Santander Customer Transaction Prediction Submitted by Devesh Srivastava

### Introduction 3

* 1. [Problem Statement 3](#_TOC_250005)
  2. Dataset 3

1. Methodology 4
   1. Exploratory Data Analysis 5
      1. Missing value analysis 6
      2. Attributes Distributions and trends 7
      3. Outlier Analysis 13
      4. Feature Selection 13
      5. Feature Engineering 14
   2. Modeling 18
      1. Model Selection 19
      2. Logistic Regression 20
      3. SMOTE or ROSE 23
      4. LightGBM. 27
2. Conclusion 28
   1. [Model Evaluation 32](#_TOC_250004)
      1. Confusion Matrix 32
      2. ROC\_AUC\_score 33
   2. [Model Selection 43](#_TOC_250003)

Appendix A - Extra Figures 39

Appendix B – Complete Python and R Code 51

[Python Code 49](#_TOC_250002)

[R code 70](#_TOC_250001)

[References 89](#_TOC_250000)

**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

Our goal in this project is to create classification models that can be used to predict which customers will make a particular purchase in the future. A sample of the Santander customer transaction dataset is given below:

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID\_code** | **target** | var\_0 | var\_1 | var\_2 | ……… | **……….** | **var\_199** |
| **train\_01** | 0 | 8.92 | -6.78 | 11.90 | …..... | …….. | -1.09 |
| **train\_02** | 0 | 11.5 | -4.14 | 13.85 | ……... | ...…… | 1.95 |
| **train\_03** | 0 | 8.60 | -2.74 | 12.08 | ...…… | ...…… | 0.39 |
| **train\_04** | 0 | 11.06 | -2.15 | 8.95 | ...…… | ...…… | -8.99 |
| **train\_05** | 0 | 9.83 | -1.48 | 12.87 | ...…… | ...…… | -8.81 |

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID\_code** | **var\_0** | var\_1 | var\_2 | var\_3 | ……… | **……….** | **var\_199** |
| **test\_01** | 11.06 | 8.92 | -6.78 | 11.90 | …..... | …….. | -1.09 |
| test\_02 | 8.53 | 11.5 | -4.14 | 13.85 | ……... | ...…… | 1.95 |
| **test\_03** | 5.48 | 8.60 | -2.74 | 12.08 | ...…… | ...…… | 0.39 |
| test\_04 | 8.53 | 11.06 | -2.15 | 8.95 | ...…… | ...…… | -8.99 |
| **test\_05** | 11.7 | 9.83 | -1.48 | 12.87 | ...…… | ...…… | -8.81 |

We have 201 variables in the table below that we must use to predict the customer transaction:

Table 1.3: Predictor Variables

|  |  |
| --- | --- |
| SL.No. | Predictor |
| **1** | ID-code |
| 2 | var0 |
| **3** | var1 |
| 4 | var2 |
| **5** | var3 |
| 6 | var4 |
| **7** | var5 |
| .… | .. … |
| ….. | |
| …. | …… |
| **….** | …… |
| …. | ……. |
| **….** | ……. |
| …. | ……. |
| **….** | …….. |
| 202 | var199 |

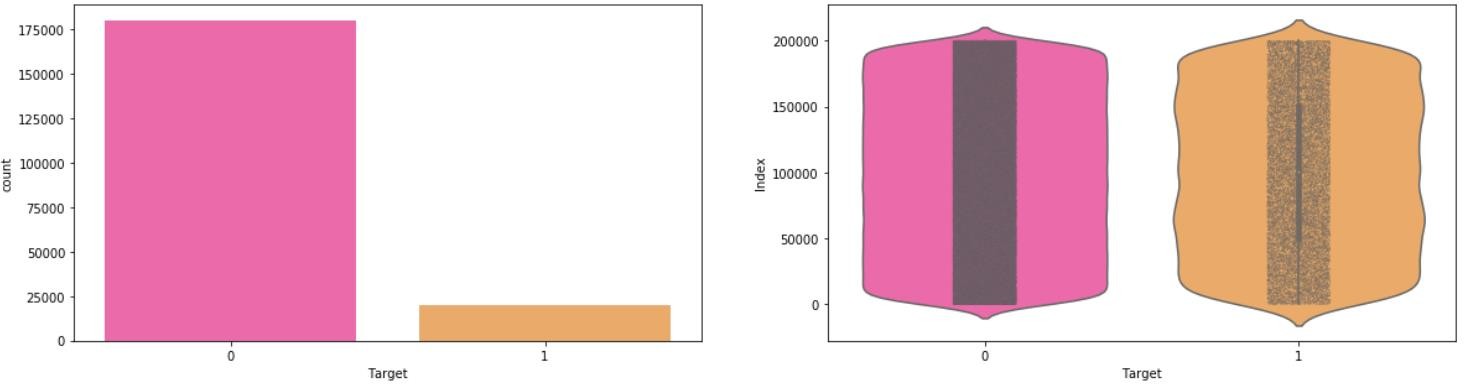
## Chapter 2

### Methodology

**2.1 Exploratory Data Analysis (EDA)**

Exploratory data analysis is one of the most important steps in data mining because it allows you to learn about the characteristics of the data. It entails loading the dataset, counting the target classes, cleaning the data, typecasting the attributes, missing value analysis, and analysing the distributions and trends of the attributes. As a result, we must clean the data, or the model's output will suffer. Now, we'll go over each of them one by one. I clarified with seaborn visualisations in this EDA.

### Target classes count



**Observation**:

We have an unbalanced data set, with 90% of the data representing the number of consumers who will not make a purchase and 10% representing those who will.

The violin plots show that there is no relationship between the data frame's objective and index, and that zero is more dominant than one.

We can see from the jitter plots with violin plots that the target appears to be evenly distributed across the data frame's indexes..

### 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: -

**#R Code:-**

#Finding the missing values in train data

missing\_val<-data.frame(missing\_val=apply(df\_train,2,function(x){s um(is.na(x))}))

missing\_val<-sum(missing\_val) missing\_val

#Finding the missing values in test data

missing\_val<-data.frame(missing\_val=apply(df\_test,2,function(x){su m(is.na(x))}))

missing\_val<-sum(missing\_val) missing\_val

**#Python Code: -**

#Finding the missing values in train & test dataset:- 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)

### Attributes distributions and trends Distribution of train attributes

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



Observation:

- We can observe that there is a considerable number of features which have significantly different distribution. For eg: var\_0,var\_1, var\_6, var\_109, var\_199 etc.

- There are also considerable number of features which have same distribution. For eg: var\_101, var\_4, var\_5, var\_193 etc.

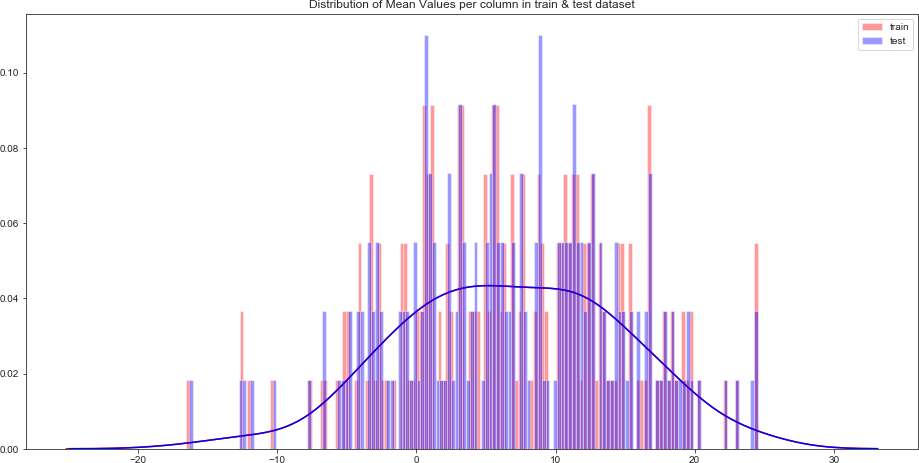
### Distribution of test attributes

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

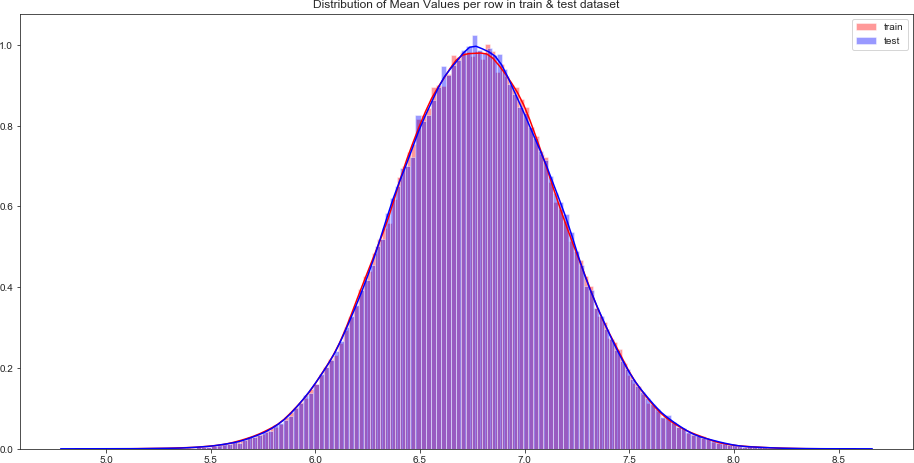


### Distribution of mean values in both train and test dataset:-

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

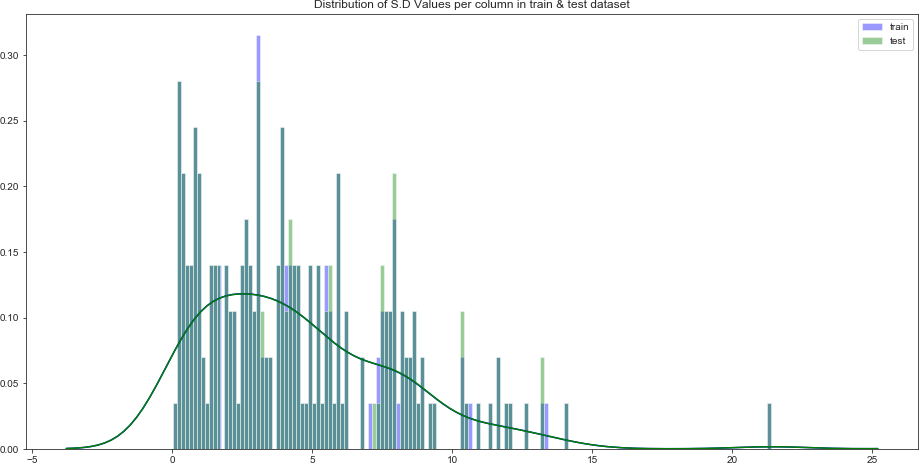


Let us look distribution of mean values per row in train and test dataset:-

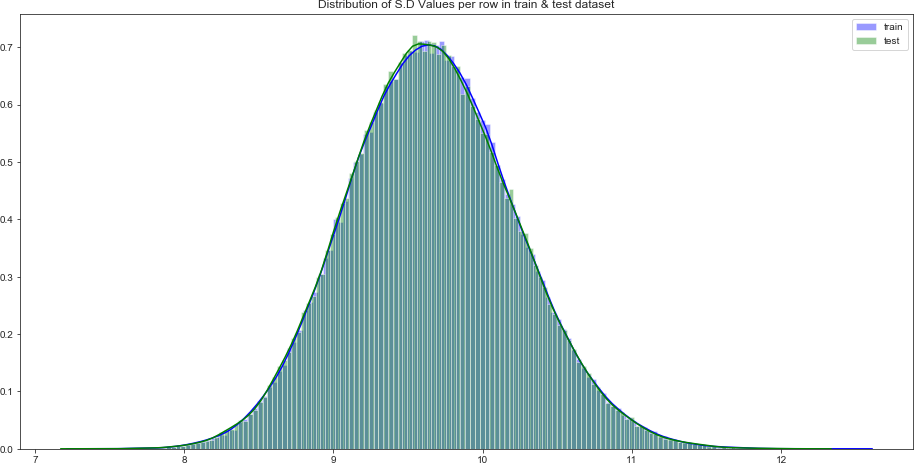


### Distribution of standard deviation (std) values in train and test dataset

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

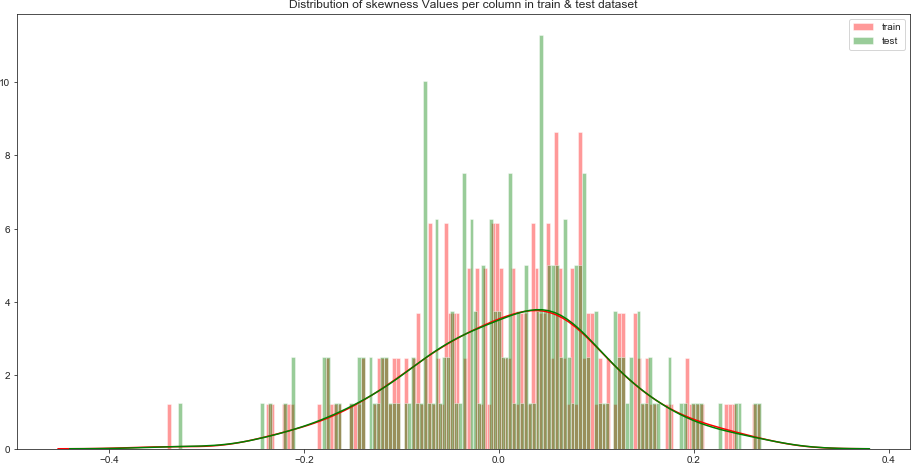


Let us look distribution of standard deviation (std) values per column in train and test dataset

