Santander Customer Transaction Prediction

Hitesh Kumar

Content

.

Introduction

Background

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

Problem statement

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

2 Data

2.1 Data

We have a train and test Data set.

Peeking inside the train data:

ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...
train_0	0	8.9255	-6.7863	11.9081	5.093	11.4607	-9.2834	5.1187	18.6266	...
train_1	0	11.5006	-4.1473	13.8588	5.389	12.3622	7.0433	5.6208	16.5338	...
train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...
train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.925	...
train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...

...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_197	var_198	var_199
...	4.4354	3.9642	3.1364	1.691	18.5227	-2.3978	7.8784	8.5635	12.7803	-1.0914
...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267	8.7889	18.356	1.9518
...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213	8.2675	14.7222	0.3965
...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275	10.2922	17.9697	-8.9996
...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267	9.5031	17.9974	-8.8104

Predictor Variable

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

Shape of Data

The shape of the train data: 200000 x 202

The shape of the test data: 200000 x 201

The details of the data attribute in the dataset are as follow

The data set consists of 200000 observations and 201 variables which are anonymized means which we have no information about the variables and 1 target variable.

Variables and their Data types

Train data:

```
float64    200
object       1
int64        1
```

Test data:

```
float64    200
object       1
```

2.2 Missing value Analysis

Here we are trying to find any missing values in our data. As we can see there are no Missing values in either train or test.

```
train.isna().sum().sum()
```

```
0
```

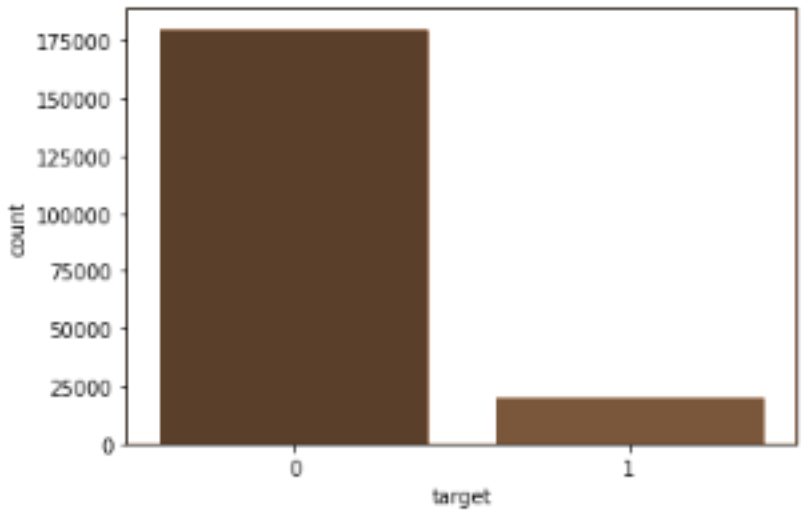
```
test.isna().sum().sum()
```

```
0
```

3 Exploratory Data Analysis

3.1 Univariate Analysis

Distribution of Target variable:



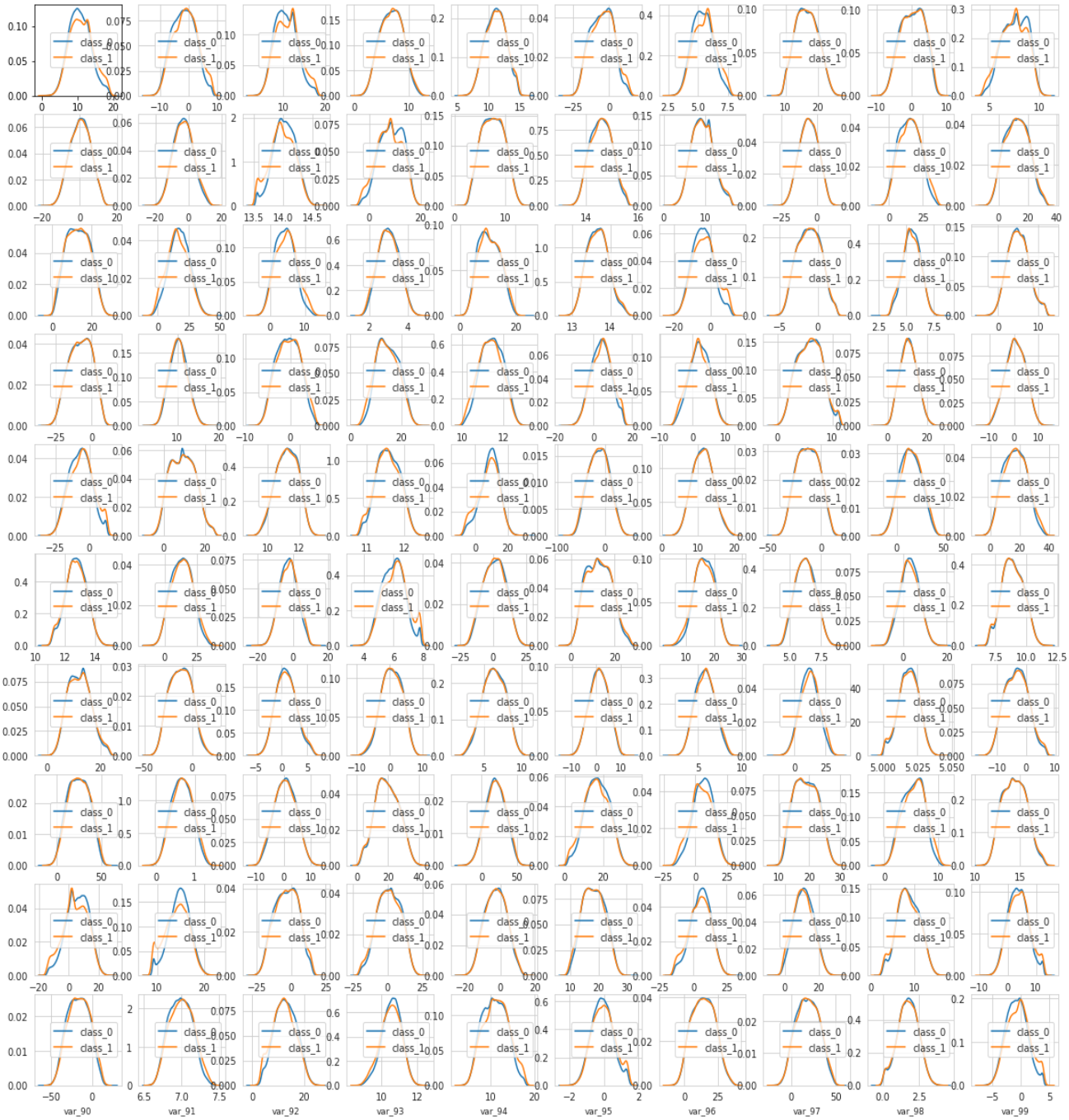
The data is unbalanced with respect with target value.

There are 10.049% target values with 1.

Density plots of features

Let's see the density plot of variables in **train** dataset.
We will represent the distribution for values with **target** value **0** and **1** with different colors.

First Hundred Variables



Next Hundred Variables from Train

We can observe that there is a considerable number of features with significant different distribution for the two target values.
For example, **var_0**, **var_1**, **var_5**, **var_9**, **var_13**, **var_106**, **var_109**, **var_139** and many others.

Also some features, like **var_2**, **var_13**, **var_26**, **var_55**, **var_175**, **var_184**, **var_196** shows a distribution that resembles a bivariate distribution.

Let's see the density plot of variables in the test dataset.

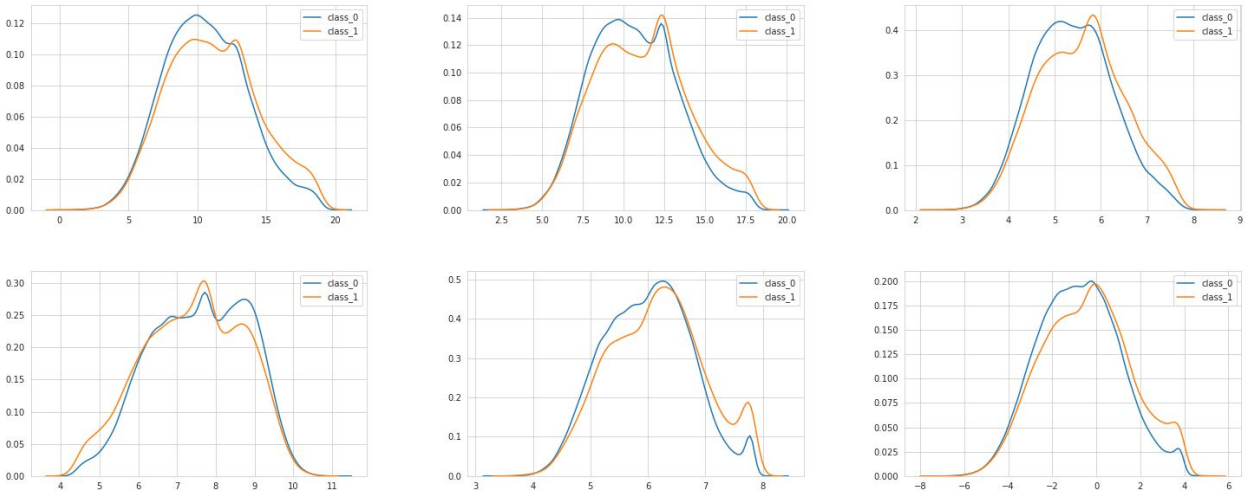
First Hundred variables

Next Hundred Variables

The train and test seems to be well balanced with respect with distribution of the numeric variables.

Observation in dataset:

We can see that some variables have bumps on both the left and right side. For example in the image below we can see bumps on the right side.



3.2 Descriptive Data Analysis:

Descriptive statistics are used to describe the basic features of the data in a study. They provide simple summaries about the sample and the measures. Together with simple graphics analysis, they form the basis of virtually every quantitative analysis of data.

Summary statistics of Train:

target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	.....
count	200000	200000	200000	200000	200000	200000	200000	...
mean	0.10049	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	...
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.62315	7.863267	...
min	0	0.4084	-15.0434	2.1171	-0.0402	5.0748	-32.5626	...
25%	0	8.45385	-4.740025	8.722475	5.254075	9.883175	-11.20035	...
50%	0	10.52475	-1.60805	10.58	6.825	11.10825	-4.83315	...
75%	0	12.7582	1.358625	12.5167	8.3241	12.261125	0.9248	...
								...

