Santander Transaction Prediction

Vishal Maurya

14 August 2019

Contents

3 Conclusion

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)

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_197	var_198	var_199
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784	8.5635	12.7803	-1.0914
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267	8.7889	18.3560	1.9518
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213	8.2675	14.7222	0.3965
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275	10.2922	17.9697	-8.9996
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267	9.5031	17.9974	-8.8104

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

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_197	var_198	var_199
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	...	-2.1556	11.8495	-1.4300	2.4508	13.7112	2.4669	4.3654	10.7200	15.4722	-8.7197
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	...	10.6165	8.8349	0.9403	10.1282	15.5765	0.4773	-1.4852	9.8714	19.1293	-20.9760
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	...	-0.7484	10.9935	1.9803	2.1800	12.9813	2.1281	-7.1086	7.0618	19.8956	-23.1794
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	...	9.5702	9.0766	1.6580	3.5813	15.1874	3.1656	3.9567	9.2295	13.0168	-4.2108
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	...	4.2259	9.1723	1.2835	3.3778	19.5542	-0.2860	-5.1612	7.2882	13.9260	-9.1846

Table 1.3: Predictor Variables

```
columns: Index(['ID_code', 'var_0', 'var_1', 'var_2', 'var_3', 'var_4', 'var_5',
  'var_6', 'var_7', 'var_8',
  ...
  'var_190', 'var_191', 'var_192', 'var_193', 'var_194', 'var_195',
  'var_196', 'var_197', 'var_198', 'var_199'],
 dtype='object', length=201)
```

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

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...	var_190	var_191	2
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	...	200000.000000	200000.000000	
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850	0.284162	...	3.234440	7.438408	
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	3.332634	...	4.559922	3.023272	
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	-10.505500	...	-14.093300	-2.691700	
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	-2.317800	...	-0.058825	5.157400	
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	0.393700	...	3.203600	7.347750	
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	2.937900	...	6.406200	9.512525	
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	10.151300	...	18.440900	16.716500	

8 rows × 201 columns

var_7	var_8	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_197	var_198	var_199
200000.000000	200000.000000	...	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
16.545850	0.284162	...	3.234440	7.438408	1.927839	3.331774	17.993784	-0.142088	2.303335	8.908158	15.870720	-3.326537
3.418076	3.332634	...	4.559922	3.023272	1.478423	3.992030	3.135162	1.429372	5.454369	0.921625	3.010945	10.438015
5.349700	-10.505500	...	-14.093300	-2.691700	-3.814500	-11.783400	8.694400	-5.261000	-14.209600	5.960600	6.299300	-38.852800
13.943800	-2.317800	...	-0.058825	5.157400	0.889775	0.584600	15.629800	-1.170700	-1.946925	8.252800	13.829700	-11.208475
16.456800	0.393700	...	3.203600	7.347750	1.901300	3.396350	17.957950	-0.172700	2.408900	8.888200	15.934050	-2.819550
19.102900	2.937900	...	6.406200	9.512525	2.949500	6.205800	20.396525	0.829600	6.556725	9.593300	18.064725	4.836800
27.691800	10.151300	...	18.440900	16.716500	8.402400	18.281800	27.928800	4.272900	18.321500	12.000400	26.079100	28.500700

var_7	var_8	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_197	var_198	var_199
200000.000000	200000.000000	...	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
16.545850	0.284162	...	3.234440	7.438408	1.927839	3.331774	17.993784	-0.142088	2.303335	8.908158	15.870720	-3.326537
3.418076	3.332634	...	4.559922	3.023272	1.478423	3.992030	3.135162	1.429372	5.454369	0.921625	3.010945	10.438015
5.349700	-10.505500	...	-14.093300	-2.691700	-3.814500	-11.783400	8.694400	-5.261000	-14.209600	5.960600	6.299300	-38.852800
13.943800	-2.317800	...	-0.058825	5.157400	0.889775	0.584600	15.629800	-1.170700	-1.946925	8.252800	13.829700	-11.208475
16.456800	0.393700	...	3.203600	7.347750	1.901300	3.396350	17.957950	-0.172700	2.408900	8.888200	15.934050	-2.819550
19.102900	2.937900	...	6.406200	9.512525	2.949500	6.205800	20.396525	0.829600	6.556725	9.593300	18.064725	4.836800
27.691800	10.151300	...	18.440900	16.716500	8.402400	18.281800	27.928800	4.272900	18.321500	12.000400	26.079100	28.500700

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.

```
# Missing value analysis
print('Train missing values:',train.isnull().sum().sum())
print('Test missing values:',test.isnull().sum().sum())
```

```
Train missing values: 0
Test missing values: 0
```

Python and R code as follows:

```
# Missing value analysis
cat('Train missing values:',sum(sum(is.na(train))))
cat('\nTest missing values:',sum(sum(is.na(test))))
```

```
# Missing value analysis
print('Train missing values:',train.isnull().sum().sum())
print('Test missing values:',test.isnull().sum().sum())
```

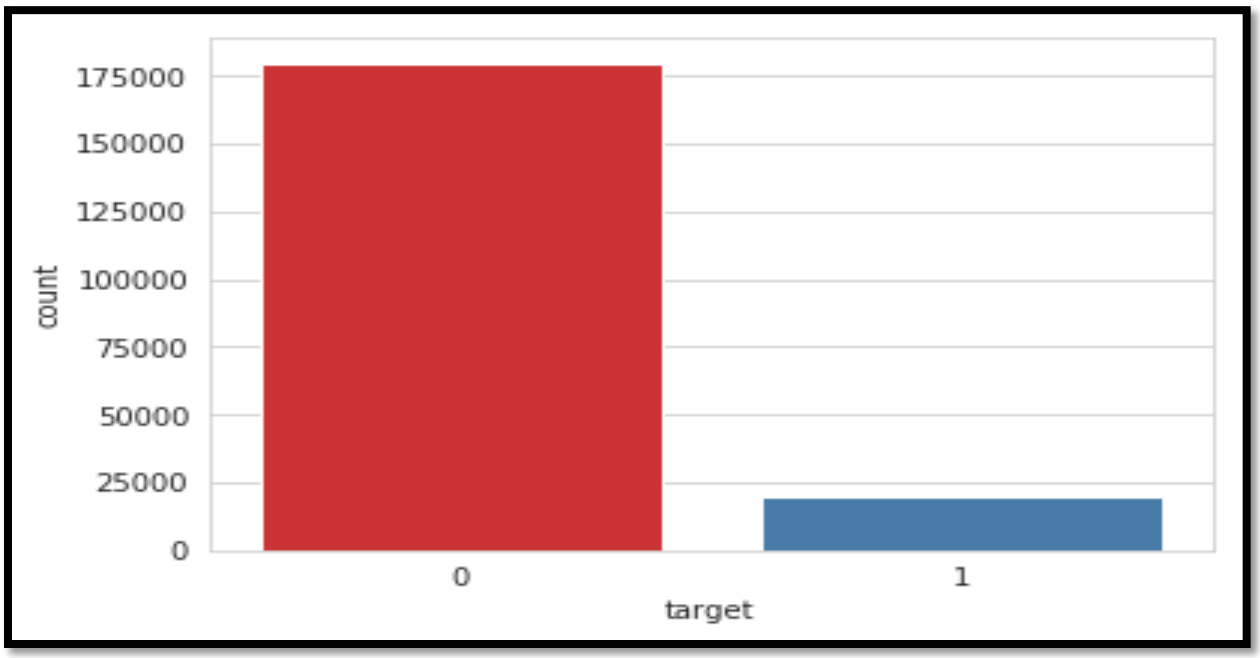
```
Train missing values: 0
Test missing values: 0
```

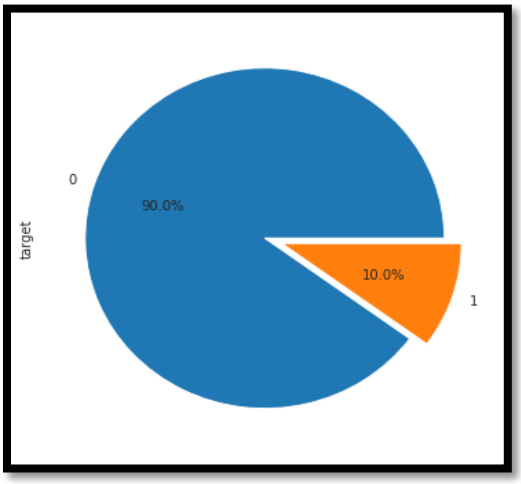
2.1.2 Visualization

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






```
Different values in target:
 0    179902
1     20098
Name: target, dtype: int64

There are 10.049% target values with 1
```

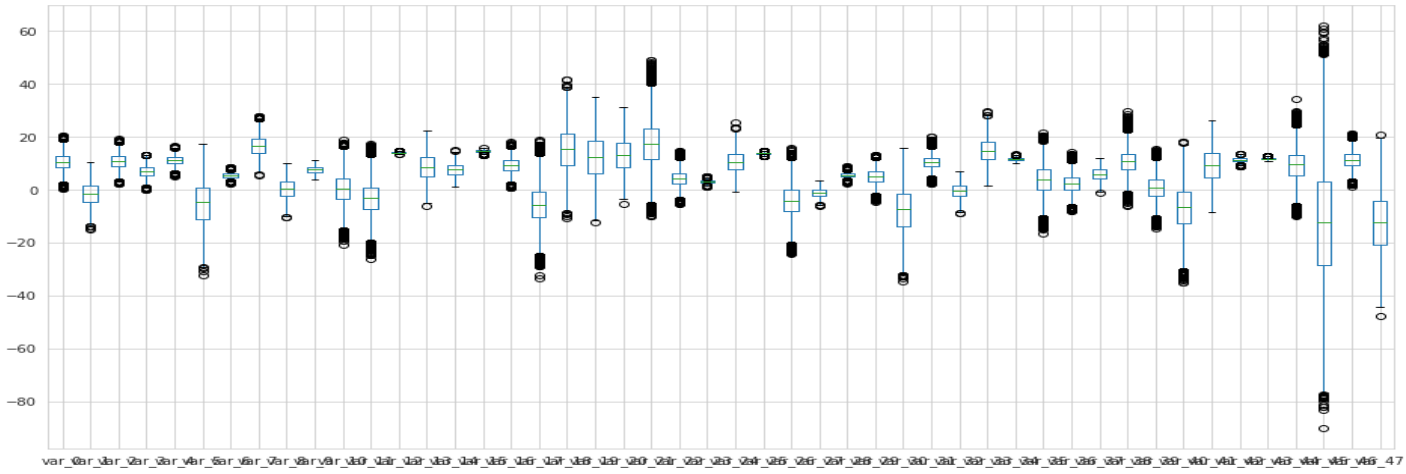
Observation:

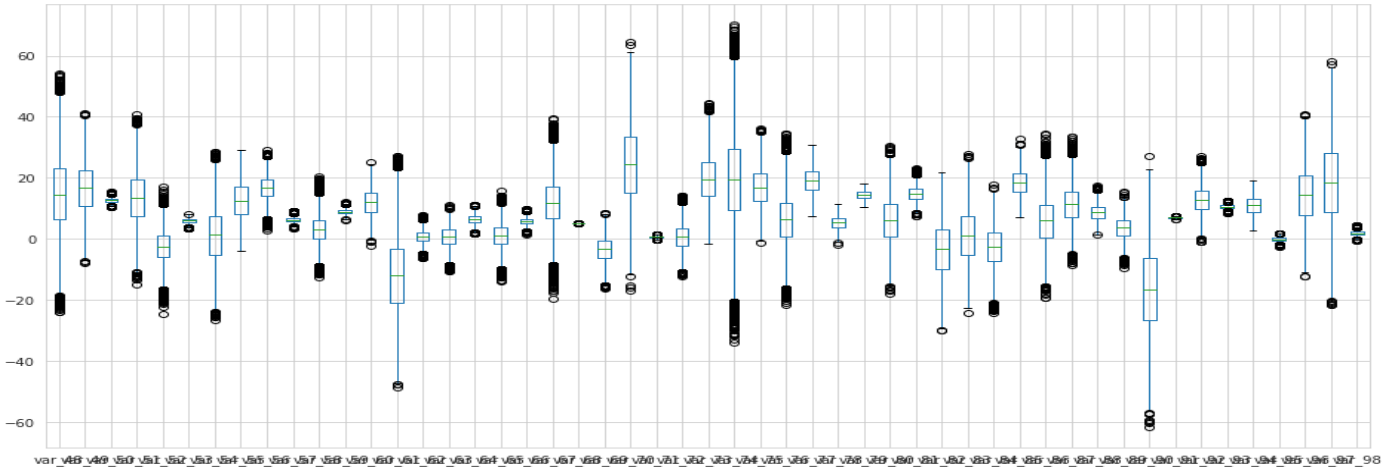
Only about 10% of total target is belong to class 1, therefore , this train dataset is imbalanced, hence need different sampling methods than random sampling.

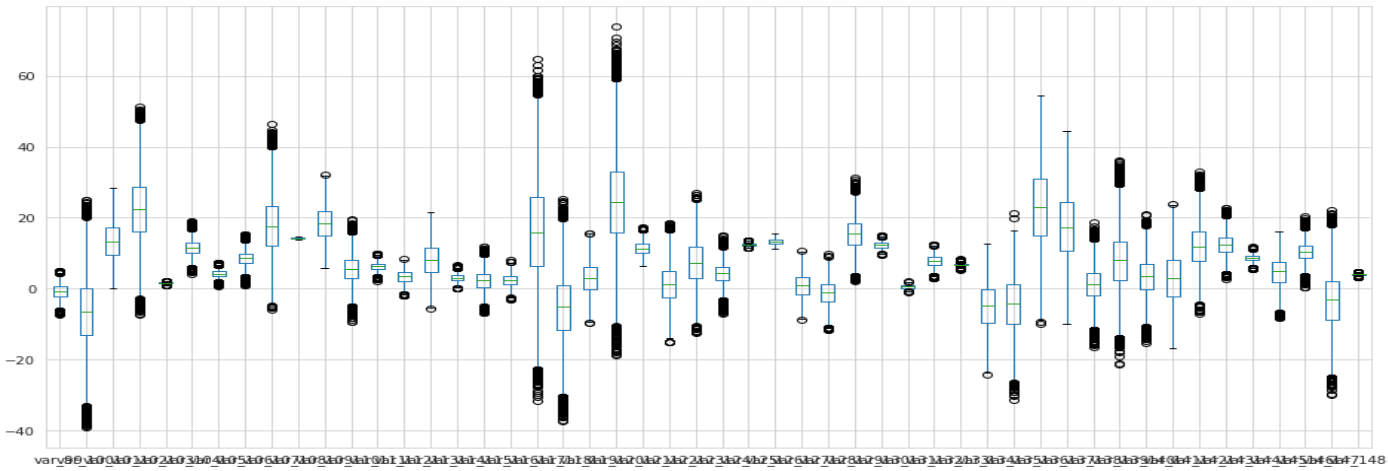
1. We have an imbalanced class problem. where 90% of the data is the number of customers those will not make a transaction and 10% of the data is those who will make a transaction. The number of customers that will not make a transaction is much higher than those that will.
1. The dataset is unbalanced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

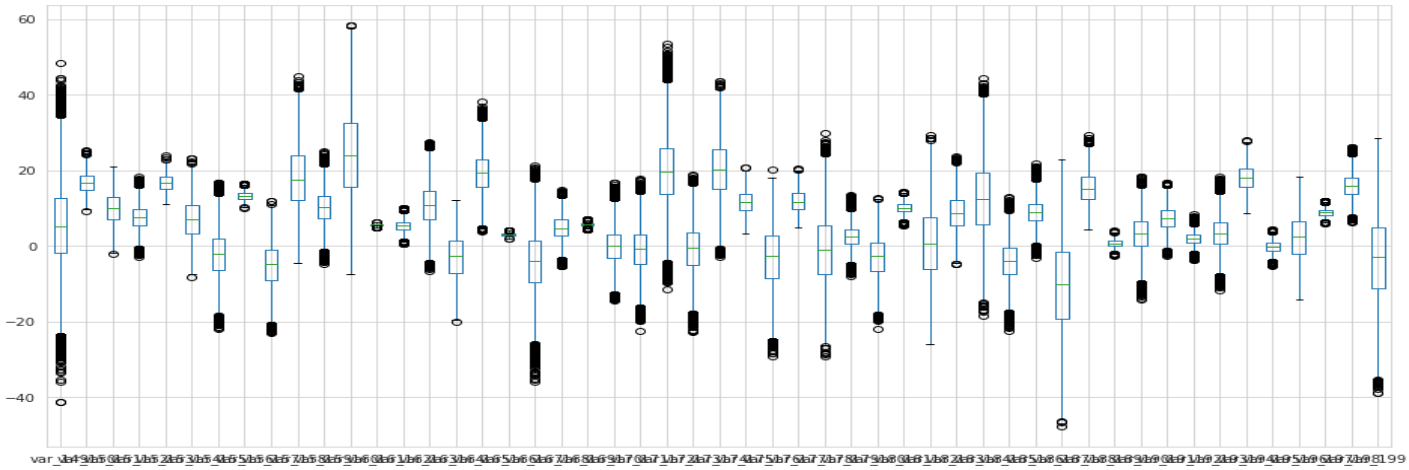
2.1.2.2 Outlier Analysis: Boxplot

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

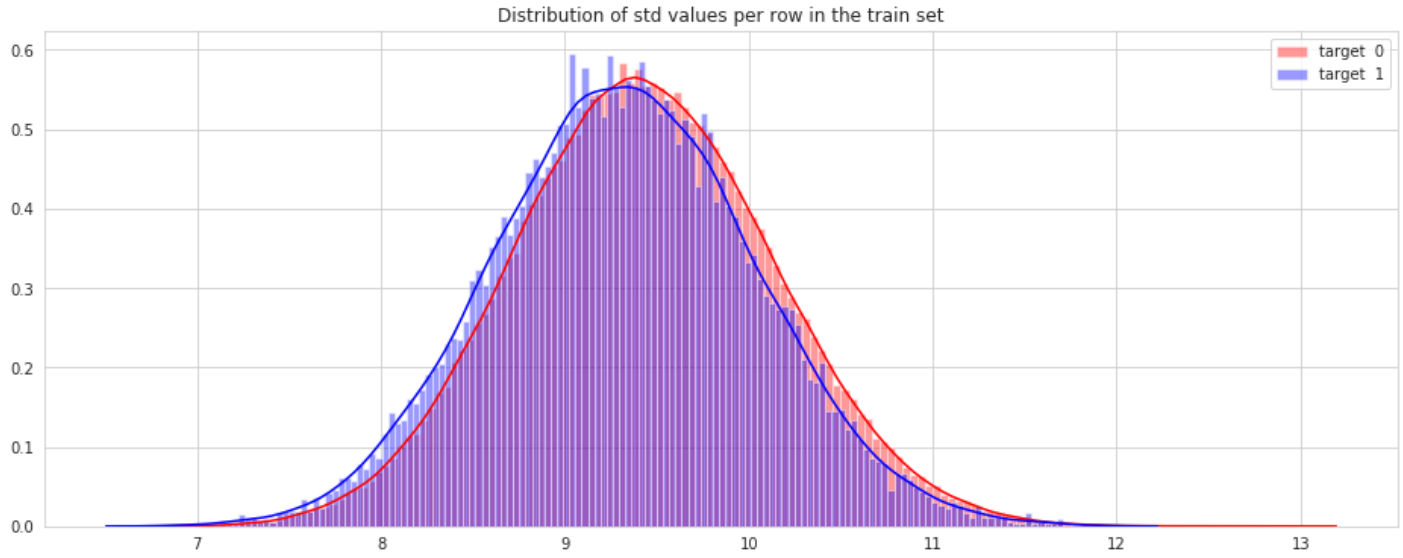

0
1
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99

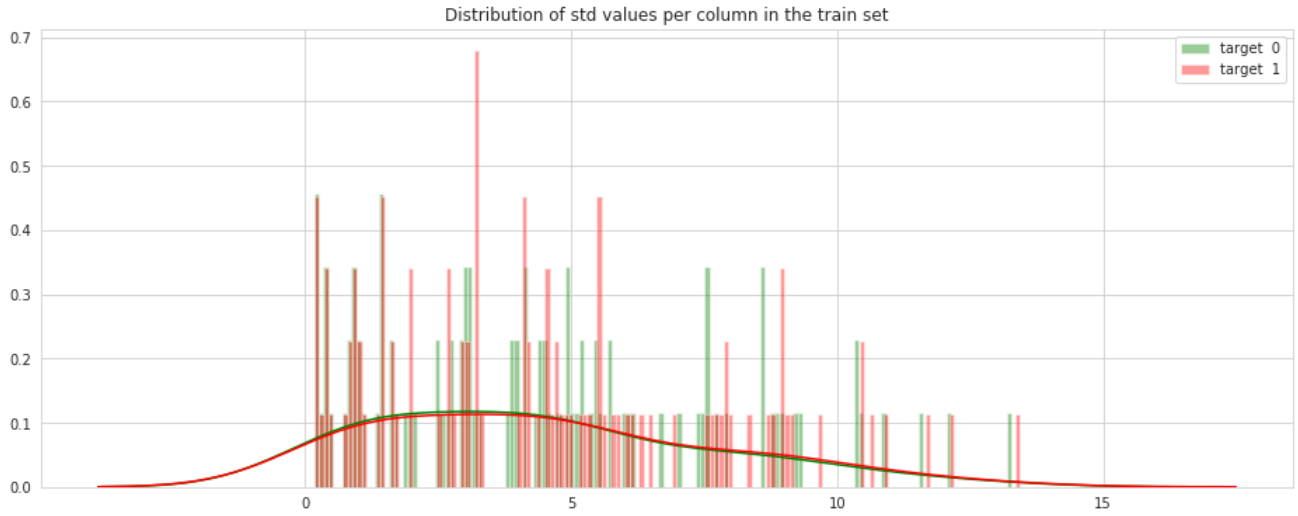
Observations:

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.
1. 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.

