Santander Customer Transaction Prediction

Submitted by

B Raghu Vamsi

Chapter 1

Introduction

1.1 Problem Statement

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

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

1.2 Data

In this project, our task is to build classification models which will be used to predict which customers will make a specific transaction in the future. Given below is a sample of the Santander customer transaction dataset:

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

ID_code	**target**	var_0	var_1	var_2	………	**……….**	**var_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

From the table below, we have the following 16 variables, using which we have to predict the bike rental count:

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

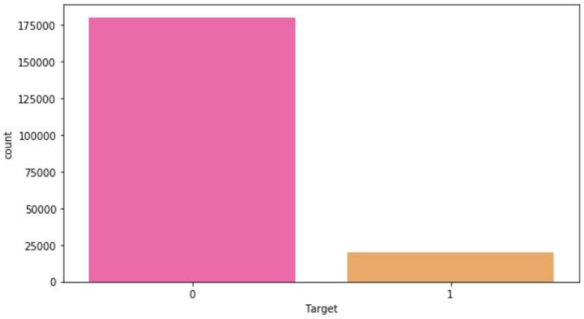
Chapter 2

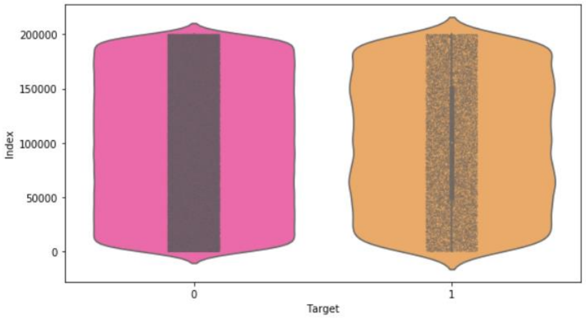
Methodology

2.1 Exploratory Data Analysis (EDA)

Exploratory data analysis is one of the most important steps in data mining 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 must clean the data otherwise it will affect on performance of the model. Now we are going to explain one by one as follows. In this EDA I explained with seaborn visualizations.

2.2.1 Target classes count





Observation:

- We are having a unbalanced data, where 90% of the data is no. of customers w ho will not make a transaction & 10 % of the data are those who will make a tra nsaction.

- From the violin plots, it seems that there is no relationship between the target a nd index of the data frame, it is more dominated by zero compare to one's.

- From the jitter plots with violin plots, we can observe that target looks uniform ly distributed over the indexes of the data frame.

2.2.2 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.

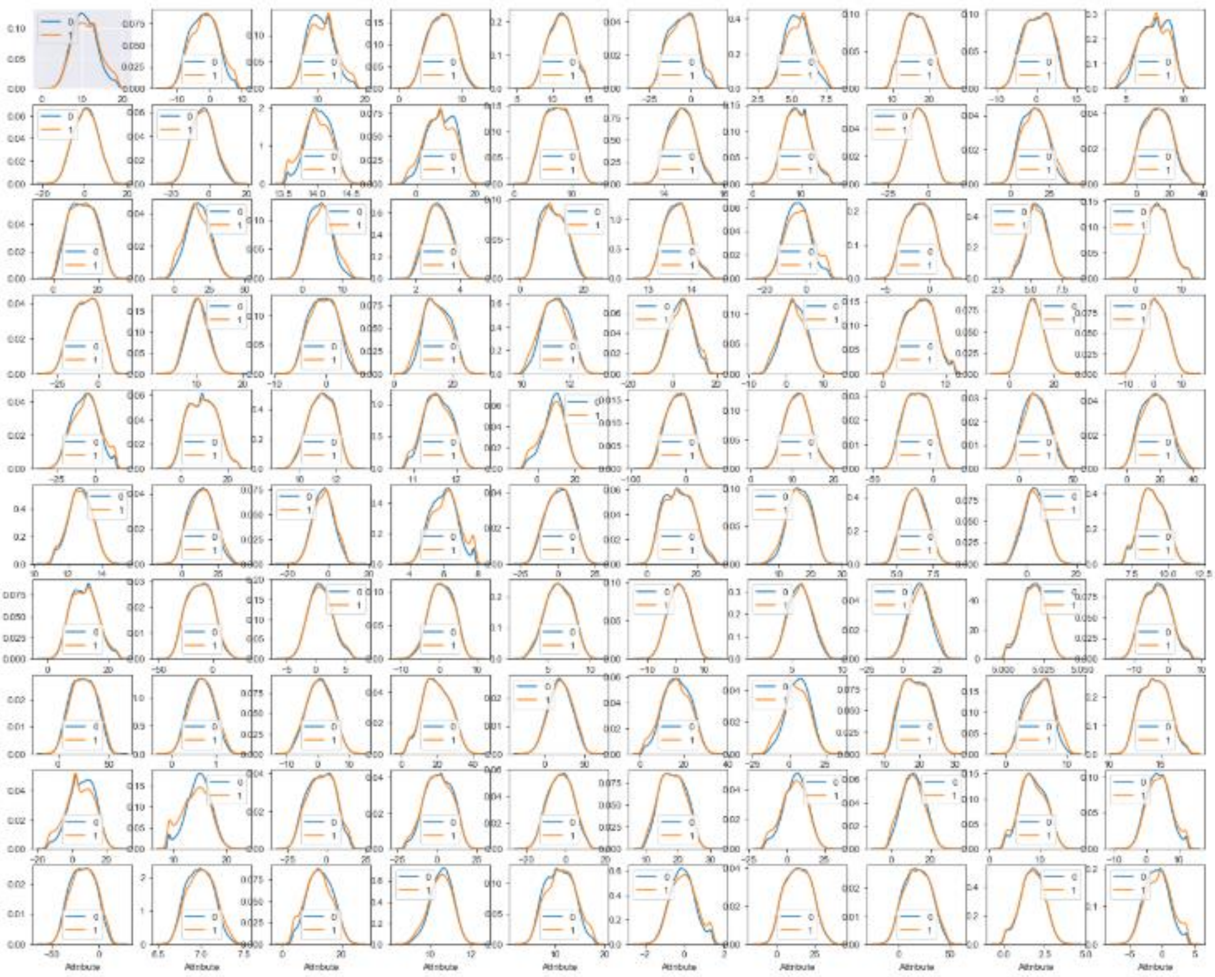
R and Python code as follows: -

```
#R Code:-

#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

#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)
```

2.1.1 Attributes distributions and trends

Distribution of train attributes

Let us look distribution of train attributes from var_0 to var_99



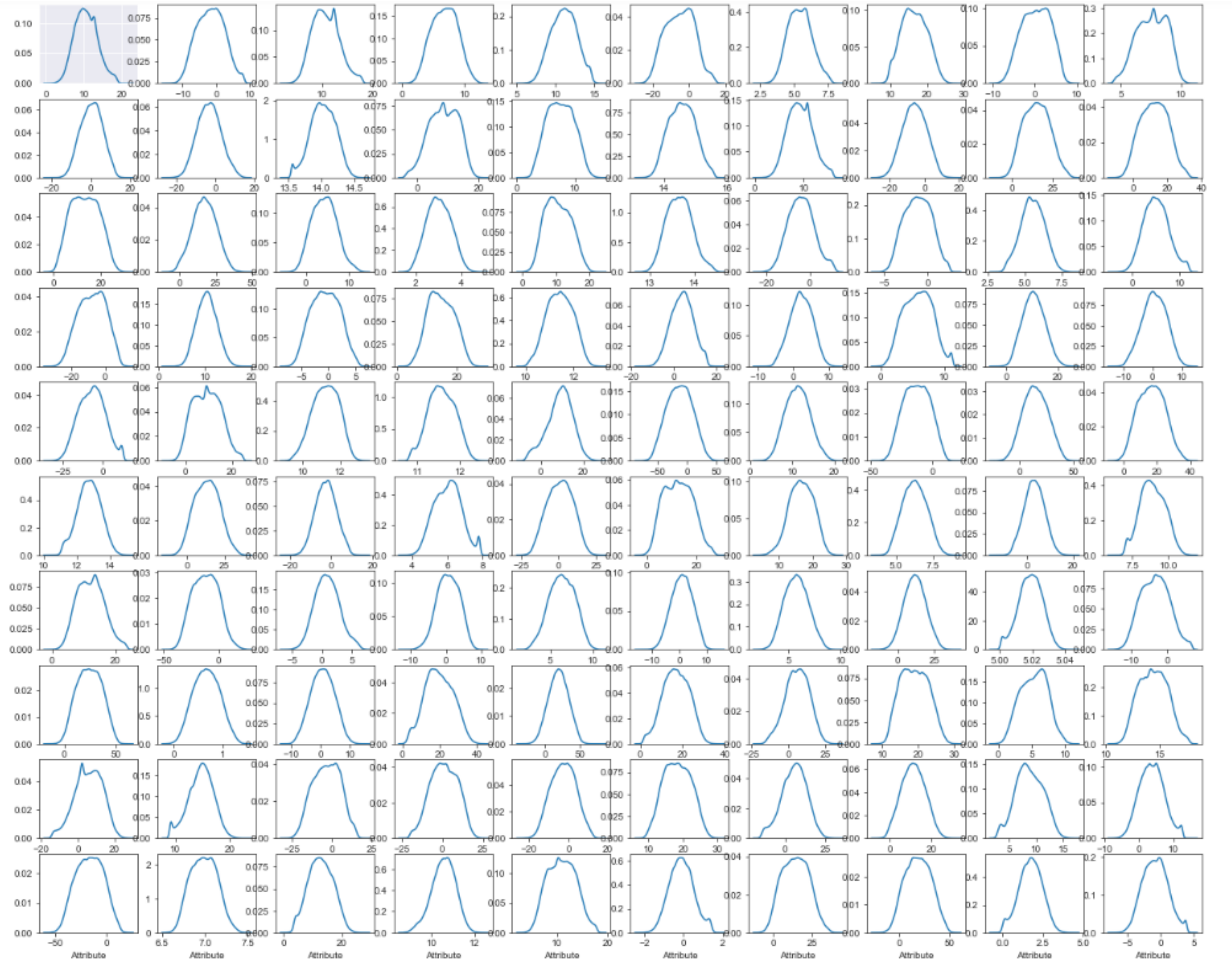
Observation:

-We can observed 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 observed 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.