Summary statistics of Test:

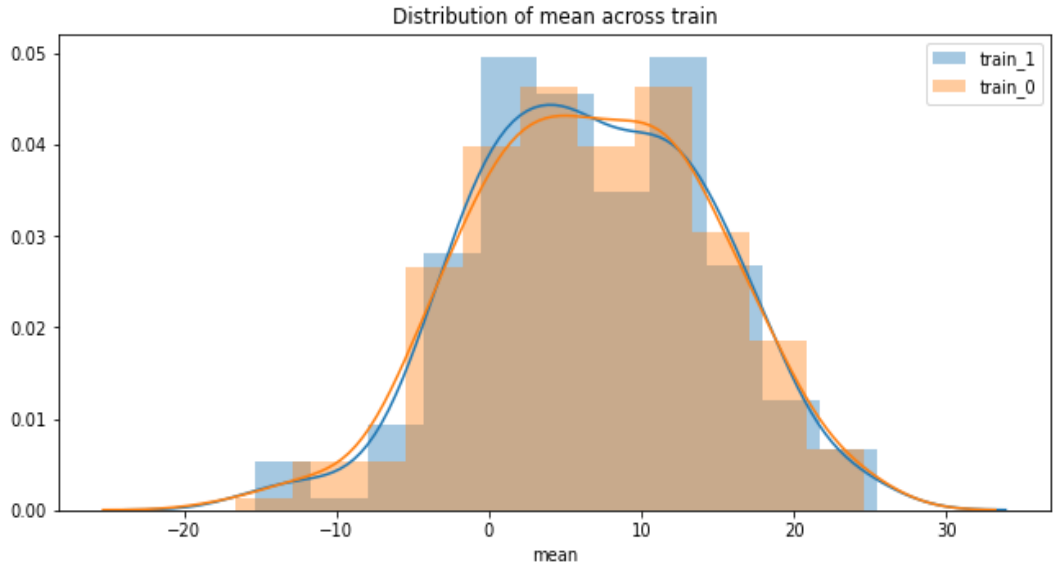
target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	.....
count	20098	20098	20098	20098	20098	20098	20098	...
mean	1	11.156418	-1.017613	11.156633	6.864113	11.131337	-4.336522	...
std	0	3.270293	4.220638	2.841075	2.070898	1.649266	8.140281	...
min	1	0.4528	-14.037	2.9462	0.374	5.8762	-28.2461	...
25%	1	8.695875	-4.203475	8.961125	5.3143	9.91125	-10.6158	...
50%	1	11.00135	-0.99265	11.0967	6.9002	11.1563	-4.10195	...
75%	1	13.3437	2.001375	13.047025	8.384725	12.32845	1.61995	...
max	1	19.4583	9.0298	18.2941	12.7069	15.6925	16.4236	...

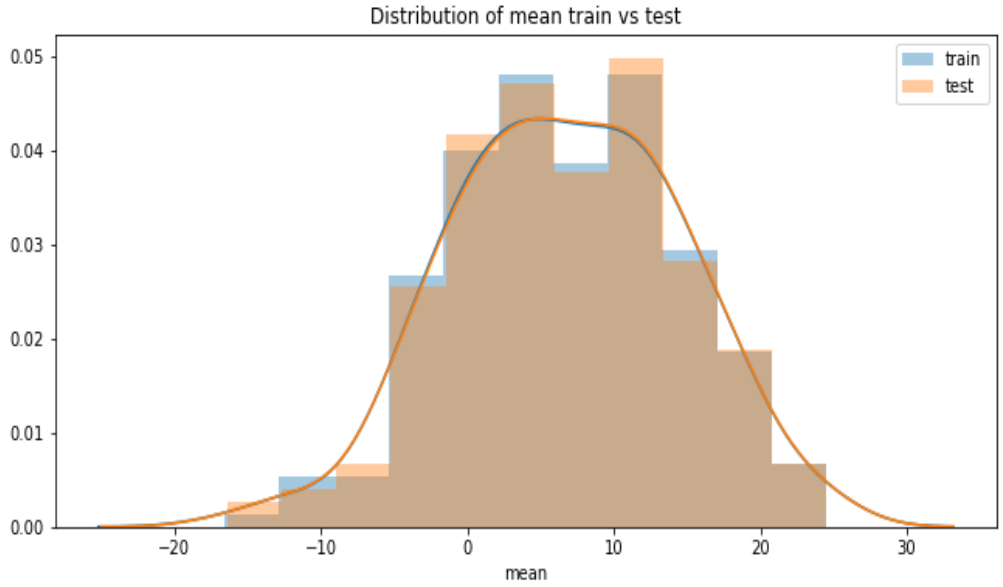
We can make few observations here:

- standard deviation is relatively large for both train and test variable data;
- min, max, mean, sdt values for train and test data looks quite close;
- mean values are distributed over a large range.

The number of values in the train and test set is the same.

Distribution of Mean Across all variables in Train and test:

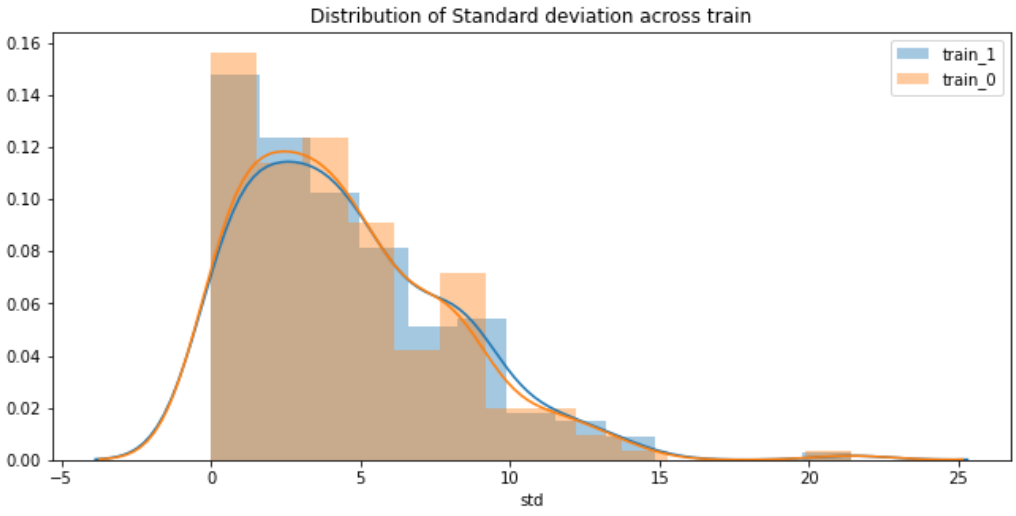


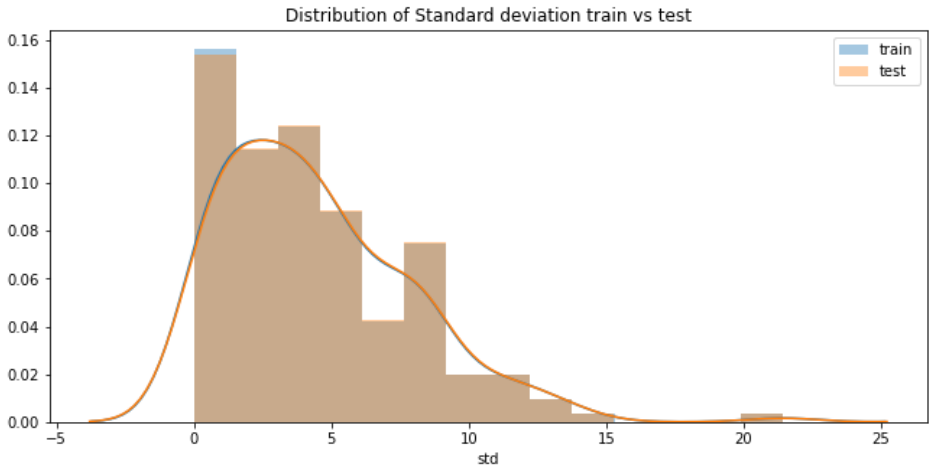


Following observation is made on the mean graphs:

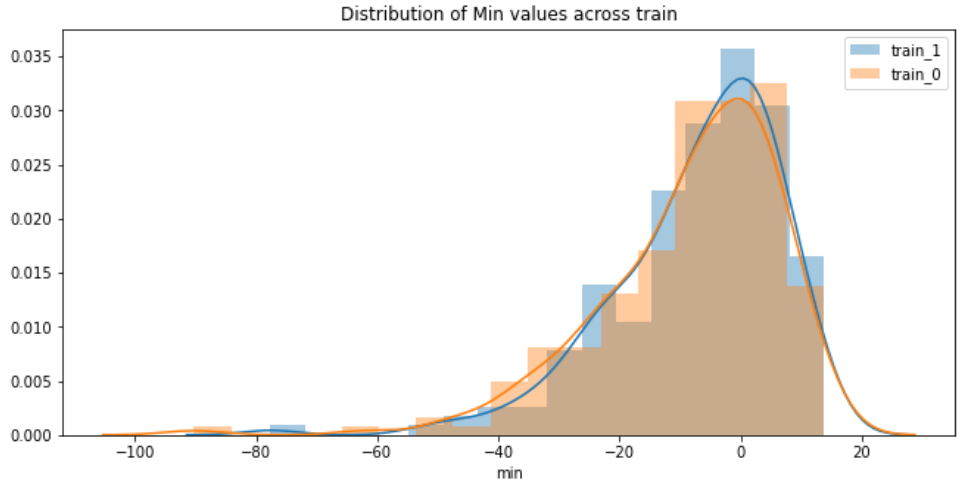
- Distribution of mean in train data across target 0 and 1
    - The column wise mean distribution is not Gaussian.
    - The majority of columns have the mean value between -10 and 20.
- Distribution of mean in train and test data
    - The distribution of mean looks identical in both train and test.

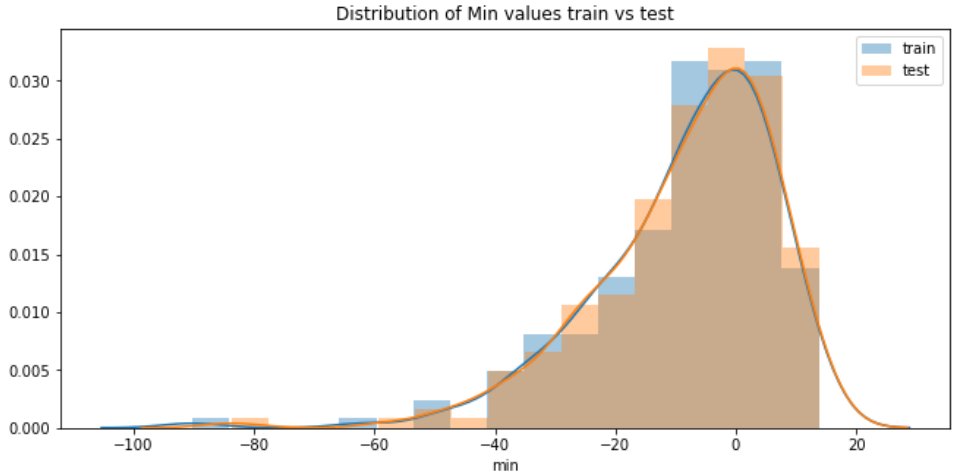
Distribution of Standard Deviation Across all variables in Train and test:



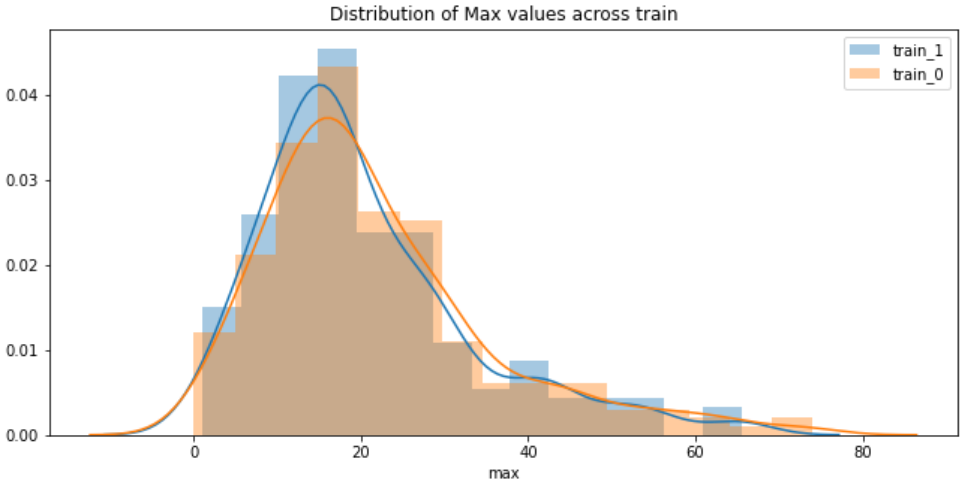


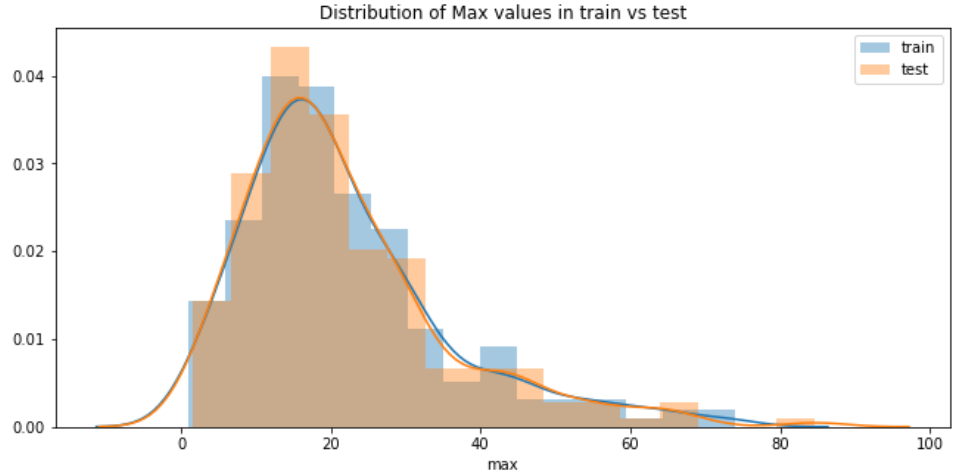
Distribution of Min values Across all variables in Train and test:





Distribution of Max values Across all variables in Train and test:

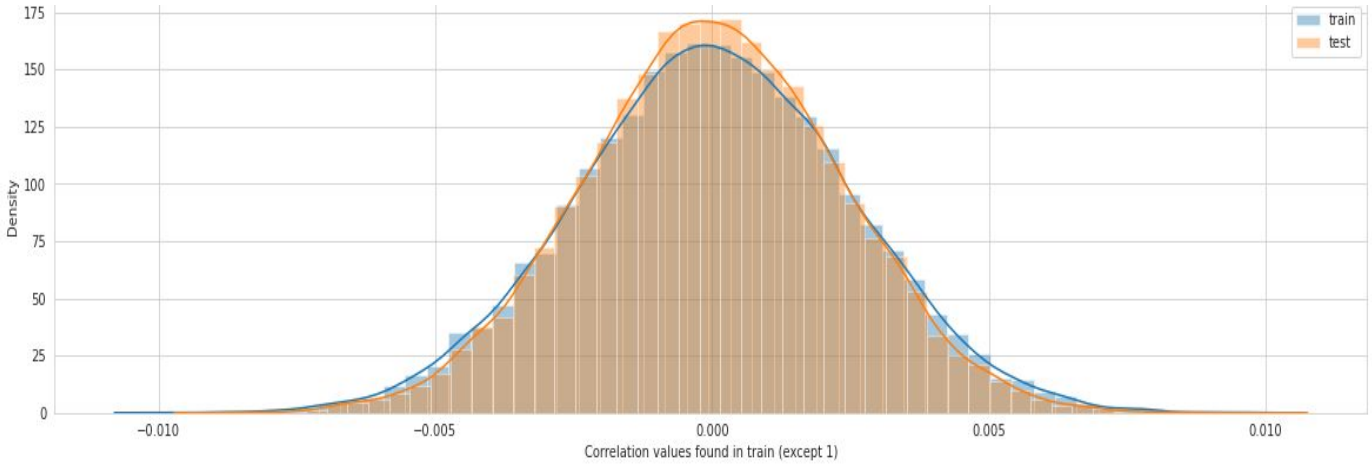




3.3 Correlation Analysis

Correlation analysis is used to find linear relationships between variables. Its values lie between -1 to 1 where 0 means no correlation and -1 means high negative linear relationship and 1 means high positive linear relationship.

Most variables that have correlation values are centered around zero as we can see in the distribution of correlation values of all variables below.



Most correlated variables

level_0	level_1	Cor_value
var_1	var_80	0.008855
var_80	var_1	0.008855
var_172	var_81	0.008936
var_81	var_172	0.008936
var_122	var_132	0.008956
var_132	var_122	0.008956
var_80	var_6	0.008958
var_6	var_80	0.008958
var_146	var_169	0.009071
var_169	var_146	0.009071
var_183	var_189	0.009359

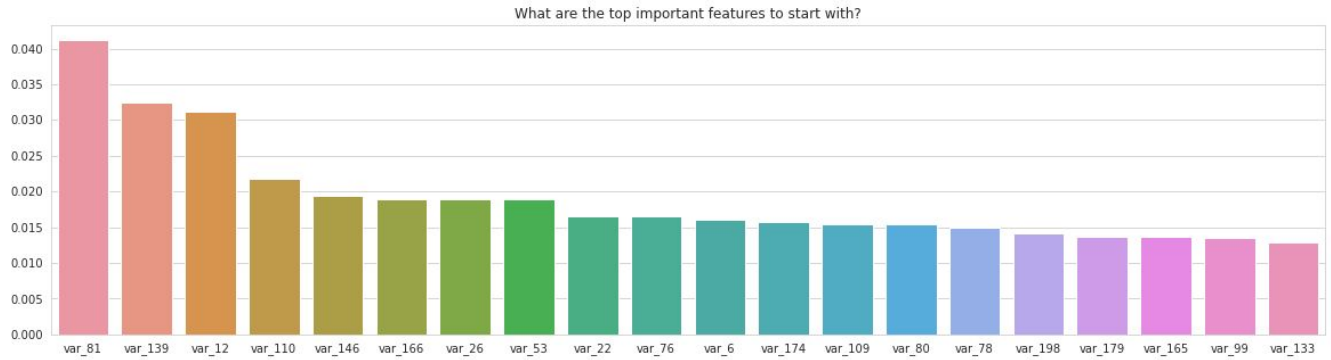
var_189	var_183	0.009359
var_174	var_81	0.00949
var_81	var_174	0.00949
var_81	var_165	0.009714
var_165	var_81	0.009714
var_53	var_148	0.009788
var_148	var_53	0.009788
var_26	var_139	0.009844
var_139	var_26	0.009844

Least correlated variables

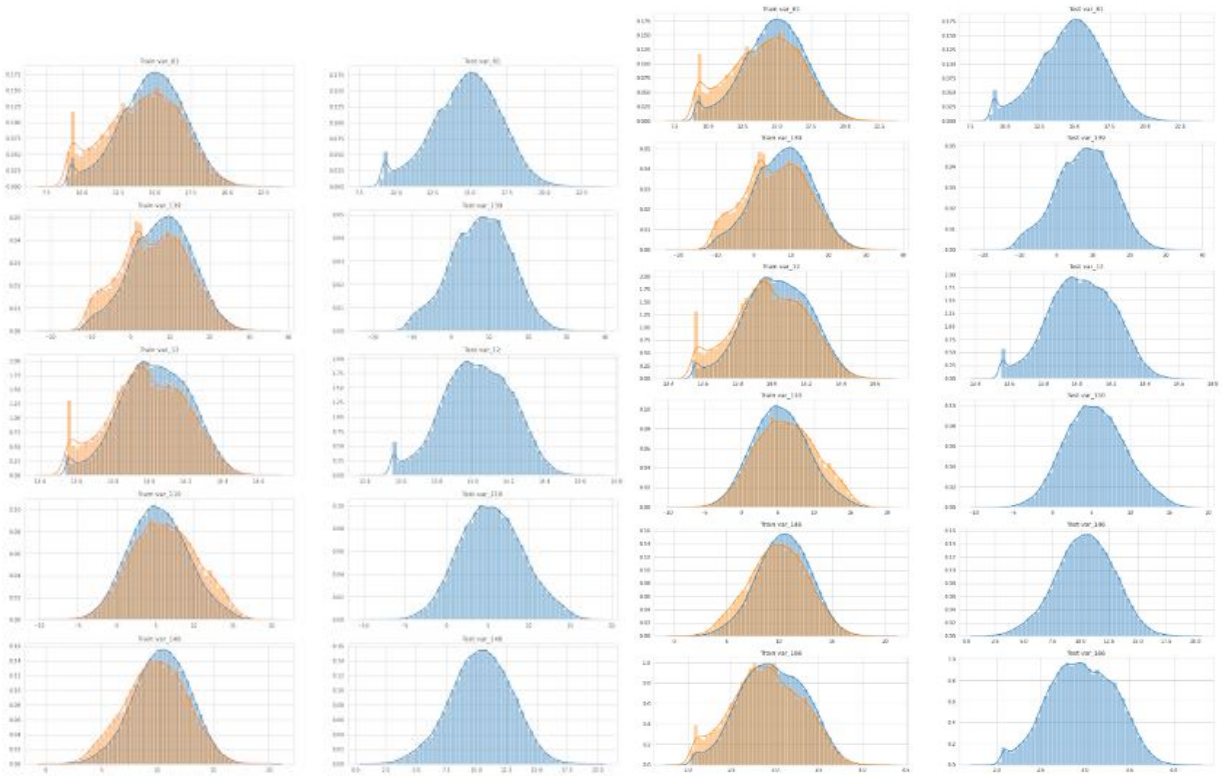
level_0	level_1	Cor_values
var_75	var_191	2.70E-08
var_191	var_75	2.70E-08
var_173	var_6	5.94E-08
var_6	var_173	5.94E-08
var_126	var_109	1.31E-07
var_109	var_126	1.31E-07
var_144	var_27	1.77E-07
var_27	var_144	1.77E-07
var_177	var_100	3.12E-07
var_100	var_177	3.12E-07

3.4 Finding Top Features

Using Random Forest



Exploring these top Features



Most top features have a bump either on the left and right side.

