Santander Transaction Prediction

Vishal Maurya

14 August 2019

Contents

# 1 Introduction

1.1 Problem Statement

1.2 Data

# 2 Methodology

2.1 Exploratory Data Analysis

2.1.1 Descriptive Analysis

2.1.1.1 Missing value analysis

2.1.2 Visualization

2.1.2.1 Target Variable: Count plot / Pie Chart

2.1.2.2 Outlier Analysis: Boxplot

2.1.2.3 Attributes Distributions and trends

2.1.2.4 Correlation Analysis / Heatmap

2.2 Data Preprocessing and Analysis

2.2.1 Outlier Handling

2.2.2 Principal component analysis (PCA)

2.2.3. Feature Selection

2.2.4 Feature Engineering

2.3 Modeling

2.3.1 Model Selection

2.3.1.1 Without StandardScale

2.3.1.2 With StandardScale

* + - * 1. SMOTE or ROSE
        2. LogisticRegression + ROSE

2.2.4 LightGBM.

2.2.4.1 Simple LightGBM

2.2.4.2 SMOTE LightGBM

# 3 Conclusion

3.1 Model Evaluation

3.1.1 Confusion Matrix

3.1.2 ROC\_AUC\_score

3.2 Model Selection

Appendix A - Extra Figures

Appendix B – Complete Python and R Code

Python Code

R code

References

# Chapter 1

Introduction

# 1.1 Problem Statement

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?

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

# 1.2 Data

We are provided with an anonymized dataset containing numeric feature variables, the binary target column, and a string ID\_code column. The task is to predict the value of target column in the test set.

Dataset:

1. train.csv - the training set.
2. test.csv - the test set.

The Train dataset contains 200000 observations of 202 columns.

1. The first two columns in the dataset store the unique ID\_code numbers of the observations and the corresponding "target" transaction prediction, respectively.
2. The columns 2-202 contain 200 real-value features that have been captured which can be used to build a model to predict weather a transaction done by customer.

The Test dataset contains 200000 observations of 201 columns.

1. The first column in the dataset store the unique ID\_code numbers of the observations and the corresponding variables columns.
2. The columns 1-201 contain 200 real-value features that have been captured which will be used to test our model prediction on weather a transaction done by customer.

In this project, our task is to build classification models which would be used to predict which customers will make a specific transaction in the future. Given below is a sample of the Santander customer transaction dataset:

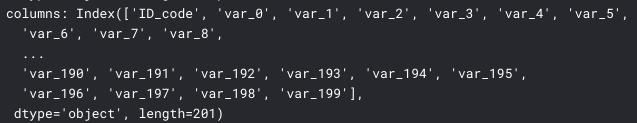
Table 1.1: Train dataset (Columns:1-202)



Table 1.2: Test Dataset (Columns: 1-201)



Table 1.3: Predictor Variables



# Chapter 2

Methodology

# 2.1 Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) is a very important step which takes place after feature engineering and acquiring data and it should be done before any modeling. This is because it is very important for a data scientist to be able to understand the nature of the data without making assumptions. The results of data exploration can be extremely useful in grasping the structure of the data, the distribution of the values, and the presence of extreme values and interrelationships within the data set. It involves the loading dataset, target classes count, data cleaning, typecasting of attributes, missing value analysis, Attributes distributions and trends.

> Purpose of EDA:

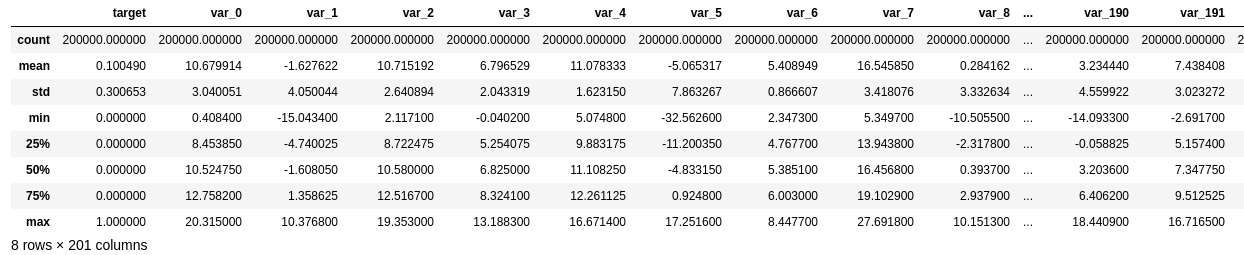
1. Summarize the statistics and visualization of data for better understanding. Crubing indication for tendencies of the data, its quality and to formulate assumptions and the hypothesis of our analysis.

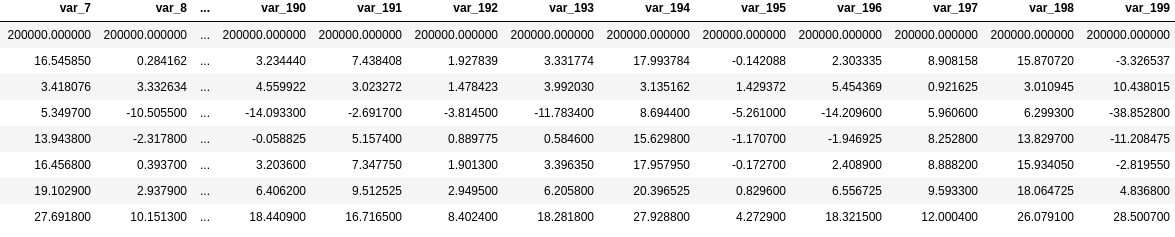
2. To create an overall picture of the data with basic statistical description and aspects, and identify

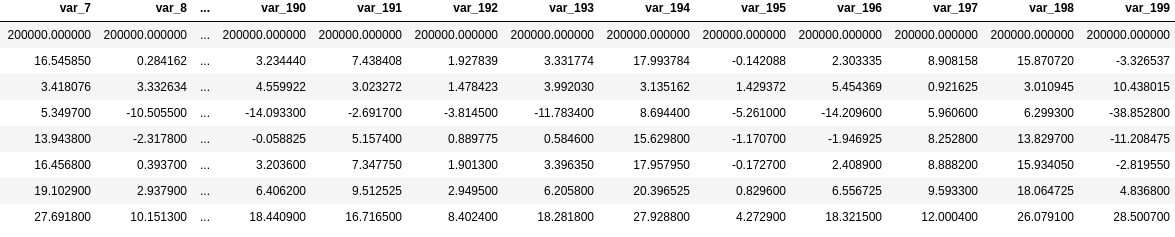
# 2.1.1 Descriptive Analysis

It is a summary statistic that quantitatively describes or summarizes features of a collection of information, process of condensing key characteristics of the data set into simple numeric metrics. Some of the common metrics used are mean, standard deviation, and correlation.

Table 2.1: Train description







**Observation:**

1. Here, we have second column "target", which is our objective to assert.

2. As can be seen above, except for the target all other features are of type float64

3. Our target is int64 type with only two values

\* Standard deviation is relatively large.

\* min, max, mean, sdt values for train and test data looks quite close

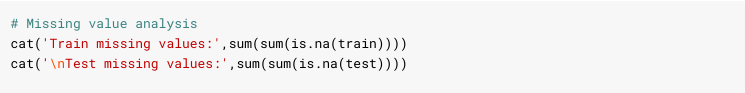
\* mean values are distributed over a large range.

# 2.1.1.1 Missing value analysis

In this, we have to find out any missing values are present in dataset. We have not found any missing values in both train and test data.



Python and R code as follows:



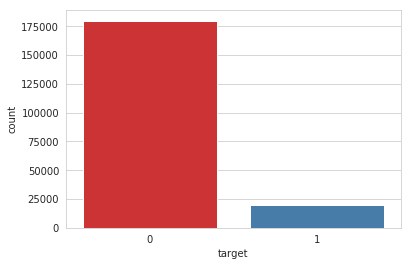


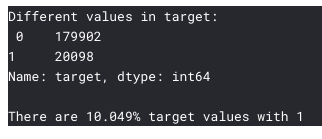
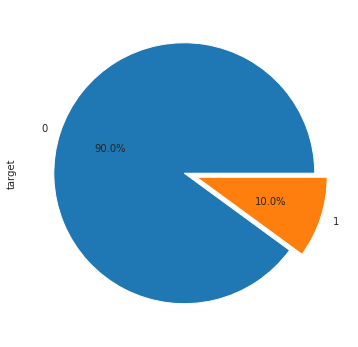
# 2.1.2 Visualization

It is the process of projecting the data, or parts of it, into Cartesian space or into abstract images. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral to yourself and stakeholders than measures of association or significance. In the data mining process, data exploration is leveraged in many different steps including preprocessing, modeling, and interpretation of results.

One of our main goals for visualizing the data here, is to observe which features are most intiutive in predicting target. The other, is to draw general trend, may aid us in model selection and hyper parameter selection.

# 2.1.2.1 Target Variable: Count plot / Pie Chart





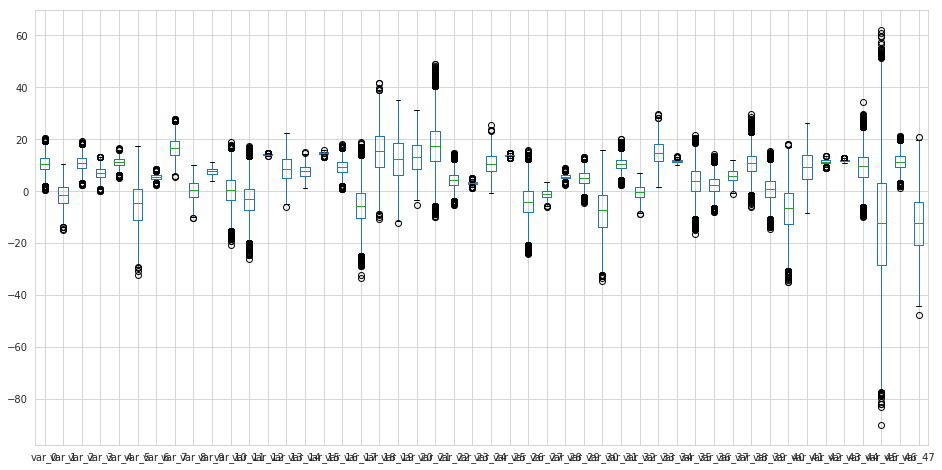
**Observation**:

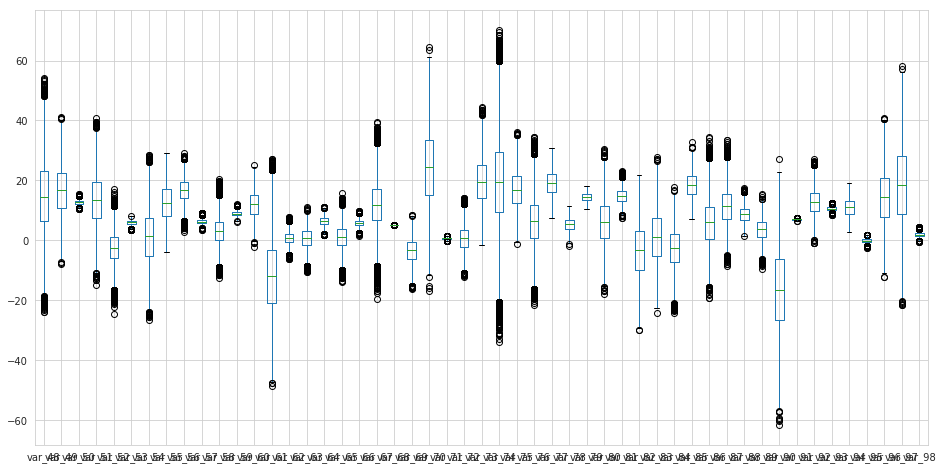
Only about 10% of total target is belong to class 1, therefore , this train dataset is imbalanced, hence need different sampling methods than random sampling.

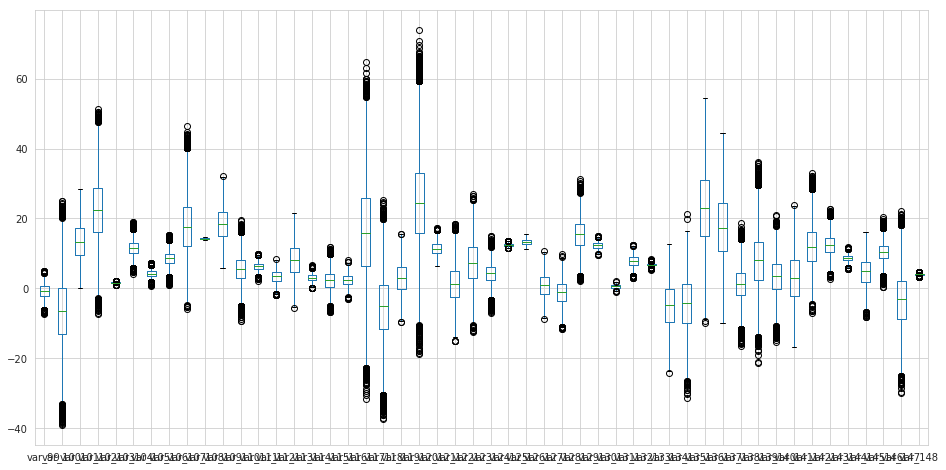
1. We have an imbalanced class problem. where 90% of the data is the number of customers those will not make a transaction and 10% of the data is those who will make a transaction. The number of customers that will not make a transaction is much higher than those that will.
2. The dataset is unbalanced with respect to the target, need to consider resampling methods.

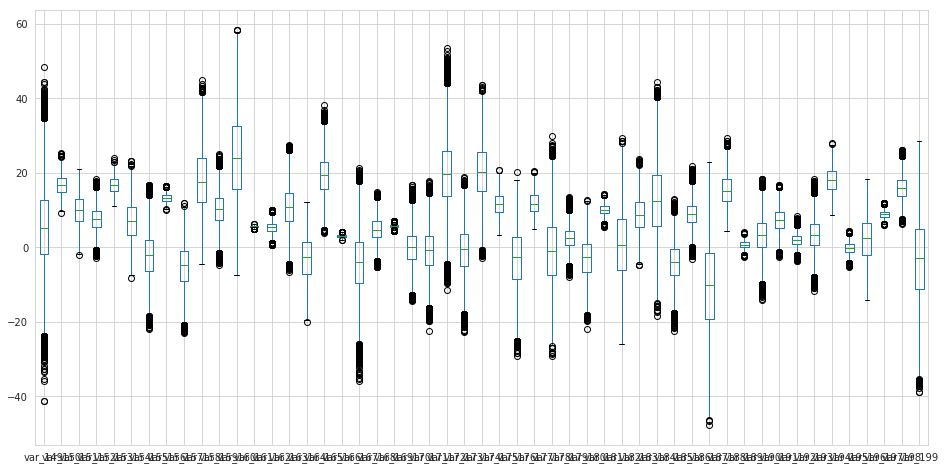
# 2.1.2.2 Outlier Analysis: Boxplot

A boxplot is a graph that gives you a good indication of how the values in the data are spread out. Although boxplots may seem primitive in comparison to a histogram or density plot, they have the advantage of taking up less space, which is useful when comparing distributions between many groups or datasets.









Observations:

1. Most of the data have outlier and range of the data variables is high.
2. Data need to be free from ouliers and need to be scaled before applying any outlier senstive model algorithms.

# 2.1.2.3 Attributes Distributions and trends

##### Distribution of train attributes



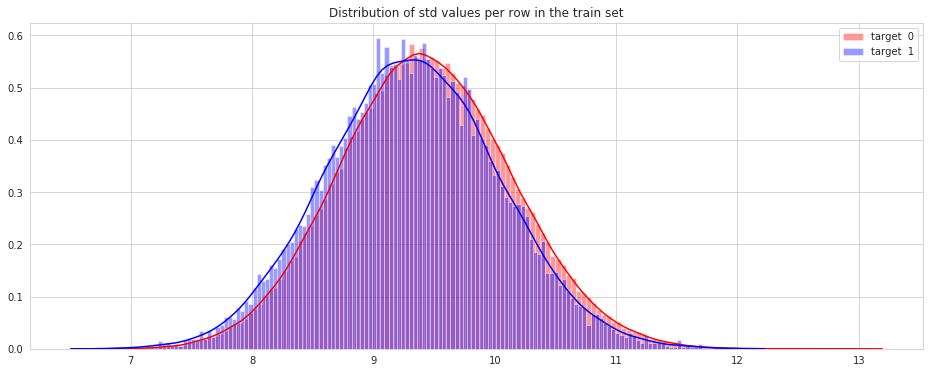


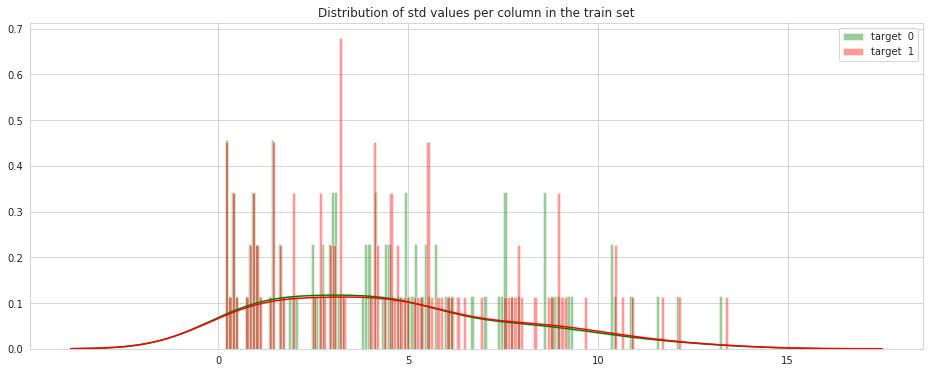
**Observations:**

1. There is a considerable number of features with significant different distribution for the two target values. For example, var\_0, var\_1, var\_2, var\_5, var\_9, var\_13, var\_106, var\_109, var\_139 and many others.
2. We can observe that there 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\_17, var\_35 etc.
3. if we look closely var\_2, var\_9, var\_12, var\_13, var\_26, var\_40, var\_53, var\_81, all of these variables have a bump of frequency that matches the rising of the probability of making a transaction. if pdf (target = 1) - pdf (target = 0) > 0, then there is a high probability of the client making a transfer.

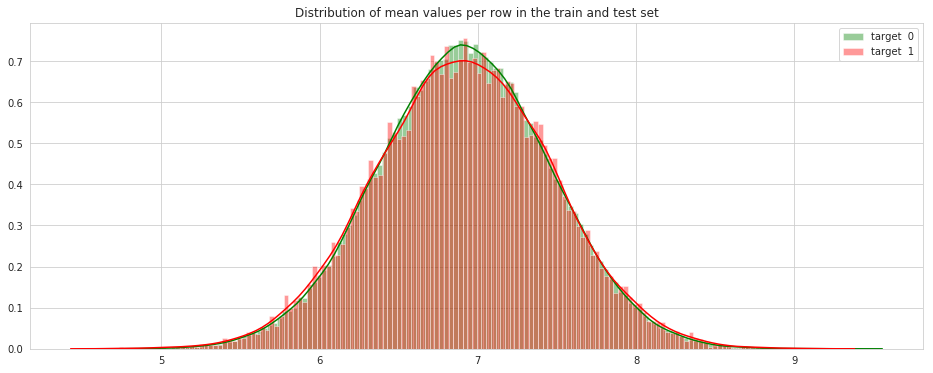
# Distribution of Aggregates along the column and row

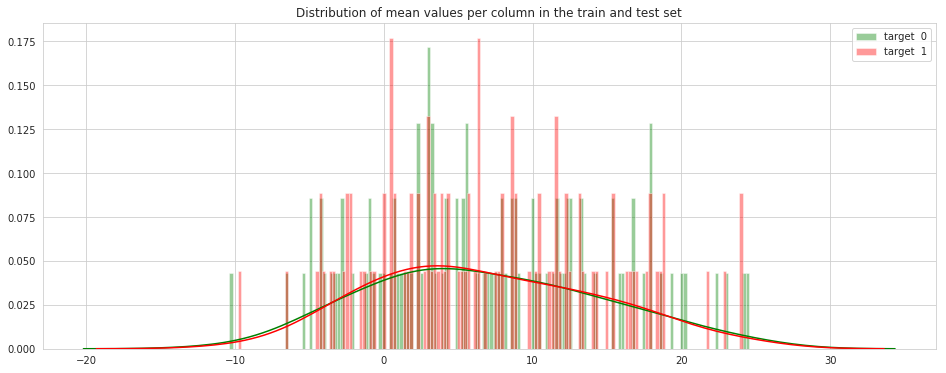
Let us look distribution of std values:



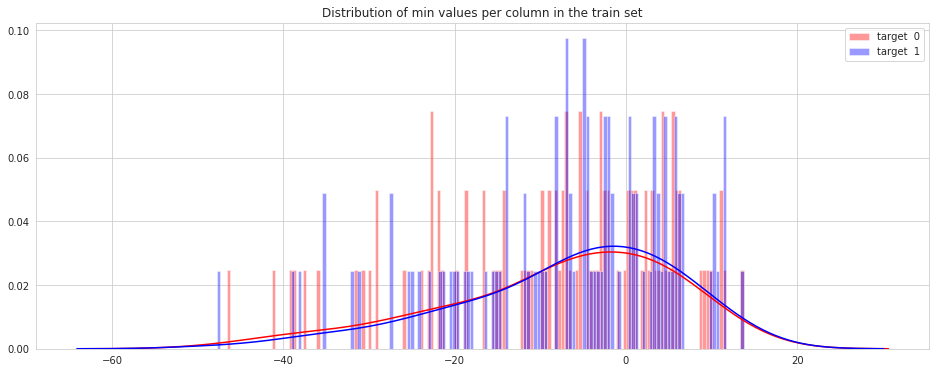
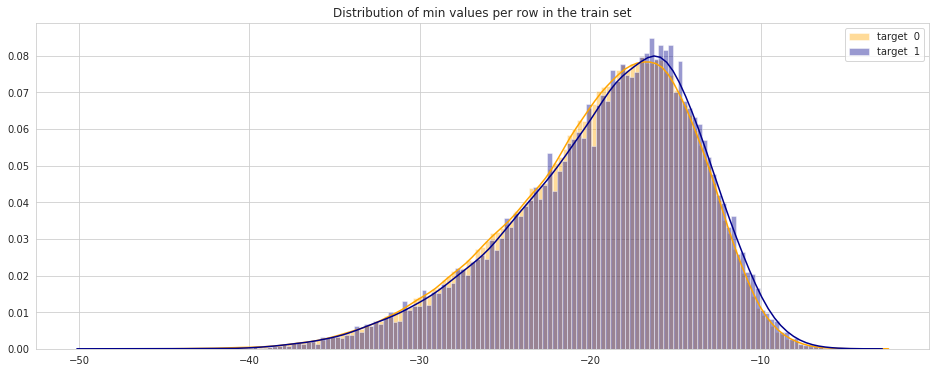