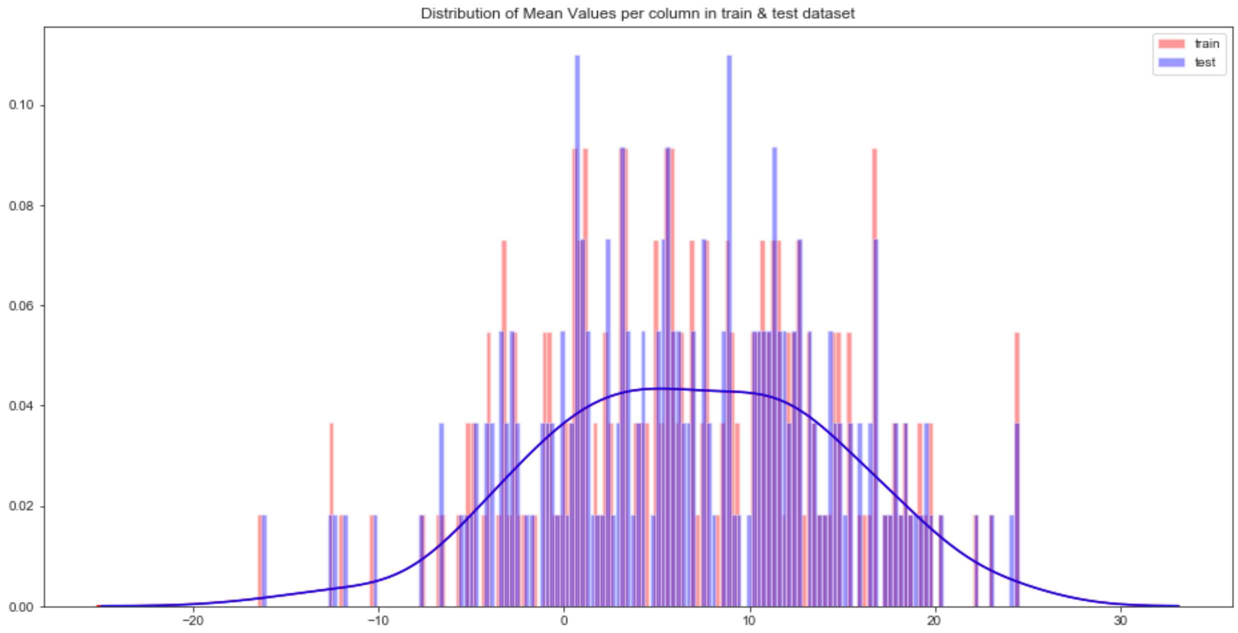
Distribution of test attributes

Let us look distribution of test attributes from var_0 to var_99

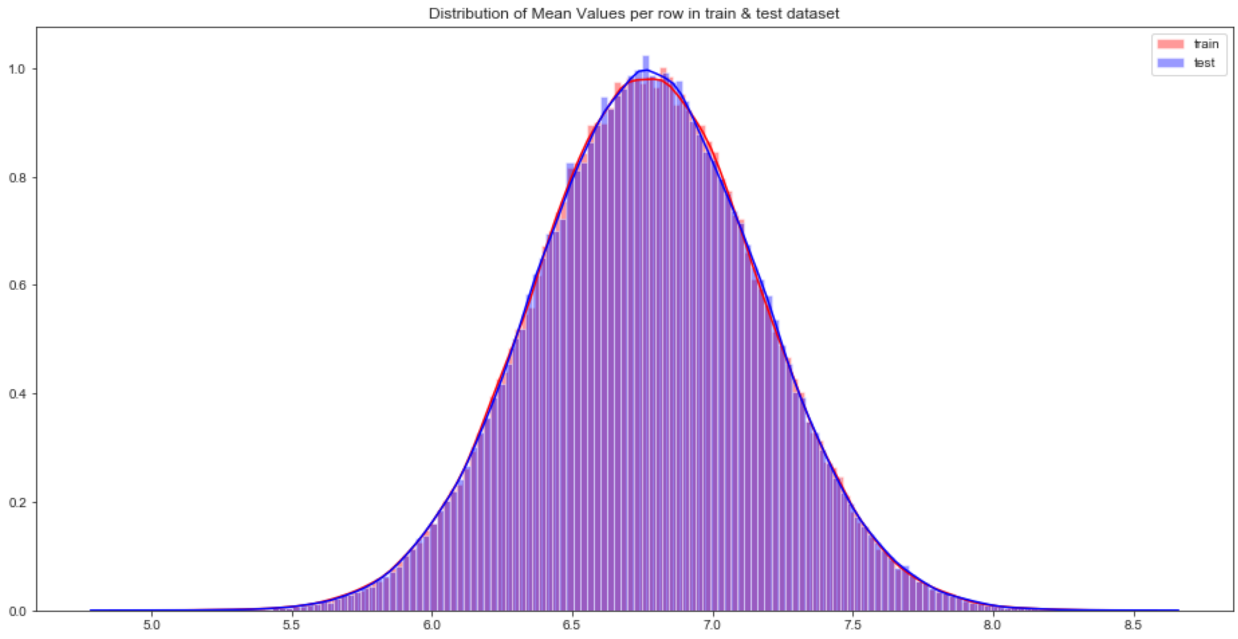


Distribution of mean values in both train and test dataset:-

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

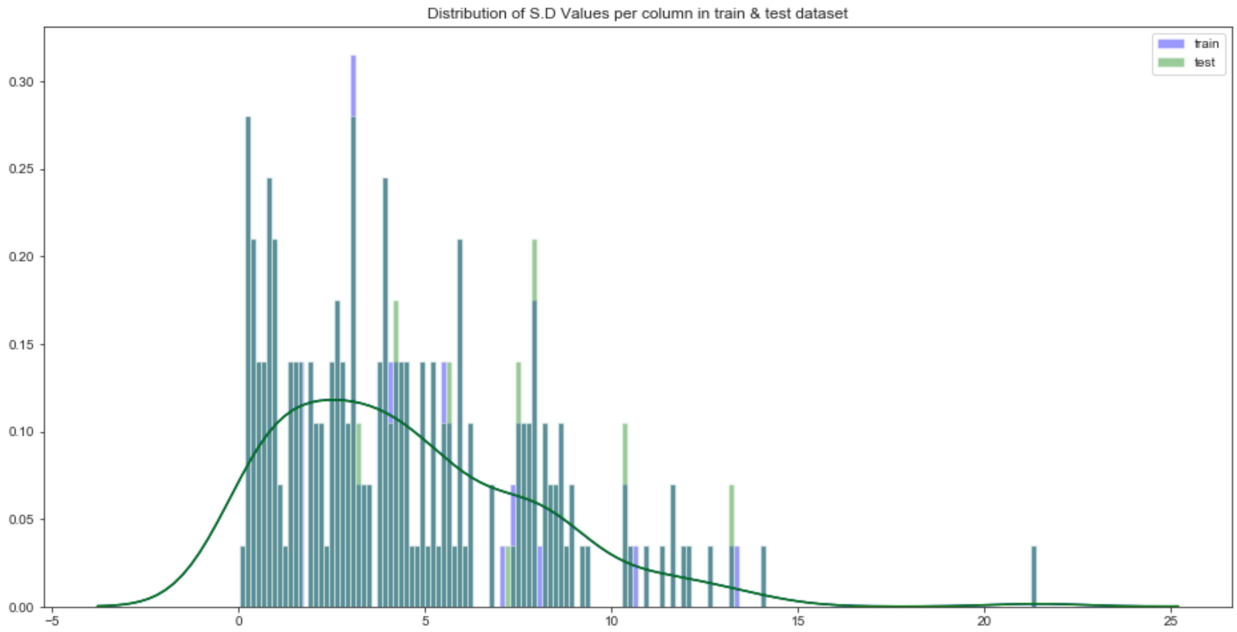


Let us look distribution of mean values per row in train and test dataset:-

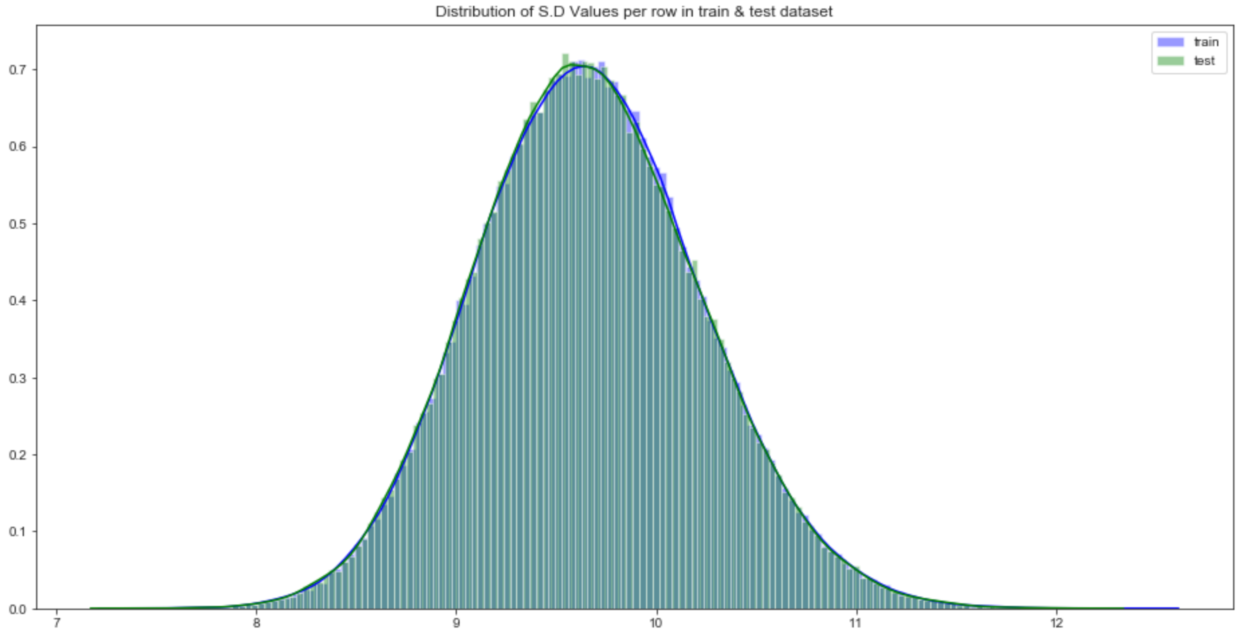


Distribution of standard deviation (std) values in train and test dataset

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

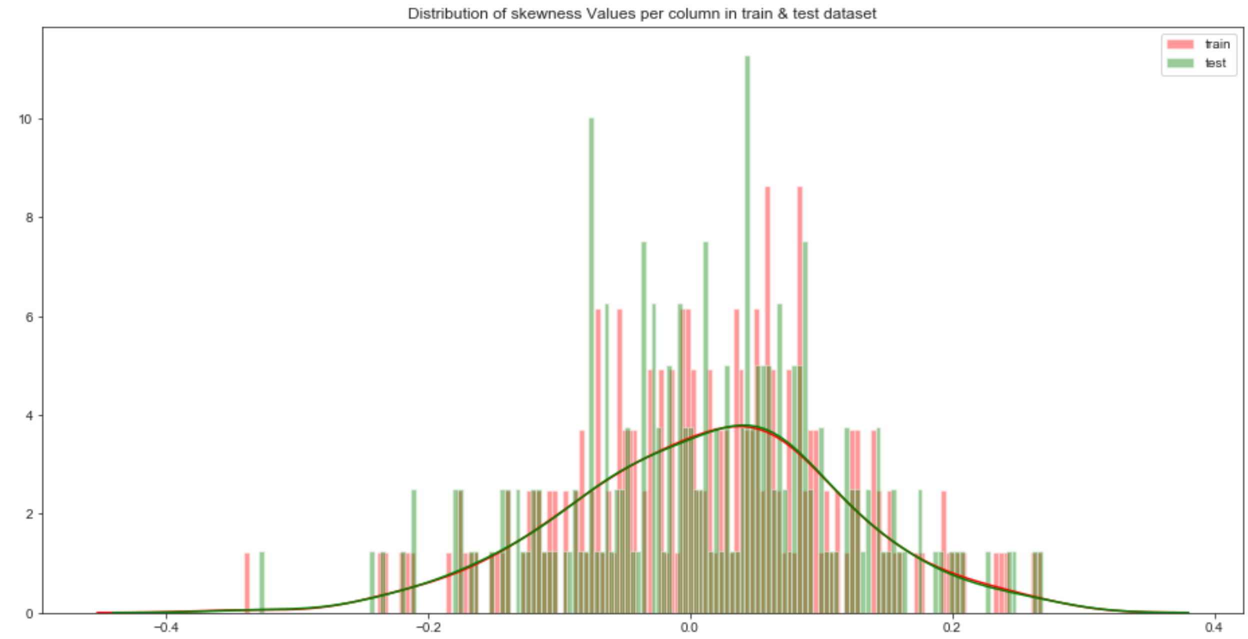


Let us look distribution of standard deviation (std) values per column in train and test dataset