Weight of Evidence (WOE)

The weight of evidence tells the predictive power of an independent variable in relation to the dependent variable. Since it evolved from the credit scoring world, it is generally described as a measure of the separation of good and bad customers. "Bad Customers" refers to the customers who defaulted on a loan. and "Good Customers" refers to the customers who paid a bank loan.
Steps of Calculating WOE For a continuous variable, split data into 10 parts (or lesser depending on the distribution). Calculate the number of events and non-events in each group (bin) Calculate the % of events and % of non-events in each group. Calculate WOE by taking natural log of division of % of non-events and % of events

Information Value (IV)

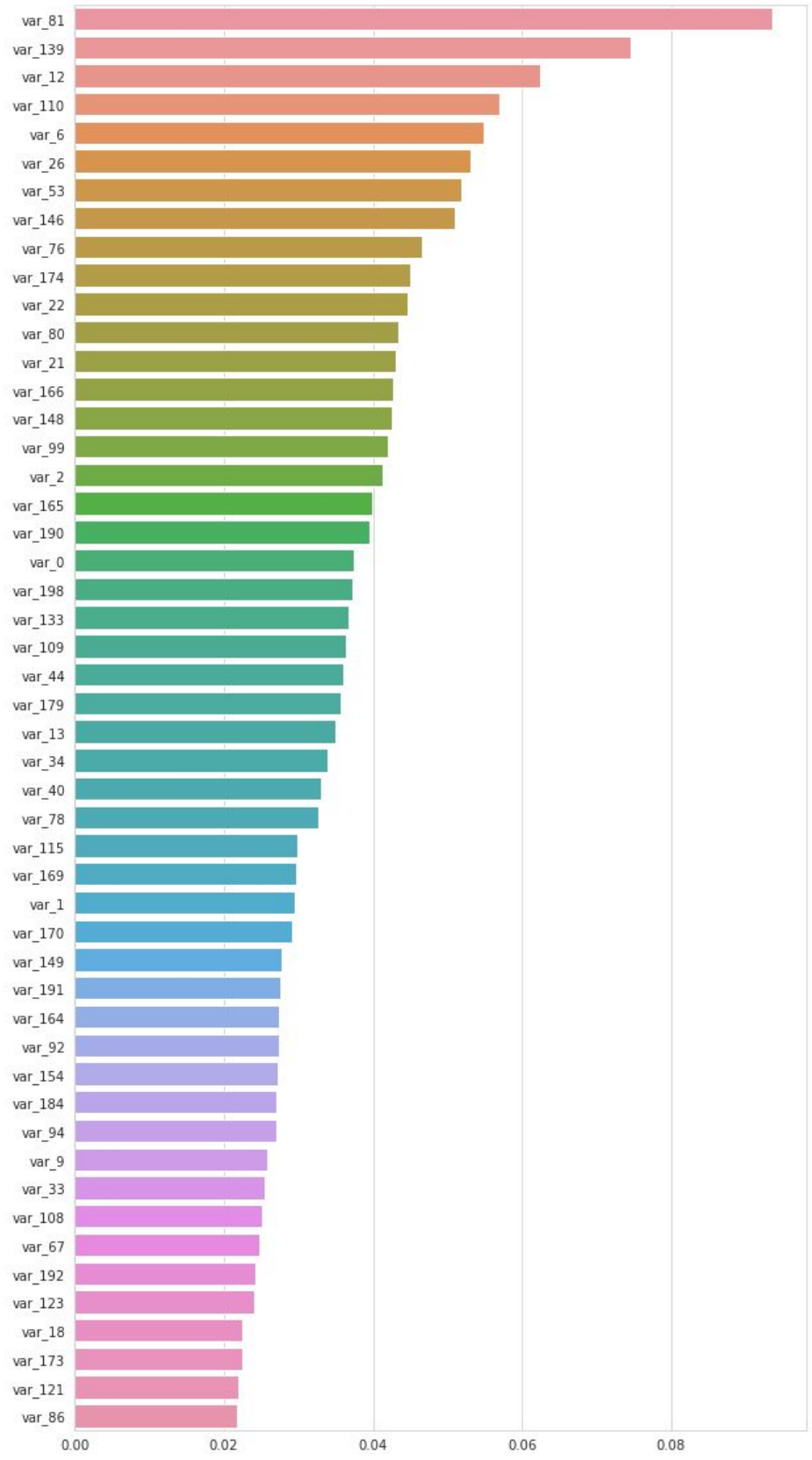
Information value is one of the most useful techniques to select important variables in a predictive model. It helps to rank variables on the basis of their importance. The IV is calculated using the following formula :

IV = ∑ (% of non-events - % of events) * WOE

Information Value Variable Predictiveness

- Less than 0.02 Not useful for prediction
- 0.02 to 0.1 Weak predictive Power
- 0.1 to 0.3 Medium predictive Power
- 0.3 to 0.5 Strong predictive Power
- 0.5 Suspicious Predictive Power

In next graph we can see the top features:


var_81
var_139
var_12
var_110
var_6
var_26
var_53
var_146
var_76
var_174
var_22
var_80
var_21
var_166
var_148
var_99
var_2
var_165
var_190
var_0
var_198
var_133
var_109
var_44
var_179
var_13
var_34
var_40
var_78
var_115
var_169
var_1
var_170
var_149
var_191
var_164
var_92
var_154
var_184
var_94
var_9
var_33
var_108
var_67
var_192
var_123
var_18
var_173
var_121
var_86
0.00
0.02
0.04
0.06
0.08

Top features:

	feature_name	iv_value	
0	var_81	0.09358
1	var_139	0.074582
2	var_12	0.062416
3	var_110	0.057046
4	var_6	0.054804
5	var_26	0.053017
6	var_53	0.051905
7	var_146	0.051071
8	var_76	0.046606
9	var_174	0.044973
10	var_22	0.044573
11	var_80	0.043412
12	var_21	0.0431
13	var_166	0.042635
14	var_148	0.042483
15	var_99	0.042041

Combined Top features from both methods:

```
['var_109', 'var_148',
 'var_198', 'var_81',
 'var_190', 'var_78',
 'var_133', 'var_0',
 'Target',  'var_166',
 'Var_21',  'var_99',
 'var_110', 'var_76',
 'var_6',   'var_2',
 'Var_12',  'var_22',
 'var_179', 'var_174',
 'Var_165', 'var_146',
 'Var_26',  'var_53',
      'Var_80',       ]
```

Dimensionality reduction using PCA?

We won't be using PCA because the correlation between variables is very weak and PCa required some highly correlated variables.

Feature Engineering:

We will be generating new features from Top features.

For generating new features we'll be using.

- Rounding variable values at different positions.
- quantile based binning
- Mean across the rows
- Median across the rows
- Min and max across the row
- Skewness and kurtosis
- Standard deviation

Dealing with class Imbalance :

We also have class imbalance problems in our target variable which can lead to overfitting.
So we will be using
Synthetic Minority Oversampling Technique(SMOTE) to deal with this problem.
So we will be using the Downsampling method to deal with class imbalance.

4 Modeling

Standardization

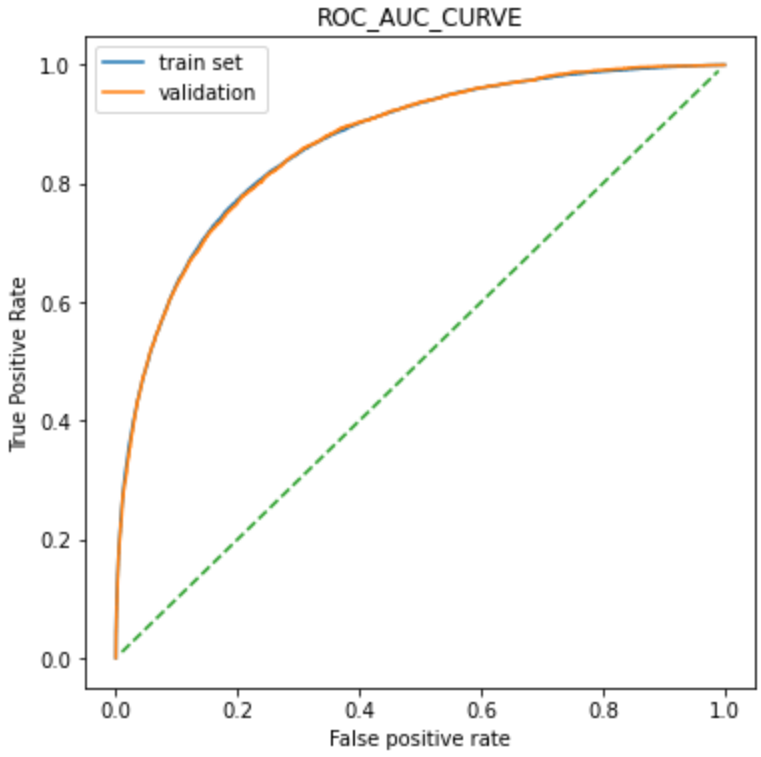
We will be standardising our data before modeling to making mean 0 and sd 1.

4.1 Logistic regression

It measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution.

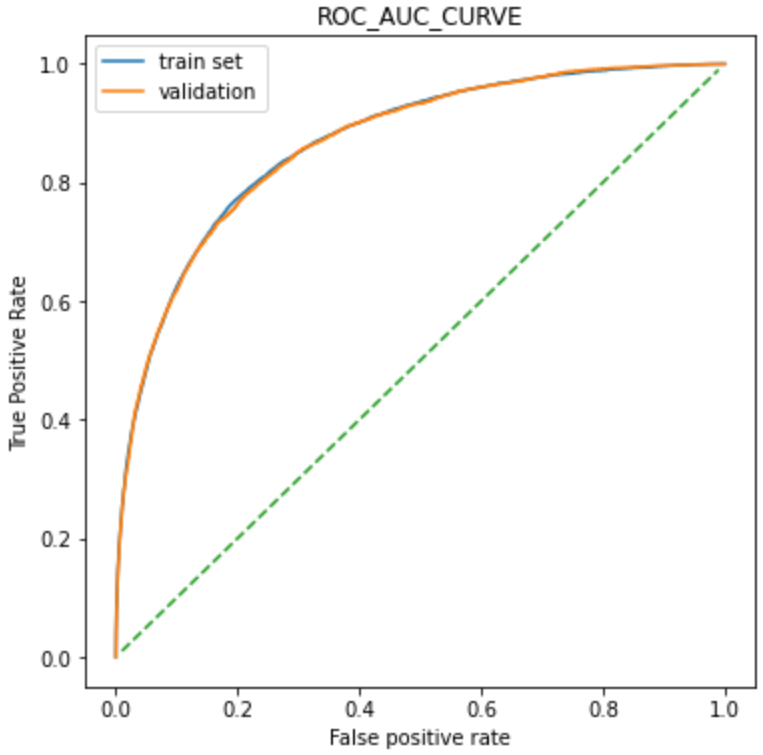
Logistic regression on normal data:

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.8675818764520795	0.4190689429984140	0.2979965503515988	0.7058453802639849	0.9169733333333333
Validation	0.8674203072849044	0.4111126701276835	0.2916003184713376	0.6966238706609605	0.91606



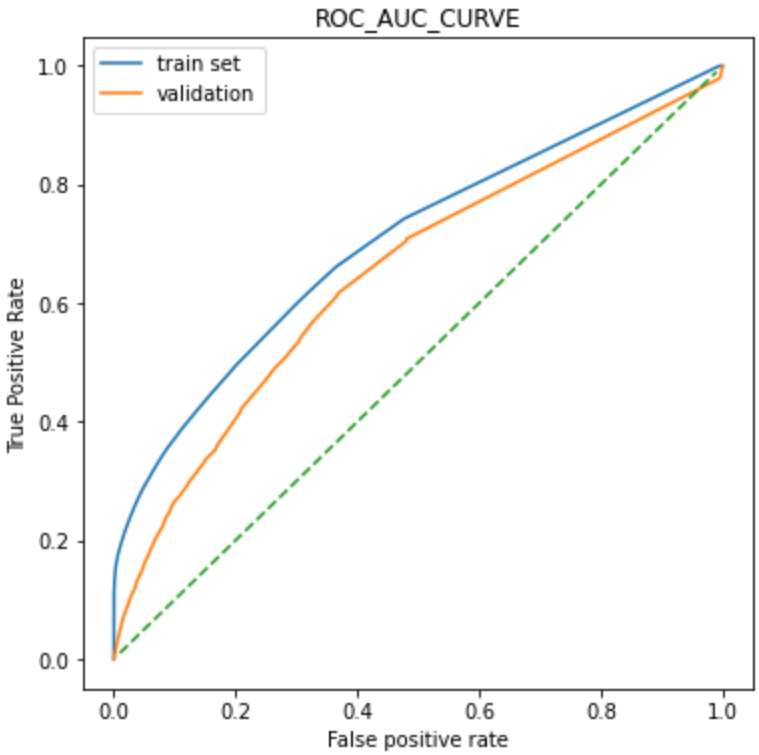
Logistic Regression on Downsampled Data

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.8667255566646745	0.42508623167576887	0.7848613506700278	0.2914757329391476	0.7866533333333333
Validation	0.8657783399200027	0.4206859592711682 5	0.78125	0.2878410090935758 5	0.7838



