Santander customer transaction prediction

Submitted by

P.Lakshmi Amrutha Sai

Contents

Chapter 1
Introduction

1.1 Problem Statement

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

Background

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

Problem Analysis

This is a binary classification problem under supervised machine learning algorithm.
The task is to predict the value of target column in the train set.

1.2 Data

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

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

	ID_code	Target	var_0	var_1	...	var_197	var_198	Var_199
0	train_0	0	8.9255	-6.7863	...	8.5635	12.7803	-1.0914
1	train_1	0	11.5006	-4.1473	...	8.7889	18.3560	1.9518
2	train_2	0	8.6093	-2.7457	...	8.2675	14.7222	0.3965
3	train_3	0	11.0604	-2.1518	...	10.2922	17.9697	-8.9996
4	train_4	0	9.8369	-1.4834	...	9.5031	17.9974	8.8104

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

	ID_code	var_0	var_1	var_2	...	var_193	var_194	var_195	var_196	var_197	var_198	var_199
0	test_0	11.0656	7.7798	12.9536	...	2.4508	13.7112	2.4669	4.3654	10.7200	15.4722	-8.7197
1	test_1	8.5304	1.2543	11.3047	...	10.1282	15.5765	0.4773	-1.4852	9.8714	19.1293	-20.9760
2	test_2	5.4827	-10.3581	10.1407	...	2.1800	12.9813	2.1281	-7.1086	7.0618	19.8956	-23.1794
3	test_3	8.5374	-1.3222	12.0220	...	3.5813	15.1874	3.1656	3.9567	9.2295	13.0168	-4.2108
4	test_4	11.7058	-0.1327	14.1295	...	3.3778	19.5542	-0.2860	-5.1612	7.2882	13.9260	-9.1846

From the table below, we have the following 202 variables, using which we have to predict which customers will make a specific transaction in the future :

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

Table 1.3: Predictor Variables

Chapter 2
Methodology

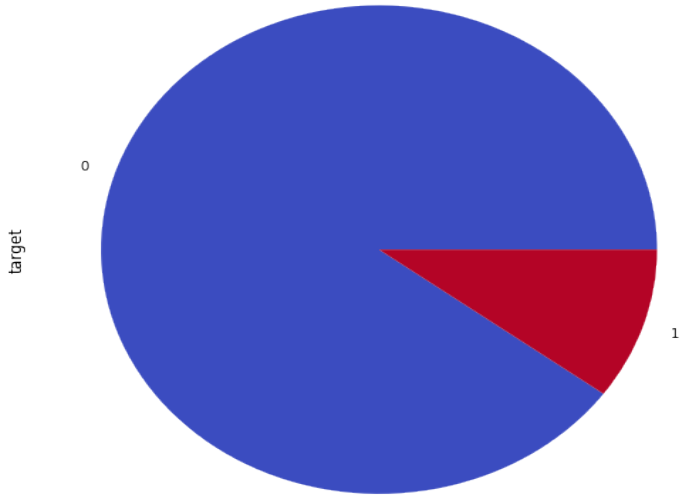
2.1 Exploratory Data Analysis (EDA)

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

Shape of data

200000 observations with 202 columns in train data and 200000 observations with 201 columns in test data .

2.1.1 Target classes count



Take away:

- Here we have a unbalanced data, where 90% of the data is the number of customers those will not make a transaction and 10% of the data is those who will make a transaction.

2.1.2 Missing value Analysis

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

R and Python code as follows :

#Missing values in train and test data

```
# R code
missing_val=
data.frame(missing_val=apply(train_df,2,function(x){sum(is.na(x))}))
missing_val=sum(missing_val)
missing_val=
data.frame(missing_val=apply(test_df,2,function(x){sum(is.na(x))}))
missing_val=sum(missing_val)

# Python code
train_missing=train_df.isnull().sum().sum()
test_missing=test_df.isnull().sum().sum()
```

2.1.3 Attributes distributions and trends

Distribution of columns :

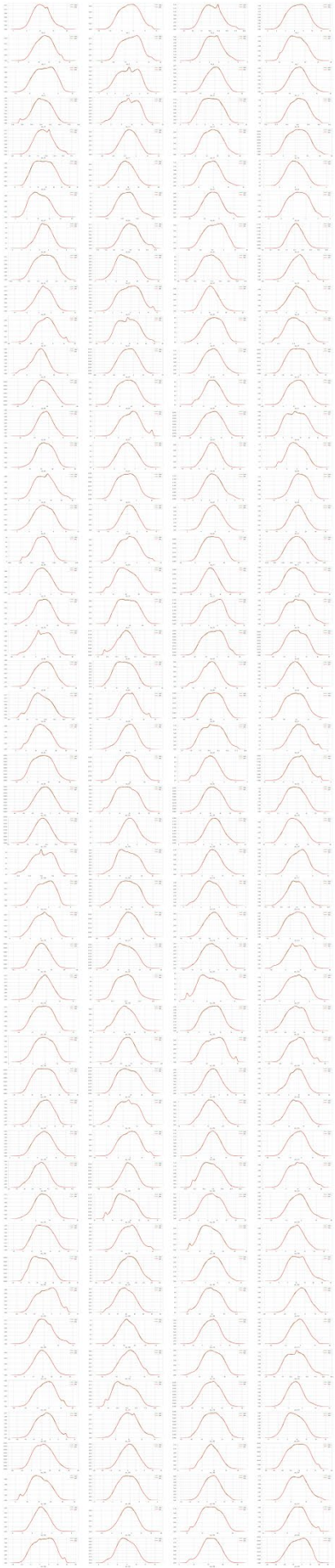
- Almost all features follow normalised distribution.

