Santander Transaction Prediction

(2ND PROJECT IN EDWISOR)

by

B.SPOORTHI

Contents

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.

Softwares used –

a) R 4.0.0 for 64 bit.
b) Anaconda3 4.4.0 for 64 bit.

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 whether 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 modelling. 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 dividing 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 find .
2. As can be seen above, except for the target all other features are of type float64 and the ID code is of object type
3. Our target is int64 type with only two values i.e 0 and 1
4. Finding the spread of variance ,standard deviation ,mean ,count, minimum and maximum

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8
count	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
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	3.332634
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	-10.505500
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	-2.317800
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	0.393700
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	2.937900
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	10.151300

8 rows × 201 columns

Fig1: Calculation of different features present in the train data

```
In [10]: test.describe()
```

Out[10]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	10.658737	-1.624244	10.707452	6.788214	11.076399	-5.050558	5.415164	16.529143	0.277135	7.569407
std	3.036716	4.040509	2.633888	2.052724	1.616456	7.869293	0.864686	3.424482	3.333375	1.231865
min	0.188700	-15.043400	2.355200	-0.022400	5.484400	-27.767000	2.216400	5.713700	-9.956000	4.243300
25%	8.442975	-4.700125	8.735600	5.230500	9.891075	-11.201400	4.772600	13.933900	-2.303900	6.623800
50%	10.513800	-1.590500	10.560700	6.822350	11.099750	-4.834100	5.391600	16.422700	0.372000	7.632000
75%	12.739600	1.343400	12.495025	8.327600	12.253400	0.942575	6.005800	19.094550	2.930025	8.584825
max	22.323400	9.385100	18.714100	13.142000	16.037100	17.253700	8.302500	28.292800	9.665500	11.003600

8 rows × 200 columns

Fig2: Calculation of different features present in the test data

Observations from the above tables:

- standard deviation is relatively large.
- Min, max, mean, standard deviation values for train and test data looks quite close.
- Mean values are distributed over a large range.

2.1.1.1 Missing value analysis

Missing value analysis helps address several concerns caused by incomplete data. If cases with missing values are systematically different from cases without missing values, the results can be misleading. Also,

missing data may reduce the precision of calculated statistics because there is less information than originally planned. Another concern is that the assumptions behind many statistical procedures are based on complete cases, and missing values can complicate the theory required.

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
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
```


```
# 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:

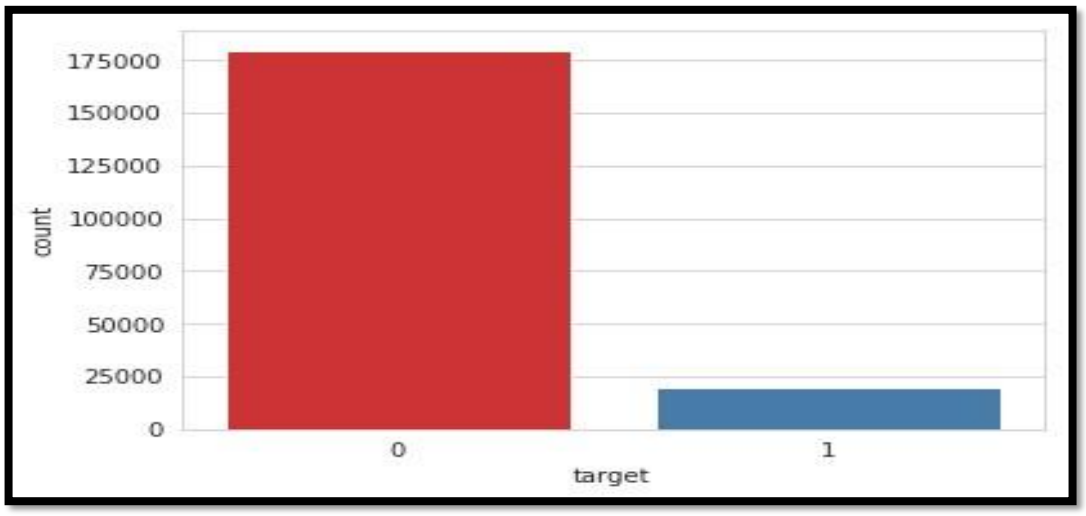
Here, we didn't find any sort of missing values in the data. So there is no need of imputing missing values also in this dataset.

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 pre-processing, modelling, and interpretation of results.

One of our main goals for visualizing the data here, is to observe which features are most intuitive 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



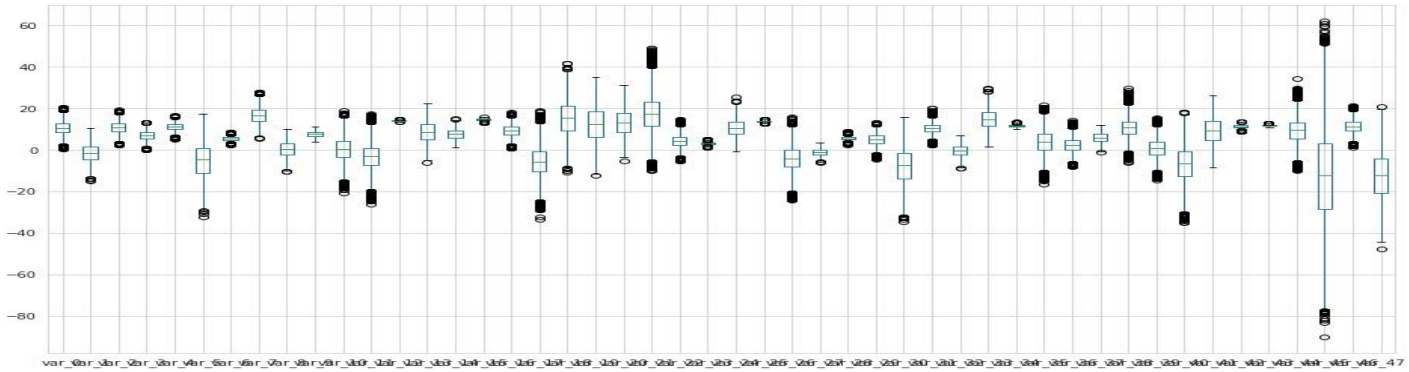
Observation:

