**Santander Customer Transaction Prediction**

Submitted By

***Sneha Potghan***

Contents

[Introduction 3](#_Toc46326778)

[1.1 Problem Statement 3](#_Toc46326779)

[1.2 Data 4](#_Toc46326780)

[Methodology 6](#_Toc46326781)

[2.1 Exploratory Data Analysis (EDA) 6](#_Toc46326782)

[2.1.1 Target classes count 6](#_Toc46326783)

[2.1.1 Attributes distributions and trends 6](#_Toc46326784)

[2.1.2 Missing Value Analysis 12](#_Toc46326785)

[2.1.2 Outlier Analysis 13](#_Toc46326786)

[2.1.3 Feature Selection 13](#_Toc46326787)

[2.1.4 Feature engineering 13](#_Toc46326788)

[Modelling 17](#_Toc46326789)

[3.1 Model Selection 17](#_Toc46326790)

[3.1.1 Logistic Regression 17](#_Toc46326791)

[3.1.2 LightGBM 21](#_Toc46326793)

[Model Evaluation 24](#_Toc46326794)

[Conclusion 29](#_Toc46326795)

[Appendix 30](#_Toc46326796)

[Python Code 30](#_Toc46326797)

[R CODE 43](#_Toc46326804)

Chapter 1

# Introduction

## 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.

## Data

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:

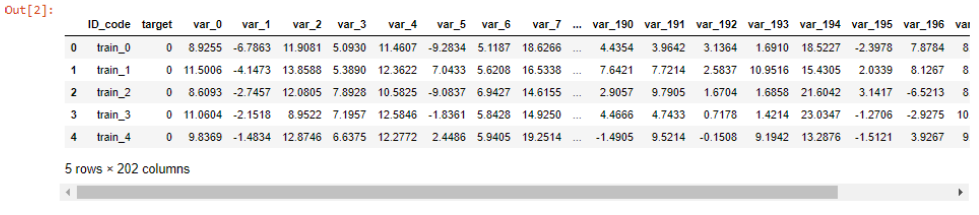


Table 1 Train Dataset

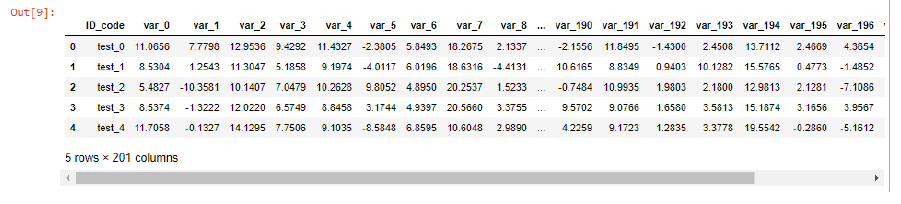


Table 2 Test Dataset

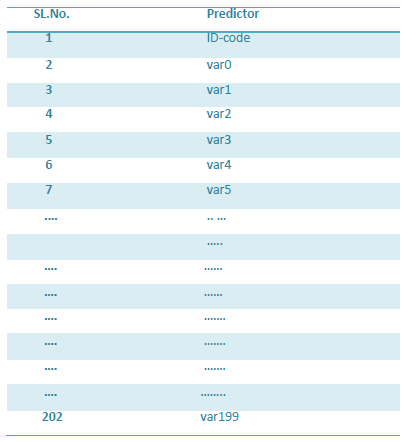


Table 3 Predictor Variable

Chapter 2

# Methodology

## 2.1 Exploratory Data Analysis (EDA)

Exploratory data analysis is one of the most important steps in data mining in order to know features of data. It involves the loading dataset, target classes count, data cleaning, typecasting of attributes, missing value analysis, Attributes distributions and trends. So, we have to clean the data. Otherwise it will have impact on performance of the model. Now we are going to explain one by one as follows. In this EDA I explained with seaborn visualizations.

## 2.1.1 Target classes count

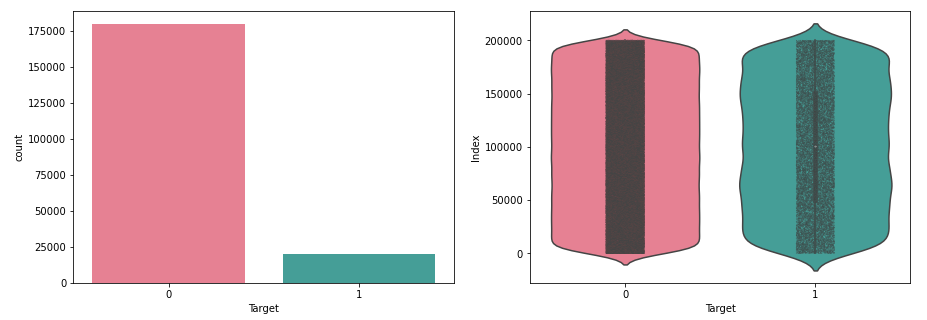


Figure 0‑1 Target Class Count

Take away:

* We have an unbalanced data, 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.
* Look at the violin plots seems that there is no relationship between the targets with the index of the train data frame. This is more dominated by the zero targets then for the ones.
* Look at the jitter plots with violin plots. We can observe that targets look uniformly distributed over the index of the data frame.

## 2.1.1 Attributes distributions and trends

**Distribution of train attributes**

Let us look distribution of train attributes from var\_0 to var\_99,



**Take away:**

* We can observe that there is a considerable number of features which are significantly have different distributions for two target variables. For example, like var\_0, var\_1, var\_9, var\_19, var\_18 etc.
* 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.

**Distribution of test attributes**

Let us look distribution of test attributes from var\_0 to var\_99,

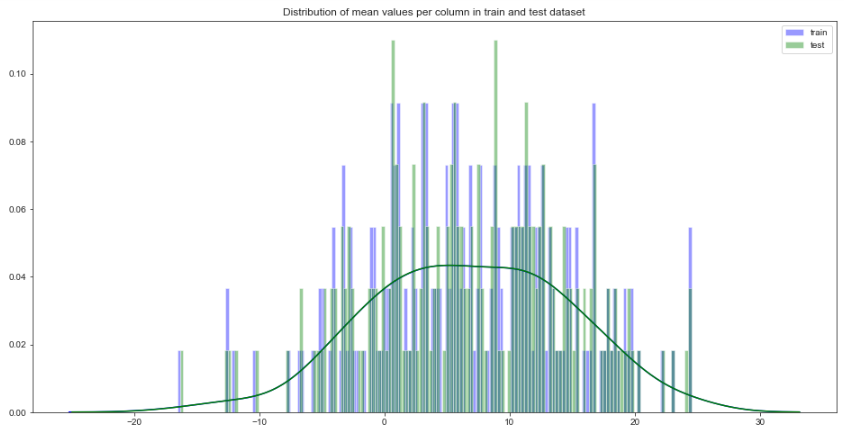


**Take away:**

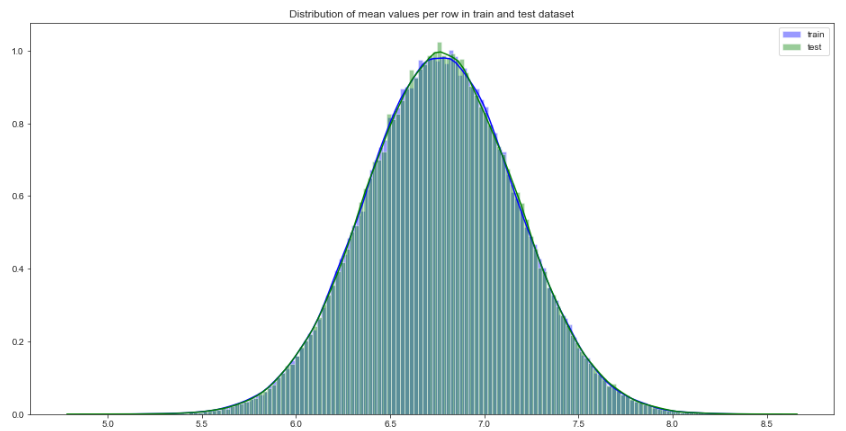
* We can observe that there is a considerable number of features which are significantly have different distributions. For example, like var\_0, var\_1, var\_9, var\_18 var\_38 etc.
* We can observe that there is a considerable number of features which are significantly have same distributions. For example, like var\_3, var\_7, var\_10, var\_17, var\_45, var\_192 etc.