Distribution of train attributes

- We can observe that there is a considerable number of features which are significantly have different distributions for two target variables. For example like var_0,var_2,var_6,var_9,var_12, var_13 etc.
- We can observe that there is a considerable number of features which are significantly have same distributions for two target variables. For example like var_3, var_7,var_10,var_14,var_17 etc.

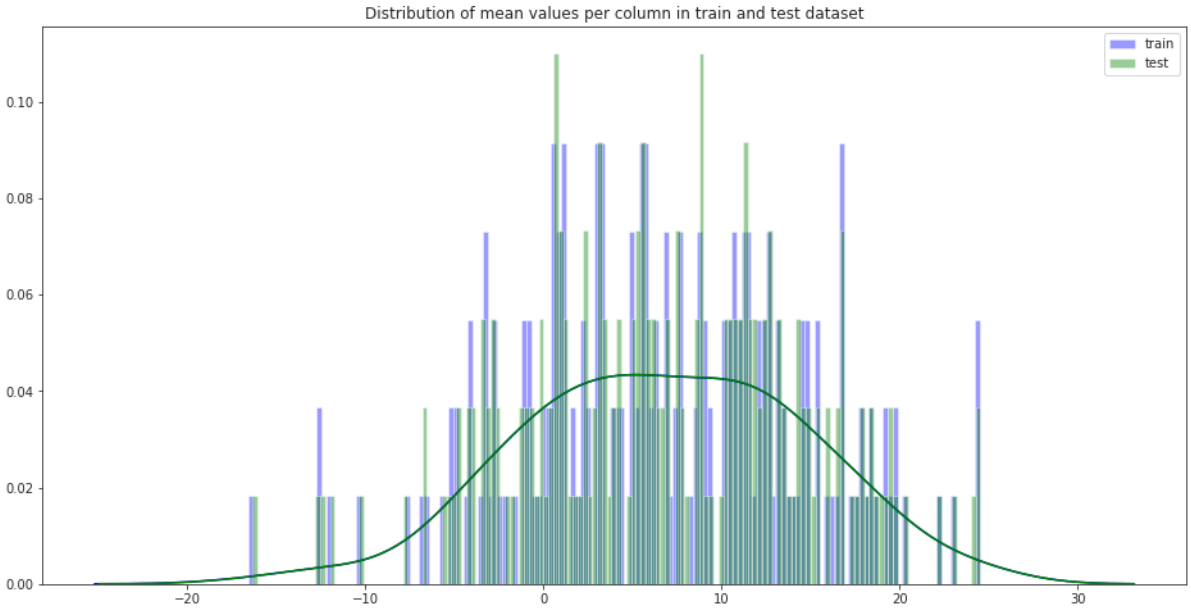
Distribution of numerical variables in train and test data

- The train and test data is well balanced with respect to distribution of numeric variables.