Let us look distribution of mean values:

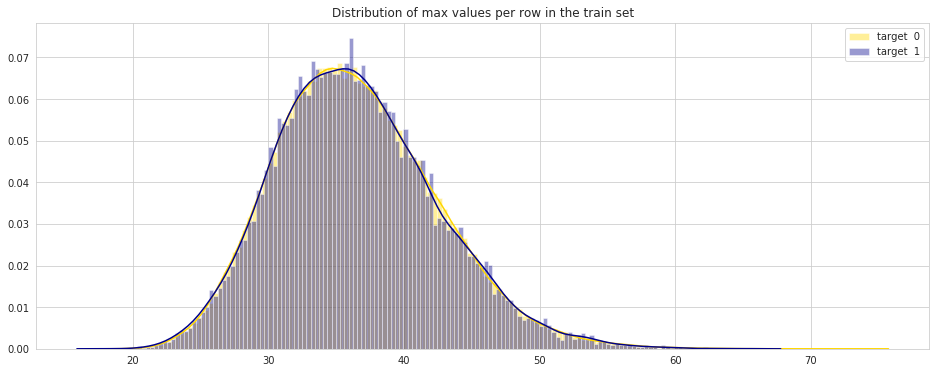


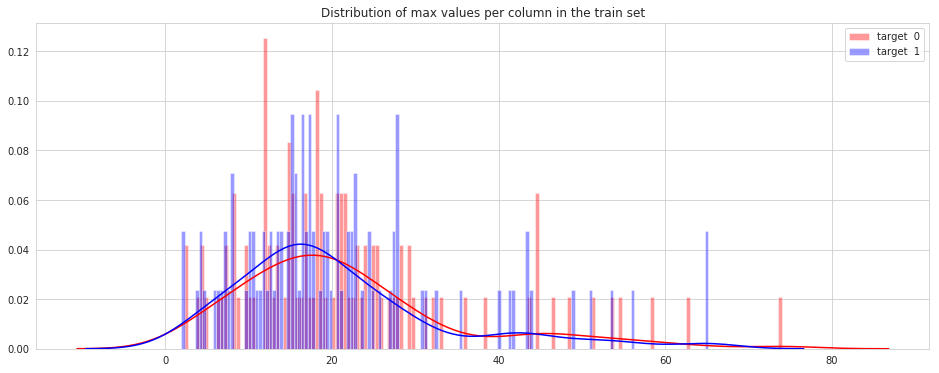


Let us look distribution of min values

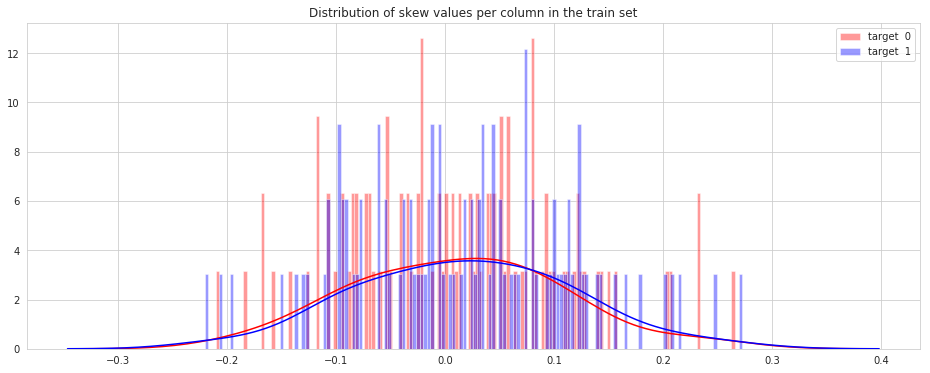
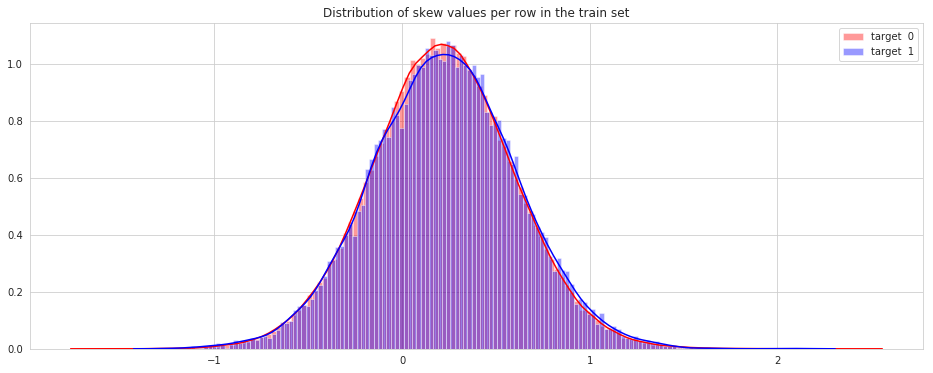


Let us look distribution of max values:

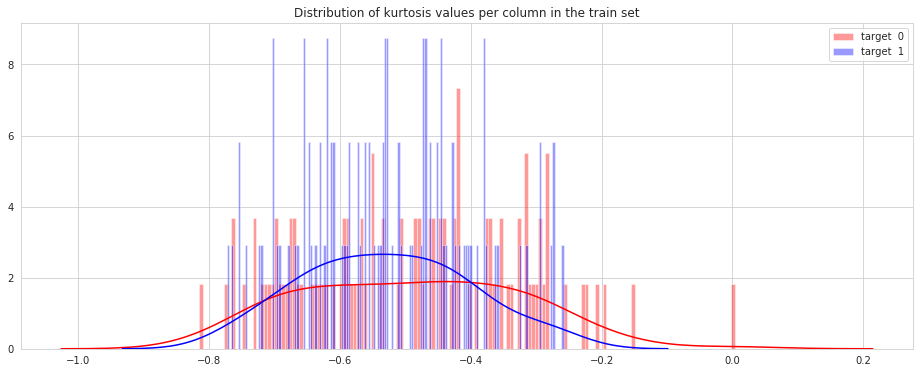
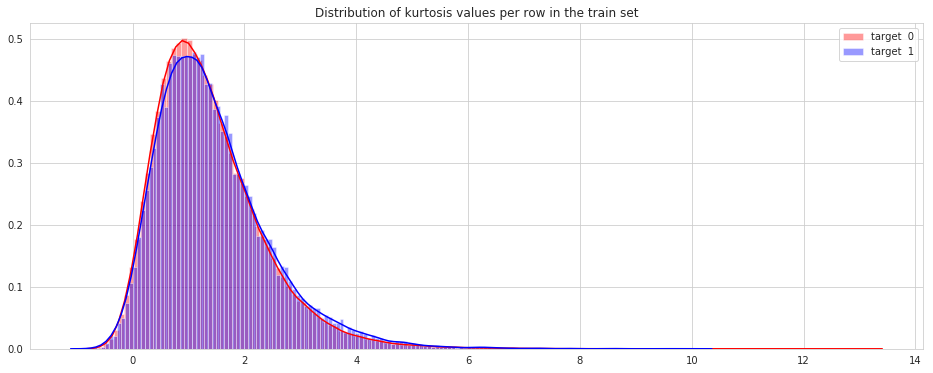




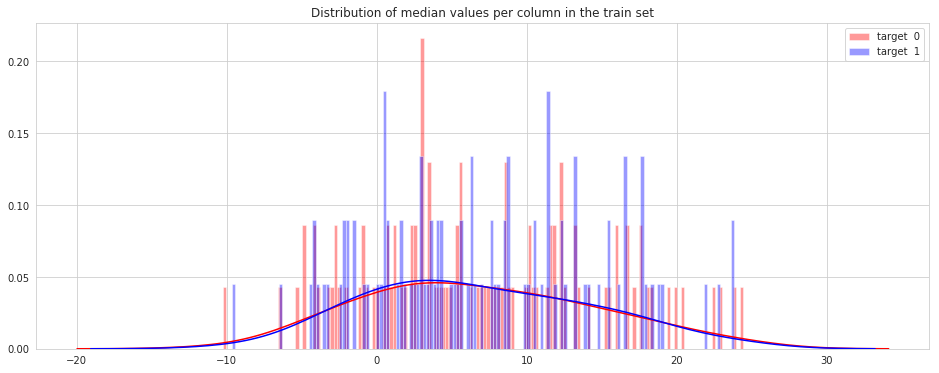
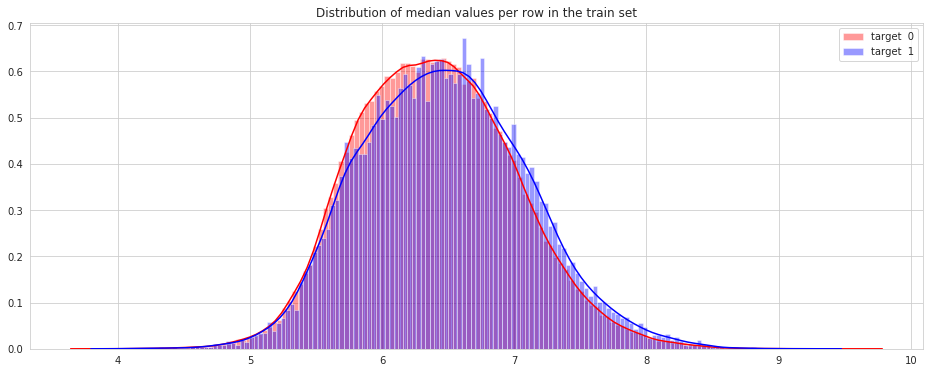
Let us look distribution of skewness values:



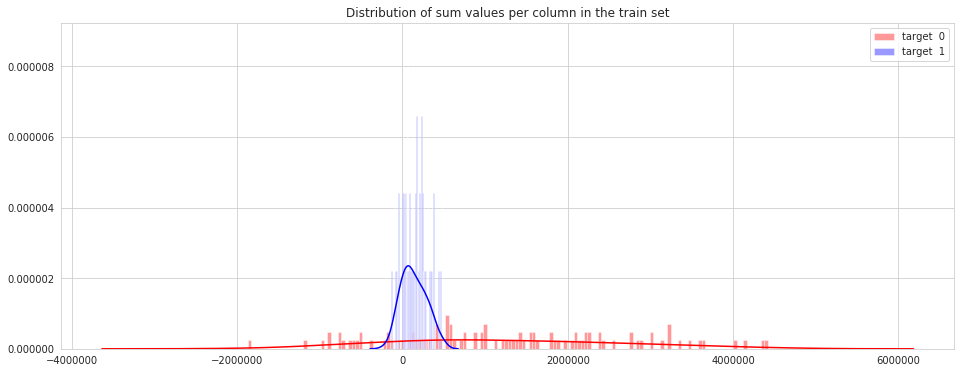
Let us look distribution of kurtosis values:



Let us look distribution of median values:



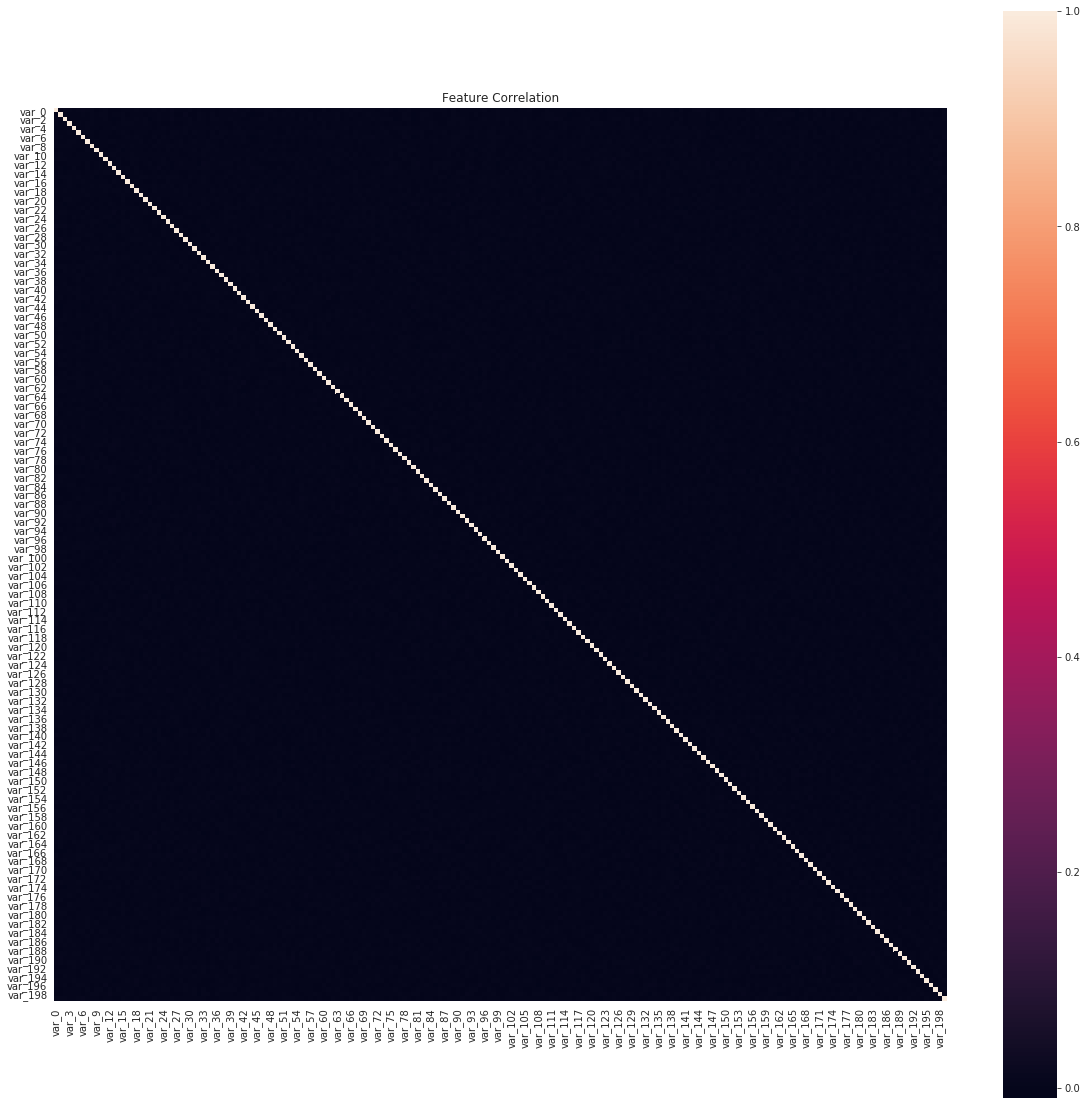
Let us look distribution of sum values:



# 2.1.2.4 Correlation Analysis / Heatmap

First, we used the method corr() on a DataFrame that calculates the correlation between each pair of features. Then, we pass the resulting correlation matrix to heatmap() from seaborn, which renders a color-coded matrix for the provided values:

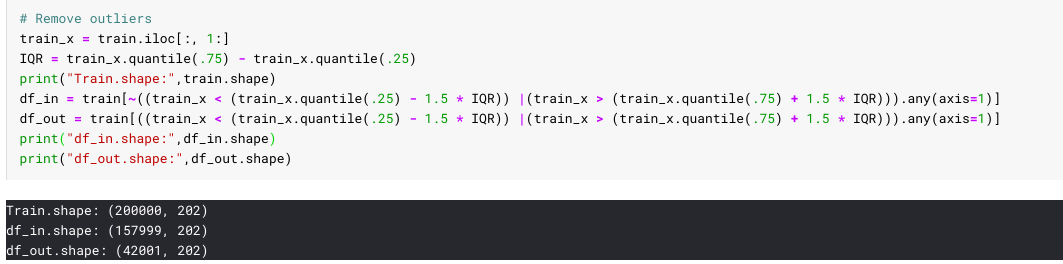


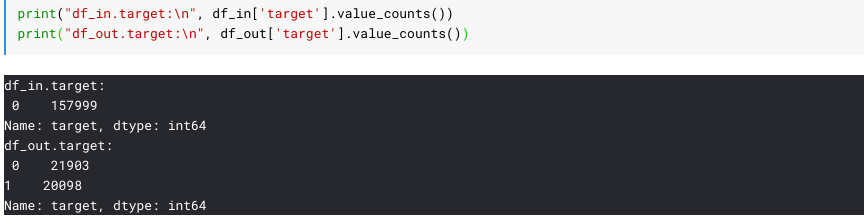
**Observation:**

* Maximum corr within all variables correlations is 0.009713 which is inferior, and hence, all the variables are almost independent i.e. no correlation between them.

# 2.2 Data Preprocessing and Analysis

# 2.2.1 Outlier Handling





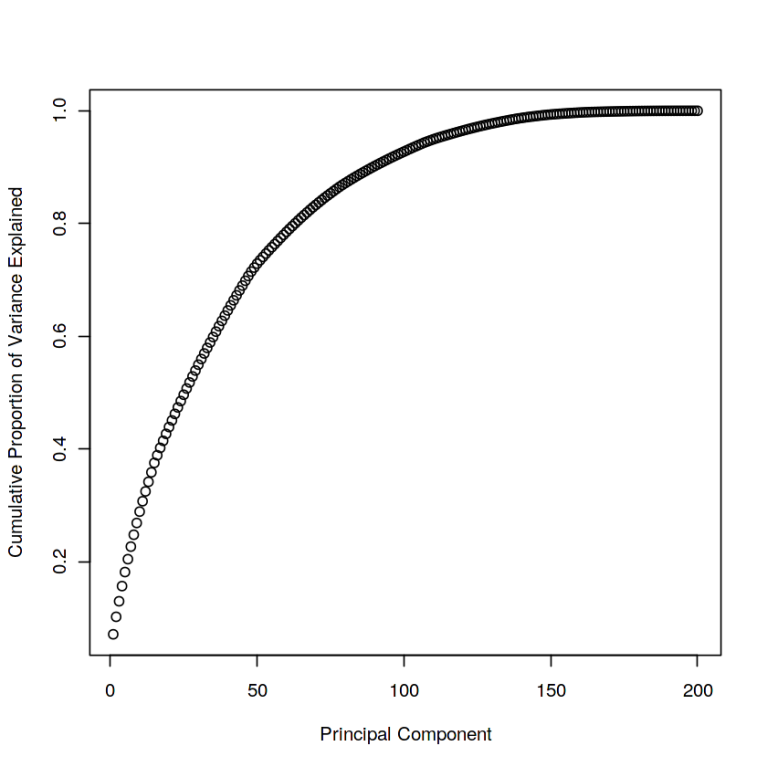
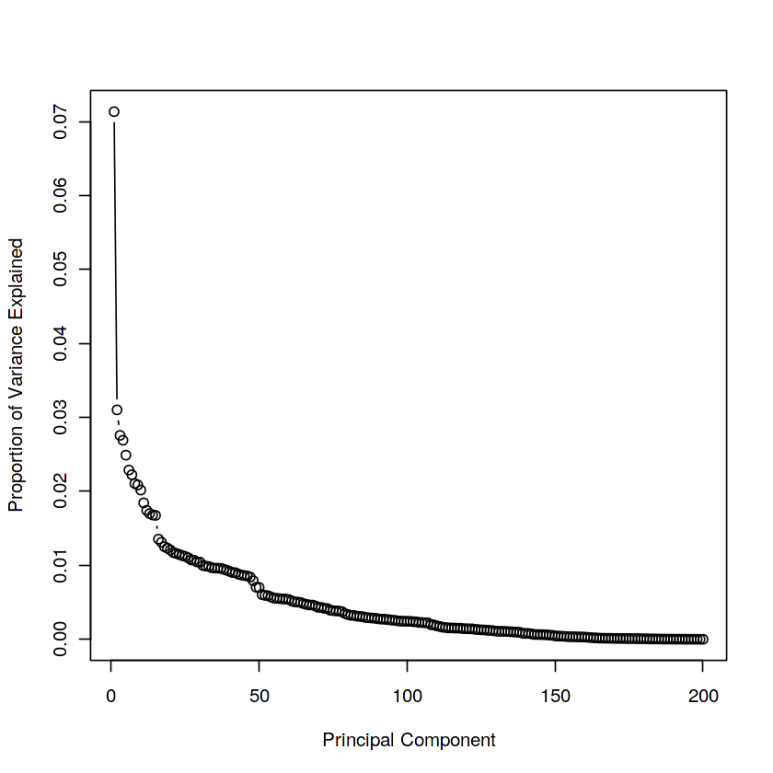
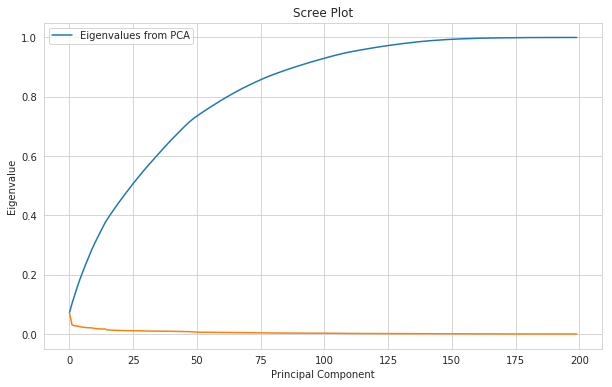
**Obeservation:**

1. After separating outliers and inliers with IQR method we found that all the target variables with label as one are outliers.
2. Outliers present in our data, are meaningful and thus can’t be removed.

# 2.2.2 Principal component analysis (PCA)

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional sub-space. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation.

In order to decide how many principal components should be retained, it is common to summarize the results of a principal components analysis by making a scree plot. Using Elbow Method to determine the right number of components to be retain.



**Observation:**

1. Only 100 components can explain our 50+ % of features.
2. However, since we found that the correlation between different features in the training dataset is not that significant, so using PCA might not be meaningful.