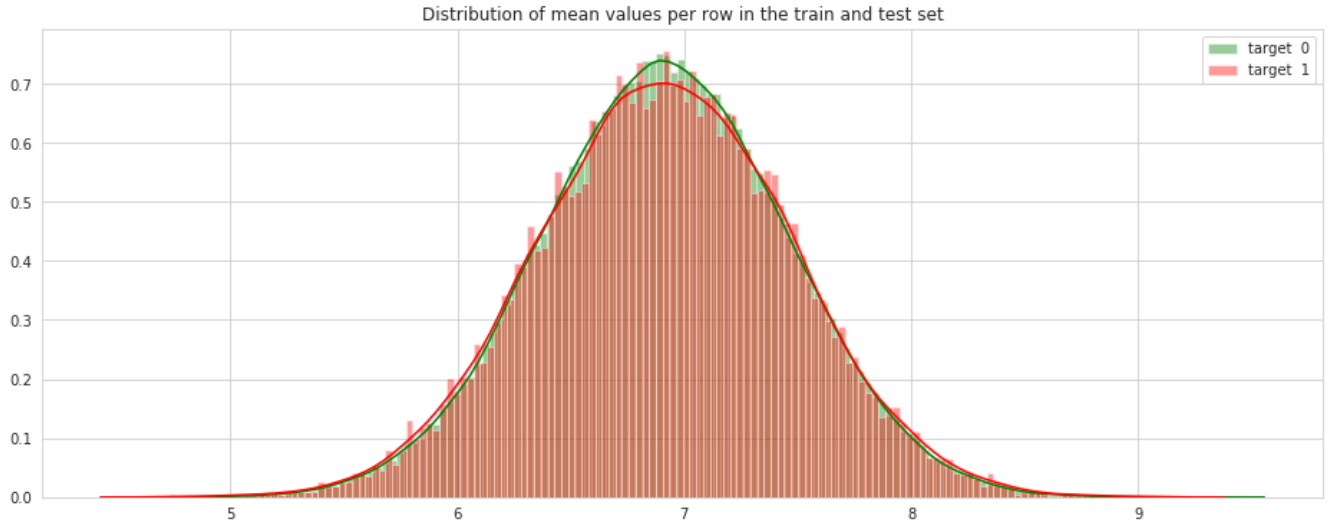
Distribution of Aggregates along the column and row

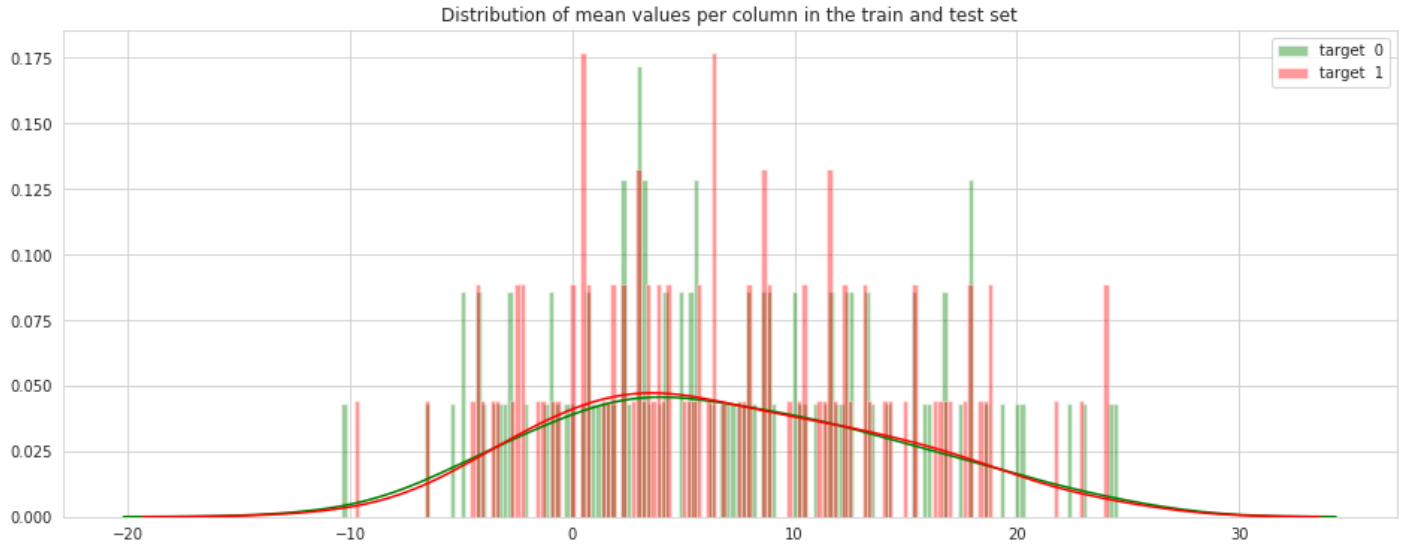
Let us look distribution of std values:



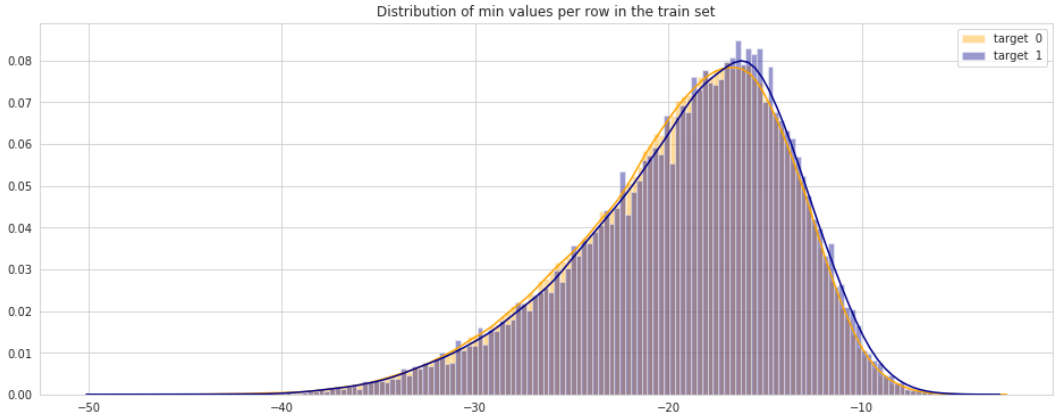


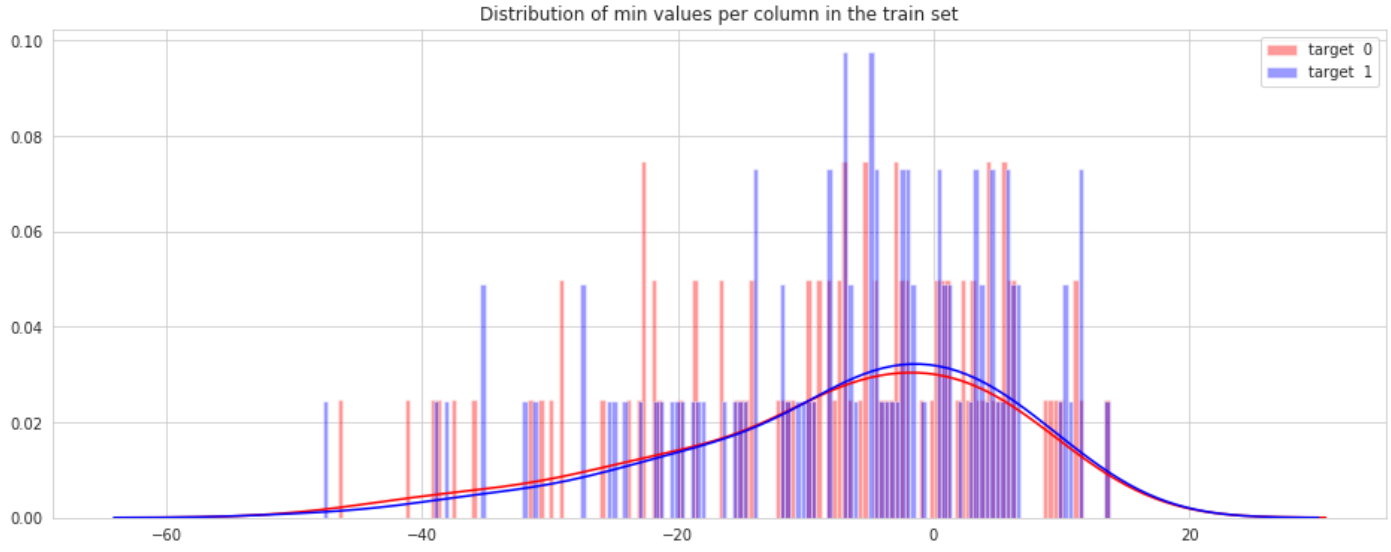
Let us look distribution of mean values:



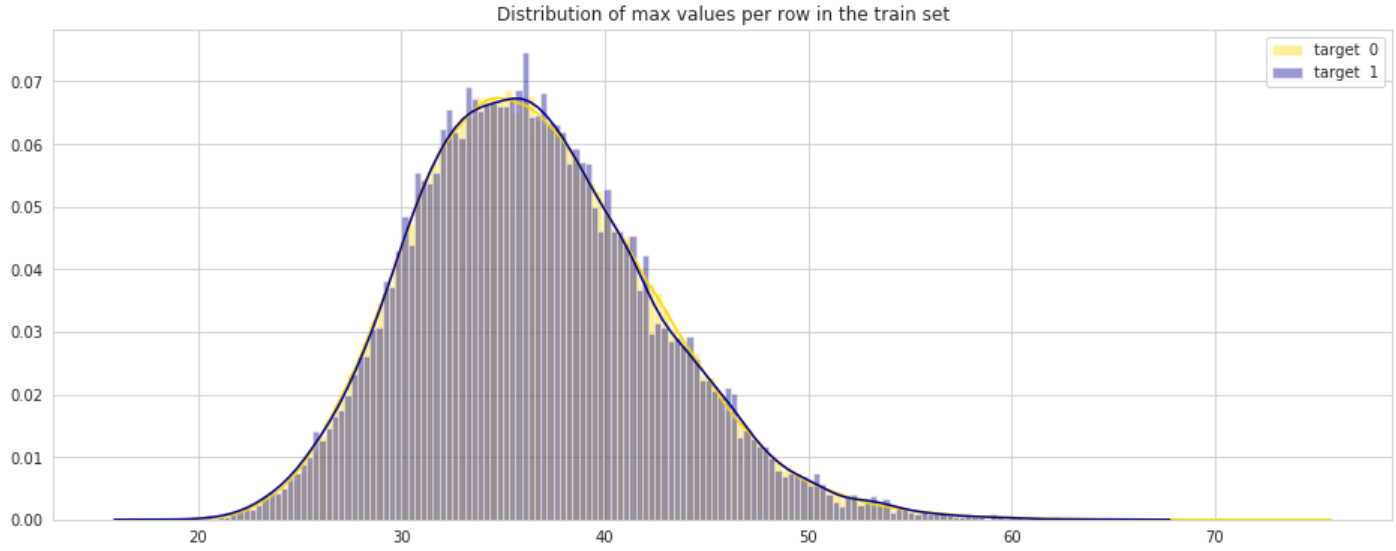


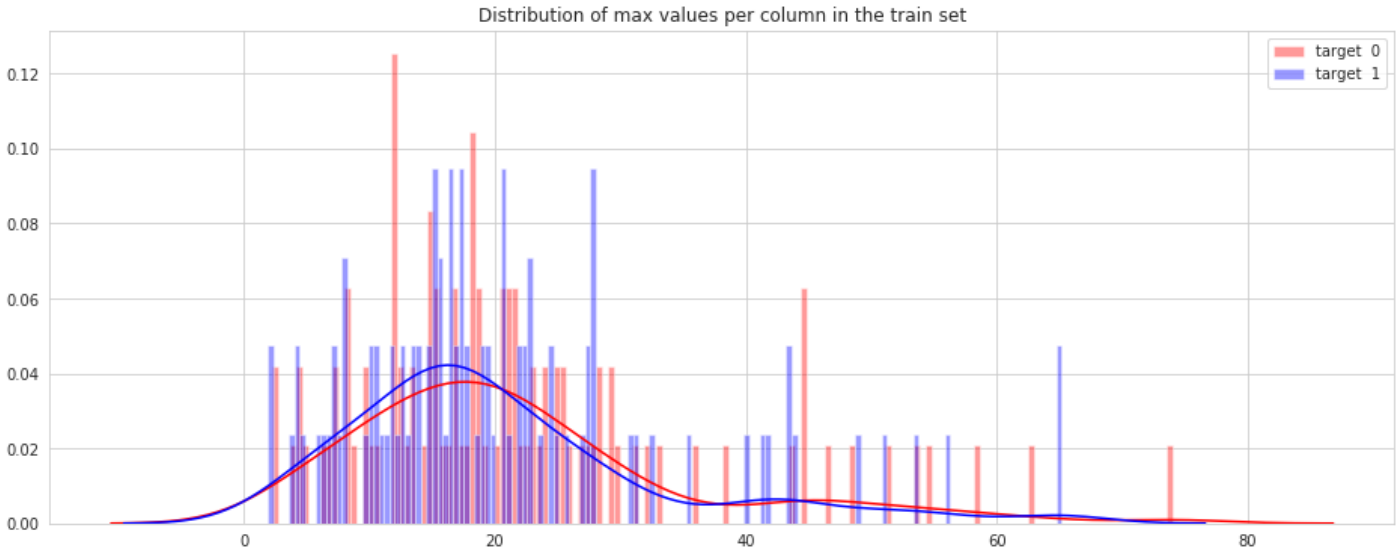
Let us look distribution of min values



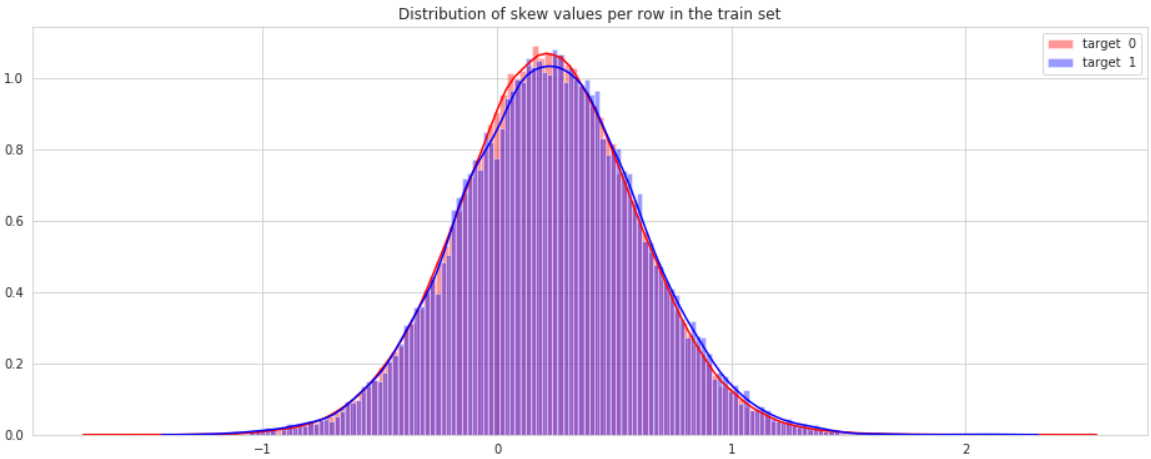


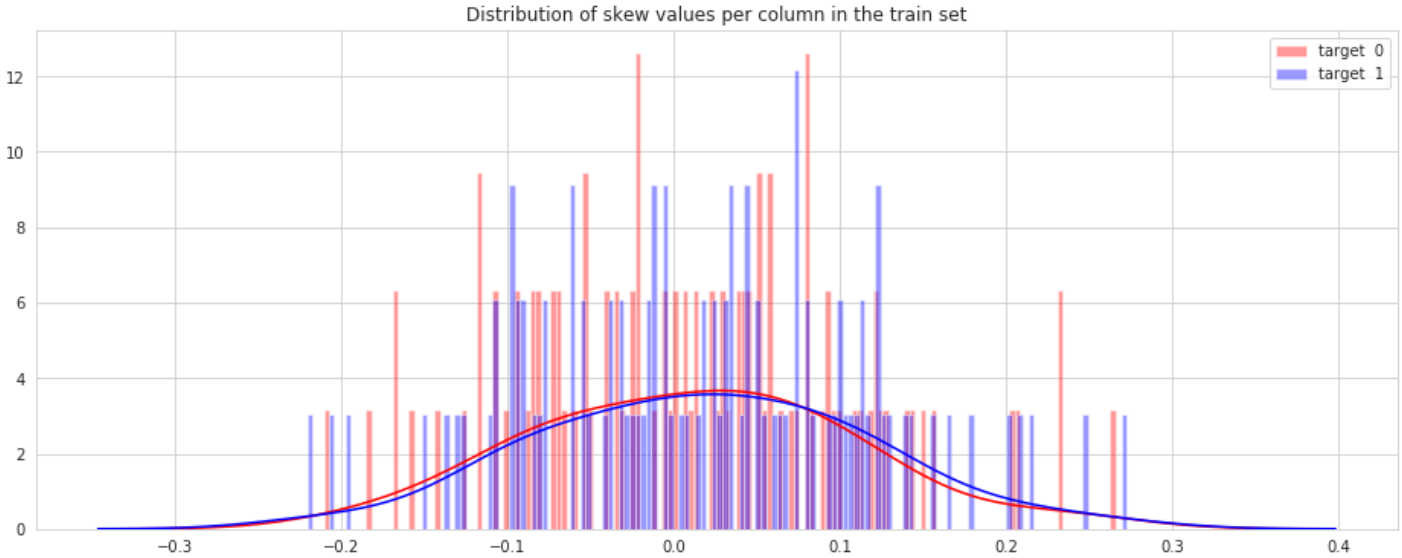
Let us look distribution of max values:



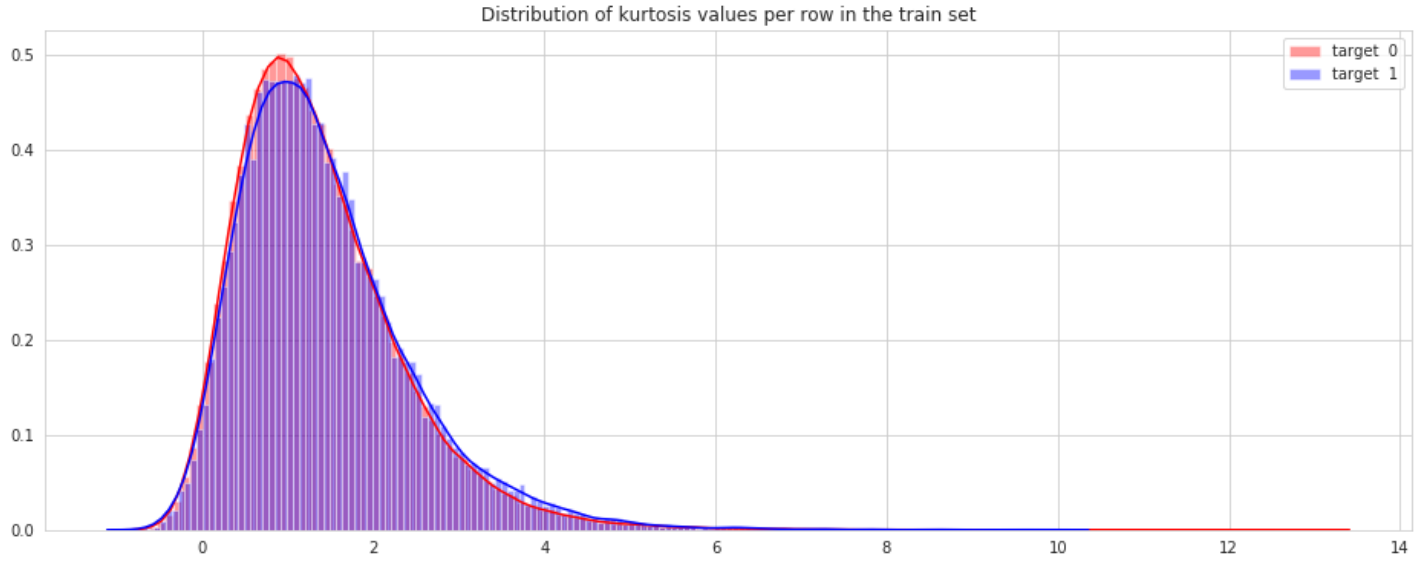


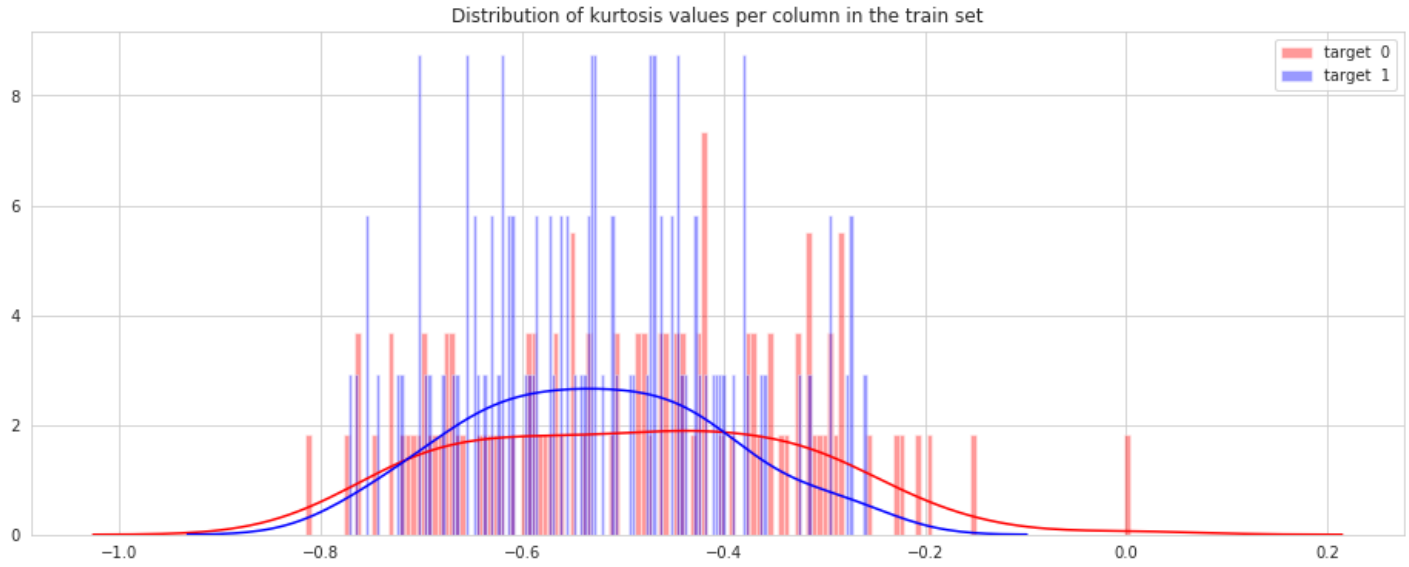
Let us look distribution of skewness values:



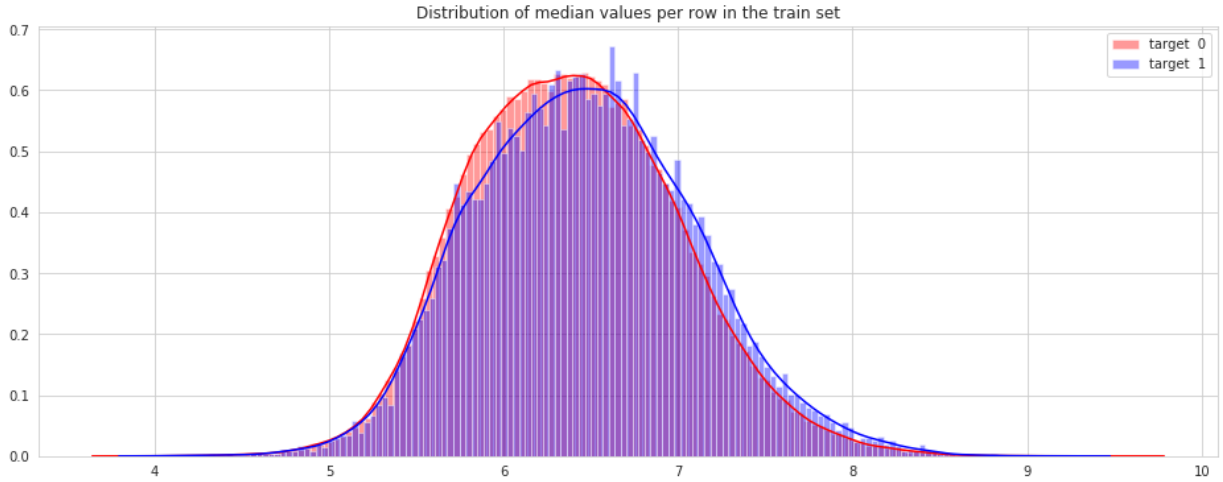


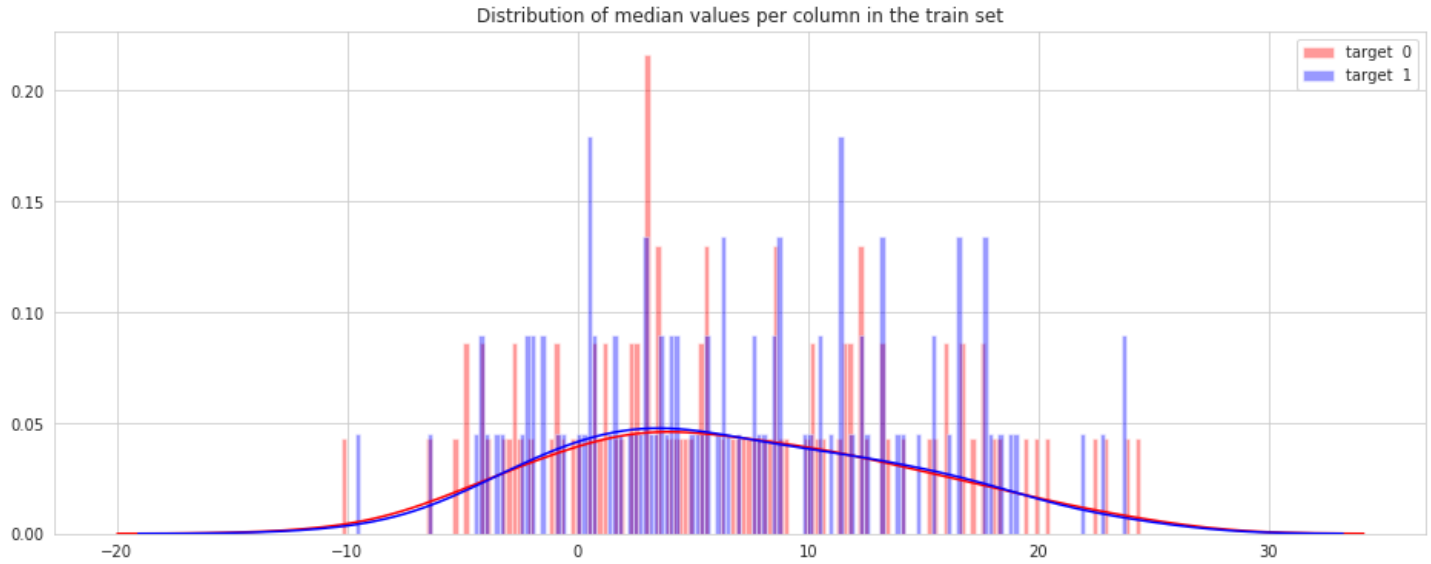
Let us look distribution of kurtosis values:





Let us look distribution of median values:





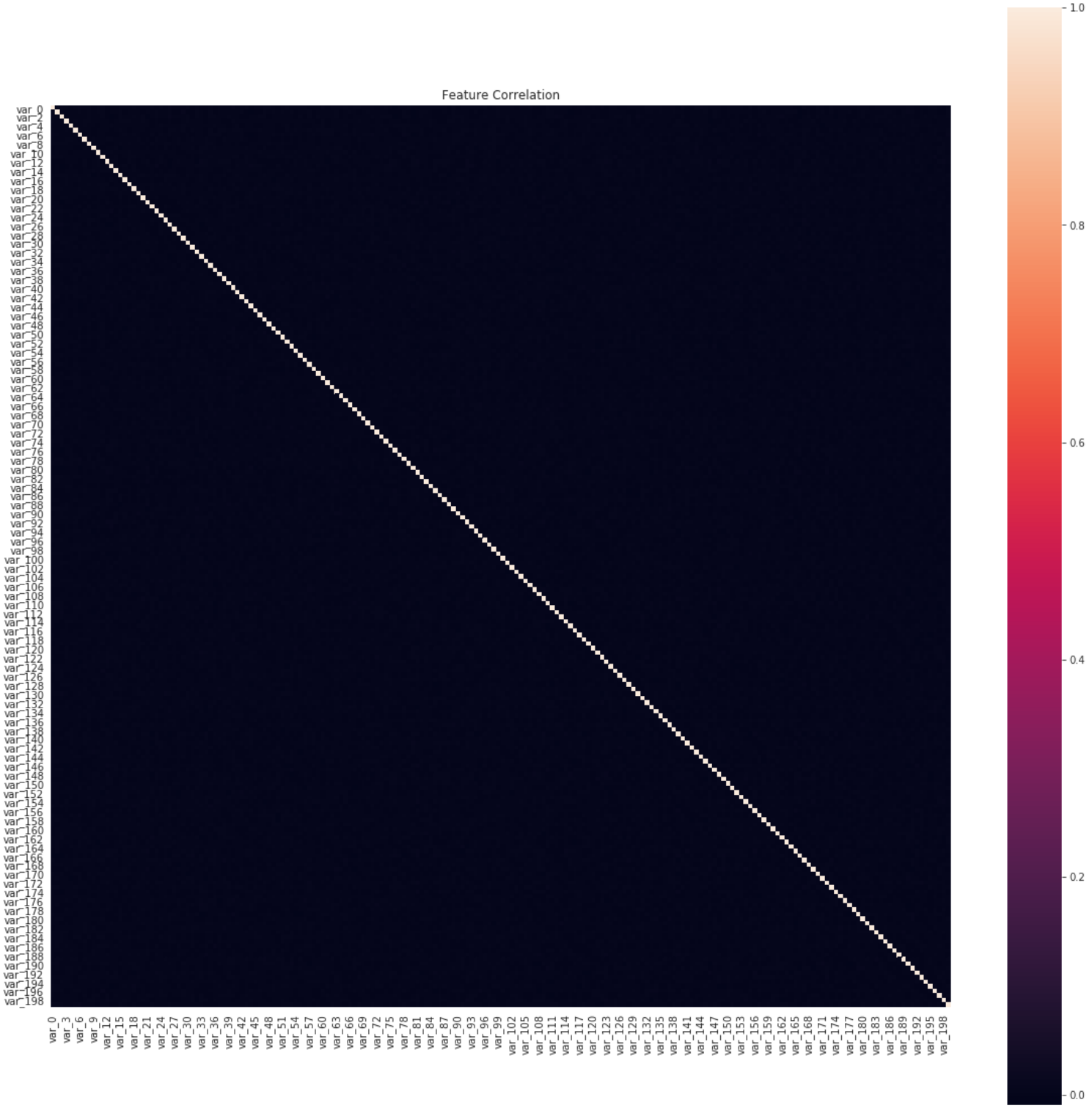
Let us look distribution of sum values:

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:

```
# 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())
```

```
Maximum corr within all variables correlations : 0.009713658349534146
```



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

```
# 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)
```

```
Train.shape: (200000, 202)
df_in.shape: (157999, 202)
df_out.shape: (42001, 202)
```

```
print("df_in.target:\n", df_in['target'].value_counts())
print("df_out.target:\n", df_out['target'].value_counts())
```

```
df_in.target:
 0    157999
Name: target, dtype: int64
df_out.target:
 0    21903
1    20098
Name: target, dtype: int64
```

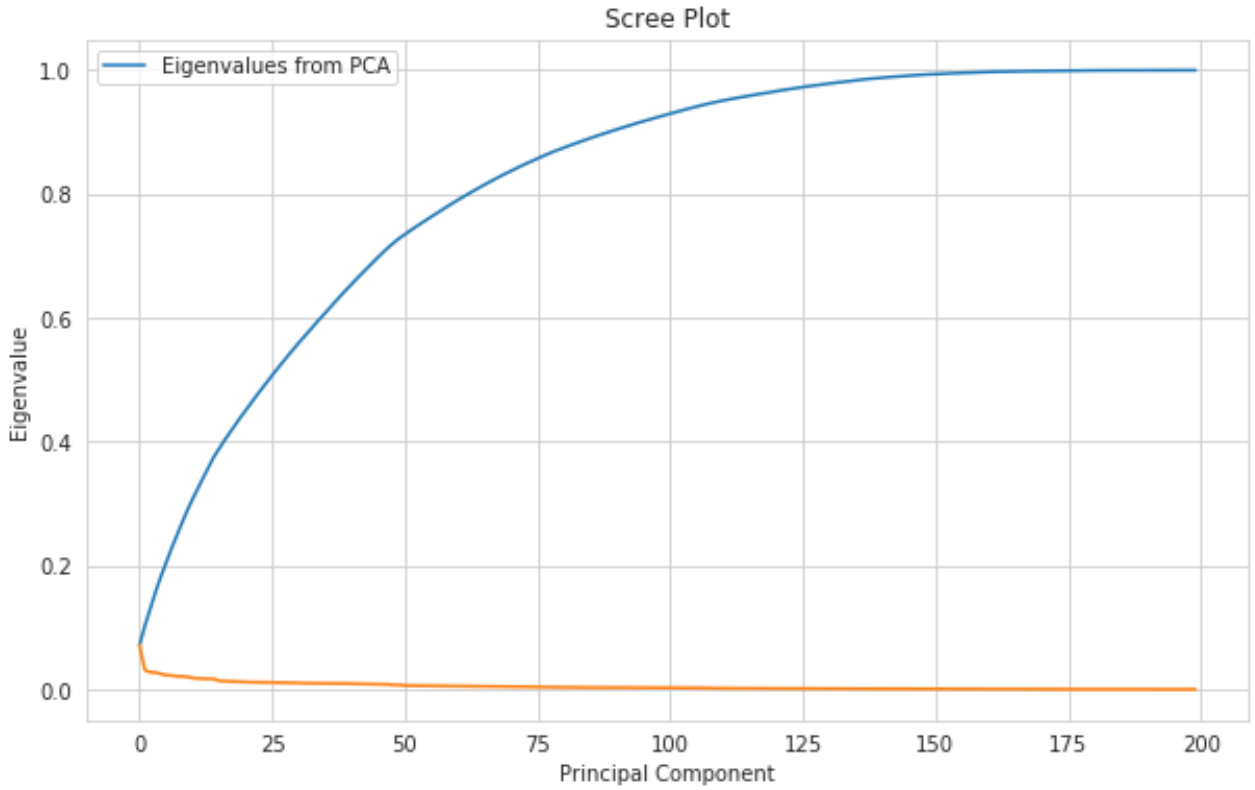
Obeservation:

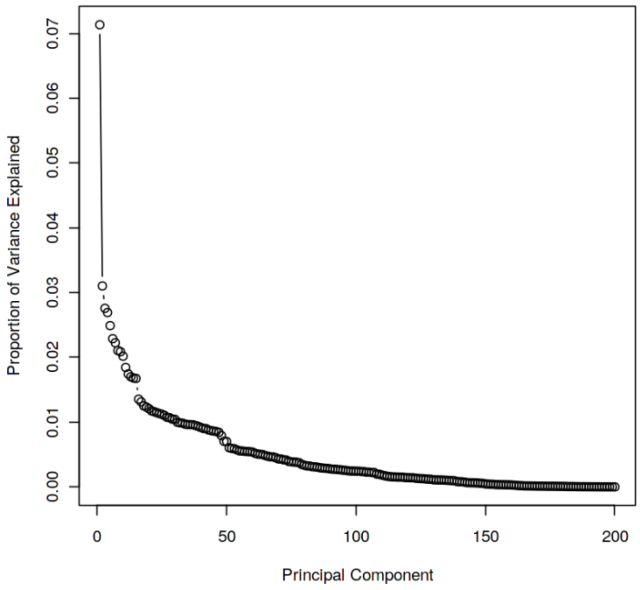
1. After separating outliers and inliers with IQR method we found that all the target variables with label as one are outliers.
2. Outliers present in our data, are meaningful and thus can't be removed.