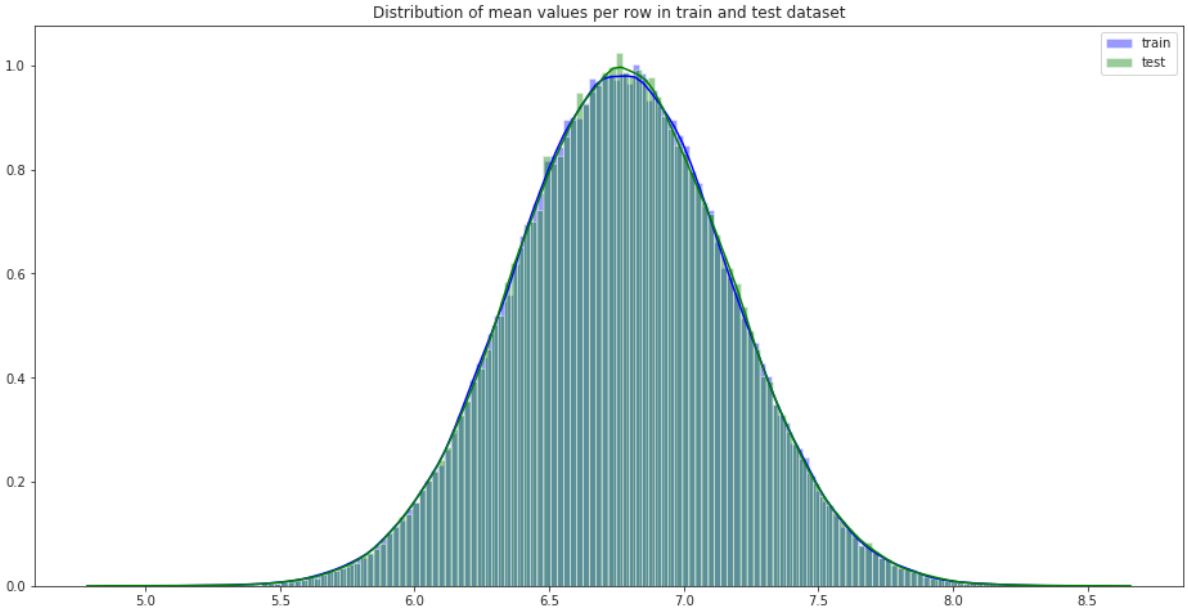


Distribution of mean values in both train and test dataset

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

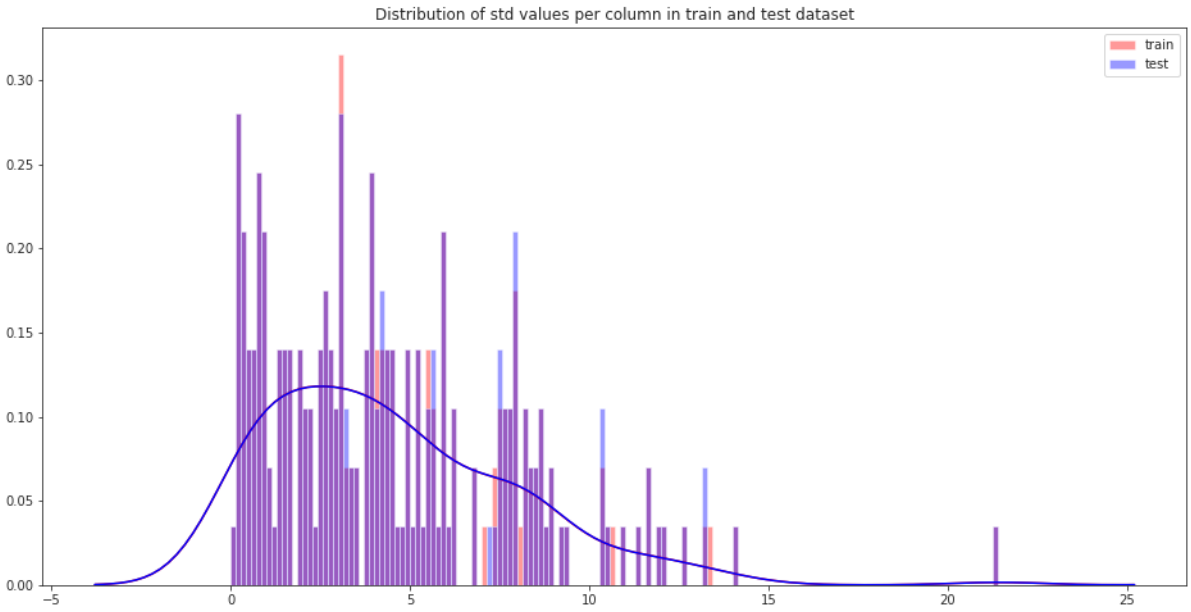


Let us look distribution of mean values per row in train and test dataset