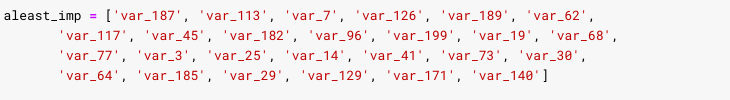
# 2.2.3. Feature Importance

We used ML algorithms to find the top features from data variables. They can serve as a starting point to discover their nature and for trying to understand the data. In addition, they may yield some ideas on how to generate new features. We used GridSearchCV strategy for best estimation, since, our target is imbalanced.

We investigated using the following algorithm:

* Random Forest Classifier

Below are the 25 least important feature according to our RFC model.

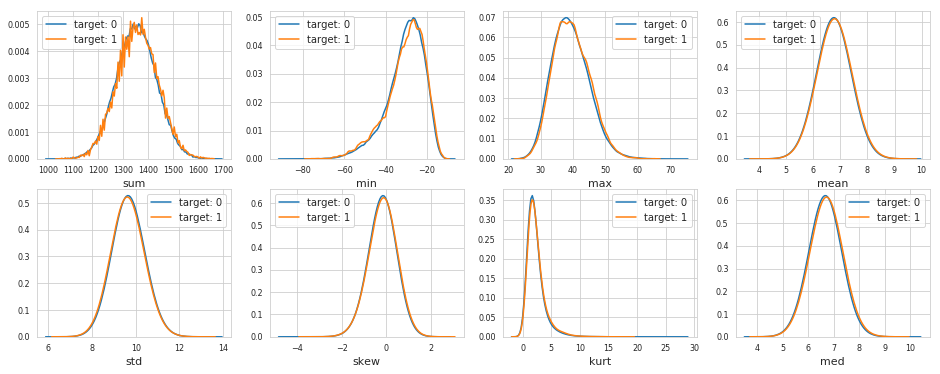


# 2.2.4 Feature Engineering

We applied FE to our data for removing columns of least importance, but results were drastically abhorrent, so we skipped it. Later, we added sum FE columns to our dataset.



Let’s take a look to our added features:



# 2.2 Modeling

# 2.2.1 Model Selection

Model selection is the process of choosing between different machine learning approaches - e.g. SVM, logistic regression, etc. - or choosing between different hyperparameters or sets of features for the same machine learning approach - e.g. deciding between the polynomial degrees/complexities for linear regression.

The dependent variable can fall in either of the four categories:

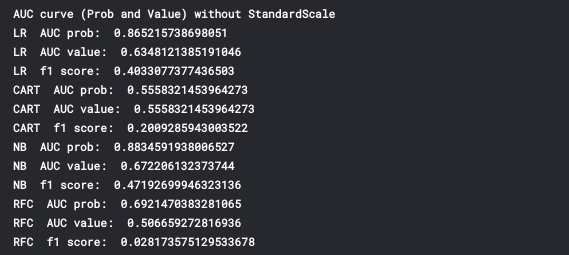
Nominal, Ordinal, Interval, Ratio

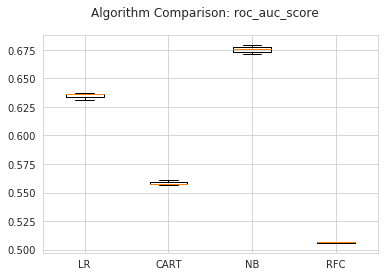
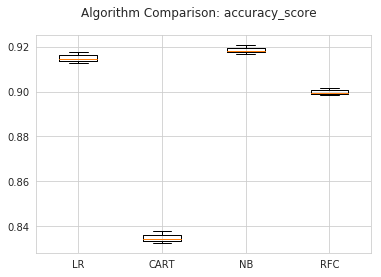
If the dependent variable is Nominal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio, the normal method is to do a Regression analysis, or classification after binning.

We used the following models for the initial evaluation of the right algorithm:

1. Logistic Regression
2. Random Forest Classifier
3. CART
4. Naïve Bayes

## 2.2.1.1 Without StandardScale





Models roc\_auc\_scores are as follows:

1. LR :: 0.634542682834447
2. CART :: 0.5582189398386278
3. NB :: 0.6753558903368968
4. RFC :: 0.5062861752933414

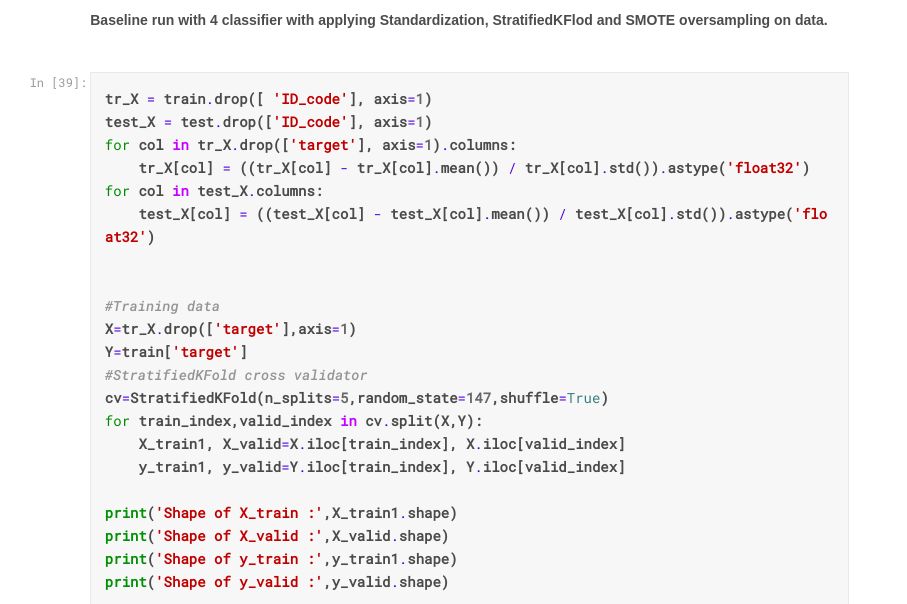
## 2.2.1.2 With StandardScale

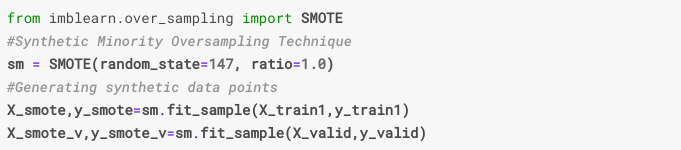
## StandardScale SMOTE / ROSE

We are dealing with imbalanced class problem, hence, we choose to oversampling our small class by using SMOTE / ROSE models.

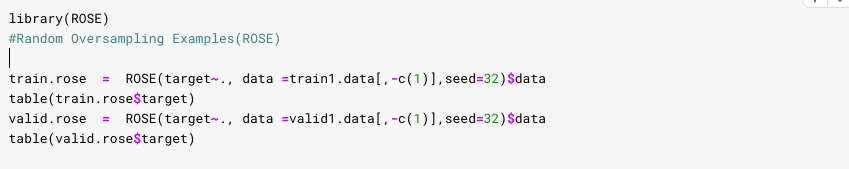
In SMOTE, the algorithm looks at n nearest neighbors, measures the distance between them and introduces a new observation at the center of n observations. While proceeding, we must keep in mind that these techniques have their own drawbacks such as:

* undersampling leads to loss of information
* oversampling leads to overestimation of minority class

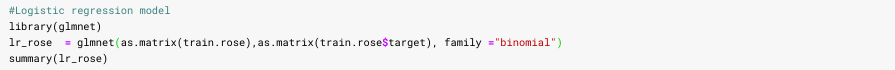


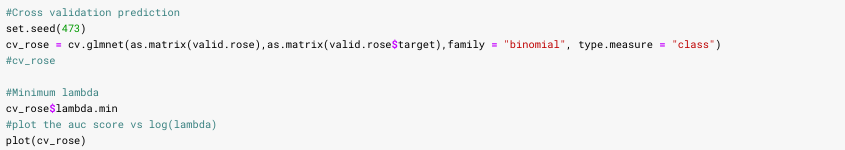


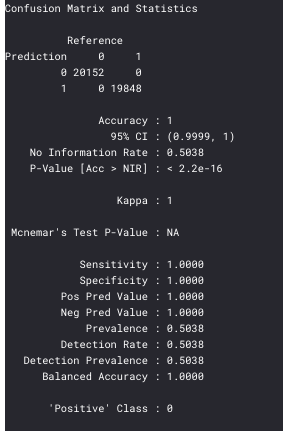
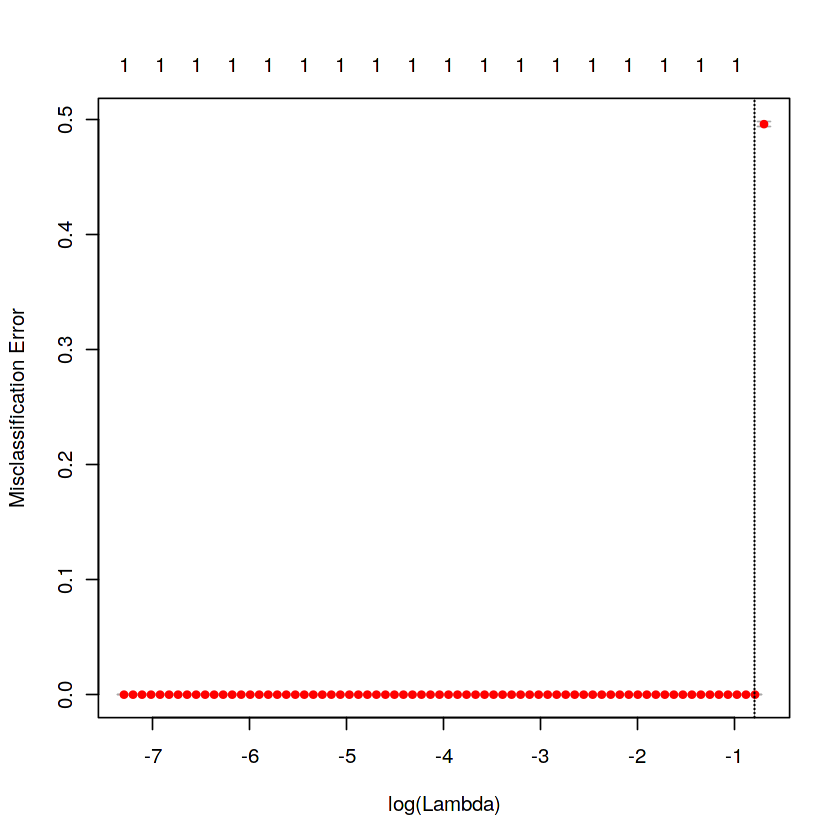
ROSE (Random Over Sampling Examples) package helps us to generate artificial data based on sampling methods and smoothed bootstrap approach. This package has well defined accuracy functions to do the tasks quickly.



1. LogisticRegression + ROSE







**Observation:**

* **LogisticRegression with ROSE gives 100 % accurate result, which it not possible, hence, disgarding the model.**

# 2.2.4 LightGBM

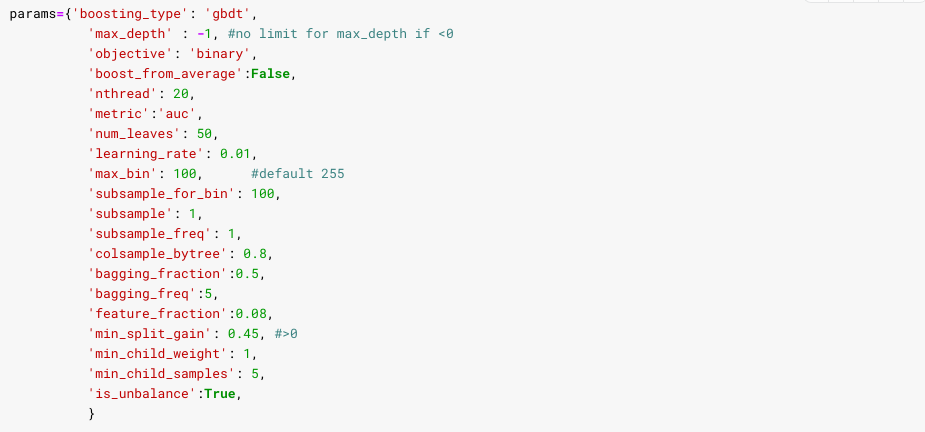
Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks.

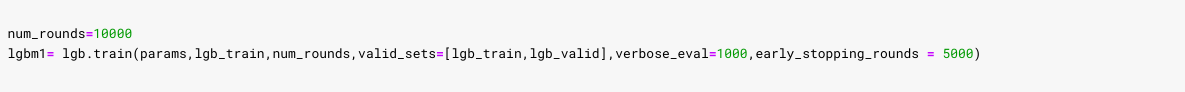
Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word ‘Light’.

## 2.2.4.1 Simple LightGBM

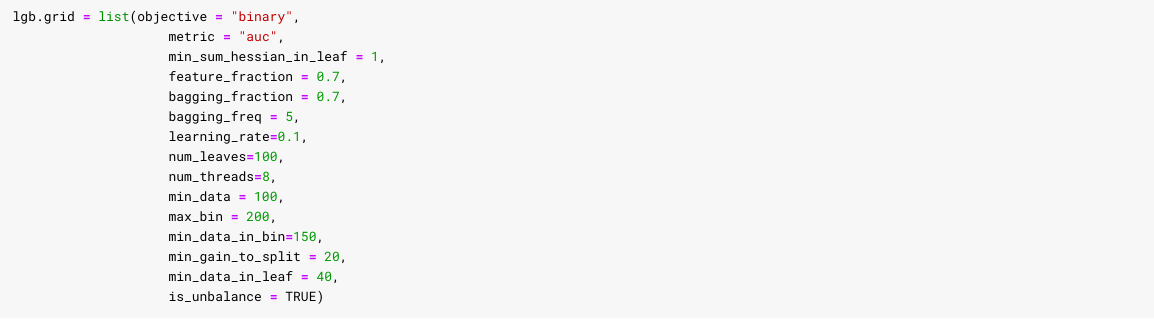
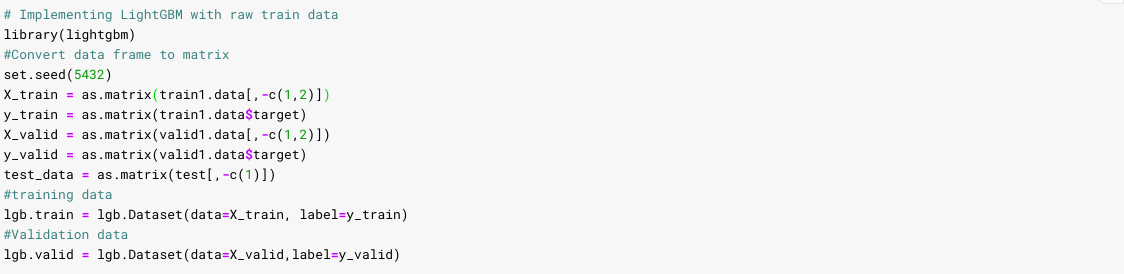
Python:







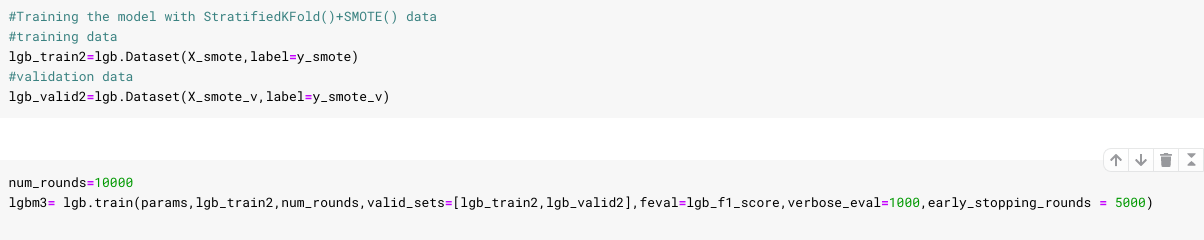
R:





## 2.2.4.2 SMOTE LightGBM../ ROSE

Python:



R:



# CHAPTER 3

Conclusion

# 3.1 Model Evaluation

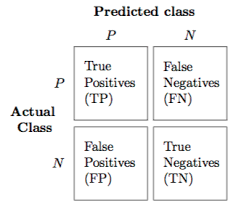
Now, we have 5 models for predicting the target variable, but we need to decide which model better for this project. There are many metrics used for model evaluation. Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

For classification problems, the confusion matrix used for evaluation. But, in our case the data is imbalanced. So, roc\_auc\_score is used for evaluation.

Receiver Operating Characteristic (ROC) curve

In statistical modeling and machine learning, a commonly reported performance measure of model accuracy for binary classification problems is Area Under the Curve (AUC).

To understand what information the ROC curve conveys, consider the so-called confusion matrix that essentially is a two-dimensional table where the classifier model is on one axis (vertical), and ground truth is on the other (horizontal) axis, as shown below. Either of these axes can take two values (as depicted)

In an ROC curve, we plot “True Positive Rate” on the Y-axis and “False Positive Rate” on the X-axis, where the values “true positive”, “false negative”, “false positive”, and “true negative” are events (or their probabilities) as described above. The rates are defined according to the following:

True positive rate (or sensitivity)}: tpr = tp / (tp + fn)

False positive rate: fpr = fp / (fp + tn)

True negative rate (or specificity): tnr = tn / (fp + tn)

In all definitions, the denominator is a row margin in the above confusion matrix. Thus, one can express

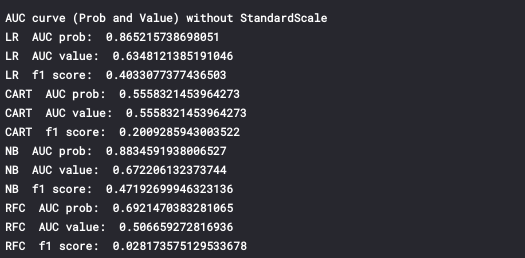
the true positive rate (tpr) as the probability that the model says "P" when the real value is indeed "P" (i.e., a conditional probability). However, this does not tell you how likely you are to be correct when calling "P" (i.e., the probability of a true positive, conditioned on the test result being "P").

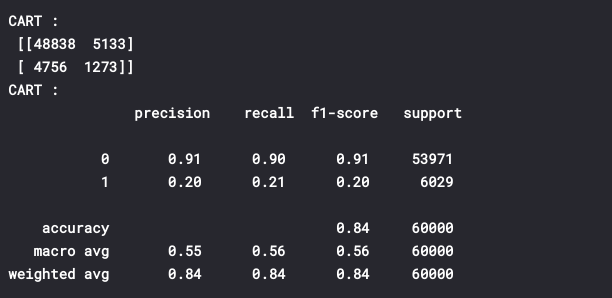
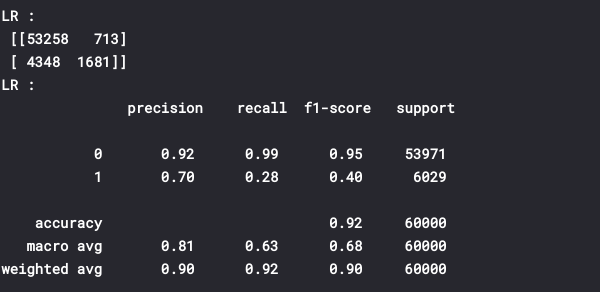
In this project, we are using two metrics for model evaluation as follows,

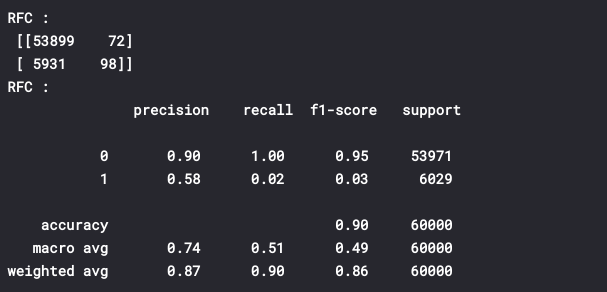
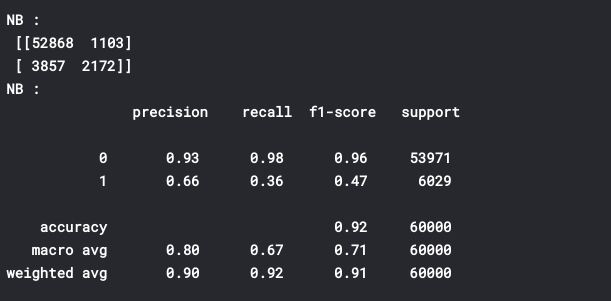
# 3.1.1 Confusion Matrix / Classification Report

Python:

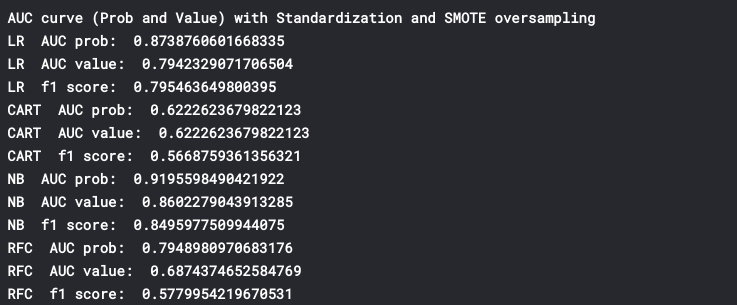
1. **AUC curve (Prob and Value) without StandardScale**

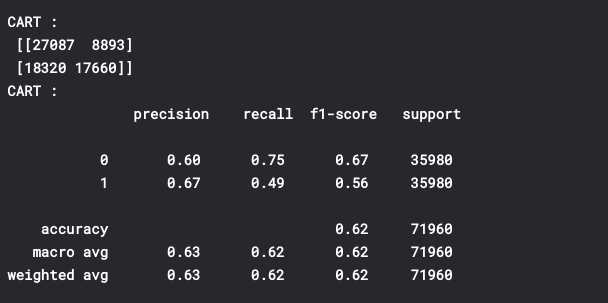
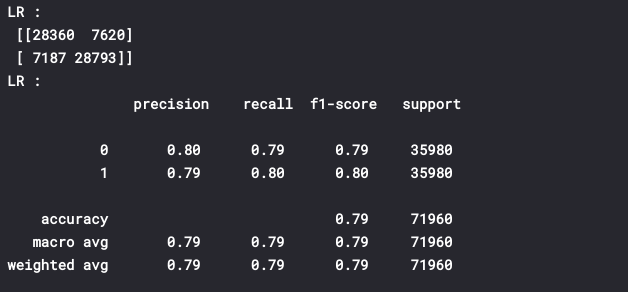