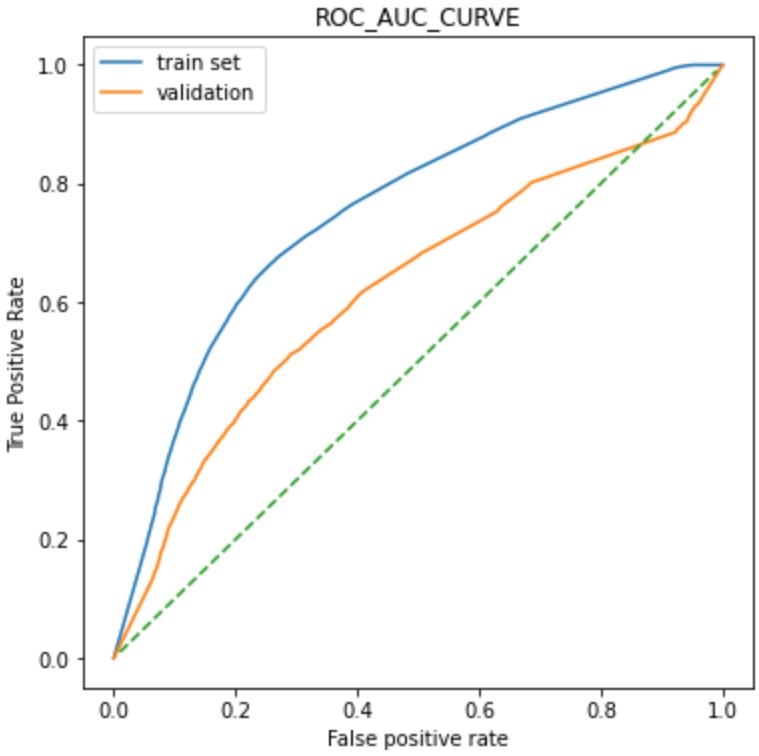
4.2 Decision tree

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.7022834295553622	0.2596846846846846	0.1529786387156936	0.8585256887565152	0.9123466666666666
Validation	0.6480554822975649	0.0982491925888152 3	0.0575238853503184 7	0.3364377182770663 7	0.8939



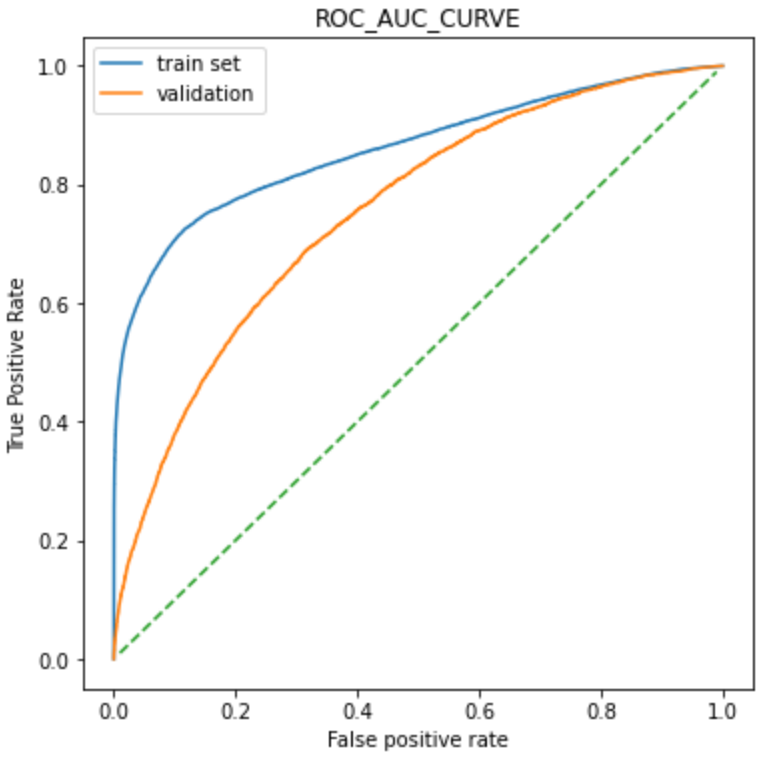
On downsampled data

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.7521637273824272	0.3425856995994186	0.6411038874883906	0.2337461300309 5976	0.7527333333333334
Validation	0.6201306655835694	0.2513476376704 3655	0.4733280254 77707	0.1711037559361059	0.71668


ROC_AUC_CURVE
train set
validation
True Positive Rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0

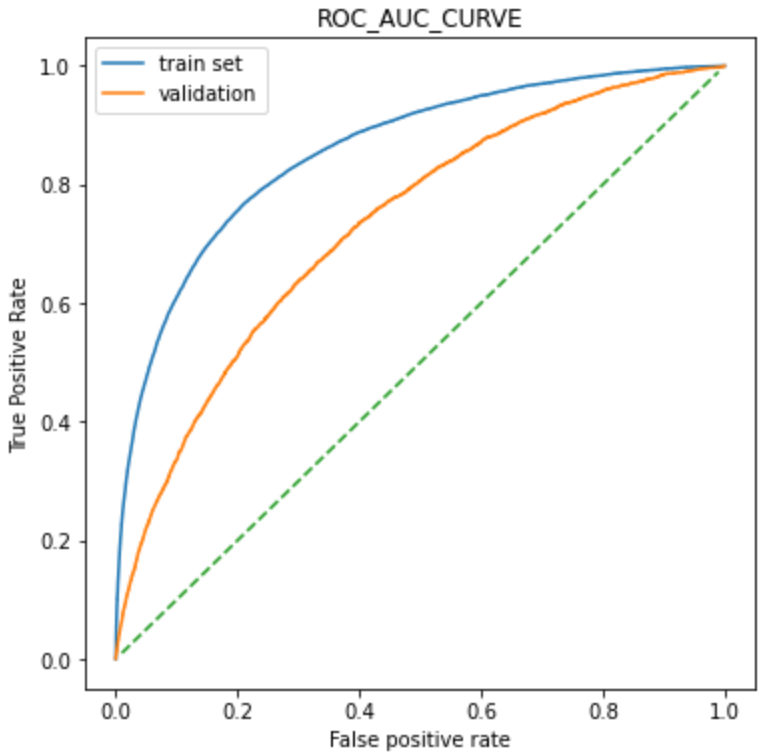
4.3 Random Forest

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.8598064130135 1	0.1456513716324271	0.0785458405201008 3	1.0	0.9074
Validation	0.7524006920817783	0.0114579217700513 62	0.0057722929936305 73	0.7631578947368421	0.89992



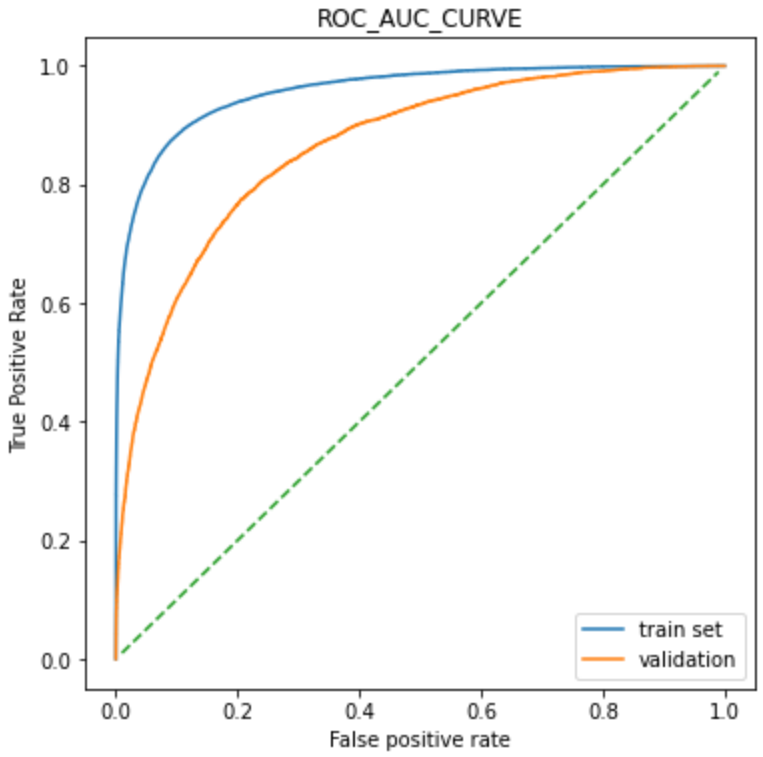
On Downsampled data

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.8561987575689796	0.3894289185905225 3	0.6411038874883906	0.2565404221257952	0.7454
Validation	0.7311687495716045	0.2962789169912252	0.4733280254777077	0.1938355743844784 3	0.70006



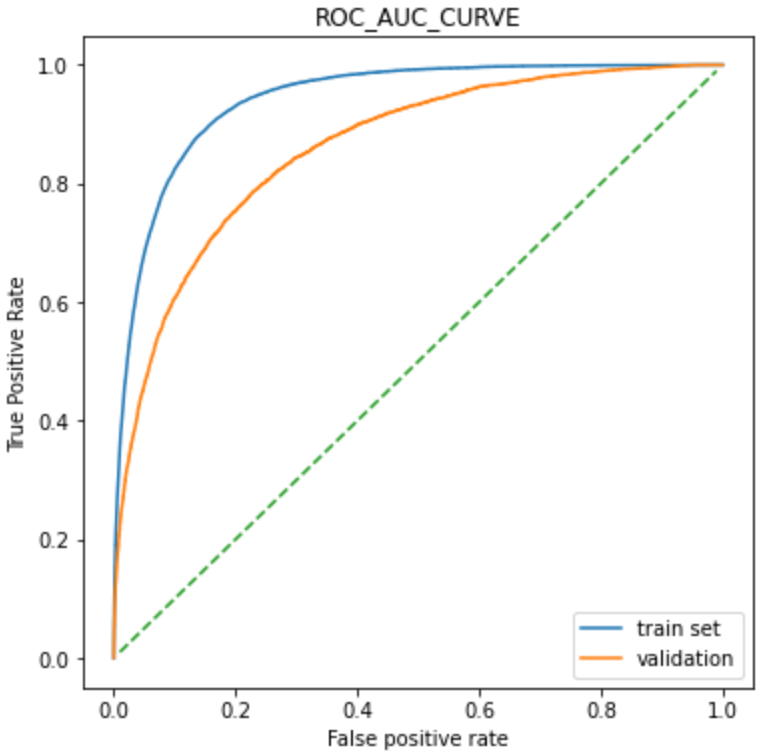
4.4 XG Boost

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.9577514866398906	0.6282205313042317	0.4682897704657025	0.9540478443032843	0.944
Validation	0.8636218022931409	0.3575685339690107	0.2388535031847338	0.7109004739336493	0.91376