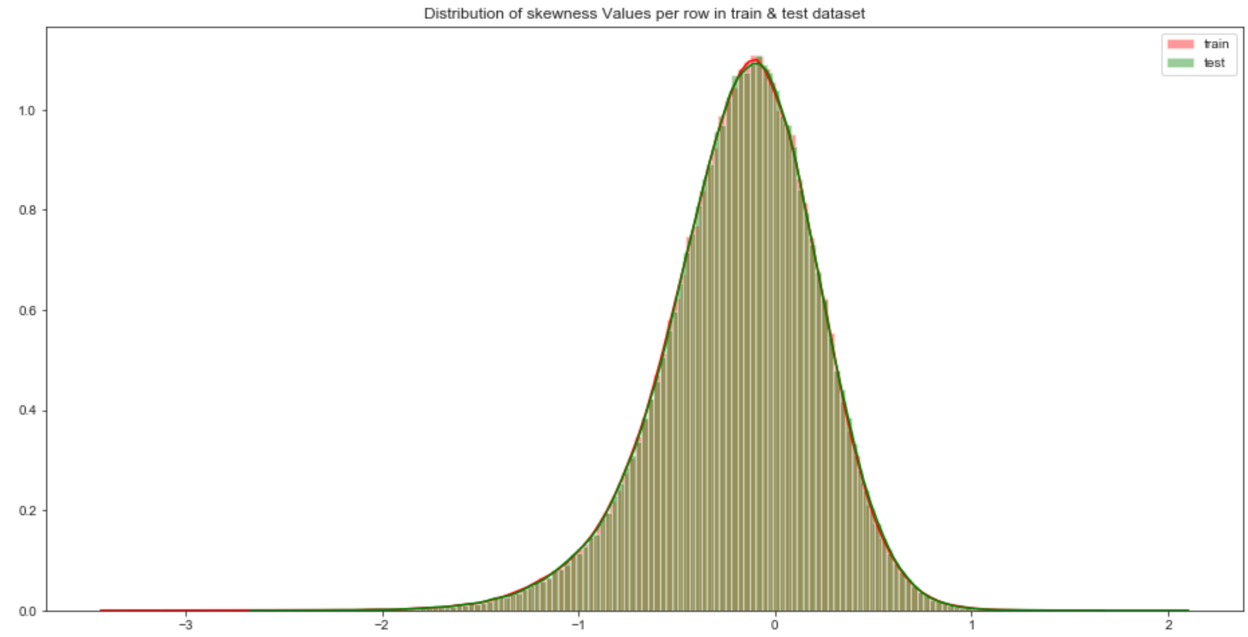


Distribution of skewness values in train and test dataset

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

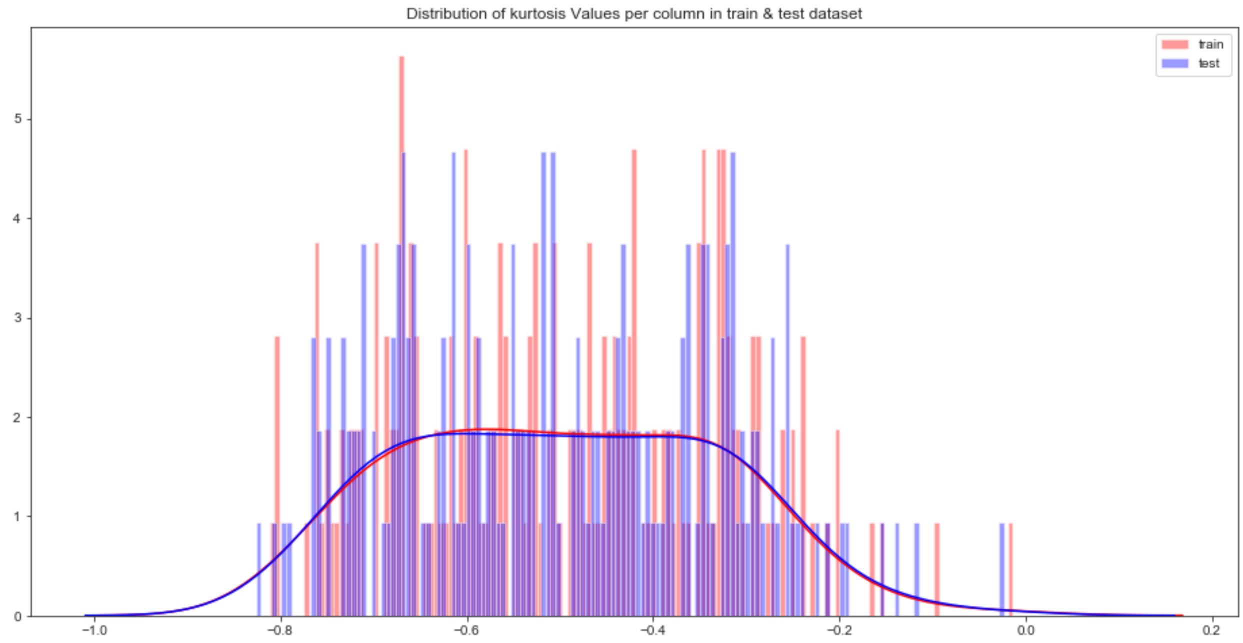


Let us look distribution of skewness values per column in train and test dataset:-

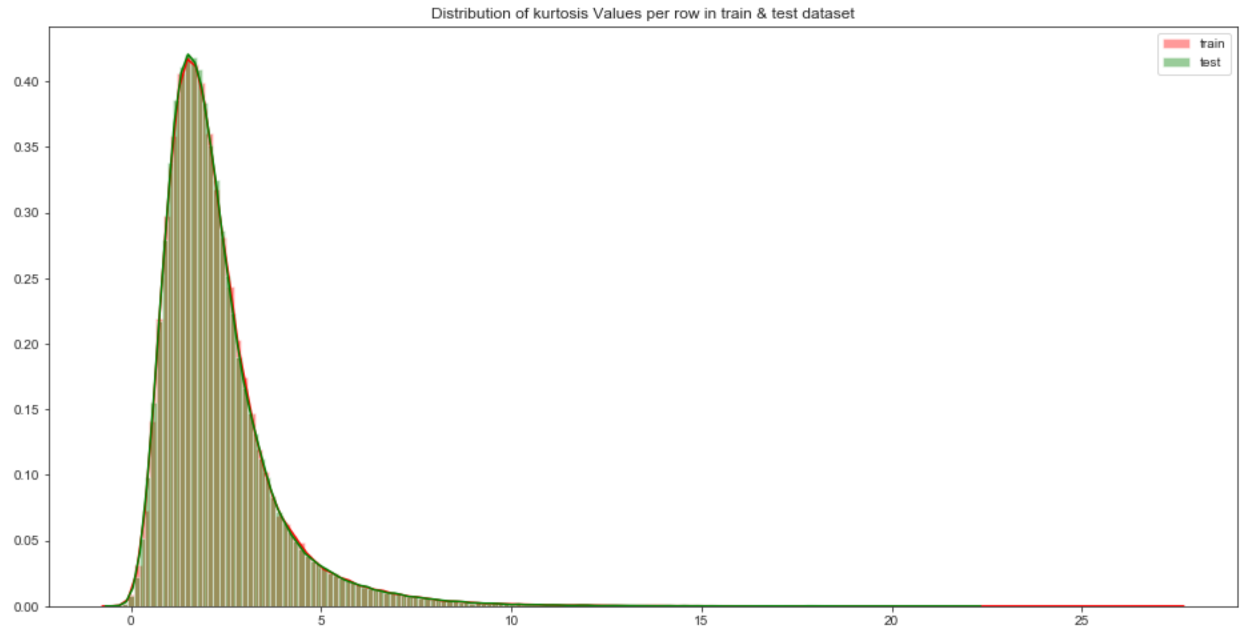


Distribution of kurtosis values in train and test dataset

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:-



2.1.2 Outlier analysis

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

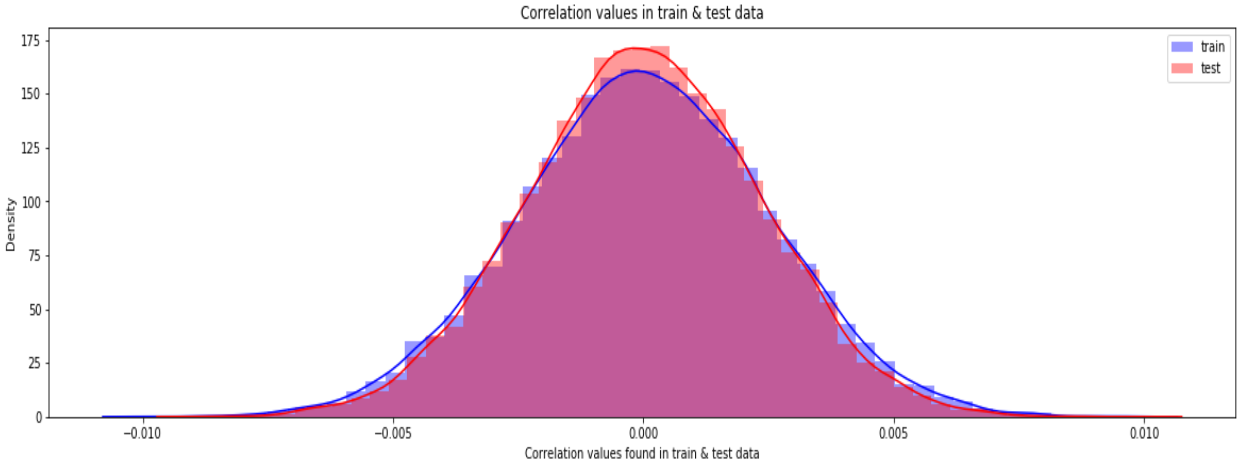
2.1.3 Feature Selection

Feature selection is very important for modelling the dataset. The every dataset have good and unwanted features. The unwanted features would effect 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, it tells about linear relationship between attributes and help us to build better models.

From correlation distribution plot, we can observed 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.

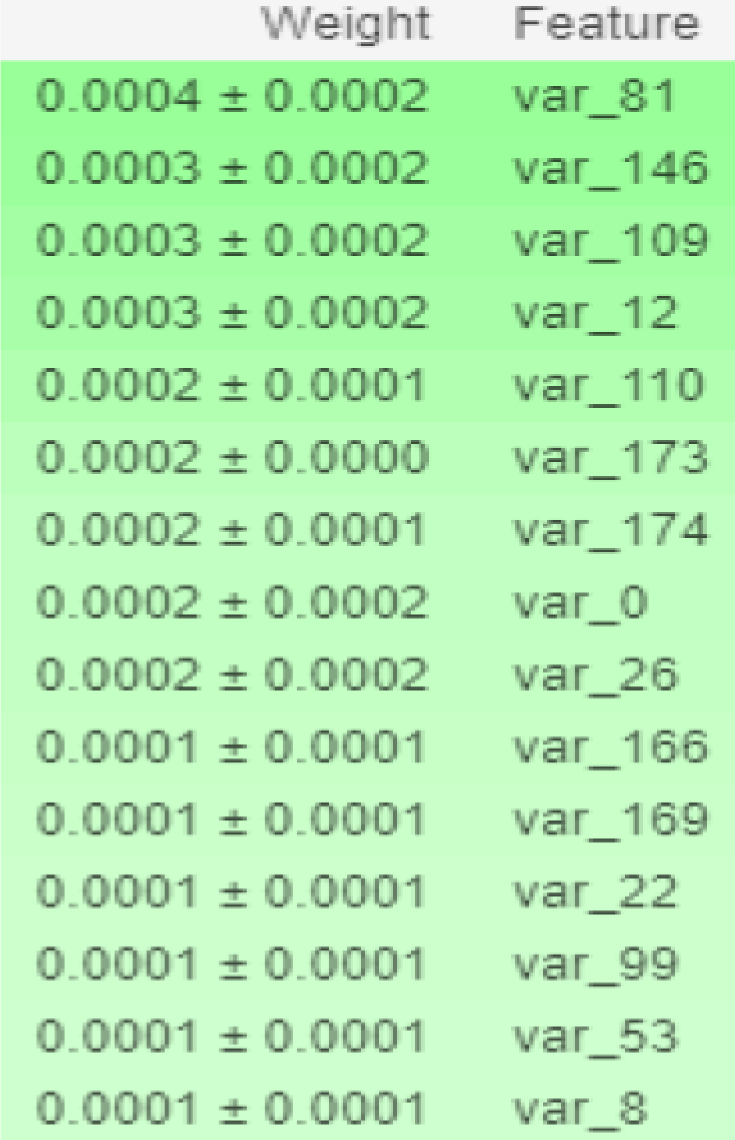
Python code

```
#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)
```



Weight	Feature
0.0004 ± 0.0002	var_81
0.0003 ± 0.0002	var_146
0.0003 ± 0.0002	var_109
0.0003 ± 0.0002	var_12
0.0002 ± 0.0001	var_110
0.0002 ± 0.0000	var_173
0.0002 ± 0.0001	var_174
0.0002 ± 0.0002	var_0
0.0002 ± 0.0002	var_26
0.0001 ± 0.0001	var_166
0.0001 ± 0.0001	var_169
0.0001 ± 0.0001	var_22
0.0001 ± 0.0001	var_99
0.0001 ± 0.0001	var_53
0.0001 ± 0.0001	var_8