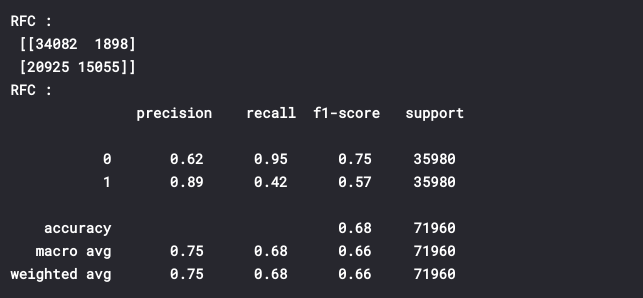
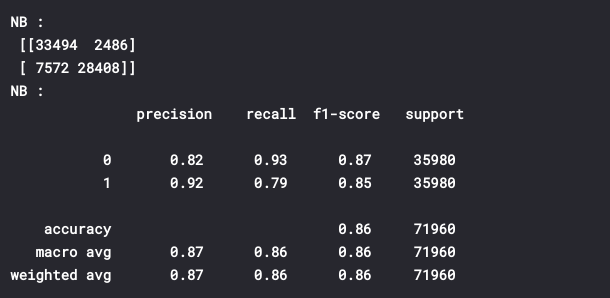




1. **AUC curve (Prob and Value) with Standardization and SMOTE oversampling**



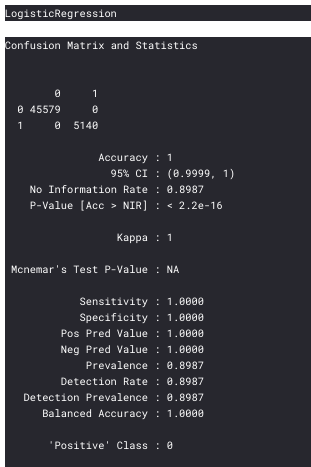
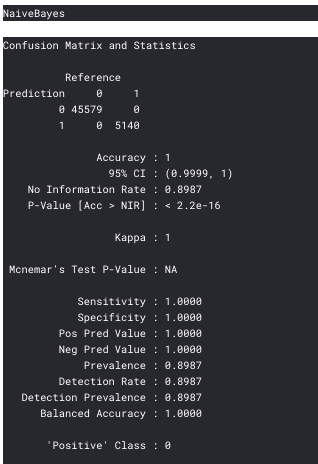
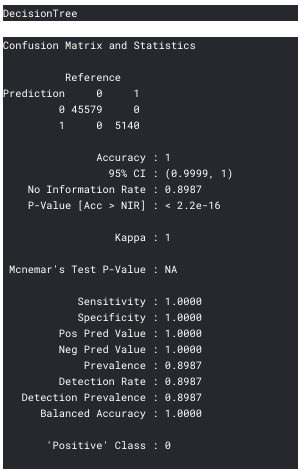
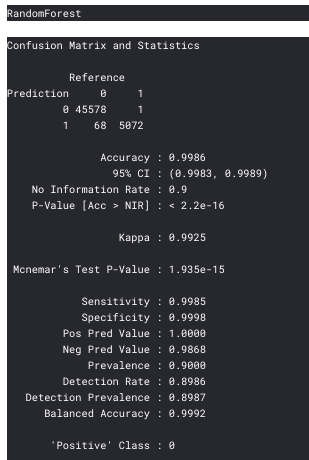




**Observations:**

* As can be see from reports and AUC score, our later model with SMOTE is doing way much better than former in all the models.
* LR and NB are the top most of these 4 models, in which NB being the highest AUC score probability : 0.919 and AUV value: 0.860.
* Also, NB have quite balanced f1-score among all which could lead on better confidence.

R:



Observation:

* None of the model correct result, they all seem to be guessing, since No Information Rate is high.

## 3.1.2 ROC\_AUC\_score

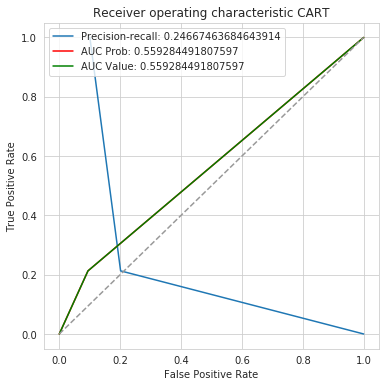
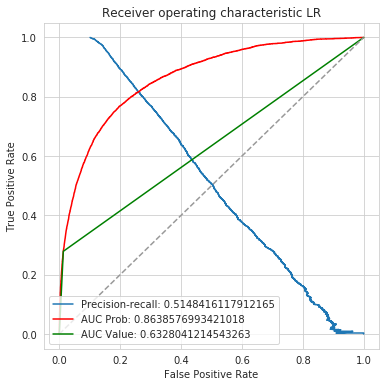
There are two diagnostic tools that help in the interpretation of probabilistic forecast for binary (two-class) classification predictive modelling problems are ROC Curves and Precision-Recall curves. ROC is a probability curve for different classes. ROC tells us how good the model is for distinguishing the given classes, in terms of the predicted probability.

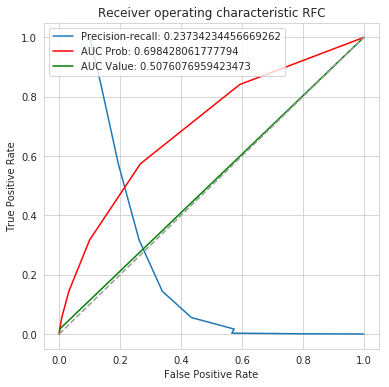
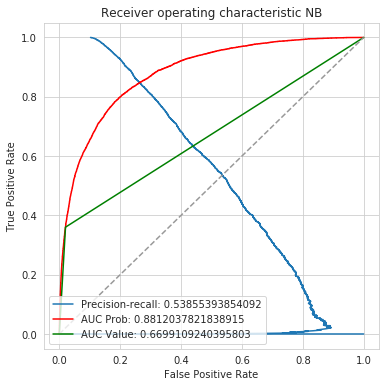
The area covered by the curve is the area between the orange line (ROC) and the axis. This area covered is AUC. The bigger the area covered, the better the machine learning models is at distinguishing the given classes. Ideal value for AUC is 1.

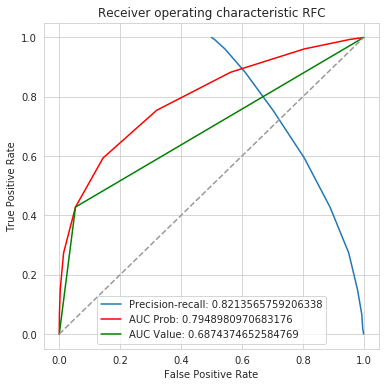
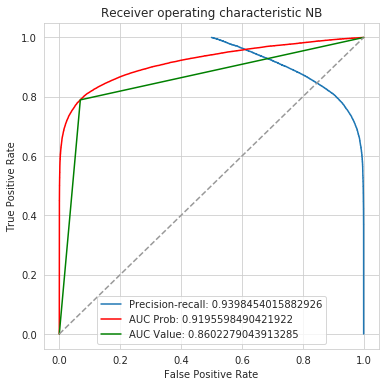
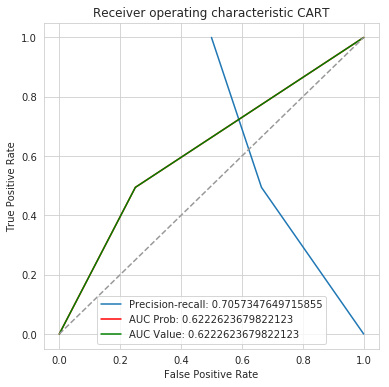
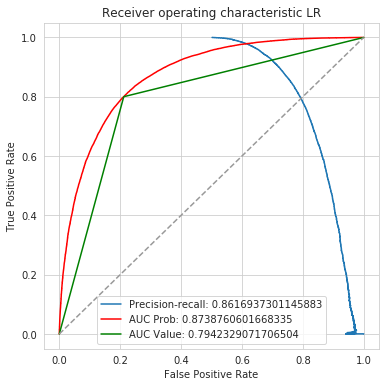
Precision is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class.

Recall is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as a sensitivity.

1. **AUC curve (Prob and Value) without Standard Scale**

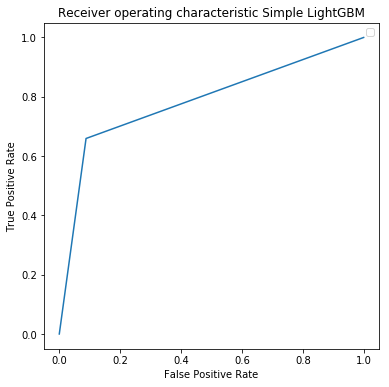




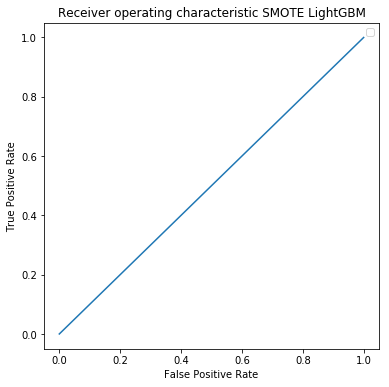
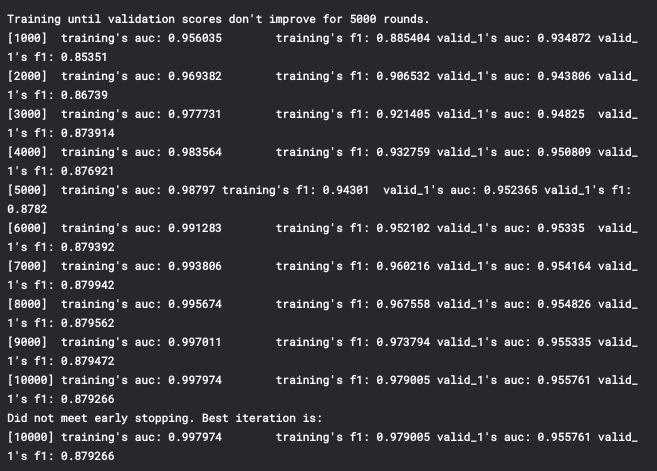
1. **AUC curve (Prob and Value) with Standardization and SMOTE oversampling**

## Python LightGBM:

* Simple LightGBM



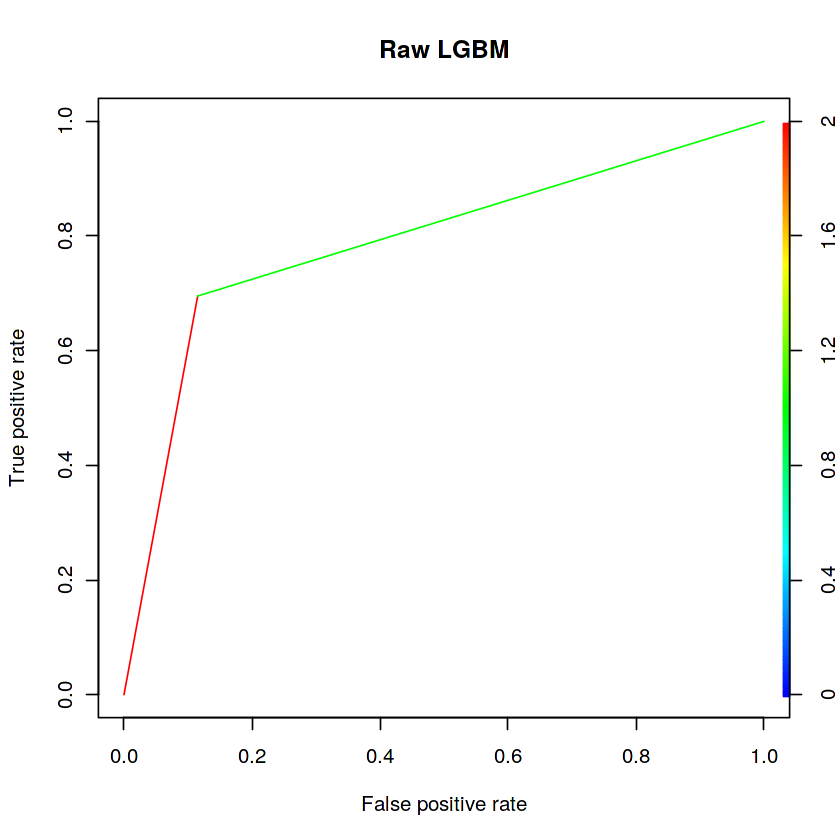
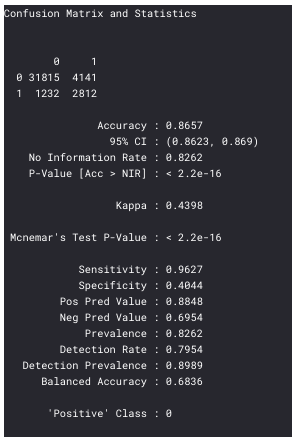
* SMOTE LightGBM:



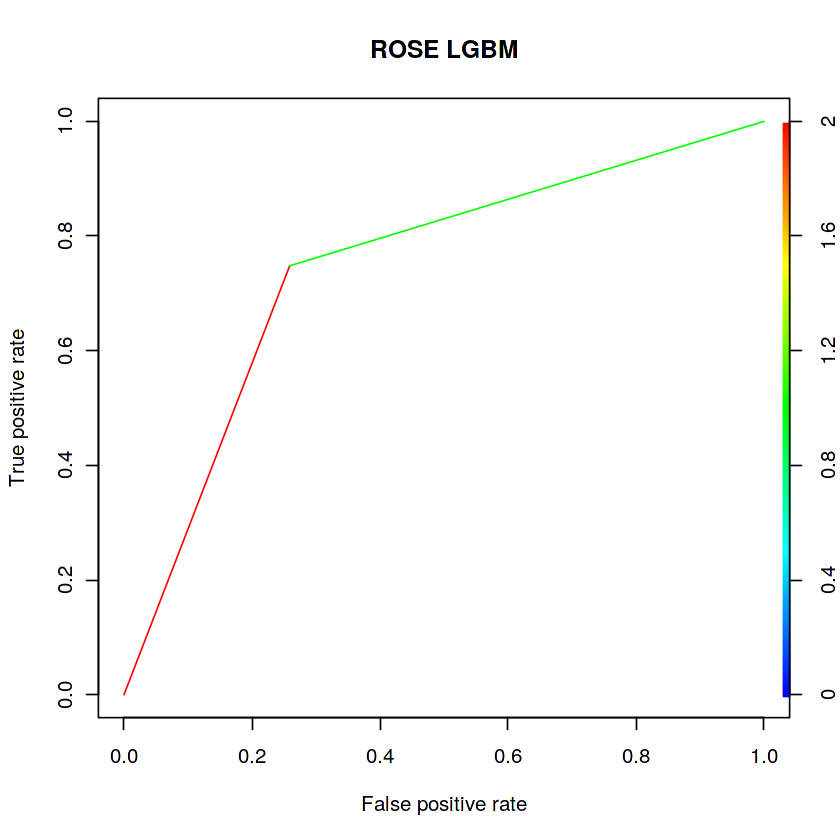
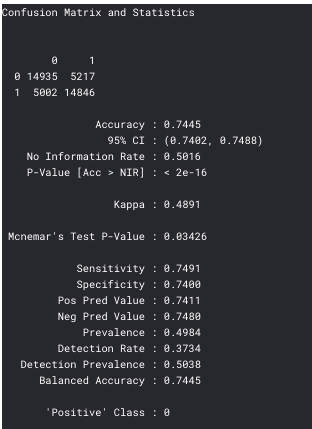


## R LightGBM:

* LightGBM with raw Data



* LightGBM with ROSE





## Observation:

Python:

* NB being the best AUC scorer among all initial 4 models.
* LightGBM with simple stratified is better than any other model with AUC: 0.892.

R:

* LightGBM with raw Data is scoring AUC: 0.889, which is best of all.

## 3.2 Model Selection ................................................................................................................ 38

After comparing scores of areas under the ROC curve of all the models for an imbalanced data. We could conclude that below points as the follow:

1. CART, RandomForestClassifier and LR models did not performed well on imbalanced data.

2.. NB performed quite well and can be considered for alternative, but it is being more time consuming.

2. We balance the imbalanced data using resampling techniques like SMOTE in python and ROSE in R.

3. LightGBM Raw data model is performed well on balanced data in R.

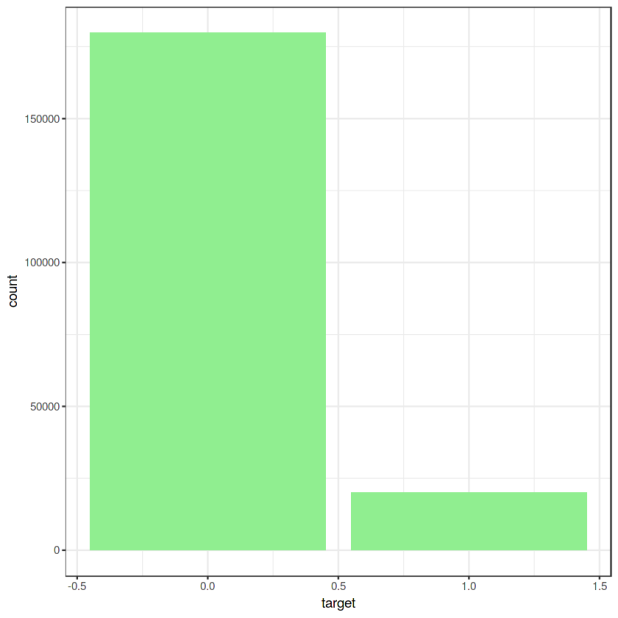
4. Simple Stratified LightGBM model performed well on imbalanced data in Python.

Finally, LightGBM is the best choice for identifying which customers will make a specific

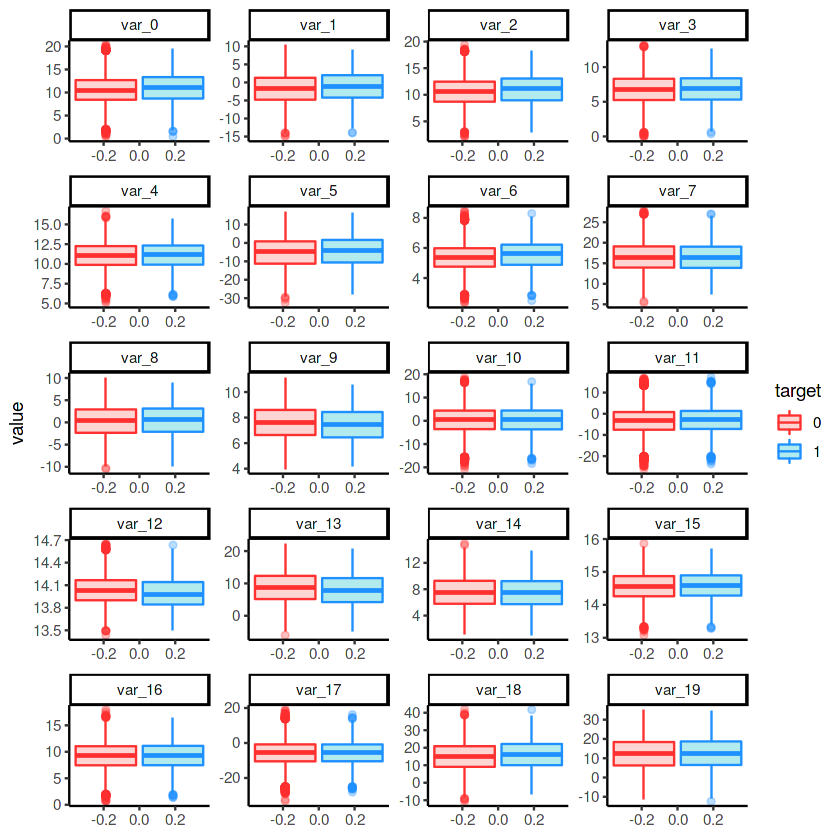
transaction in the future, irrespective of the amount of money transacted.