On downsampled data

	Roc_Auc	F1-Score	Recall	Precision	Accuracy
Train data	0.9422084385673736	0.5084152367754052	0.9248374684887887	0.3505670530842155 4	0.8202733333333333
Validation	0.8604358121394398	0.4183169659812273	0.7673168789808917	0.2875363616021481	0.78558


ROC_AUC_CURVE
True Positive Rate
False positive rate
0.0
0.2
0.4
0.6
0.8
1.0
train set
validation

5 Conclusion

5.1 Model Evaluation

Error matrix

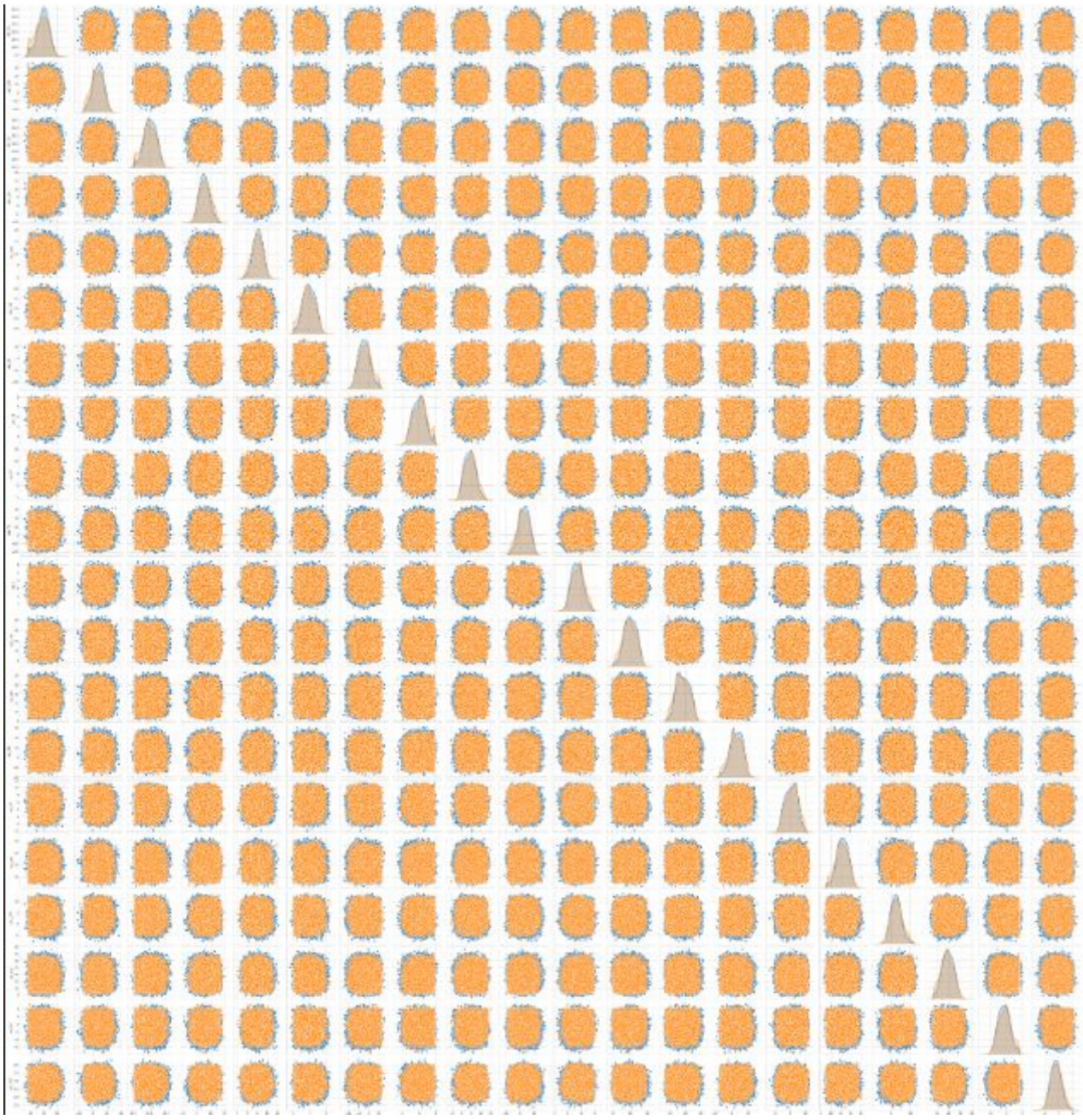
Error Metrics → Model ↓	Roc_auc	F1	Recall	Precision	Accuracy
Logistic Regression	0.8674203072849044	0.4111126701276835	0.2916003184713376	0.6966238706609605	0.91606
Decision tree	0.6480554822975649	0.09824919258881523	0.05752388535031847	0.3364377182770637	0.8939
Random Forest	0.7524006920817783	0.011457921770051362	0.005772292993630573	0.7631578947368421	0.89992
XG Boost	0.8636218022931409	0.3575685339690107	0.2388535031847338	0.7109004739336493	0.91376

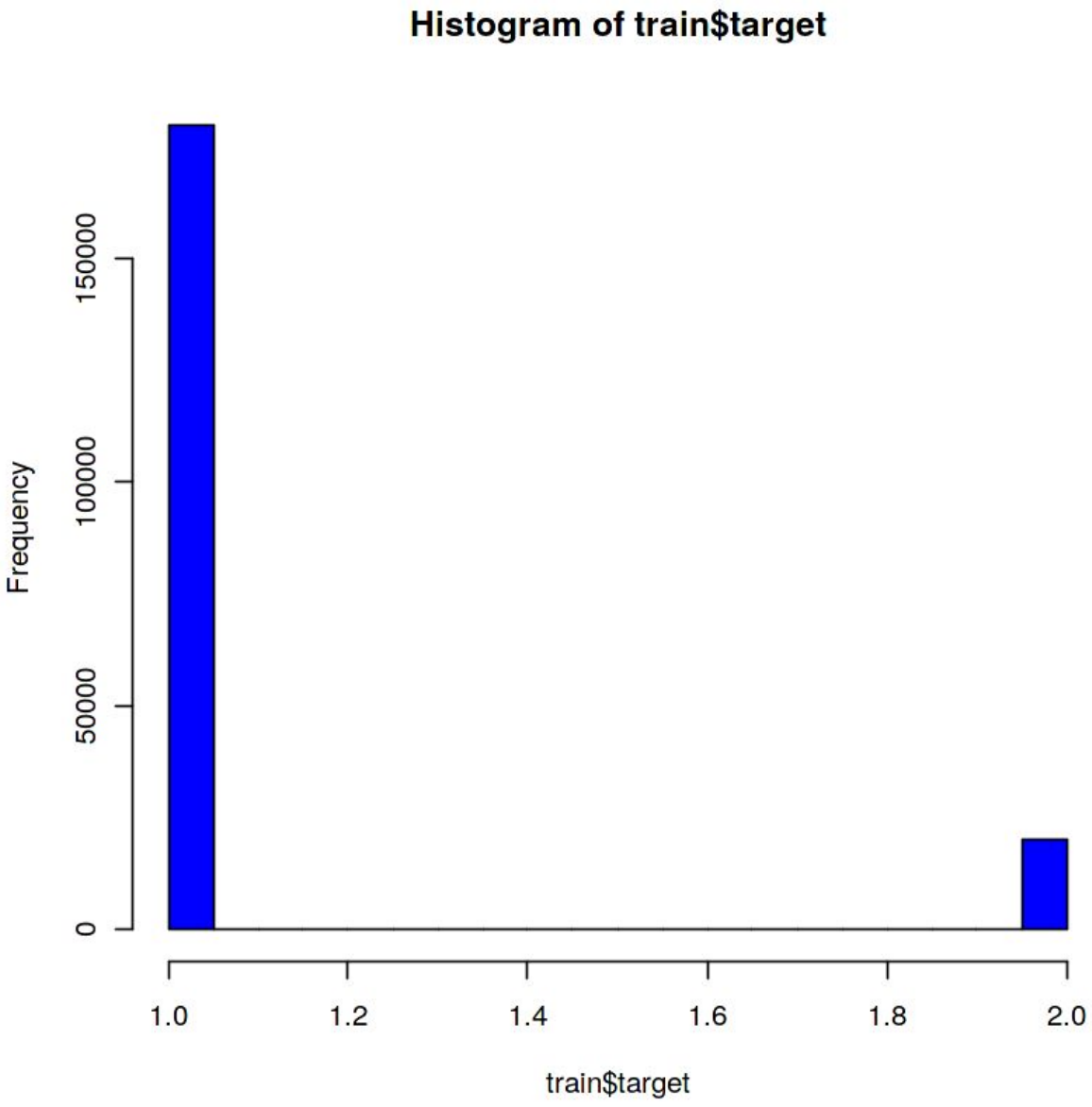
5.2 Model Selection

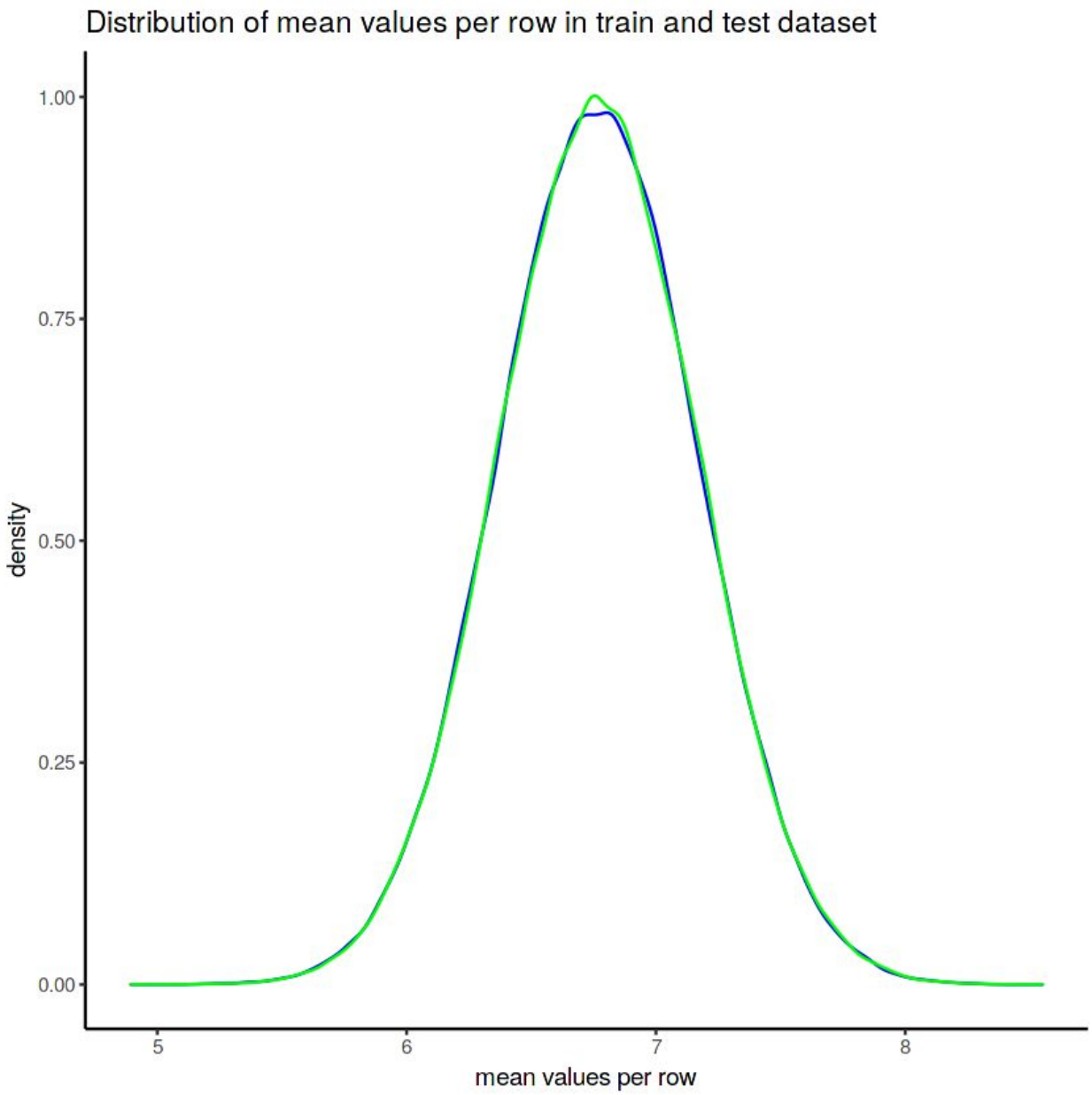
On basis of Roc, precision and Accuracy we will be choosing XG Boost

