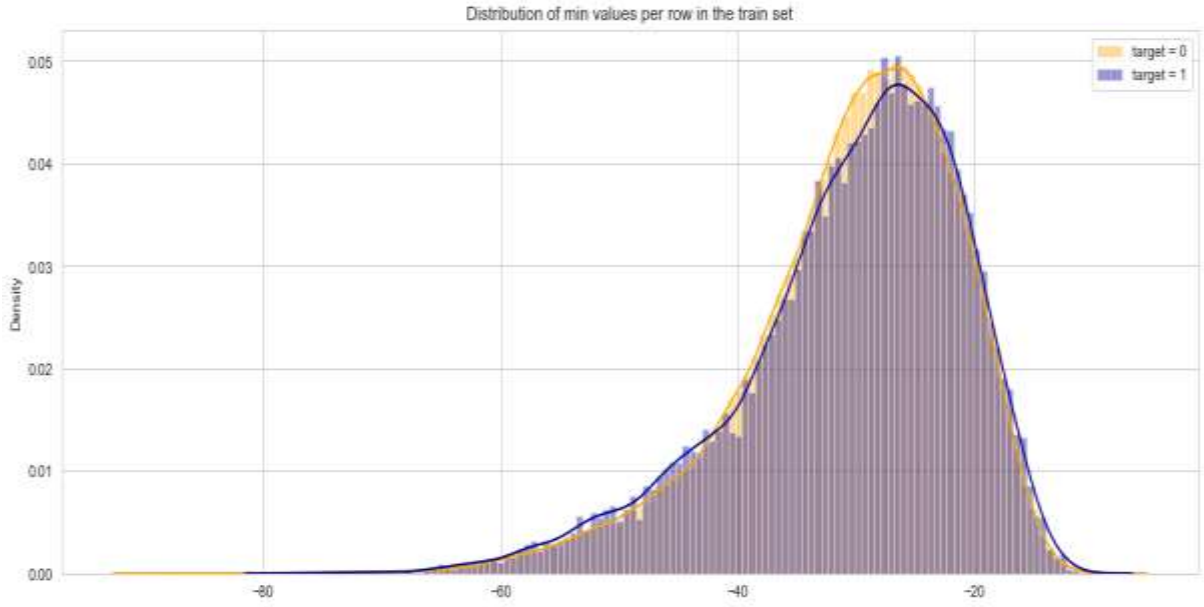
Distribution of max values per row in the train and test set



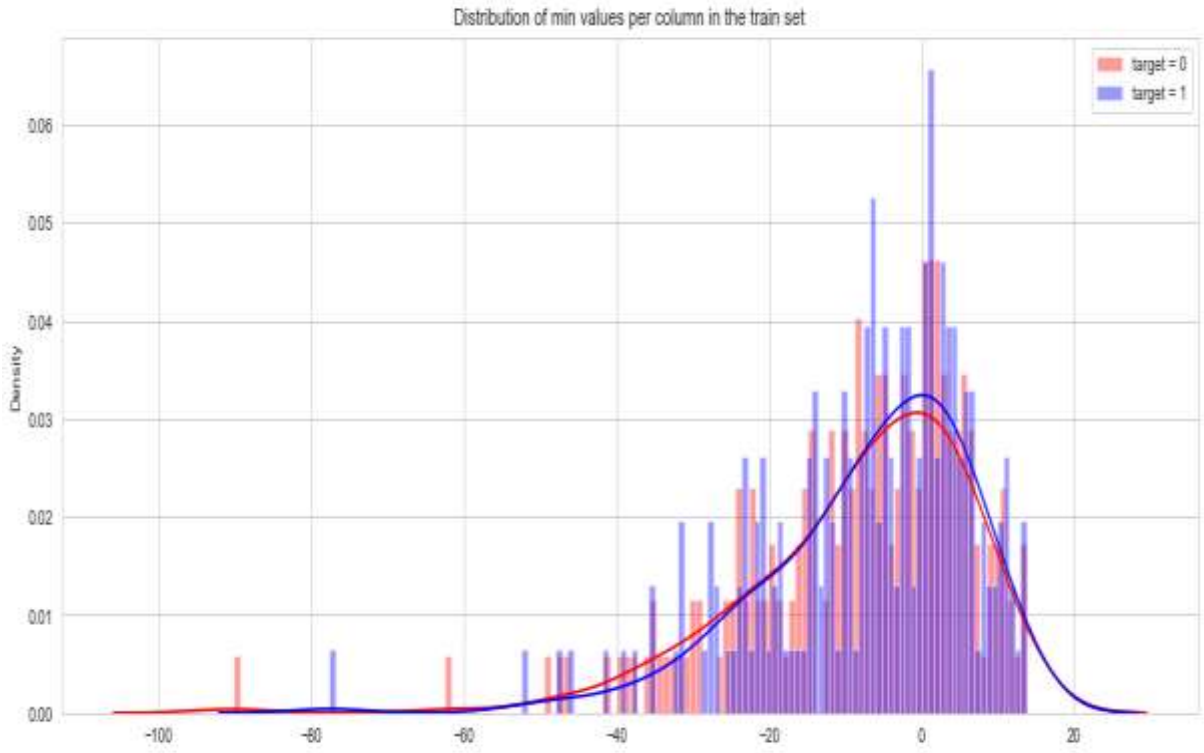
Distribution of max values per column in the train and test set



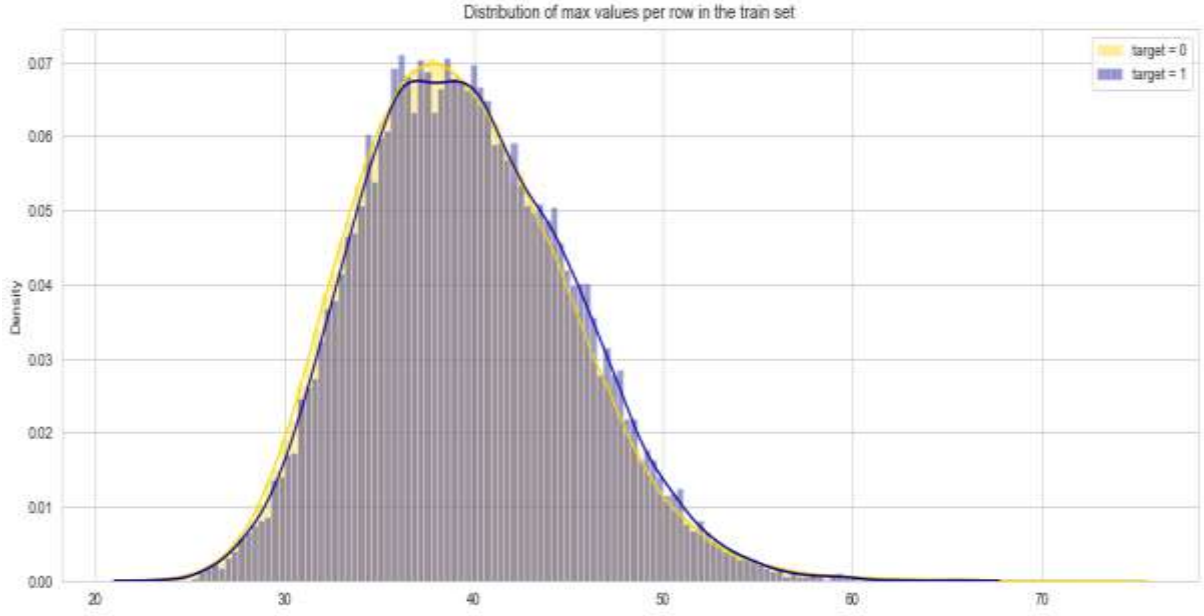
Distribution of min values per row in the train set