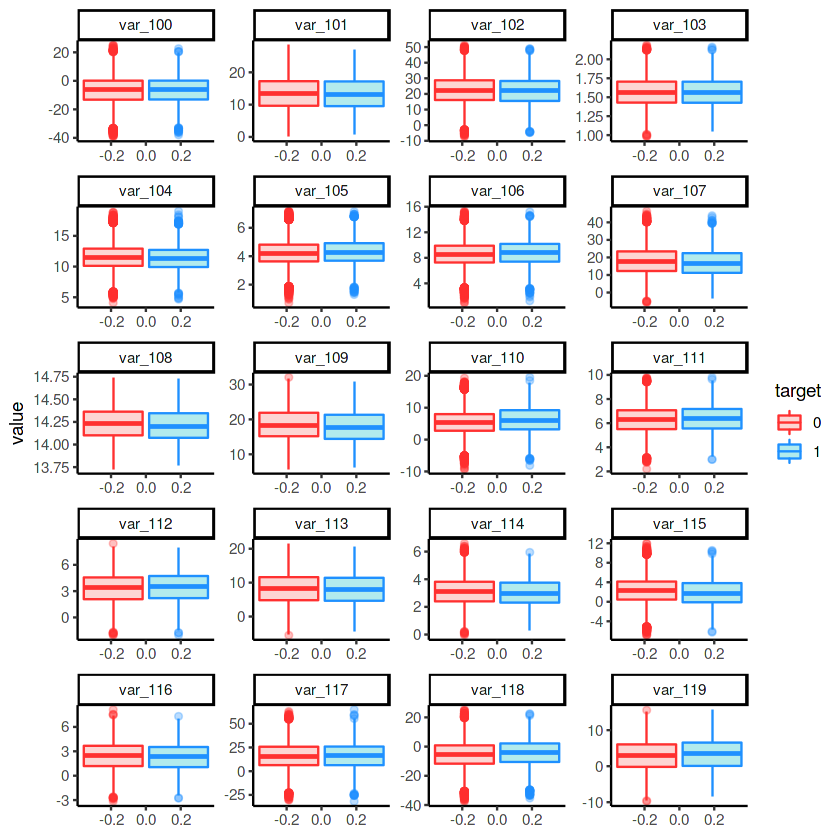
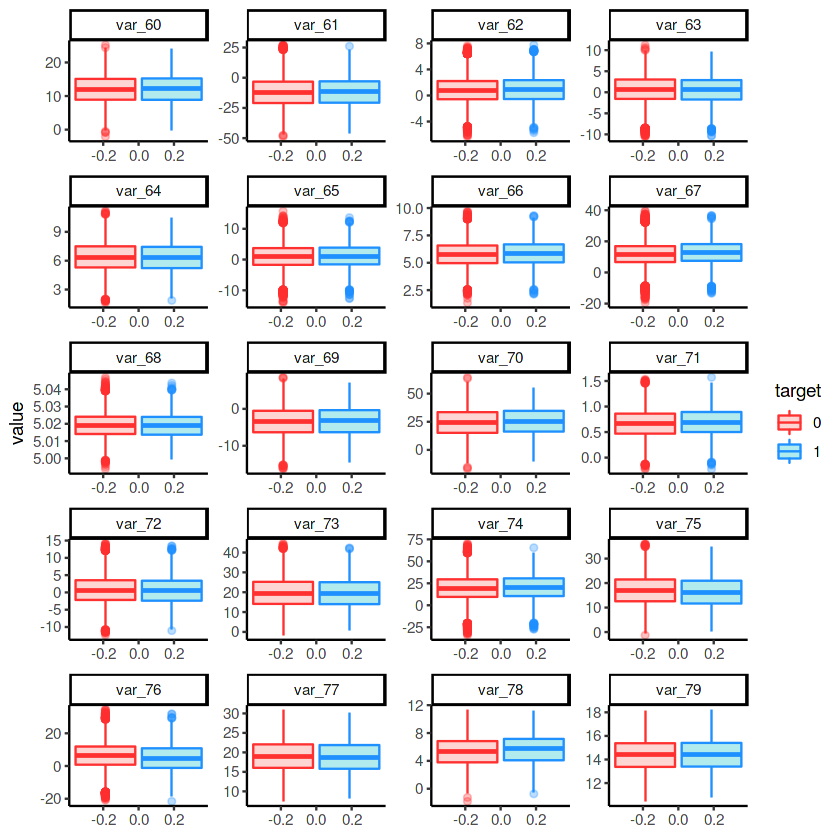
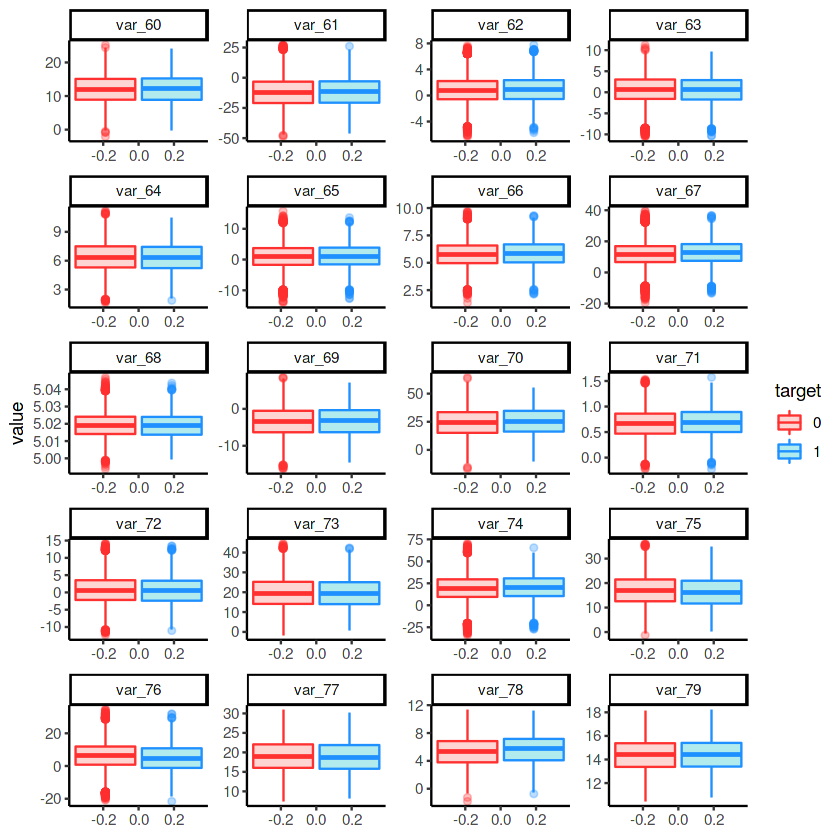
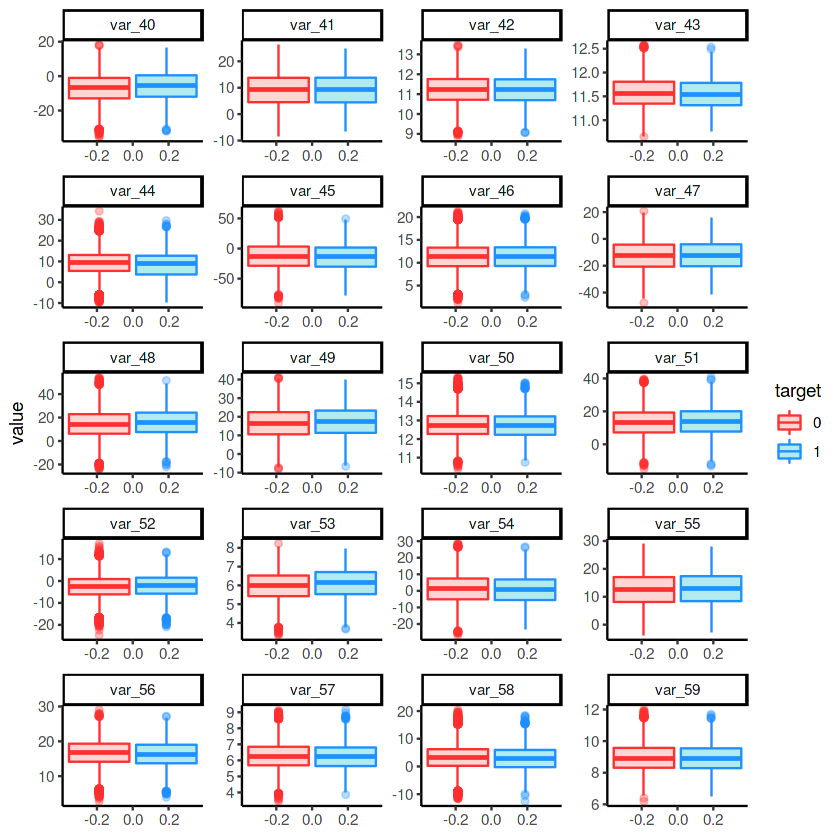
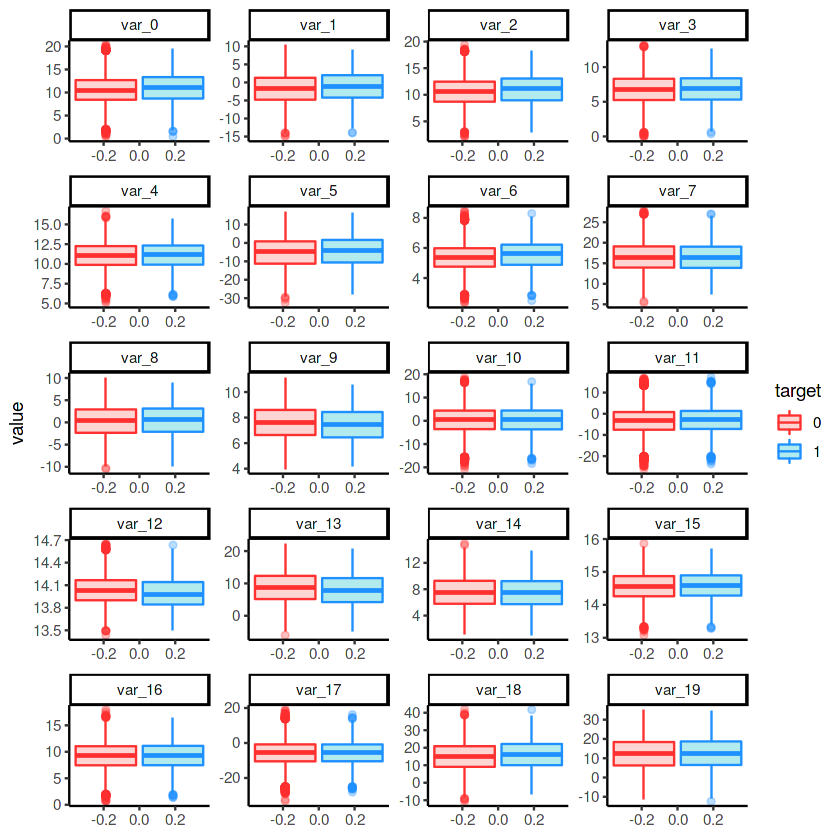
Appendix A - Extra Figures

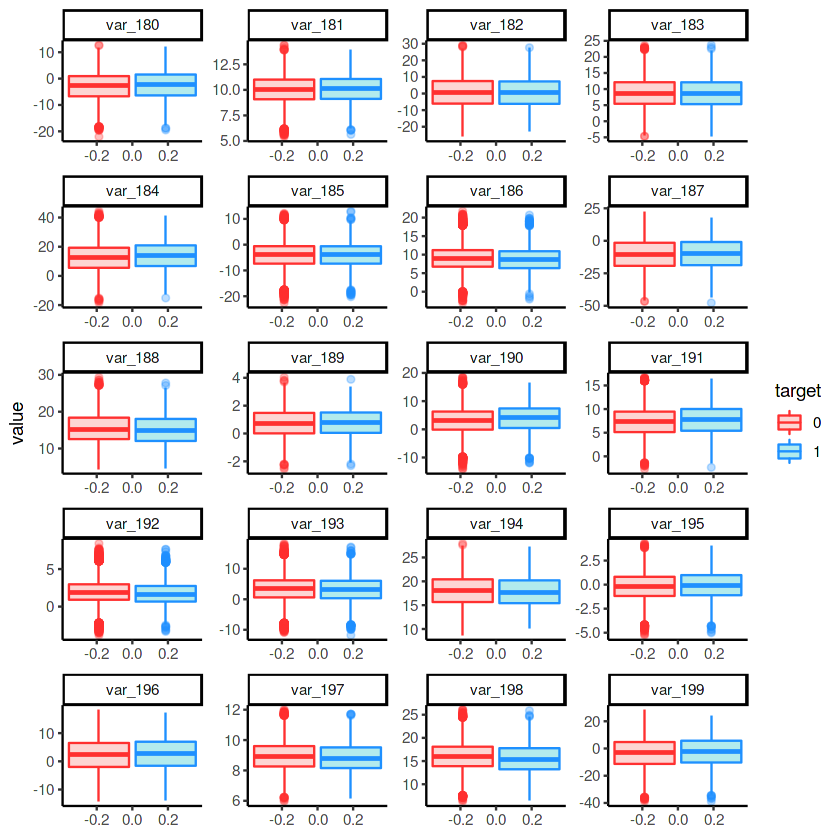
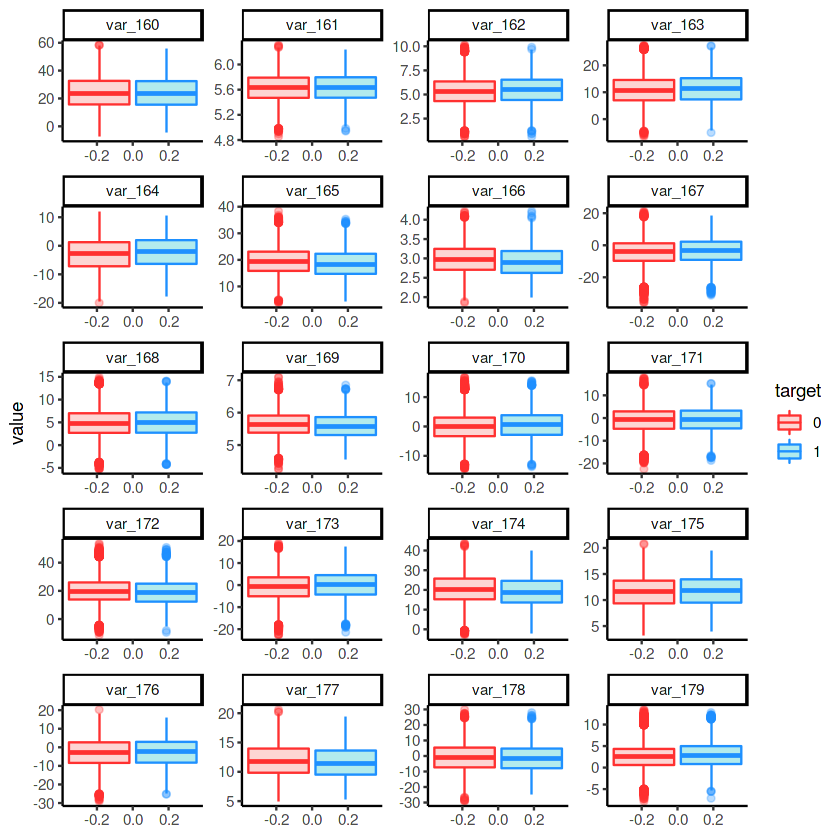
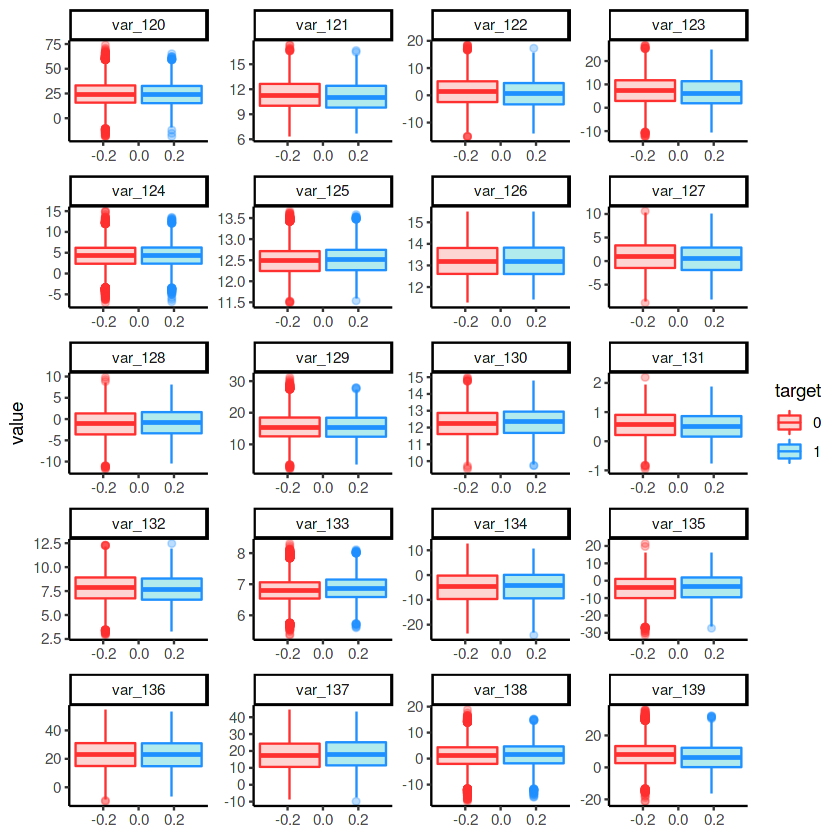
Target classes count