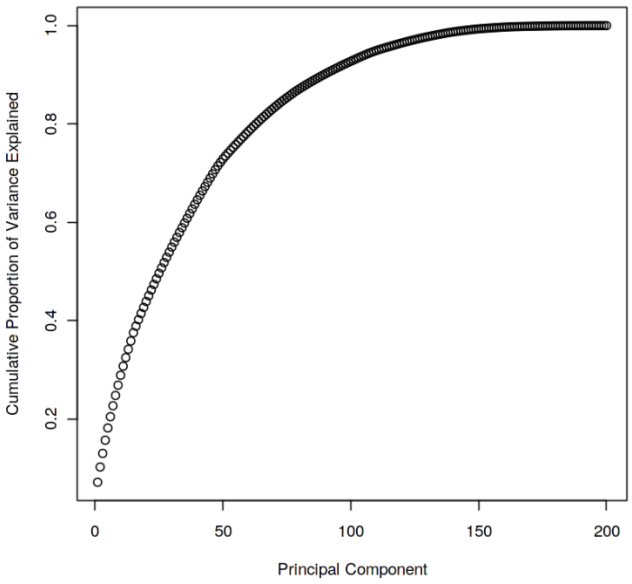
2.2.2 Principal component analysis (PCA)

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.

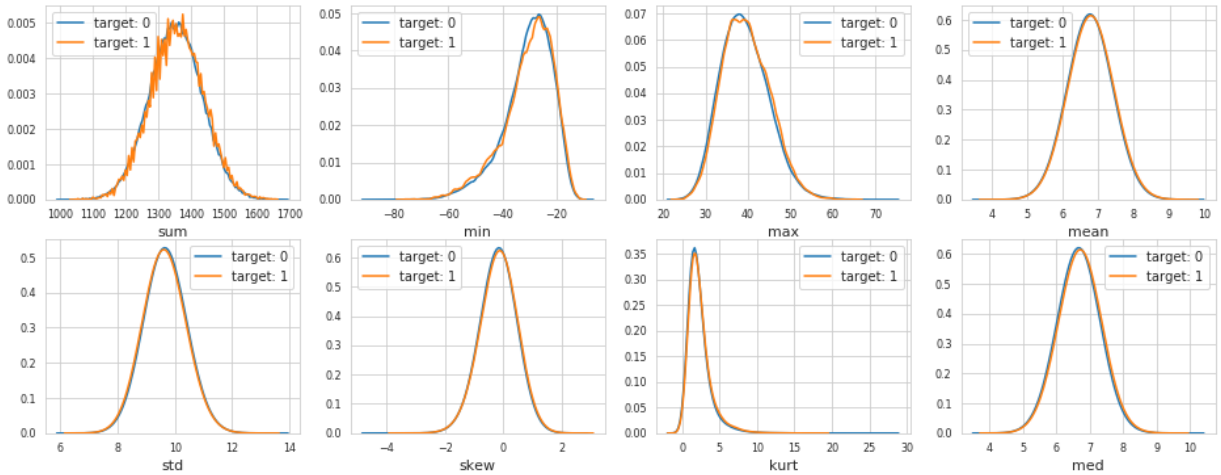
```
aleast_imp = ['var_187', 'var_113', 'var_7', 'var_126', 'var_189', 'var_62',
       'var_117', 'var_45', 'var_182', 'var_96', 'var_199', 'var_19', 'var_68',
       'var_77', 'var_3', 'var_25', 'var_14', 'var_41', 'var_73', 'var_30',
       'var_64', 'var_185', 'var_29', 'var_129', 'var_171', 'var_140']
```

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.

```
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)
```

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

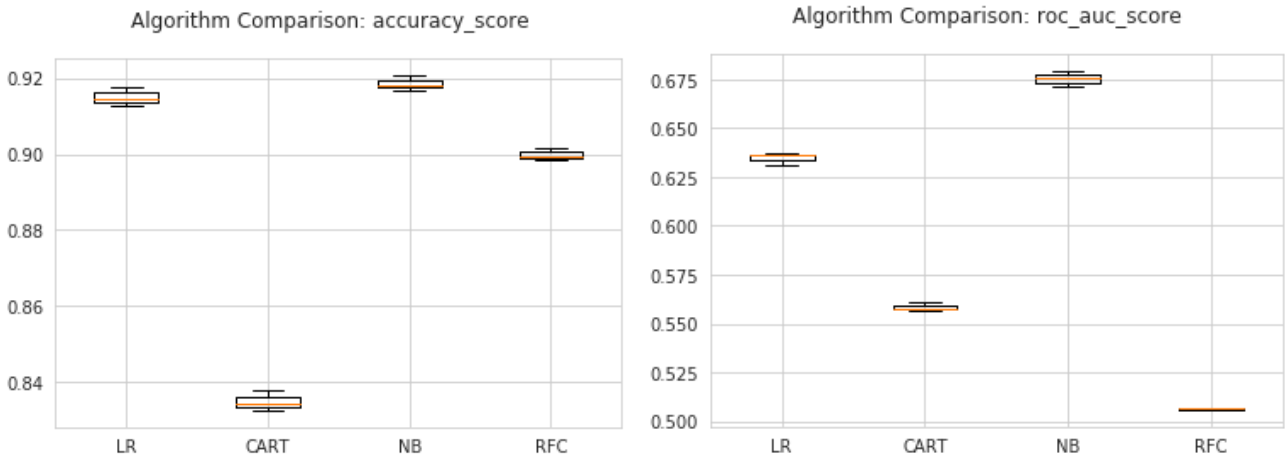
If the dependent variable is Nominal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio, the normal method is to do a Regression analysis, or classification after binning.

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

```
AUC curve (Prob and Value) without StandardScale
LR  AUC prob:  0.865215738698051
LR  AUC value:  0.6348121385191046
LR  f1 score:  0.4033077377436503
CART  AUC prob:  0.5558321453964273
CART  AUC value:  0.5558321453964273
CART  f1 score:  0.2009285943003522
NB  AUC prob:  0.8834591938006527
NB  AUC value:  0.672206132373744
NB  f1 score:  0.47192699946323136
RFC  AUC prob:  0.6921470383281065
RFC  AUC value:  0.506659272816936
RFC  f1 score:  0.028173575129533678
```



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

A. 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

Baseline run with 4 classifier with applying Standardization, StratifiedKFlod and SMOTE oversampling on data.

```
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('flo
at32')


#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)
```

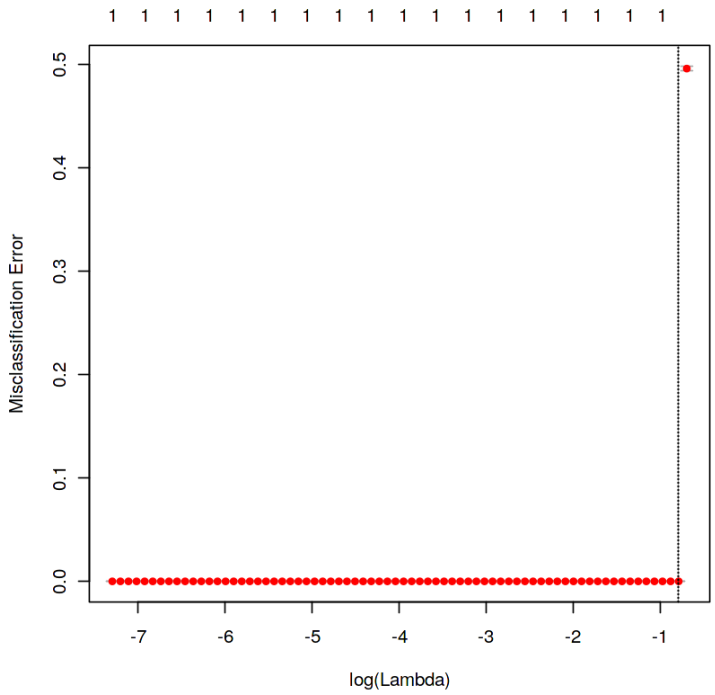
ROSE (Random Over Sampling Examples) package helps us to generate artificial data based on sampling methods and smoothed bootstrap approach. This package has well defined accuracy functions to do the tasks quickly.

```
library(ROSE)
#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)
```

B. LogisticRegression + ROSE

```
#Logistic regression model
library(glmnet)
lr_rose  = glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family ="binomial")
summary(lr_rose)
```

```
#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),family = "binomial", type.measure = "class")
#cv_rose

#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)
```




```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 20152     0
         1     0 19848

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.5038
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5038
         Detection Rate : 0.5038
   Detection Prevalence : 0.5038
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

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:

```
#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,
          }
```

```
num_rounds=10000
lgbm1= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],verbose_eval=1000,early_stopping_rounds = 5000)
```

R:

```
# Implementing LightGBM with raw train data
library(lightgbm)
#Convert data frame to matrix
set.seed(5432)
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)
```

2.2.4.2 SMOTE LightGBM../ ROSE

Python:

```
#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=10000
lgbm3= lgb.train(params,lgb_train2,num_rounds,valid_sets=[lgb_train2,lgb_valid2],feval=lgb_f1_score,verbose_eval=1000,early_stopping_rounds = 5000)
```

R:

```
# 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)
```

CHAPTER 3

Conclusion

3.1 Model Evaluation

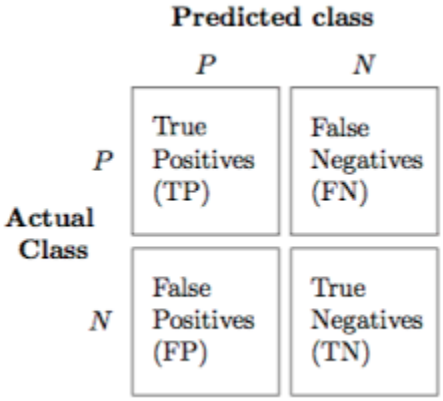
Now, we have 5 models for predicting the target variable, but we need to decide which model better for this project. There are many metrics used for model evaluation. Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

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:

i. True positive rate (or sensitivity)}: tpr = tp / (tp + fn)
ii. False positive rate: fpr = fp / (fp + tn)
iii. 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**

```
AUC curve (Prob and Value) without StandardScale
LR  AUC prob:  0.865215738698051
LR  AUC value:  0.6348121385191046
LR  f1 score:  0.4033077377436503
CART  AUC prob:  0.5558321453964273
CART  AUC value:  0.5558321453964273
CART  f1 score:  0.2009285943003522
NB  AUC prob:  0.8834591938006527
NB  AUC value:  0.672206132373744
NB  f1 score:  0.47192699946323136
RFC  AUC prob:  0.6921470383281065
RFC  AUC value:  0.506659272816936
RFC  f1 score:  0.028173575129533678
```

```
LR :
 [[53258   713]
 [ 4348  1681]]
LR :
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     53971
           1       0.70      0.28      0.40      6029

    accuracy                           0.92     60000
   macro avg       0.81      0.63      0.68     60000
weighted avg       0.90      0.92      0.90     60000
```

```
CART :
 [[48838  5133]
 [ 4756  1273]]
CART :
              precision    recall  f1-score   support

           0       0.91      0.90      0.91     53971
           1       0.20      0.21      0.20      6029

    accuracy                           0.84     60000
   macro avg       0.55      0.56      0.56     60000
weighted avg       0.84      0.84      0.84     60000
```

```
NB :
 [[52868  1103]
 [ 3857  2172]]
NB :
              precision    recall  f1-score   support

           0       0.93      0.98      0.96     53971
           1       0.66      0.36      0.47      6029

    accuracy                           0.92     60000
   macro avg       0.80      0.67      0.71     60000
weighted avg       0.90      0.92      0.91     60000
```

```
RFC :
 [[53899    72]
 [ 5931    98]]
RFC :
              precision    recall  f1-score   support

           0       0.90      1.00      0.95     53971
           1       0.58      0.02      0.03      6029

    accuracy                           0.90     60000
   macro avg       0.74      0.51      0.49     60000
weighted avg       0.87      0.90      0.86     60000
```

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

```
AUC curve (Prob and Value) with Standardization and SMOTE oversampling
LR  AUC prob:  0.8738760601668335
LR  AUC value:  0.7942329071706504
LR  f1 score:  0.795463649800395
CART  AUC prob:  0.6222623679822123
CART  AUC value:  0.6222623679822123
CART  f1 score:  0.5668759361356321
NB  AUC prob:  0.9195598490421922
NB  AUC value:  0.8602279043913285
NB  f1 score:  0.8495977509944075
RFC  AUC prob:  0.7948980970683176
RFC  AUC value:  0.6874374652584769
RFC  f1 score:  0.5779954219670531
```

```
LR :
 [[28360  7620]
 [ 7187 28793]]
LR :
              precision    recall  f1-score   support

           0       0.80      0.79      0.79     35980
           1       0.79      0.80      0.80     35980

    accuracy                           0.79     71960
   macro avg       0.79      0.79      0.79     71960
weighted avg       0.79      0.79      0.79     71960
```

```
CART :
 [[27087  8893]
 [18320 17660]]
CART :
              precision    recall  f1-score   support

           0       0.60      0.75      0.67     35980
           1       0.67      0.49      0.56     35980

    accuracy                           0.62     71960
   macro avg       0.63      0.62      0.62     71960
weighted avg       0.63      0.62      0.62     71960
```

```
NB :
 [[33494  2486]
 [ 7572 28408]]
NB :
              precision    recall  f1-score   support

           0       0.82      0.93      0.87     35980
           1       0.92      0.79      0.85     35980

    accuracy                           0.86     71960
   macro avg       0.87      0.86      0.86     71960
weighted avg       0.87      0.86      0.86     71960
```

```
RFC :
 [[34082  1898]
 [20925 15055]]
RFC :
              precision    recall  f1-score   support

           0       0.62      0.95      0.75     35980
           1       0.89      0.42      0.57     35980

    accuracy                           0.68     71960
   macro avg       0.75      0.68      0.66     71960
weighted avg       0.75      0.68      0.66     71960
```

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:

RandomForest

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 45578     1
         1    68  5072

               Accuracy : 0.9986
                 95% CI : (0.9983, 0.9989)
    No Information Rate : 0.9
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9925

 Mcnemar's Test P-Value : 1.935e-15

            Sensitivity : 0.9985
            Specificity : 0.9998
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9868
             Prevalence : 0.9000
         Detection Rate : 0.8986
   Detection Prevalence : 0.8987
      Balanced Accuracy : 0.9992

       'Positive' Class : 0
```

DecisionTree

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 45579     0
         1     0  5140

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.8987
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.8987
         Detection Rate : 0.8987
   Detection Prevalence : 0.8987
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

NaiveBayes

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 45579     0
         1     0  5140

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.8987
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.8987
         Detection Rate : 0.8987
   Detection Prevalence : 0.8987
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