**Distribution of mean values in both train and test dataset**

Let us look distribution of mean values per column in train and test dataset,

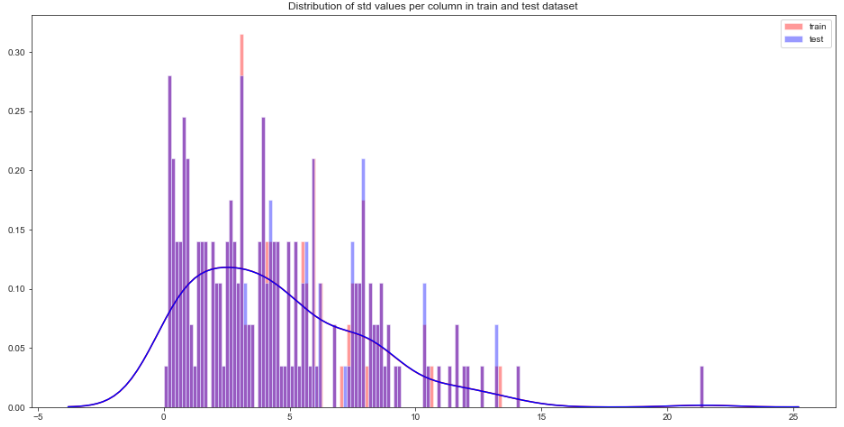


Let us look distribution of mean values per row in train and test dataset,

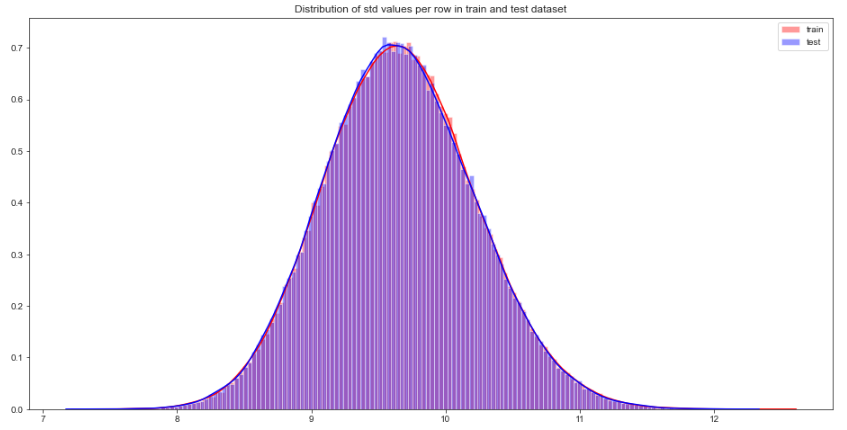


**Distribution of standard deviation (std) values in train and test dataset**

Let us look distribution of standard deviation (std) values per column in train and test dataset,



Let us look distribution of standard deviation (std) values per row in train and test dataset,



**Distribution of skewness values in train and test dataset**

Let us look distribution of skewness values per column in train and test dataset

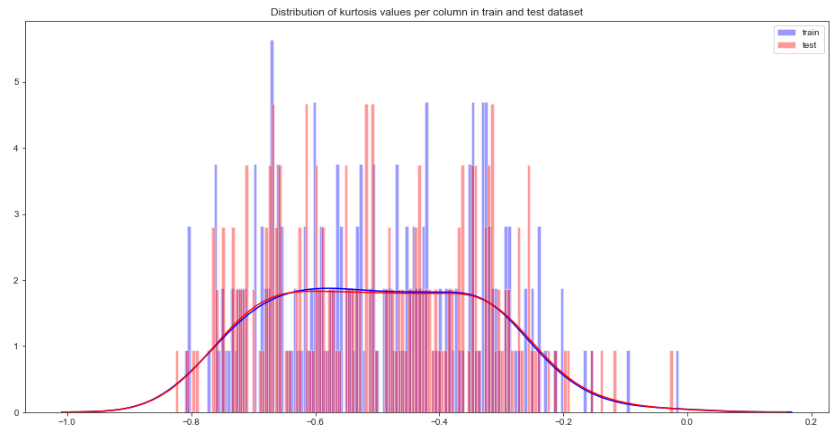


Let us look distribution of skewness per row in train and test dataset,

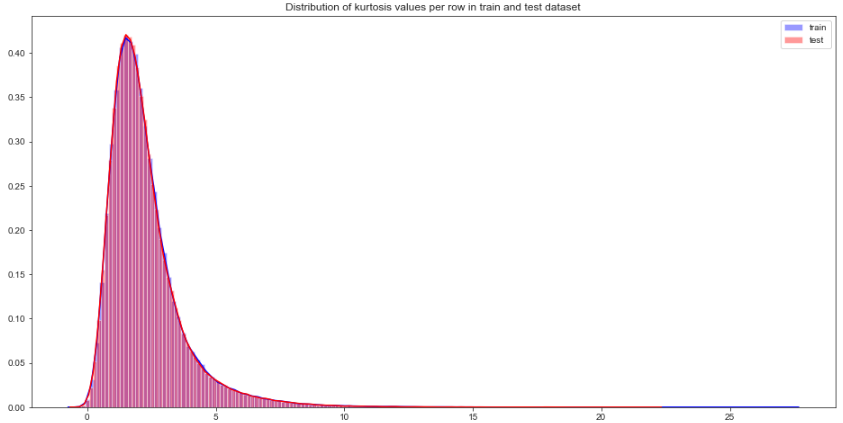


**Distribution of kurtosis values in train and test dataset**

Let us look distribution of kurtosis values per column in train and test dataset



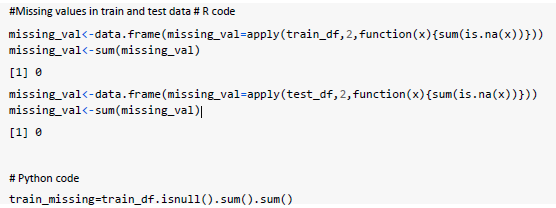
Let us look distribution of kurtosis values per row in train and test dataset



## 2.1.2 Missing Value Analysis

In this, we have to find out any missing values are present in dataset. If it’s present then either delete or impute the values using mean, median and KNN imputation method. We have not found any missing values in both train and test data.

R and Python code as follows,



## 2.1.2 Outlier Analysis

In this project, we haven’t performed outlier analysis due to the data is imbalanced and also not required for imbalanced data.

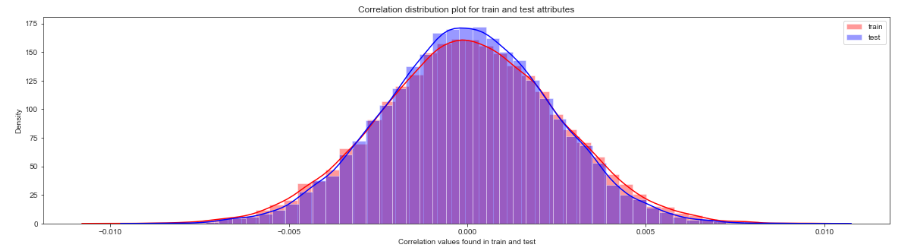
## 2.1.3 Feature Selection

Feature selection is very important for modelling the dataset. Every dataset has good and unwanted features. The unwanted features would affect on performance of model, so we have to delete those features. We have to select best features by using ANOVA, Chi-Square test and correlation matrix statistical techniques and so on. In this, we are selecting best features by using Correlation matrix.