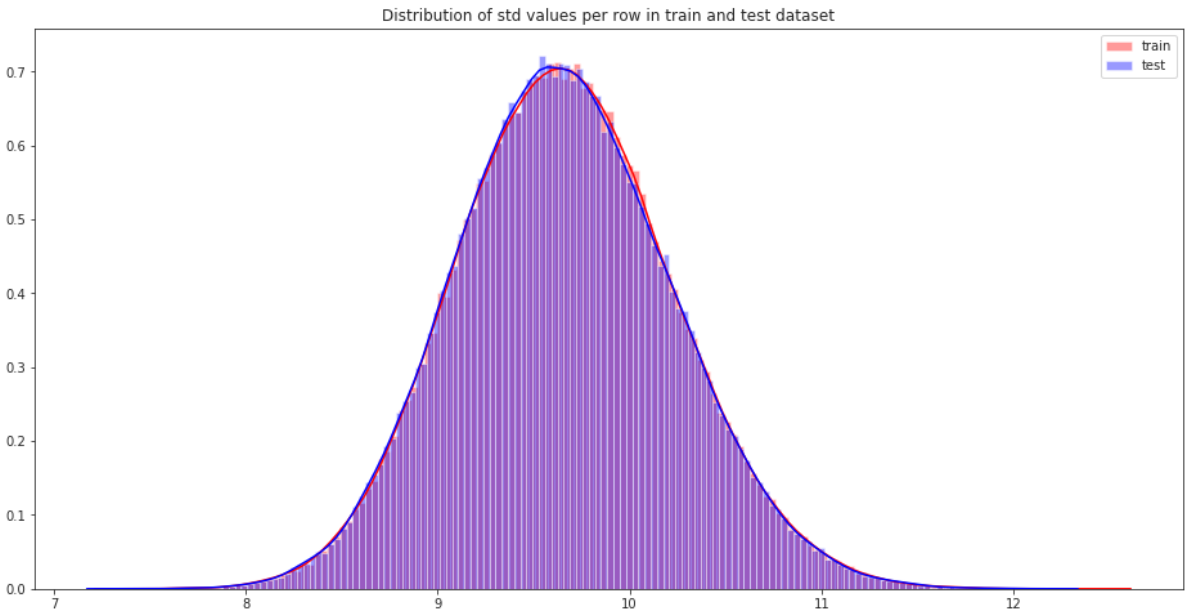


Distribution of standard deviation (std) values in train and test dataset

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

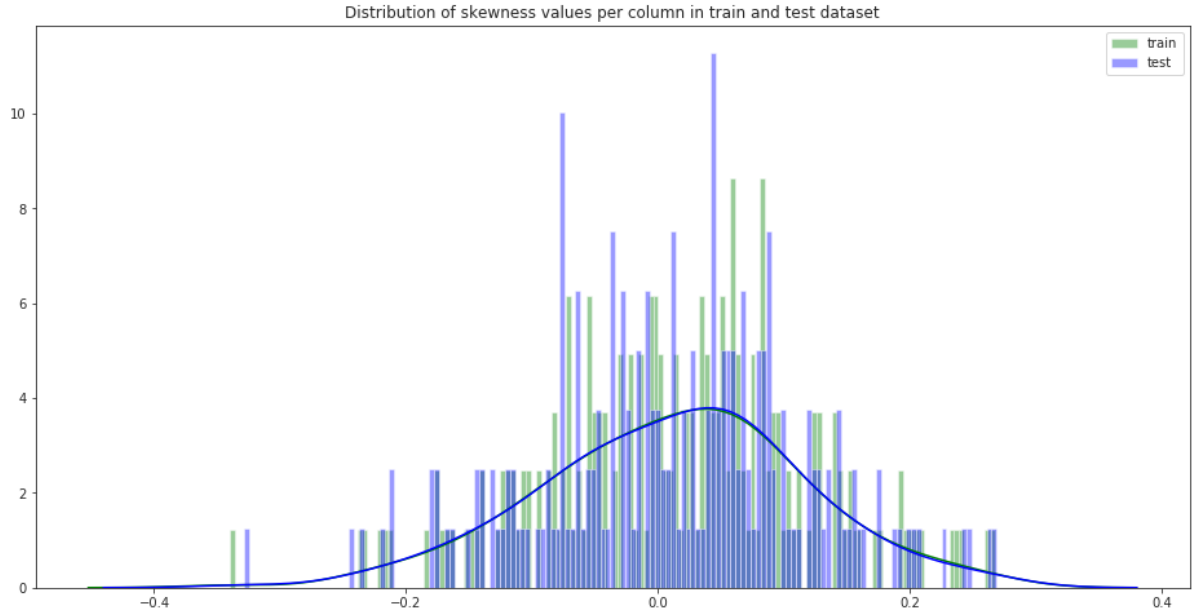


Let us look distribution of standard deviation (std) values per row in train and test dataset