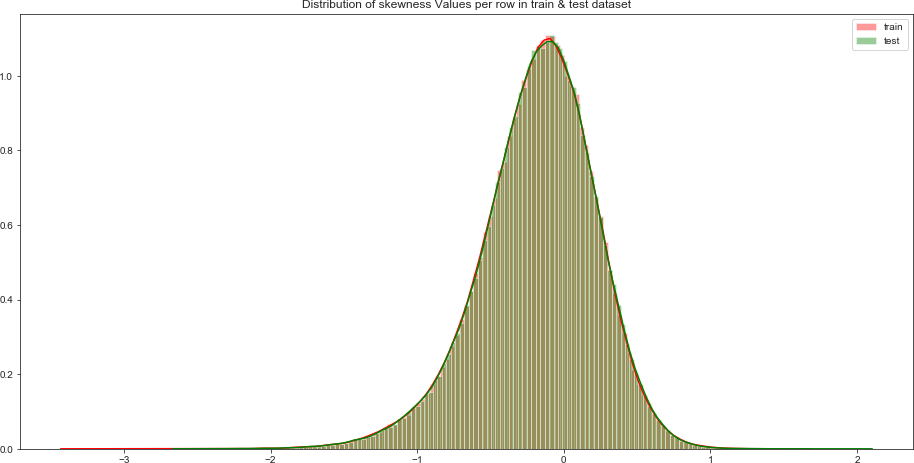


### Distribution of skewness values in train and test dataset

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

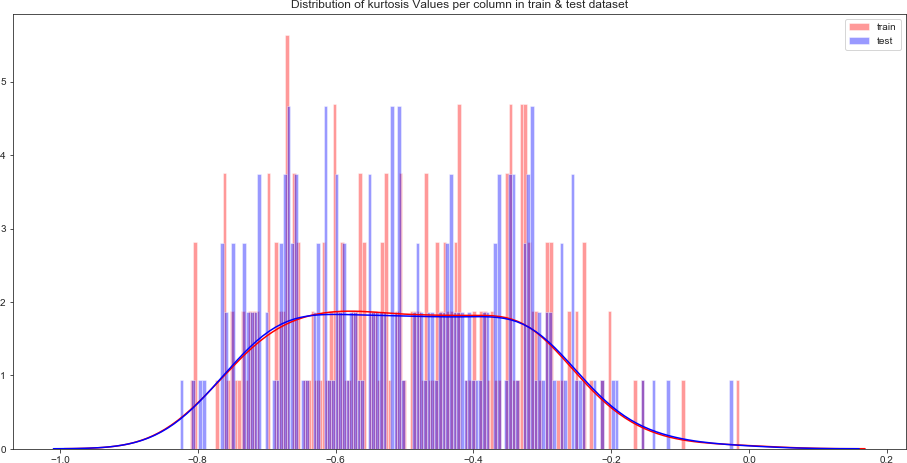


Let us look distribution of skewness values per column in train and test dataset:-

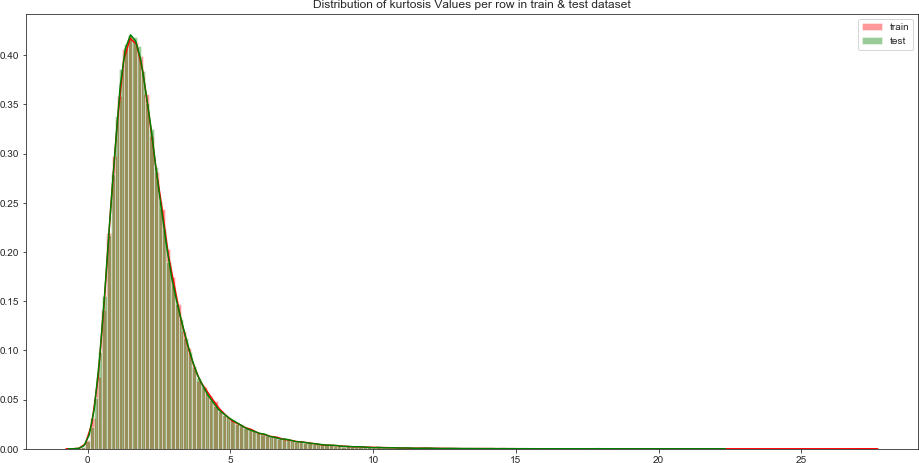


### Distribution of kurtosis values in train and test dataset

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:-



### Outlier analysis

We did not conduct outlier analysis in this project because the data was unbalanced, and it was also unnecessary for unbalanced data.

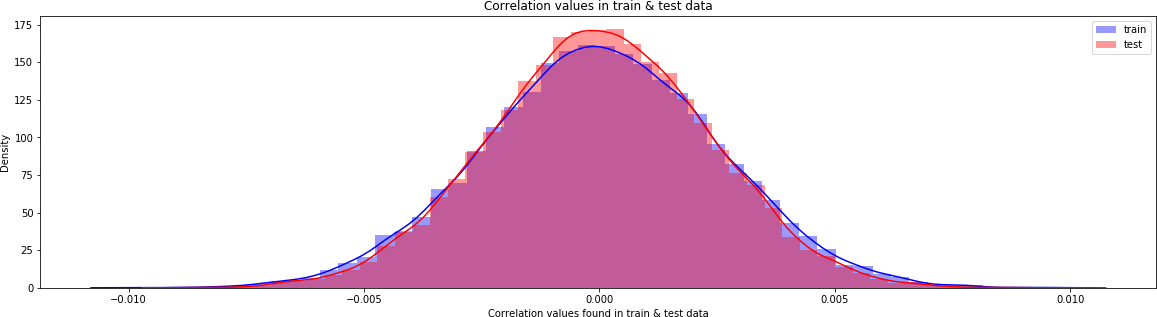
### Feature Selection

For modelling the dataset, feature selection is critical. Every dataset has both desirable and undesirable characteristics. Since the unnecessary features will have an impact on the model's results, we must remove them. We must use statistical techniques such as ANOVA, Chi-Square test, and correlation matrix statistical techniques to pick the best features. Using the Correlation matrix, we pick the best functions.

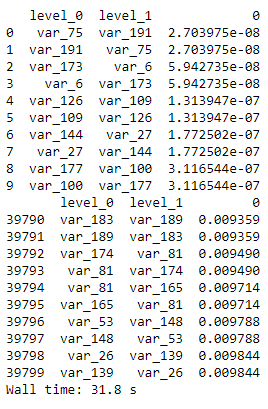
### Correlation matrix

The correlation matrix describes the linear relationship between attributes and aids in the creation of better models.

We can see from the correlation distribution plot that the correlation between train and test attributes is very weak. It implies that both train and test attributes are unrelated to one another.

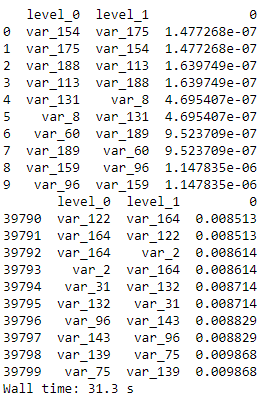
 **Correlation plot for train and test dataset.**

**Correlation in train attributes**



Observation – Correlation between the train attributes is very small.

**Correlation in Test attributes**



Observation – Correlation between the test attributes is very small.

### Feature engineering

Let us do some feature engineering by using

* + - 1. Permutation importance
      2. Partial dependence plots

### Permutation importance:-

In a random forest for classification and regression, the permutation variable value test is used. The variables that have the greatest impact on the model's prediction.

**Python code**

#Training & testing data: X=df\_train.drop(columns=['ID\_code','target'],axis=1) test=df\_test.drop(columns=['ID\_code'],axis=1) y=df\_train['target']

#Split the train data:- X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=42)

**Random Forest Classifier:-**

%%time rf\_model=RandomForestClassifier(n\_estimators=10,random\_state=42)

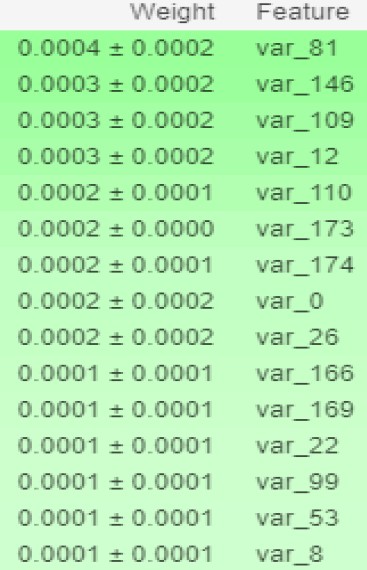
#fitting the model:- rf\_model.fit(X\_test,y\_test)

#Permutation Importance:-

from eli5.sklearn import PermutationImportance perm\_imp=PermutationImportance(rf\_model,random\_state=42)

#fitting the model:- perm\_imp.fit(X\_test,y\_test)

#Important Features:- eli5.show\_weights(perm\_imp,feature\_names=X\_test.columns.tolist(),top=200)



### R code:-

#Split the training data using simple random sampling train\_index<-sample(1:nrow(df\_train),0.75\*nrow(df\_train)) #train data

train\_data<-df\_train[train\_index,] #validation data

valid\_data<-df\_train[-train\_index,] #dimension of train and validation data dim(train\_data)

dim(valid\_data)

#Random forest classifier:-

#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)

#Feature importance by random forest- #Variable importance

VarImp<-importance(rf,type=2) VarImp

Observation: - Features having highest to lowest importance is in decreasing order as shown in the prediction. Features shown in green having higher positive impact on prediction and features having zero impact is in no colour.

### Partial dependence plots

The marginal effect of a variable on the class likelihood or classification is depicted graphically in a partial dependency map. Although feature significance depicts the variables have the greatest influence on predictions, partial dependency plots depict how a feature influences predictions.

**Python code**

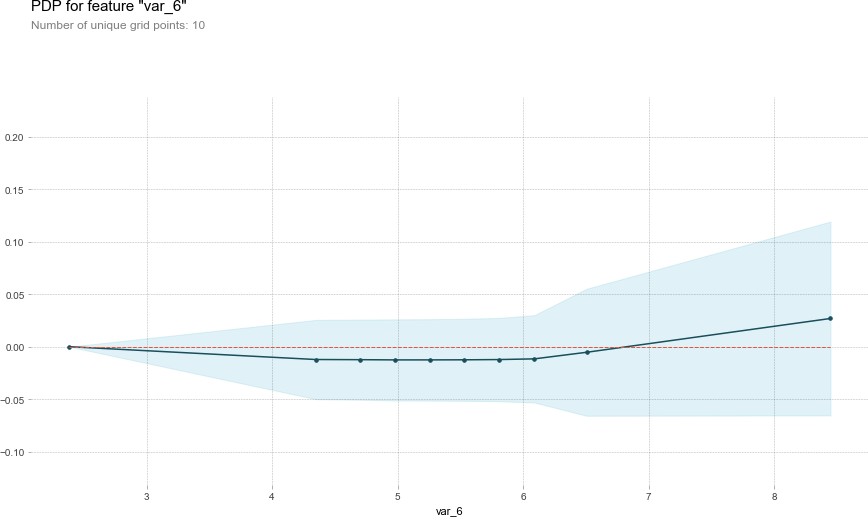
#Calculation of partial dependence plots on random forest:-

#we are observing impact of main features which are discovered in previous section by using PDP Plot.

features=[v for v in X\_test.columns if v not in ['ID\_code','target']]

pdp\_data=pdp.pdp\_isolate(rf\_model, dataset=X\_test, model\_features=features, feature='var\_6')

#Plot feature for var\_6:- pdp.pdp\_plot(pdp\_data,'var\_6') plt.show()



### Observation-

* The y axis does not indicate the predictor value, but rather how the value changes as the predictor variable changes.
* The level of confidence of var 6 is indicated by the blue shaded patch.
* A positive value on the y-axis indicates that for that particular value of the predictor variable, it is less likely to predict the correct class, while a positive value indicates that it has a positive effect on predicting the correct class.