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
```

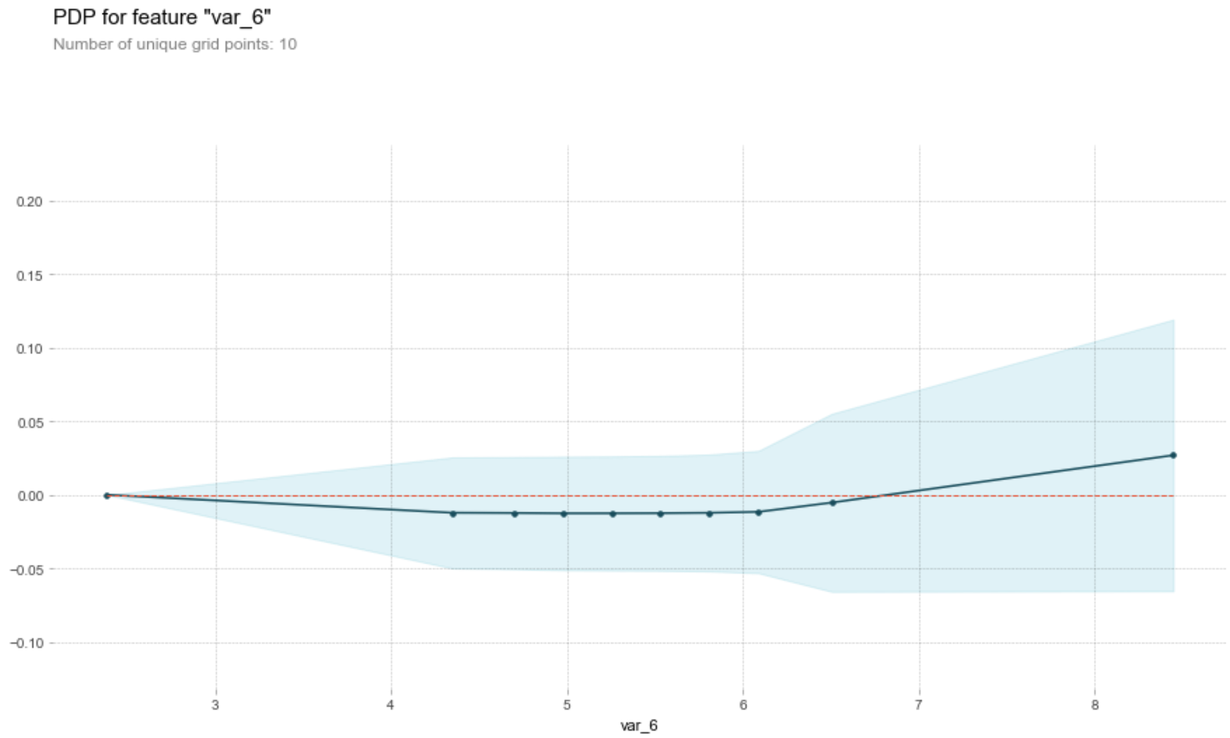
Observation: - We can observed that the top important features are var_12, var_26, var_22,v var_174, var_198 and so on based on Mean decrease gini.

Partial dependence plots

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

```
#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-

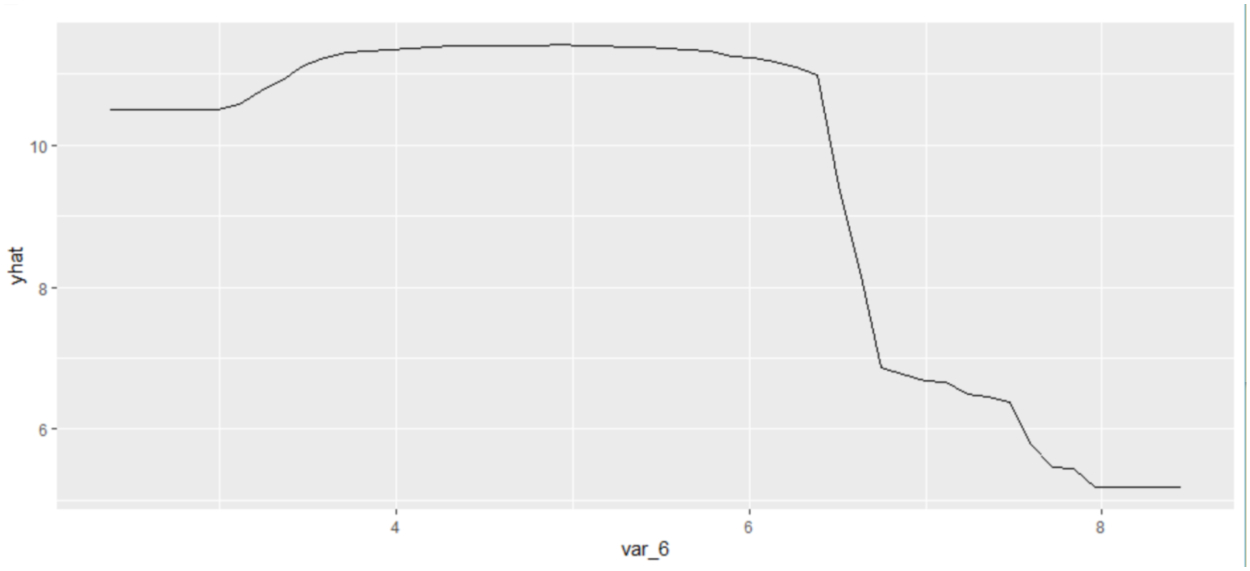
- The y axis dosen't show the predictor value instead how the value changing with the change in given predictor variable.

- The blue shaded area indicates level of confidence of var_6.

- On y-axis having a +ve value means for that particular value of predictor vriable its is less likely to predict the correct class & having & having a +ve value means it has +ve impact 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
```



2.1 Modeling

2.1.1 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.

2.1.2 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.

Oversample Minority Class:-

-Adding more copies of minority class.

-It can 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 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 more best 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.

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

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
```

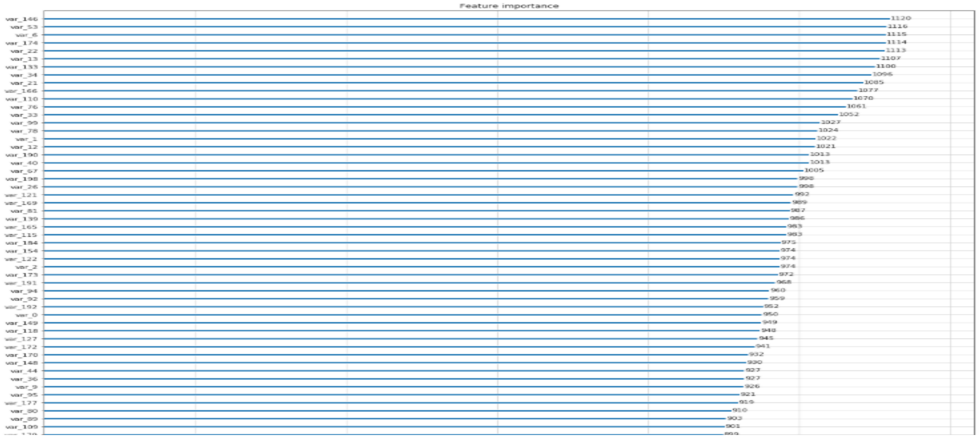
R Code:-

```
#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

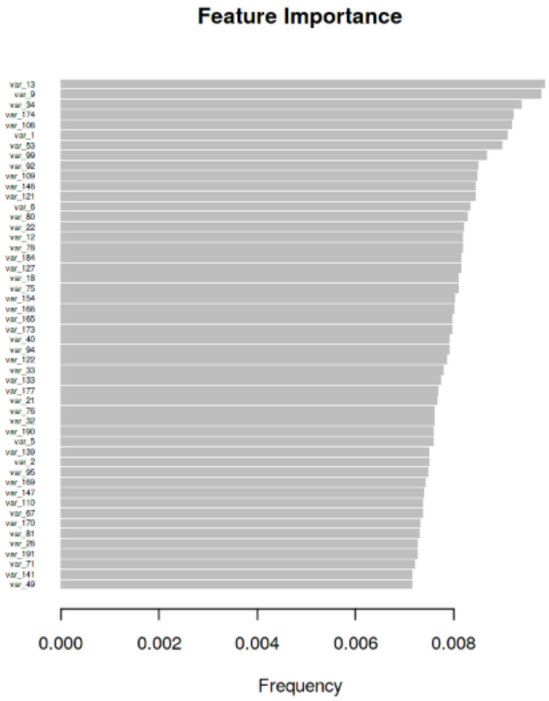
Python code

```
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

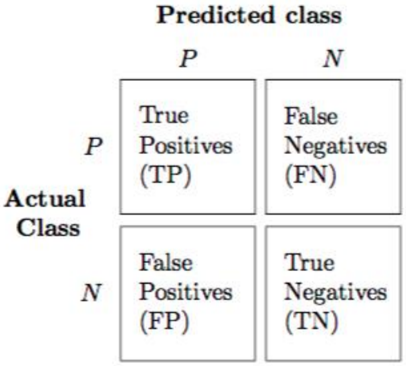
3.1 Model Evaluation

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.



		Predicted class	
		P	*N*
Actual Class	*P*	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

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 = $2*Precision*RecallPrecision+Recall$

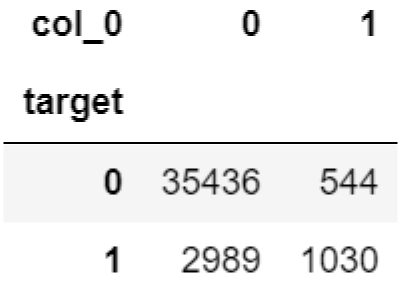
Receiver operating characteristics (ROC)_Area under curve(AUC) Score

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:-

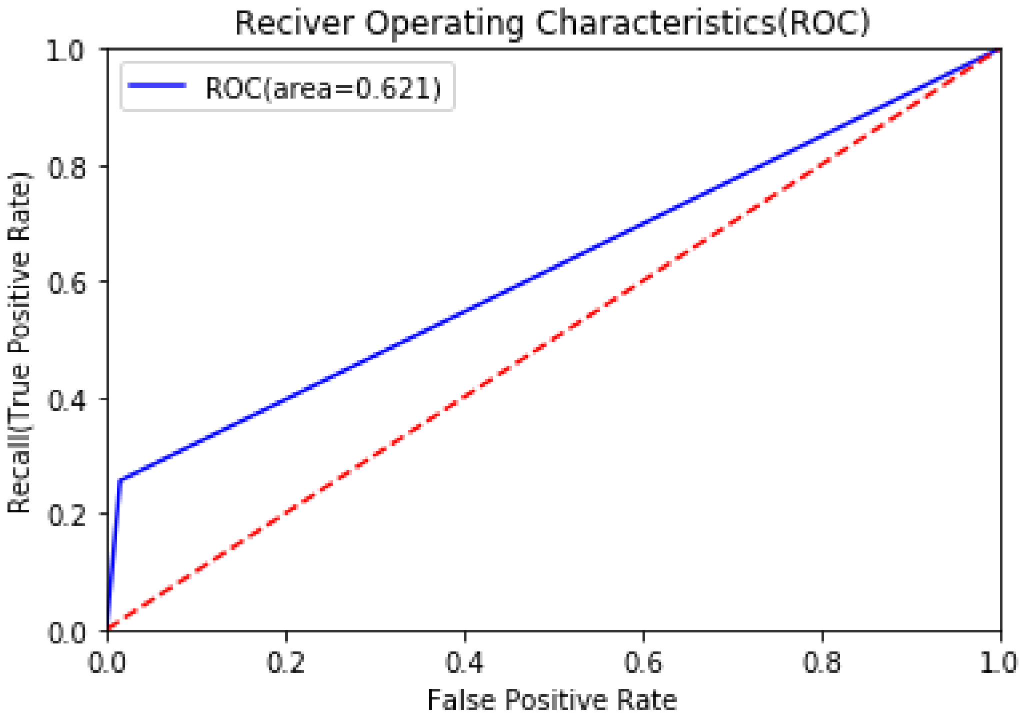
```
cm=confusion_matrix(y_valid,cv_predict)

cm=pd.crosstab(y_valid,cv_predict)

cm
```



```
col_0       0     1
target
     0  35436   544
     1   2989  1030
```

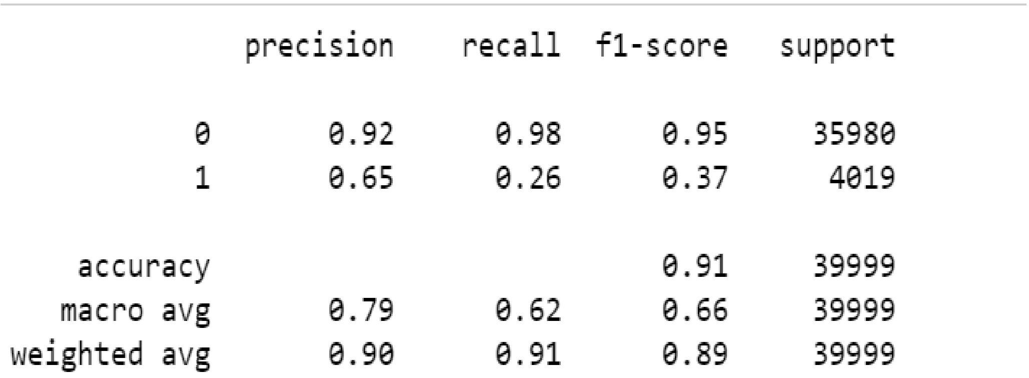
#ROC_AUC_Curve:-

```
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