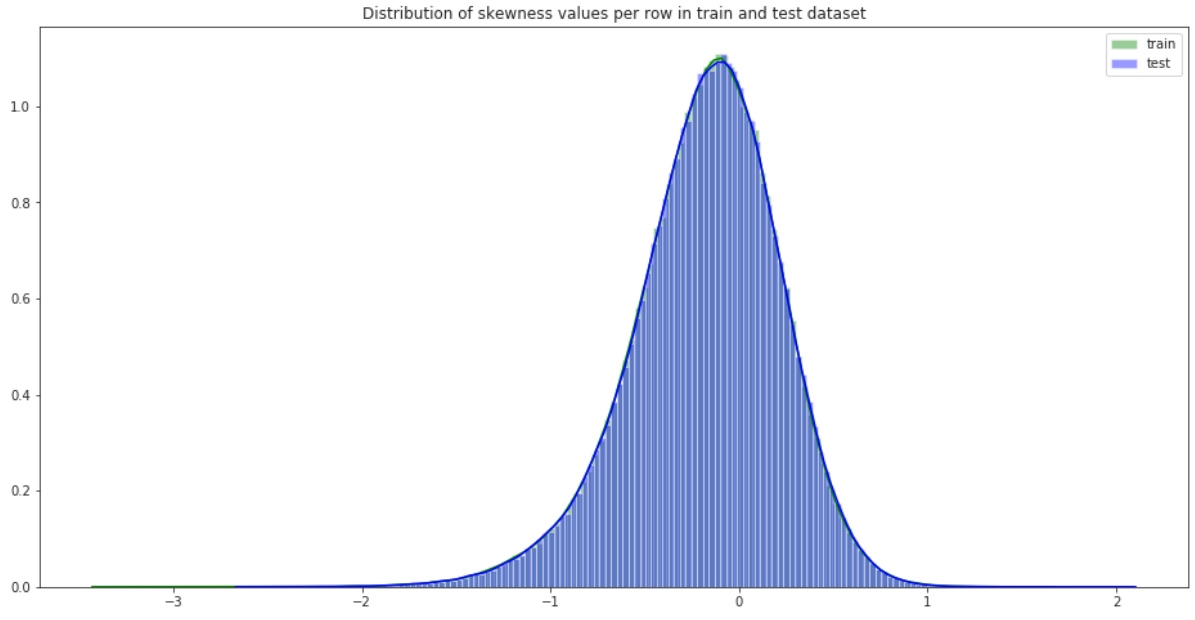


Distribution of skewness values in train and test dataset

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

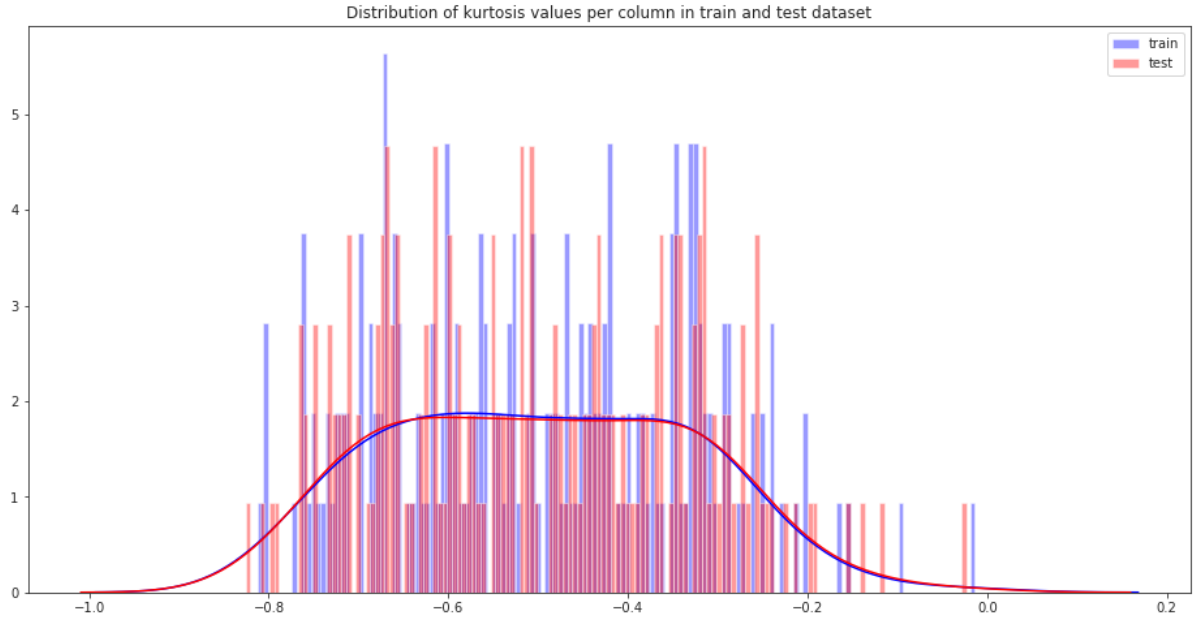


Let us look distribution of skewness per row in train and test dataset

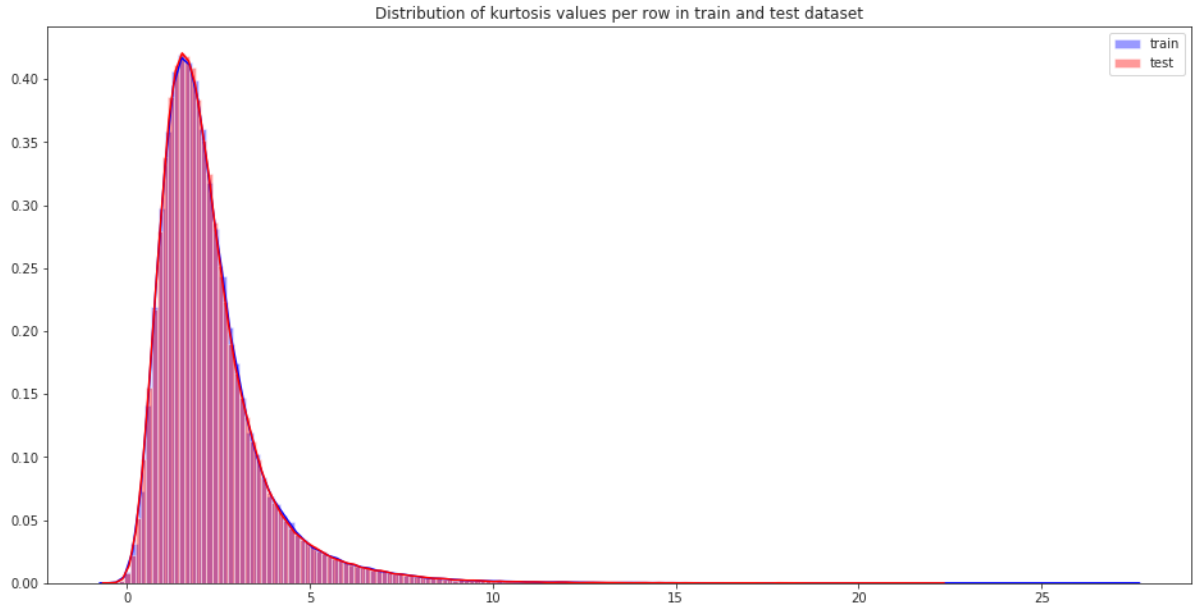


Distribution of kurtosis values in train and test dataset

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



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



Take away:

- Standard deviation is relatively large for both train and test variable data