**Correlation matrix**

Correlation matrix tells us about linear relationship between attributes and help us to build better

models. From correlation distribution plot, we can observe that correlation between both train and test attributes are very small. It means that all both train and test attributes are independent to each other.



## 2.1.4 Feature engineering

Let us do some feature engineering by using

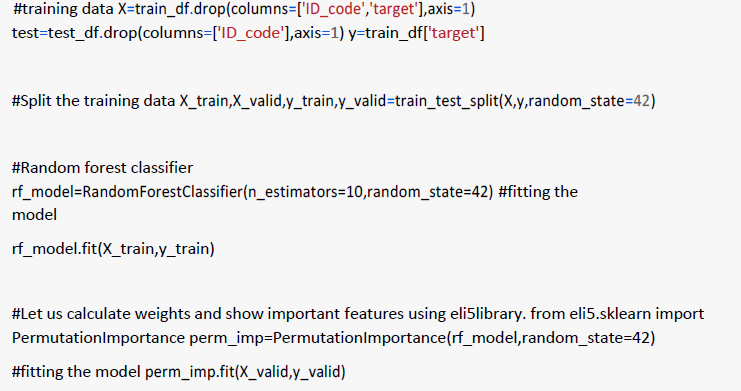
* Permutation importance
* Partial dependence plots

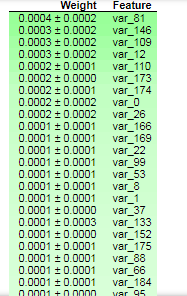
**Permutation importance**

Permutation variable importance measure in a random forest for classification and regression. The

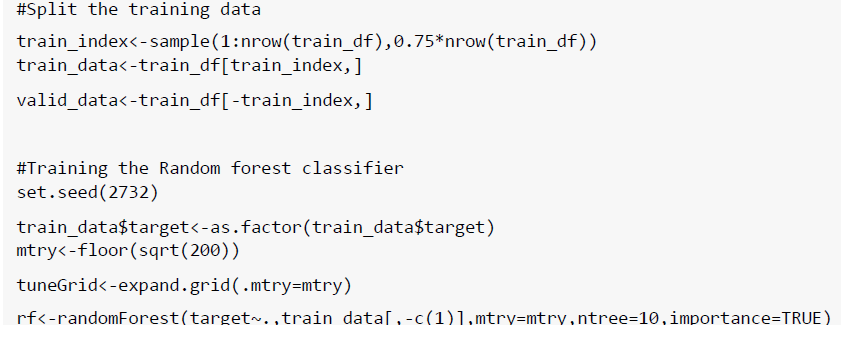
variables which are mostly contributed to predict the model.

**Python code**

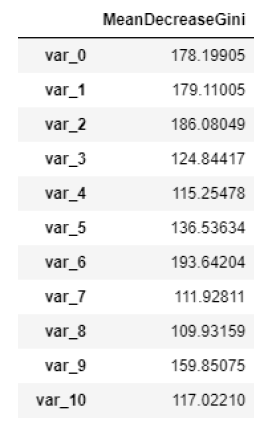




**R Code**



**Variable importance based on Mean Decrease Gini**



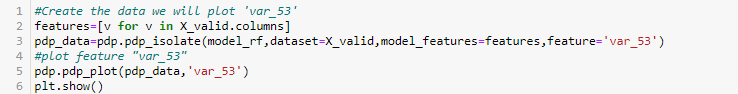
**Take away:**

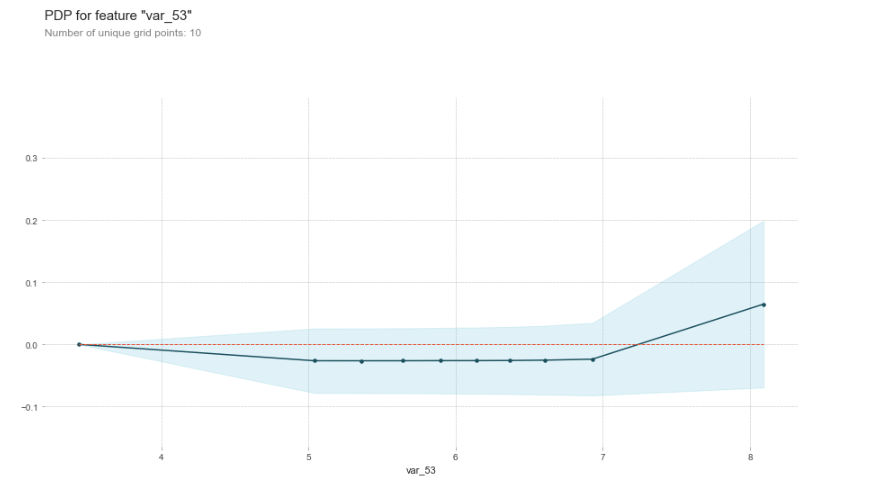
* We can observe that the top important features are var\_12, var\_26, var\_22, var\_174, var\_198 and so on based on Mean decrease Gini.

**Partial dependence plots**

Partial dependence plot gives a graphical depiction of the marginal effect of a variable on the class probability or classification. While feature importance shows what variables most affect predictions, but partial dependence plots show how a feature affects predictions.

**Python code**





**Take away:**

* The y-axis does not show the predictor value instead how the value changing with the change in given predictor variable.
* The blue shaded area indicates the level of confidence of 'var\_53'.
* On y-axis having a positive value means for that particular value of predictor variable it is

less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.

Chapter 3

# Modelling

# 3.1 Model Selection

After all early stages of pre-processing, then model the data. So, we have to select best model for this project with the help of some metrics.

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

1. Nominal

2. Ordinal

3. Interval

4. 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 like this project, the normal method is to do a **Regression** analysis, or classification after binning.

**Handling of imbalance data**

Now we are going to explore 5 different approaches for dealing with imbalanced datasets.

* Change the performance metric
* Oversample minority class
* Under sample majority class
* Synthetic Minority Oversampling Technique (SMOTE) in Python or Random
* Oversampling Examples (ROSE) in R
* Change the algorithm

We always start model building from the simplest to more complex.

## 3.1.1 Logistic Regression

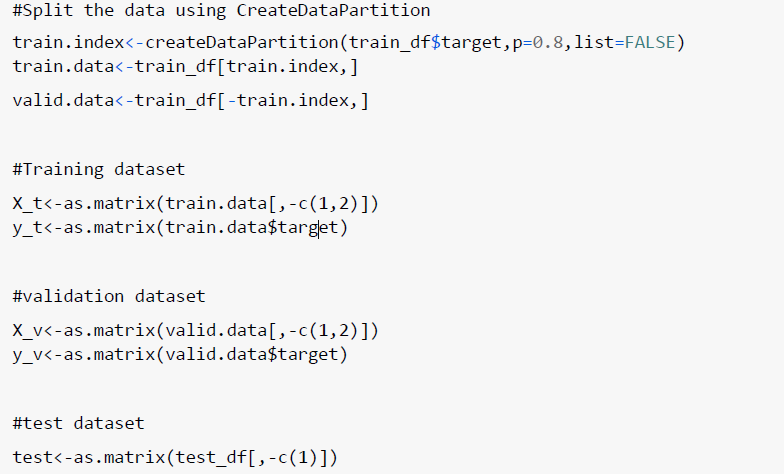
We will use a Logistic Regression to predict the values of our target variable.