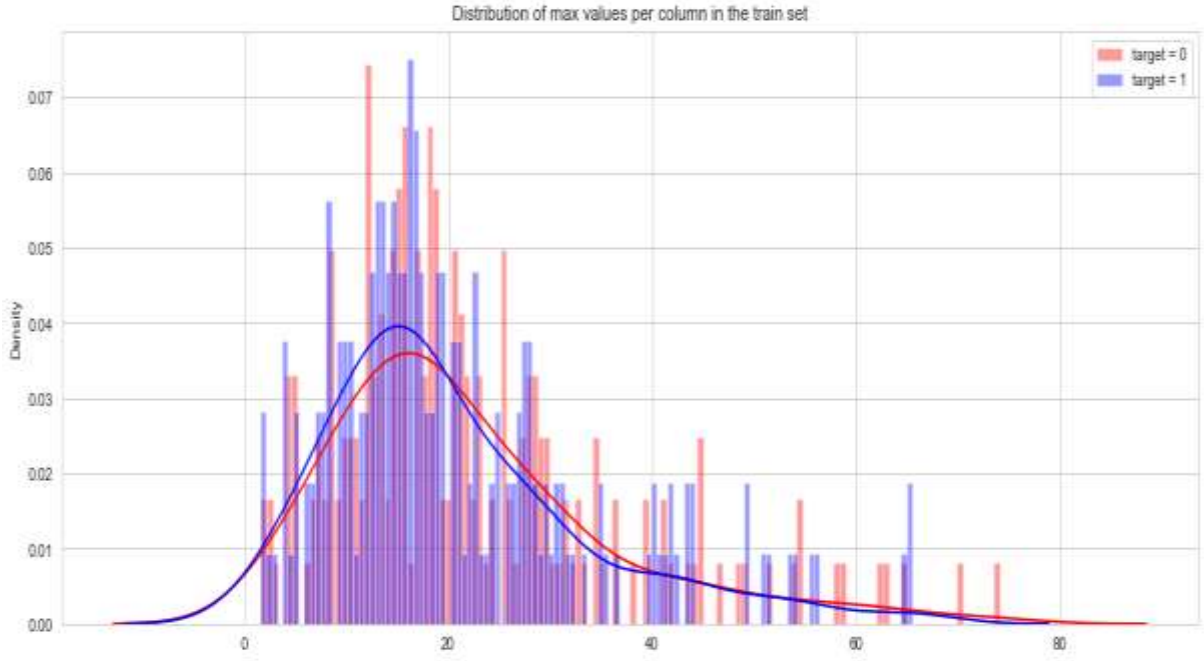
Distribution of min values per column in the train set



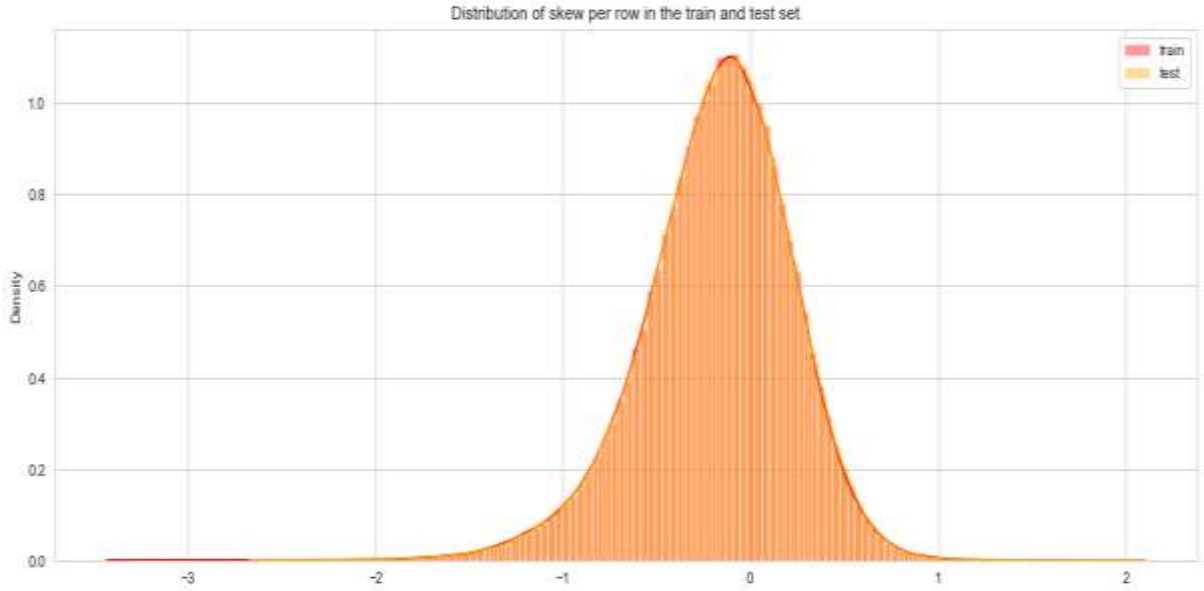
Distribution of max values per row in the train set



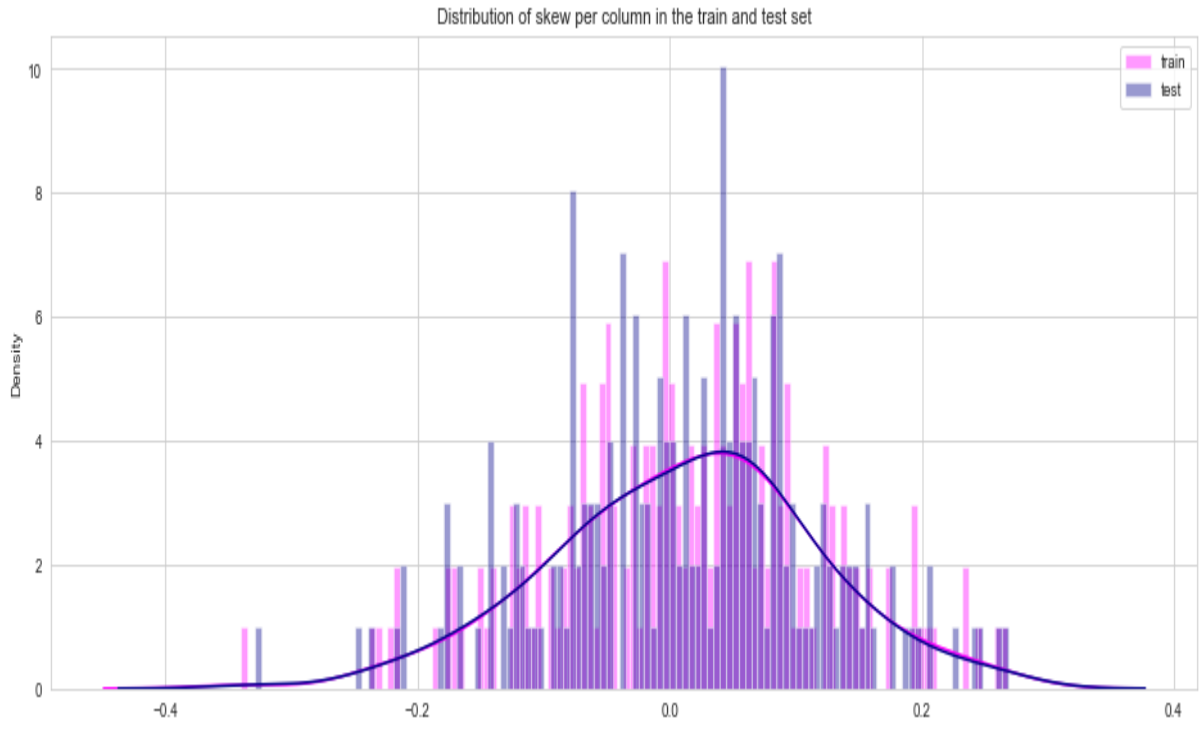
Distribution of max values per column in the train set