```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     35980
           1       0.65      0.26      0.37      4019

    accuracy                           0.91     39999
   macro avg       0.79      0.62      0.66     39999
weighted avg       0.90      0.91      0.89     39999
```

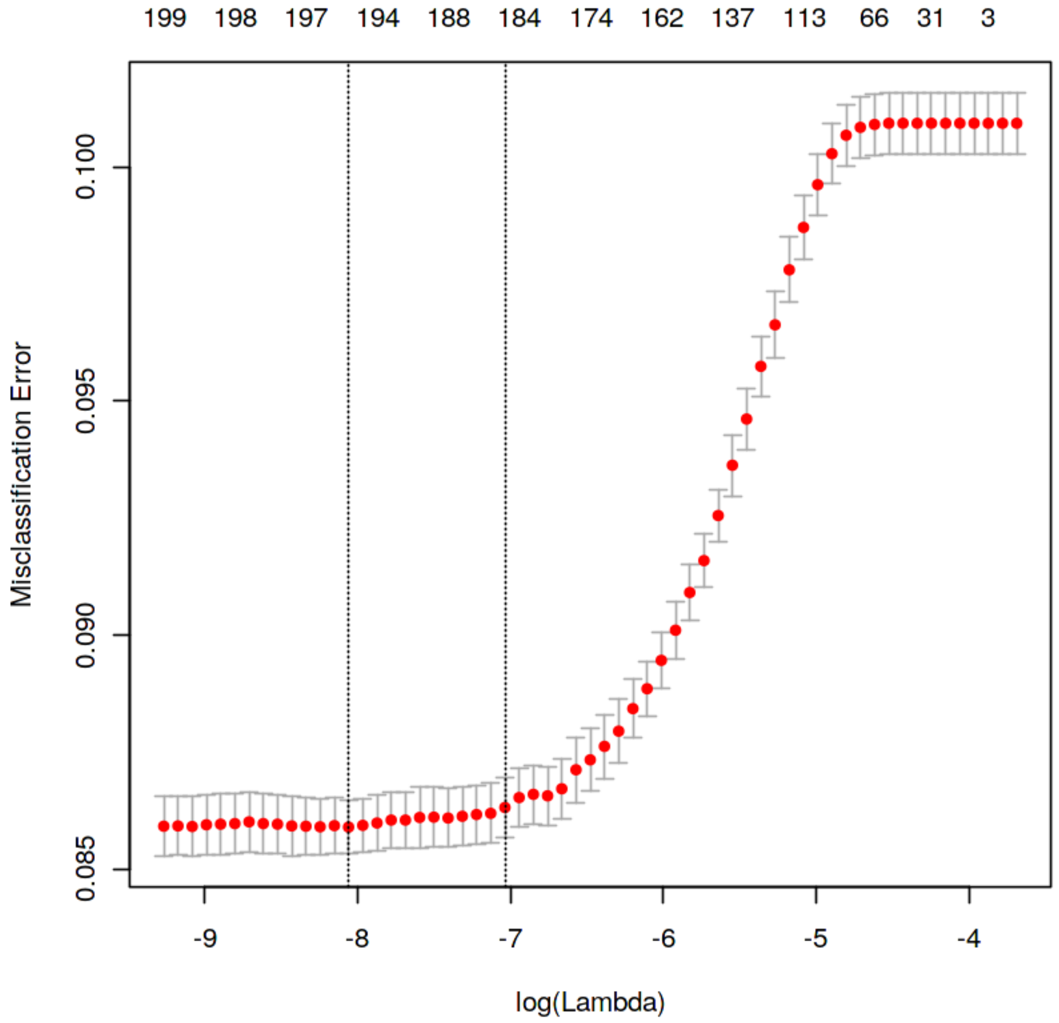
We can observed that f1 score is high for number of customers those who will not make a transaction then who will make a transaction. So, we are going to change the algorithm.

R code

Logistic Regression

#Cross validation prediction

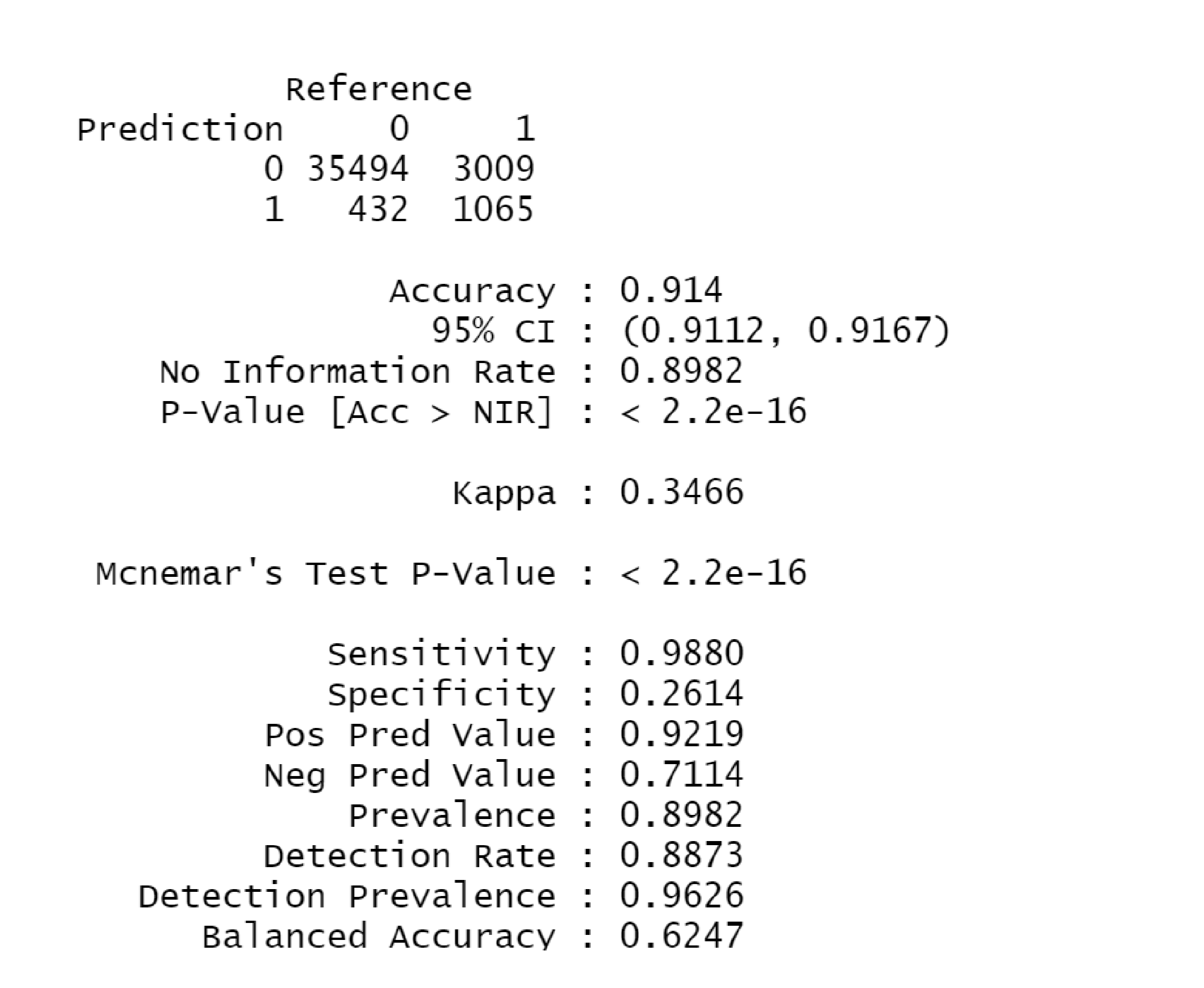
```
set.seed(8909)

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)
```

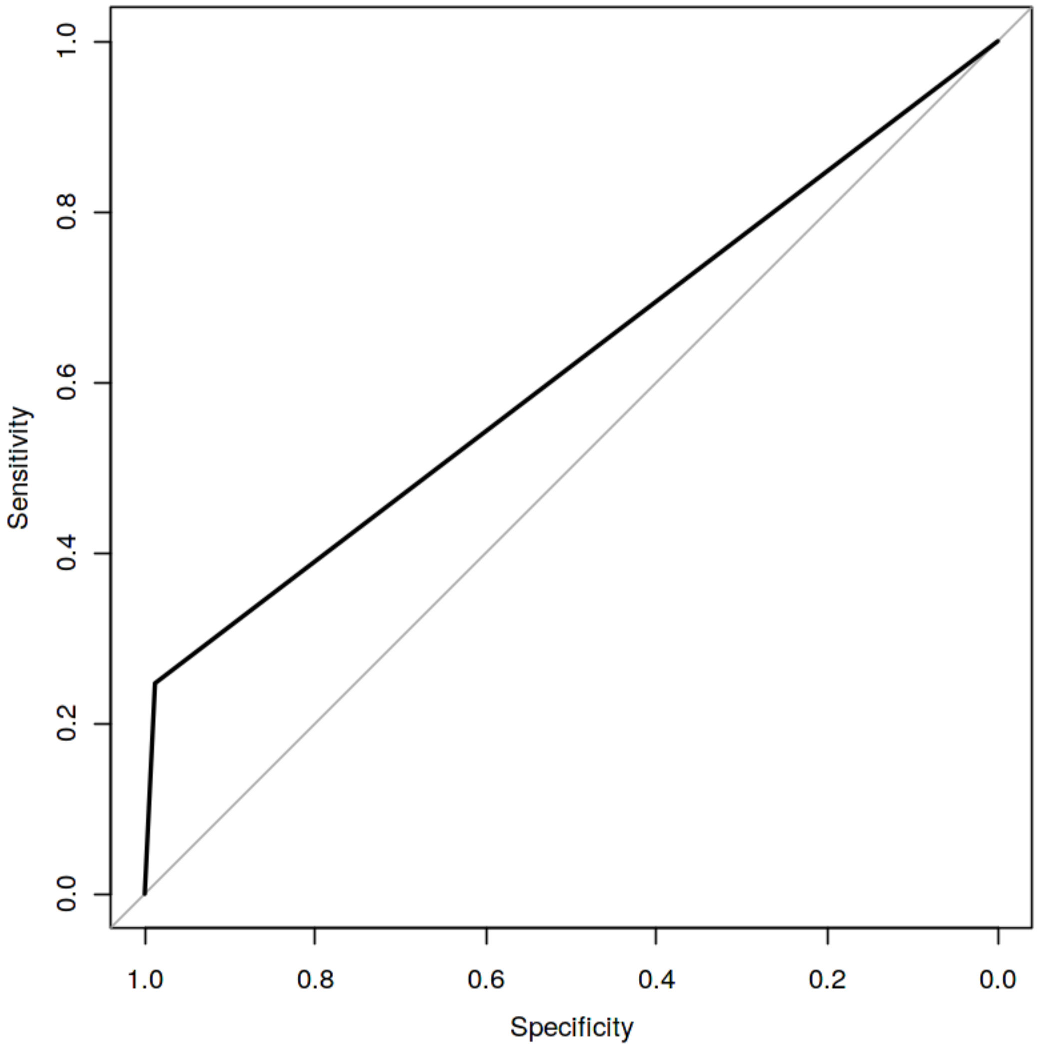
```
#predicted target variable

#convert to factor

cv_predict.lr<-as.factor(cv_predict.lr)

confusionMatrix(data=cv_predict.lr,reference=target)
```



```
          Reference
Prediction     0     1
         0 35494  3009
         1   432  1065

               Accuracy : 0.914
                 95% CI : (0.9112, 0.9167)
    No Information Rate : 0.8982
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3466

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

            Sensitivity : 0.9880
            Specificity : 0.2614
         Pos Pred Value : 0.9219
         Neg Pred Value : 0.7114
             Prevalence : 0.8982
         Detection Rate : 0.8873
   Detection Prevalence : 0.9626
      Balanced Accuracy : 0.6247
```

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

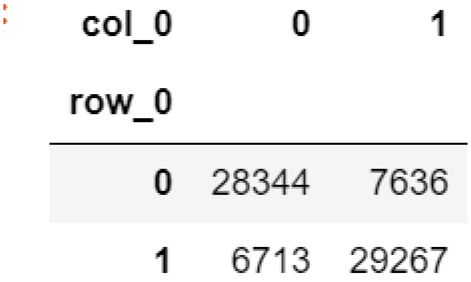
```
#ROC_AUC score and curveset.seed(892)