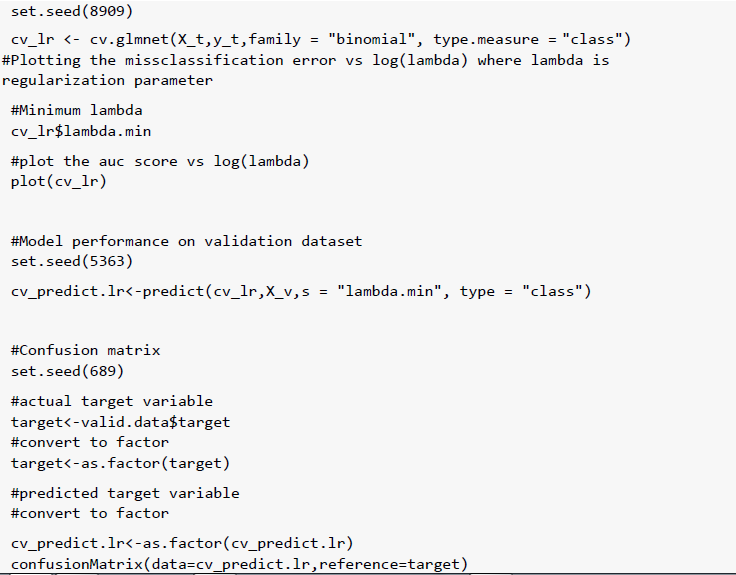
**Python code**

## 

**R code**

Glmnet is a package that fits a generalized linear model via penalized maximum likelihood.





Accuracy of the model is not the best metric to use when evaluating the imbalanced datasets as it may be misleading. So, we are going to change the performance metric.

**Oversample minority class:**

* It can be defined as adding more copies of minority class.
* It can be a good choice when we don't have a ton of data to work with.
* Drawback is that we are adding information. This may lead to overfitting and poor performance on test data.

**Under sample majority class:**

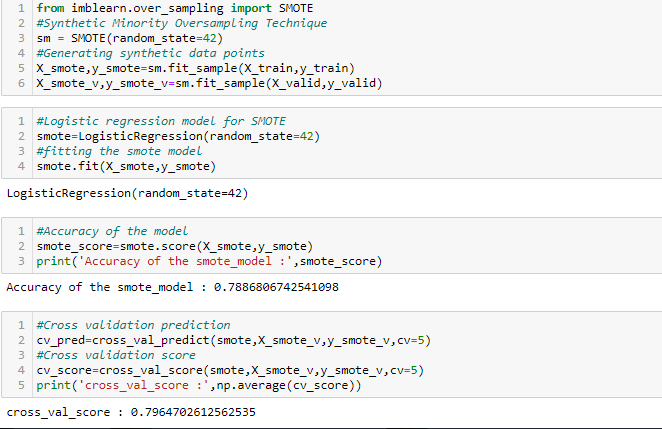
* It can be defined as removing some observations of the majority class.
* It can be a good choice when we have a ton of data -think millions of rows.
* Drawback is that we are removing information that may be valuable. This may leads
* to under fitting and poor performance on test data.

Both Oversampling and under sampling techniques have some drawbacks. So, we are not going to use these models for this problem and also we will use other best algorithms.

**Synthetic Minority Oversampling Technique (SMOTE)**

SMOTE uses a nearest neighbor’s algorithm to generate new and synthetic data to use for training the model. In order to balance imbalanced data, we are going to use SMOTE samplingmethod.

**Python code**



**R code**

**Random Oversampling Examples (ROSE)**

It creates a sample of synthetic data by enlarging the features space of minority and majority class examples. In order to balance imbalanced data we are going to use SMOTE sampling method.

#Random Oversampling Examples(ROSE)

set.seed(699)

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

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

table(train.rose$target)

valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=42)$data

#Baseline logistic regression model

set.seed(462)

lr\_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target),

family = "binomial")

summary(lr\_rose)

#Cross validation prediction

set.seed(473)

cv\_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),

family = "binomial", type.measure = "class")

#Minimum lambda

cv\_rose$lambda.min

#plot the auc score vs log(lambda)

plot(cv\_rose)

#Model performance on validation dataset

set.seed(442)

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

type = "class")

cv\_predict.rose

#Confusion matrix

set.seed(478)

#actual target variable

target<-valid.rose$target

#convert to factor

target<-as.factor(target)

#predicted target variable

#convert to factor

cv\_predict.rose<-as.factor(cv\_predict.rose)

## 3.1.2 LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. We are going to use LightGBM model.

**Python code**

Let us build LightGBM model

#Training the model

#training data

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

#validation data

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

#Selecting best hyper parameters by tuning of different parameters

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

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

'objective': 'binary',

'boost\_from\_average':False,

'nthread': 8,

'metric':'auc',

'num\_leaves': 100,

'learning\_rate': 0.03,

'max\_bin': 950, #default 255

'subsample\_for\_bin': 200,

'subsample': 1,

'subsample\_freq': 1,

'colsample\_bytree': 0.8,

'reg\_alpha': 1.2, #L1 regularization(>0)

'reg\_lambda': 1.2,#L2 regularization(>0)

'min\_split\_gain': 0.5, #>0

'min\_child\_weight': 1,

'min\_child\_samples': 5,

'is\_unbalance':True,

}

num\_rounds=3000

**R code**

#Convert data frame to matrix

X\_train<-as.matrix(train\_data[,-c(1,2)])

y\_train<-as.matrix(train\_data$target) X\_valid<-

as.matrix(valid\_data[,-c(1,2)]) y\_valid<-

as.matrix(valid\_data$target) test\_data<-

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

#training data

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

#Validation data

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

#Choosing parameters

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

metric = "auc",

boost =”gbdt”

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

feature\_fraction = 0.7,

bagging\_fraction = 0.7,

bagging\_freq = 5,

learning\_rate=0.05,

num\_leaves=80,

num\_threads=10,

min\_data = 100,

max\_bin = 200,

lambda\_l1 = 8,

#plot the important features

lgb.plot\_importance(lgbm,max\_num\_features=150,importance\_type="split",figsize=(20,

50))

lambda\_l2 = 1.3,

min\_data\_in\_bin=150,

min\_gain\_to\_split = 20,

min\_data\_in\_leaf = 40,

is\_unbalance = TRUE)

lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,

SANTANDER CUSTOMER TRANSACTION PREDICTION

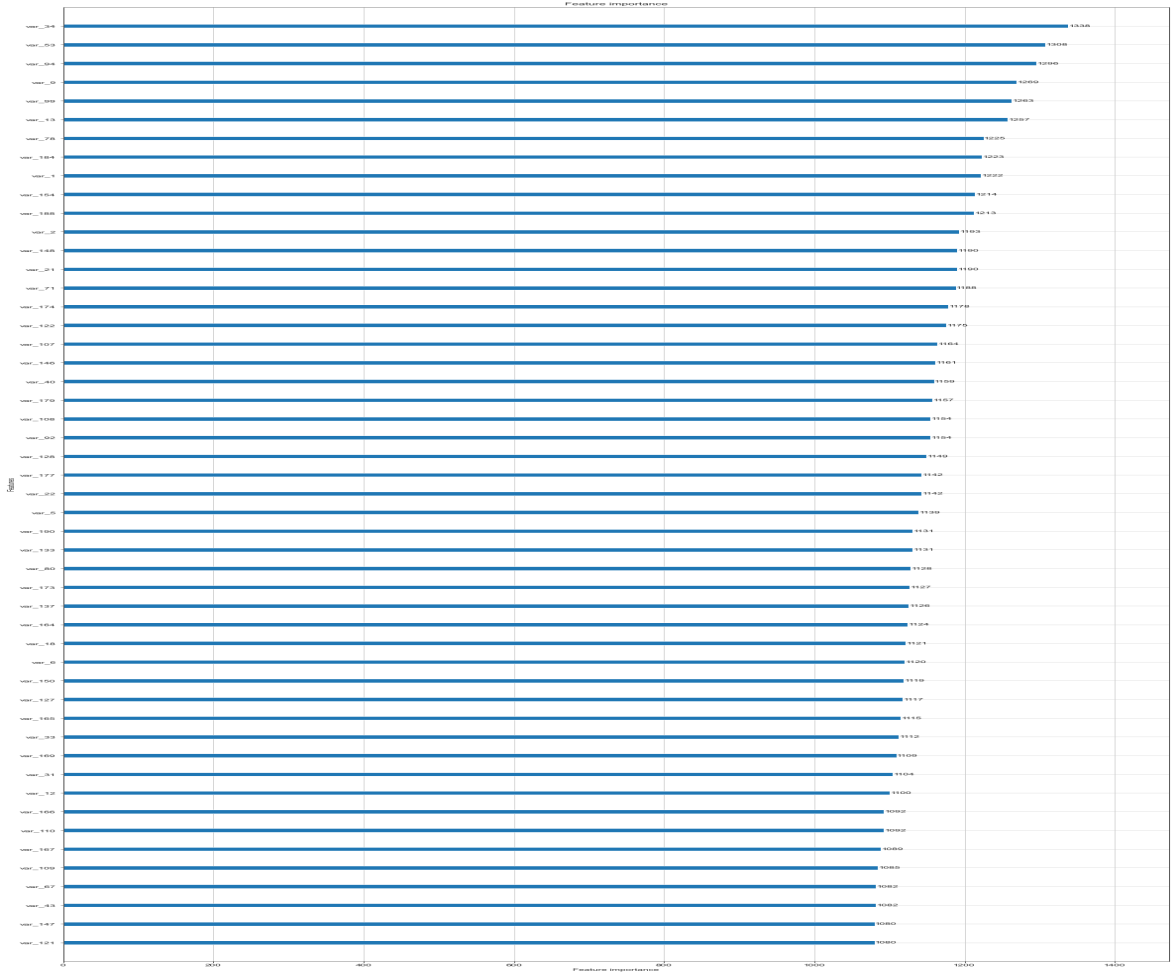
**Important features plot**

**Python code**

#plot the important features

lgb.plot\_importance(lgbm,max\_num\_features=150,importance\_type="split",figsize=(20,

50))



Chapter 4

# Model Evaluation

Now, we have three models for predicting the target variable, but we need to decide which model better for this project. There are many metrics used for model evaluation.

Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

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

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

**Confusion Matrix:** - It is a technique for summarizing the performance of a classification algorithm. The number of correct predictions and incorrect predictions are summarized with count values and broken down by each class.



**Accuracy**: - The ratio of correct predictions to total predictions

**Misclassification error:** - The ratio of incorrect predictions to total predictions

Accuracy=1-Error rate

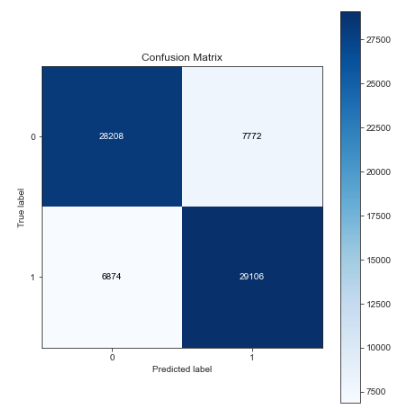
**F1 score :-** Harmonic mean of precision and recall, used to indicate balance between them.

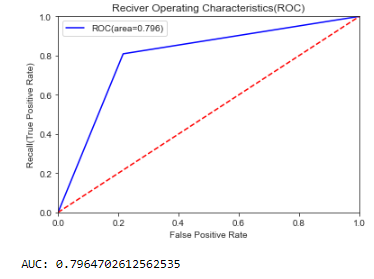
**Receiver operating characteristics (ROC)\_Area under curve(AUC) Score**

**roc\_auc\_score** :- It is a metric that computes the area under the Roc curve and also used metric for imbalanced data.

Roc curve is plotted true positive rate or Recall on y axis against false positive rate or specificity on x axis. The larger the area under the roc curve better the performance of the model.

**Logistic Regression**





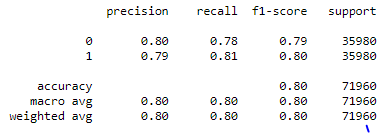
When we compare the roc\_auc\_score and cross validation score, conclude that model is not performing well on imbalanced data.

**Classification report**

#Classification report

scores=classification\_report(y\_valid,cv\_predict)

print(scores)



We can observe that f1 score is high for number of customers those who will not make a transaction then who will make a transaction. So, we are going to change the algorithm.

**R code**

**Logistic Regression**

#Cross validation prediction

set.seed(8909)

cv\_lr <- cv.glmnet(X\_t,y\_t,family = "binomial", type.measure = "class")

#Plotting misclassification error vs log(lambda)

#Minimum lambda-Regularization parameter

****

We can observed that miss classification error increases as increasing the log(Lambda).

Chapter 4

# Conclusion

When we compare scores of areas under the ROC curve of all the models for an imbalanced

data. We could conclude that below points as follow,

1. Logistic regression model is not performed well on imbalanced data.
2. We balance the imbalanced data using resampling techniques like SMOTE in python

and ROSE in R.

1. Baseline logistic regression model is performed well on balanced data.
2. LightGBM model performed well on imbalanced data.

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

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

# Appendix

# Python Code

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split,cross\_val\_predict,cross\_val\_

score

from sklearn.metrics import roc\_auc\_score,confusion\_matrix,make\_scorer,classifica

tion\_report,roc\_curve,auc

from sklearn.model\_selection import StratifiedKFold

from imblearn.over\_sampling import SMOTE, RandomOverSampler

from imblearn.under\_sampling import ClusterCentroids,NearMiss, RandomUnderSampler

import lightgbm as lgb

import eli5

from eli5.sklearn import PermutationImportance

from sklearn import tree

import graphviz

from pdpbox import pdp, get\_dataset, info\_plots

import scikitplot as skplt

from scikitplot.metrics import plot\_confusion\_matrix,plot\_precision\_recall\_curve

from scipy.stats import randint as sp\_randint

import warnings

warnings.filterwarnings('ignore')

random\_state=42

np.random.seed(random\_state)

#importing the train dataset

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

train\_df.head()

**Target classes count**

def plot\_train\_attribute\_distribution(x,y,X\_target,Y\_target,train\_attributes):

i=0

sns.set\_style('whitegrid')

fig=plt.figure()

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

for attribute in train\_attributes:

i+=1

plt.subplot(10,10,i)

sns.distplot(x[attribute],hist=False,label=X\_target)

sns.distplot(y[attribute],hist=False,label=Y\_target)

plt.legend()

plt.xlabel('Attribute',)

sns.set\_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})

plt.show()

x=df\_train[df\_train.target.values==0]

#corresponding to positive class

y=df\_train[df\_train.target.values==1]

#train attributes from 2 to 202

train\_attributes=df\_train.columns.values[2:102]

#plot distribution of train attributes

plot\_train\_attribute\_distribution(x,y,'0','1',train\_attributes)

#train attributes from 2 to 102

train\_attributes=df\_train.columns.values[102:]

#plot distribution of train attributes

plot\_train\_attribute\_distribution(x,y,'0','1',train\_attributes)

df\_test = pd.read\_csv("test.csv")

df\_test.head()