LogisticRegression

```
Confusion Matrix and Statistics


       0     1
  0 45579     0
  1     0  5140

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.8987
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.8987
         Detection Rate : 0.8987
   Detection Prevalence : 0.8987
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

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

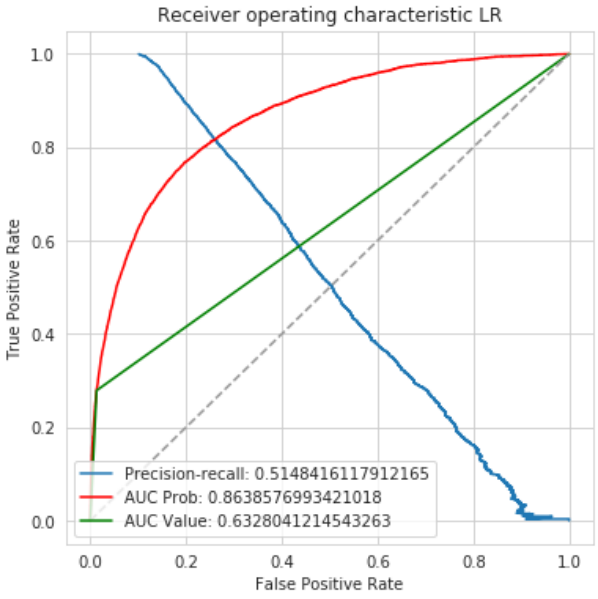
There are two diagnostic tools that help in the interpretation of probabilistic forecast for binary (two-class) classification predictive modelling problems are ROC Curves and Precision-Recall curves. ROC is a probability curve for different classes. ROC tells us how good the model is for distinguishing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

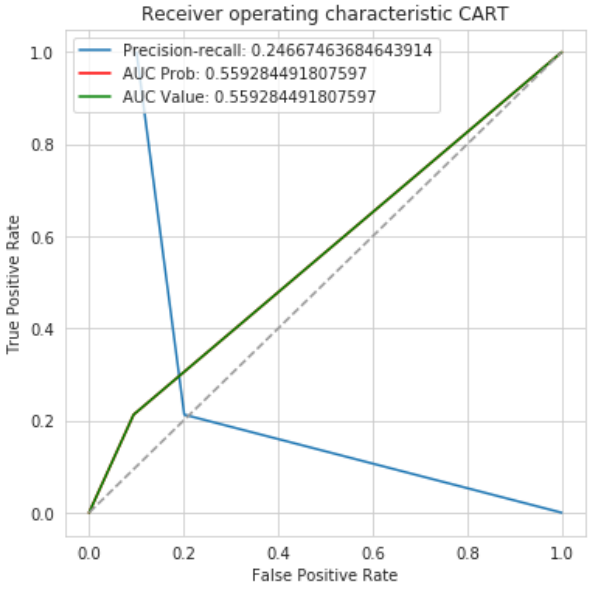
The area covered by the curve is the area between the orange line (ROC) and the axis. This area covered is AUC. The bigger the area covered, the better the machine learning models is at distinguishing the given classes. Ideal value for AUC is 1.

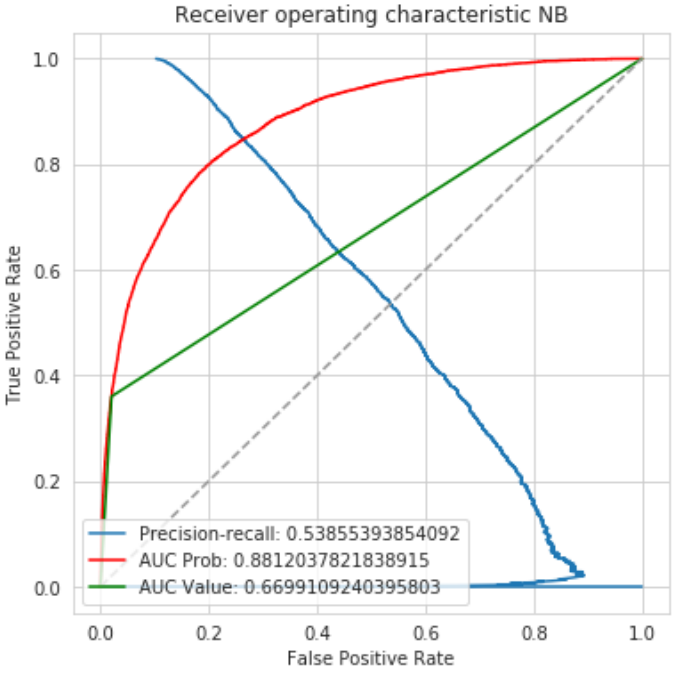
Precision is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class.

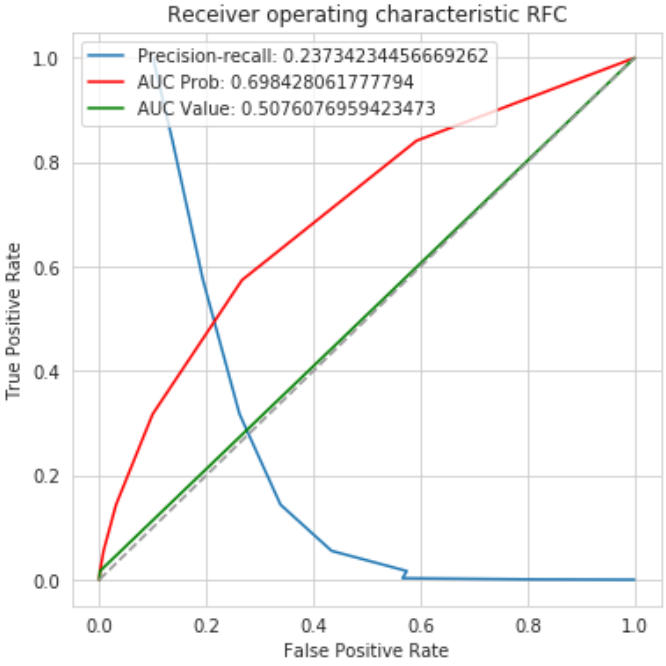
Recall is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as a sensitivity.

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

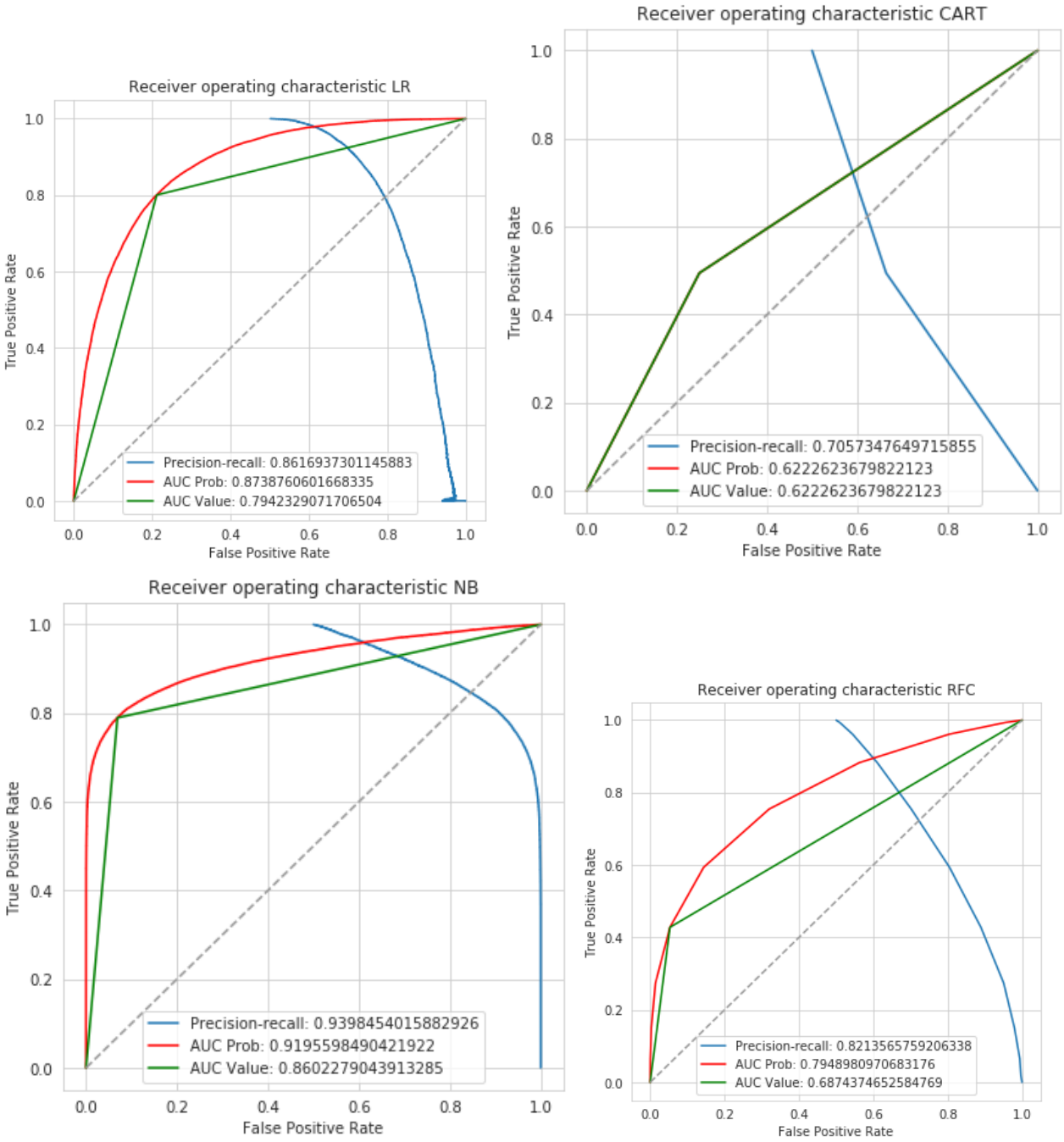








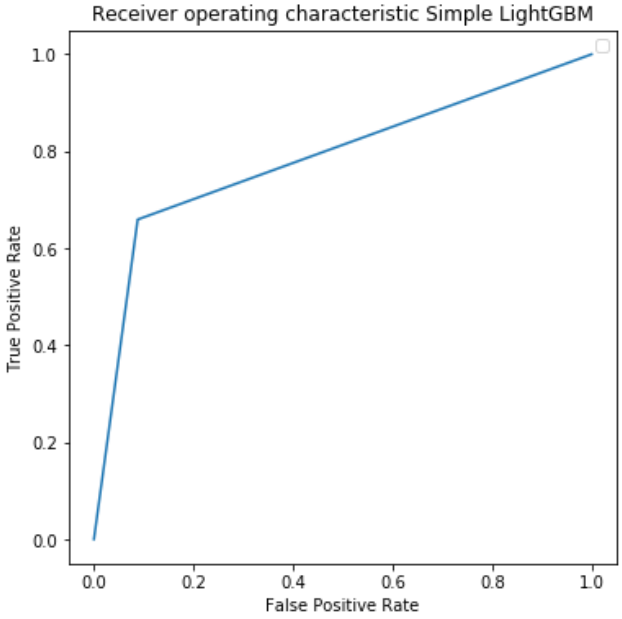
2. **AUC curve (Prob and Value) with Standardization and SMOTE oversampling**



Python LightGBM:

- Simple LightGBM

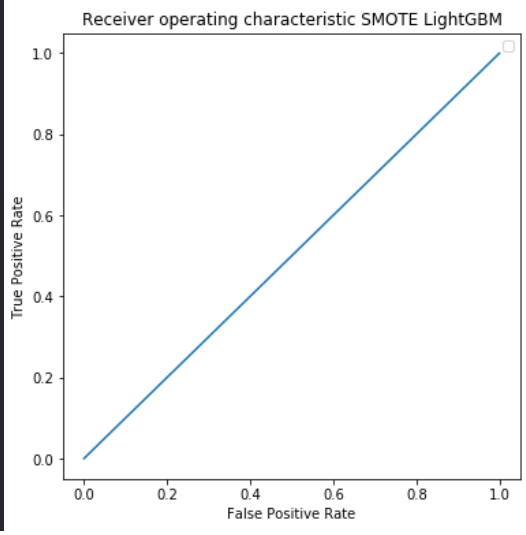
```
Training until validation scores don't improve for 2500 rounds.
[500]   training's auc: 0.932015        training's f1: 0.551476 valid_1's auc: 0.880548 valid_1's f1: 0.480557
[1000]  training's auc: 0.944824        training's f1: 0.578055 valid_1's auc: 0.885342 valid_1's f1: 0.491975
[1500]  training's auc: 0.955212        training's f1: 0.604461 valid_1's auc: 0.887776 valid_1's f1: 0.501054
[2000]  training's auc: 0.963982        training's f1: 0.631032 valid_1's auc: 0.889427 valid_1's f1: 0.508861
[2500]  training's auc: 0.971163        training's f1: 0.656567 valid_1's auc: 0.890489 valid_1's f1: 0.515303
[3000]  training's auc: 0.977155        training's f1: 0.681178 valid_1's auc: 0.891281 valid_1's f1: 0.521284
[3500]  training's auc: 0.982114        training's f1: 0.705639 valid_1's auc: 0.891674 valid_1's f1: 0.525803
[4000]  training's auc: 0.986078        training's f1: 0.730148 valid_1's auc: 0.891892 valid_1's f1: 0.53033
[4500]  training's auc: 0.989255        training's f1: 0.753596 valid_1's auc: 0.892199 valid_1's f1: 0.53579
[5000]  training's auc: 0.991808        training's f1: 0.774696 valid_1's auc: 0.892128 valid_1's f1: 0.53902
Did not meet early stopping. Best iteration is:
[5000]  training's auc: 0.991808        training's f1: 0.774696 valid_1's auc: 0.892128 valid_1's f1: 0.53902
```




```
Confusion matrix: Simple Lightgbm
array([[46470,  7501],
       [ 1615,  4414]])
```

- SMOTE LightGBM:

```
Training until validation scores don't improve for 5000 rounds.
[1000]  training's auc: 0.956035        training's f1: 0.885404 valid_1's auc: 0.934872 valid_
1's f1: 0.85351
[2000]  training's auc: 0.969382        training's f1: 0.906532 valid_1's auc: 0.943806 valid_
1's f1: 0.86739
[3000]  training's auc: 0.977731        training's f1: 0.921405 valid_1's auc: 0.94825  valid_
1's f1: 0.873914
[4000]  training's auc: 0.983564        training's f1: 0.932759 valid_1's auc: 0.950809 valid_
1's f1: 0.876921
[5000]  training's auc: 0.98797 training's f1: 0.94301  valid_1's auc: 0.952365 valid_1's f1:
0.8782
[6000]  training's auc: 0.991283        training's f1: 0.952102 valid_1's auc: 0.95335  valid_
1's f1: 0.879392
[7000]  training's auc: 0.993806        training's f1: 0.960216 valid_1's auc: 0.954164 valid_
1's f1: 0.879942
[8000]  training's auc: 0.995674        training's f1: 0.967558 valid_1's auc: 0.954826 valid_
1's f1: 0.879562
[9000]  training's auc: 0.997011        training's f1: 0.973794 valid_1's auc: 0.955335 valid_
1's f1: 0.879472
[10000] training's auc: 0.997974        training's f1: 0.979005 valid_1's auc: 0.955761 valid_
1's f1: 0.879266
Did not meet early stopping. Best iteration is:
[10000] training's auc: 0.997974        training's f1: 0.979005 valid_1's auc: 0.955761 valid_
1's f1: 0.879266
```

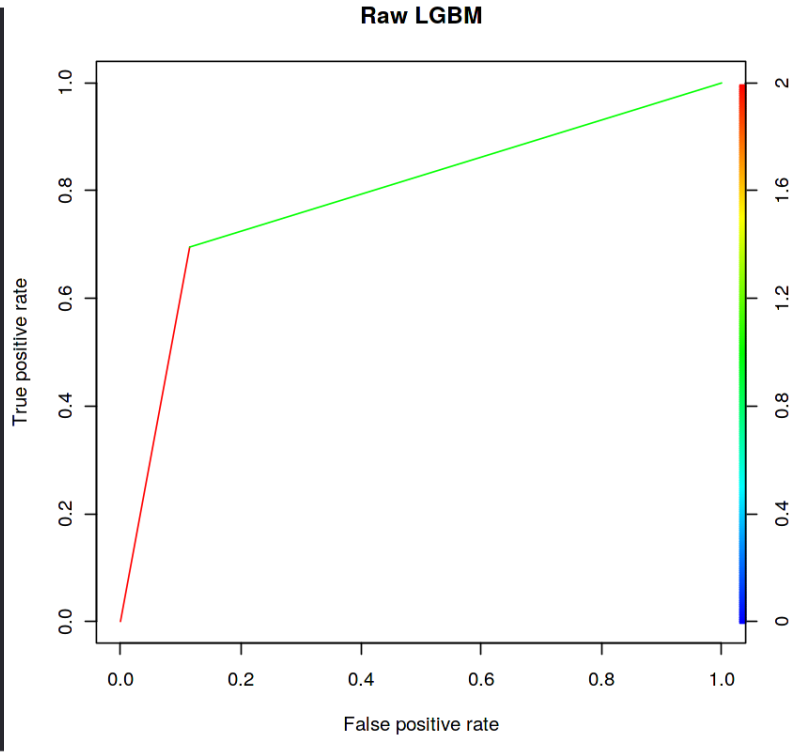


```
Confusion matrix: SMOTE Lightgbm
array([[53971,     0],
       [ 6029,     0]])
```

R LightGBM:

- LightGBM with raw Data

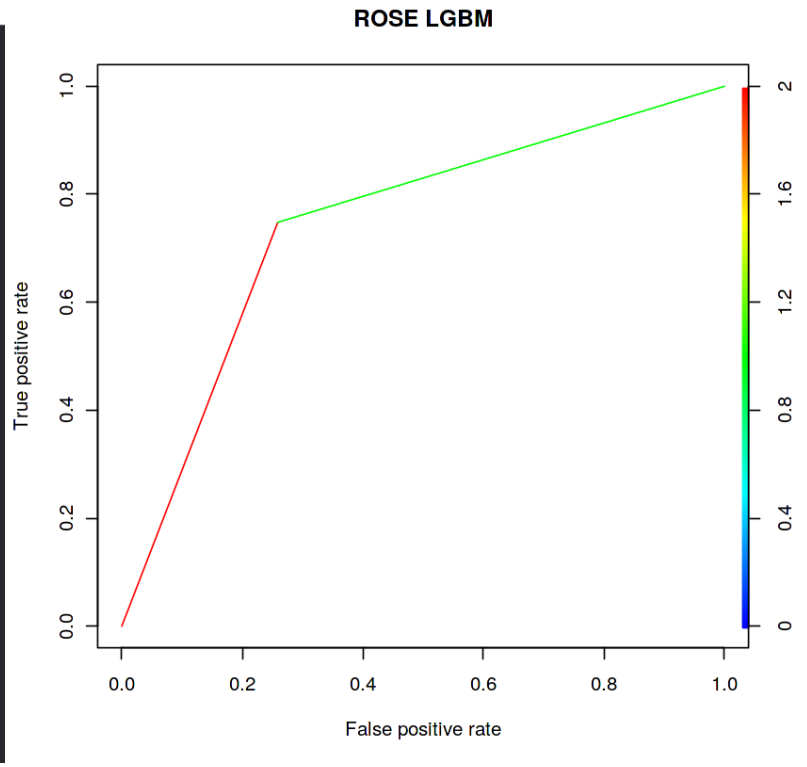
```
Confusion Matrix and Statistics


      0     1
 0 31815  4141
 1  1232  2812

               Accuracy : 0.8657
                 95% CI : (0.8623, 0.869)
    No Information Rate : 0.8262
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4398

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9627
            Specificity : 0.4044
         Pos Pred Value : 0.8848
         Neg Pred Value : 0.6954
             Prevalence : 0.8262
         Detection Rate : 0.7954
   Detection Prevalence : 0.8989
      Balanced Accuracy : 0.6836

       'Positive' Class : 0
```




```
[1]:    val1's auc:0.709226     val2's auc:0.672636
[101]:  val1's auc:0.973234     val2's auc:0.874723
[201]:  val1's auc:0.983516     val2's auc:0.886059
[301]:  val1's auc:0.983398     val2's auc:0.888715
[401]:  val1's auc:0.983054     val2's auc:0.889376
[501]:  val1's auc:0.983054     val2's auc:0.889376
```

- LightGBM with ROSE

```
Confusion Matrix and Statistics


       0     1
  0 14935  5217
  1  5002 14846

               Accuracy : 0.7445
                 95% CI : (0.7402, 0.7488)
    No Information Rate : 0.5016
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.4891

 Mcnemar's Test P-Value : 0.03426

            Sensitivity : 0.7491
            Specificity : 0.7400
         Pos Pred Value : 0.7411
         Neg Pred Value : 0.7480
             Prevalence : 0.4984
         Detection Rate : 0.3734
   Detection Prevalence : 0.5038
      Balanced Accuracy : 0.7445