Error Metrics → Model ↓	Roc_auc	F1	Recall	Precision	Accuracy
XG Boost	0.8636218022931409	0.3575685339690107	0.2388535031847338	0.7109004739336493	0.91376

APPENDIX A EXTRA FIGURES




Histogram of train$target
Frequency
0
50000
100000
150000
1.0
1.2
1.4
1.6
1.8
2.0
train$target


Distribution of mean values per row in train and test dataset
density
1.00
0.75
0.50
0.25
0.00
5
6
7
8
mean values per row

APPENDIX - CODE

Python

```python
#!/usr/bin/env python
# coding: utf-8
# # Santander Customer Transaction Prediction
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
# sklearn models & tools
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.metrics import roc_auc_score
from sklearn.metrics import make_scorer
from sklearn.mixture import GaussianMixture
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA
train = pd.read_csv("/kaggle/input/customer-transaction-prediction/train.csv")
test = pd.read_csv("/kaggle/input/customer-transaction-prediction/test.csv")

train.head()
train.info()
test.info()
train.dtypes.value_counts()
test.dtypes.value_counts()
train.select_dtypes(include=['object'])
train.nunique(axis=0).sort_values()
test.nunique(axis=0).sort_values()
train.isna().sum().sum()
test.isna().sum().sum()#
# # EDA
# ## Descriptive analysis
train_summary = train.describe()
train_summary
```

```
train_summary_1 = train[train['target'] == 1].describe()
Train_summary_1
train_summary_0 = train[train['target'] == 0].describe()
Train_summary_0
test_summary = test.describe()
test_summary
fig = plt.figure(figsize=(25,5))
fig.subplots_adjust(hspace=0.3, wspace=0.3, )
ax = fig.add_subplot(1, 2, 1)
sns.distplot(train_summary_1.iloc[1], label="train_1" )
sns.distplot(train_summary_0.iloc[1], label="train_0" )
plt.title("Distribution of mean across train")
plt.legend()
ax = fig.add_subplot(1, 2, 2)
sns.distplot(train_summary.iloc[1], label="train")
sns.distplot(test_summary.iloc[1], label="test" )
plt.title("Distribution of mean train vs test")
plt.legend();

fig = plt.figure(figsize=(25,5))
fig.subplots_adjust(hspace=0.3, wspace=0.3, )
ax = fig.add_subplot(1, 2, 1)
sns.distplot(train_summary_1.iloc[2], label="train_1" )
sns.distplot(train_summary_0.iloc[2], label="train_0" )
plt.title("Distribution of Standard deviation across train")
plt.legend()
ax = fig.add_subplot(1, 2, 2)
sns.distplot(train_summary.iloc[2], label="train")
sns.distplot(test_summary.iloc[2], label="test" )
plt.title("Distribution of Standard deviation train vs test")
plt.legend();


fig = plt.figure(figsize=(25,5))
fig.subplots_adjust(hspace=0.3, wspace=0.3, )
ax = fig.add_subplot(1, 2, 1)
sns.distplot(train_summary_1.iloc[3], label="train_1" )
sns.distplot(train_summary_0.iloc[3], label="train_0" )
plt.title("Distribution of Min values across train")
```

```
plt.legend()
ax = fig.add_subplot(1, 2, 2)
sns.distplot(train_summary.iloc[3], label="train")
sns.distplot(test_summary.iloc[3], label="test" )
plt.title("Distribution of Min values train vs test")
plt.legend();
fig = plt.figure(figsize=(25,5))
fig.subplots_adjust(hspace=0.3, wspace=0.3, )
ax = fig.add_subplot(1, 2, 1)
sns.distplot(train_summary_1.iloc[7], label="train_1" )
sns.distplot(train_summary_0.iloc[7], label="train_0" )
plt.title("Distribution of Max values across train")
plt.legend()
ax = fig.add_subplot(1, 2, 2)
sns.distplot(train_summary.iloc[7], label="train")
sns.distplot(test_summary.iloc[7], label="test" )
plt.title("Distribution of Max values in train vs test")
plt.legend();

features = train.columns.values[2:202]
# # Univariate analysis
# ### Target class
sns.countplot(train.target)
def density_plot(df1,df2,feat):
    plt.figure()
    fig,ax=plt.subplots(10,10,figsize=(20,20))
       for i,f in enumerate(feat):
           plt.subplot(10,10,i+1)
           sns.set_style('whitegrid')
           sns.kdeplot(df1[f])
           sns.kdeplot(df2[f])
           plt.legend(["class_0","class_1"])
           plt.xlabel(f,fontsize=9)
       plt.show()
  train.columns

  density_plot(train.loc[train.target==0],train.loc[train.target==1],train.colu
mns[2:102])