**Writing a function to plot test attributes distributions**

#Distribution of test attributes

def plot\_test\_attribute\_distribution(test\_attributes):

i=0

sns.set\_style('whitegrid')

fig=plt.figure()

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

for attribute in test\_attributes:

i+=1

plt.subplot(10,10,i)

sns.distplot(df\_test[attribute],hist=False)

plt.xlabel('Attribute',)

sns.set\_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})

plt.show()

#test attribiutes from 1 to 101

test\_attributes=df\_test.columns.values[1:101]

#plot distribution of test attributes

plot\_test\_attribute\_distribution(test\_attributes)

# Visualizations of different distributions

#Distribution of mean values per column in train and test dataset

plt.figure(figsize=(16,8))

#train attributes

train\_attributes=df\_train.columns.values[2:202]

#test attributes

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for mean values per column in train attributes

sns.distplot(df\_train[train\_attributes].mean(axis=0),color='blue',kde=True,bins=150,label='train')

#Distribution plot for mean values per column in test attributes

sns.distplot(df\_test[test\_attributes].mean(axis=0),color='green',kde=True,bins=150,label='test')

plt.title('Distribution of mean values per column in train and test dataset')

plt.legend()

plt.show()

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

plt.figure(figsize=(16,8))

#Distribution plot for mean values per row in train attributes

sns.distplot(df\_train[train\_attributes].mean(axis=1),color='blue',kde=True,bins=150,label='train')

#Distribution plot for mean values per row in test attributes

sns.distplot(df\_test[test\_attributes].mean(axis=1),color='green',kde=True, bins=150, label='test')

plt.title('Distribution of mean values per row in train and test dataset')

plt.legend()

plt.show()

#Distribution of std values per column in train and test dataset

plt.figure(figsize=(16,8))

#train attributes

train\_attributes=df\_train.columns.values[2:202]

#test attributes

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for std values per column in train attributes

sns.distplot(df\_train[train\_attributes].std(axis=0),color='red',kde=True,bins=150,label='train')

#Distribution plot for std values per column in test attributes

sns.distplot(df\_test[test\_attributes].std(axis=0),color='blue',kde=True,bins=150,label='test')

plt.title('Distribution of std values per column in train and test dataset')

plt.legend()

plt.show()

#Distribution of std values per row in train and test dataset

plt.figure(figsize=(16,8))

#Distribution plot for std values per row in train attributes

sns.distplot(df\_train[train\_attributes].std(axis=1),color='red',kde=True,bins=150,label='train')

#Distribution plot for std values per row in test attributes

sns.distplot(df\_test[test\_attributes].std(axis=1),color='blue',kde=True, bins=150, label='test')

plt.title('Distribution of std values per row in train and test dataset')

plt.legend()

plt.show()

#Distribution of kurtosis values per column in train and test dataset

plt.figure(figsize=(16,8))

#train attributes

train\_attributes=df\_train.columns.values[2:202]

#test attributes

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for kurtosis values per column in train attributes

sns.distplot(df\_train[train\_attributes].kurtosis(axis=0),color='blue',kde=True,bins=150,label='train')

#Distribution plot for kurtosis values per column in test attributes

sns.distplot(df\_test[test\_attributes].kurtosis(axis=0),color='red',kde=True,bins=150,label='test')

plt.title('Distribution of kurtosis values per column in train and test dataset')

plt.legend()

plt.show()

#Distribution of kutosis values per row in train and test dataset

plt.figure(figsize=(16,8))

#Distribution plot for kurtosis values per row in train attributes

sns.distplot(df\_train[train\_attributes].kurtosis(axis=1),color='blue',kde=True,bins=150,label='train')

#Distribution plot for kurtosis values per row in test attributes

sns.distplot(df\_test[test\_attributes].kurtosis(axis=1),color='red',kde=True, bins=150, label='test')

plt.title('Distribution of kurtosis values per row in train and test dataset')

plt.legend()

plt.show()

#Distribution of kurtosis values per column in train and test dataset

plt.figure(figsize=(16,8))

#train attributes

train\_attributes=df\_train.columns.values[2:202]

#test attributes

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for kurtosis values per column in train attributes

sns.distplot(df\_train[train\_attributes].kurtosis(axis=0),color='blue',kde=True,bins=150,label='train')

#Distribution plot for kurtosis values per column in test attributes

sns.distplot(df\_test[test\_attributes].kurtosis(axis=0),color='red',kde=True,bins=150,label='test')

plt.title('Distribution of kurtosis values per column in train and test dataset')

plt.legend()

plt.show()

#Distribution of kutosis values per row in train and test dataset

plt.figure(figsize=(16,8))

#Distribution plot for kurtosis values per row in train attributes

sns.distplot(df\_train[train\_attributes].kurtosis(axis=1),color='blue',kde=True,bins=150,label='train')

#Distribution plot for kurtosis values per row in test attributes

sns.distplot(df\_test[test\_attributes].kurtosis(axis=1),color='red',kde=True, bins=150, label='test')

plt.title('Distribution of kurtosis values per row in train and test dataset')

plt.legend()

plt.show()

# Missing Value Analaysis

#Finding the missing values in train and test data

train\_missing=df\_train.isnull().sum().sum()

test\_missing=df\_test.isnull().sum().sum()

print('Missing values in train data :',train\_missing)

print('Missing values in test data :',test\_missing)

# Correlation in train and test datasets

#Correlations in train attributes

train\_attributes=df\_train.columns.values[2:202]

correlations\_train=df\_train[train\_attributes].corr().abs().unstack().sort\_values(kind='quicksort').reset\_index()

correlations\_train=correlations\_train[correlations\_train['level\_0']!=correlations\_train['level\_1']]

print(correlations\_train.head(10))

print(correlations\_train.tail(10))

#Correlations in test attributes

test\_attributes=df\_test.columns.values[1:201]

correlations\_test=df\_test[test\_attributes].corr().abs().unstack().sort\_values(kind='quicksort').reset\_index()

correlations\_test=correlations\_test[correlations\_test['level\_0']!=correlations\_test['level\_1']]

print(correlations\_test.head(10))

print(correlations\_test.tail(10))

#Correlations in train data

correlations\_train=df\_train[train\_attributes].corr()

correlations\_train=correlations\_train.values.flatten()

correlations\_train=correlations\_train[correlations\_train!=1]

#Correlations in test data

correlations\_test=df\_test[test\_attributes].corr()

correlations\_test=correlations\_test.values.flatten()

correlations\_test=correlations\_test[correlations\_test!=1]

plt.figure(figsize=(20,5))

#Distribution plot for correlations in train data

sns.distplot(correlations\_train, color="Red", label="train")

#Distribution plot for correlations in test data

sns.distplot(correlations\_test, color="Blue", label="test")

plt.xlabel("Correlation values found in train and test")

plt.ylabel("Density")

plt.title("Correlation distribution plot for train and test attributes")

plt.legend()

# Split The dataset

from sklearn.model\_selection import train\_test\_split,cross\_val\_predict,cross\_val\_score

#Split the training data

X\_train,X\_valid,y\_train,y\_valid=train\_test\_split(X,y,random\_state=42)

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

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

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

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

**Permutation importance**

#Permutation importance

from eli5.sklearn import PermutationImportance

perm\_imp=PermutationImportance(model\_rf,random\_state=42)

#fitting the model

perm\_imp.fit(X\_valid,y\_valid)

#Important features

eli5.show\_weights(perm\_imp,feature\_names=X\_valid.columns.tolist(),top=200)

#Create the data we will plot 'var\_53'

features=[v for v in X\_valid.columns]

pdp\_data=pdp.pdp\_isolate(model\_rf,dataset=X\_valid,model\_features=features,feature='var\_53')