**R Code**:-

#We will plot "var\_6"

par.var\_6 <- partial(rf, pred.var = c("var\_6"), chull = TRUE) plot.var\_6 <- autoplot(par.var\_6, contour = TRUE) plot.var\_6

### Modeling

* + 1. **Model Selection**

After all early stages of preprocessing, 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.

### Logistic Regression

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

Python Code:-

#Spliting the data via Sratified KFold Cross Validator:- #Training Data: X=df\_train.drop(['ID\_code','target'],axis=1) Y=df\_train['target']

#Stratified KFold Cross Validator:- skf=StratifiedKFold(n\_splits=5, random\_state=42, shuffle=True) for train\_index, valid\_index in skf.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 Model:-

%%time lr\_model=LogisticRegression(random\_state=42) #fitting the model-

lr\_model.fit(X\_train,y\_train) #Accuracy of model-

lr\_score=lr\_model.score(X\_train,y\_train)

print('Accuracy of lr\_model :',lr\_score) Accuracy of lr\_model : 0.9148942819107381

%%time

#Cross validation prediction of lr\_model- cv\_predict=cross\_val\_predict(lr\_model,X\_valid,y\_valid,cv=5) #Cross validation score- cv\_score=cross\_val\_score(lr\_model,X\_valid,y\_valid,cv=5) print('cross val score :',np.average(cv\_score))

cross val score : 0.9116728528566072

### R code

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

#Split the data using CreateDataPartition

train.index<-createDataPartition(df\_train$target,p=0.8,list=FALSE) train.data<-df\_train[train.index,]

valid.data<-df\_train[-train.index,]

#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(df\_test[,-c(1)])

#Logistic regression model set.seed(667)

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")

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter

#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") #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

lr\_pred<-predict(lr\_model,df\_test[,-c(1)],type='class')

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.

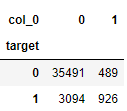
**Confusion Matrix :-**

#Confusion Matrix

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

cm = pd.crosstab(y\_valid,cv\_predict)

cm



**ROC AUC Curve**

#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('Receiver Operating Characterstics(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')

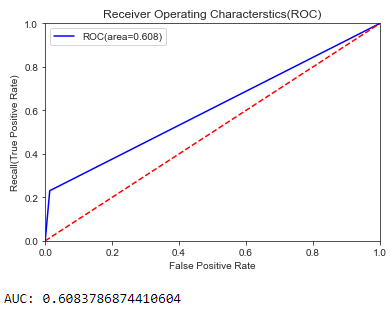
plt.show()

print('AUC:',roc\_auc)

#Classification Report

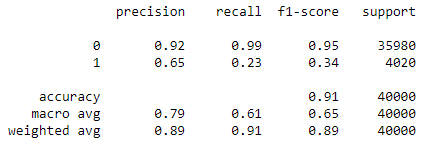
classification\_scores=classification\_report(y\_valid,cv\_predict)

print(classification\_scores)



Observation - On comparing roc\_auc\_score and model accuracy, model is not performing well on imbalanced data

**Classification Report -**



Observation - As we see that f1 score is high for the customers who will not make a transaction, compare to those who will make a transaction. So, we are going to change the algorithm.

### Oversample Minority Class:-

-Adding more copies of minority class.

-It can be a good option when we don’t have that much large data to work.

-Drawback of this process is we are adding info. That can lead to overfitting or poor performance on test data.

### Undersample Majority class:-

-Removing some copies of majority class.

-It can be a good option if we have very large amount of data say in millions to work.

-Drawback of this process is we are removing some valuable info. that can leads to underfitting & poor performance on test data.

As per the drawbacks of both the model we will use SMOTE (Synthetic Minority Oversampling technique) that is better than the above as compare to above one's.

### 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 sampling method.

The first step is to ignore the majority class examples.

For every minority instance, choose its K nearest neighbors

For e.g. 300 % replication, 3 neighbors are chosen.

Create new instances halfway between the first instance and its neighbors.

**Python Code:-**

%%time

from imblearn.over\_sampling import SMOTE #SMOTE:-

sm = SMOTE(random\_state=42, ratio=1.0) #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)

B**uilding Logistsic regression model on synthetic data points:-**

%%time

#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)

Accuracy of the smote\_model : 0.7986096635677659

%%time

#Cross validation prediction for SMOTE:- 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 validation score :',np.average(cv\_score))

Cross validation score : 0.800597554196776

#ROC\_AUC SCORE:-

roc\_score = roc\_auc\_score(y\_smote\_v,cv\_pred)

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

#ROC\_AUC Curve:-

plt.figure()

false\_positive\_rate,recall,thresholds=roc\_curve(y\_smote\_v,cv\_pred)

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

plt.title('Receiver Operating Characterstics(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])

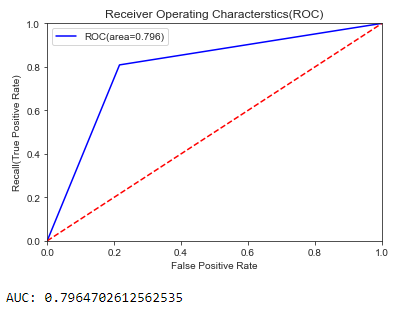
plt.ylabel('Recall(True Positive Rate)')

plt.xlabel('False Positive Rate')

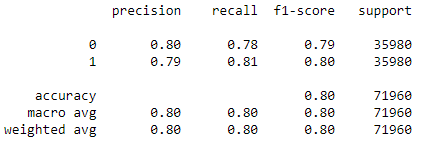
plt.show()

print('AUC:',roc\_auc)

ROC Score – 0.79647



Observation – On comparing the ROC-AUC score we can find that the model is performing better.



Observation - We can see that f1 score is high for customers who will not make a transaction, as well as who will not make a transaction.

### 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.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

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter:-

#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)

### 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 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,  }  #Training lgbm model:- 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  Training until validation scores don't improve for 5000 rounds. | | |
| [1000] | training's auc: 0.938996 | valid\_1's auc: 0.885963 |
| [2000] | training's auc: 0.958629 | valid\_1's auc: 0.890769 |
| [3000] | training's auc: 0.972001 | valid\_1's auc: 0.89195 |
| [4000] | training's auc: 0.981625 | valid\_1's auc: 0.892447 |
| [5000] | training's auc: 0.988357 | valid\_1's auc: 0.892444 |

[6000] training's auc: 0.992858 valid\_1's auc: 0.892633

[7000] training's auc: 0.995834 valid\_1's auc: 0.892332

[8000] training's auc: 0.997652 valid\_1's auc: 0.89205

[9000] training's auc: 0.99874 valid\_1's auc: 0.891803

[10000] training's auc: 0.999366 valid\_1's auc: 0.891481 Did not meet early stopping. Best iteration is:

[10000] training's auc: 0.999366 valid\_1's auc: 0.891481 R Code:-

#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(df\_test[,-c(1)])

#training data

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

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

#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)

#Training the lgbm model set.seed(7663)

lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,eval\_freq =1 000,

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

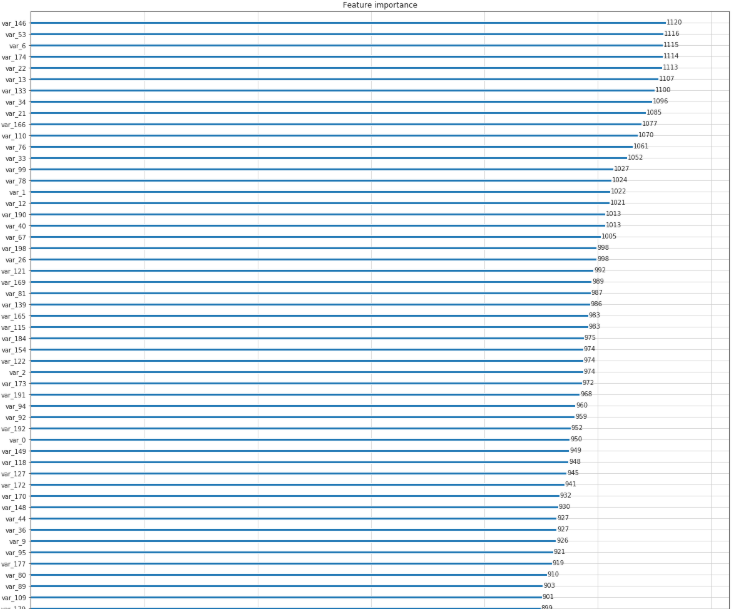
#lgbm model performance on test data set.seed(6532)

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)

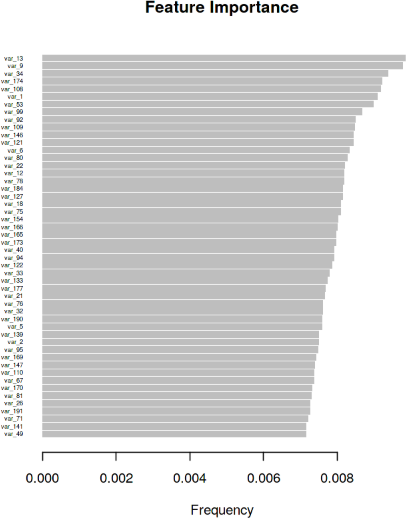
### Important features plot Python code

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



**R code**

tree\_imp <- lgb.importance(lgbm.model, percentage = TRUE) lgb.plot.importance(tree\_imp, top\_n = 50, measure = "Frequency", left\_margin = 10)



**Chapter 3**

**Conclusion**

### Model Evaluation

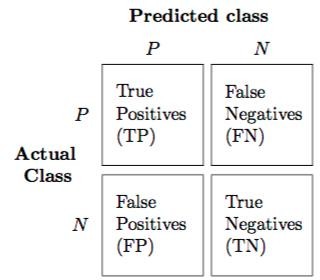
Now, we have a three models for predicting the target variable, but we need to decide which model better for this project. There are many metrics used for model evaluation. Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

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 Accuracy = (𝑇𝑃+𝑇𝑁)/𝑇𝑜𝑡𝑎𝑙 𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑖𝑜𝑛𝑠

Misclassification error: - The ratio of incorrect predictions to total predictions Error rate = (𝐹𝑁+𝐹𝑃)/𝑇𝑜𝑡𝑎𝑙 𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑖𝑜𝑛𝑠

Accuracy=1-Error rate

True Positive Rate (TPR) = 𝑇𝑃/(𝑇𝑃+𝐹𝑁) ↔ Recall Precision = 𝑇𝑃/(𝑇𝑃+𝐹𝑃 )

True Negative Rate (TNR) = 𝑇𝑁/(𝑇𝑁+𝐹𝑃) ↔ Specificity False Positive Rate (FPR) = 𝐹𝑃/(𝐹𝑃+𝑇𝑁 )

False Negative rate (FNR) = 𝐹𝑁/(𝐹𝑁+𝑇𝑃 )

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

F1 score = 2∗𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛∗𝑅𝑒𝑐𝑎𝑙𝑙𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛+𝑅𝑒𝑐𝑎𝑙𝑙

### 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.

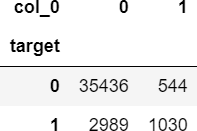
Roc curve is plotted true positive rate or Recall on y axis against false positive rate or specificity on x axis. The larger the area under the roc curve better the performance of the model.