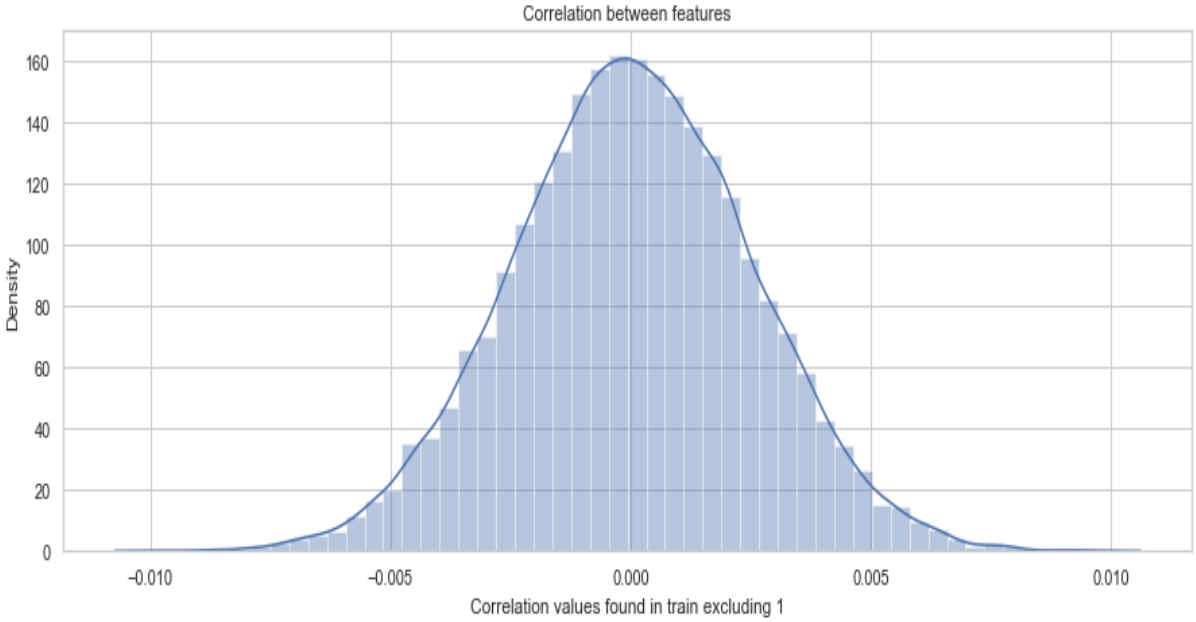
Distribution of skew per row in the train and test set



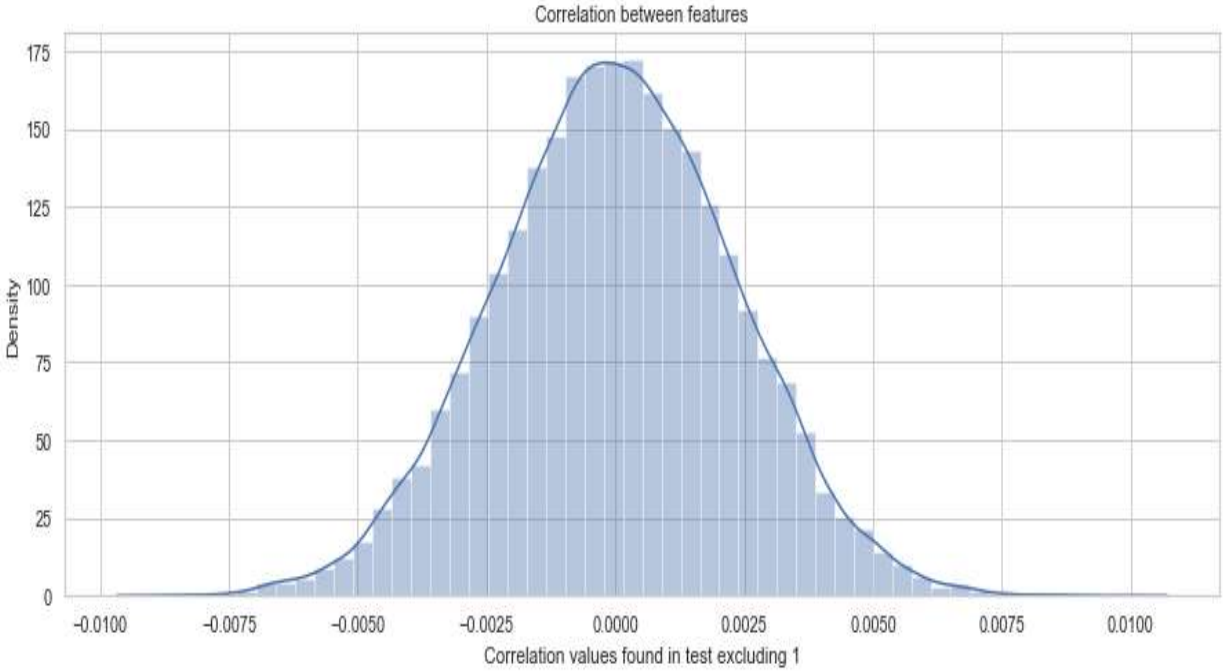
Distribution of skew per column in the train and test set