```

```
112.
113.  density_plot(train.loc[train.target==0],train.loc[train.target==1],train.colu
     mns[102:])
114.  def density_plt(df,feat):
115.      plt.figure()
116.      fig,ax=plt.subplots(10,10,figsize=(25,25))
117.      for i,f in enumerate(feat):
118.          plt.subplot(10,10,i+1)
119.          sns.kdeplot(df[f])
120.          plt.xlabel(f,fontsize=9)
121.      plt.show()
122.  density_plt(test,test.columns[1:101])
123.  density_plt(test,test.columns[101:])
124.  fig = plt.figure(figsize=(25,10))
125.  fig.subplots_adjust(hspace=0.3, wspace=0.3, )
126.  ax = fig.add_subplot(2,3,1)
127.  sns.kdeplot(train.loc[train.target==0]['var_0'])
128.  sns.kdeplot(train.loc[train.target==1]['var_0'])
129.  plt.legend(["class_0","class_1"])
130.
131.  ax = fig.add_subplot(2,3,2)
132.  sns.kdeplot(train.loc[train.target==0]['var_2'])
133.  sns.kdeplot(train.loc[train.target==1]['var_2'])
134.  plt.legend(["class_0","class_1"])
135.
136.  ax = fig.add_subplot(2,3,3)
137.  sns.kdeplot(train.loc[train.target==0]['var_6'])
138.  sns.kdeplot(train.loc[train.target==1]['var_6'])
139.  plt.legend(["class_0","class_1"])
140.
141.  ax = fig.add_subplot(2,3,4)
142.  sns.kdeplot(train.loc[train.target==0]['var_9'])
143.  sns.kdeplot(train.loc[train.target==1]['var_9'])
144.  plt.legend(["class_0","class_1"])
145.
146.  ax = fig.add_subplot(2,3,5)
147.  sns.kdeplot(train.loc[train.target==0]['var_53'])
148.  sns.kdeplot(train.loc[train.target==1]['var_53'])
149.  plt.legend(["class_0","class_1"])
```

```
150.
151.    ax = fig.add_subplot(2,3,6)
152.    sns.kdeplot(train.loc[train.target==0]['var_99'])
153.    sns.kdeplot(train.loc[train.target==1]['var_99'])
154.    plt.legend(["class_0","class_1"])
155.    # ## Finding Important Features
156.    # ### Correlation analysis
157.    correlations =
    train[features].corr().abs().unstack().sort_values(kind="quicksort").reset_index
    ()
158.    correlations = correlations[correlations['level_0'] !=
    correlations['level_1']]
159.    # #### Most correlated features
160.    correlations.tail(20)
161.    # #### least correlated
162.    correlations.head(10)
163.    train_correlations = train.drop(["target"], axis=1).corr()
164.    train_correlations = train_correlations.values.flatten()
165.    train_correlations = train_correlations[train_correlations != 1]
166.
167.    test_correlations = test.corr()
168.    test_correlations = test_correlations.values.flatten()
169.    test_correlations = test_correlations[test_correlations != 1]
170.    plt.figure(figsize=(20,5))
171.    sns.distplot(train_correlations,  label="train")
172.    sns.distplot(test_correlations,  label="test")
173.    plt.xlabel("Correlation values found in train (except 1)")
174.    plt.ylabel("Density")
175.    plt.title("Is there correlation between features?");
176.    plt.legend();
177.    train["Id"] = train.index.values
178.    original_trainid = train.ID_code.values
179.
180.    train.drop("ID_code", axis=1, inplace=True)
181.    train.head()
182.    # ### Top features through random Forest
183.    parameters = {'min_samples_leaf': [20, 25]}
184.    forest = RandomForestClassifier(max_depth=15, n_estimators=15)
```

```
185.    grid = GridSearchCV(forest, parameters, cv=3, n_jobs=-1, verbose=2,
    scoring=make_scorer(roc_auc_score))
186.    grid.fit(train.drop(["target",], axis=1).values, train.target.values)
187.    grid.best_score_
188.    grid.best_params_
189.    n_top = 20
190.    importances = grid.best_estimator_.feature_importances_
191.    idx = np.argsort(importances)[::-1][0:n_top]
192.    feature_names = train.drop("target", axis=1).columns.values
193.
194.    plt.figure(figsize=(20,5))
195.    sns.barplot(x=feature_names[idx], y=importances[idx]);
196.    plt.title("What are the top important features to start with?");
197.    fig, ax = plt.subplots(n_top,2,figsize=(20,5*n_top))
198.
199.    for n in range(n_top):
200.        sns.distplot(train.loc[train.target==0, feature_names[idx][n]],
    ax=ax[n,0],  norm_hist=True)
201.        sns.distplot(train.loc[train.target==1, feature_names[idx][n]],
    ax=ax[n,0],  norm_hist=True)
202.        sns.distplot(test.loc[:, feature_names[idx][n]], ax=ax[n,1],
    norm_hist=True)
203.        ax[n,0].set_title("Train {}".format(feature_names[idx][n]))
204.        ax[n,1].set_title("Test {}".format(feature_names[idx][n]))
205.        ax[n,0].set_xlabel("")
206.        ax[n,1].set_xlabel("")
207.    top = train.loc[:, feature_names[idx]]
208.    top.describe()
209.    top = top.join(train.target)
210.    sns.pairplot(top, hue="target")
211.    def woe(X, y):
212.        tmp = pd.DataFrame()
213.        tmp["variable"] = X
214.        tmp["target"] = y
215.        var_counts = tmp.groupby("variable")["target"].count()
216.        var_events = tmp.groupby("variable")["target"].sum()
217.        var_nonevents = var_counts - var_events
218.        tmp["var_counts"] = tmp.variable.map(var_counts).astype("float64")
219.        tmp["var_events"] = tmp.variable.map(var_events).astype("float64")
```

```
        tmp["var_nonevents"] = tmp.variable.map(var_nonevents).astype("float64")
        events = sum(tmp["target"] == 1)
        nonevents = sum(tmp["target"] == 0)
        tmp["woe"] = np.log(((tmp["var_nonevents"])/nonevents)/((tmp["var_events"])/events))
        tmp["woe"] = tmp["woe"].replace(np.inf, 0).replace(-np.inf, 0)
        tmp["iv"] = (tmp["var_nonevents"]/nonevents - tmp["var_events"]/events) * tmp["woe"]
        iv = tmp.groupby("variable")["iv"].last().sum()
        return tmp["woe"], tmp["iv"], iv
    iv_values = []
    feats = ["var_{}".format(i) for i in range(200)]
    y = train["target"]
    for f in feats:
        X = pd.qcut(train[f], 10, duplicates='drop')
        _, _, iv = woe(X, y)
        iv_values.append(iv)

    iv_inds = np.argsort(iv_values)[::-1][:50]
    iv_values = np.array(iv_values)[iv_inds]
    feats = np.array(feats)[iv_inds]
     plt.figure(figsize=(10, 20))
    sns.barplot(y=feats, x=iv_values, orient='h')
    plt.show()

    imp = {}
    imp["feature_name"] = feats
    imp["iv_value"] = iv_values
    imp_features = pd.DataFrame(data=imp)
    imp_features.sort_values(by = ["iv_value"],ascending=False)
    # # Feature Engineering
    top_features = imp_features["feature_name"].iloc[0:20].values
    top_features = top_features.tolist()
    top_features
    # adding feature from top to top_features
    for i in top.columns:
        top_features.append(i)

    #removing duplicates
```

```
top_features = list(set(top_features))
top_features
new_top = train[top_features]
new_top.head()
new_top.shape
encoder = LabelEncoder()
for your_feature in top_features:
    if(your_feature == "target"):
        pass
    elif(your_feature == "ID"):
        pass
    else:
        train[your_feature + "_qbinned"] = pd.qcut(
            train.loc[:, your_feature].values,
            q=10,
            duplicates='drop',
            labels=False
        )
        train[your_feature + "_qbinned"] = encoder.fit_transform(
            train[your_feature + "_qbinned"].values.reshape(-1, 1)
        )


        train[your_feature + "_rounded"] = np.round(train.loc[:,
your_feature].values)
        train[your_feature + "_rounded_10"] = np.round(10*train.loc[:,
your_feature].values)
        train[your_feature + "_rounded_100"] = np.round(100*train.loc[:,
your_feature].values)
encoder = LabelEncoder()
for your_feature in top_features:
    if(your_feature == "target"):
        pass
    else:
        test[your_feature + "_qbinned"] = pd.qcut(
            test.loc[:, your_feature].values,
            q=10,
            duplicates='drop',
            labels=False
```

```
            )
            test[your_feature + "_qbinned"] = encoder.fit_transform(
                test[your_feature + "_qbinned"].values.reshape(-1, 1)
            )


            test[your_feature + "_rounded"] = np.round(test.loc[:,
    your_feature].values)
            test[your_feature + "_rounded_10"] = np.round(10*test.loc[:,
    your_feature].values)
            test[your_feature + "_rounded_100"] = np.round(100*test.loc[:,
    your_feature].values)
    train.head()
    # Deriving the New Features for the training dataset
    mean_r=train.iloc[:,2:].mean(axis=1).values
    min_r=train.iloc[:,2:].min(axis=1).values
    max_r=train.iloc[:,2:].max(axis=1).values
    std_r=train.iloc[:,2:].std(axis=1).values
    skew_r=train.iloc[:,2:].skew(axis=1).values
    kurto_r=train.iloc[:,2:].kurtosis(axis=1).values
    median_r=train.iloc[:,2:].median(axis=1).values
    ## Adding the new features to the training dataset
    train["mean_r"]=mean_r
    train["min_r"]=min_r
    train["max_r"]=max_r
    train["std_r"]=std_r
    train["skew_r"]=skew_r
    train["kurto_r"]=kurto_r
    train["median_r"]=median_r
    # Deriving the New Features for the testing dataset
    mean_r=test.iloc[:,2:].mean(axis=1).values
    min_r=test.iloc[:,2:].min(axis=1).values
    max_r=test.iloc[:,2:].max(axis=1).values
    std_r=test.iloc[:,2:].std(axis=1).values
    skew_r=test.iloc[:,2:].skew(axis=1).values
    kurto_r=test.iloc[:,2:].kurtosis(axis=1).values
    median_r=test.iloc[:,2:].median(axis=1).values

    ## Adding the new features to the testing dataset
```

```
test["mean_r"]=mean_r
test["min_r"]=min_r
test["max_r"]=max_r
test["std_r"]=std_r
test["skew_r"]=skew_r
test["kurto_r"]=kurto_r
test["median_r"]=median_r

for i in train.columns.values:
    if i not in test.columns.values:
        print( i)
# # New feature importances

iv_values = []
feats = train.columns.values
y = train["target"]
for f in feats:
    X = pd.qcut(train[f], 10, duplicates='drop')
    _, _, iv = woe(X, y)
    iv_values.append(iv)

iv_inds = np.argsort(iv_values)[::-1][:50]
iv_values = np.array(iv_values)[iv_inds]
feats = np.array(feats)[iv_inds]
plt.figure(figsize=(10, 20))
sns.barplot(y=feats, x=iv_values, orient='h')
plt.show()
```

CODE-R

```
#Loading the library
library(DMwR)
library(rpart)
library(ggplot2)
library(corrgram)
library(ggpubr)
library(C50)
library(MASS)
library(dplyr)
library(Metrics)
library(randomForest)
library(ggthemes)
```

```
library(glmnet)

#loading the train and test data
train = read.csv("../input/customer-transaction-prediction/train.csv", header = T ,
stringsAsFactors = F)
test = read.csv("../input/customer-transaction-prediction/test.csv", header = T ,
stringsAsFactors = F)

#let's look at names of the 2 datasets
names(train)
names(test)

str(train)

str(test)

summary(train)

summary(test)

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

str(train)

#lets check the count of target classes
require(gridExtra)
table(train$target)

#percentage count of target classes
table(train$target)/length(train$target)*100

#histogram of train data
train$target= as.numeric(train$target)
hist(train$target , col = "blue")

#distribution of mean values per row and column in train and test dataset
#Applying the function to find mean values per row in train and test data.
train_mean = apply(train[,-c(1,2)],MARGIN=1,FUN=mean)
test_mean = apply(test[,-c(1)],MARGIN=1,FUN=mean)

ggplot()+
  #Distribution of mean values per row in train data

geom_density(data=train[,-c(1,2)],aes(x=train_mean),kernel='gaussian',show.legend=TRUE,c
olor='blue')+theme_classic()+
  #Distribution of mean values per row in test data
```

```
geom_density(data=test[,-c(1)],aes(x=test_mean),kernel='gaussian',show.legend=TRUE,color
='green')+
  labs(x='mean values per row',title="Distribution of mean values per row in train and test
dataset")

train_mean = apply(train[,-c(1,2)],MARGIN=2,FUN=mean)
test_mean = apply(test[,-c(1)],MARGIN=2,FUN=mean)

ggplot()+
  #Distribution of mean values per column in train data

geom_density(aes(x=train_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme
_classic()+
  #Distribution of mean values per column in test data
  geom_density(aes(x=test_mean),kernel='gaussian',show.legend=TRUE,color='green')+
  labs(x='mean values per column',title="Distribution of mean values per row in train and
test dataset")

train_sd = apply(train[,-c(1,2)],MARGIN=2,FUN=sd)
test_sd = apply(test[,-c(1)],MARGIN=2,FUN=sd)
ggplot()+
  #Distribution of sd values per column in train data

geom_density(aes(x=train_sd),kernel='gaussian',show.legend=TRUE,color='red')+theme_clas
sic()+
  #Distribution of sd values per column in test data
  geom_density(aes(x=test_sd),kernel='gaussian',show.legend=TRUE,color='blue')+
  labs(x='sd values per column',title="Distribution of std values per column in train and test
dataset")

sum(is.na(train))
sum(is.na(test))
#removing variable "ID_Code" from train and test dataset
train = train[, -1, drop = FALSE]
test = test[, -1, drop = FALSE]

train[] = lapply(train[], as.numeric)

#Correlation in train data
#convert factor to int
train$target = as.numeric(train$target)
train_correlations = cor(train[,c(1:201)])
train_correlations

#Correlation in test data
test_correlations = cor(test[,c(1:200)])
test_correlations
```

```
########variable imortance############
#variable importance is used to see top features in the dataset
#lets build a simple model to find important features
#Split the training data using simple random sampling
df_index = sample(1:nrow(train),0.75*nrow(train))
#train data
train_data = train[df_index,]
#validation data
valid_data = train[-df_index,]
#dimension of train and validation data
dim(train_data)
dim(valid_data)

#Training the Random forest classifier
set.seed(2732)
#convert to int to factor
# train_data$target = as.factor(train_data$target)
#setting the mtry
mtry<-floor(sqrt(200))
#setting the tunegrid
tuneGrid<-expand.grid(.mtry=mtry)


#fitting the random forest
rf = randomForest(target~.,train_data,mtry=mtry,ntree=10,importance=TRUE)

#Variable importance
Imp_feats = importance(rf,type=2)
Imp_feats

set.seed(689)

train.index = sample(1:nrow(train),0.8*nrow(train))
#train data
train.data = train[train.index,]
#validation data
valid.data = train[-train.index,]

#dimension of train data
dim(train.data)
#dimension of validation data
dim(valid.data)

table(train.data$target)
table(valid.data$target)

#Training dataset
```

```
X_t = as.matrix(train.data[,-c(1,2)])
y_t = as.matrix(train.data$target)
#validation dataset
X_v = as.matrix(valid.data[,-c(1,2)])
y_v = as.matrix(valid.data$target)
#test dataset
test = as.matrix(test[,-c(1)])

install.packages("glmnet", dependencies=TRUE)

set.seed(667) # to reproduce results
lr_model  = glmnet(X_t,y_t, family = "binomial")
summary(lr_model)

#Cross validation prediction
set.seed(8909)
cv_lr = cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
cv_lr

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

#Model performance on validation dataset
set.seed(5363)
cv_predict.lr = predict(cv_lr,X_v,s = "lambda.min", type = "class")
cv_predict.lr

#Confusion matrix
set.seed(689)
#actual target variable
target = valid.data$target
#convert to factor
target = as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr = as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)


#ROC_AUC score and curve
set.seed(892)
cv_predict.lr = as.numeric(cv_predict.lr)
roc(data=valid.data[,-c(1,2)],response=target,predictor=cv_predict.lr,auc=TRUE,plot=TRUE)
```

Reference

Stackoverflow.com

Edwisor.com

https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html