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)
```

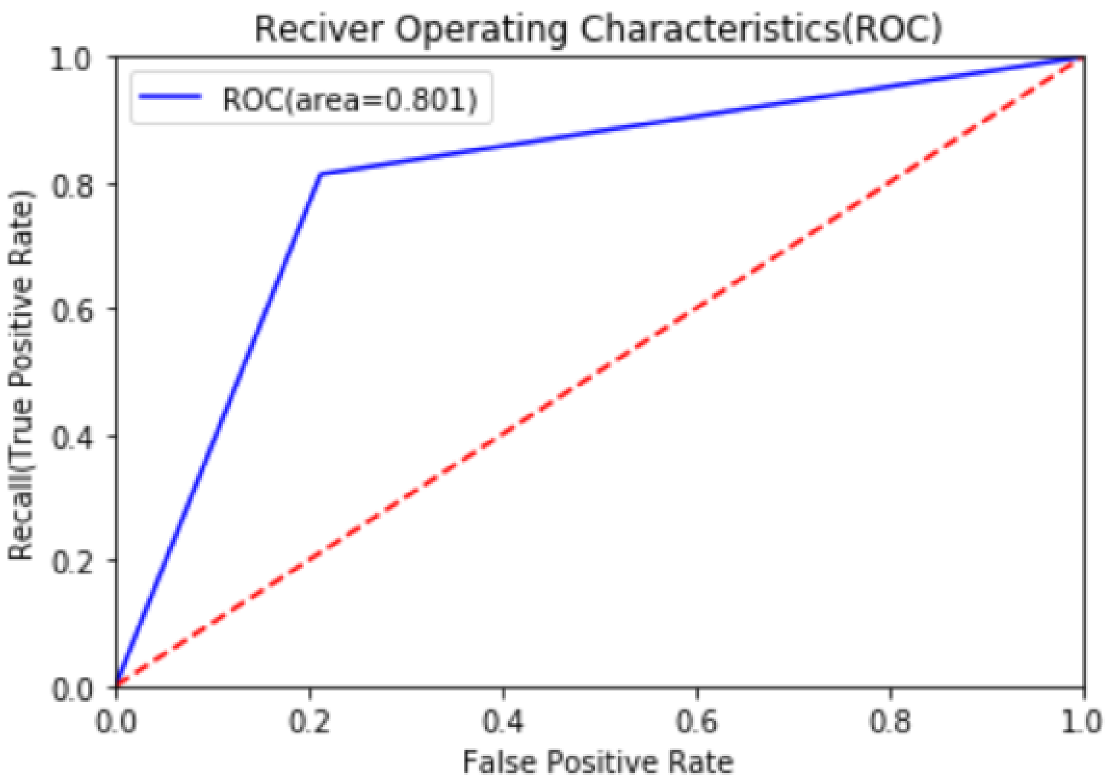


```
col_0      0      1
row_0
    0  28344   7636
    1   6713  29267
```

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

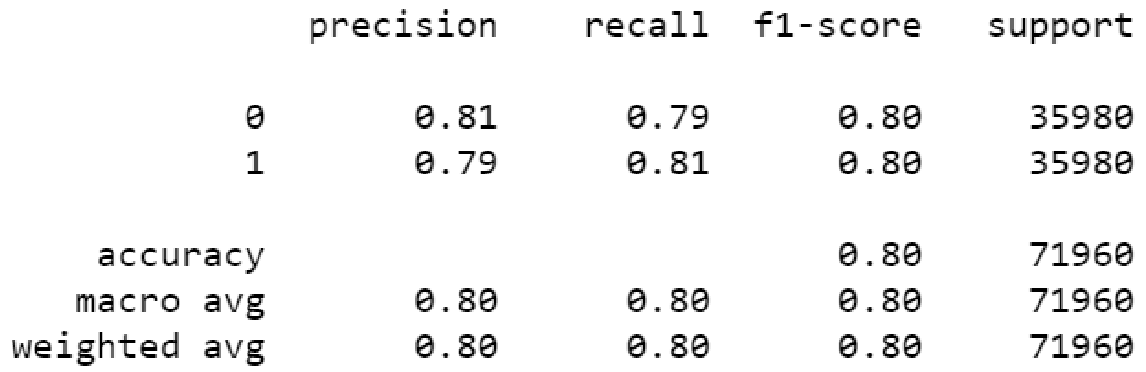
```
#ROC_AUC Curve:-

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)
```



```
              precision    recall  f1-score   support

           0       0.81      0.79      0.80     35980
           1       0.79      0.81      0.80     35980

    accuracy                           0.80     71960
   macro avg       0.80      0.80      0.80     71960
weighted avg       0.80      0.80      0.80     71960
```

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

R code
Random Oversampling Examples (ROSE)

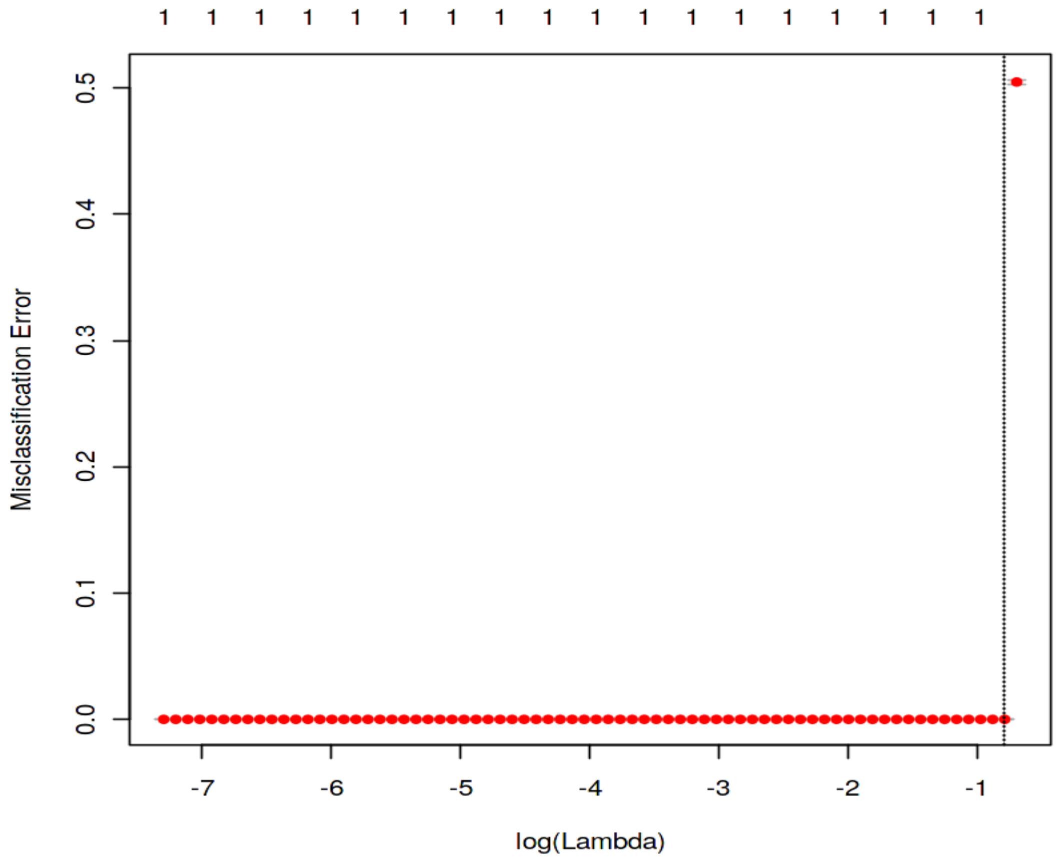
#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter:-

#Minimum lambda

cv_rose$lambda.min

#plot the auc score vs log(lambda)

plot(cv_rose)

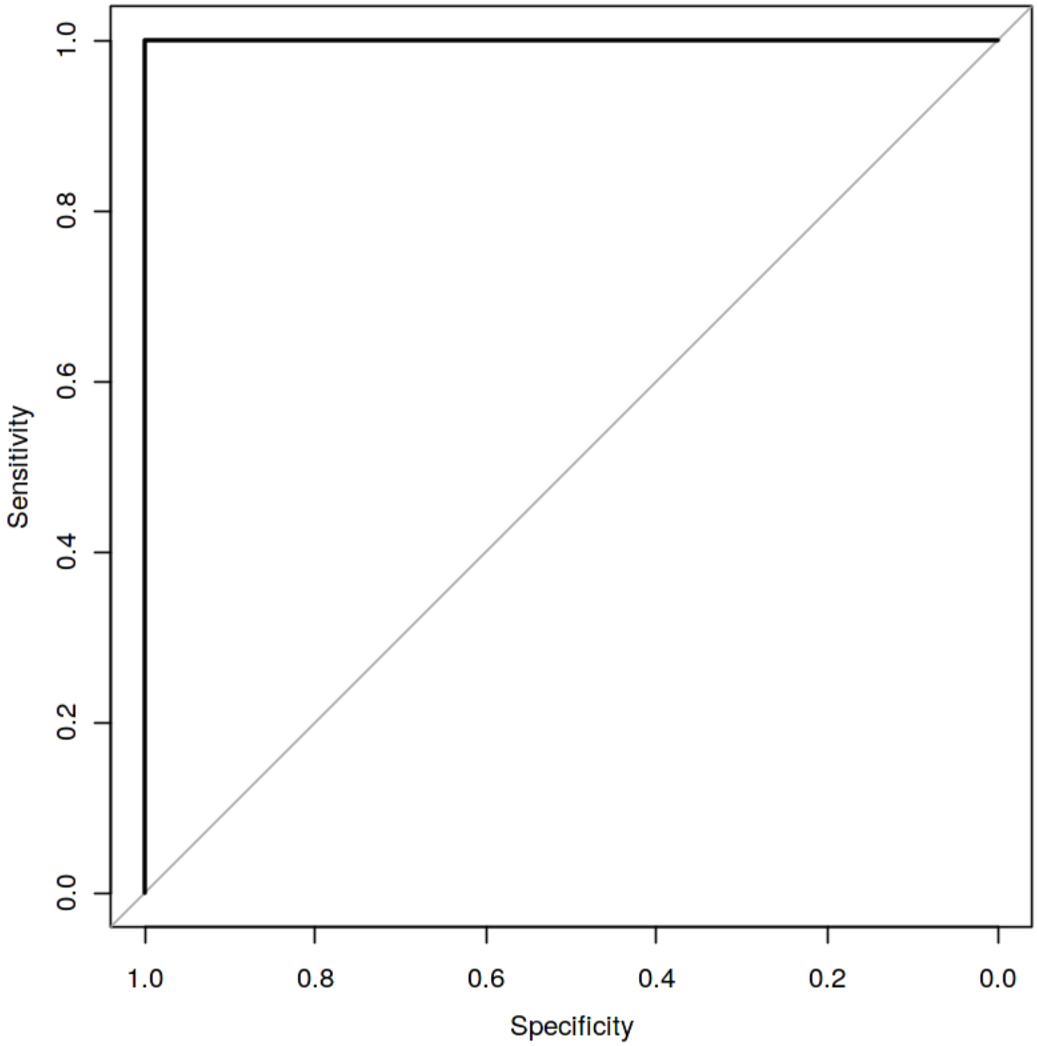


#Confusion matrix

```
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.

3.2 Model Selection

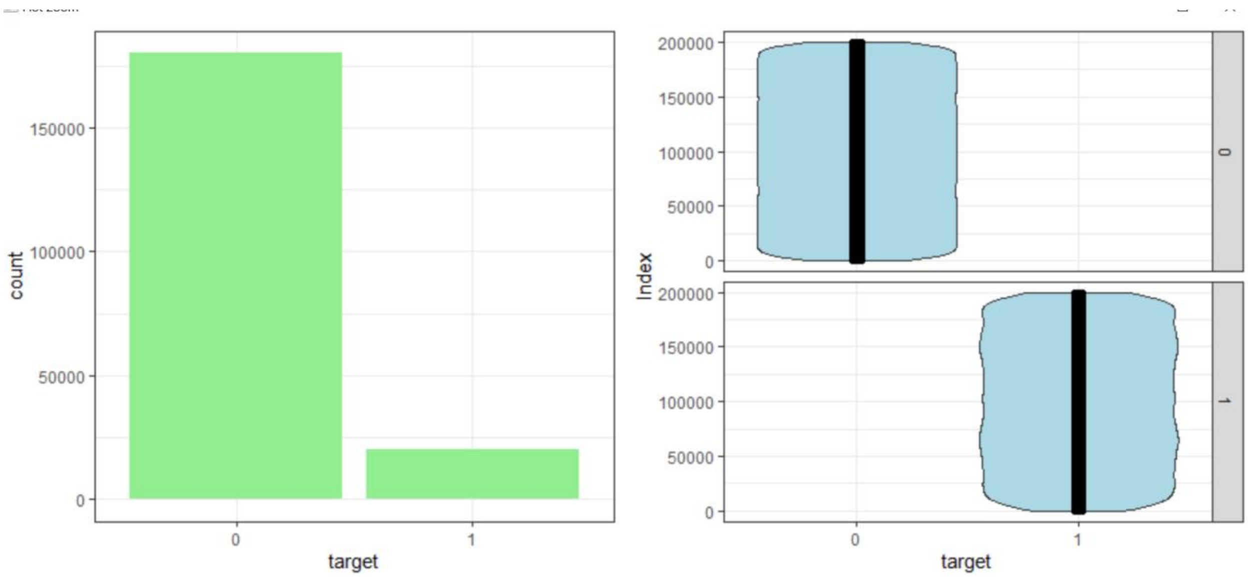
When we compare scores of area under the ROC curve of all the models for an imbalanced data. We could conclude that below points as follow,
1. Logistic regression model is not performed well on imbalanced data.
2. We balance the imbalanced data using resampling techniques like SMOTE in python and ROSE 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.