#plot feature "var\_53"

pdp.pdp\_plot(pdp\_data,'var\_53')

plt.show()

Cross Validation

#StratifiedKFold cross validator

cv=StratifiedKFold(n\_splits=5,random\_state=42,shuffle=True)

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

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

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

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

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

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

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

# Logistic Regression

#Logistic regression model

model\_lr=LogisticRegression(random\_state=42)

#fitting the lr model

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

#Accuracy of the model

score\_lr=model\_lr.score(X\_train,y\_train)

print('Accuracy of the lr\_model :',score\_lr)

#Cross validation prediction

cv\_predict=cross\_val\_predict(model\_lr,X\_valid,y\_valid,cv=5)

#Cross validation score

cv\_score=cross\_val\_score(model\_lr,X\_valid,y\_valid,cv=5)

print('cross\_val\_score :',np.average(cv\_score))

#Confusion matrix

cm=confusion\_matrix(y\_valid,cv\_predict)

#Plot the confusion matrix

plot\_confusion\_matrix(y\_valid,cv\_predict,normalize=False,figsize=(7,8))

#ROC\_AUC score

roc\_score=roc\_auc\_score(y\_valid,cv\_predict)

print('ROC score :',roc\_score)

#ROC\_AUC curve

plt.figure()

false\_positive\_rate,recall,thresholds=roc\_curve(y\_valid,cv\_predict)

roc\_auc=auc(false\_positive\_rate,recall)

plt.title('Reciver Operating Characteristics(ROC)')

plt.plot(false\_positive\_rate,recall,'b',label='ROC(area=%0.3f)' %roc\_auc)

plt.legend()

plt.plot([0,1],[0,1],'r--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.0])

plt.ylabel('Recall(True Positive Rate)')

plt.xlabel('False Positive Rate')

plt.show()

print('AUC:',roc\_auc)

#Classification report

scores=classification\_report(y\_valid,cv\_predict)

print(scores)

#Predicting the model

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

pred\_lr=model\_lr.predict(X\_test)

print(pred\_lr)

**SMOTE**

from imblearn.over\_sampling import SMOTE

#Synthetic Minority Oversampling Technique

sm = SMOTE(random\_state=42)

#Generating synthetic data points

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

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

#Logistic regression model for SMOTE

smote=LogisticRegression(random\_state=42)

#fitting the smote model

smote.fit(X\_smote,y\_smote)

#Accuracy of the model

smote\_score=smote.score(X\_smote,y\_smote)

print('Accuracy of the smote\_model :',smote\_score)

#Cross validation prediction

cv\_pred=cross\_val\_predict(smote,X\_smote\_v,y\_smote\_v,cv=5)

#Cross validation score

cv\_score=cross\_val\_score(smote,X\_smote\_v,y\_smote\_v,cv=5)

print('cross\_val\_score :',np.average(cv\_score))

# Light GBM Model

#Training the model

#training data

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

#validation data

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

#Selecting best hyperparameters by tuning of different parameters

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

lgbm= lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid],verbose\_eval=1000,early\_stopping\_rounds = 5000)

lgbm

#predict the model

#probability predictions

lgbm\_predict\_prob=lgbm.predict(X\_test,random\_state=42,num\_iteration=lgbm.best\_iteration)

#Convert to binary output 1 or 0

lgbm\_predict=np.where(lgbm\_predict\_prob>=0.5,1,0)

print(lgbm\_predict\_prob)

print(lgbm\_predict)

#plot the important features

lgb.plot\_importance(lgbm,max\_num\_features=50,importance\_type="split",figsize=(20,50))

#final prediction

predict\_df=pd.DataFrame({'ID\_code':df\_test['ID\_code'].values})

predict\_df['lgbm\_predict\_prob']=lgbm\_predict\_prob

predict\_df['lgbm\_predict']=lgbm\_predict

predict\_df.to\_csv('test\_predict.csv',index=False)

predict\_df.head()

# R CODE

library(tidyverse)

library(moments)

library(DataExplorer)

library(caret)

library(Matrix)

library(pdp)

library(mlbench)

library(caTools)

library(randomForest)

library(glmnet)

library(mlr)

library(vita)

library(rBayesianOptimization)

library(lightgbm)

library(pROC)

library(DMwR)

library(ROSE)

library(yardstick)

list.files(path = "C:/Users/rkocherlakota/Desktop/ds\_projects/Santander Customer Transaction")

#loading the train data

train\_df<-read.csv('C:/Users/rkocherlakota/Desktop/ds\_projects/Santander Customer Transaction/train.csv')

head(train\_df)

#Dimension of train data

dim(train\_df)

#Summary of the dataset

str(train\_df)

#convert to factor

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

require(gridExtra)

#Count of target classes

table(train\_df$target)

#Percenatge counts of target classes

table(train\_df$target)/length(train\_df$target)\*100

#Bar plot for count of target classes

plot1<-ggplot(train\_df,aes(target))+theme\_bw()+geom\_bar(stat='count',fill='lightgreen')

#Violin with jitter plots for target classes

plot2<-ggplot(train\_df,aes(x=target,y=1:nrow(train\_df)))+theme\_bw()+geom\_violin(fill='lightblue')+

facet\_grid(train\_df$target)+geom\_jitter(width=0.02)+labs(y='Index')

grid.arrange(plot1,plot2, ncol=2)

#Distribution of train attributes from 3 to 102

for (var in names(train\_df)[c(3:102)]){

target<-train\_df$target

plot<-ggplot(train\_df, aes(x=train\_df[[var]],fill=target)) +

geom\_density(kernel='gaussian') + ggtitle(var)+theme\_classic()

print(plot)

}

#Distribution of train attributes from 103 to 202

for (var in names(train\_df)[c(103:202)]){

target<-train\_df$target

plot<-ggplot(train\_df, aes(x=train\_df[[var]], fill=target)) +

geom\_density(kernel='gaussian') + ggtitle(var)+theme\_classic()

print(plot)

}

#loading test data

test\_df<-read.csv('C:/Users/rkocherlakota/Desktop/ds\_projects/Santander Customer Transaction/test.csv')

head(test\_df)

#Dimension of test dataset

dim(test\_df)

#Distribution of test attributes from 2 to 101

plot\_density(test\_df[,c(2:101)], ggtheme = theme\_classic(),geom\_density\_args = list(color='blue'))

#Distribution of test attributes from 102 to 201

plot\_density(test\_df[,c(102:201)], ggtheme = theme\_classic(),geom\_density\_args = list(color='blue'))

#Applying the function to find mean values per row in train and test data.

train\_mean<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=mean)

test\_mean<-apply(test\_df[,-c(1)],MARGIN=1,FUN=mean)

ggplot()+

#Distribution of mean values per row in train data

geom\_density(data=train\_df[,-c(1,2)],aes(x=train\_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of mean values per row in test data

geom\_density(data=test\_df[,-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")

#Applying the function to find mean values per column in train and test data.

train\_mean<-apply(train\_df[,-c(1,2)],MARGIN=2,FUN=mean)

test\_mean<-apply(test\_df[,-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")

#Applying the function to find standard deviation values per row in train and test data.

train\_sd<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=sd)

test\_sd<-apply(test\_df[,-c(1)],MARGIN=1,FUN=sd)

ggplot()+

#Distribution of sd values per row in train data

geom\_density(data=train\_df[,-c(1,2)],aes(x=train\_sd),kernel='gaussian',show.legend=TRUE,color='red')+theme\_classic()+

#Distribution of mean values per row in test data

geom\_density(data=test\_df[,-c(1)],aes(x=test\_sd),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='sd values per row',title="Distribution of sd values per row in train and test dataset")