### Logistic Regression

#Confusion matrix:-

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

cm



#ROC\_AUC\_Curve:-

plt.figure() false\_positive\_rate,recall,thresholds=roc\_curve(y\_valid,cv\_predict) roc\_auc=auc(false\_positive\_rate,recall)

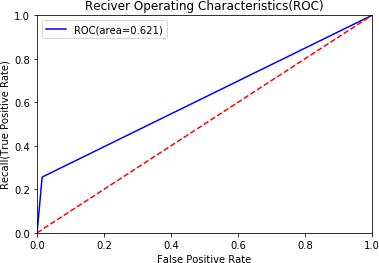
plt.title('Reciver Operating Characteristics(ROC)') plt.plot(false\_positive\_rate,recall,'b',label='ROC(area=%0.3f)' %roc\_auc) plt.legend()

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

plt.xlim([0.0,1.0])

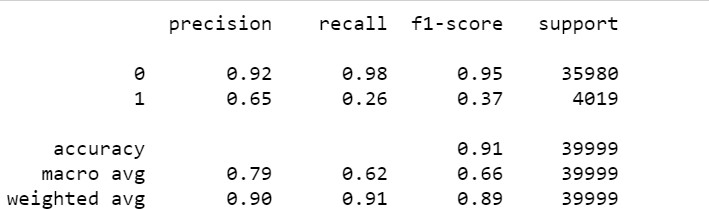
plt.ylim([0.0,1.0]) plt.ylabel('Recall(True Positive Rate)') plt.xlabel('False Positive Rate') plt.show()

print('AUC:',roc\_auc)



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

### Classification report



We can observed 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

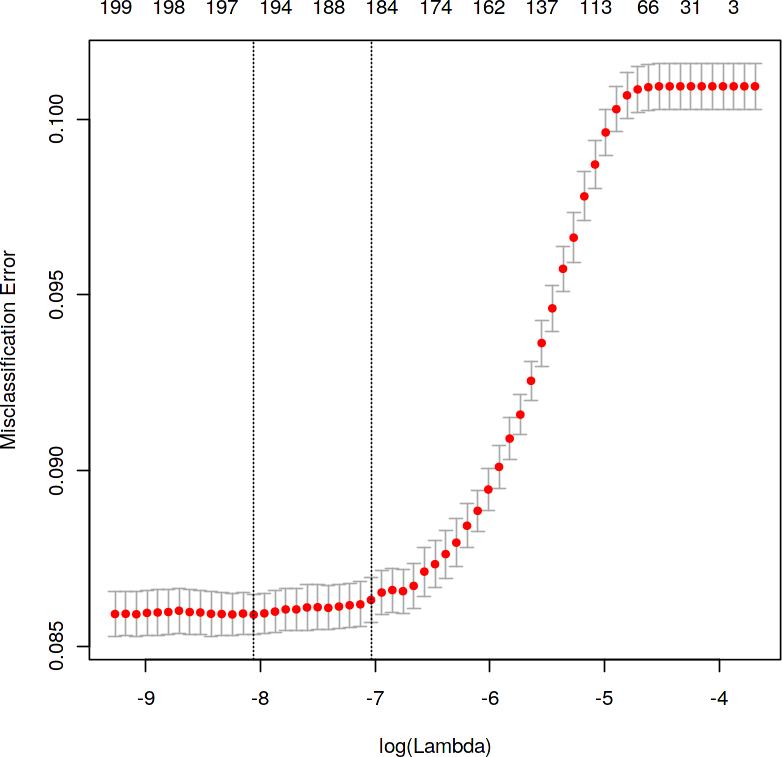
set.seed(8909)

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

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter

#Minimum lambda cv\_lr$lambda.min

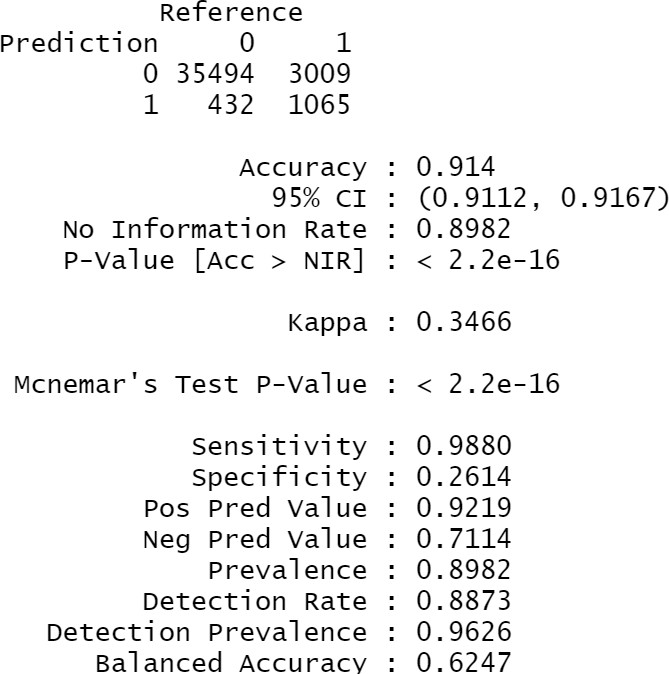
#plot the auc score vs log(lambda) plot(cv\_lr)



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

#Confusion Matrix:- set.seed(689)

#actual target variable target<-valid.data$target #convert to factor target<-as.factor(target)



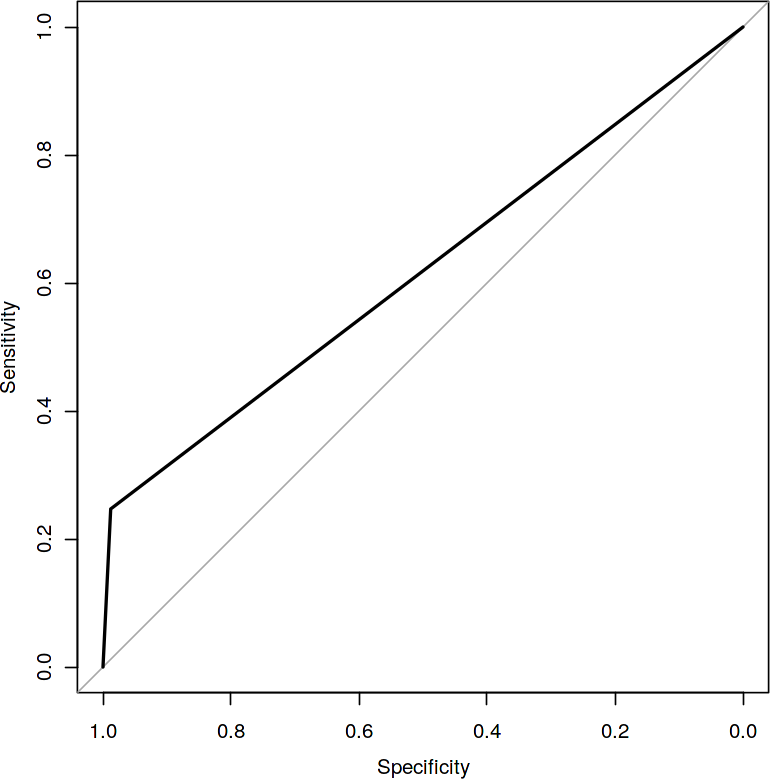
#predicted target variable #convert to factor

cv\_predict.lr<-as.factor(cv\_predict.lr) confusionMatrix(data=cv\_predict.lr,reference=target)

### Reciever operating characteristics(ROC)-Area under curve(AUC) score and curve

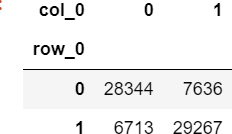
#ROC\_AUC score and curveset.seed(892) cv\_predict.lr<-as.numeric(cv\_predict.lr) roc(data=valid.data[,-

c(1,2)],response=target,predictor=cv\_predict.lr,auc=TRUE,plot=TRUE)



### Python code

**Synthetic Minority Oversampling Technique (SMOTE)**



%%time #Confusion matrix:-

cm=confusion\_matrix(y\_smote\_v,cv\_pred) cm=pd.crosstab(y\_smote\_v,cv\_pred)

**Reciever operating characteristics (ROC)-Area under curve (AUC) score and curve**

#ROC\_AUC Curve:- plt.figure()

false\_positive\_rate,recall,thresholds=roc\_curve(y\_smote\_v,cv\_pred) roc\_auc=auc(false\_positive\_rate,recall)

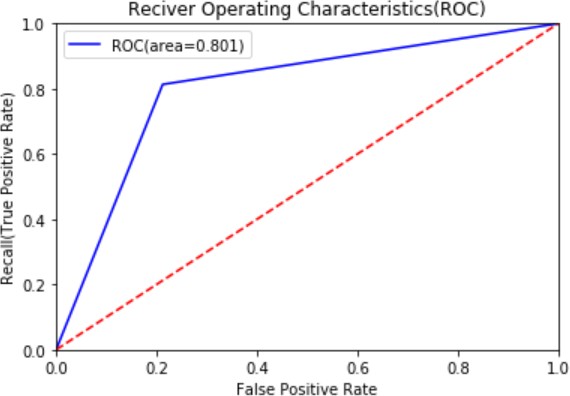
plt.title('Reciver Operating Characteristics(ROC)') plt.plot(false\_positive\_rate,recall,'b',label='ROC(area=%0.3f)' %roc\_auc) plt.legend()

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

plt.xlim([0.0,1.0])

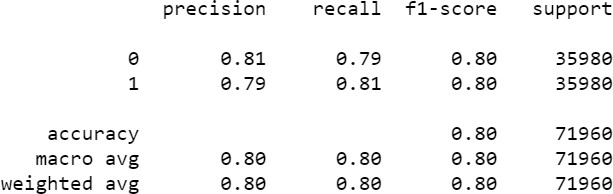
plt.ylim([0.0,1.0]) plt.ylabel('Recall(True Positive Rate)') plt.xlabel('False Positive Rate') plt.show()

print('AUC:',roc\_auc)



**Classification report**

#Classification Report:- scores=classification\_report(y\_smote\_v,cv\_pred) print(scores)



We can observed that smote model is performing well on imbalance data compare to baseline logistic regression.

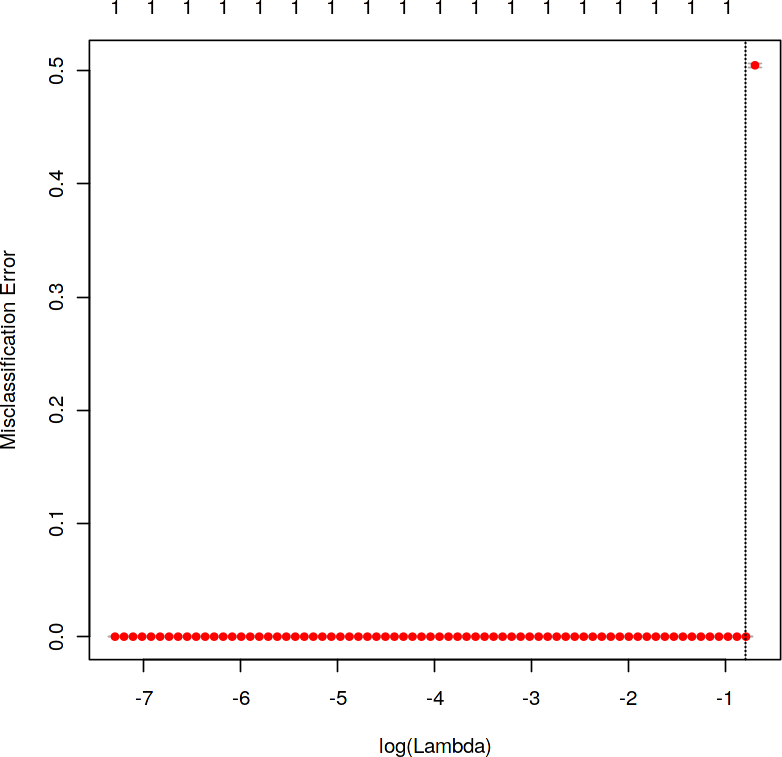
### R code

**Random Oversampling Examples (ROSE)**

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter:-

#Minimum lambda cv\_rose$lambda.min

#plot the auc score vs log(lambda) plot(cv\_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)

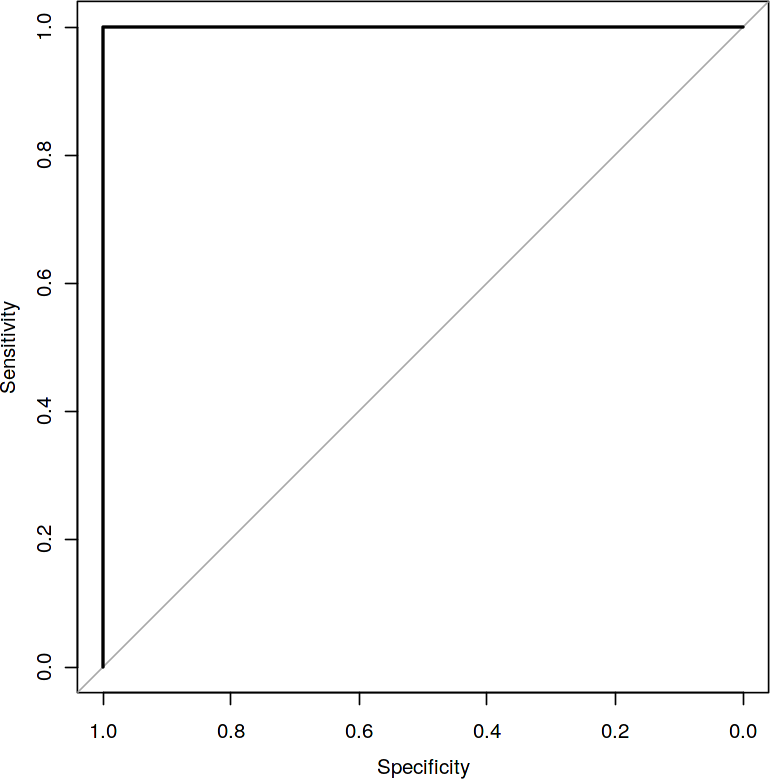
### Reciever operating characteristics (ROC)-Area under curve(AUC) score and curve

#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)



I tried different ways to get good accuracy like changing count of one target class variable. Finally got area under ROC curve is 1 but this may not be possible.