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

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

2.1.2.2 Outlier Analysis: Boxplot

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



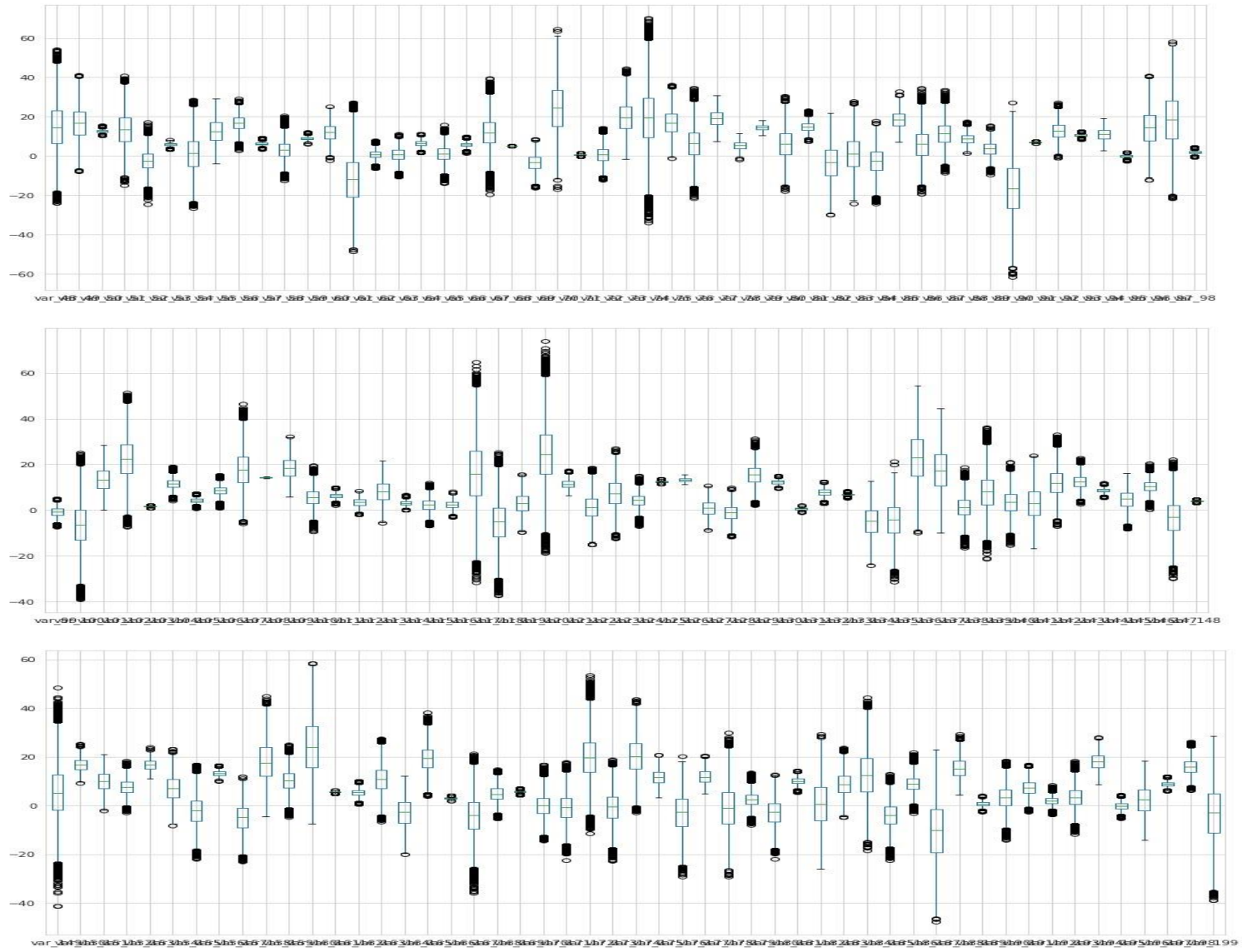


Fig: The above illustrates the box plot for 200 observations of train data.
We have defined a function to deal with outliers i.e. all values less than the 25 percentile and all values greater than the 75 percentile of the boxplot.
Arrange the data given by the client in ascending order then after:
Let the Q1=median in the upper data (25th percentile of data)
Q2=median of entire data
Q3=median of the bottom data (75th percentile of data)
By measuring the inter quartile range =Q3-Q1
The upper and lower quartiles can be calculated
Higher quartile range=Q3+1.5*IQR lower
quartile range=Q3-1.5*IQR
The outliers which are above lower quartile range and after higher quartile range are removed and then minimum and maximum values are calculated

Observations:

1. Most of the data have outlier and range of the data variables is high.

2. Data need to be free from outliers and need to be scaled before applying any outlier senstive model algorithms.
3. But here in the data after counting for the outliers ,most of persons who performs a transaction are found in outliers .Since the data is imbalanced we need not go for removing outliers.

2.1.2.3 Attributes Distributions and trends

Distribution of train attributes

The below figures are the distributions for 200 observations. These 200 observations specify that distributions for 1 and 0 in the train data columns. There are variations which do not follow a pattern. It is concluded that each and every observation has its own importance and labelled with distributions appropriately.

Moreover the distribution is plotted with the range that the features like var_0,var_1 accumulated as shown below. It is clear that all the features follow Gaussian normal distribution involving minor disturbances in the curves.

Fig: The above figure emphasis the normal distribution of train observations from 1 to 99

Fig:Distributions for train data observations from 101 to 200

Observations: All the train observations are having bell curve shaped distribution with mean 0 and variance 1 along with symmetry for targets '0' and '1'

1. There is a considerable number of features with significant different distribution for the two target values. For example, var_0, var_1, var_2, var_5, var_9, var_13, var_106, var_109, var_139 and many others.
2. We can observe that there is a considerable number of features which are significantly have same distributions for two target variables. For example, like var_3, var_7, var_10, var_17, var_35 etc.
3. if we look closely var_2, var_9, var_12, var_13, var_26, var_40, var_53, var_81, all of these variables have a bump of frequency that matches the rising of the probability of making a transaction. If pdf (target = 1) - pdf (target = 0) > 0, then there is a high probability of the client making a transfer.

Distribution of test attributes for some of the observations:

```
In [27]: #test attribiutes from 1 to 101 -
         test_attributes=test.columns.values[1:101]

         #Plot distribution of test attributes -
         plot_test_attribute_distribution(test_attributes)
```

```
<Figure size 432x288 with 0 Axes>
```

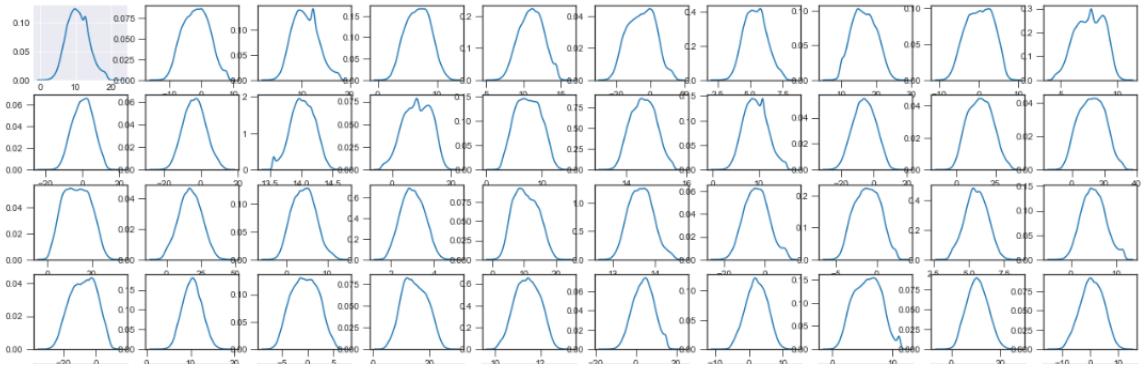


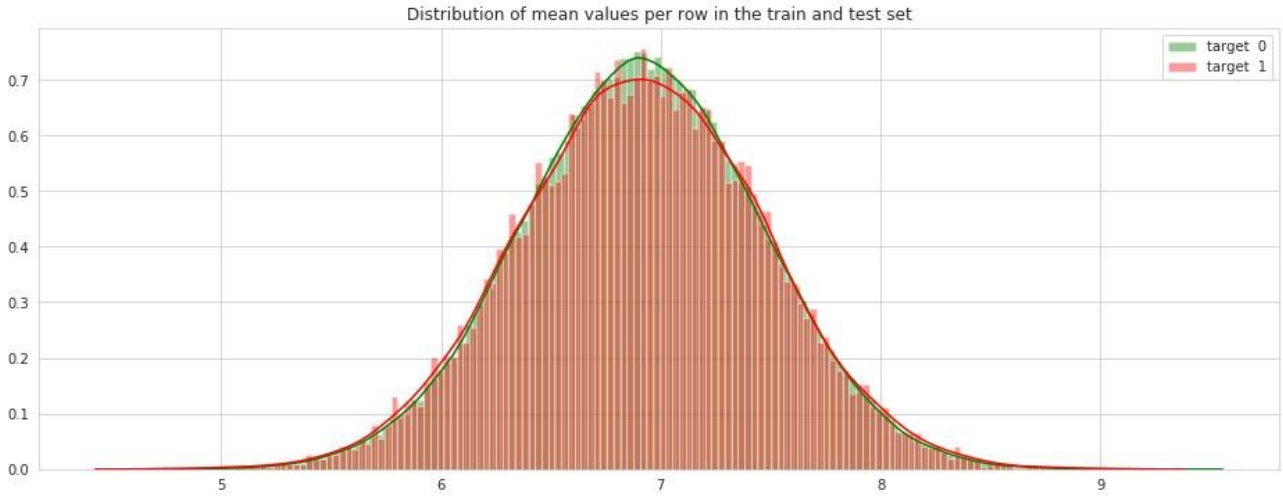
Fig:The above figure represents the normal distribution of some test attributes