Correlation among test and train dataset



Thus, we can conclude that there is no correlation in train dataset



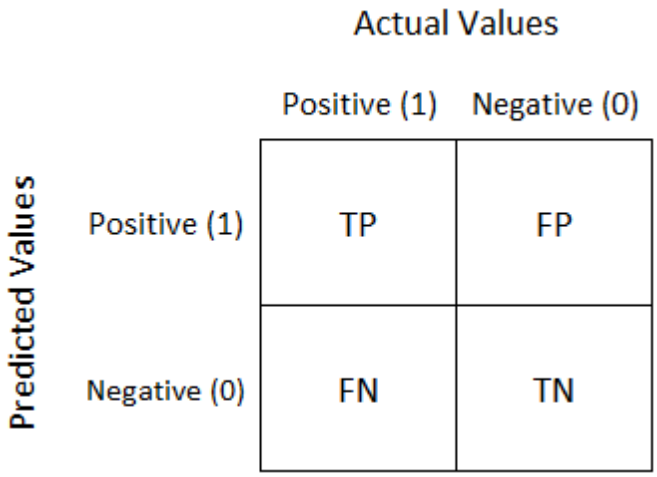
Thus, we can conclude that there is no correlation in test dataset

4. Classification Model Testing and Training

In this process we first predict the accuracy and then construct the confusion matrix and classification report. A **confusion matrix** is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

There are four ways to check if the predictions are right or wrong:

1. **TN / True Negative**: the case was negative and predicted negative
2. **TP / True Positive**: the case was positive and predicted positive
3. **FN / False Negative**: the case was positive but predicted negative
4. **FP / False Positive**: the case was negative but predicted positive



The **classification report** visualizer displays the precision, recall, F1, and support
scores for the model.

Precision —Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class, it is defined as the ratio of true positives to the sum of a true positive and false positive.

Precision: - Accuracy of positive predictions.

Precision = TP/ (TP + FP)

Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

Recall: - Fraction of positives that were correctly identified.
Recall = TP/ (TP+FN)

The **F1 score** is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support doesn't change between models but instead diagnoses the evaluation process.

AUC ROC

The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR** against **FPR** at various threshold values and essentially **separates the 'signal' from the 'noise'**. The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

- When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If, however, the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.
- When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.
- When AUC=0.5, then the classifier is not able to distinguish between Positive and Negative class points. Meaning either the classifier is predicting random class or constant class for all the data points.

In a ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance between False positives and False negatives.

5. Different Machine Learning Algorithms Implementation

LogisticRegression

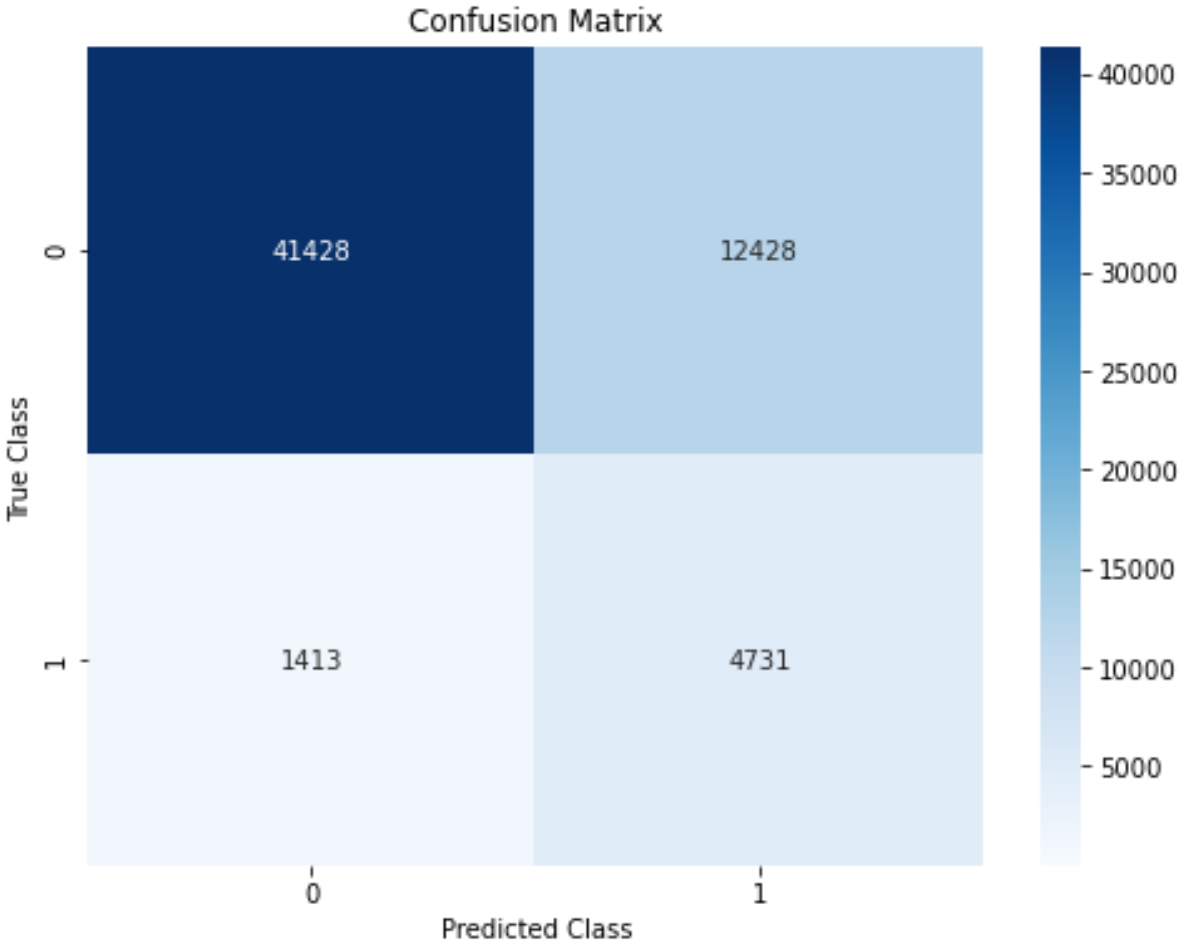
Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated, and it's convenient to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multiclass problems. Since this is an unbalanced dataset, we need to define another paramenter 'class_weight = balanced' which will give equal weights to both the targets irrespective of their reperesentation in the training dataset. We can even define classwise weights using this parameter, if needed.