### Model Selection

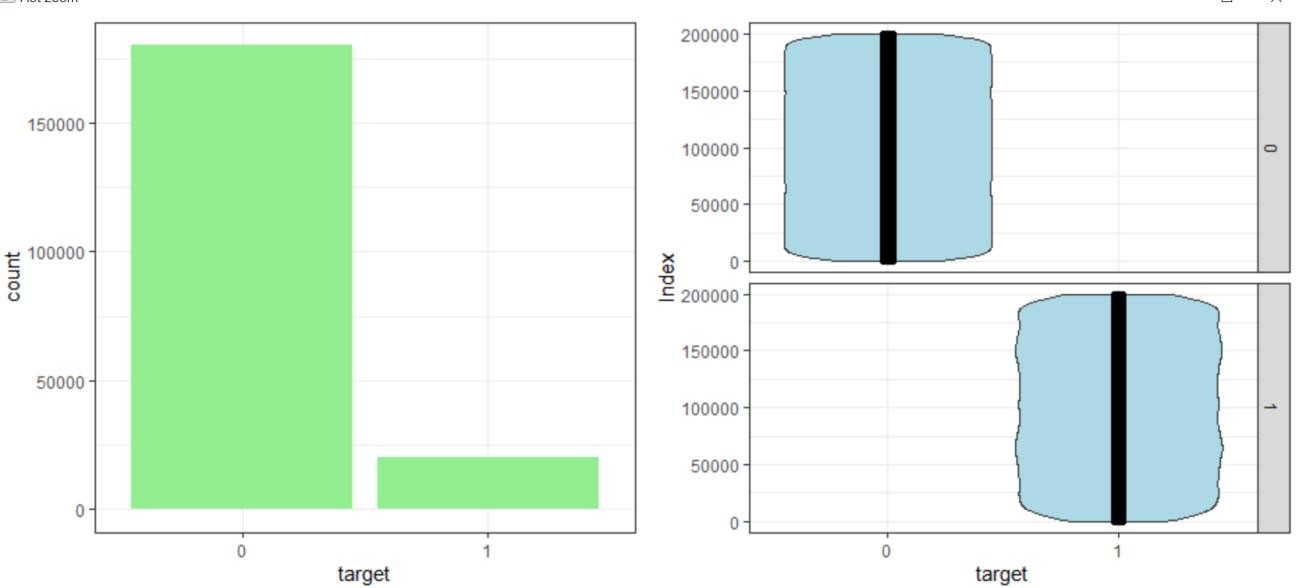
When we compare scores of area under the ROC curve of all the models for an imbalanced data. We could conclude that below points as follow,

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.
3. Baseline logistic regression model is performed well on balanced data.
4. LightGBM model performed well on imbalanced data.

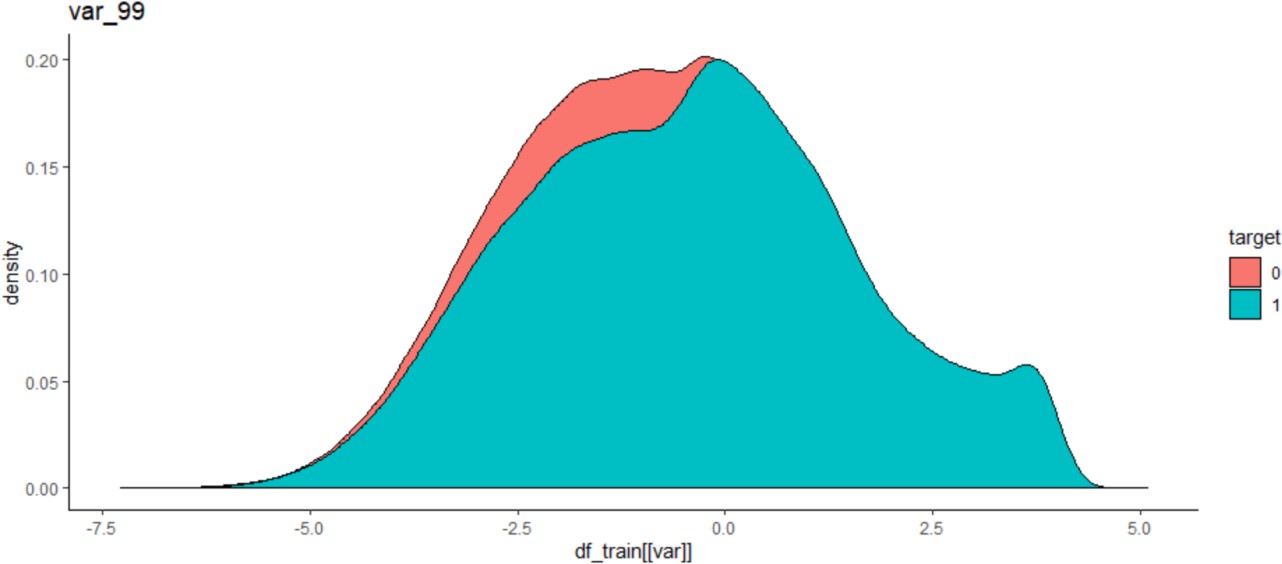
Finally LightGBM is best choice for identifying which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