       'Positive' Class : 0
```




```
[1]:    val1's auc:0.663042     val2's auc:0.637355
[101]:  val1's auc:0.862433     val2's auc:0.801729
[201]:  val1's auc:0.874838     val2's auc:0.814694
[301]:  val1's auc:0.878209     val2's auc:0.819733
[401]:  val1's auc:0.879061     val2's auc:0.82138
[501]:  val1's auc:0.879061     val2's auc:0.82138
[601]:  val1's auc:0.879061     val2's auc:0.82138
```

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.

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.

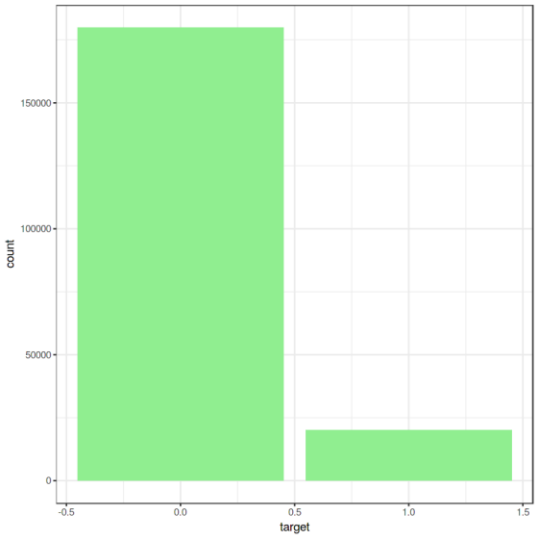
4. Simple Stratified LightGBM model performed well on imbalanced data in Python.

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

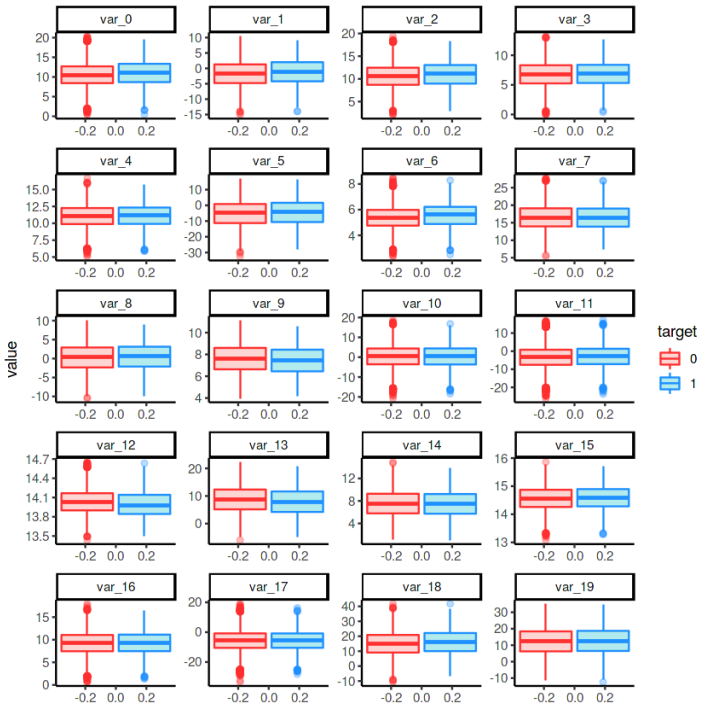
transaction in the future, irrespective of the amount of money transacted.

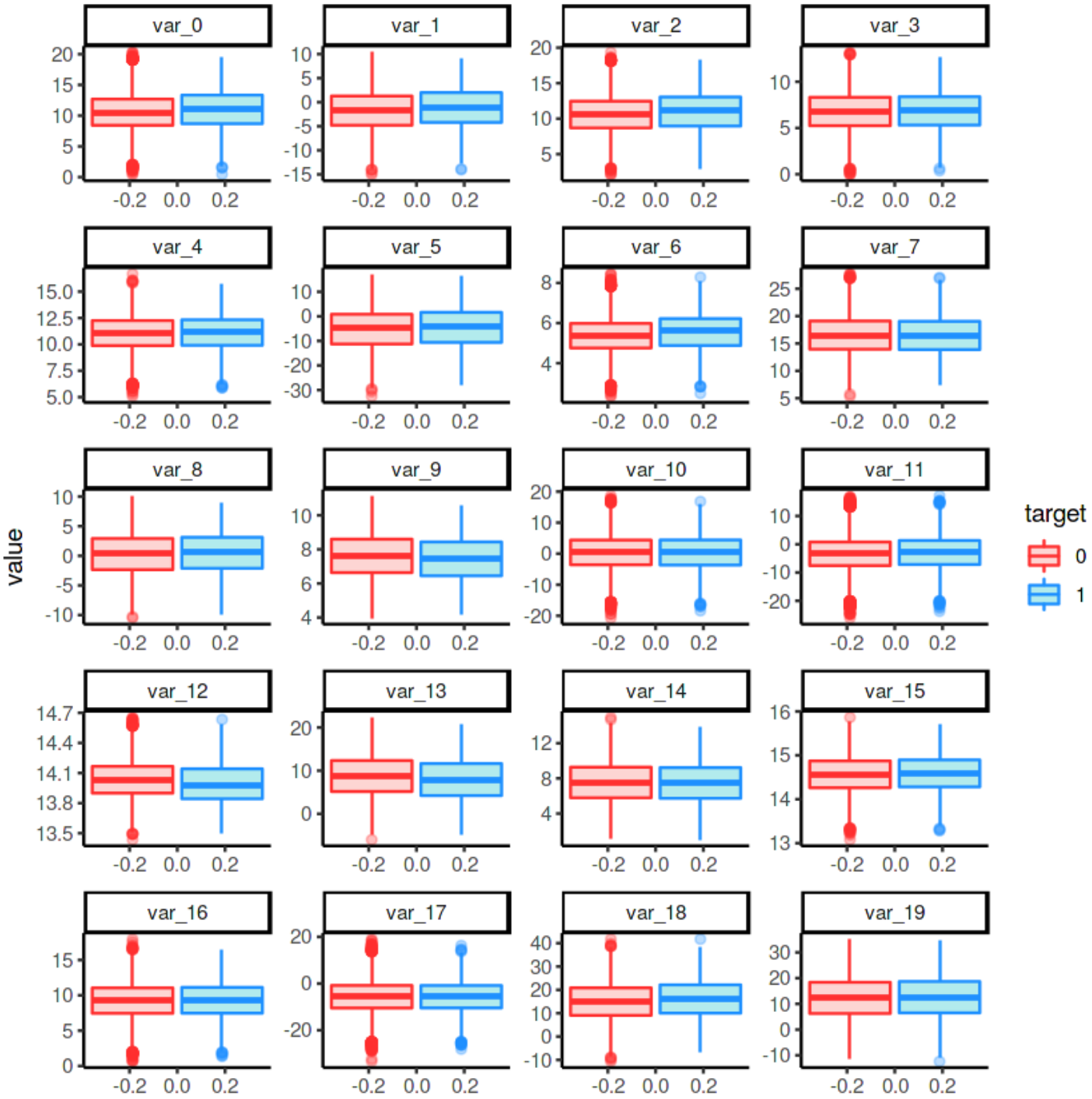
Appendix A - Extra Figures

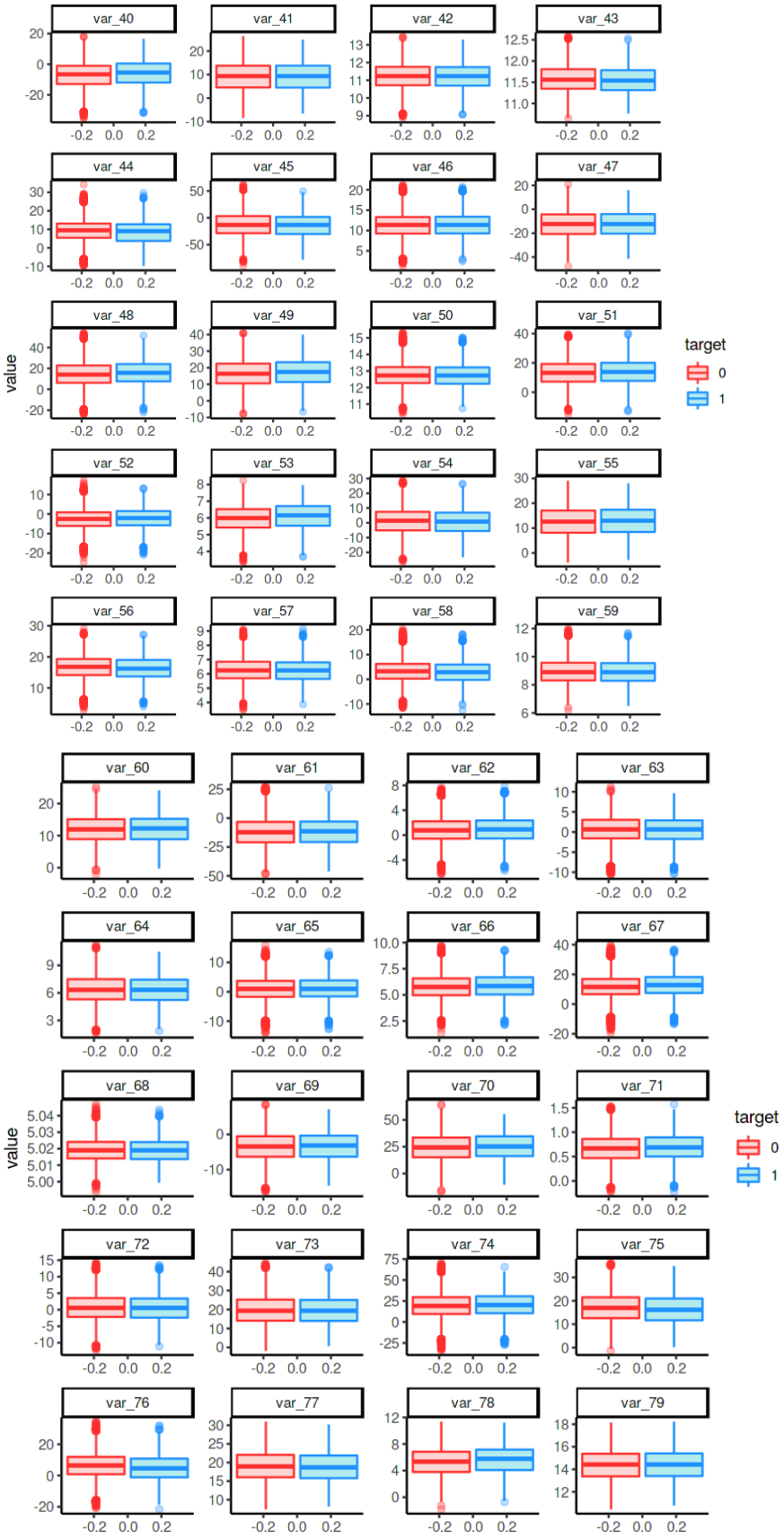
Target classes count

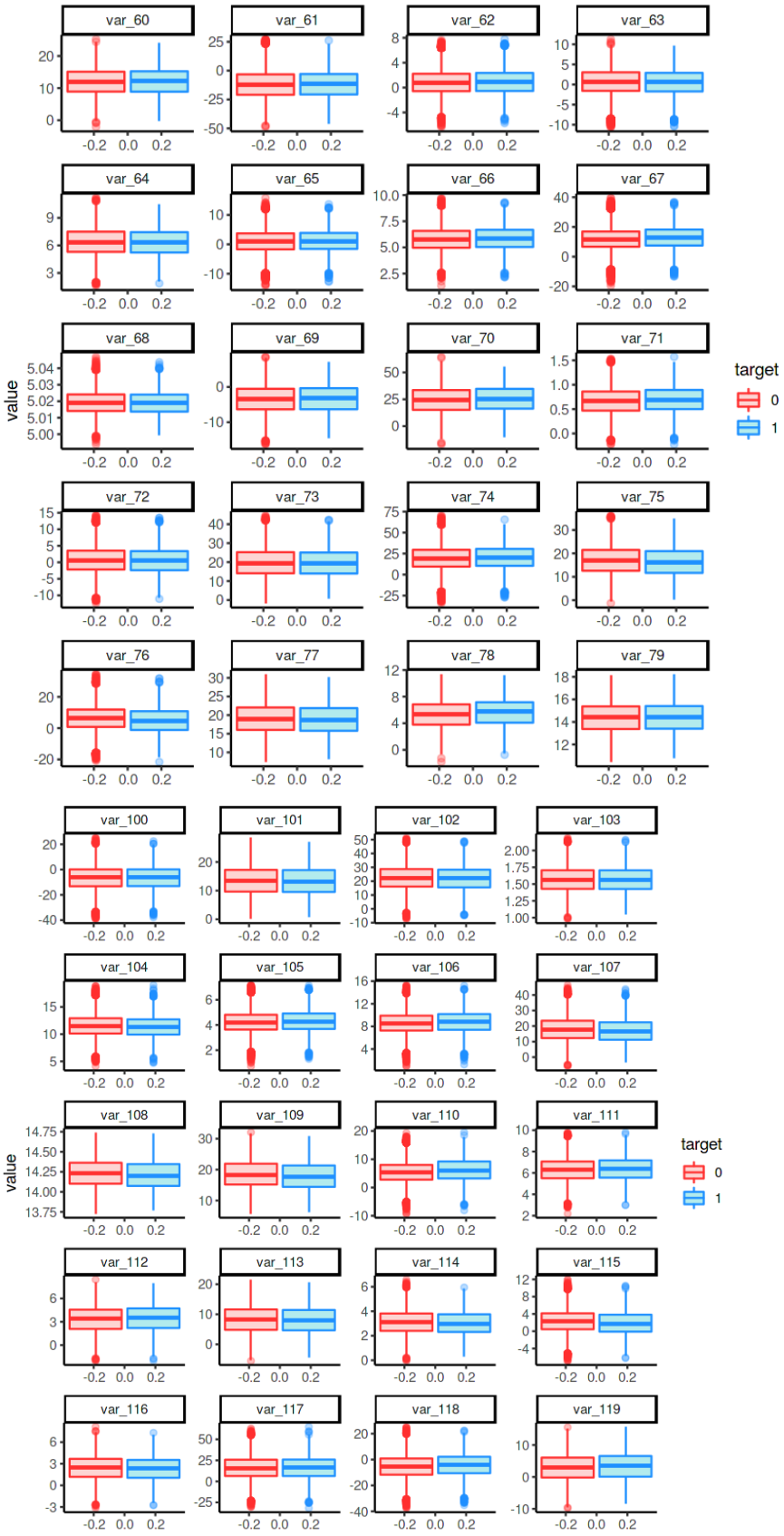


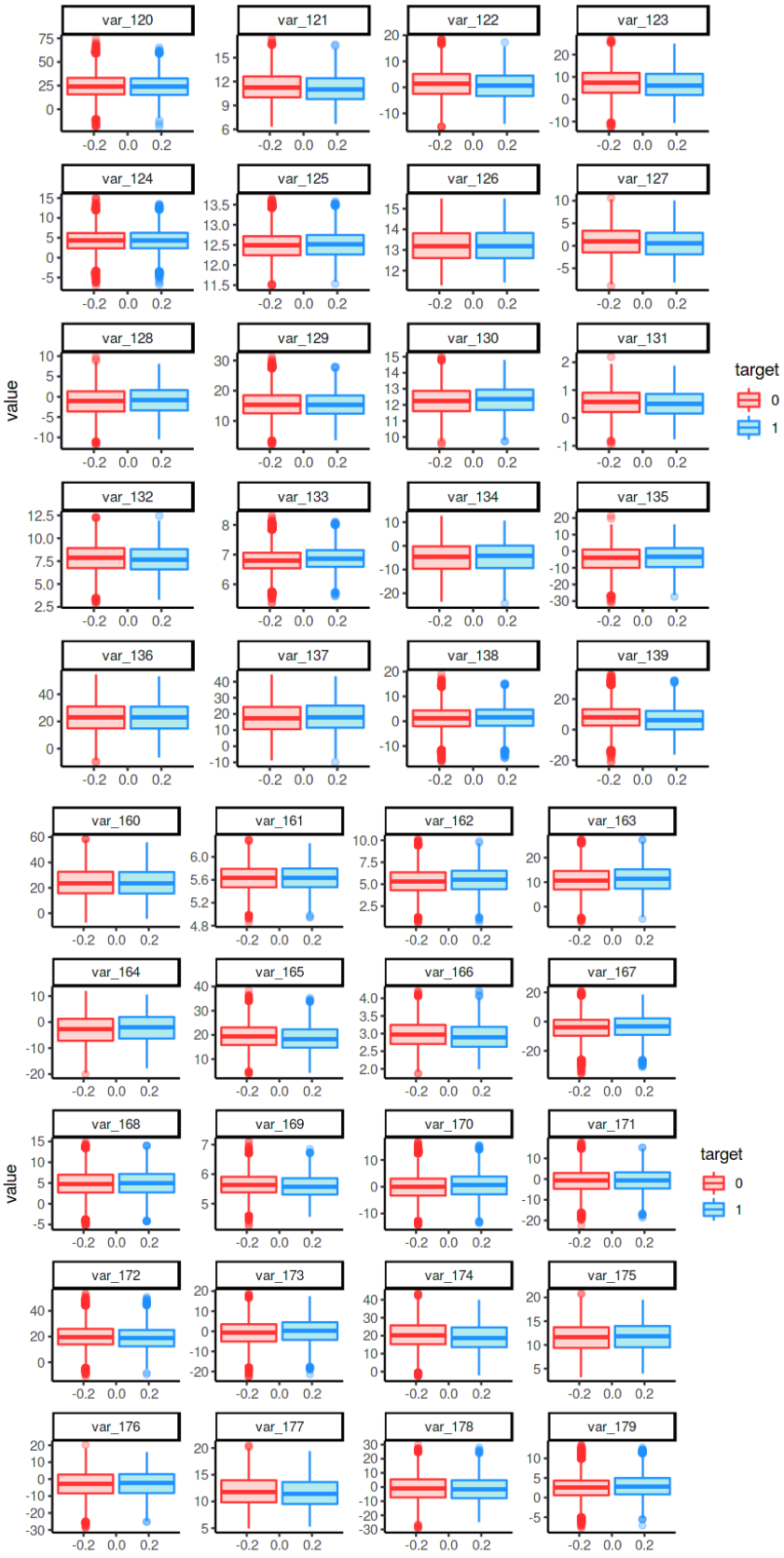
BoxPlot Train data

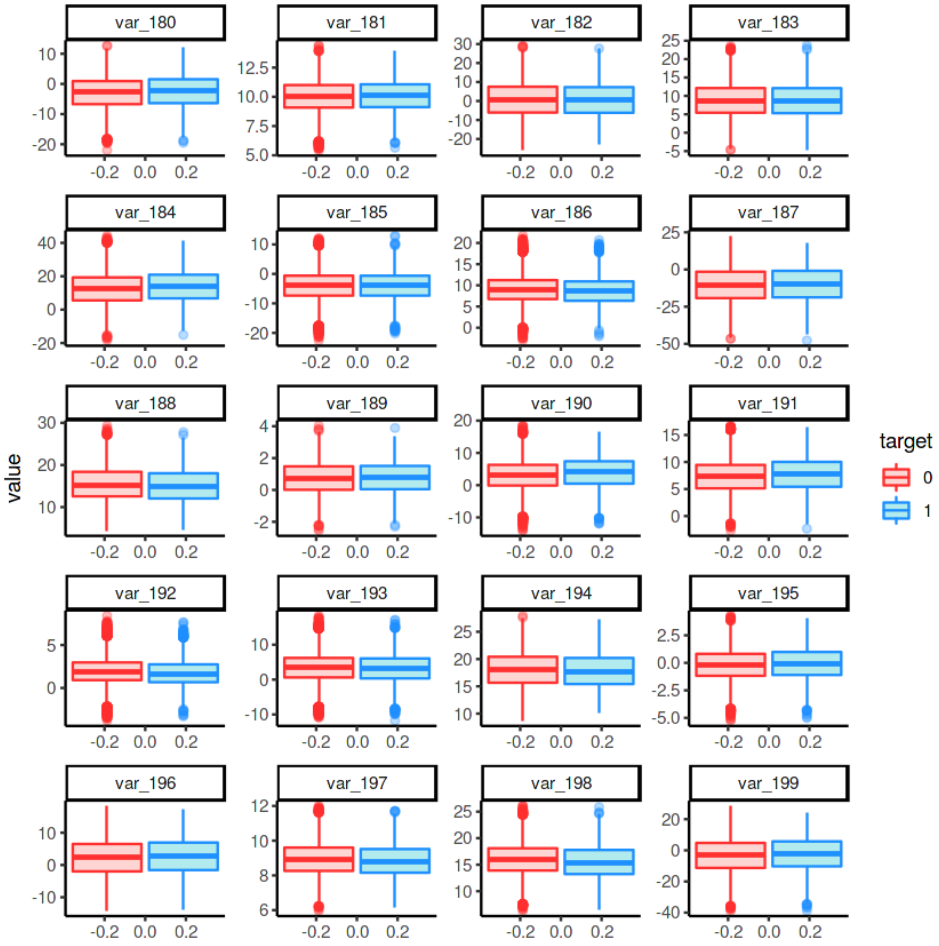



var_0
var_1
var_2
var_3
var_4
var_5
var_6
var_7
var_8
var_9
var_10
var_11
var_12
var_13
var_14
var_15
var_16
var_17
var_18
var_19
value
target
0
1

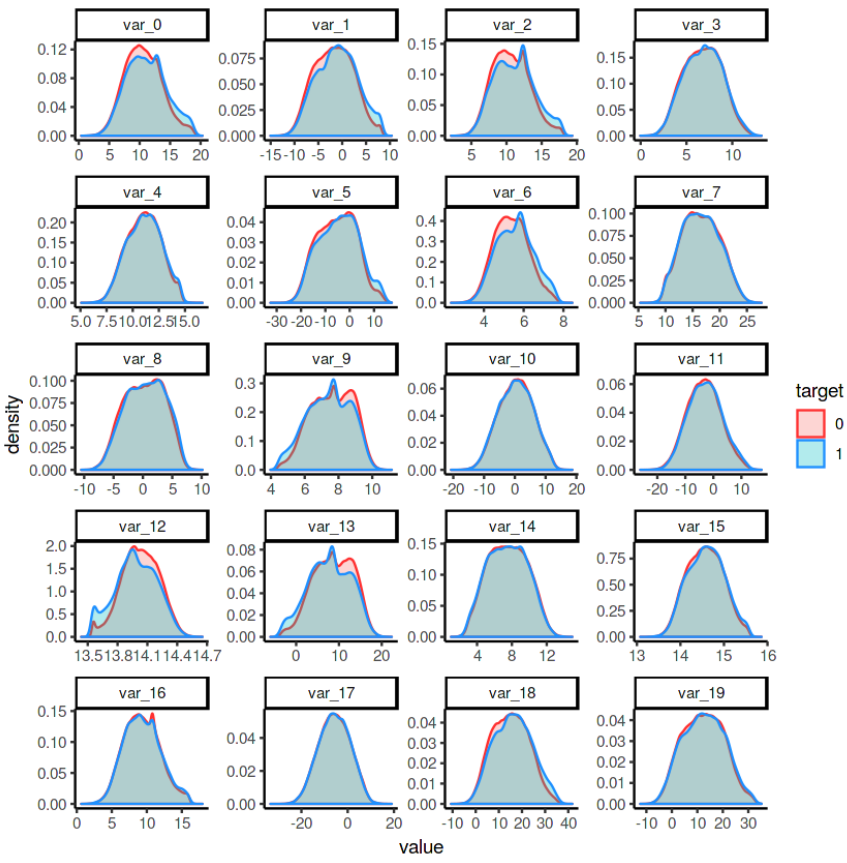

var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59
value
target
0
1
var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79
value
target
0
1

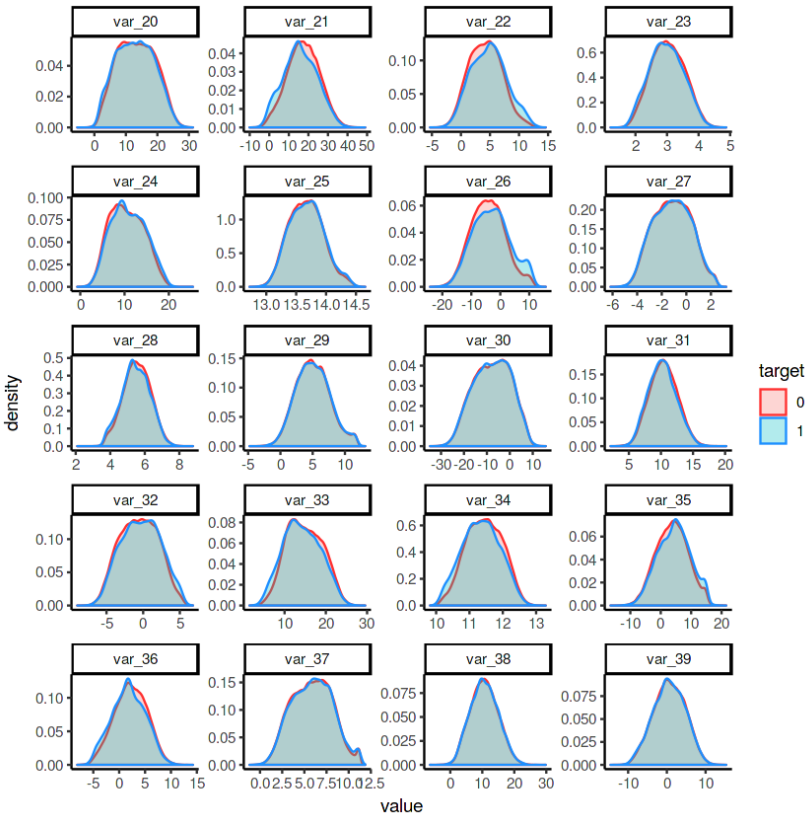

var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79
var_100
var_101
var_102
var_103
var_104
var_105
var_106
var_107
var_108
var_109
var_110
var_111
var_112
var_113
var_114
var_115
var_116
var_117
var_118