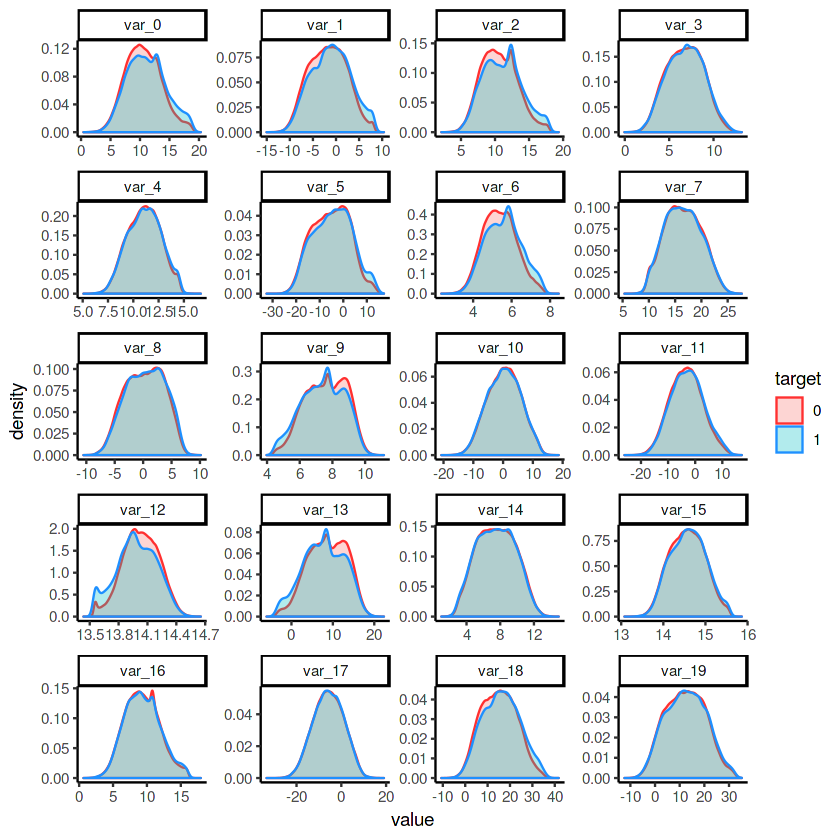
BoxPlot Train data

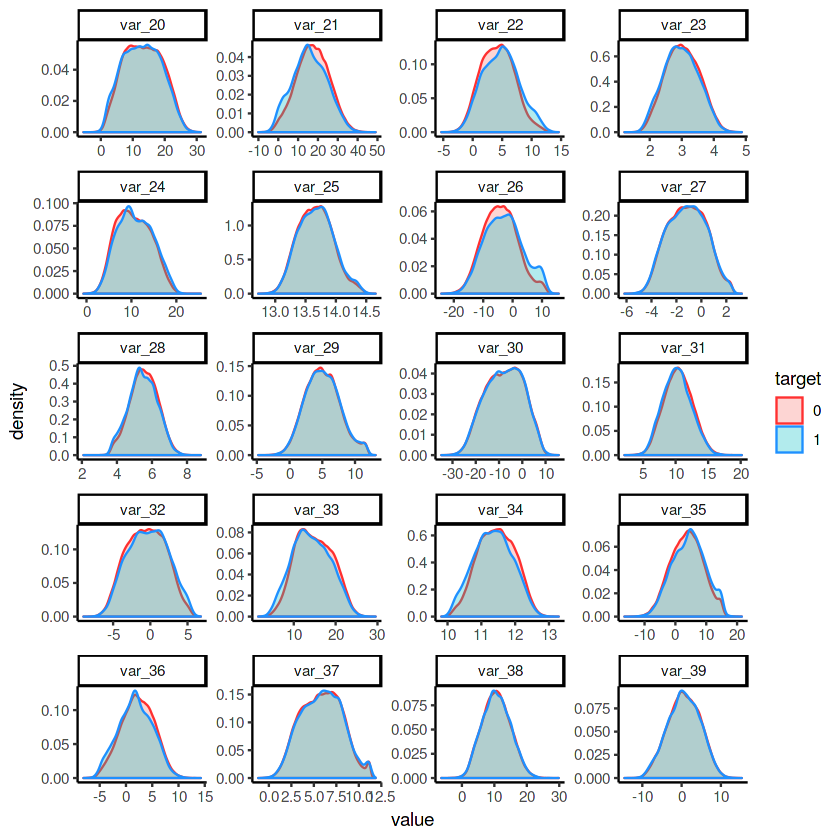


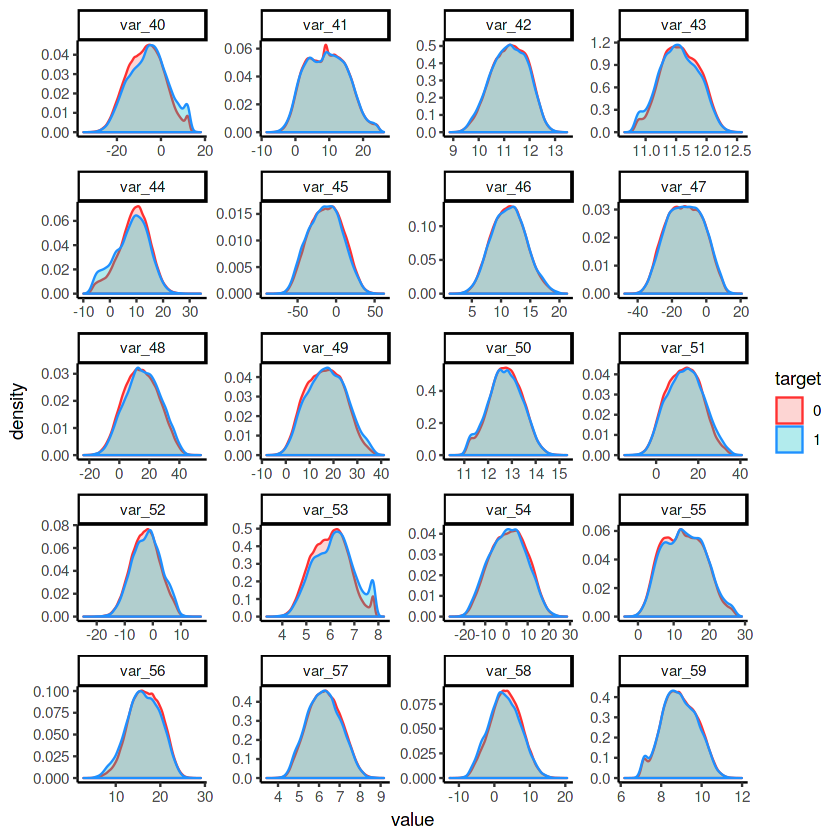


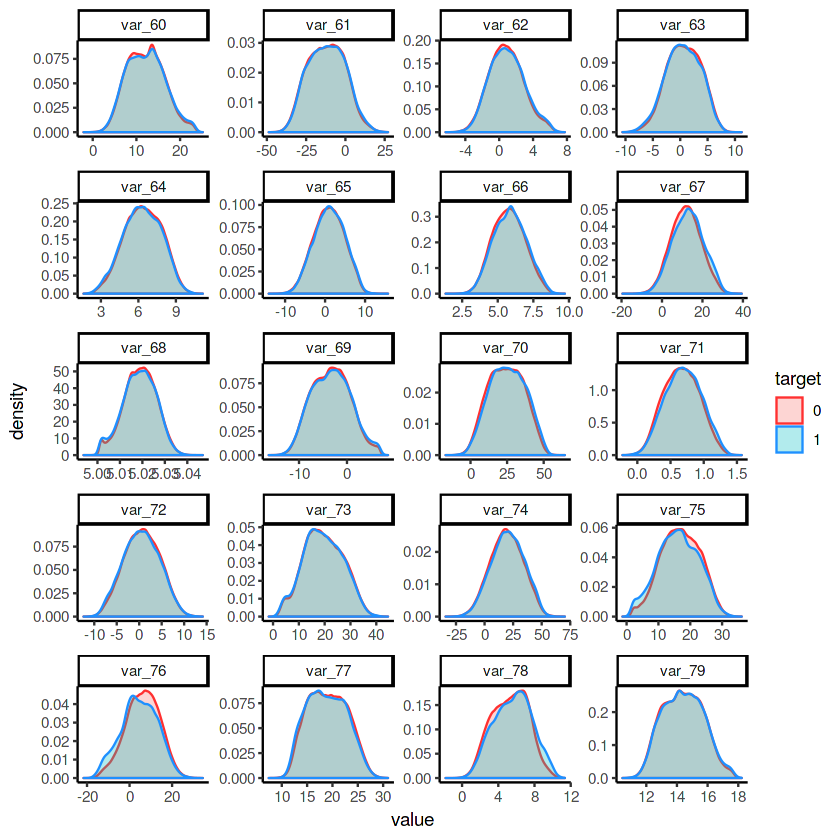


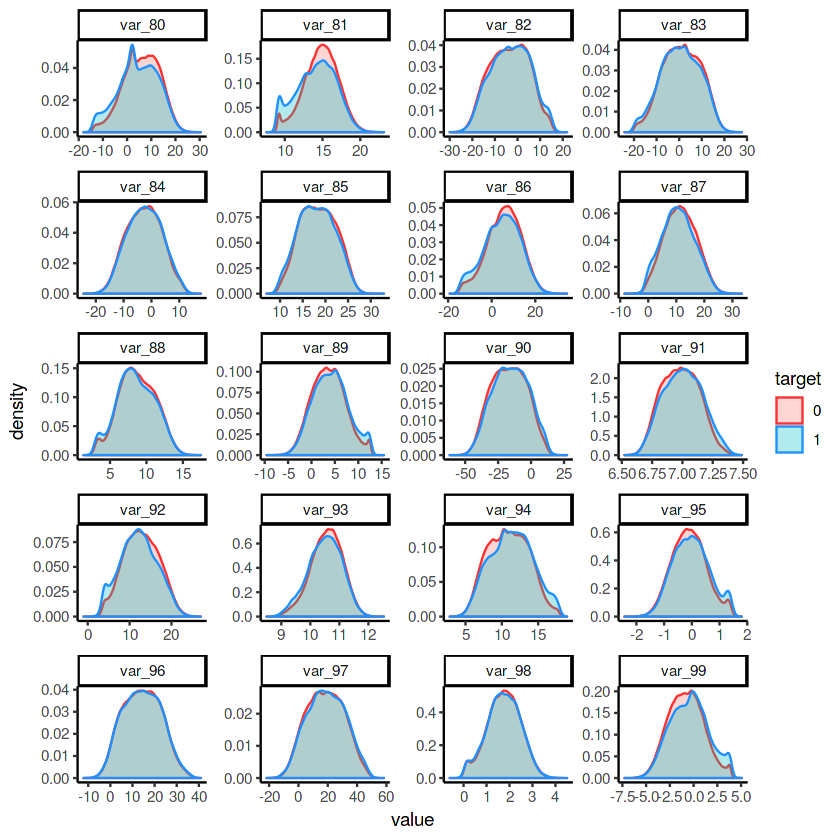
Distribution of train attributes

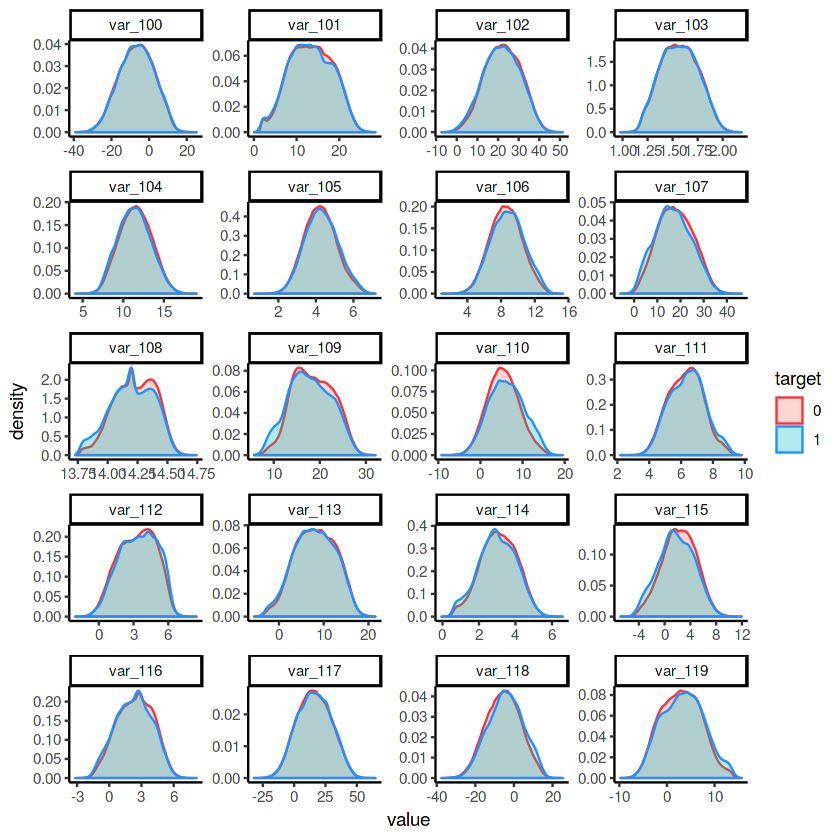


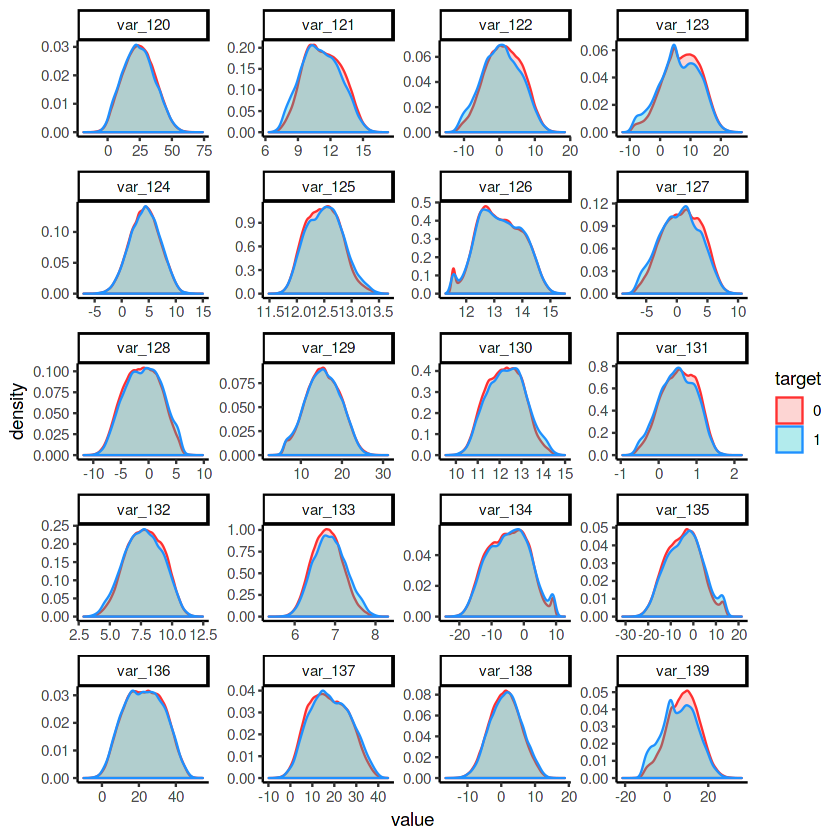


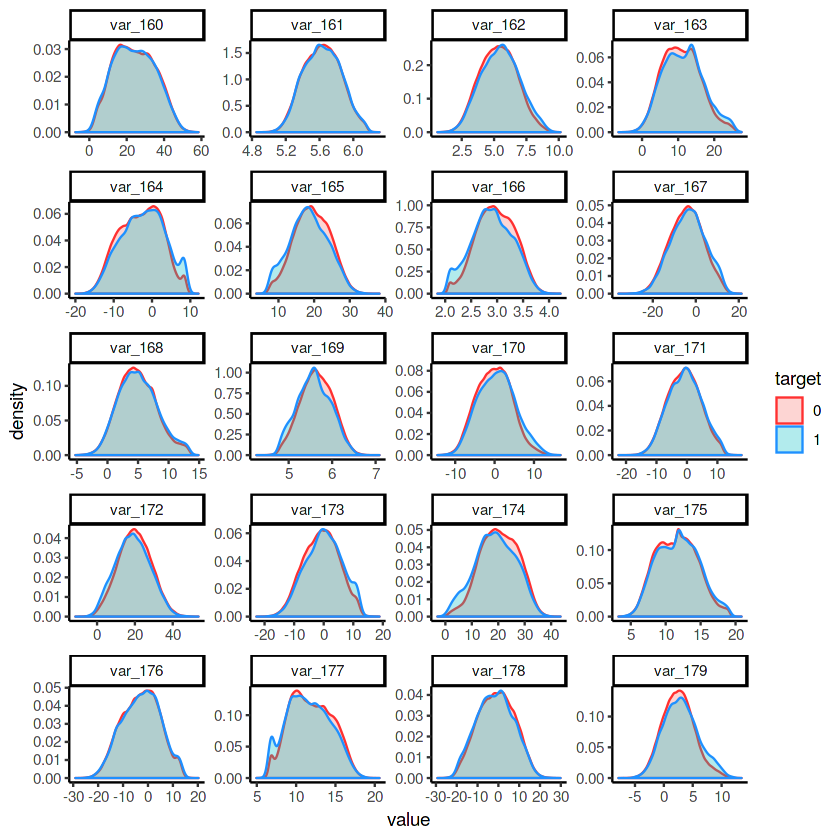


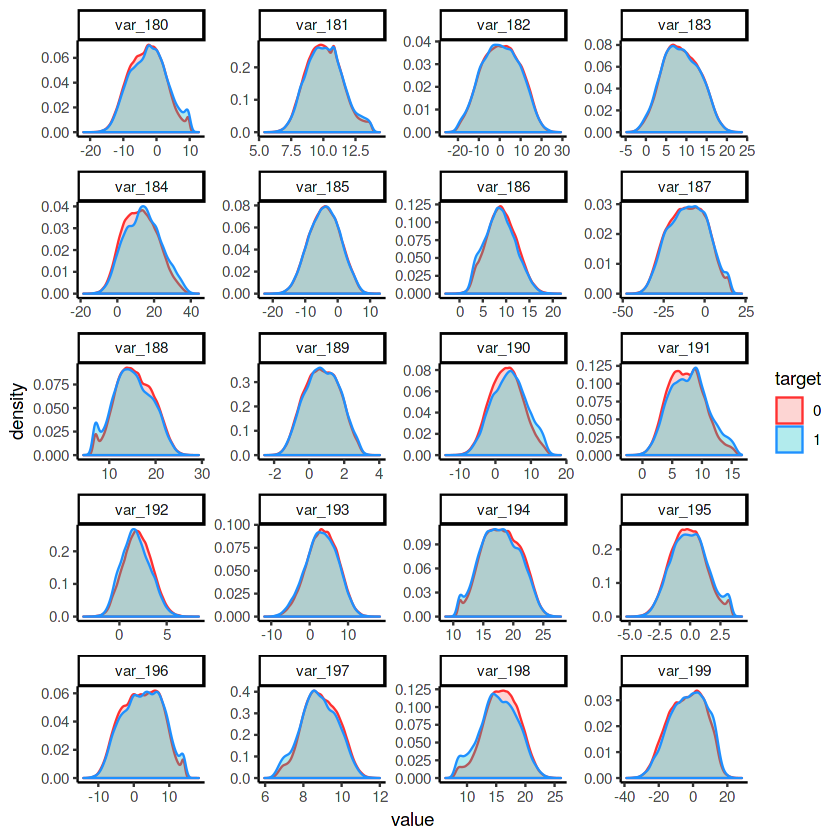


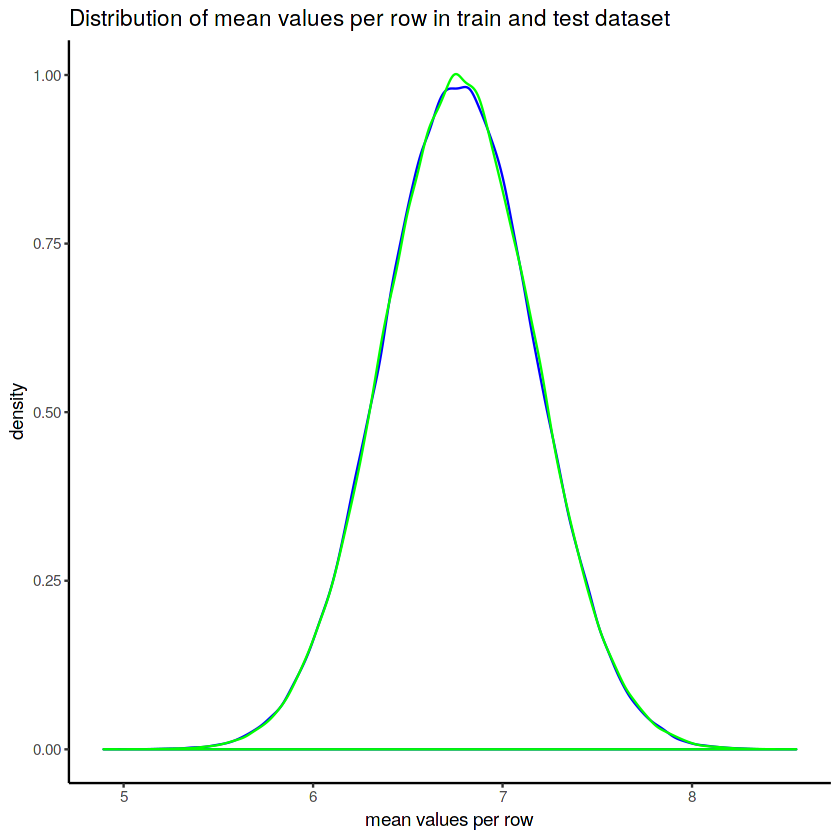


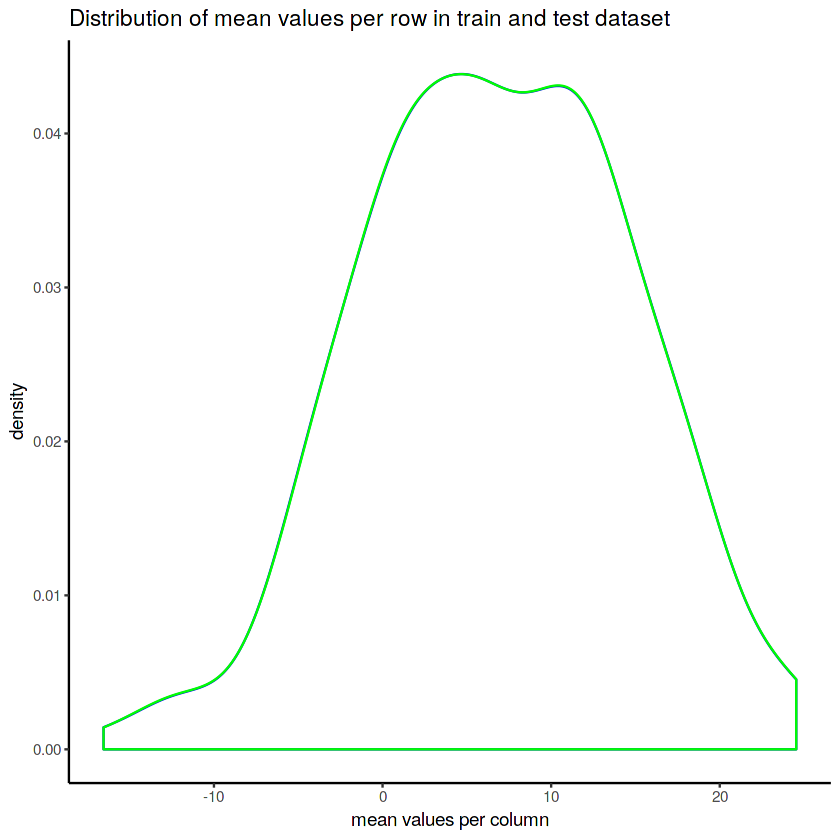


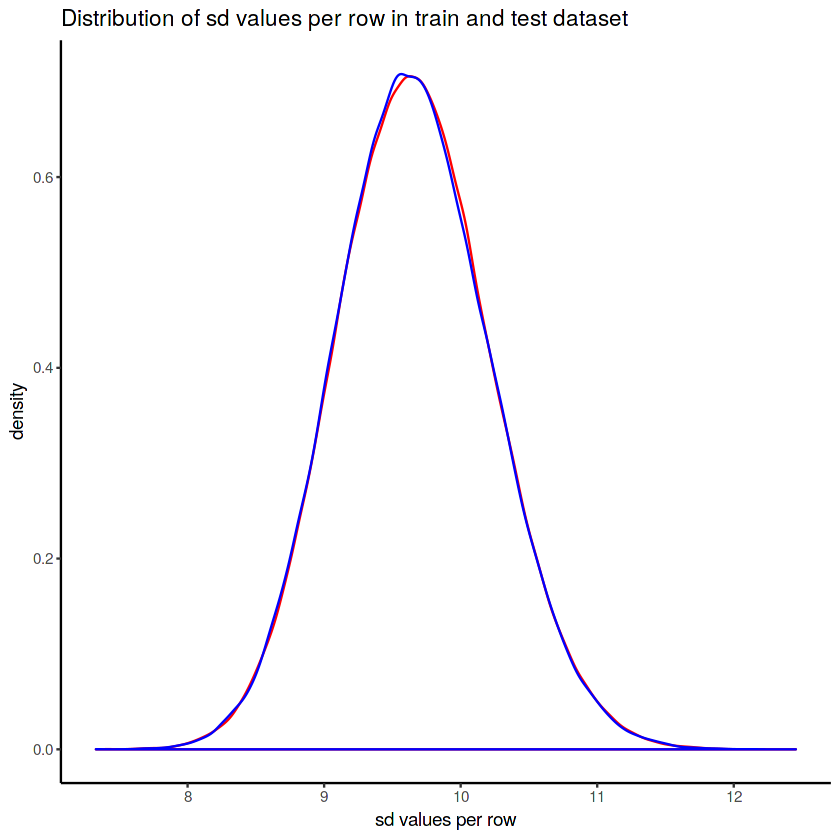


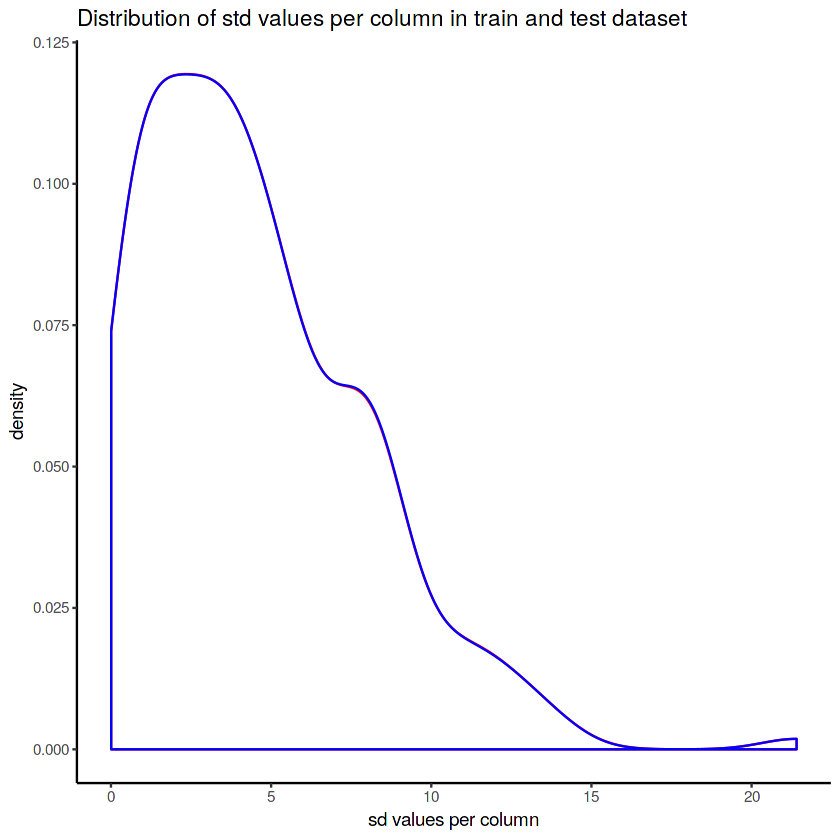


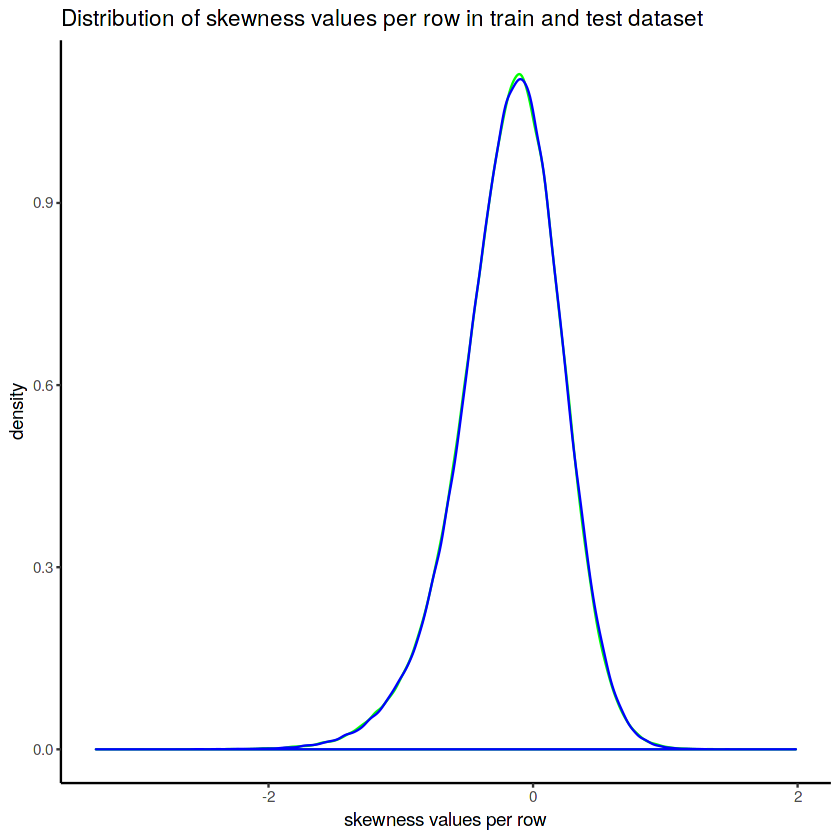


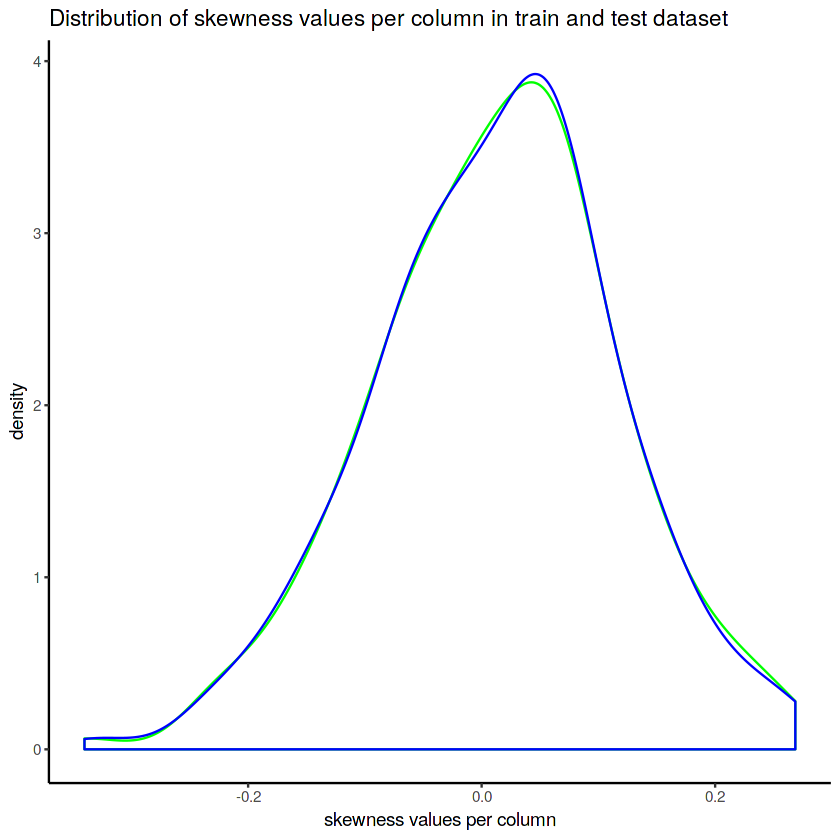


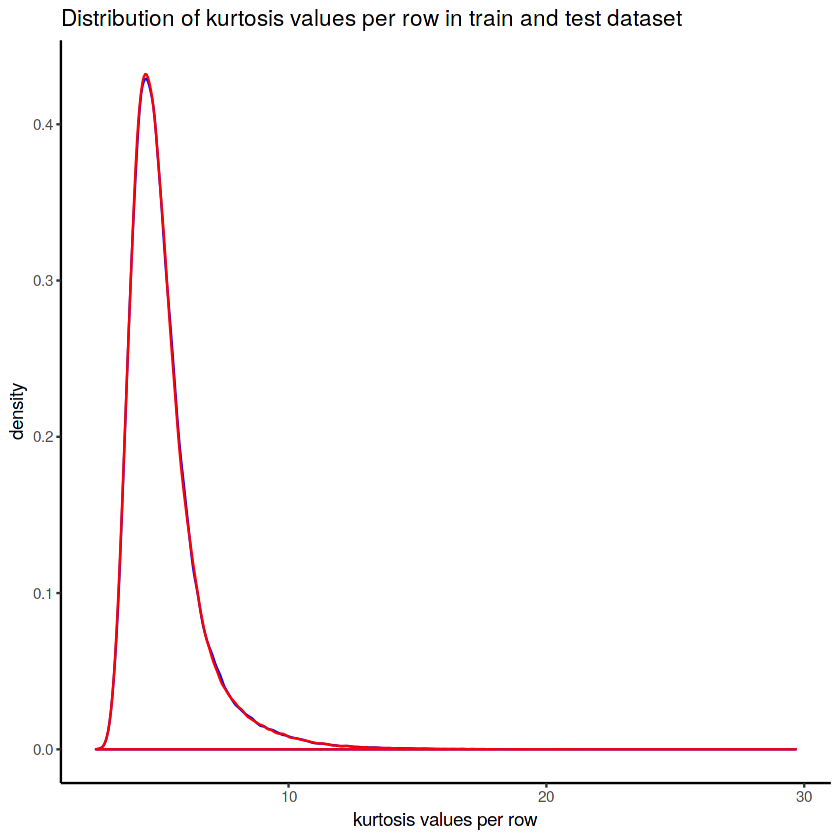


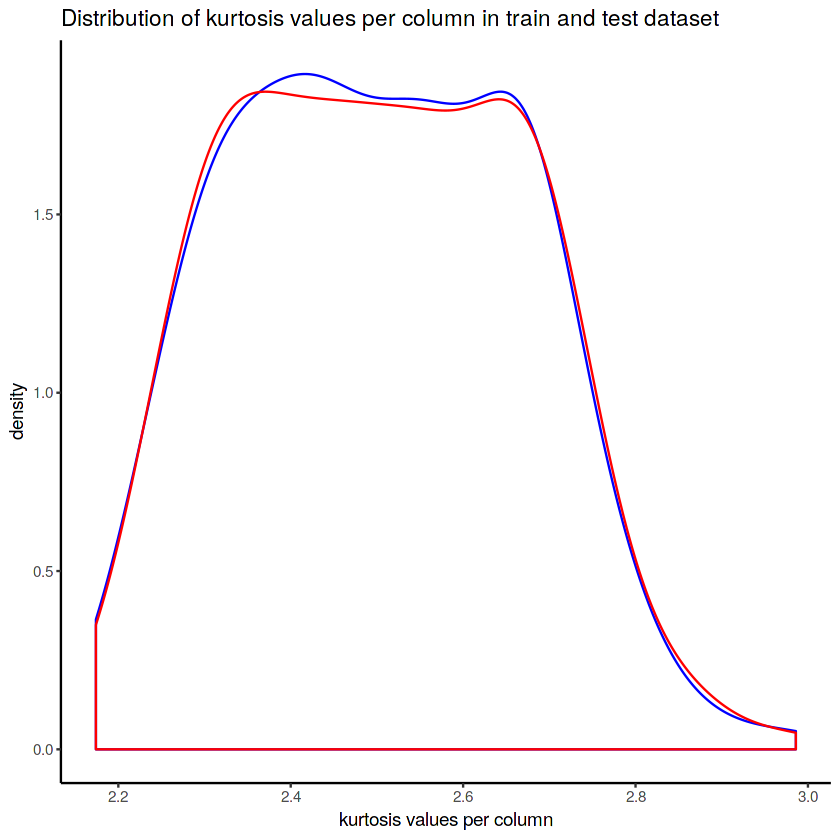












# Appendix B – Complete Python and R Code

## Python Code

import os

import warnings

import numpy as np # linear algebra # data manipulation

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# visualization

import seaborn as sns

import matplotlib.pyplot as plt

import gc

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.model\_selection import KFold, cross\_val\_score, StratifiedKFold

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import make\_scorer, roc\_auc\_score, auc, precision\_score, recall\_score, classification\_report, roc\_curve, accuracy\_score, f1\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix, f1\_score, precision\_recall\_curve

from sklearn.decomposition import PCA

import lightgbm as lgb

from imblearn.over\_sampling import SMOTE

warnings.filterwarnings("ignore")

print(os.listdir("/input"))

# loading the data files

train = pd.read\_csv('/input/train.csv', sep=',')

test = pd.read\_csv('/input/test.csv', sep=',')

# taking sneak peak to datasets

print(f'Dimension of our Train data {train.shape} \n Data feature informations')

print(train.info())

print(f'Dimension of our Test data {test.shape} \n Data feature informations')

print(test.info())

print(train.head(), test.head())

print(f'Train columns: {train.columns}\nTest columns: {test.columns}')

### EDA for understanding datasets and getting clues for feature selections.

# Datatypes in dataset

print('Train target column datatype:',train.target.dtype)

print('Train var\_0 column datatype:',train.var\_0.dtype)

print('Train Describe:\n',train.describe(),'\nTest Describe:\n', test.describe())

print('Different values in target:\n',train.target.unique())

# Missing value analysis

print('Train missing values:',train.isnull().sum().sum())

print('Test missing values:',test.isnull().sum().sum())

# Looking Variance

print('Train Variance:\n',train.var(),'\nTest Variance:\n', test.var())

# Looking Skewness

print('Train skewness:\n',train.skew(),'\nTest skewness:\n', test.skew())

# Digging target variable

target = train['target']

print('Different values in target:\n',target.value\_counts())

print('')

print("There are {}% target values with 1".format(100 \*target.value\_counts()[1]/(target.value\_counts()[1] + target.value\_counts()[0])))

sns.countplot(train['target'], palette='Set1')

plt.figure(figsize=(10,6))

train['target'].value\_counts().plot.pie(autopct='%1.1f%%', explode=([0,0.1]))

plt.show()

# Boxplot Analysis

# Plot features.

train.iloc[:, 2:50].plot(kind='box', figsize=[16,8])

train.iloc[:, 50:101].plot(kind='box', figsize=[16,8])

train.iloc[:, 101:151].plot(kind='box', figsize=[16,8])

train.iloc[:, 151:].plot(kind='box', figsize=[16,8])

# Distribution plot Analysis

# Function for quick plot of distribution

def plot\_feature\_distribution(df1, df2, label1, label2, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(10,10,figsize=(18,22))

for feature in features:

i += 1

plt.subplot(10,10,i)

sns.distplot(df1[feature], hist=False,label=label1)

sns.distplot(df2[feature], hist=False,label=label2)

plt.xlabel(feature, fontsize=9)

locs, labels = plt.xticks()

plt.tick\_params(axis='x', which='major', labelsize=6)

plt.tick\_params(axis='y', which='major', labelsize=6)

plt.show()

t0 = train.loc[train['target'] == 0]

t1 = train.loc[train['target'] == 1]

# First 100 features dustribution

features = train.columns.values[2:102]

plot\_feature\_distribution(t0, t1, '0', '1', features)

# Rest 100 features dustribution

features = train.columns.values[102:]

plot\_feature\_distribution(t0, t1, '0', '1', features)

# 1. STD distribution of target variable

t0 = train.loc[train['target'] == 0]

t1 = train.loc[train['target'] == 1]

plt.figure(figsize=(16,6))

plt.title("Distribution of std values per row in the train set")

sns.distplot(t0[features].std(axis=1),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].std(axis=1),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of std values per column in the train set")

sns.distplot(t0[features].std(axis=0),color="green", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].std(axis=0),color="red", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# 2. Mean distribution of target variable

plt.figure(figsize=(16,6))

plt.title("Distribution of mean values per row in the train and test set")

sns.distplot(t0[features].mean(axis=1),color="green", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].mean(axis=1),color="red", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of mean values per column in the train and test set")

sns.distplot(t0[features].mean(axis=0),color="green", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].mean(axis=0),color="red", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# 3. Min distribution

plt.figure(figsize=(16,6))

plt.title("Distribution of min values per row in the train set")

sns.distplot(t0[features].min(axis=1),color="orange", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].min(axis=1),color="darkblue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of min values per column in the train set")

sns.distplot(t0[features].min(axis=0),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].min(axis=0),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# 4. Max distribution

plt.figure(figsize=(16,6))

plt.title("Distribution of max values per row in the train set")

sns.distplot(t0[features].max(axis=1),color="gold", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].max(axis=1),color="darkblue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of max values per column in the train set")

sns.distplot(t0[features].max(axis=0),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].max(axis=0),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# 5. Skew distribution

plt.figure(figsize=(16,6))

plt.title("Distribution of skew values per row in the train set")

sns.distplot(t0[features].skew(axis=1),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].skew(axis=1),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of skew values per column in the train set")

sns.distplot(t0[features].skew(axis=0),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].skew(axis=0),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# 6. Kurtosis distribution

plt.figure(figsize=(16,6))

plt.title("Distribution of kurtosis values per row in the train set")

sns.distplot(t0[features].kurtosis(axis=1),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].kurtosis(axis=1),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of kurtosis values per column in the train set")

sns.distplot(t0[features].kurtosis(axis=0),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].kurtosis(axis=0),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# 7. Median distribution

plt.figure(figsize=(16,6))

plt.title("Distribution of median values per row in the train set")