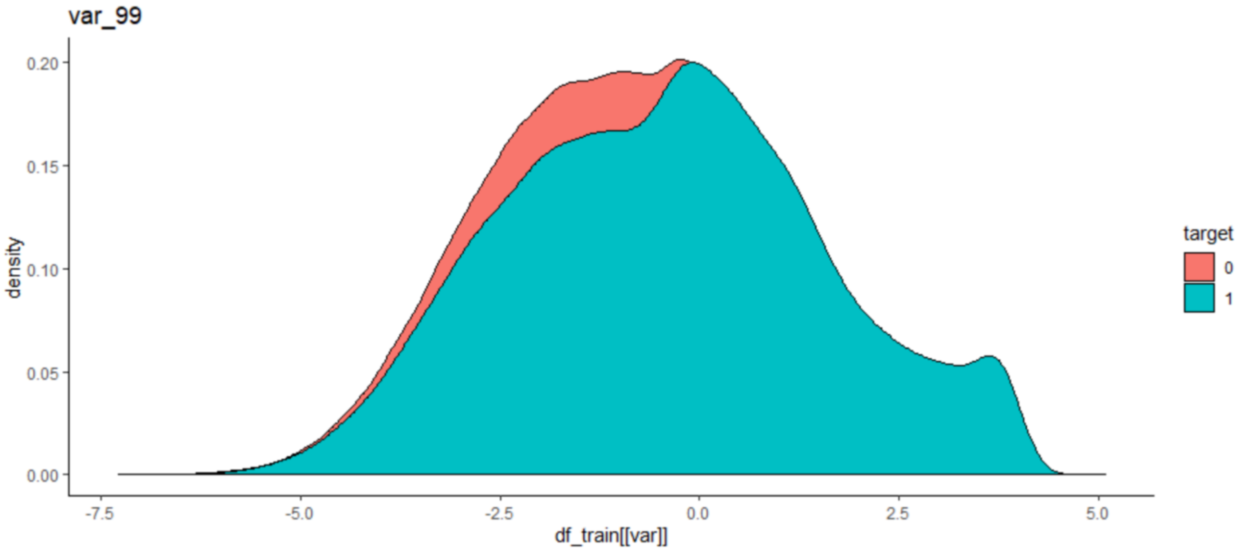
Appendix A - Extra Figures

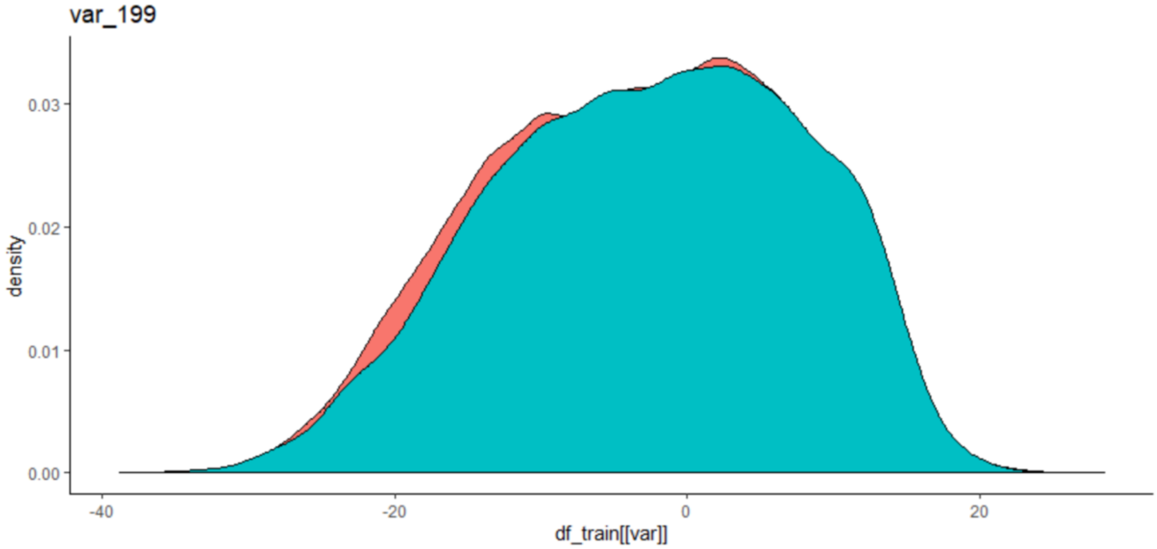
ggplot2 visualizations

Target classes count

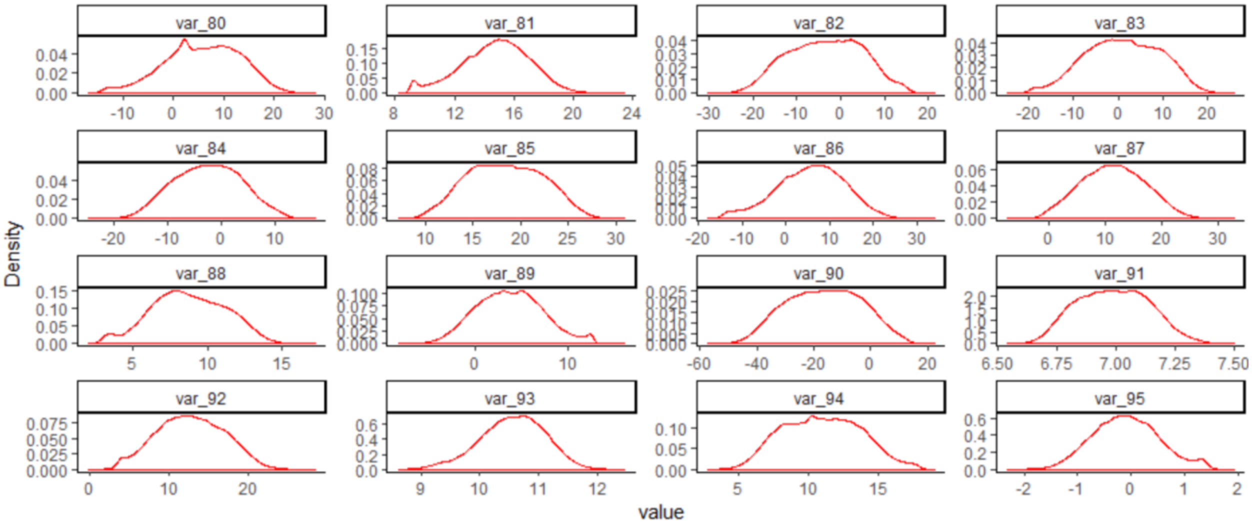


Distribution of train attributes

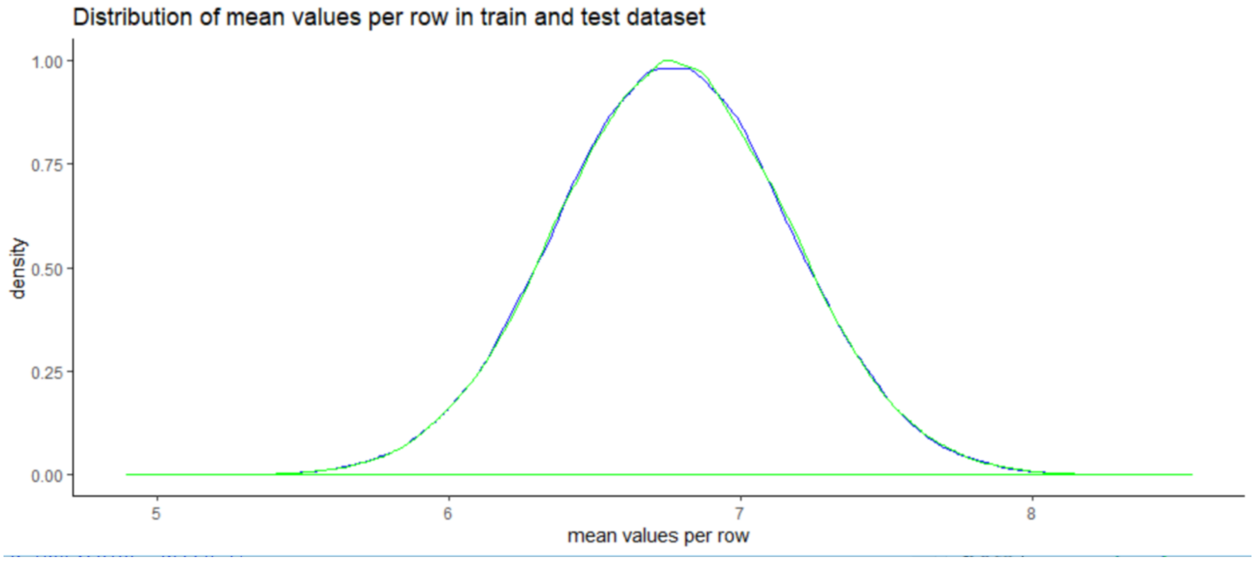




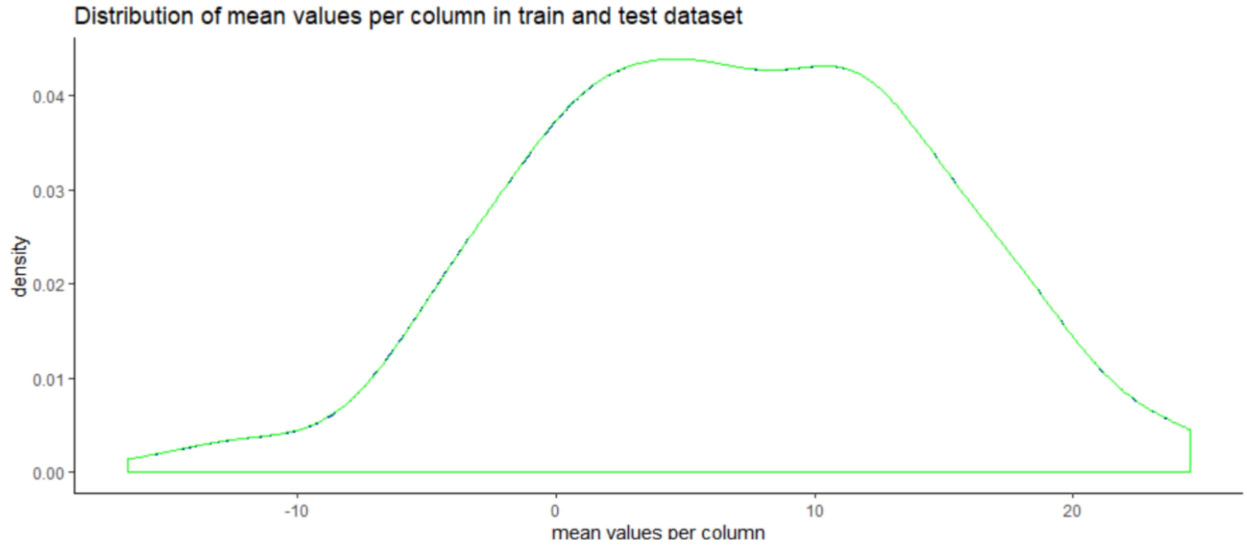
Distribution of test attributes



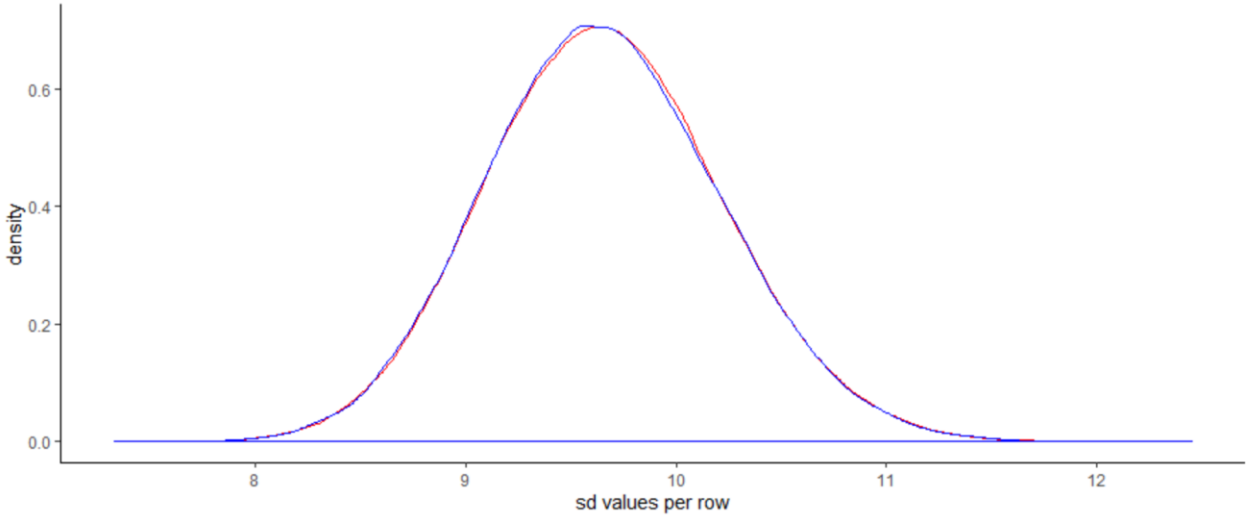
Distribution of mean values per row in train & test data