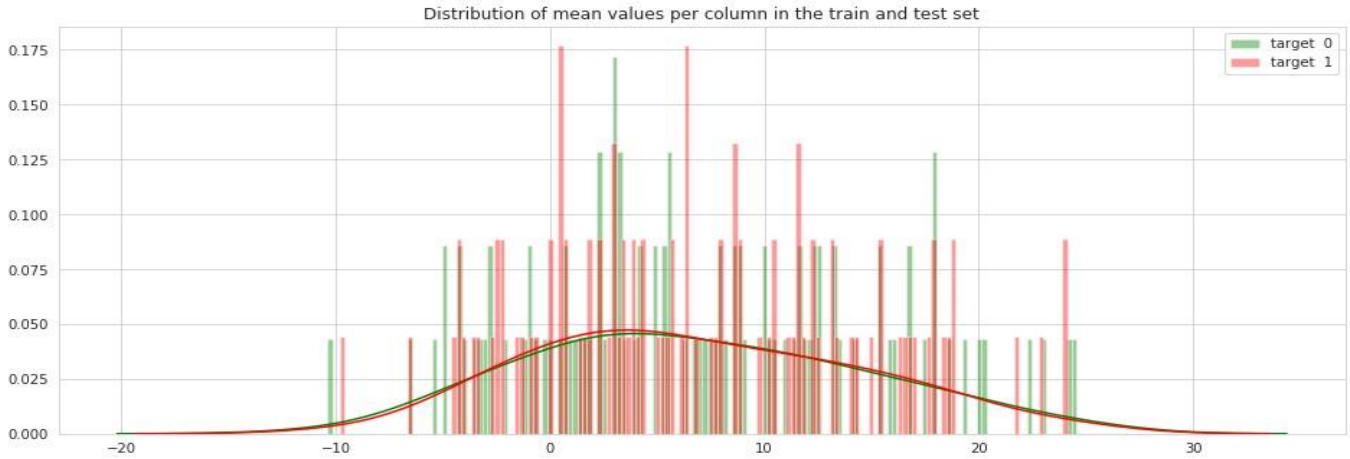
Distribution of Aggregates along the column and row

Observation: As there is not target defined as of now for test attributes just the distribution for each feature such as var_0,var_1,var_2,var_3 …. For 200 features it is plotted and for the 20 of them the distribution accordingly with their range is plotted and also concluded that the distributions of all the remaining features are in the same way i.e bell curved with added symmetry.

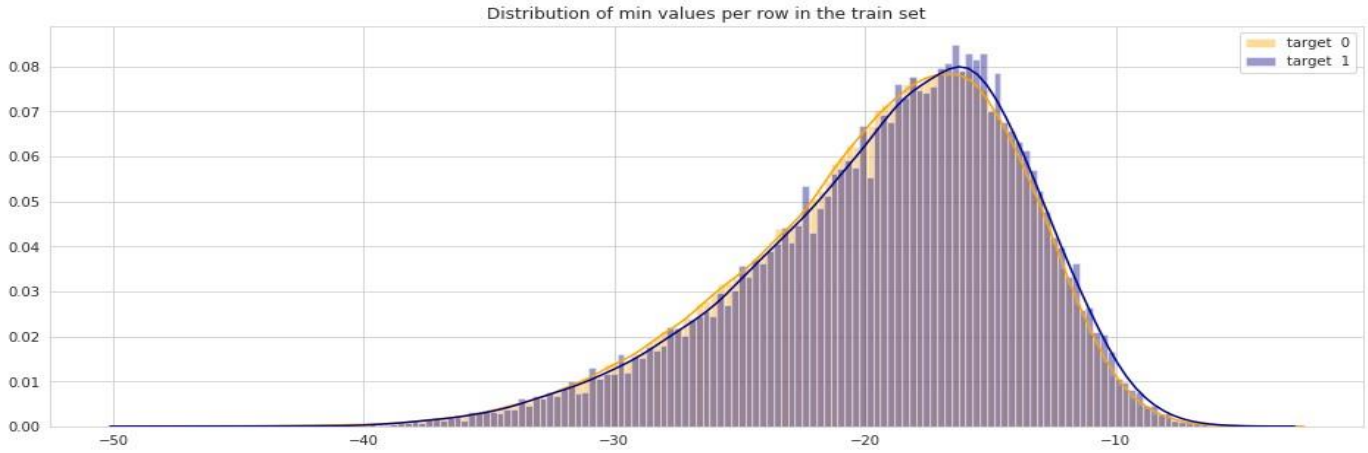
We shall check how our variables are distributed along the row and column with some parameters such as mean, median, minimum, maximum, skewness, sum and evaluate them for extracting characteristics

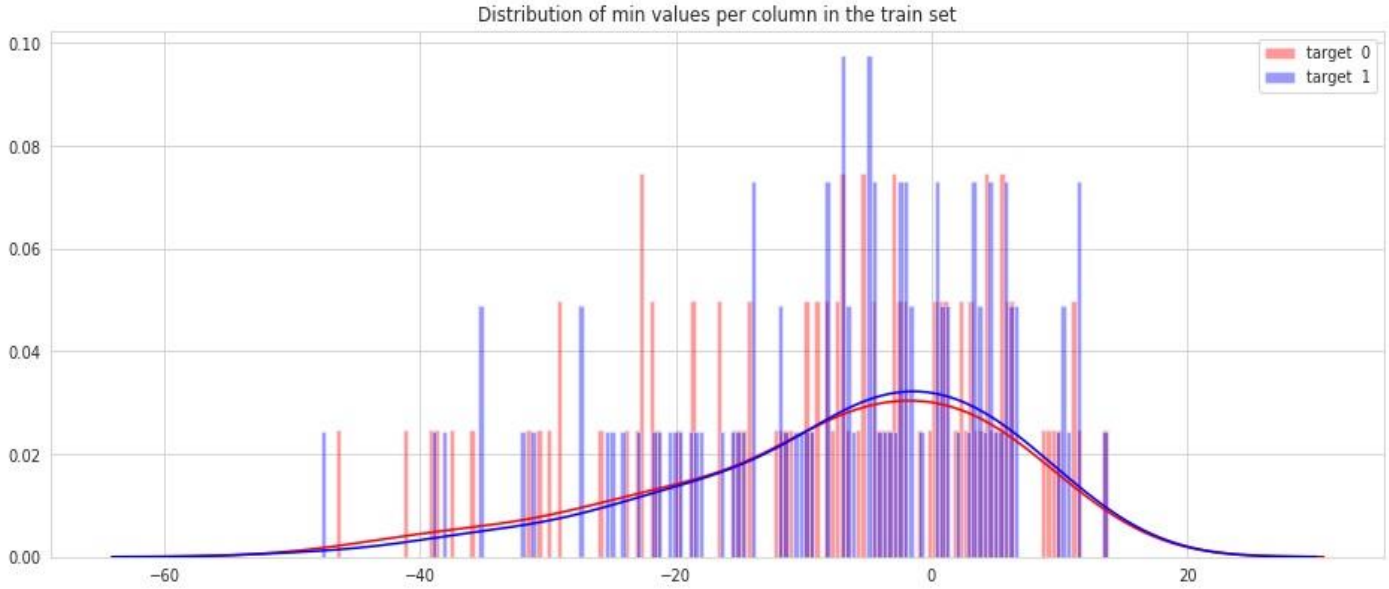
Let us look distribution of mean values:



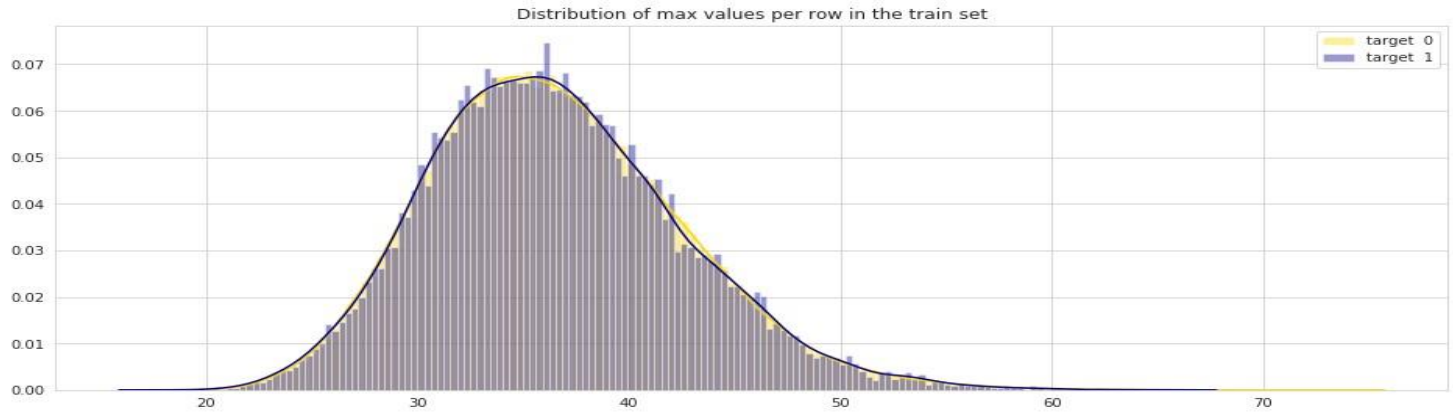


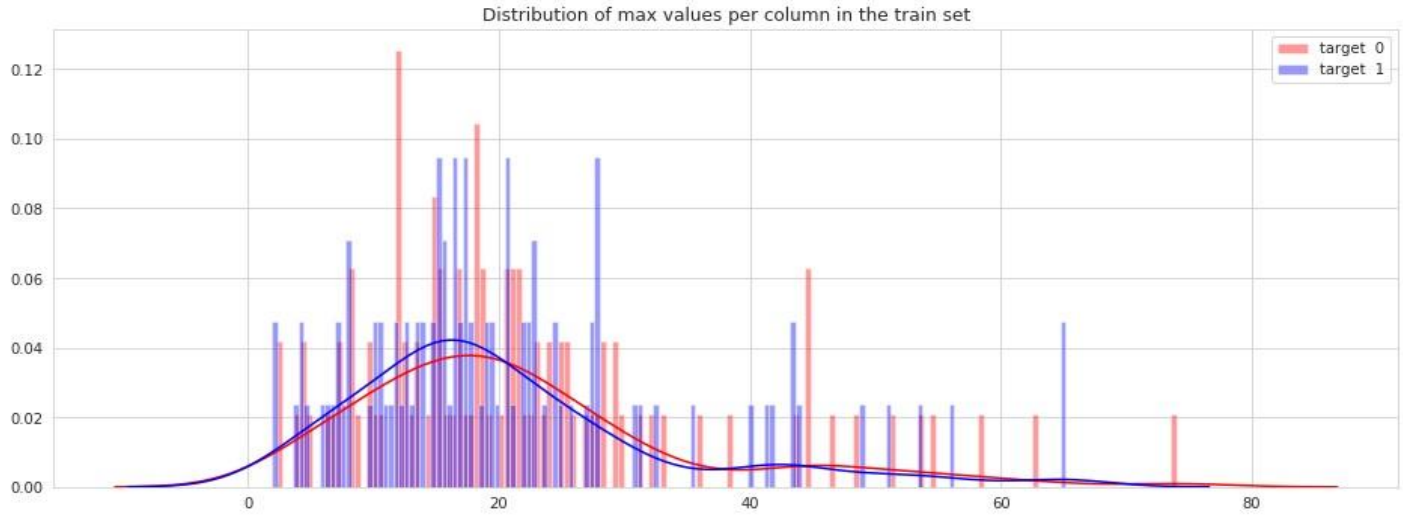
Let us look for the distributions of min values



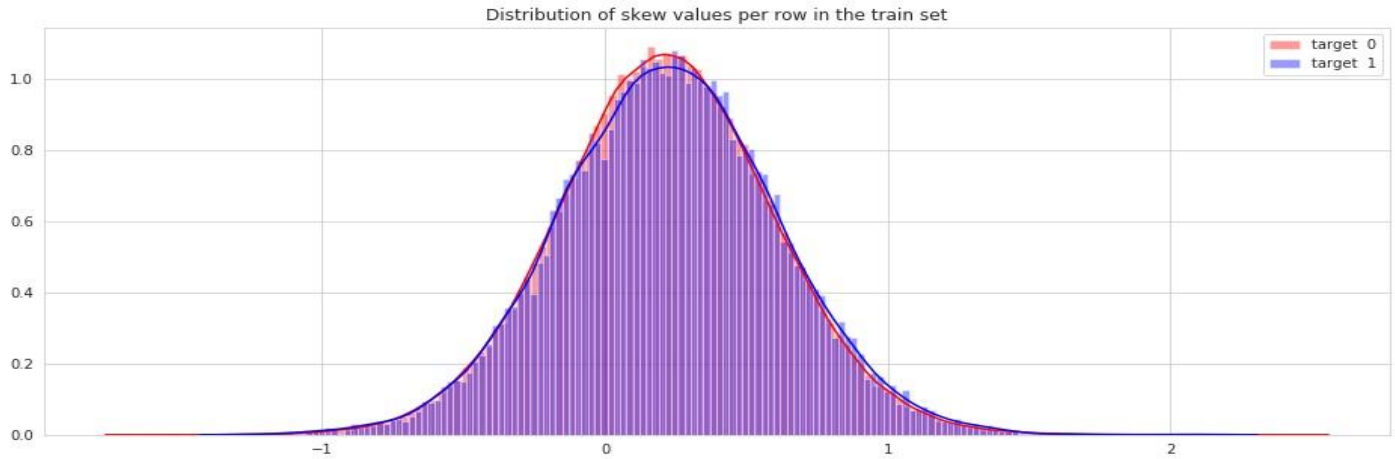


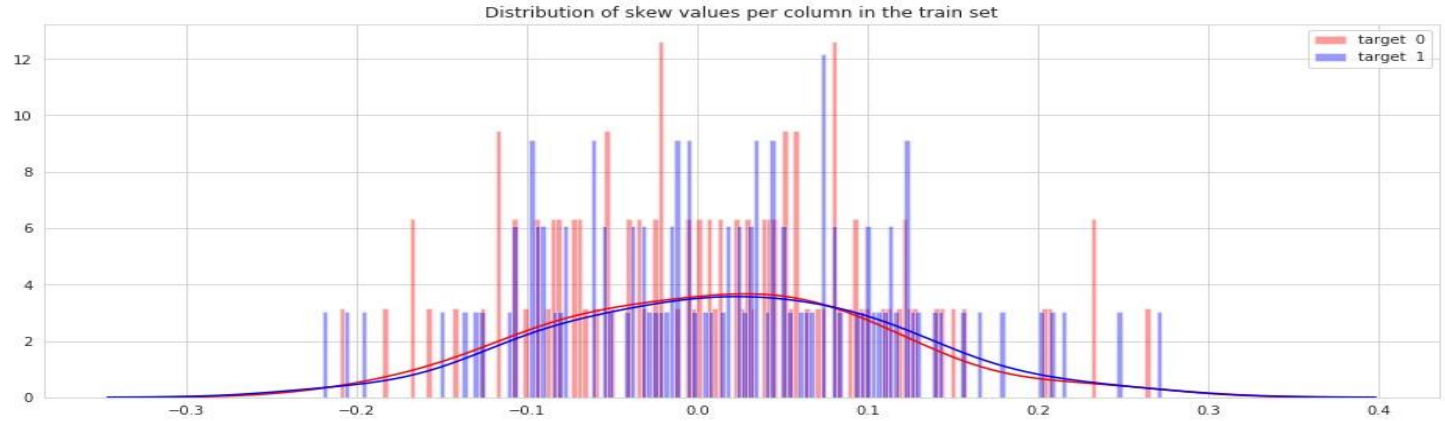
Let us look distribution of max values:



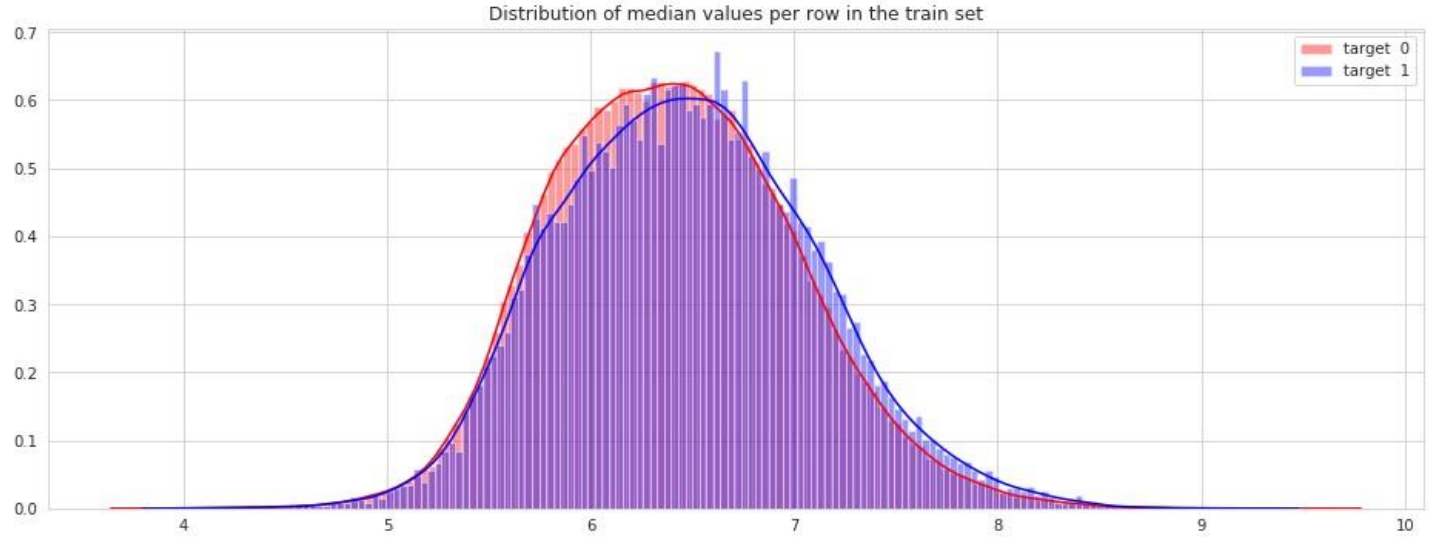


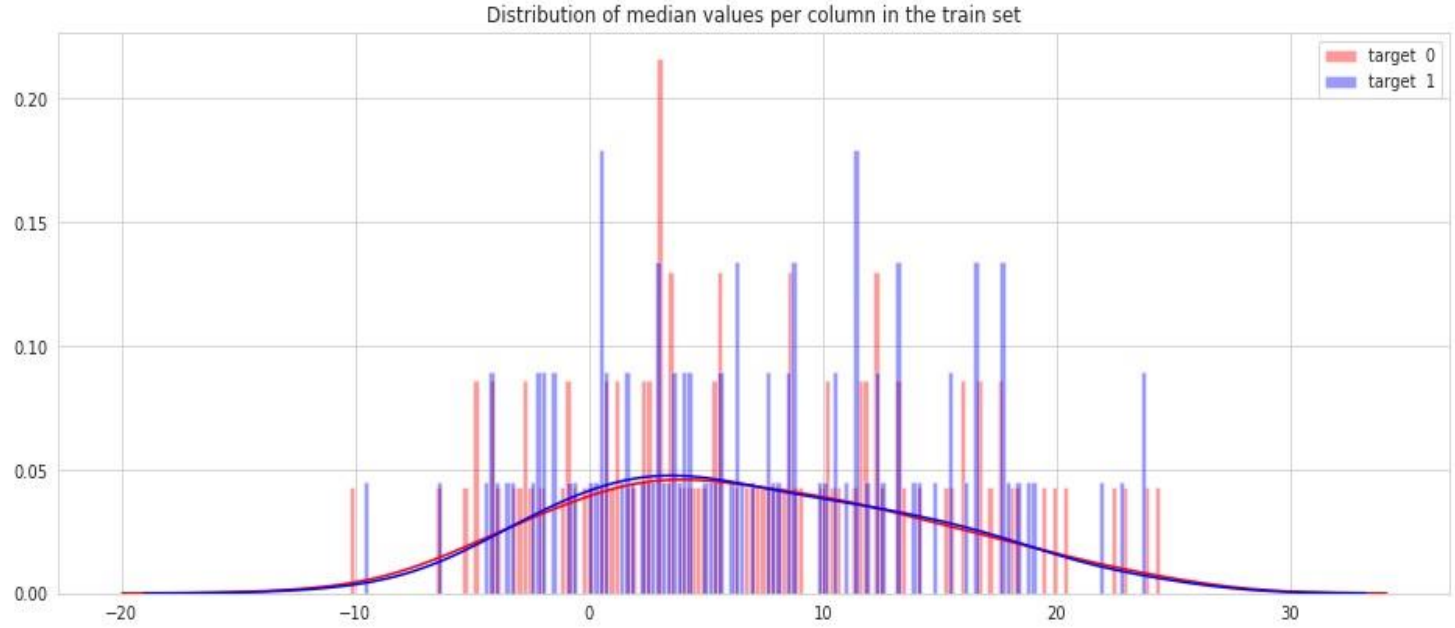
Let us look distribution of skewness values:



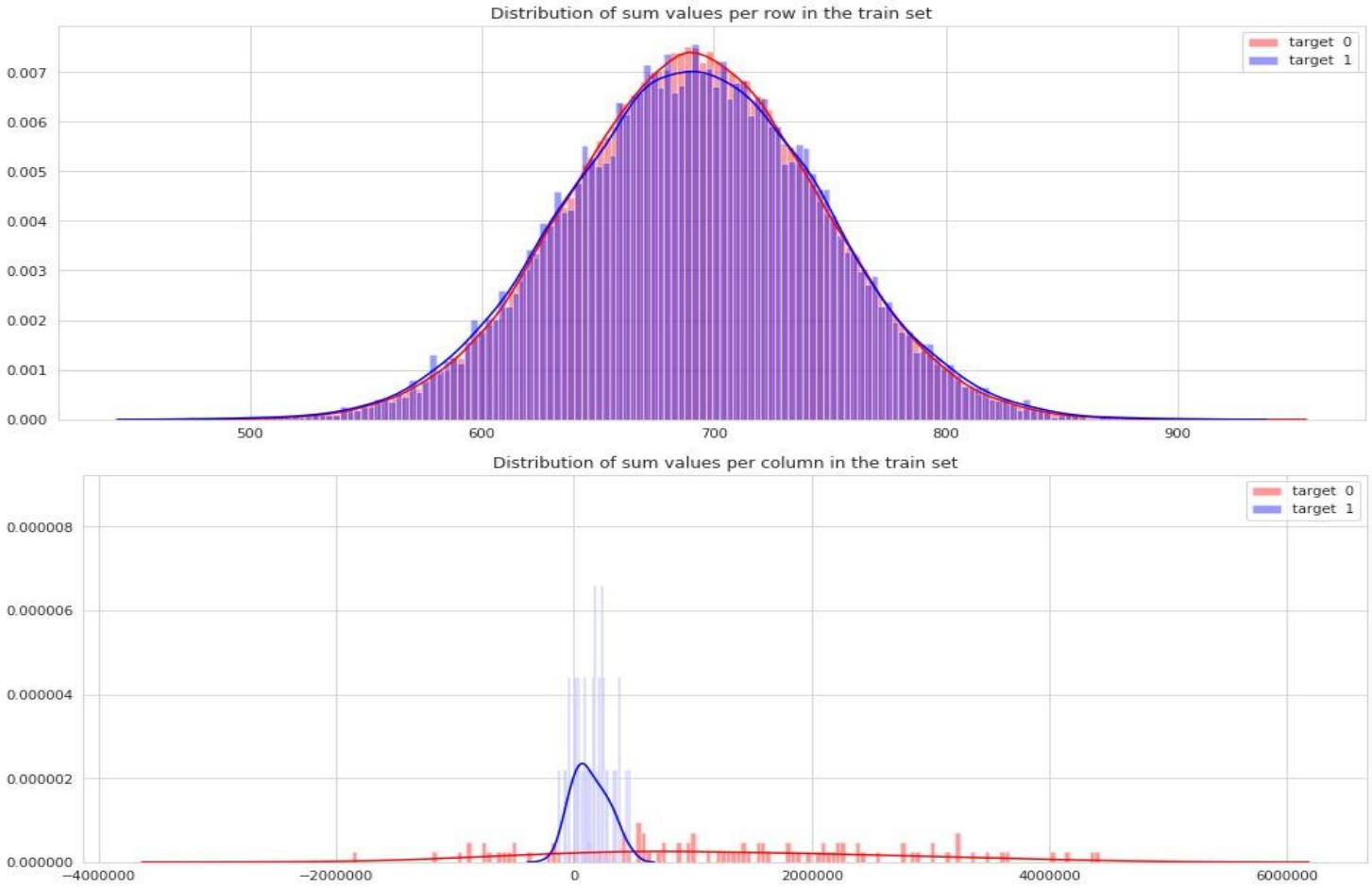


Distribution of mean values along row and column:





Let us look distribution of sum values:

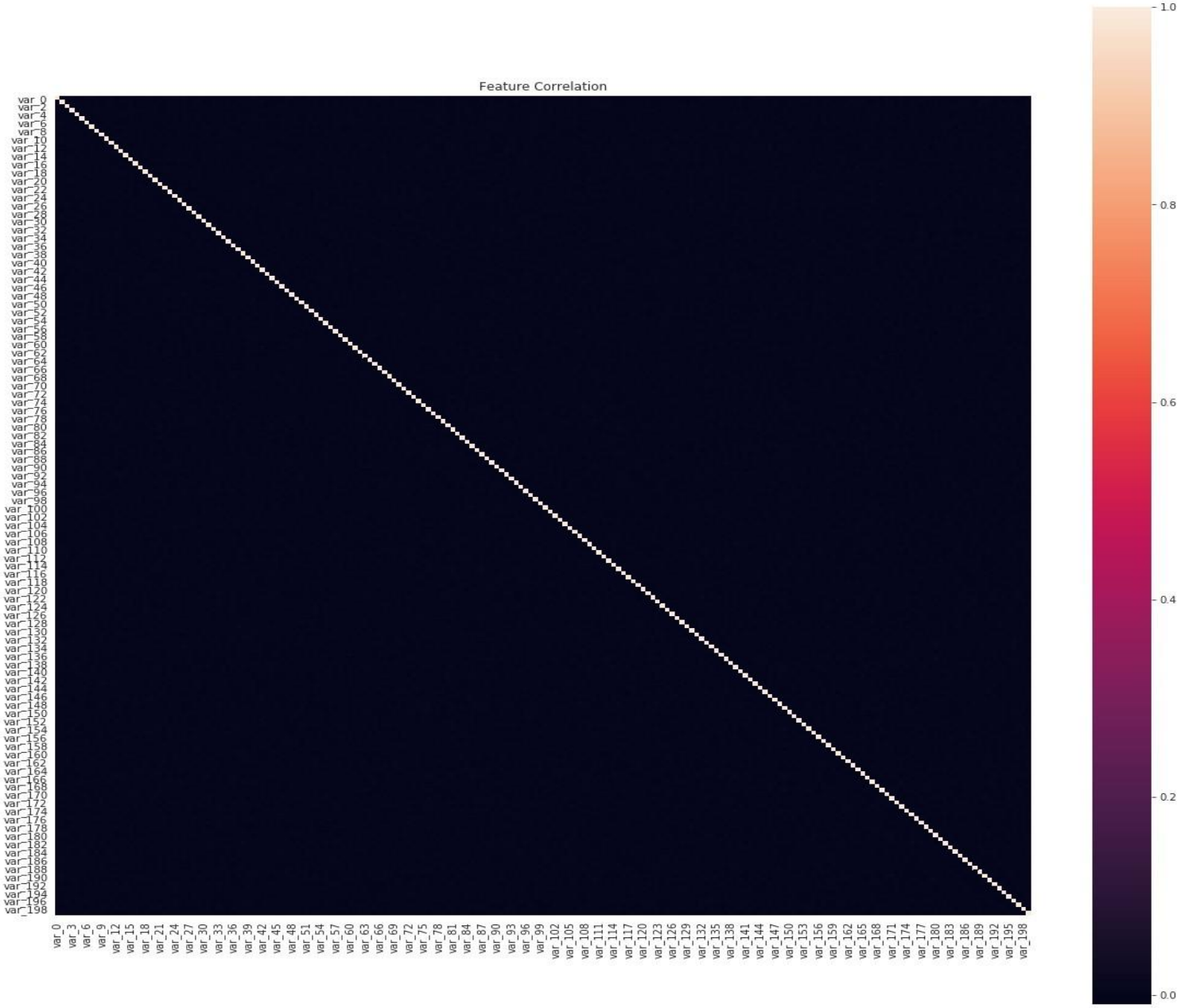


2.1.2.4 Correlation Analysis / Heatmap

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:

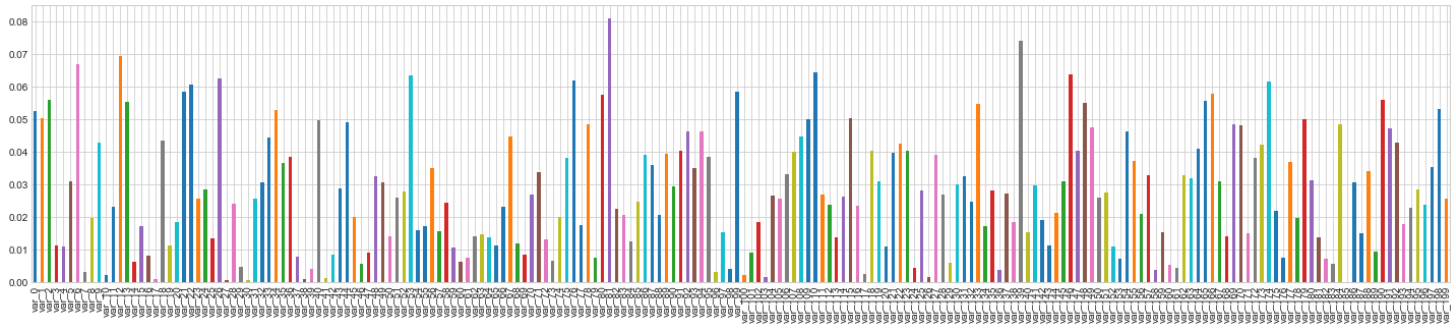
```
# Correlation Analysis
data_corr=train.drop(['target','ID_code'], axis=1).corr()
print('Maximum corr within all variables correlations :', np.sort(train.drop(['target','ID_code'], axis=1).corr())[:,-2:-1].max())
```

```
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. Hence we need not remove any of the variables and almost all variables have equal importance and contribute for the output
  - The below plot represents the correlation between the each attribute and the target variable and plotted accordingly. It illustrates that each and every variable contributes it is own importance in defining the target variable. The X-axis represents train attributes for configuring and Y axis represents the probability of explaining the target variable



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
```

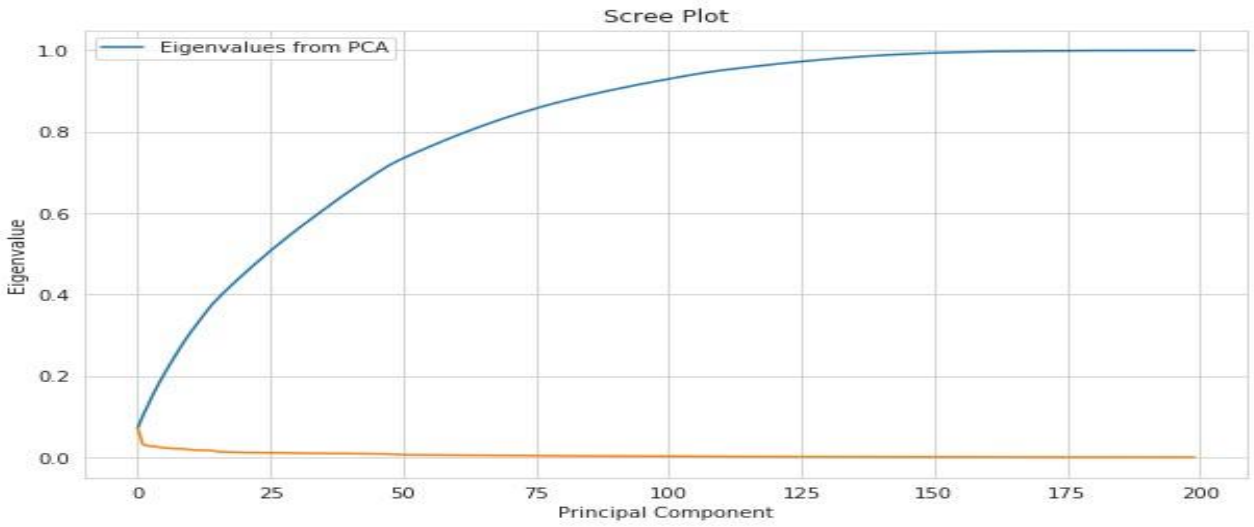
Observations:

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.
Below are the 25 least important feature according to our 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']
```

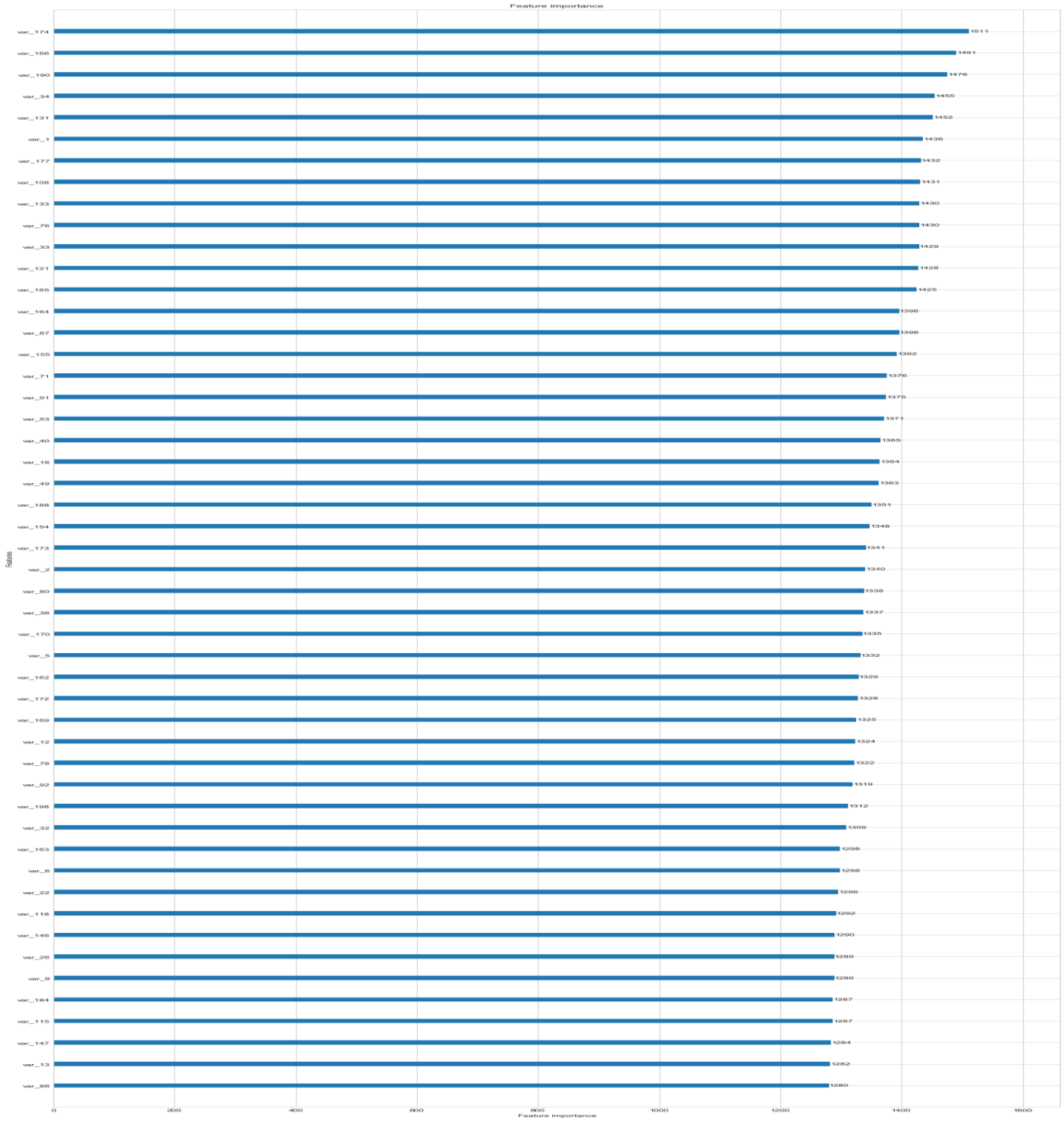


Fig: The below shown diagram represents the each attribute dependency on the target variable from train data set.

Observation:

Since Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction.Here it seems like all variables are having almost same feature importance.

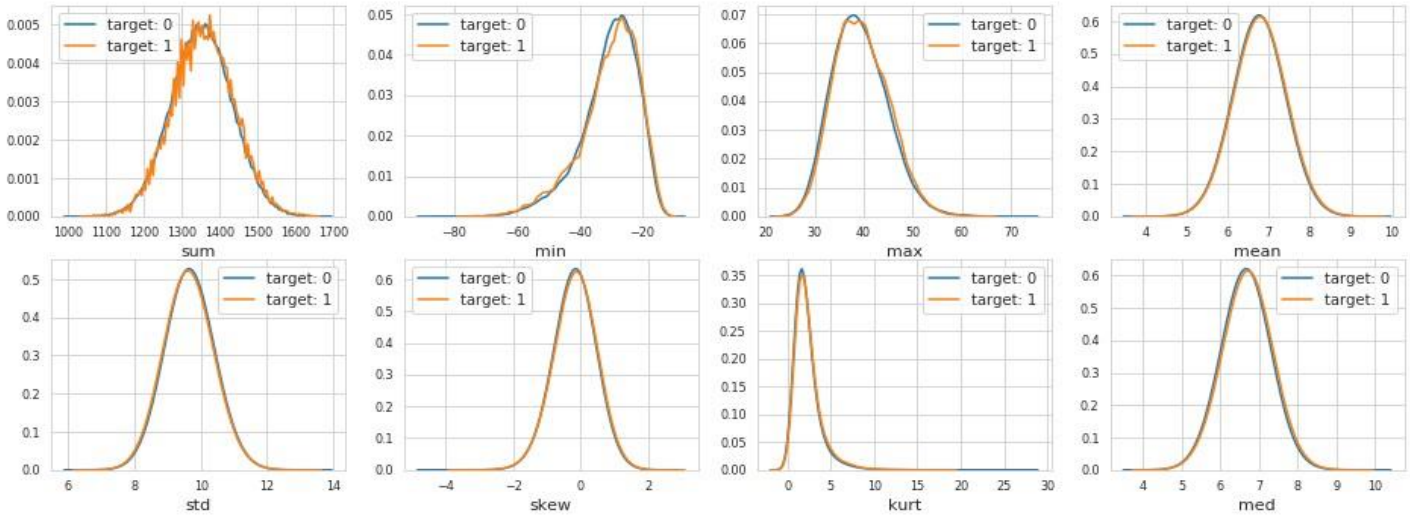
2.2.4 Feature Engineering

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 Modelling

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

LOGISTIC REGRESSION:

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).

RANDOM FOREST CLASSIFIER:

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

CART:

Classification and regression trees (CART) are a non-parametric decision tree learning technique that produces either classification or regression trees, depending on whether the dependent variable is categorical or numeric, respectively.

Decision trees are formed by a collection of rules based on variables in the modelling data set:

- Rules based on variables' values are selected to get the best split to differentiate observations based on the dependent variable
- Once a rule is selected and splits a node into two, the same process is applied to each "child" node (i.e. it is a recursive procedure)
- Splitting stops when CART detects no further gain can be made, or some pre-set stopping rules are met.

NAÏVE BAYES:

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable

2.3.1.1Without Standard Scale:

For logistic regression and for decision tree classifier with cart, the confusion matrix and classification report are as shown below.

```
 without StandardScale
LR :
 [[53248   723]
 [ 4434  1595]]
LR :
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     53971
           1       0.69      0.26      0.38      6029

avg / total       0.90      0.91      0.90     60000

CART :
 [[48806  5165]
 [ 4767  1262]]
CART :
              precision    recall  f1-score   support

           0       0.91      0.90      0.91     53971
           1       0.20      0.21      0.20      6029

avg / total       0.84      0.83      0.84     60000
```

Similarly, for naïve Bayes and for Random forest classifier the similar parameters are calculated as shown below.

```
NB :
 [[53105   866]
 [ 3860  2169]]
NB :
              precision    recall  f1-score   support

           0       0.93      0.98      0.96     53971
           1       0.71      0.36      0.48      6029

avg / total       0.91      0.92      0.91     60000

RFC :
 [[53882    89]
 [ 5936    93]]
RFC :
              precision    recall  f1-score   support

           0       0.90      1.00      0.95     53971
           1       0.51      0.02      0.03      6029

avg / total       0.86      0.90      0.85     60000
```

Observations: It is found that the naïve Bayes is doing its best in this dataset. It has got 0.91 precision and best recall comparatively. But this is not good in case of imbalanced data set containing unequal count of 0's and 1's.One such technique of solving imbalanced data is to adopt under sampling or oversampling .To acquire better accuracy and best parameter values we implement Smote (in python)and rose(in R) .

2.3.1.2 Standard Scale SMOTE:

We are dealing with imbalanced class problem, hence, we choose to oversampling our small class by using SMOTE 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:

- under sampling 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)
```

The classification report after performing smote analysis is drafted below from python:

Observation: The smote(synthetic minority oversampling technique) analysis resulted in decrease of performance for all the models such as logistic regression ,Random Forest,CART,Naïve bayes in terms of accuracy.