sns.distplot(t0[features].median(axis=1),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].median(axis=1),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of median values per column in the train set")

sns.distplot(t0[features].median(axis=0),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].median(axis=0),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# 8. Sum distribution

plt.figure(figsize=(16,6))

plt.title("Distribution of sum values per row in the train set")

sns.distplot(t0[features].sum(axis=1),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].sum(axis=1),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

plt.figure(figsize=(16,6))

plt.title("Distribution of sum values per column in the train set")

sns.distplot(t0[features].sum(axis=0),color="red", kde=True,bins=150, label='target 0')

sns.distplot(t1[features].sum(axis=0),color="blue", kde=True,bins=150, label='target 1')

plt.legend()

plt.show()

# Correlation Analysis

data\_corr=train.drop(['target','ID\_code'], axis=1).corr()

print('Maximum corr within all variables correlations :', np.sort(train.drop(['target','ID\_code'], axis=1).corr())[:,-2:-1].max())

# Correlation Heatmap

plt.figure(figsize=(20,20))

sns.heatmap(data\_corr, square=True)

plt.title('Feature Correlation')

plt.show()

# Data Preprocessiing

# Remove outliers

train\_x = train.iloc[:, 1:]

IQR = train\_x.quantile(.75) - train\_x.quantile(.25)

print("Train.shape:",train.shape)

df\_in = train[~((train\_x < (train\_x.quantile(.25) - 1.5 \* IQR)) |(train\_x > (train\_x.quantile(.75) + 1.5 \* IQR))).any(axis=1)]

df\_out = train[((train\_x < (train\_x.quantile(.25) - 1.5 \* IQR)) |(train\_x > (train\_x.quantile(.75) + 1.5 \* IQR))).any(axis=1)]

print("df\_in.shape:",df\_in.shape)

print("df\_out.shape:",df\_out.shape)

print("df\_in.target:\n", df\_in['target'].value\_counts())

print("df\_out.target:\n", df\_out['target'].value\_counts())

# PCA Analysis

# feature extraction

pca = PCA().fit(train.drop(['target','ID\_code'], axis=1))

plt.figure(figsize=(10,6))

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

plt.plot(pca.explained\_variance\_ratio\_)

plt.title('Scree Plot')

plt.xlabel('Principal Component')

plt.ylabel('Eigenvalue')

leg = plt.legend(['Eigenvalues from PCA'], loc='best', borderpad=0.3,shadow=False,markerscale=0.4)

plt.grid(True)

plt.show()

# Using Stratified sampling

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train.drop(['target', 'ID\_code'], axis=1), train['target'], test\_size=0.3, random\_state=147, stratify=train.target)

print('Shape:',X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

# RFC imporant features

parameters = {'min\_samples\_leaf': [10,25]}

forest = RandomForestClassifier(max\_depth=15, n\_estimators=15)

grid\_rfc = GridSearchCV(forest, parameters, cv=3, n\_jobs=1, verbose=3, scoring=make\_scorer(roc\_auc\_score))

grid\_rfc.fit(X\_train, y\_train)

imp = grid\_rfc.best\_estimator\_.feature\_importances\_

idx = np.argsort(imp)[::-1][-26:]

remove\_features\_RFC = train.columns[2:]

#train.drop(remove\_features\_RFC[idx],axis=1, inplace=True)

#test.drop(remove\_features\_RFC[idx],axis=1, inplace=True)

remove\_col = remove\_features\_RFC[idx]

print('Removing features:', remove\_col)

print('Train shape:',train.shape)

remove\_col = ['var\_187', 'var\_113', 'var\_7', 'var\_126', 'var\_189', 'var\_62',

'var\_117', 'var\_45', 'var\_182', 'var\_96', 'var\_199', 'var\_19', 'var\_68',

'var\_77', 'var\_3', 'var\_25', 'var\_14', 'var\_41', 'var\_73', 'var\_30',

'var\_64', 'var\_185', 'var\_29', 'var\_129', 'var\_171', 'var\_140']

trainFE = train.drop(remove\_col,axis=1)

testFE = test.drop(remove\_col,axis=1)

print('Removing features:', remove\_col)

print('Columns left in Train :',trainFE.shape)

print('Columns left in Test :',testFE.shape)

# Creating features

print('Featuring Engineering raw data: Adding aggregates :')

idx = features = train.columns[2:]

for df in [test, train]:

df['sum'] = df[idx].sum(axis=1)

df['min'] = df[idx].min(axis=1)

df['max'] = df[idx].max(axis=1)

df['mean'] = df[idx].mean(axis=1)

df['std'] = df[idx].std(axis=1)

df['skew'] = df[idx].skew(axis=1)

df['kurt'] = df[idx].kurtosis(axis=1)

df['med'] = df[idx].median(axis=1)

print('Train:', train.shape)

print('Test:' , test.shape)

# Distribution plot

def plot\_new\_feature\_distribution(df1, df2, label1, label2, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(2,4,figsize=(16,6))

for feature in features:

i += 1

plt.subplot(2,4,i)

sns.kdeplot(df1[feature], bw=0.5,label=label1)

sns.kdeplot(df2[feature], bw=0.5,label=label2)

plt.xlabel(feature, fontsize=11)

locs, labels = plt.xticks()

plt.tick\_params(axis='x', which='major', labelsize=8)

plt.tick\_params(axis='y', which='major', labelsize=8)

plt.show();

t0 = train.loc[train['target'] == 0]

t1 = train.loc[train['target'] == 1]

features = train.columns.values[202:]

plot\_new\_feature\_distribution(t0, t1, 'target: 0', 'target: 1', features)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(train.drop(['target','ID\_code'], axis=1), train.target, test\_size=0.3, random\_state=147, stratify=train.target)

print('Shape:',X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

# Modeling

# Spot-Check Algorithms

models = []

models.append(( 'LR' , LogisticRegression(solver='liblinear')))

models.append(( 'CART' , DecisionTreeClassifier()))

models.append(( 'NB' , GaussianNB()))

models.append(('RFC', RandomForestClassifier()))

def cv\_auc\_score(models,scoring, num\_folds=3):

seed = 147

results = []

names = []

print('-> 3-Fold cross-validation ',scoring.\_\_name\_\_,'score for the training data for 4 classifiers.')

for name, model in models:

kfold = KFold( n\_splits=num\_folds, random\_state=seed)

cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold,verbose=3 ,scoring=make\_scorer(scoring))

results.append(cv\_results)

names.append(name)

print("Algo: ", name,'::',np.mean(cv\_results))

# Compare Algorithms

fig = plt.figure()

fig.suptitle( 'Algorithm Comparison: {}'.format(scoring.\_\_name\_\_ ))

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()

# AUC score

num\_folds = 3

scoring=roc\_auc\_score

print("Scores without StandardScale")

cv\_auc\_score(models, scoring=scoring, num\_folds=num\_folds)

# Accuracy score

scoring = accuracy\_score

print("Scores without StandardScale")

cv\_auc\_score(models, scoring=scoring, num\_folds=num\_folds)

def aur\_prob\_value\_precision\_recall\_curve(models, X\_train, X\_test,y\_train, y\_test):

for name, model in models:

model.fit(X\_train, y\_train)

y\_pred = model.predict\_proba(X\_test)

y\_pred2 = model.predict(X\_test)

print(name,' AUC prob: ',roc\_auc\_score(y\_test, y\_pred[:,1]))

print(name,' AUC value: ',roc\_auc\_score(y\_test, y\_pred2))

print(name,' f1 score: ',f1\_score(y\_test, y\_pred2))

precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_pred[:,1])

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred[:,1])

fpr2, tpr2, thresholds2 = roc\_curve(y\_test, y\_pred2)

fig, ax = plt.subplots(1,1, figsize=(6,6))

ax.plot(precision, recall)

ax.plot(fpr, tpr, color='red')

ax.plot(fpr2, tpr2, color='green')

ax.plot([0, 1], [0, 1], '--', color=(0.6, 0.6, 0.6))

ax.legend([f'Precision-recall: {auc(recall, precision)}',f'AUC Prob: {auc(fpr, tpr)}',f'AUC Value: {auc(fpr2, tpr2)}'])

# ax.legned()

ax.set\_xlabel('False Positive Rate')

ax.set\_ylabel('True Positive Rate')

ax.set\_title('Receiver operating characteristic {}'.format(name))

def classification\_report\_models(models,X\_train, X\_test, y\_train, y\_test ):

for name, model in models:

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print(name, ':\n', confusion\_matrix(y\_test, y\_pred))

print(name,':\n',classification\_report(y\_test, y\_pred))

print("AUC curve (Prob and Value) without StandardScale")

aur\_prob\_value\_precision\_recall\_curve(models=models, X\_train = X\_train, X\_test= X\_test, y\_train=y\_train, y\_test=y\_test)

classification\_report\_models(models=models, X\_train = X\_train, X\_test= X\_test, y\_train=y\_train, y\_test=y\_test)

# Standarizaion

tr\_X = train.drop([ 'ID\_code'], axis=1)

test\_X = test.drop(['ID\_code'], axis=1)

for col in tr\_X.drop(['target'], axis=1).columns:

tr\_X[col] = ((tr\_X[col] - tr\_X[col].mean()) / tr\_X[col].std()).astype('float32')

for col in test\_X.columns:

test\_X[col] = ((test\_X[col] - test\_X[col].mean()) / test\_X[col].std()).astype('float32')

#Training data

X=tr\_X.drop(['target'],axis=1)

Y=train['target']

#StratifiedKFold cross validator

cv=StratifiedKFold(n\_splits=5,random\_state=147,shuffle=True)

for train\_index,valid\_index in cv.split(X,Y):

X\_train1, X\_valid=X.iloc[train\_index], X.iloc[valid\_index]

y\_train1, y\_valid=Y.iloc[train\_index], Y.iloc[valid\_index]

print('Shape of X\_train :',X\_train1.shape)

print('Shape of X\_valid :',X\_valid.shape)

print('Shape of y\_train :',y\_train1.shape)

print('Shape of y\_valid :',y\_valid.shape)

#from imblearn.over\_sampling import SMOTE

#Synthetic Minority Oversampling Technique

sm = SMOTE(random\_state=147, ratio=1.0)

#Generating synthetic data points

X\_smote,y\_smote=sm.fit\_sample(X\_train1,y\_train1)

X\_smote\_v,y\_smote\_v=sm.fit\_sample(X\_valid,y\_valid)

print("AUC curve (Prob and Value) with Standardization and SMOTE oversampling")

aur\_prob\_value\_precision\_recall\_curve(models=models, X\_train = X\_smote, X\_test= X\_smote\_v, y\_train=y\_smote, y\_test=y\_smote\_v)

classification\_report\_models(models=models, X\_train = X\_smote, X\_test= X\_smote\_v, y\_train=y\_smote, y\_test=y\_smote\_v)

# LightGBM Model

#Training the model with simple train\_test\_split stratified data

#training data

lgb\_train=lgb.Dataset(X\_train,label=y\_train)

#validation data

lgb\_valid=lgb.Dataset(X\_test,label=y\_test)

params={'boosting\_type': 'gbdt',

'max\_depth' : -1, #no limit for max\_depth if <0

'objective': 'binary',

'boost\_from\_average':False,

'nthread': 20,

'metric':'auc',

'num\_leaves': 50,

'learning\_rate': 0.01,

'max\_bin': 100, #default 255

'subsample\_for\_bin': 100,

'subsample': 1,

'subsample\_freq': 1,

'colsample\_bytree': 0.8,

'bagging\_fraction':0.5,

'bagging\_freq':5,

'feature\_fraction':0.08,

'min\_split\_gain': 0.45, #>0

'min\_child\_weight': 1,

'min\_child\_samples': 5,

'is\_unbalance':True,

}

# f1\_score calculator function

def lgb\_f1\_score(y\_hat, data):

y\_true = data.get\_label()

y\_hat = np.round(y\_hat) # scikits f1 doesn't like probabilities

return 'f1', f1\_score(y\_true, y\_hat), True

evals\_result = {}

# lgbm simple

num\_rounds=5000

lgbm1= lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid],feval=lgb\_f1\_score,verbose\_eval=500,early\_stopping\_rounds = 2500)

# confusion matrix

print('Confusion matrix: Simple Lightgbm')

confusion\_matrix(y\_test, lgbm1.predict(X\_test).round())

def plot\_roc(y\_test, y\_pred, name):

fig, ax = plt.subplots(1,1, figsize=(6,6))

fpr2, tpr2, thresholds2 = roc\_curve(y\_test, y\_pred)

ax.legend([f' {auc(fpr2, tpr2)}'])

# ax.legned()

ax.set\_xlabel('False Positive Rate')

ax.set\_ylabel('True Positive Rate')

ax.set\_title('Receiver operating characteristic {}'.format(name))

ax.plot(fpr2, tpr2)

plot\_roc(y\_test, lgbm1.predict(X\_test).round(), name='Simple LightGBM')

#Training the model with StratifiedKFold()+SMOTE() data

#training data

lgb\_train2=lgb.Dataset(X\_smote,label=y\_smote)

#validation data

lgb\_valid2=lgb.Dataset(X\_smote\_v,label=y\_smote\_v)

num\_rounds=5000

lgbm3= lgb.train(params,lgb\_train2,num\_rounds,valid\_sets=[lgb\_train2,lgb\_valid2],feval=lgb\_f1\_score,verbose\_eval=500,early\_stopping\_rounds = 2500)

# confusion matrix

print('Confusion matrix: SMOTE Lightgbm')

confusion\_matrix(y\_test, lgbm3.predict(X\_test).round())

plot\_roc(y\_test, lgbm3.predict(X\_test).round(), name='SMOTE LightGBM')

#final submission

X\_test=test.drop(['ID\_code'],axis=1)

#predict the model, probability predictions

lightgbm\_predict\_prob3=lgbm3.predict(X\_test,random\_state=42,num\_iteration=lgbm3.best\_iteration)

lightgbm\_predict\_prob1=lgbm1.predict(X\_test,random\_state=42,num\_iteration=lgbm1.best\_iteration)

#Convert to binary output 1 or 0

lightgbm\_predict3=lightgbm\_predict\_prob3.round()

lightgbm\_predict1=lightgbm\_predict\_prob1.round()

submit=pd.DataFrame({'ID\_code':test['ID\_code'].values})

submit1=pd.DataFrame({'ID\_code':test['ID\_code'].values})

#submit['lightgbm\_predict\_prob']=lightgbm\_predict\_prob3

submit['target']=lightgbm\_predict3.astype(int)

submit1['target']=lightgbm\_predict1.astype(int)

submit.to\_csv('submission.csv',index=False)

submit1.to\_csv('submission1.csv',index=False)

submit1.head()

submit.head()

## R code

library(glmnet)

library(lightgbm)

library(pROC)

library(ROCR)

library(PRROC)

library(ROSE)

library(caret)

library(ROCR)

library(PRROC)

library(randomForest)

library(e1071) # for skewness

library(ggthemes)

library(ggplot2)

library(gridExtra)

library(dplyr)

library(devtools)

library(party)

library(caTools)

library(tidyr)

library(dplyr)

library(lazyeval)

#library(Rfast)

list.files('/input')

#load dataset

train = read.csv('/input//train.csv')

test = read.csv('/input//test.csv')

# taking sneak peak to datasets

cat('Dimension of our Train data', dim(train),' \n Data feature informations')

print(summary(train))

cat('Dimension of our Test data', dim(test),'\n Data feature informations')

print(summary(test))

print(head(train))

print(head(test))

### EDA for understanding datasets and getting clues for feature selections.

# Datatypes in dataset

cat('Train target column datatype: ',class(train$target))

cat('\nTrain var\_0 column datatype: ',class(train$var\_0))

cat('\nDataype values in target: ',class(train$target))

cat("\nValues in target: ", unique(train$target))

# Missing value analysis

cat('Train missing values:',sum(sum(is.na(train))))

cat('\nTest missing values:',sum(sum(is.na(test))))

# Looking Skewness

cat('\nTrain skewness:\n',skewness(train[,-c(1,2)]),'\nTest skewness:\n', skewness(test[,-c(1)]))

# Digging target variable

target = train$target

cat('Different values in target:\n')

print(table(target))

cat("\nThe target values with 1 and 0:\n 1 0\n",table(target)/length(target)\*100)

ggplot(train,aes(target))+theme\_bw()+geom\_bar(stat='count',fill='lightgreen')

# Boxplot Analysis

boxplot\_function = function(train, feature\_groups){

col\_names <- colnames(train)[c(2,feature\_groups)]

temp <- gather(train[,col\_names], key="features", value="value", -target)

temp$target <- factor(temp$target)

temp$features <- factor(temp$features, levels=col\_names[-1], labels=col\_names[-1])

ggplot(data=temp, aes(y=value)) +

geom\_boxplot(aes(fill=target, color=target), alpha=0.3) +

scale\_color\_manual(values = c("1" = "dodgerblue", "0"="firebrick1")) +

theme\_classic() +

facet\_wrap(~ features, ncol = 4, scales = "free")

}

boxplot\_function(train, 3:22)

boxplot\_function(train, 3:22+20)

boxplot\_function(train, 3:22+40)

boxplot\_function(train, 3:22+60)

boxplot\_function(train, 3:22+80)

boxplot\_function(train, 3:22+100)

boxplot\_function(train, 3:22+120)

boxplot\_function(train, 3:22+160)

boxplot\_function(train, 3:22+180)

#histogram anaylsis

hist\_function = function(train, feature\_groups){

col\_names <- colnames(train)[c(2,feature\_groups)]

temp <- gather(train[,col\_names], key="features", value="value", -target)

temp$target <- factor(temp$target)

temp$features <- factor(temp$features, levels=col\_names[-1], labels=col\_names[-1])

ggplot(data=temp, aes(x=value)) +

geom\_density(aes(fill=target, color=target), alpha=0.3) +

scale\_color\_manual(values = c("1" = "dodgerblue", "0"="firebrick1")) +

theme\_classic() +

facet\_wrap(~ features, ncol = 4, scales = "free")

}

hist\_function(train, 3:22)

hist\_function(train, 3:22+20)

hist\_function(train, 3:22+40)

hist\_function(train, 3:22+60)

hist\_function(train, 3:22+80)

hist\_function(train, 3:22+100)

hist\_function(train, 3:22+120)

hist\_function(train, 3:22+160)

hist\_function(train, 3:22+180)

# Creating New features

col = train$target

col = data.frame(col)

col$mean = apply(train[,-c(1,2)],MARGIN=1,FUN=mean)

col$sum = apply(train[,-c(1,2)],MARGIN=1,FUN=sum)

col$sd = apply(train[,-c(1,2)],MARGIN=1,FUN=sd)

col$skewness = apply(train[,-c(1,2)],MARGIN=1,FUN=skewness)

col$max = apply(train[,-c(1,2)],MARGIN=1,FUN=max)

col$min = apply(train[,-c(1,2)],MARGIN=1,FUN=min)

col$kurtosis = apply(train[,-c(1,2)],MARGIN=1,FUN=kurtosis)

col$median = apply(train[,-c(1,2)],MARGIN=1,FUN=median)

colnames(col) = c("mean", "target", 'sum', 'sd', 'skewness', 'max', 'min', 'kurt', 'median')

col1 = subset(col, select = c(mean, sum, sd, skewness, max,min, kurt, median))

# Looking at Distribution trnds

dist\_FE = function(data, column,name){

ggplot(data, aes(x=column, fill=as.factor(col$target)), height=3) +

geom\_density(alpha=.5, position="identity")+ ggtitle('Distribution of values per row in train dataset:',name)

}

#dist\_FE(col, col$mean, 'mean')

#dist\_FE(col, col$sum, 'sum')

#dist\_FE(col, col$min, 'min')

#dist\_FE(col, col$max, 'max')

#dist\_FE(col, col$skewness, 'skewness')

#dist\_FE(col, col$kurtosis, 'kurtosis')

#dist\_FE(col, col$median, 'median')

#dist\_FE(col, col$sd, 'sd')

# Correlation Analysis

cormat <- cor(train[,-c(1,2)])

summary(cormat[upper.tri(cormat)])

# heatmap

heatmap(cormat, scale = 'none')

## Data Preprocesiing

# PCA Analysis

pca = prcomp(train[,-c(1,2)], scale = F)

pca\_var = pca$sdev^2

pca\_var\_explain = pca\_var/sum(pca\_var)

plot(pca\_var\_explain, xlab = "Principal Component",

ylab = "Proportion of Variance Explained",

type = "b")

plot(cumsum(pca\_var\_explain), xlab = "Principal Component",

ylab = "Cumulative Proportion of Variance Explained",

type = "b")

# Feature Importance

library(randomForest)

#Training the Random forest classifier

#convert to int to factor

train$target<-as.factor(train$target)

#setting the mtry

mtry<-floor(sqrt(200))

#fitting the ranndom forest

rf<-randomForest(target~.,train[,-c(1)],mtry=mtry,ntree=10,importance=TRUE)

# Looking into important features

VarImp<-importance(rf,type=2)

VarImp= data.frame(VarImp)

VarImp$var = row.names(VarImp)

sort\_Var = VarImp[order(VarImp$MeanDecreaseGini, decreasing = F),][1:25,c(2)]

remove\_col = strsplit(sort\_Var, ' ')

remove\_col

# Adding New features to data

train\_FE = cbind.data.frame(train, col1)

train\_FE$ID\_code = NULL

train\_FE$target = as.factor(train\_FE$target)

test2 = test

test2$ID\_code=NULL

test2$mean = apply(test2,MARGIN=1,FUN=mean)

test2$sum = apply(test2,MARGIN=1,FUN=sum)

test2$sd = apply(test2,MARGIN=1,FUN=sd)

test2$skewness = apply(test2,MARGIN=1,FUN=skewness)

test2$max = apply(test2,MARGIN=1,FUN=max)

test2$min = apply(test2,MARGIN=1,FUN=min)

test2$kurtosis = apply(test2,MARGIN=1,FUN=kurtosis)

test2$median = apply(test2,MARGIN=1,FUN=median)

dim(train\_FE)

dim(test2)

# stratified sampling

require(sampling)

s = strata(train\_FE, c('target'), size=c(1600, 400), method = 'srswor')

stratas\_data = getdata(train\_FE, s)

stratas\_data$ID\_unit = NULL

stratas\_data$Prob = NULL

stratas\_data$Stratum = NULL

head(stratas\_data)

# train\_test\_split

split = sample.split(stratas\_data, SplitRatio = 0.75)

training\_set = subset(train\_FE, split == TRUE)

test\_set = subset(train\_FE, split == FALSE)

head(training\_set)

head(test\_set)

dim(training\_set)

dim(test\_set)

#Training the Random forest classifier

#setting the mtry

mtry=floor(sqrt(200))

#fitting the ranndom forest

rf=randomForest(target~.,training\_set,mtry=mtry,ntree=10,importance=TRUE)

rf\_predict = predict(rf, newdata = test\_set)

# DEcision Tree

library(C50)

x.ct = C5.0(target ~ ., data=training\_set)

x.ct\_predict = predict(x.ct, newdata = test\_set)

#NaiveBayes

nb = naiveBayes(target~.,data=training\_set)

nb\_predict = predict(nb, newdata = test\_set)

#library(glm)

#Logistic regression model

lr\_model =glm(target~., data=training\_set ,family = "binomial")

lr\_predict = predict(lr\_model,newdata = test\_set, type='response')

cat('RandomForest')

confusionMatrix( as.factor(test\_set$target),rf\_predict)

cat('DecisionTree')

confusionMatrix( as.factor(test\_set$target),x.ct\_predict )

cat('NaiveBayes')

confusionMatrix( as.factor(test\_set$target),nb\_predict )

cat('LogisticRegressor')

confusionMatrix(table( as.factor(test\_set$target),ifelse(lr\_predict>0.5,1,0) ))

cat('RandomForest')

test.forest = predict(rf, newdata = test\_set, type='prob')

forestpred = prediction( test.forest[,2],test\_set$target)

forestperf = performance(forestpred, 'tpr', 'fpr')

forestperf2 = performance(forestpred, 'prec', 'rec')

cat('DecisionTree')

test.ct = predict(x.ct, newdata = test\_set, type='prob')

ct\_pred = prediction(test.ct[,2], test\_set$target)

ct\_perf = performance(ct\_pred, 'tpr', 'fpr')

ct\_perf2 = performance(ct\_pred, 'prec', 'rec')

cat('NaiveBayes')

test.nb = predict(nb, newdata = test\_set, type='raw')

nb\_pred = prediction(test.nb[,2], test\_set$target)

nb\_perf = performance(nb\_pred, 'tpr', 'fpr')

nb\_perf2 = performance(nb\_pred, 'prec', 'rec')

cat('LogisticRegressor')

test.lr = predict(lr\_model, newdata = test\_set, type='response')

lr\_pred = prediction(ifelse(test.lr>0.5,1,0), test\_set$target)

lr\_perf = performance(lr\_pred, 'tpr', 'fpr')

lr\_perf2 = performance(lr\_pred, 'prec', 'rec')

plot(forestperf, main='RDF ROC', colorize=T)

plot(ct\_perf, main='DCT ROC', colorize=T)

plot(nb\_perf, main='NB ROC', colorize=T)

plot(lr\_perf, main='LR ROC', colorize=T)

# LogisticRegression with ROSE

#Split the data using CreateDataPartition

train1.index = createDataPartition(train$target,p=0.8,list=FALSE)

train1.data = train[train1.index,]

valid1.data = train[-train1.index,]

dim(train1.data)

dim(valid1.data)

#Random Oversampling Examples(ROSE)

train.rose = ROSE(target~., data =train1.data[,-c(1)],seed=32)$data

table(train.rose$target)

valid.rose = ROSE(target~., data =valid1.data[,-c(1)],seed=32)$data

table(valid.rose$target)

#Logistic regression model

lr\_rose = glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family ="binomial")

summary(lr\_rose)

#Cross validation prediction

cv\_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),family = "binomial", type.measure = "class")

#Minimum lambda

cv\_rose$lambda.min

#plot the auc score vs log(lambda)

plot(cv\_rose)

#Model performance on validation dataset

cv\_predict.rose = predict(cv\_rose,as.matrix(valid.rose),s = "lambda.min",type = "class")

# Confusion Matrix

confusionMatrix(table(data=as.factor(cv\_predict.rose),reference=as.factor(valid.rose$target)))

#### LightGBM

# Implementing LightGBM with raw train data

#Convert data frame to matrix

X\_train = as.matrix(train1.data[,-c(1,2)])

y\_train = as.matrix(train1.data$target)

X\_valid = as.matrix(valid1.data[,-c(1,2)])

y\_valid = as.matrix(valid1.data$target)

test\_data = as.matrix(test[,-c(1)])

#training data

lgb.train = lgb.Dataset(data=X\_train, label=y\_train)

#Validation data

lgb.valid = lgb.Dataset(data=X\_valid,label=y\_valid)

lgb.grid = list(objective = "binary",

metric = "auc",

min\_sum\_hessian\_in\_leaf = 1,

feature\_fraction = 0.7,

bagging\_fraction = 0.7,

bagging\_freq = 5,

learning\_rate=0.1,

num\_leaves=100,

num\_threads=8,

min\_data = 100,

max\_bin = 200,

min\_data\_in\_bin=150,

min\_gain\_to\_split = 20,

min\_data\_in\_leaf = 40,

is\_unbalance = TRUE)

# training the model

lgbm.model1 = lgb.train(params = lgb.grid, data = lgb.train, nrounds =1000,eval\_freq =100,valids=list(val1=lgb.train,val2=lgb.valid),

early\_stopping\_rounds = 300)

#Coufusion Matrix

confusionMatrix(table( as.matrix(y\_valid), ifelse(predict(lgbm.model1,X\_valid)>0.5,1,0)) )

# ROC curve

lgbm\_p1 = prediction(ifelse(predict(lgbm.model1,X\_valid)>0.5,1,0),as.matrix(y\_valid))

plot( performance(lgbm\_p1, 'tpr', 'fpr') , colorize=T, main='Raw LGBM',auc=T)

#lgbm model performance on test data

lgbm\_pred\_prob1 = predict(lgbm.model1,test\_data)

head(lgbm\_pred\_prob1)

#Convert to binary, threshold 0.5

lgbm\_pred1 = ifelse(lgbm\_pred\_prob1>0.5,1,0)

head(lgbm\_pred1)

# Implementing LightGBM with ROSE raw train data

library(lightgbm)

#Convert data frame to matrix

set.seed(5432)

X\_train = as.matrix(train.rose[,-c(1)])

y\_train = as.matrix(train.rose$target)

X\_valid = as.matrix(valid.rose[,-c(1)])

y\_valid = as.matrix(valid.rose$target)

test\_data = as.matrix(test[,-c(1)])

#training data

lgb.train = lgb.Dataset(data=X\_train, label=y\_train)

#Validation data

lgb.valid =lgb.Dataset(data=X\_valid,label=y\_valid)

# training the model

lgbm.model2= lgb.train(params = lgb.grid, data = lgb.train, nrounds =1000,eval\_freq =100,valids=list(val1=lgb.train,val2=lgb.valid),

early\_stopping\_rounds = 300)

#Confusion matrix

confusionMatrix(table( as.matrix(y\_valid), ifelse(predict(lgbm.model2,X\_valid)>0.5,1,0)) )

# ROC curve

lgbm\_p = prediction(ifelse(predict(lgbm.model2,X\_valid)>0.5,1,0),as.matrix(y\_valid))

plot( performance(lgbm\_p, 'tpr', 'fpr') , colorize=T, main='ROSE LGBM',auc=T)

#lgbm model performance on test data

lgbm\_pred\_prob2 = predict(lgbm.model2,test\_data)

head(lgbm\_pred\_prob2)

#Convert to binary, threshold 0.5

lgbm\_pred2 = ifelse(lgbm\_pred\_prob2>0.5,1,0)

head(lgbm\_pred2)

# saving lgbm prediction

submit = data.frame(ID\_code=test$ID\_code,target=lgbm\_pred3)

colnames(submit) = c('ID\_code','target')

write.csv(submit,'submission.csv',row.names=F)

submit2 = data.frame(ID\_code=test$ID\_code,target=lgbm\_pred2)

colnames(submit2) = c('ID\_code','target')

write.csv(submit2,'submission2.csv',row.names=F)

submit1 = data.frame(ID\_code=test$ID\_code,target=lgbm\_pred1)

colnames(submit1) = c('ID\_code','target')

write.csv(submit1,'submission1.csv',row.names=F)

head(submit)

head(submit1)

head(submit2)

# Refrences

# <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>

# <https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc>

# <https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/>

# <https://www.analyticsvidhya.com/blog/2016/03/practical-guide-deal-imbalanced-classification-problems/>

# https://www.kaggle.com/pritamjena/testing-the-imbalanced-data-using-rose-package