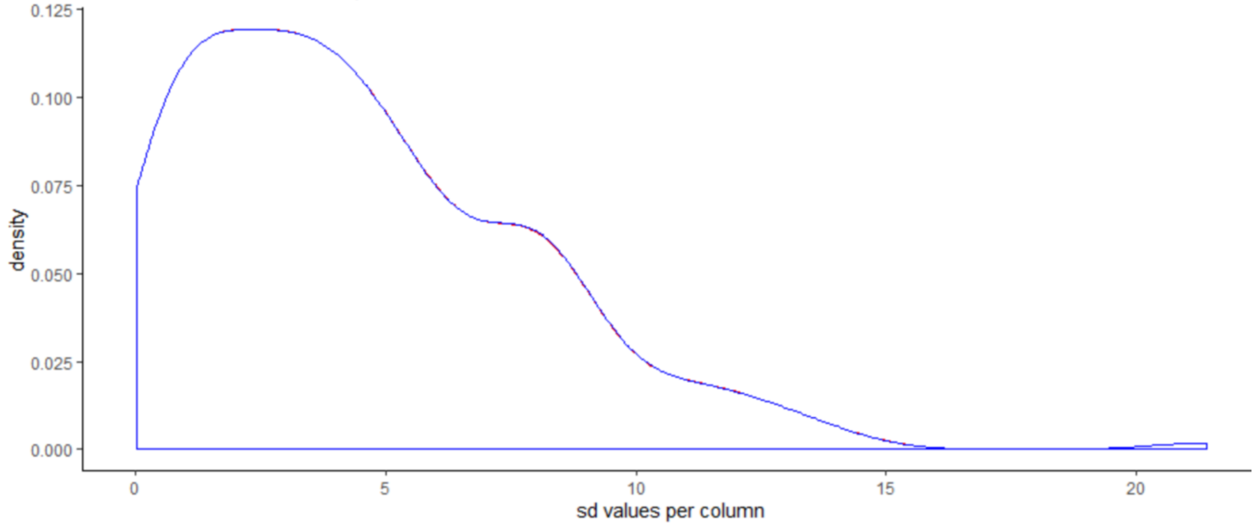
Distribution of mean values per columns in train & test data



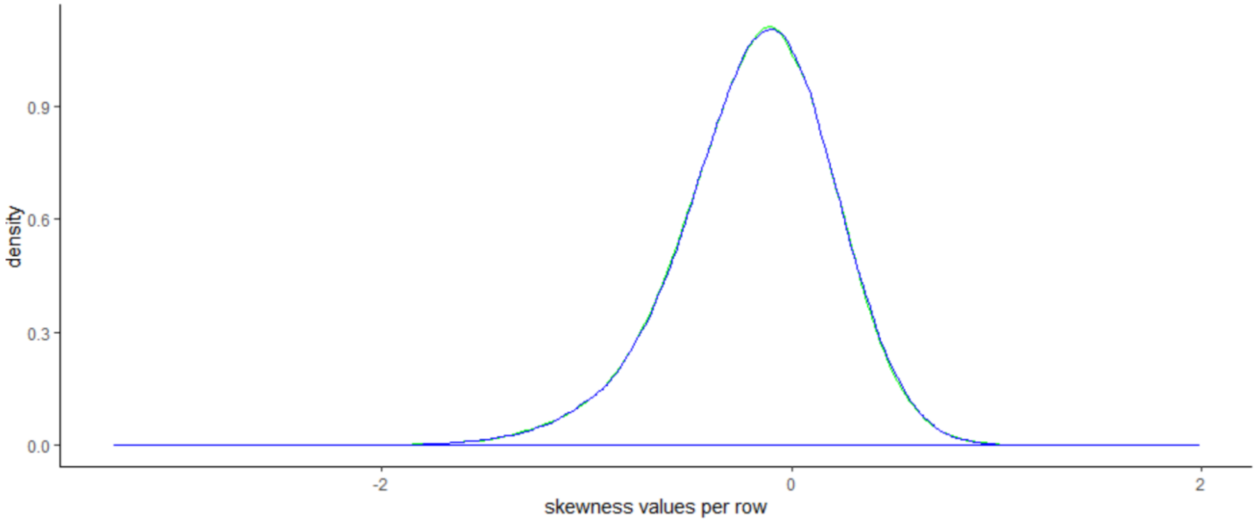
Distribution of sd values per row in train and test dataset

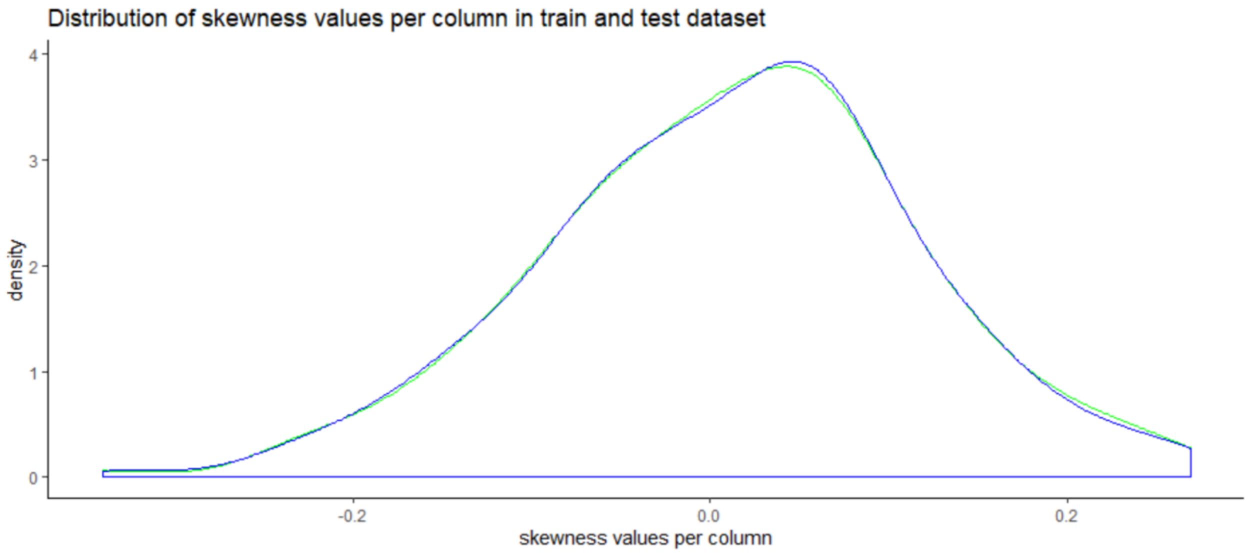


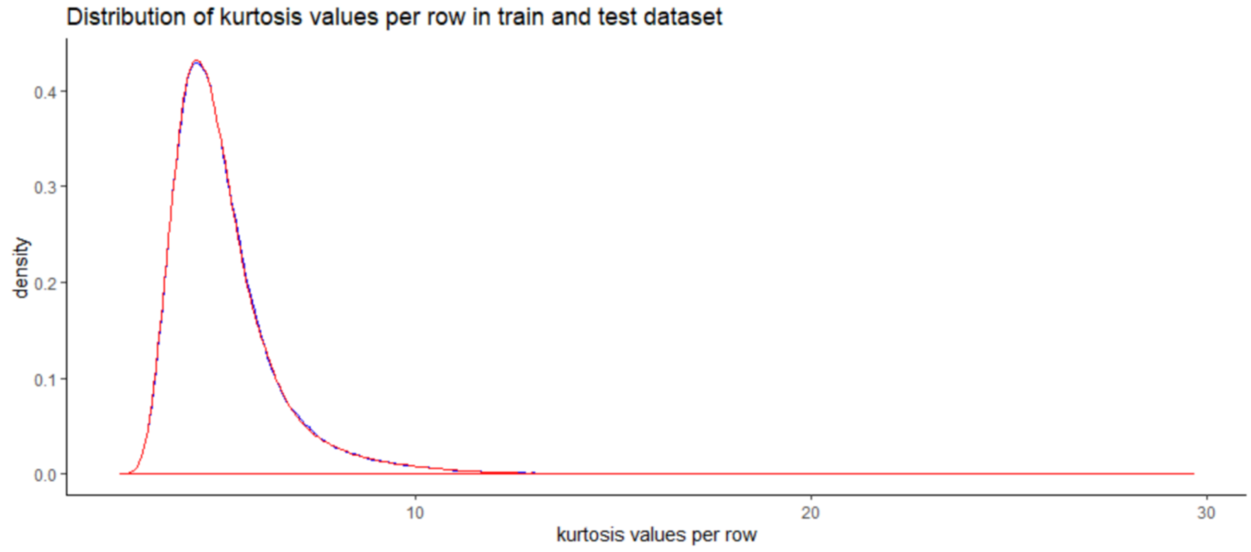
Distribution of std values per column in train and test dataset

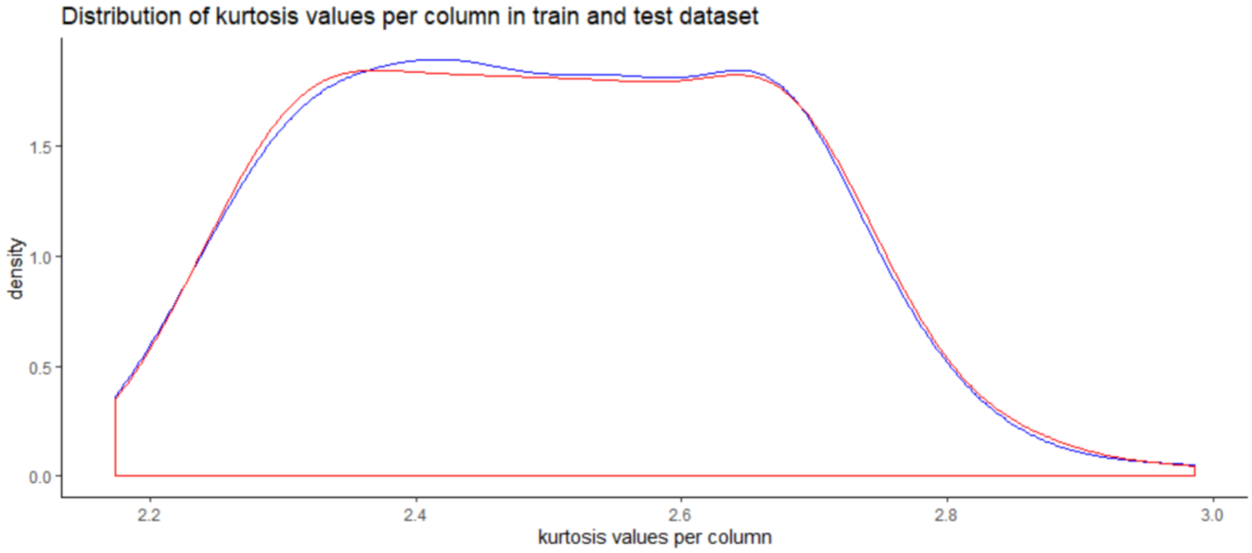


Distribution of skewness values per row in train and test dataset




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


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


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

Appendix B – Complete Python and R Code

Python Code

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',linewidth=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='test')

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='test')

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='train')

#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='test')

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='train')

#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='test')

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,label='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,label='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,label='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='quicksort').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='quicksort').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=1000,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_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)
```

Plotting important features:-

```
lgb.plot_importance(lgbm,max_num_features=50,importance_type="split",figsize=(20,50))
```

```
#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_classic()+

  #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')+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 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')+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(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')+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")
```

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=lgbm_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/