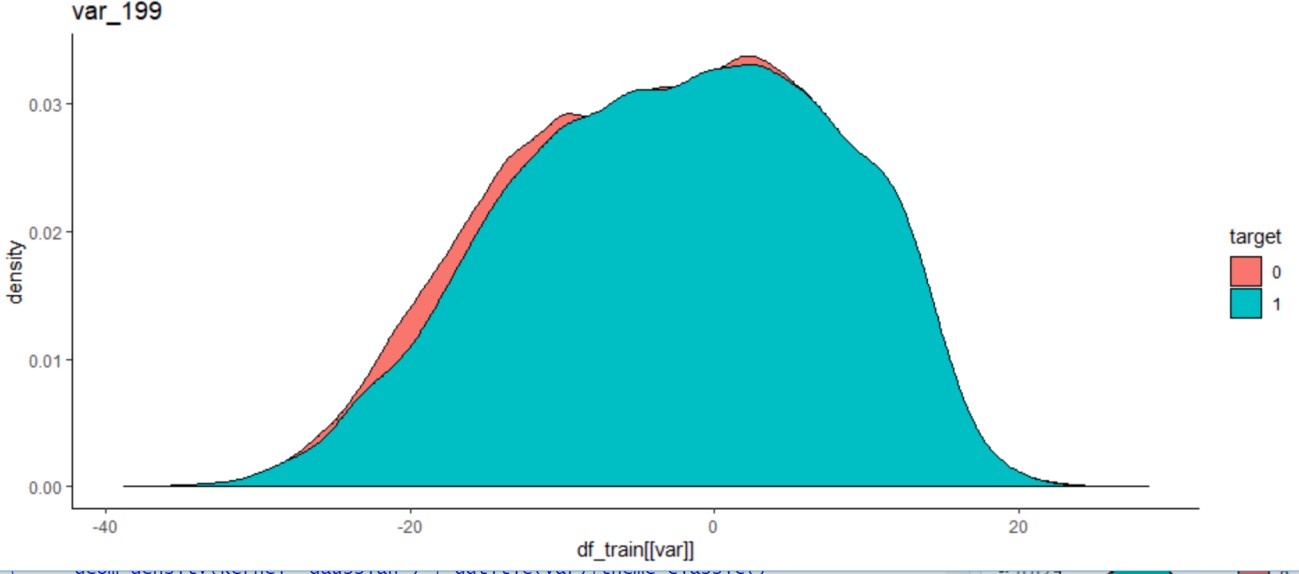
## Appendix A - Extra Figures

### ggplot2 visualizations Target classes count

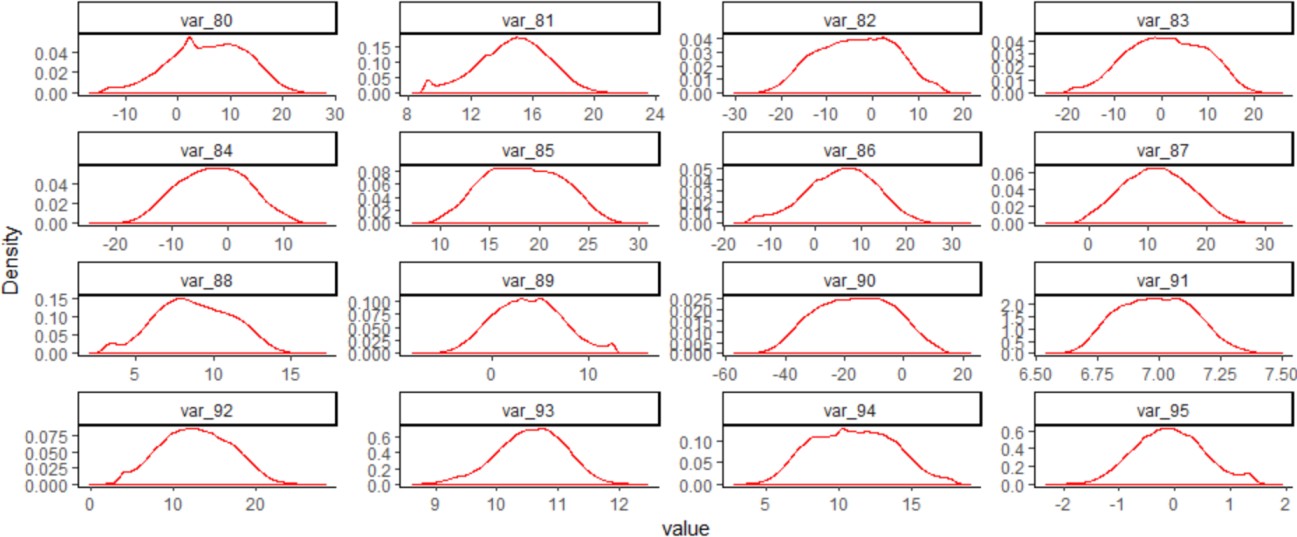


**Distribution of train attributes**

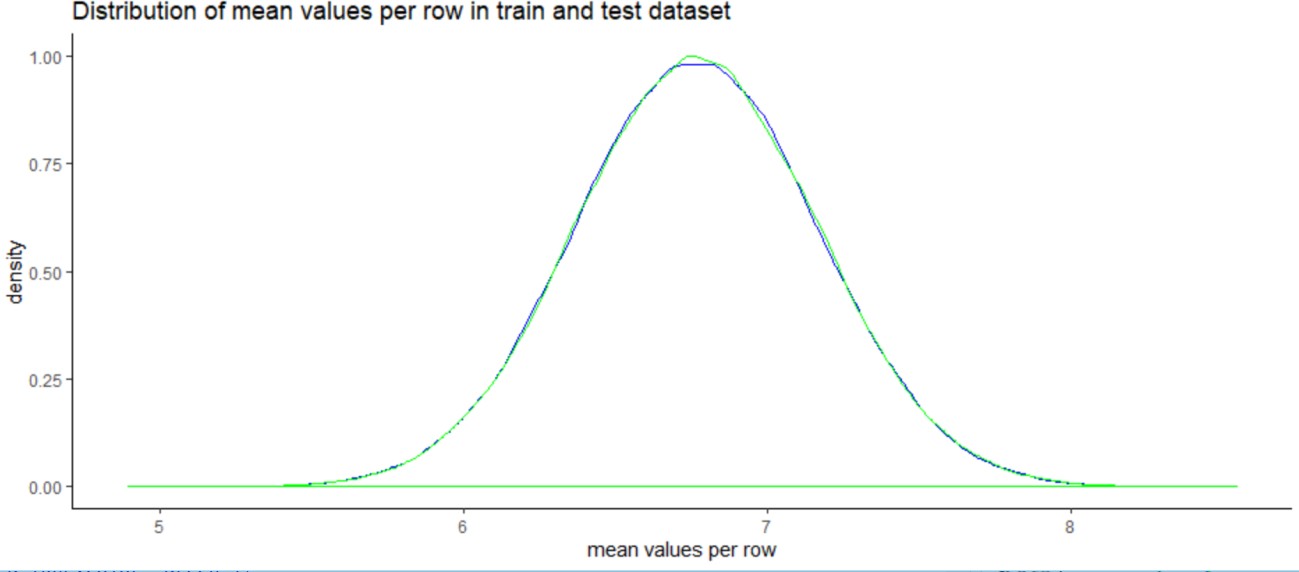




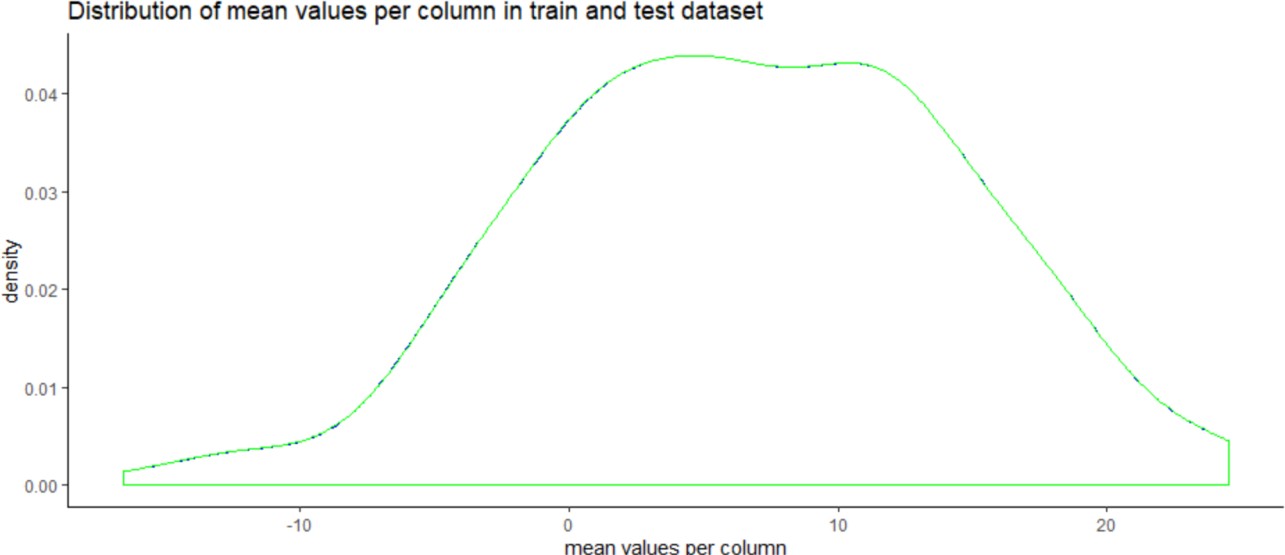
**Distribution of test attributes**

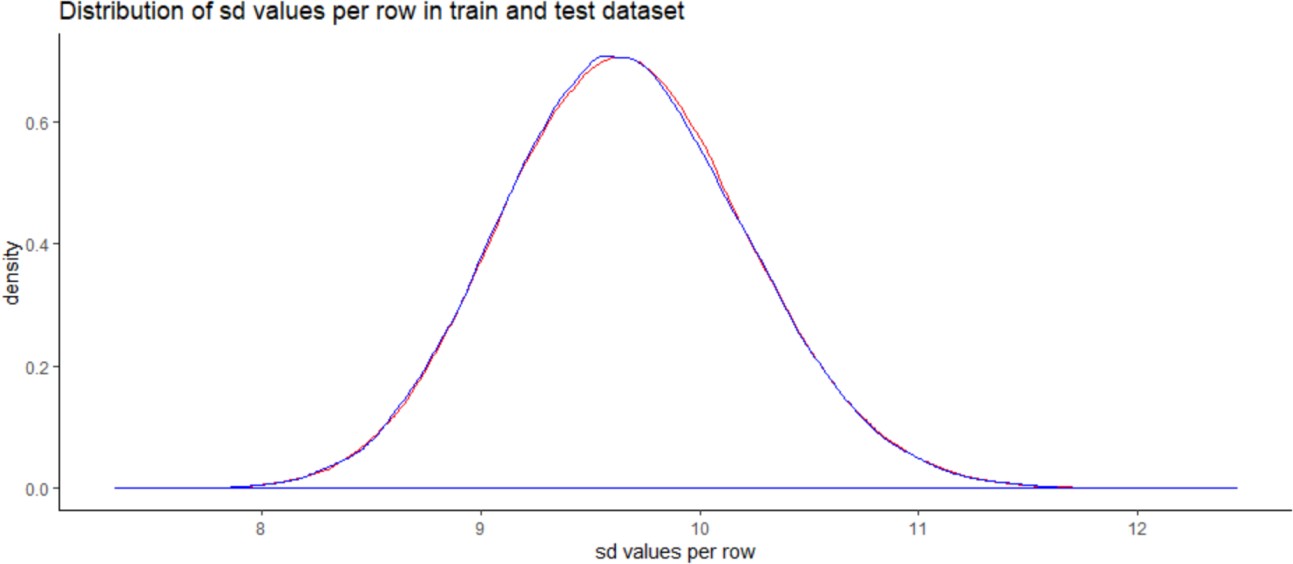


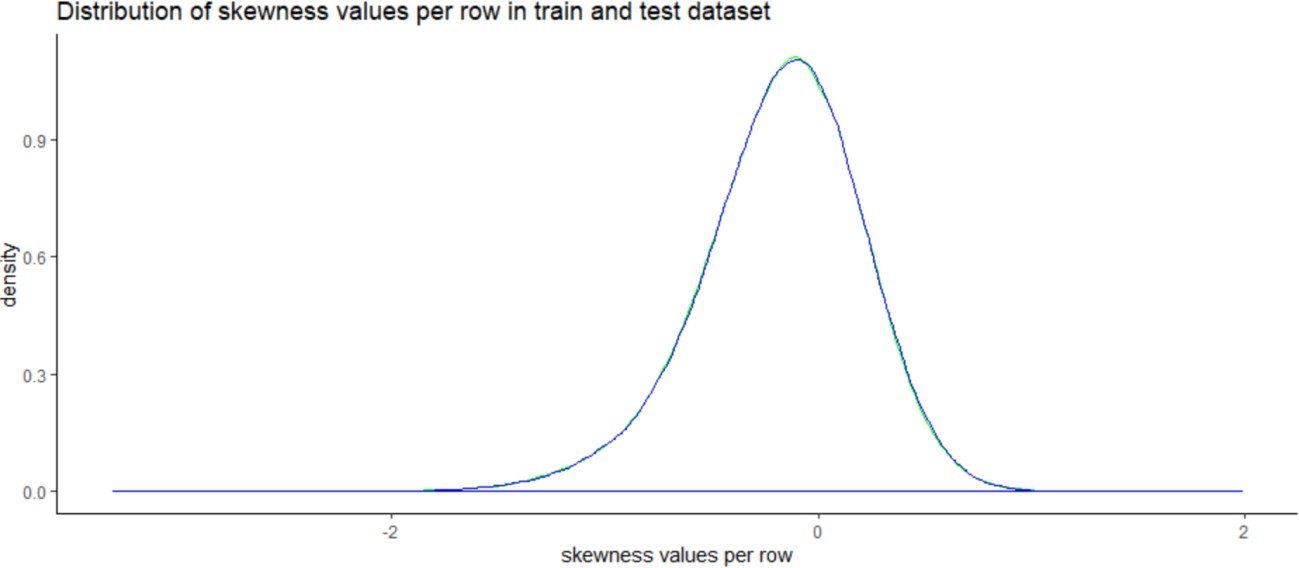
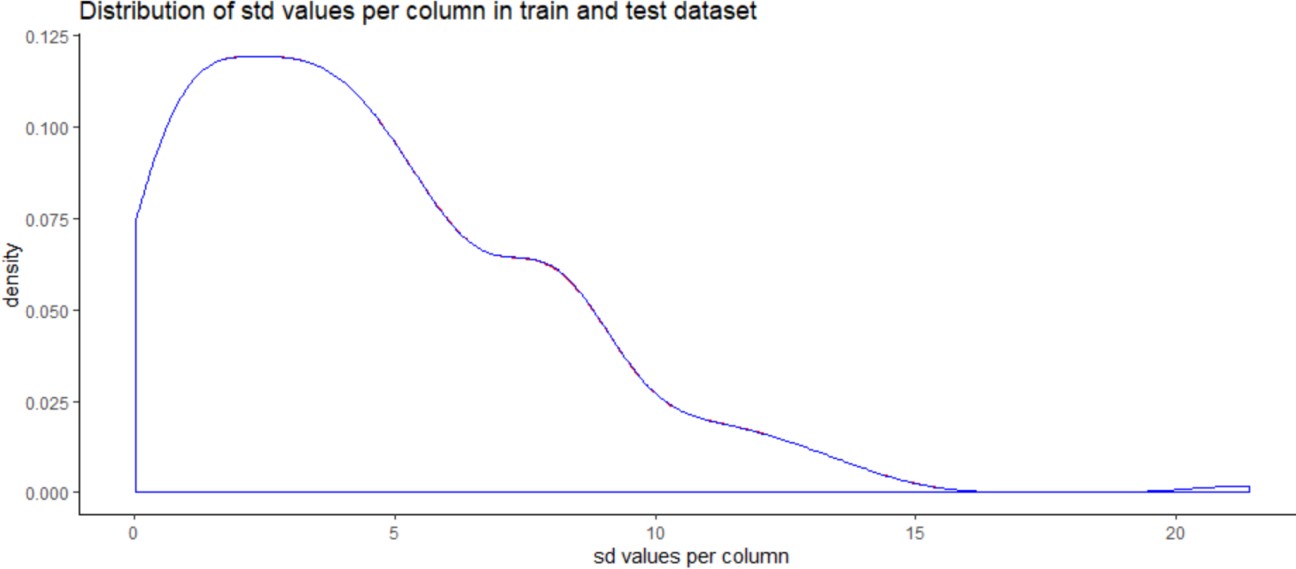
**Distribution of mean values per row in train & test data**