- min, max, mean, median, sdt values for train and test data looks quite close.
- Mean values are distributed over a large range.
- Moreover mean and median have similar distribution.
- Both train and test are Leptokurtic and negatively skewed.

2.1.4 Outlier analysis

Outlier Analysis

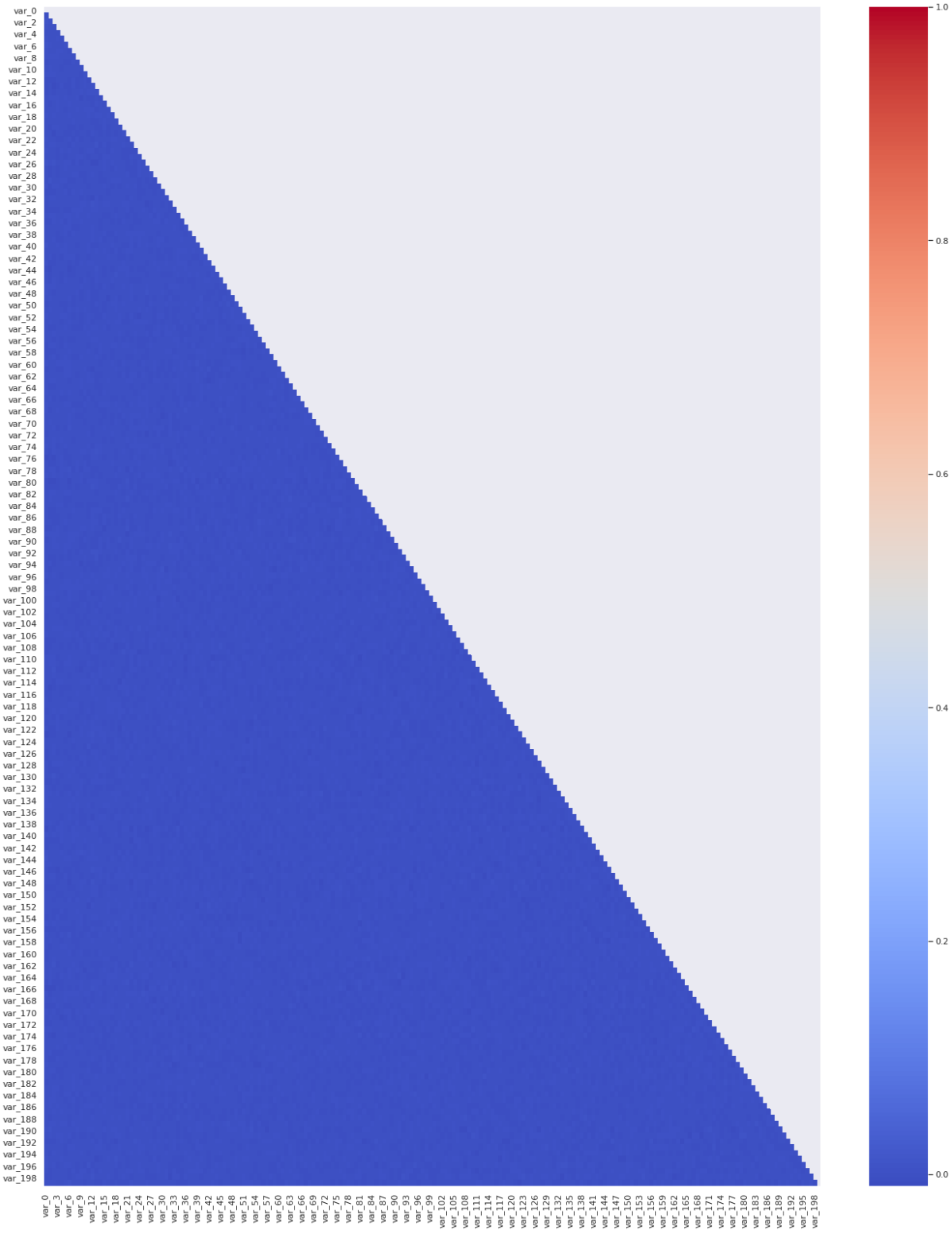
I can see from the above plots, that both the training and test sets are very similar to one another and that the distributions do not have any anomalies. And also in this project, we haven't perform outlier analysis as data is imbalanced .