var_119
value
target
0
1

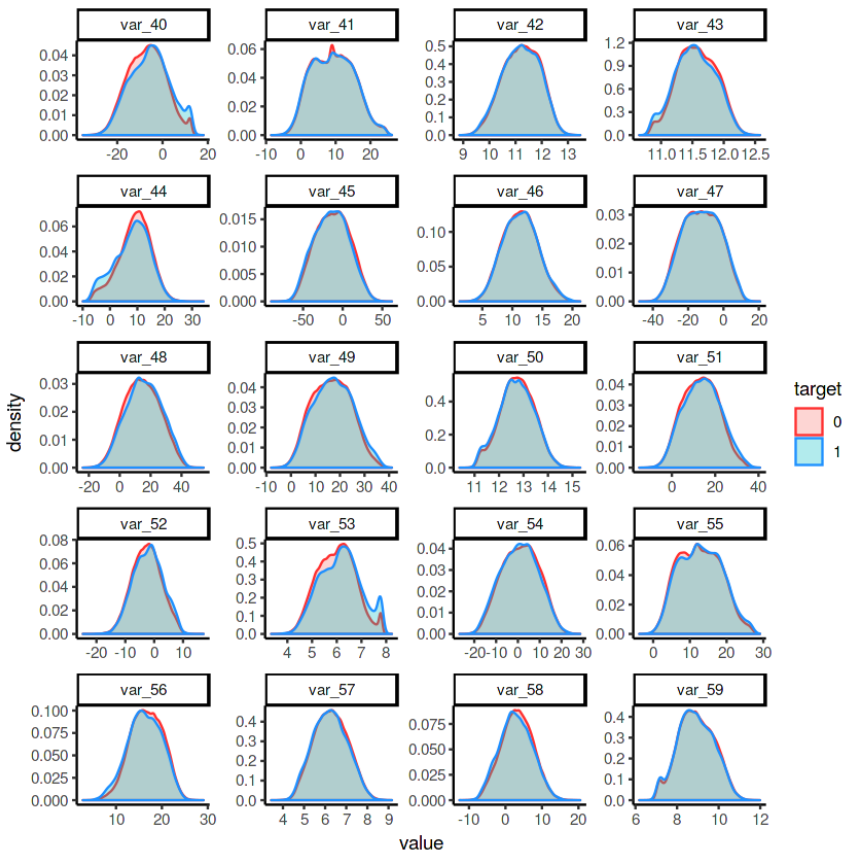

var_120
var_121
var_122
var_123
var_124
var_125
var_126
var_127
var_128
var_129
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139
var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179
value
target
0
1
-0.2
0.0
0.2

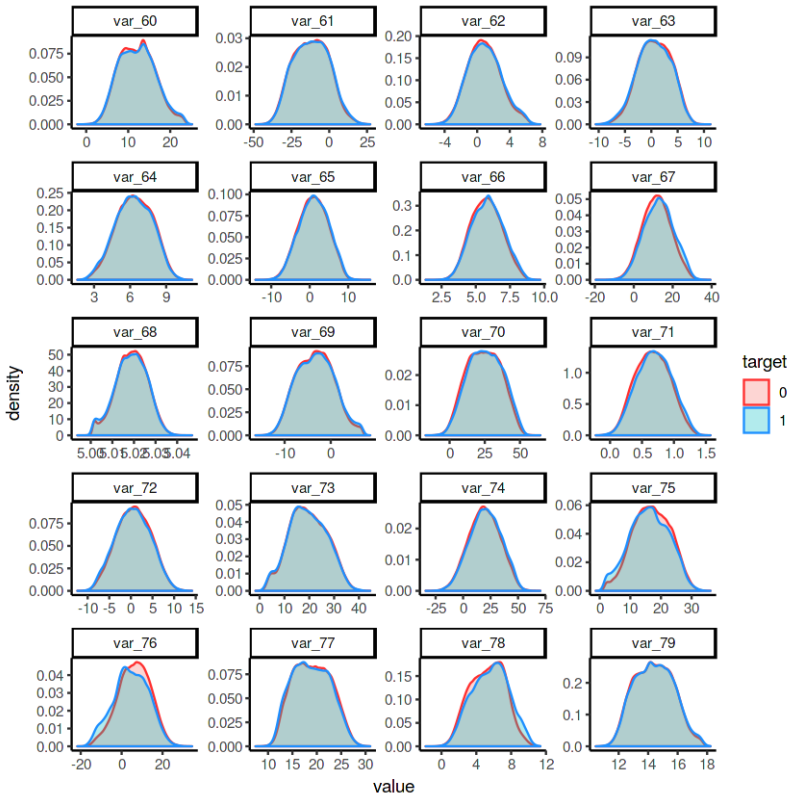


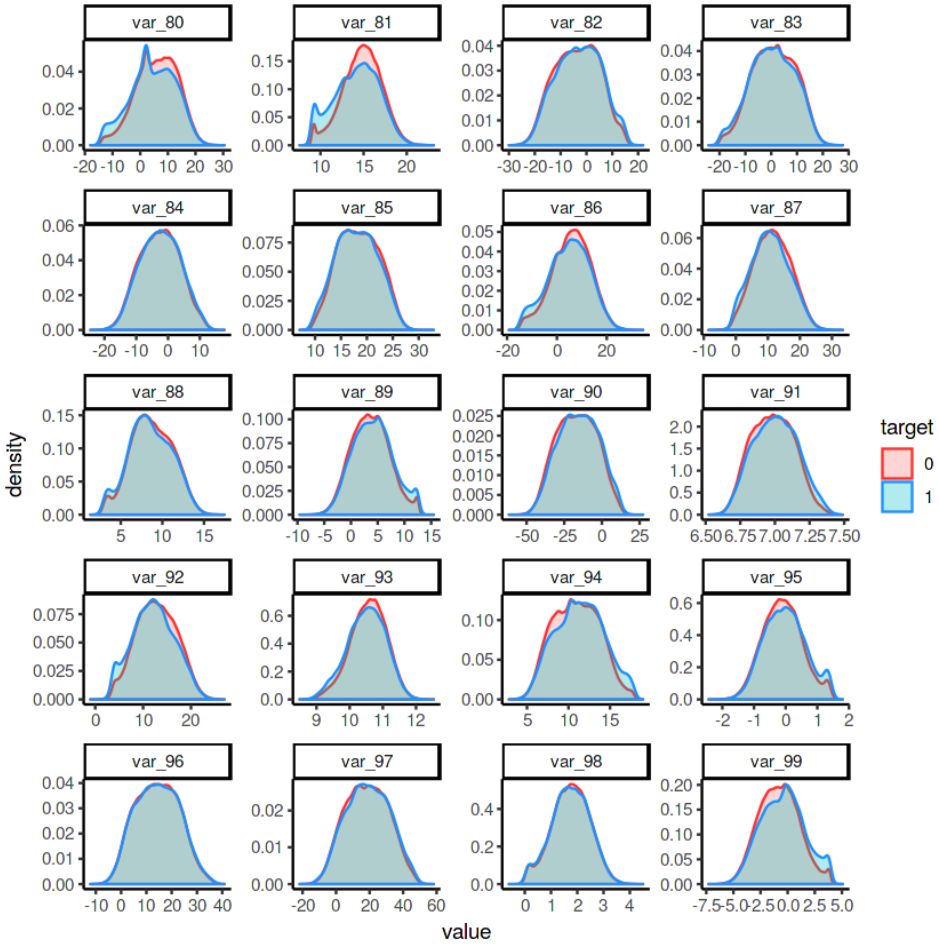
Distribution of train attributes

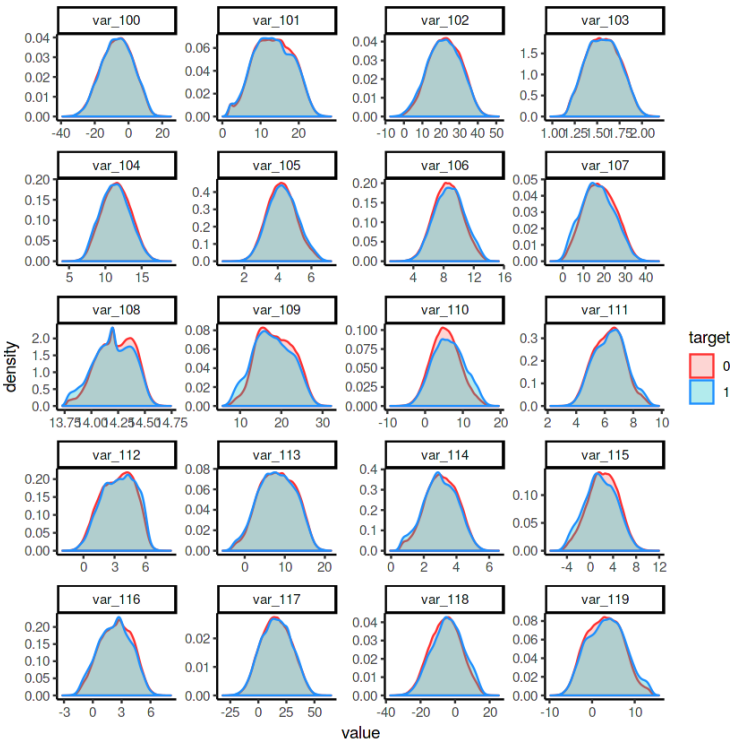

var_0
var_1
var_2
var_3
var_4
var_5
var_6
var_7
var_8
var_9
var_10
var_11
var_12
var_13
var_14
var_15
var_16
var_17
var_18
var_19
density
value
target
0
1

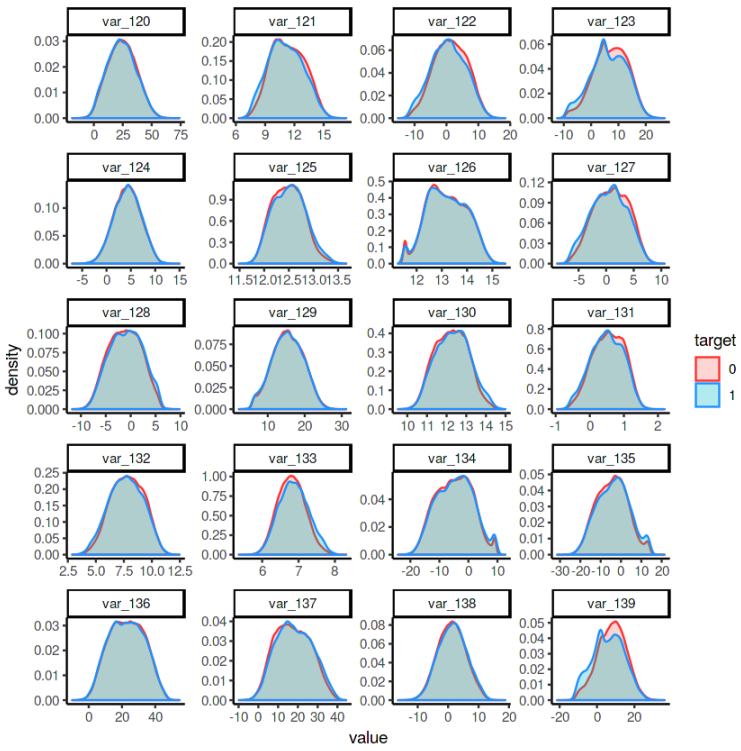

var_20
var_21
var_22
var_23
var_24
var_25
var_26
var_27
var_28
var_29
var_30
var_31
var_32
var_33
var_34
var_35
var_36
var_37
var_38
var_39
density
value
target
0
1

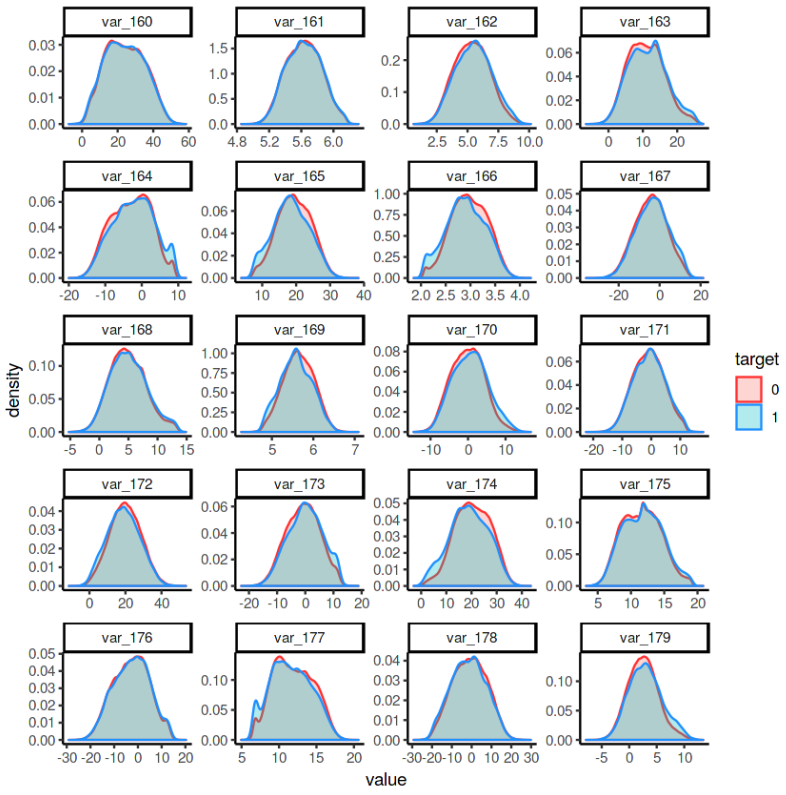

var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59
density
value
target
0
1

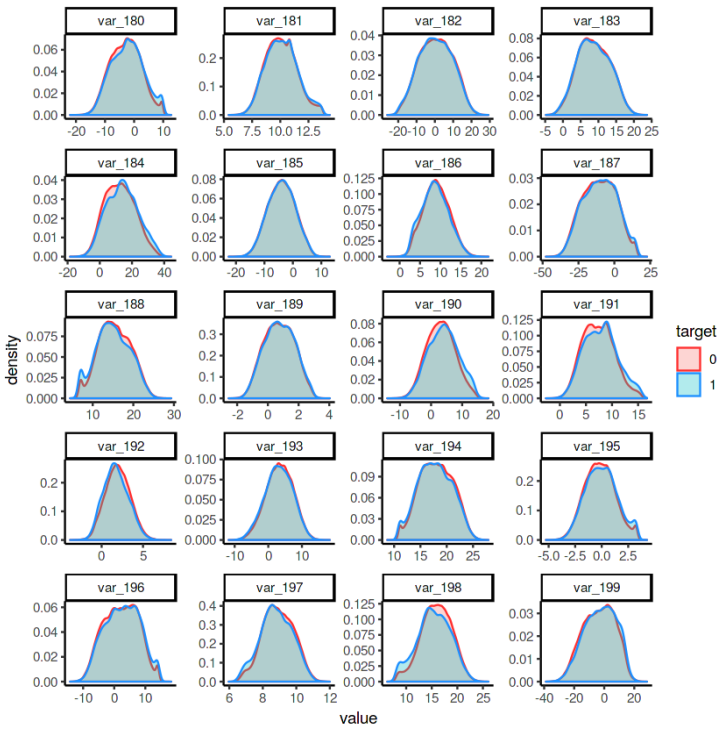

var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79
density
value
target
0
1

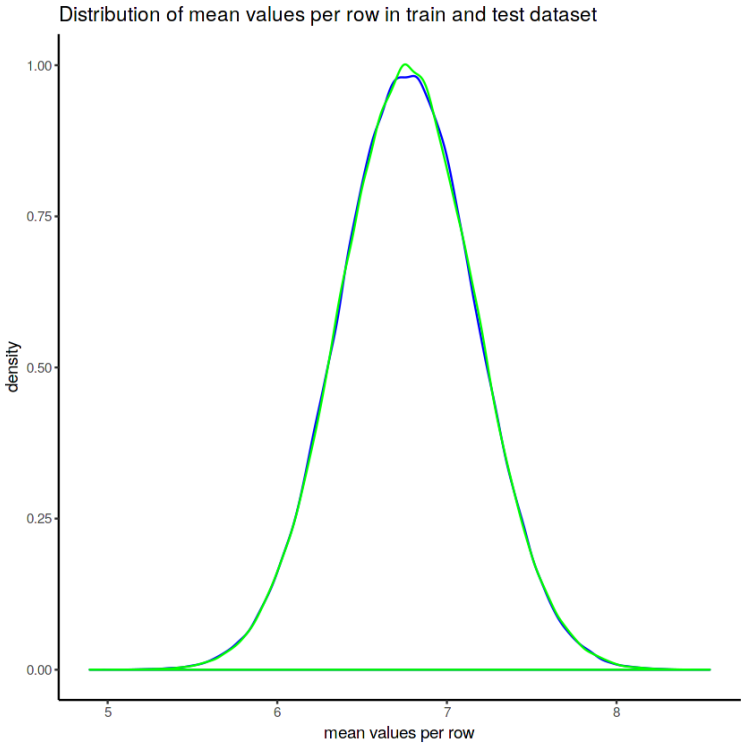

var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99
density
value
target
0
1

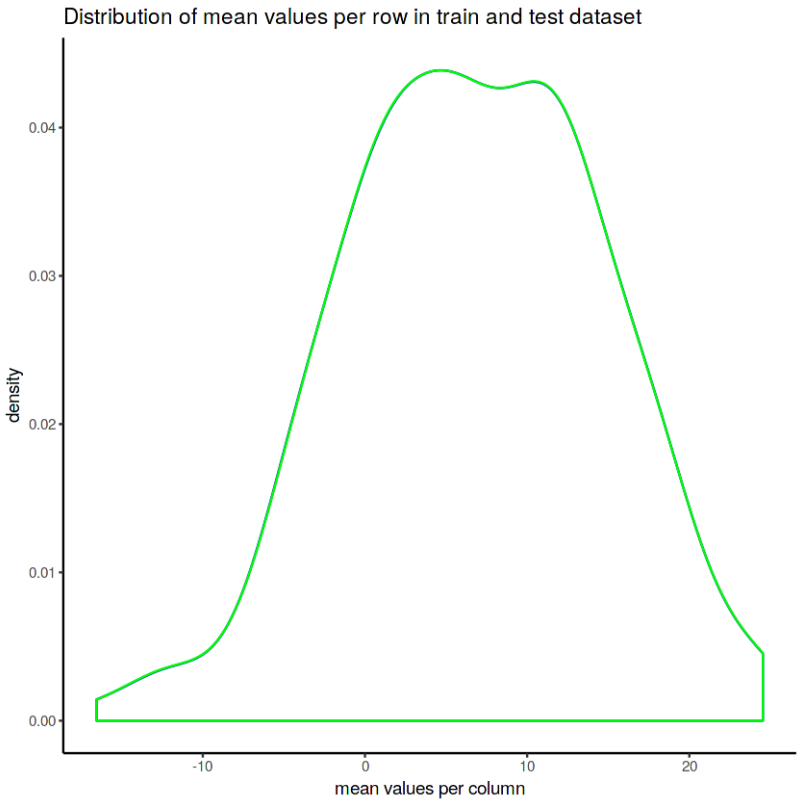

var_100
var_101
var_102
var_103
var_104
var_105
var_106
var_107
var_108
var_109
var_110
var_111
var_112
var_113
var_114
var_115
var_116
var_117
var_118
var_119
density
value
target
0
1

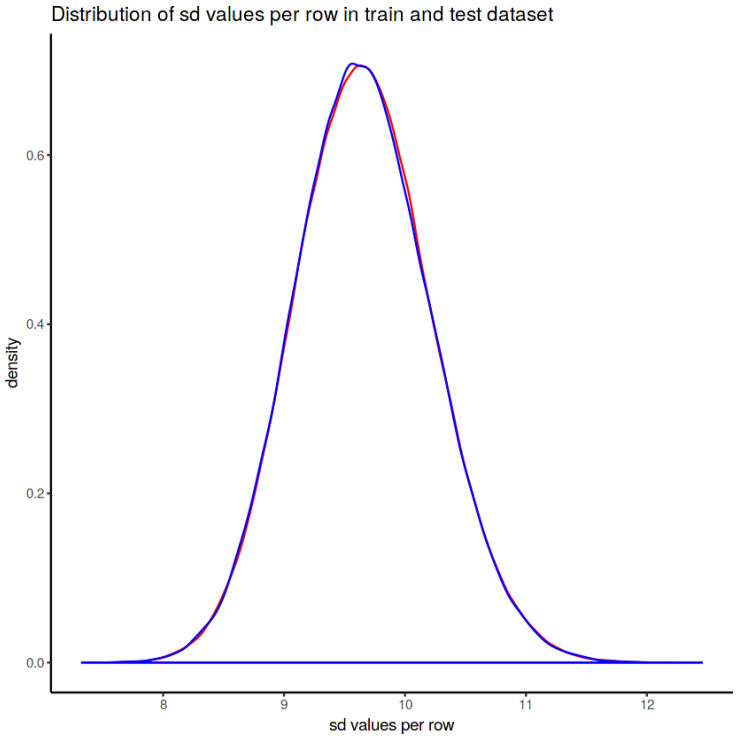

var_120
var_121
var_122
var_123
var_124
var_125
var_126
var_127
var_128
var_129
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139
density
value
target
0
1

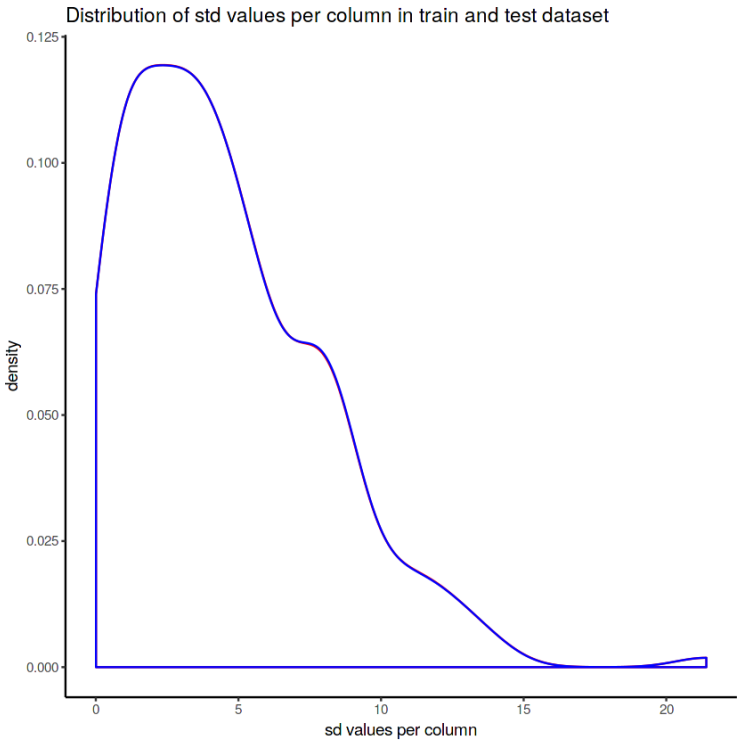