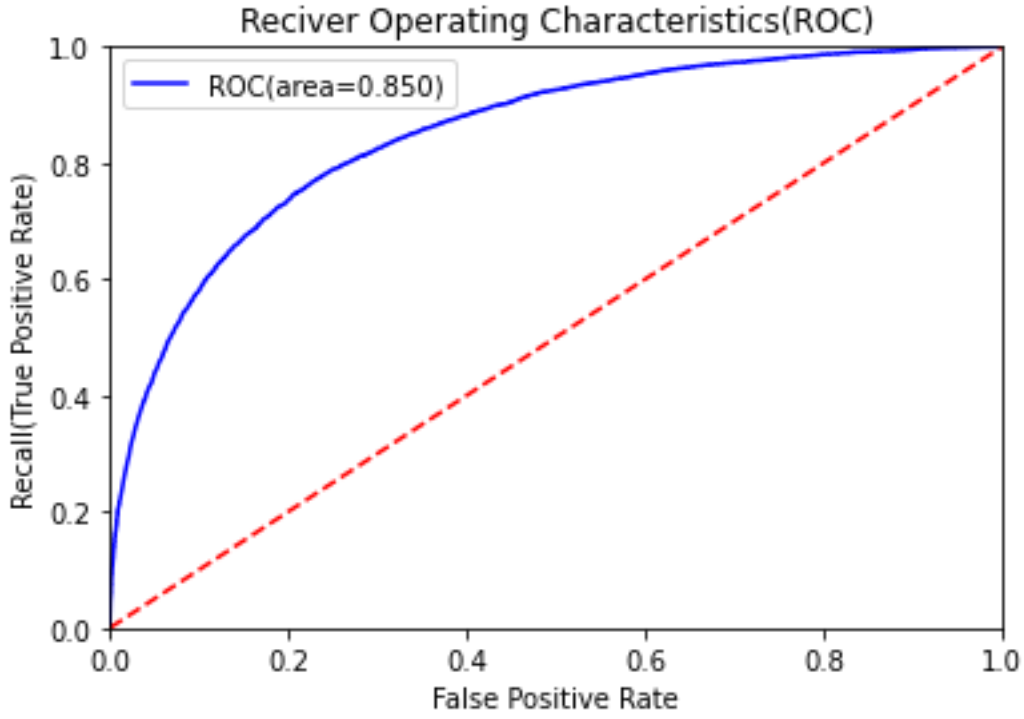
```
Test accuracy is 0.769

Classification report:
              precision    recall  f1-score   support

           0       0.97      0.77      0.86     53856
           1       0.28      0.77      0.41      6144

    accuracy                           0.77     60000
   macro avg       0.62      0.77      0.63     60000
weighted avg       0.90      0.77      0.81     60000
```


Confusion Matrix
41428
12428
1413
4731
True Class
Predicted Class
0
1
40000
35000
30000
25000
20000
15000
10000
5000


Reciver Operating Characteristics(ROC)
ROC(area=0.850)
Recall(True Positive Rate)
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Decision Tree :

Moving on to a slightly advanced algorithm, decision trees. Again, the parameters here are class_weight to deal with unbalanced target variable, random_state for reproducability of same trees. The feature max_features and min_sample_leaf are used to prune the tree and avoid overfitting to the training data.

Max_features defines what proportion of available input features will be used to create tree.

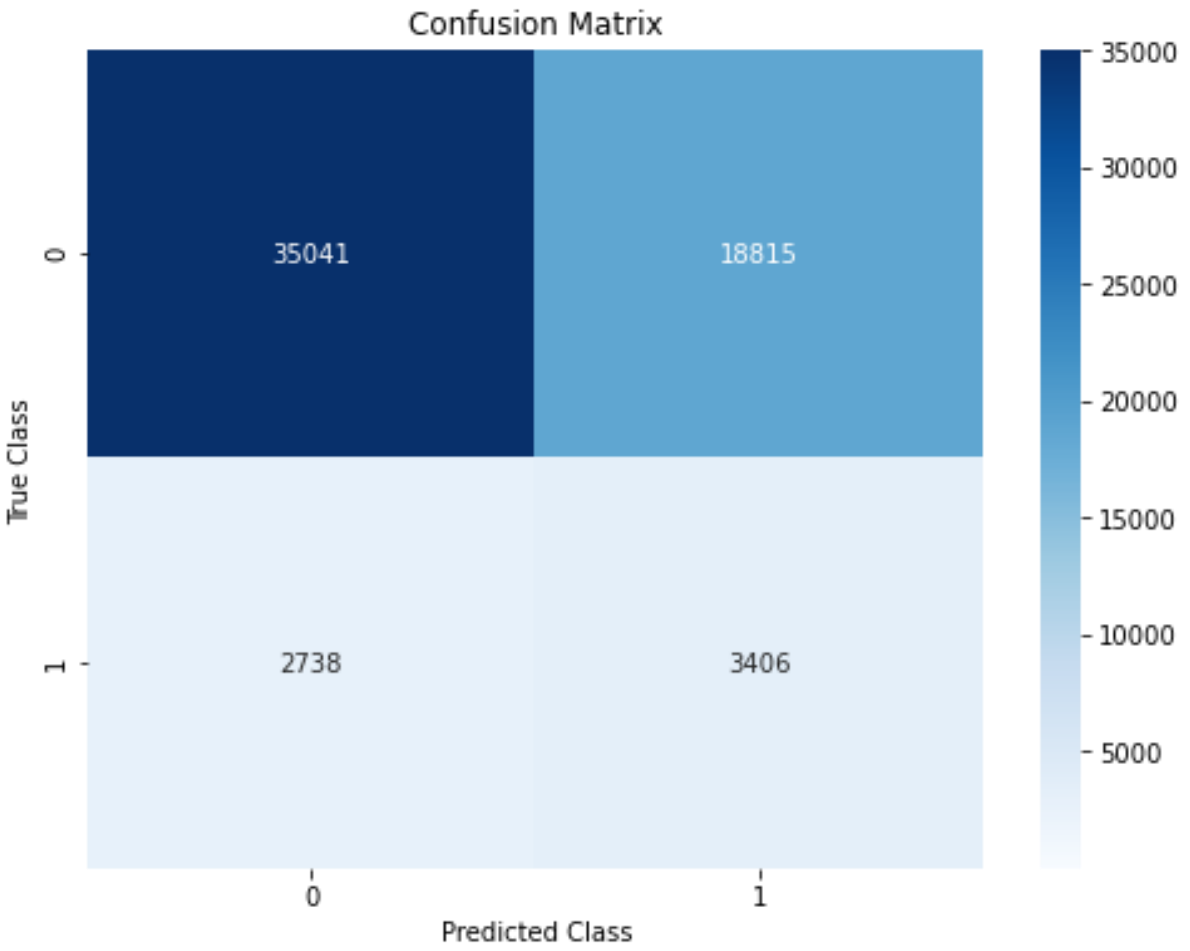
Min_sample_leaf restricts the minimum number of samples in a leaf node, making sure none of the leaf nodes has less than 80 samples in it. If leaf nodes have less samples it implies we have grown the tree too much and trying to predict each sample very precisely, thus leading to overfitting