Now for the balanced dataset Naïve Bayes is doing best with almost 89% accuracy on it's average precision ,recall, F1_score, But the false negative rate observed on this model is good and found to be appropriate in this sort of customer predictions transaction classification problem.

```
 [[ 7299 28661]]
LR :
              precision    recall  f1-score   support

           0       0.80      0.79      0.79     35980
           1       0.79      0.80      0.79     35980

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

CART :
 [[26855  9125]
 [18261 17719]]
CART :
              precision    recall  f1-score   support

           0       0.60      0.75      0.66     35980
           1       0.66      0.49      0.56     35980

    accuracy                           0.62     71960
   macro avg       0.63      0.62      0.61     71960
weighted avg       0.63      0.62      0.61     71960
```

```
NB :
              precision    recall  f1-score   support

           0       0.81      0.95      0.88     35980
           1       0.94      0.78      0.85     35980

    accuracy                           0.87     71960
   macro avg       0.88      0.87      0.86     71960
weighted avg       0.88      0.87      0.86     71960

RFC :
 [[34134  1846]
 [20776 15204]]
RFC :
              precision    recall  f1-score   support

           0       0.62      0.95      0.75     35980
           1       0.89      0.42      0.57     35980

    accuracy                           0.69     71960
   macro avg       0.76      0.69      0.66     71960
weighted avg       0.76      0.69      0.66     71960
```

2.2.3 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'.

2.2.3.1 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)
```

2.2.3.1 SMOTE:

SMOTE is a technique based on nearest neighbours judged by Euclidean Distance between data points in feature space. There is a percentage of Over-Sampling which indicates the number of synthetic samples to be created

In this technique synthetic samples are created for only for the minority class. It is used to obtain a synthetically class-balanced or nearly class-balanced training set, which is then used to train the classifier.

CHAPTER 3

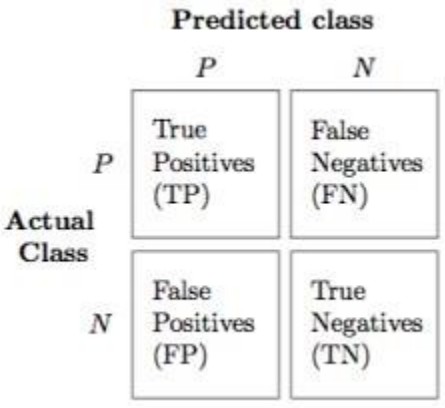
Conclusion

This was a binary classification problem on a typically unbalanced data set with no missing values. Predictor variables are anonymous and numeric and target variable is categorical. Visualizing descriptive features and finally I got to know that these variables are not correlated among themselves. After that I decided to treat imbalanced data set. Modelling with LightGBm , using the standard model parameters with feature engineered data I got AUC-Score of 0.899 and after tuning parameters with K fold stratified sampling final value of AUC Score is 0.952 and F1 score of 0.89 for test data.

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.



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").

ROC_AUC SCORE:

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings

LR :: 0.6264835546403751
CART :: 0.5605205214209749
NB :: 0.6741809150880748
RFC :: 0.5001406039841858

ACCURACY RESULTS:

After performing 3 fold cross validation the accuracy results for the models are:
LR :: 0.9136142831544322
CART :: 0.8351071414627561
NB :: 0.9213500026450546
RFC ::0.8995071325406924

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

```
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 ,our later model with SMOTE is doing way much better than former in all the models.
- Also, NB have quite balanced f1-score among all which could lead on better confidence.

.

Python Light GBM:

- **Simple Light GBM**
- **Confusion matrix:**

array([[46422, 7549],
       [1609, 4420]], dtype=int64)

Prediction on test data:

```
sub_df =  pd.DataFrame({"ID_code":test["ID_code"].values})
sub_df["target"] = predictions
sub_df.to_csv("lgbm.csv", index=False)
```

```
lgbm=pd.read_csv('lgbm.csv')
lgbm.head()
```

	ID_code	target
0	test_0	0.054296
1	test_1	0.214405
2	test_2	0.210391
3	test_3	0.220053
4	test_4	0.042658

```
In [66]: num_rounds=1000
         lgbm1= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],feval=lgb_f1_score,verbose_eval=100,early_stopping_
```

```
Training until validation scores don't improve for 500 rounds
[100]   training's auc: 0.90473 training's f1: 0.501978 valid_1's auc: 0.861808 valid_1's f1: 0.453369
[200]   training's auc: 0.916043        training's f1: 0.520313 valid_1's auc: 0.871636 valid_1's f1: 0.463072
[300]   training's auc: 0.921524        training's f1: 0.530338 valid_1's auc: 0.874694 valid_1's f1: 0.471082
[400]   training's auc: 0.925516        training's f1: 0.538659 valid_1's auc: 0.877351 valid_1's f1: 0.475187
[500]   training's auc: 0.929301        training's f1: 0.544808 valid_1's auc: 0.879744 valid_1's f1: 0.476082
[600]   training's auc: 0.933148        training's f1: 0.550923 valid_1's auc: 0.881459 valid_1's f1: 0.481687
[700]   training's auc: 0.935784        training's f1: 0.555645 valid_1's auc: 0.882739 valid_1's f1: 0.484115
[800]   training's auc: 0.938713        training's f1: 0.562333 valid_1's auc: 0.884134 valid_1's f1: 0.486005
[900]   training's auc: 0.941561        training's f1: 0.570638 valid_1's auc: 0.885102 valid_1's f1: 0.489819
[1000]  training's auc: 0.944108        training's f1: 0.574578 valid_1's auc: 0.885893 valid_1's f1: 0.491166
Did not meet early stopping. Best iteration is:
[1000]  training's auc: 0.944108        training's f1: 0.574578 valid_1's auc: 0.885893 valid_1's f1: 0.491166
```

- **SMOTE Light GBM:**

  Confusion matrix:

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

Prediction on test data:

```
Training until validation scores don't improve for 5000 rounds
[1000]  training's auc: 0.956207       training's f1: 0.885727 valid_1's auc: 0.93558  valid_1's f1: 0.854468
[2000]  training's auc: 0.969384       training's f1: 0.906972 valid_1's auc: 0.944535 valid_1's f1: 0.867009
[3000]  training's auc: 0.977196       training's f1: 0.920779 valid_1's auc: 0.948302 valid_1's f1: 0.871963
[4000]  training's auc: 0.982755       training's f1: 0.931635 valid_1's auc: 0.950366 valid_1's f1: 0.874556
[5000]  training's auc: 0.987019       training's f1: 0.940849 valid_1's auc: 0.951436 valid_1's f1: 0.875119
[6000]  training's auc: 0.990332       training's f1: 0.949184 valid_1's auc: 0.952014 valid_1's f1: 0.875143
[7000]  training's auc: 0.992891       training's f1: 0.957019 valid_1's auc: 0.952321 valid_1's f1: 0.875251
[8000]  training's auc: 0.99484 training's f1: 0.963922 valid_1's auc: 0.952444 valid_1's f1: 0.875006
[9000]  training's auc: 0.996309       training's f1: 0.97027  valid_1's auc: 0.95262  valid_1's f1: 0.874528
[10000] training's auc: 0.997386       training's f1: 0.975572 valid_1's auc: 0.952645 valid_1's f1: 0.873892
Did not meet early stopping. Best iteration is:
[10000] training's auc: 0.997386       training's f1: 0.975572 valid_1's auc: 0.952645 valid_1's f1: 0.873892
```

- Smote analysis with Light BGM is ensuring correct prediction on our dataset and this model is classifying the customers' will not perform the transaction significantly when contrasted with all other modelling techniques.
- This confusion matrix gives 89% accuracy and 0.9579 ROC_AUC score and sits as the best model.

Observation:

Python:

- NB being the best scorer among all initial 4 models.
- Light GBM with simple stratified and smote better than any other model with accuracy of 0.95

R:

- Light GBM with raw Data is scoring accuracy 0.889, and which is best of all getting significant F1 score

Model Selection

. We could conclude that below points as the follow:

1. CART, Random Forest Classifier and LR models did not performed well on imbalanced data.

2.Naive Bayes performed well and has a better recall.

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

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

Finally, Light GBM approach is the best choice for identifying which customers will make a specific transaction in the future, irrespective of the amount of money transferred.

References

- https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
- https://medium.com/@pushkarmandot/https-medium-com-pushkarmandotwhat-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters60347819b7fc
- https://machinelearningmastery.com/roc-curves-and-precision-recall-curvesfor-classification-in-python/
- https://www.analyticsvidhya.com/blog/2016/03/practical-guide-dealimbalanced-classification-problems/
- https://www.kaggle.com/pritamjena/testing-the-imbalanced-data-using-rosepackage