var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179
density
value
target
0
1

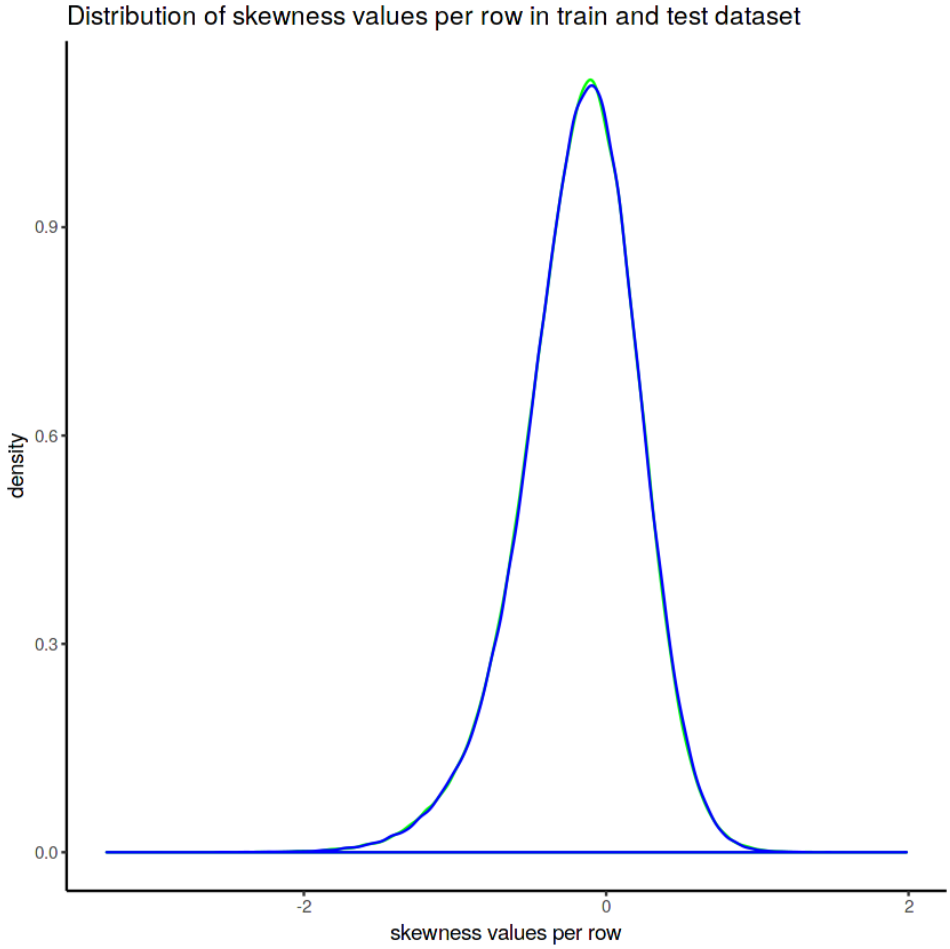

var_180
var_181
var_182
var_183
var_184
var_185
var_186
var_187
var_188
var_189
var_190
var_191
var_192
var_193
var_194
var_195
var_196
var_197
var_198
var_199
density
value
target
0
1

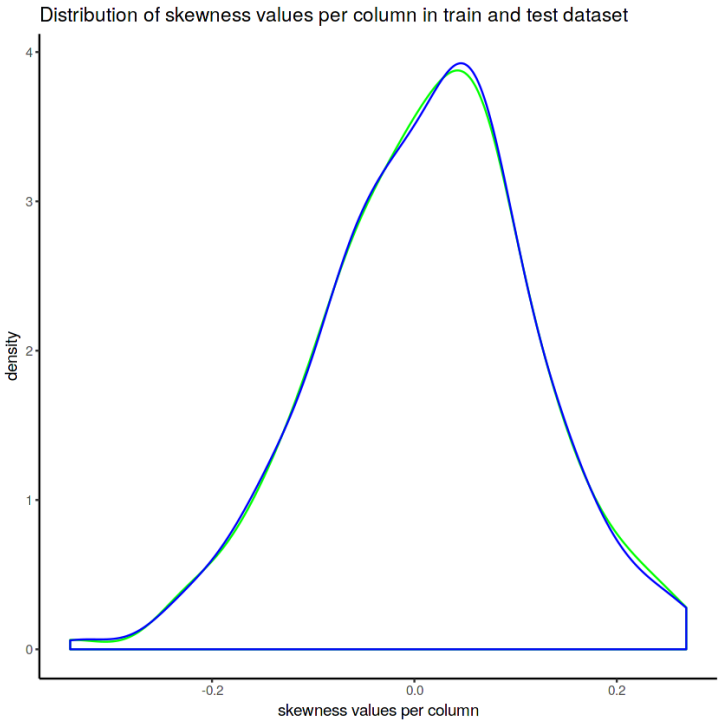

Distribution of mean values per row in train and test dataset
density
mean values per row

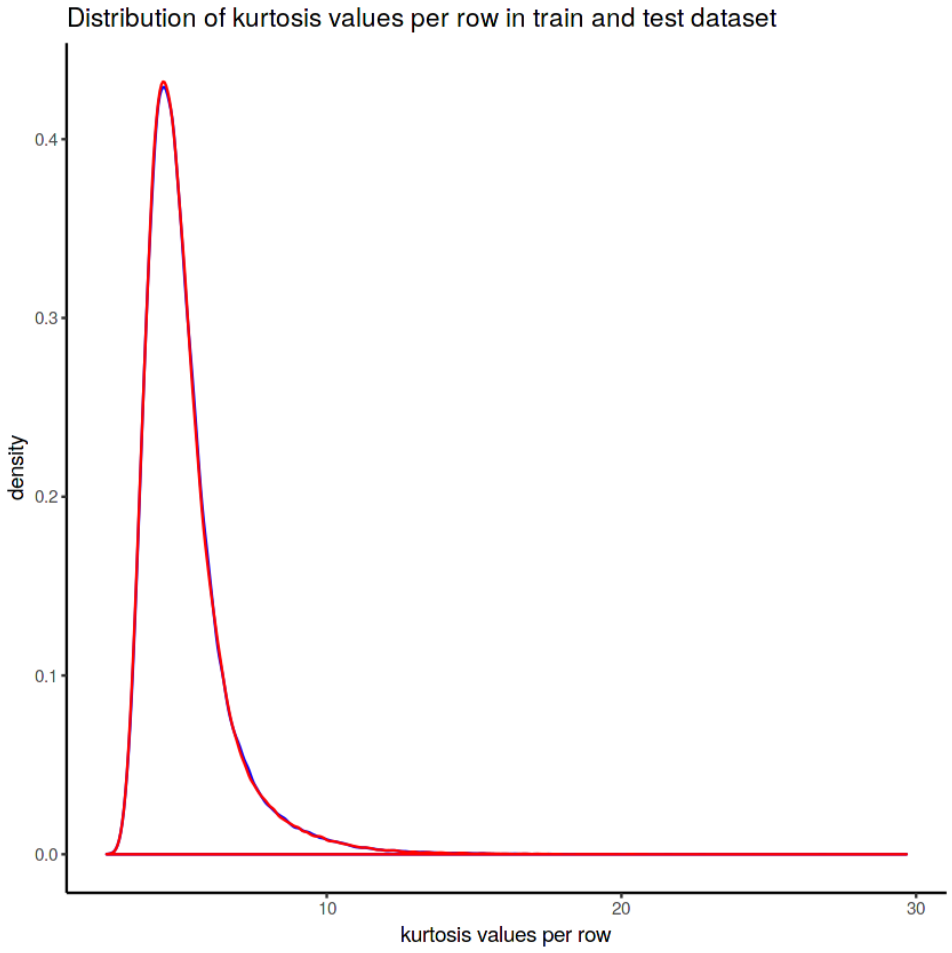

Distribution of mean values per row in train and test dataset
density
0.04
0.03
0.02
0.01
0.00
-10
0
10
20
mean values per column

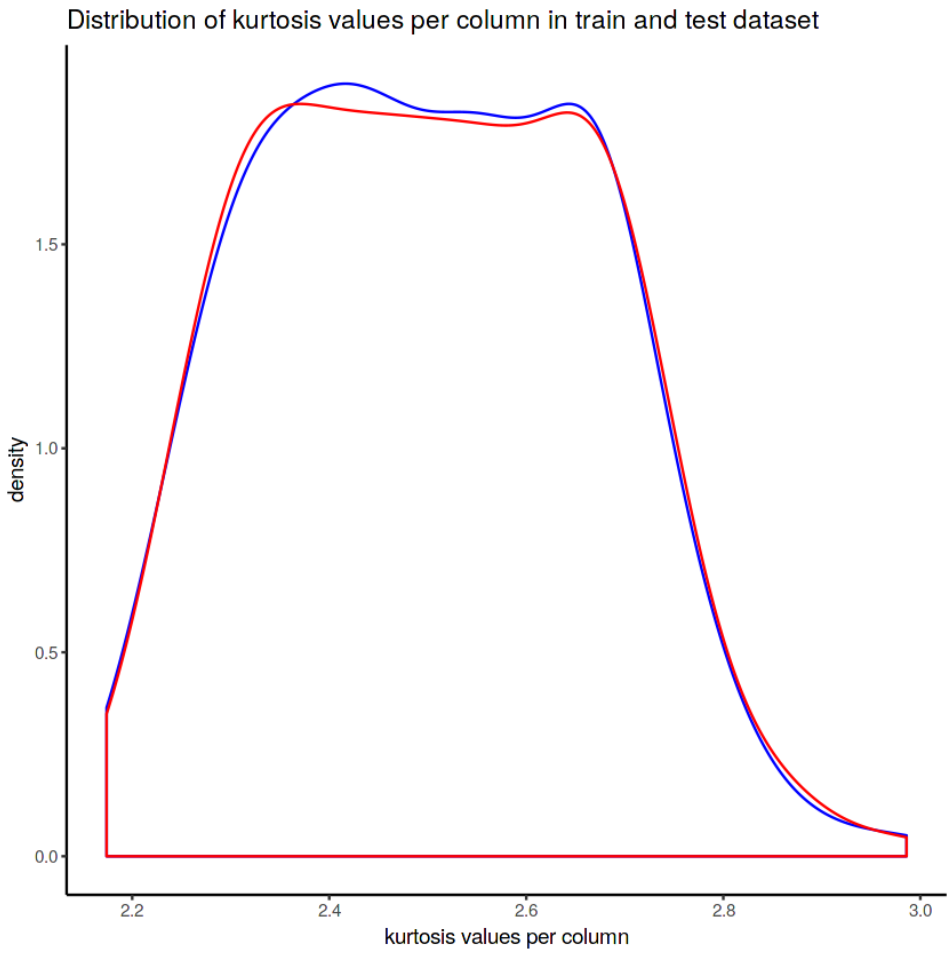

Distribution of sd values per row in train and test dataset
density
0.6
0.4
0.2
0.0
8
9
10
11
12
sd values per row


Distribution of std values per column in train and test dataset
density
0.125
0.100
0.075
0.050
0.025
0.000
0
5
10
15
20
sd values per column


Distribution of skewness values per row in train and test dataset
density
0.9
0.6
0.3
0.0
-2
0
2
skewness values per row


Distribution of skewness values per column in train and test dataset
density
0
1
2
3
4
-0.2
0.0
0.2
skewness values per column


Distribution of kurtosis values per row in train and test dataset
density
0.0
0.1
0.2
0.3
0.4
10
20
30
kurtosis values per row


Distribution of kurtosis values per column in train and test dataset
density
1.5
1.0
0.5
0.0
2.2
2.4
2.6
2.8
3.0
kurtosis values per column

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_round
s = 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_ro
unds = 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('\nDatatype 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