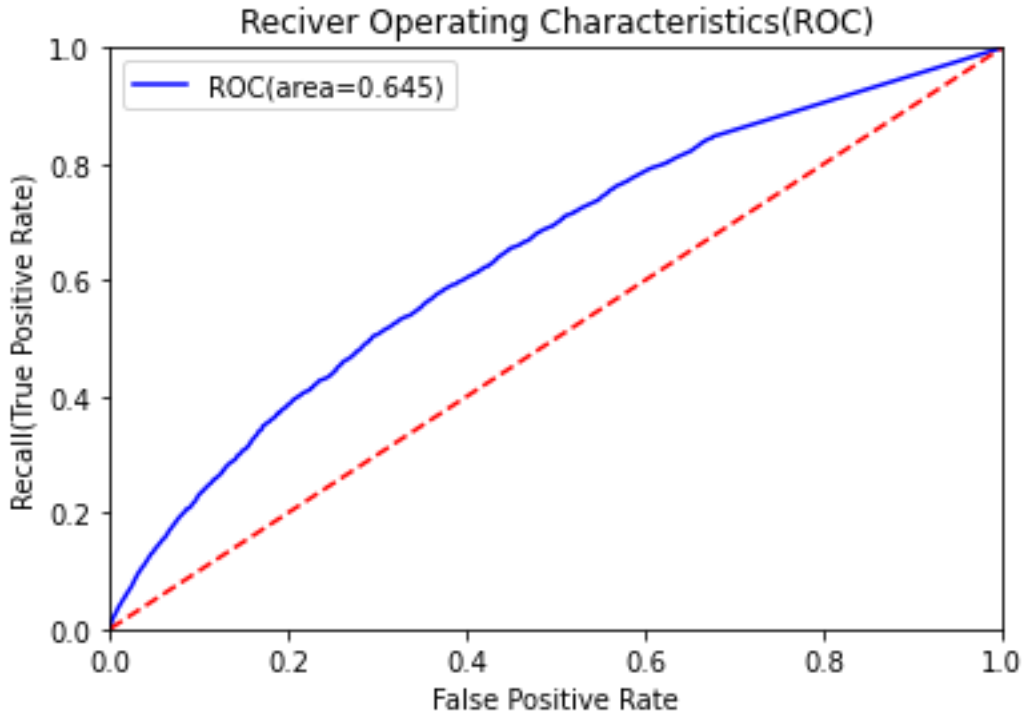
```
Test accuracy is 0.6407

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.65      0.76     53856
           1       0.15      0.55      0.24      6144

    accuracy                           0.64     60000
   macro avg       0.54      0.60      0.50     60000
weighted avg       0.85      0.64      0.71     60000
```


Confusion Matrix
35041
18815
2738
3406
True Class
Predicted Class
0
1
35000
30000
25000
20000
15000
10000
5000


Reciver Operating Characteristics(ROC)
ROC(area=0.645)
Recall(True Positive Rate)
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Random Forest Classifier

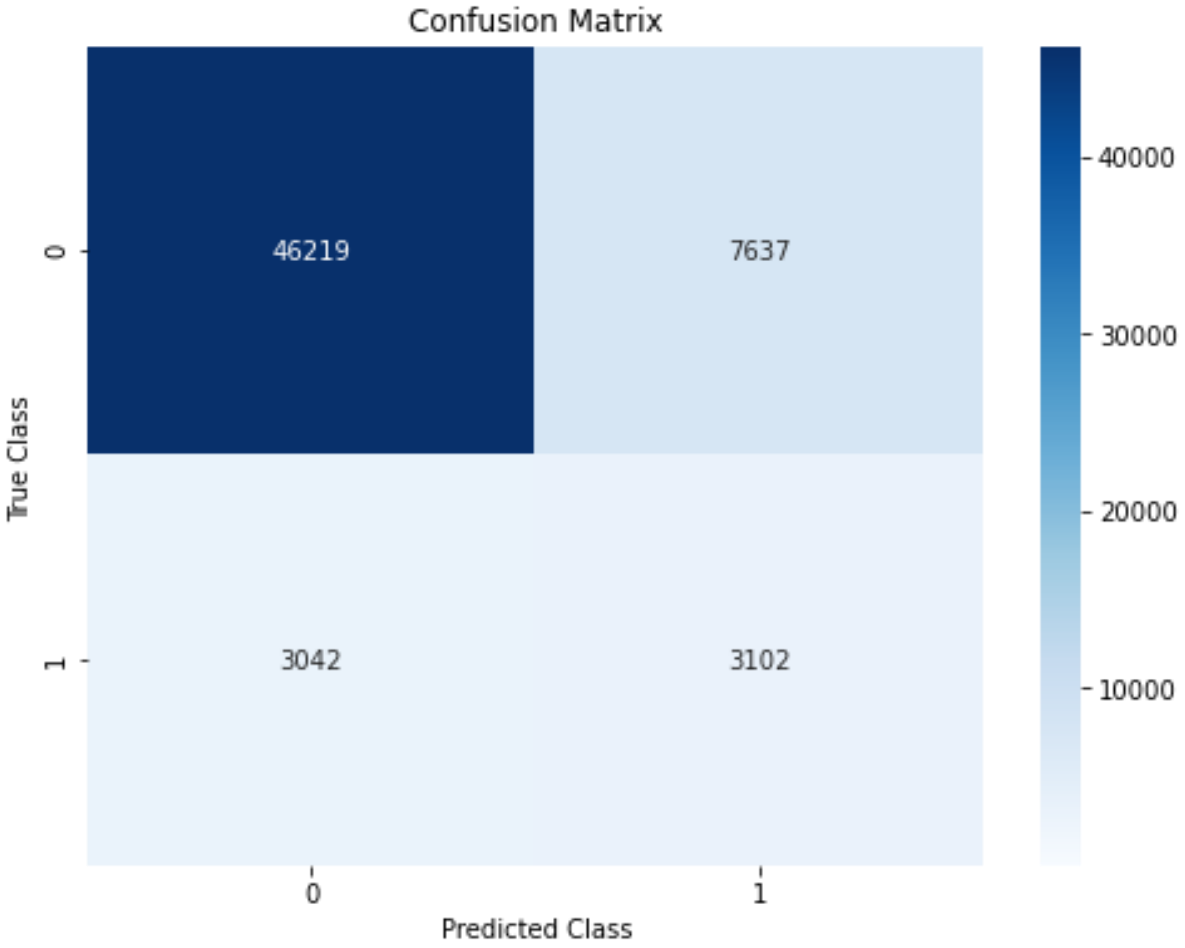
Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

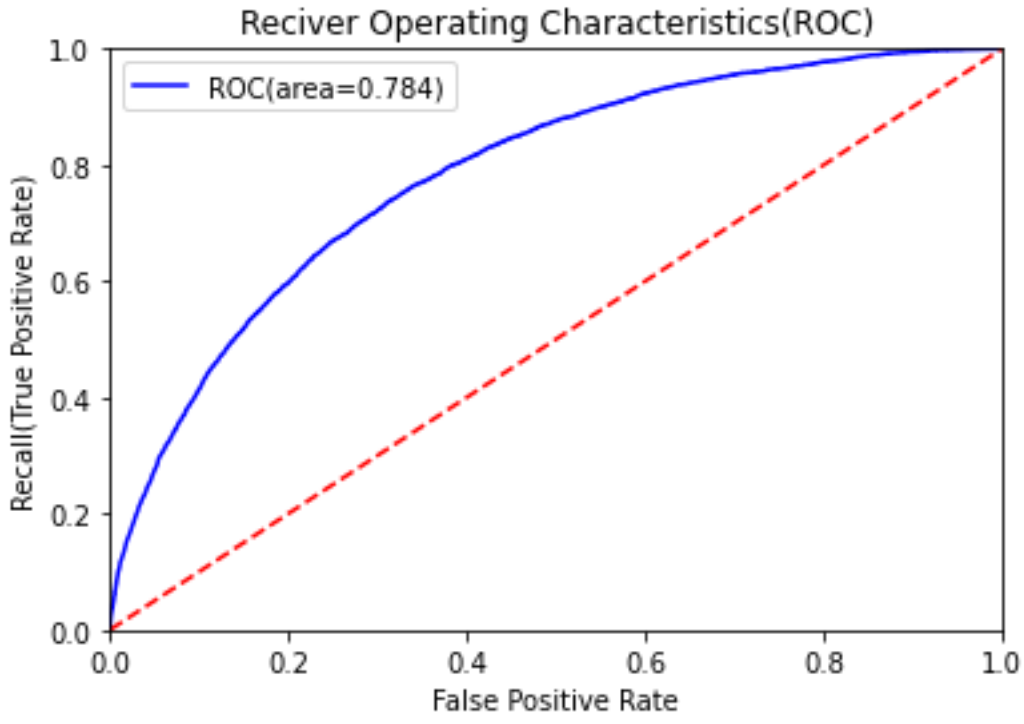
```
Test accuracy is 0.8220166666666666

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.86      0.90     53856
           1       0.29      0.50      0.37      6144

    accuracy                           0.82     60000
   macro avg       0.61      0.68      0.63     60000
weighted avg       0.87      0.82      0.84     60000
```