2.1.5 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.

Heatmap

A **heatmap** is a graphical representation of data in which data values are represented as colors. That is, it uses color in order to communicate a value to the reader. This is a great tool to assist the audience towards the areas that matter the most when you have a large volume of data.



- The figure shows that most of the correlations between the numerical data are close to zero, in fact is between 0 and 0.2. That means that most of the numerical data are almost uncorrelated between them.

2.1.6 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.

Variable importance based on Mean Decrease Gini :

	MeanDecreaseGini
var_0	178.19905
var_1	179.11005
var_2	186.08049
var_3	124.84417
var_4	115.25478
var_5	136.53634
var_6	193.64204
var_7	111.92811
var_8	109.93159
var_9	159.85075
var_10	117.02210

Dealing Imbalanced dataset

Before modelling for this dataset let us understand how to deal with imbalanced dataset for classification problem. Traditional Machine Learning algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets. For any imbalanced data set, if the event to be predicted belongs to the minority class and the event rate is less than 10%, it is usually referred to as a rare event. The conventional model evaluation methods do not accurately measure model performance when faced with imbalanced datasets. Standard classifier algorithms like Decision Tree and Logistic Regression have a bias towards classes which have large number of instances. They tend to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.

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

2.2 Modeling

2.2.1 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.

Logistic regression

This is the classification problem. We can use logistic regression for this problem.
Logistic Regression is used when the dependent variable(target) is categorical. It has a bias towards classes which have large number of instances. It tends to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class.

Naive Bayes

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. It performs well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption). Also numerical variables are very less correlated.On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.

Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point inn-dimensional space (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes very well. Support Vectors are simply the coordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). It is effective in cases where the number of dimensions is greater than the number of samples. But it doesn't perform well, when we have large data set because the required training time is higher.

Random Forest

Random Forest is a bagging based ensemble learning model. Random forests is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming. Thus result (AUC-score) shown below for random Forest is not modified if num round would increase the score will be better but speed will be very slow. These results are shown for default values of Parameters.

LightGBM

LightGBM is Gradient Boosting ensemble model which is faster in speed and accuracy as compared to bagging and adaptive boosting. It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST. But parameter tuning in LightGBM should be done carefully.

Models Used

- Logistic regression
- LightGBM

2.2.2 Logistic Regression

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

Python code

```
#Training dataset for modelling
#Training data
X=train_df.drop(['ID_code','target'],axis=1)
Y=train_df['target']
#StratifiedKFold cross validator
cv=StratifiedKFold(n_splits=5,random_state=42,shuffle=True)
for train_index,valid_index in cv.split(X,Y):
        X_train,      X_valid=X.iloc[train_index],      X.iloc[valid_index]y_train,
        y_valid=Y.iloc[train_index], Y.iloc[valid_index]

#Logistic regression model
lr_model=LogisticRegression(random_state=42)
#fitting the lr model
lr_model.fit(X_train,y_train)
#Accuracy of the model
lr=lr_model.score(X_train,y_train)
```

Accuracy of the model : **0.914**

```
#Cross validation prediction
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 : 0.9132

```
#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
lr_pred=lr_model.predict(X_test)
```

R code

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

```
#Split the data using CreateDataPartition
train.index<-createDataPartition(train_df$target,p=0.8,list=FALSE)
train.data<-train_df[train.index,]
valid.data<-train_df[-train.index,]
#Training dataset
X_t<-as.matrix(train.data[,-c(1,2)])
y_t<-as.matrix(train.data$target)
```

```
#validation dataset
X_v<-as.matrix(valid.data[,-c(1,2)])
y_v<-as.matrix(valid.data$target)
#test dataset
test<-as.matrix(test_df[,-c(1)])
#Logistic regression model
set.seed(667)
lr_model <-glmnet(X_t,y_t, family = "binomial")
summary(lr_model)
#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
#Plotting the missclassification error vs log(lambda) where lambda is
regularization parameter
#Minimum lambda
cv_lr$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,test_df[,-c(1)],type='class')
```

Take away:

- 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.
- Both Oversampling and under sampling 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. In order to balance imbalanced data we are going to use SMOTE sampling method.
- **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.

2.2.3 LightGBM

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

Python code

Let us build LightGBM model