#Applying the function to find sd values per column in train and test data.

train\_sd<-apply(train\_df[,-c(1,2)],MARGIN=2,FUN=sd)

test\_sd<-apply(test\_df[,-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\_classic()+

#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")

#Applying the function to find skewness values per row in train and test data.

train\_skew<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=skewness)

test\_skew<-apply(test\_df[,-c(1)],MARGIN=1,FUN=skewness)

ggplot()+

#Distribution of skewness values per row in train data

geom\_density(aes(x=train\_skew),kernel='gaussian',show.legend=TRUE,color='green')+theme\_classic()+

#Distribution of skewness values per column in test data

geom\_density(aes(x=test\_skew),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='skewness values per row',title="Distribution of skewness values per row in train and test dataset")

#Applying the function to find skewness values per column in train and test data.

train\_skew<-apply(train\_df[,-c(1,2)],MARGIN=2,FUN=skewness)

test\_skew<-apply(test\_df[,-c(1)],MARGIN=2,FUN=skewness)

ggplot()+

#Distribution of skewness values per column in train data

geom\_density(aes(x=train\_skew),kernel='gaussian',show.legend=TRUE,color='green')+theme\_classic()+

#Distribution of skewness values per column in test data

geom\_density(aes(x=test\_skew),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='skewness values per column',title="Distribution of skewness values per column in train and test dataset")

#Applying the function to find kurtosis values per row in train and test data.

train\_kurtosis<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=kurtosis)

test\_kurtosis<-apply(test\_df[,-c(1)],MARGIN=1,FUN=kurtosis)

ggplot()+

#Distribution of sd values per column in train data

geom\_density(aes(x=train\_kurtosis),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of sd values per column in test data

geom\_density(aes(x=test\_kurtosis),kernel='gaussian',show.legend=TRUE,color='red')+

labs(x='kurtosis values per row',title="Distribution of kurtosis values per row in train and test dataset")

#Applying the function to find kurtosis values per column in train and test data.

train\_kurtosis<-apply(train\_df[,-c(1,2)],MARGIN=2,FUN=kurtosis)

test\_kurtosis<-apply(test\_df[,-c(1)],MARGIN=2,FUN=kurtosis)

ggplot()+

#Distribution of sd values per column in train data

geom\_density(aes(x=train\_kurtosis),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of sd values per column in test data

geom\_density(aes(x=test\_kurtosis),kernel='gaussian',show.legend=TRUE,color='red')+

labs(x='kurtosis values per column',title="Distribution of kurtosis values per column in train and test dataset")

#Finding the missing values in train data

missing\_val<-data.frame(missing\_val=apply(train\_df,2,function(x){sum(is.na(x))}))

missing\_val<-sum(missing\_val)

missing\_val

#Finding the missing values in test data

missing\_val<-data.frame(missing\_val=apply(test\_df,2,function(x){sum(is.na(x))}))

missing\_val<-sum(missing\_val)

missing\_val

#Correlations in train data

#convert factor to int

train\_df$target<-as.numeric(train\_df$target)

train\_correlations<-cor(train\_df[,c(2:202)])

train\_correlations

#Correlations in test data

test\_correlations<-cor(test\_df[,c(2:201)])

test\_correlations

#Split the training data using simple random sampling

train\_index<-sample(1:nrow(train\_df),0.75\*nrow(train\_df))

#train data

train\_data<-train\_df[train\_index,]

#validation data

valid\_data<-train\_df[-train\_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 ranndom forest

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

#Variable importance

VarImp<-importance(rf,type=2)

VarImp

#We will plot "var\_13"

par.var\_13 <- partial(rf, pred.var = c("var\_13"), chull = TRUE)

plot.var\_13 <- autoplot(par.var\_13, contour = TRUE)

plot.var\_13

#We will plot "var\_34"

par.var\_34 <- partial(rf, pred.var = c("var\_34"), chull = TRUE)

plot.var\_34 <- autoplot(par.var\_34, contour = TRUE)

plot.var\_34

#Split the data using CreateDataPartition

set.seed(689)

#train.index<-createDataPartition(train\_df$target,p=0.8,list=FALSE)

train.index<-sample(1:nrow(train\_df),0.8\*nrow(train\_df))

#train data

train.data<-train\_df[train.index,]

#validation data

valid.data<-train\_df[-train.index,]

#dimension of train data

dim(train.data)

#dimension of validation data

dim(valid.data)

#target classes in train data

table(train.data$target)

#target classes in validation data

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\_df[,-c(1)])

#Logistic regression model

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)

#predict the model

#set.seed(763)

#lr\_pred<-predict(lr\_model,test,type='class')

#Random Oversampling Examples(ROSE)

set.seed(699)

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

#target classes in balanced train data

table(train.rose$target)

valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=42)$data

#target classes in balanced valid data

table(valid.rose$target)

#Logistic regression model

set.seed(462)

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

summary(lr\_rose)

#Cross validation prediction

set.seed(473)

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

cv\_rose

#Minimum lambda

cv\_rose$lambda.min

#plot the auc score vs log(lambda)

plot(cv\_rose)

#Model performance on validation dataset

set.seed(442)

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

cv\_predict.rose

#Confusion matrix

set.seed(478)

#actual target variable

target<-valid.rose$target

#convert to factor

target<-as.factor(target)

#predicted target variable

#convert to factor

cv\_predict.rose<-as.factor(cv\_predict.rose)

#Confusion matrix

confusionMatrix(data=cv\_predict.rose,reference=target)

#ROC\_AUC score and curve

set.seed(843)

#convert to numeric

cv\_predict.rose<-as.numeric(cv\_predict.rose)

roc(data=valid.rose[,-c(1,2)],response=target,predictor=cv\_predict.rose,auc=TRUE,plot=TRUE)

#predict the model

#set.seed(6543)

#rose\_pred<-predict(lr\_rose,test,type='class')

#Convert data frame to matrix

set.seed(5432)

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

y\_train<-as.matrix(train.data$target)

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

y\_valid<-as.matrix(valid.data$target)

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

#training data

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

#Validation data

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

#Selecting best hyperparameters

set.seed(653)

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

metric = "auc",

boost='gbdt',

max\_depth=-1,

boost\_from\_average='false',

min\_sum\_hessian\_in\_leaf = 12,

feature\_fraction = 0.05,

bagging\_fraction = 0.45,

bagging\_freq = 5,

learning\_rate=0.02,

tree\_learner='serial',

num\_leaves=20,

num\_threads=5,

min\_data\_in\_bin=150,

min\_gain\_to\_split = 30,

min\_data\_in\_leaf = 90,

verbosity=-1,

is\_unbalance = TRUE)

set.seed(7663)

lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,eval\_freq =1000,

valids=list(val1=lgb.train,val2=lgb.valid),early\_stopping\_rounds = 5000)

#lgbm model performance on test data

set.seed(6532)

lgbm\_pred\_prob <- predict(lgbm.model,test\_data)

print(lgbm\_pred\_prob)

#Convert to binary output (1 and 0) with threshold 0.5

lgbm\_pred<-ifelse(lgbm\_pred\_prob>0.5,1,0)

print(lgbm\_pred)

set.seed(6521)

#feature importance plot

tree\_imp <- lgb.importance(lgbm.model, percentage = TRUE)

lgb.plot.importance(tree\_imp, top\_n = 50, measure = "Frequency", left\_margin = 10)

sub\_df<-data.frame(ID\_code=test\_df$ID\_code,lgb\_predict\_prob=lgbm\_pred\_prob,lgb\_predict=lgbm\_pred)

write.csv(sub\_df,'Submission.CSV',row.names=F)

head(sub\_df)