Confusion Matrix
46219
7637
3042
3102
0
1
True Class
0
1
Predicted Class
40000
30000
20000
10000


Reciver Operating Characteristics(ROC)
ROC(area=0.784)
Recall(True Positive Rate)
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

Light Gradient Boosting Method

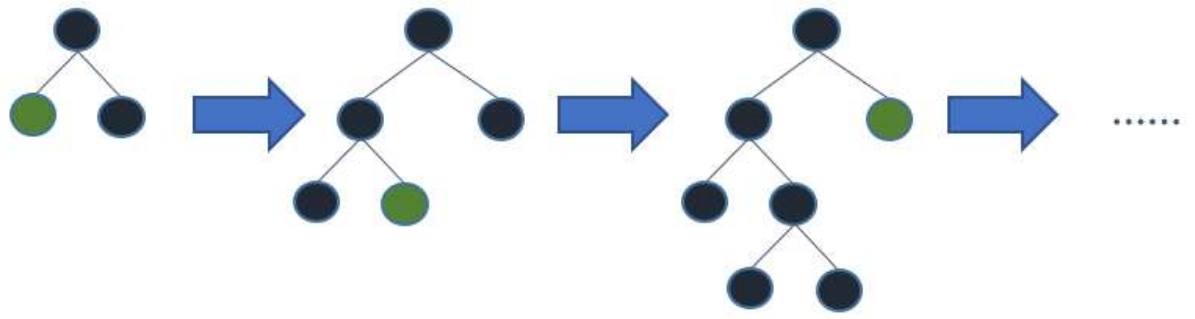
Light GBM is a gradient boosting framework that uses tree based learning algorithm. It grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

WHY USE LGB?

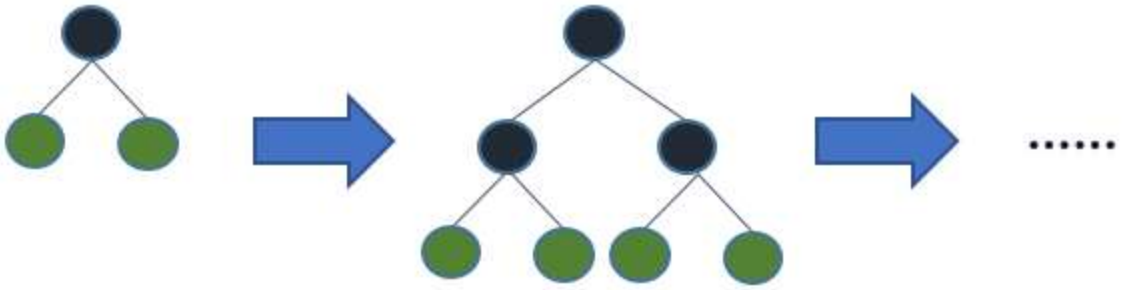
It is 'Light' because of its high speed. It can handle large data, requires low memory to run and focuses on accuracy of results and also supports GPU learning

Important points about tree-growth

- If we grow the full tree, **best-first (leaf-wise)** and **depth-first (level-wise)** will result in the same tree. The difference is in the order in which the tree is expanded. Since we don't normally grow trees to their full depth, order matters.
- Application of early stopping criteria and pruning methods can result in very different trees. Because leaf-wise chooses splits based on their contribution to the global loss and not just the loss along a particular branch, it often (not always) will learn lower-error trees "faster" than level-wise.
- For a small number of nodes, leaf-wise will probably out-perform level-wise. As we add more nodes, without stopping or pruning they will converge to the same performance because they will literally build the same tree eventually.