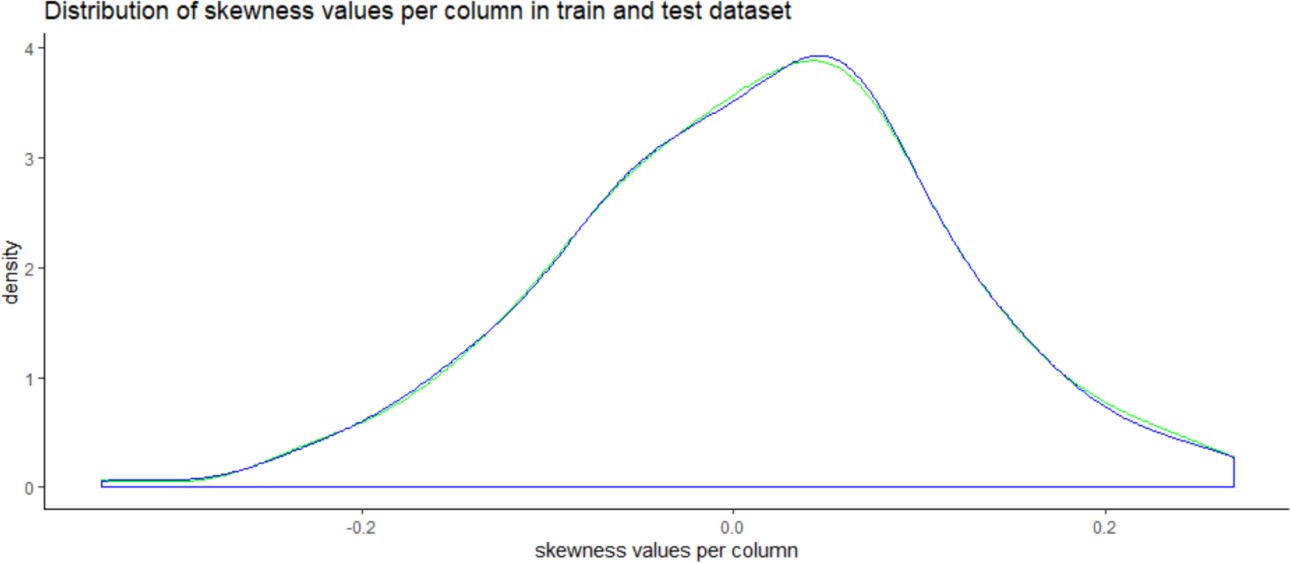


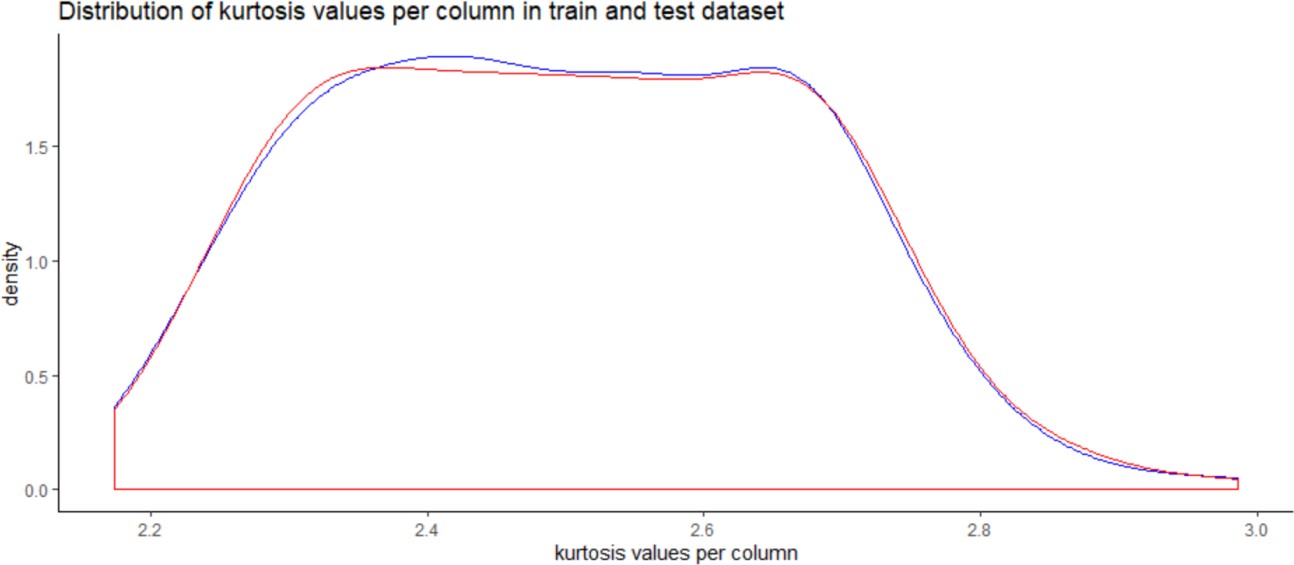
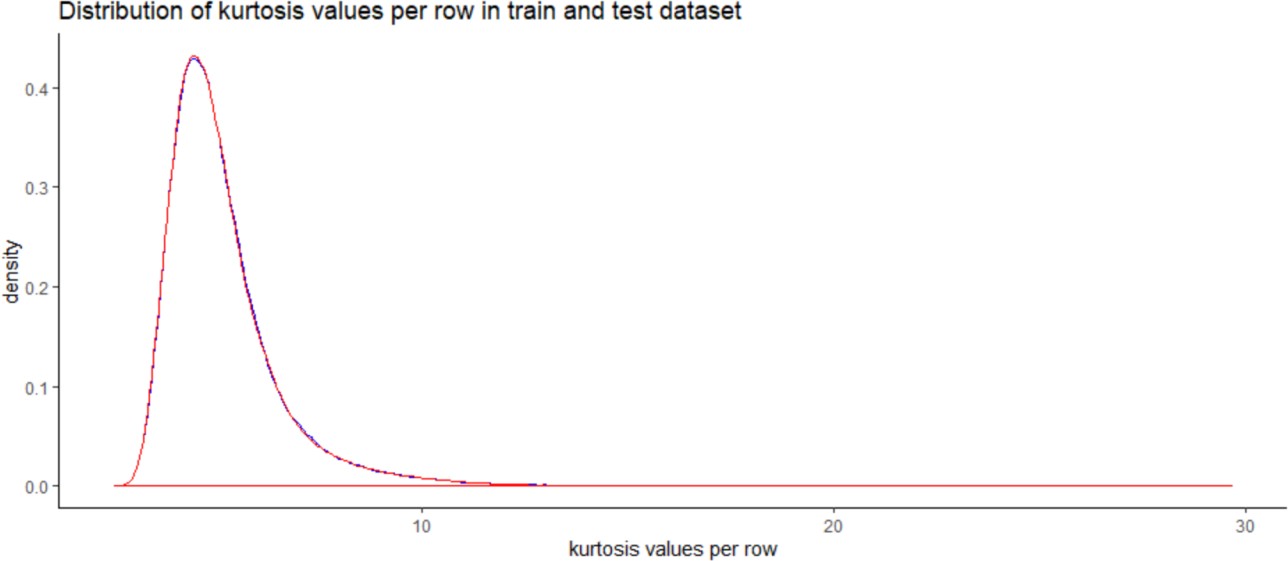
**Distribution of mean values per columns in train & test data**











**Appendix B – Complete Python and R Code**

### Python Code

**Exploratory Data Analysis**

#Loading Libraries:-

import os

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt import lightgbm as lgb

import eli5

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

from pdpbox import pdp, get\_dataset, info\_plots from sklearn.model\_selection import StratifiedKFold from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix,roc\_auc\_score,roc\_curve,classification\_report,roc\_curve,auc

random\_state=42 np.random.seed(random\_state) import warnings warnings.filterwarnings('ignore')

os.chdir("D:/Practice-Python") os.getcwd()

Importing the train dataset df\_train=pd.read\_csv("train.csv") pd.options.display.max\_columns = None

#Shape of the dataset df\_train.shape

#Summary of the dataset df\_train.describe()

#Target Class Count target\_class=df\_train['target'].value\_counts() print('Count of the target class :\n',target\_class)

#Percentage of target class count per\_target\_class=df\_train['target'].value\_counts()/len(df\_train)\*100 print('Percentage of target class count :\n',per\_target\_class)

#Count plot & violin plot for target class fig,ax=plt.subplots(1,2,figsize=(20,5)) sns.countplot(df\_train.target.values,ax=ax[0],palette='spring')

sns.violinplot(x=df\_train.target.values,y=df\_train.index.values,ax=ax[1],palette='spring')

sns.stripplot(x=df\_train.target.values,y=df\_train.index.values,jitter=True,color='black',lin ewidth=0.5,size=0.5,alpha=0.5,ax=ax[1],palette='spring')

ax[0].set\_xlabel('Target') ax[1].set\_xlabel('Target') ax[1].set\_ylabel('Index')

#Distribution of train attributes-

def plot\_train\_attribute\_distribution(t0,t1,label1,label2,train\_attributes): i=0

sns.set\_style('darkgrid')

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(t0[attribute],hist=False,label=label1) sns.distplot(t1[attribute],hist=False,label=label2) plt.legend()

plt.xlabel('Attribute',)

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

**Observing first 100 train attributes** #Corresponding to negative class- t0=df\_train[df\_train.target.values==0]

#Corresponding to possitive class- t1=df\_train[df\_train.target.values==1]

#train attributes from 2 to 102 - train\_attributes=df\_train.columns.values[2:102]

#Plot distribution of train attributes- plot\_train\_attribute\_distribution(t0,t1,'0','1',train\_attributes)

### Observing next 100 train attributes

#train attributes from 102 to 202 - train\_attributes=df\_train.columns.values[102:202]

#Plot distribution of train attributes- plot\_train\_attribute\_distribution(t0,t1,'0','1',train\_attributes)

#Importing the test dataset:- df\_test=pd.read\_csv("test.csv")

#Distribution of test attributes-

def plot\_test\_attribute\_distribution(test\_attributes): i=0

sns.set\_style('darkgrid')

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)

#test attributes from 101 to 202- test\_attributes=df\_test.columns.values[101:202]

#Plot the distribution of test attributes- plot\_test\_attribute\_distribution(test\_attributes)

#Distribution of Mean Values per column in train & 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='red',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='blue',kde=True,bins=150,label='t est')

plt.title('Distribution of Mean Values per column in train & test dataset') plt.legend()

plt.show()

#Distribution of Mean Values per column in train & test dataset:- plt.figure(figsize=(16,8))

#Distribution plot for mean values per rows in train attributes:

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

#Distribution plot for mean values per rows in test attributes:

sns.distplot(df\_test[test\_attributes].mean(axis=1),color='blue',kde=True,bins=150,label='t est')

plt.title('Distribution of Mean Values per row in train & test dataset') plt.legend()

plt.show()

#Distribution of S.D Values per column in train & 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 S.D values per column in train attributes:

sns.distplot(df\_train[train\_attributes].std(axis=0),color='blue',kde=True,bins=150,label='t rain')

#Distribution plot for S.D values per column in test attributes:

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

plt.title('Distribution of S.D Values per column in train & test dataset') plt.legend()

plt.show()

#Distribution of S.D Values per column in train & test dataset:- plt.figure(figsize=(16,8))

#Distribution plot for S.D values per rows in train attributes:

sns.distplot(df\_train[train\_attributes].std(axis=1),color='blue',kde=True,bins=150,label='t rain')

#Distribution plot for S.D values per rows in test attributes:

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

plt.title('Distribution of S.D Values per row in train & test dataset') plt.legend()

plt.show()

#Distribution of skew Values per column in train & 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 skew values per column in train attributes:

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

#Distribution plot for skew values per column in test attributes:

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

plt.title('Distribution of skewness Values per column in train & test dataset') plt.legend()

plt.show()

#Distribution of skew Values per column in train & test dataset:- plt.figure(figsize=(16,8))

#Distribution plot for skew values per rows in train attributes:

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

#Distribution plot for skew values per rows in test attributes:

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

plt.title('Distribution of skewness Values per row in train & test dataset') plt.legend()

plt.show()

#Distribution of kurtosis Values per column in train & 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='red',kde=True,bins=150,lab el='train')

#Distribution plot for kurtosis values per column in test attributes: sns.distplot(df\_test[test\_attributes].kurtosis(axis=0),color='blue',kde=True,bins=150,label

='test')

plt.title('Distribution of kurtosis Values per column in train & test dataset') plt.legend()

plt.show()

#Distribution of kurtosis Values per column in train & test dataset:- plt.figure(figsize=(16,8))

#Distribution plot for kurtosis values per rows in train attributes:

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

#Distribution plot for kurtosis values per rows in test attributes:

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

plt.title('Distribution of kurtosis Values per row in train & test dataset') plt.legend()

plt.show()

#Finding the missing values in train & test dataset:- 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 attiributes- train\_attributes=df\_train.columns.values[2:202]

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

train\_correlation=train\_correlation[train\_correlation['level\_0']!=train\_correlation['level\_1 ']]

print(train\_correlation.head(10)) print(train\_correlation.tail(10))

#Correlation in test attiributes- test\_attributes=df\_test.columns.values[1:201]

test\_correlation=df\_test[train\_attributes].corr().abs().unstack().sort\_values(kind='quickso rt').reset\_index()

test\_correlation=test\_correlation[test\_correlation['level\_0']!=test\_correlation['level\_1']] print(test\_correlation.head(10))

print(test\_correlation.tail(10))

**Correlation plot for train and test data:** train\_correlation=df\_train[train\_attributes].corr() train\_correlation=train\_correlation.values.flatten() train\_correlation=train\_correlation[train\_correlation!=1]

test\_correlation=df\_test[test\_attributes].corr() test\_correlation=test\_correlation.values.flatten() test\_correlation=test\_correlation[test\_correlation!=1]

plt.figure(figsize=(20,5)) sns.distplot(train\_correlation,color="blue",label="train") sns.distplot(test\_correlation,color="red",label="test") plt.xlabel("Correlation values found in train & test data") plt.ylabel("Density")

plt.title ("Correlation values in train & test data") plt.legend()

**Feature Engineering :-** Performing feature engineering by using-

* Permutation Importance
* Partial dependence plots

#Training & testing data:

X=df\_train.drop(columns=['ID\_code','target'],axis=1) test=df\_test.drop(columns=['ID\_code'],axis=1) y=df\_train['target']