Leaf-wise tree growth



Level-wise tree growth

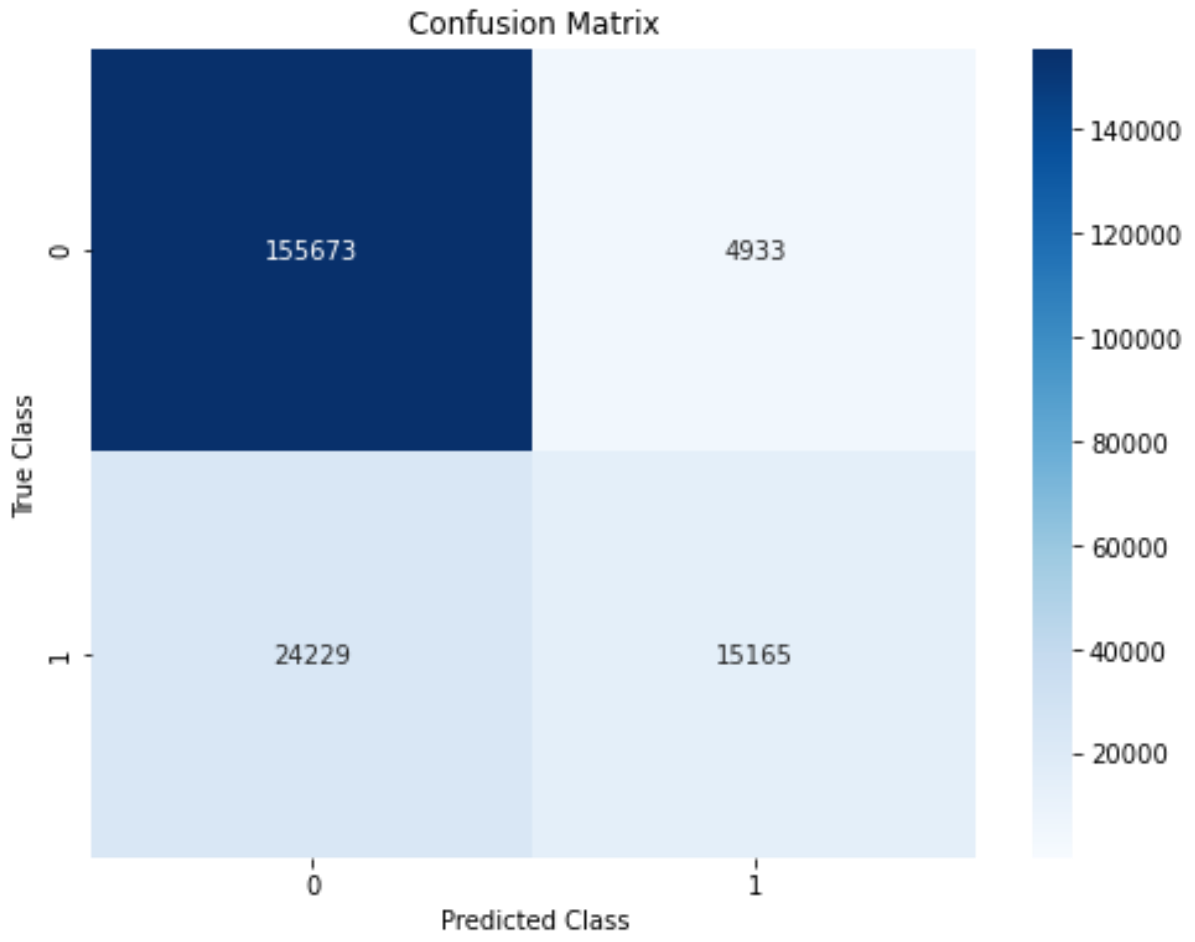
Accuracy Score for LightGBM is 0.854

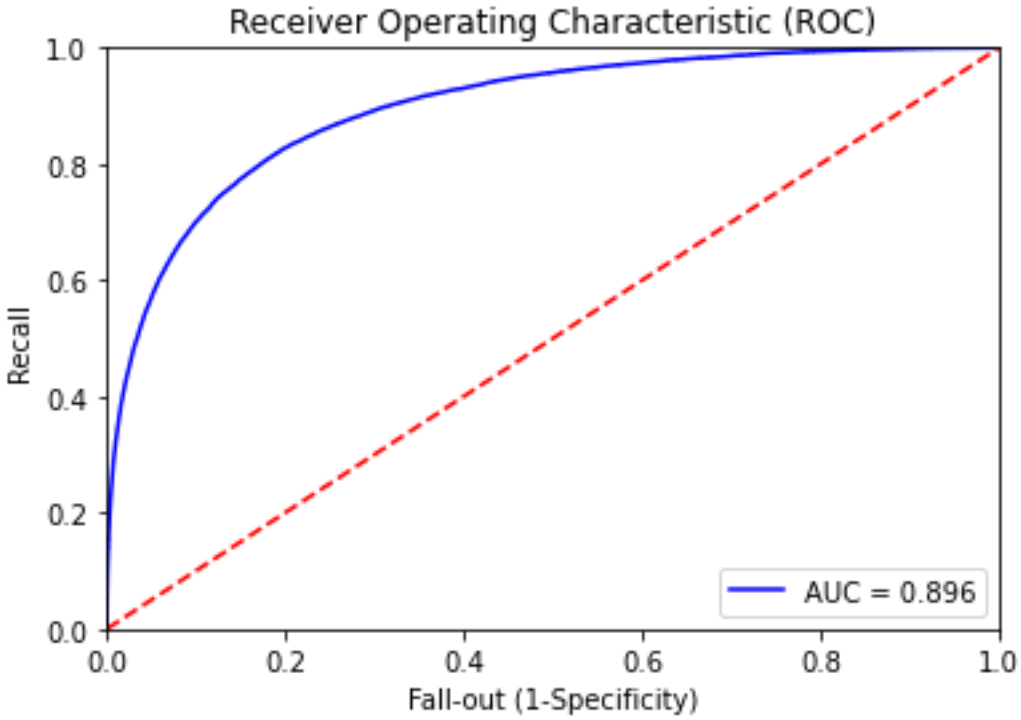
Precision Score for LightGBM is 0.754

Recall Score for LightGBM is 0.385

f1 Score for LightGBM is 0.51

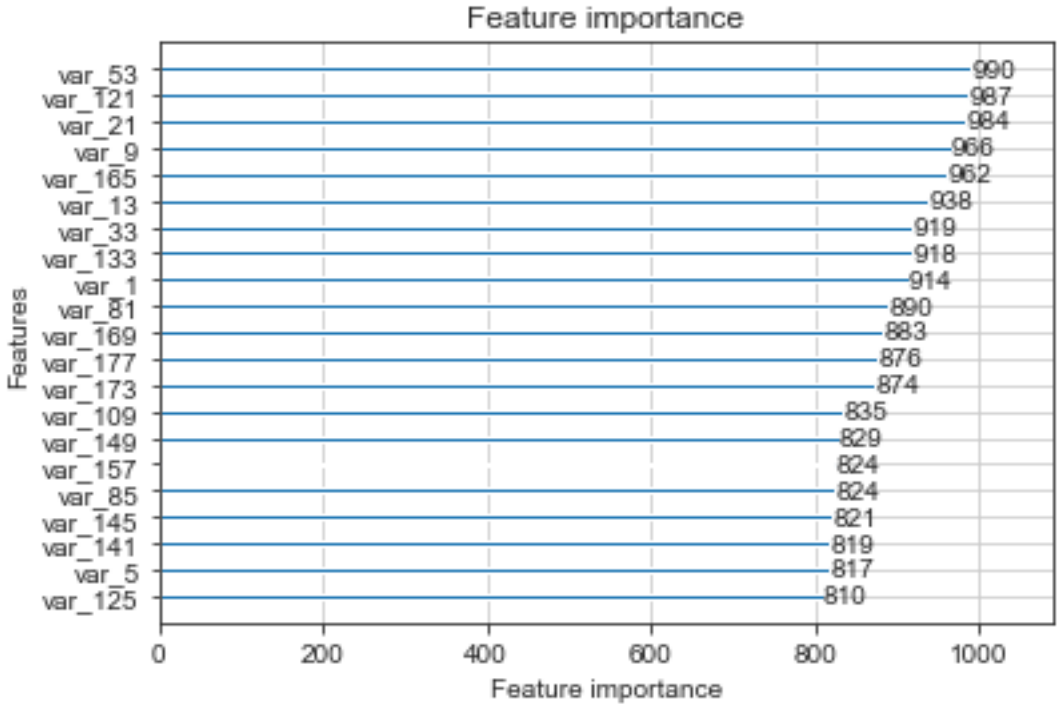
CV score: 0.896


Confusion Matrix
155673
4933
24229
15165
True Class
Predicted Class
0
1
140000
120000
100000
80000
60000
40000
20000


Receiver Operating Characteristic (ROC)
Recall
Fall-out (1-Specificity)
AUC = 0.896
0.0
0.2
0.4
0.6
0.8
1.0

Feature Importance

Feature importance refers to techniques that assign a score to input features based on how useful they are at predicting a target variable.
There are many types and sources of feature importance scores, although popular examples include statistical correlation scores, coefficients calculated as part of linear models, decision trees, and permutation importance scores.
Feature importance scores play an important role in a predictive modelling project, including providing insight into the data, insight into the model, and the basis for dimensionality reduction and feature selection that can improve the efficiency and effectiveness of a predictive model on the problem.



6. Conclusion

Upon training different machine learning algorithms, it is evident that LightGBM performs better compared to other models. Since it is highly imbalanced datset, we consider various parameters like accuracy score, confusion matrix, classification report, AUC-ROC scores etc to predict which model performs best on the dataset.