**Building a simple model to find the features which are more important:** #Split the train data:- X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=42)

### Random Forest Classifier:-

rf\_model=RandomForestClassifier(n\_estimators=10,random\_state=42)

#fitting the model:- rf\_model.fit(X\_test,y\_test)

#Permutation Importance:-

from eli5.sklearn import PermutationImportance perm\_imp=PermutationImportance(rf\_model,random\_state=42)

#fitting the model:- perm\_imp.fit(X\_test,y\_test)

#Important Features:- eli5.show\_weights(perm\_imp,feature\_names=X\_test.columns.tolist(),top=200)

#Calculation of partial dependence plots on random forest:-

#we are observing impact of main features which are discovered in previous section by using PDP Plot.

features=[v for v in X\_test.columns if v not in ['ID\_code','target']]

pdp\_data=pdp.pdp\_isolate(rf\_model, dataset=X\_test, model\_features=features, feature='var\_6')

#Plot feature for var\_6:- pdp.pdp\_plot(pdp\_data,'var\_6') plt.show()

#Plot feature for var\_53:-

pdp\_data=pdp.pdp\_isolate(rf\_model, dataset=X\_test, model\_features=features, feature='var\_53')

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

### Logistic Regression Model:-

#Spliting the data via Sratified KFold Cross Validator:-

#Training Data: X=df\_train.drop(['ID\_code','target'],axis=1) Y=df\_train['target']

#Stratified KFold Cross Validator:- skf=StratifiedKFold(n\_splits=5, random\_state=42, shuffle=True) for train\_index, valid\_index in skf.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)

lr\_model=LogisticRegression(random\_state=42) #fitting the model-

lr\_model.fit(X\_train,y\_train)

#Accuracy of model- lr\_score=lr\_model.score(X\_train,y\_train) print('Accuracy of lr\_model :',lr\_score)

#Cross validation prediction of lr\_model- cv\_predict=cross\_val\_predict(lr\_model,X\_valid,y\_valid,cv=5) #Cross validation score- cv\_score=cross\_val\_score(lr\_model,X\_valid,y\_valid,cv=5) print('cross val score :',np.average(cv\_score))

#Confusion matrix:- cm=confusion\_matrix(y\_valid,cv\_predict) cm=pd.crosstab(y\_valid,cv\_predict)

cm

#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:- classification\_scores=classification\_report(y\_valid,cv\_predict) print(classification\_scores)

#Model performance on test data:- X\_test=df\_test.drop(['ID\_code'],axis=1) lr\_pred=lr\_model.predict(X\_test) print(lr\_pred)

from imblearn.over\_sampling import SMOTE #SMOTE:-

sm = SMOTE(random\_state=42, ratio=1.0) #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 for SMOTE:- 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 validation score :',np.average(cv\_score))

#Confusion matrix:- cm=confusion\_matrix(y\_smote\_v,cv\_pred) cm=pd.crosstab(y\_smote\_v,cv\_pred)

#ROC\_AUC SCORE:-

roc\_score=roc\_auc\_score(y\_smote\_v,cv\_pred) print('ROC score:',roc\_score)

#ROC\_AUC Curve:- plt.figure()

false\_positive\_rate,recall,thresholds=roc\_curve(y\_smote\_v,cv\_pred) 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\_smote\_v,cv\_pred) print(scores)

#Predicting the model- X\_test=df\_test.drop(['ID\_code'],axis=1) smote\_pred=smote.predict(X\_test) print(smote\_pred)

LightGBM:-

#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,

}

#Training lgbm model:- num\_rounds=10000 lgbm=

lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid],verbose\_eval=10 00,early\_stopping\_rounds = 5000)

lgbm

**LGBM model performance on test data:-** X\_test=df\_test.drop(['ID\_code'],axis=1) #Predict the model:-

#probability predictions

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

#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)

Plotting important features:-

lgb.plot\_importance(lgbm,max\_num\_features=50,importance\_type="split",figsize=(20,5 0))

#Final submission:- df\_sub=pd.DataFrame({'ID\_code':df\_test['ID\_code'].values}) df\_sub['lgbm\_predict\_prob']=lgbm\_predict\_prob df\_sub['lgbm\_predict']=lgbm\_predict df\_sub.to\_csv('submission.csv',index=False)

df\_sub.head()

### R Code:-

**#Loading Libraries:-** 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)

### #Setting Directory:-

setwd("D:/Practice\_R")

### #Importing the training Data:-

df\_train=read.csv("train.csv")

head(df\_train)

#Dimension of the train data:- dim(df\_train)

#Summary of the train dataset:- str(df\_train)

#Typecasting the target variable:- df\_train$target=as.factor(df\_train$target)

#Target class count in train data:- table(df\_train$target)

#Percentage count of taregt class in train data:- table(df\_train$target)/length(df\_train$target)\*100

#Bar plot for count of target classes in train data:- plot1=ggplot(df\_train,aes(target))+theme\_bw()+geom\_bar(stat='count',fill='lightgreen')

#Violin with jitter plots for target classes

plot2=ggplot(df\_train,aes(x=target,y=1:nrow(df\_train)))+theme\_bw()+geom\_violin(fill=' lightblue')+

facet\_grid(df\_train$target)+geom\_jitter(width=0.02)+labs(y='Index') grid.arrange(plot1,plot2, ncol=2)

#Observation:- We are having a unbalanced data, where 90% of the data is no. of customers who will not make a transaction & 10 % of the data are those who will make a transaction.

#Distribution of train attributes from 3 to 102:- for (var in names(df\_train)[c(3:102)]){

target<-df\_train$target

plot<-ggplot(df\_train, aes(df\_train[[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(df\_train)[c(103:202)]){

target<-df\_train$target

plot<-ggplot(df\_train, aes(df\_train[[var]],fill=target)) + geom\_density(kernel='gaussian') + ggtitle(var)+theme\_classic() print(plot)

}

#Importing the test data:- df\_test=read.csv("test.csv") head(df\_test)

#Dimension of test dataset:- dim(df\_test)

#Distribution of test attributes from 2 to 101:-

plot\_density(df\_test[,c(2:101)],ggtheme = theme\_classic(),geom\_density\_args = list(color='red'))

#Distribution of test attributes from 102 to 201:-

plot\_density(df\_test[,c(102:201)],ggtheme = theme\_classic(),geom\_density\_args = list(color='red'))

**#Mean value per rows and columns in train & test dataset:-** #Applying the function to find mean values per row in train and test data. train\_mean<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=mean) test\_mean<-apply(df\_test[,-c(1)],MARGIN=1,FUN=mean)

ggplot()+

#Distribution of mean values per row in train data geom\_density(data=df\_train[,-

c(1,2)],aes(x=train\_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_cla ssic()+

#Distribution of mean values per row in test data

geom\_density(data=df\_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")

#Applying the function to find mean values per column in train and test data. train\_mean<-apply(df\_train[,-c(1,2)],MARGIN=2,FUN=mean)

test\_mean<-apply(df\_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')+th eme\_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 column in train and test dataset")

### Standard Deviation Distribution:-

#Applying the function to find standard deviation values per row in train and test data. train\_sd<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=sd)

test\_sd<-apply(df\_test[,-c(1)],MARGIN=1,FUN=sd) ggplot()+

#Distribution of sd values per row in train data

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

#Distribution of sd values per row in test data

geom\_density(data=df\_test[,- 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(df\_train[,-c(1,2)],MARGIN=2,FUN=sd)

test\_sd<-apply(df\_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

\_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")

### Skewness Distribution:-

#Applying the function to find skewness values per row in train and test data. train\_skew<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=skewness) test\_skew<-apply(df\_test[,-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')+t heme\_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(df\_train[,-c(1,2)],MARGIN=2,FUN=skewness)

test\_skew<-apply(df\_test[,-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')+t heme\_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")

### Kurtosis Distribution:-

#Applying the function to find kurtosis values per row in train and test data. train\_kurtosis<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=kurtosis) test\_kurtosis<-apply(df\_test[,-c(1)],MARGIN=1,FUN=kurtosis)

ggplot()+

#Distribution of kurtosis values per row in train data

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

#Distribution of kurtosis values per row 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(df\_train[,-c(1,2)],MARGIN=2,FUN=kurtosis) test\_kurtosis<-apply(df\_test[,-c(1)],MARGIN=2,FUN=kurtosis)

ggplot()+

#Distribution of kurtosis values per column in train data

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

#Distribution of kurtosis 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")

### #Missing Value Analysis:-

#Finding the missing values in train data

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

missing\_val

#Correlations in train data:- #convert factor to int

df\_train$target<-as.numeric(df\_train$target) train\_correlation<-cor(df\_train[,c(2:202)]) train\_correlation

#Observation:- We can observe that correlation between train attributes is very small.

#Correlations in test data test\_correlation<-cor(df\_test[,c(2:201)]) test\_correlation

#Observation:- We can observe that correlation between test attributes is very small.

**#Feature Enginnering**:- Performing some feature engineering on datasets:-

#Variable Importance:-Variable importance is used to see top features in dataset based on mean decreases gini .

#Building a simple model to find features which are imp:-

#Split the training data using simple random sampling train\_index<-sample(1:nrow(df\_train),0.75\*nrow(df\_train)) #train data

train\_data<-df\_train[train\_index,] #validation data

valid\_data<-df\_train[-train\_index,] #dimension of train and validation data dim(train\_data)

dim(valid\_data)

#Random forest classifier:-

#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)

#Feature importance by random forest- #Variable importance

VarImp<-importance(rf,type=2)

VarImp

#Observation:-We can observed that the top important features are var\_12, var\_26, var\_22,v var\_174, var\_198 and so on based on Mean decrease gini.

#Partial dependence plots:-PDP gives a graphical depiction of marginal effect of a variable on the class probability or classification. It shows how a feature effects predictions.

#Calculation of partial dependence plots on random forest:-

#we are observing impact of main features which are discovered in previous section by using PDP Plot.

#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\_6"

par.var\_6 <- partial(rf, pred.var = c("var\_6"), chull = TRUE) plot.var\_6 <- autoplot(par.var\_6, contour = TRUE) plot.var\_6

**#Handling of imbalanced data**- Now we are going to explore 5 different approaches for dealing with imbalanced datasets.

#Change the performance metric #Oversample minority class #Undersample majority class #ROSE

#LightGBM

#Logistic Regression Model:-

#Split the data using simple random sampling:- set.seed(689)

train.index<-sample(1:nrow(df\_train),0.8\*nrow(df\_train)) #train data

train.data<-df\_train[train.index,] #validation data

valid.data<-df\_train[-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 and validation dataset

#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(df\_test[,-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

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter

#Minimum lambda cv\_lr$lambda.min

#plot the auc score vs log(lambda) plot(cv\_lr)

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

#Model performance on validation dataset set.seed(5363)

cv\_predict.lr<-predict(cv\_lr,X\_v,s = "lambda.min", type = "class") cv\_predict.lr

#Observation:-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.

#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)

#Reciever operating characteristics(ROC)-Area under curve(AUC) score and curve:- #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)

### #Oversample Minority Class:-

#-Adding more copies of minority class.

#-It cab be a good option we dont have that much large data to work.

#-Drawback of this process is we are adding info. That can lead to overfitting or poor performance on test data.

### #Undersample Mojorityclass:-

#-Removing some copies of majority class.

#-It can be a good option if we have very large amount of data say in millions to work.

#-Drawback of this process is we are removing some valuable info. that can leads to underfitting & poor performance on test data.

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

**#Random Oversampling Examples(ROSE)-** It creates a sample of synthetic data by enlarging the features space of minority and majority class examples.

#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

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter:-

#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)

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

#Training and validation dataset

#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(df\_test[,-c(1)])

#training data

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

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

#Choosing best hyperparameters

#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)

#Training the lgbm model

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)

#Let us plot the important features 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)

#We tried model with logistic regression,ROSE and lightgbm. But,lightgbm is performing well on imbalanced data compared to other models based on scores of roc\_auc\_score.

#Final submission sub\_df<-

data.frame(ID\_code=df\_test$ID\_code,lgb\_predict\_prob=lgbm\_pred\_prob,lgb\_predict=lg bm\_pred)

write.csv(sub\_df,'submission-R.CSV',row.names=F) head(sub\_df)

# References

* + <https://stackoverflow.com/>
  + <https://medium.com/>
  + <https://www.youtube.com/user/krishnaik06>
  + <https://www.rdocumentation.org/>
  + <https://www.analyticsvidhya.com/blog>
  + <https://pydata.org/>