```
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)
#Selecting best hyper parameters by tuning of different parameters
params={'boosting_type': 'gbdt',
'max_depth' : -1, #no limit for max_depth if <0
'objective': 'binary',
'boost_from_average':False,
'nthread': 8,
'metric':'auc',
'num_leaves': 100,
'learning_rate': 0.03,
'max_bin': 950, #default 255
```

```
'subsample_for_bin': 200,
'subsample': 1,
'subsample_freq': 1,
'colsample_bytree': 0.8,
'reg_alpha': 1.2, #L1 regularization(>0)
'reg_lambda': 1.2,#L2 regularization(>0)
'min_split_gain': 0.5, #>0
'min_child_weight': 1,
'min_child_samples': 5,
'is_unbalance':True,
}
num_rounds=3000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],
verbose_eval=100,early_stopping_rounds = 1000)
X_test=test_df.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)
```

R code

```
#Convert data frame to matrix
X_train<-as.matrix(train_data[,-c(1,2)])
y_train<-as.matrix(train_data$target)
X_valid<-as.matrix(valid_data[,-c(1,2)])
y_valid<-as.matrix(valid_data$target)
test_data<-as.matrix(test_df[,-c(1)])
#training data
lgb.train <- lgb.Dataset(data=X_train, label=y_train)
#Validation data
lgb.valid <- lgb.Dataset(data=X_valid,label=y_valid)
#Choosing parameters
lgb.grid = list(objective = "binary",
metric = "auc",
boost =”gbdt”
min_sum_hessian_in_leaf = 1,
feature_fraction = 0.7,
bagging_fraction = 0.7,
bagging_freq = 5,
learning_rate=0.05,
num_leaves=80,
```

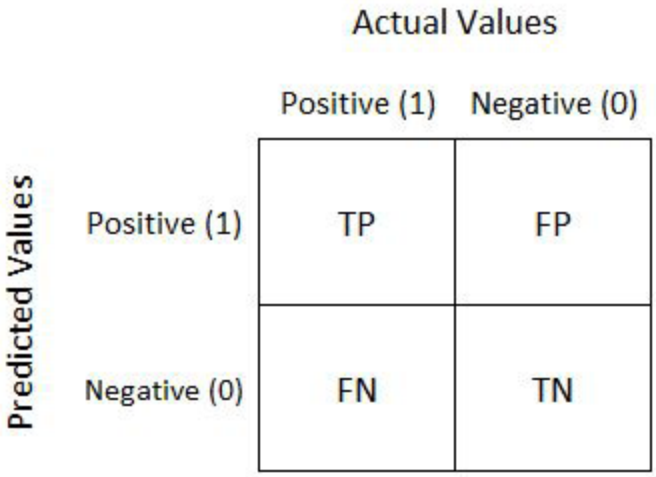
```
num_threads=10,
min_data = 100,
max_bin = 200,
lambda_l1 = 8,
lambda_l2 = 1.3,
min_data_in_bin=150,
min_gain_to_split = 20,
min_data_in_leaf = 40,
is_unbalance = TRUE)
lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,
eval_freq
=1000,valids=list(val1=lgb.train,val2=lgb.valid),early_stopping_rounds = 5000)
```

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.



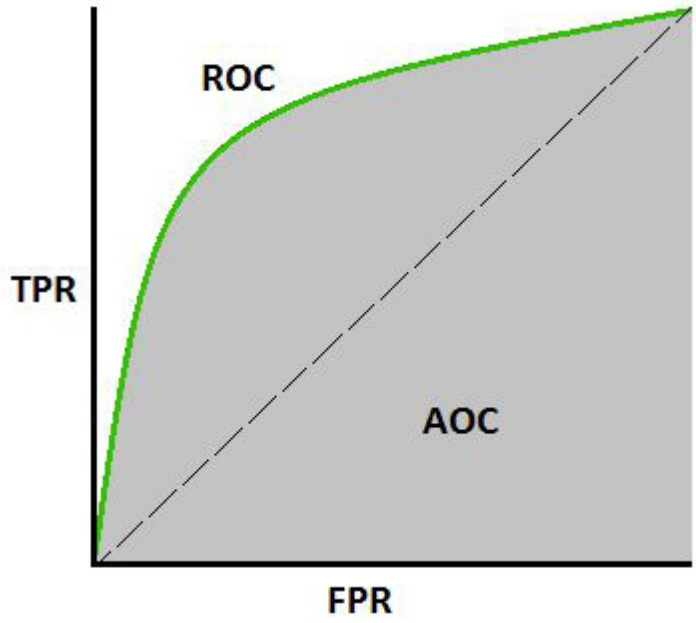
True Positive: You predicted positive and it's true.
True Negative: You predicted negative and it's true.
False Positive: (Type 1 Error) You predicted positive and it's false.
False Negative: (Type 2 Error) You predicted negative and it's false.
Based on confusion matrix we have following evaluation metrics-
Recall - Out of all the positive classes, how much we predicted correctly. It should be
high as possible.
Precision- Out of all the classes, how much we predicted correctly. It should be high as possible.

F-measure- It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure
Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

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

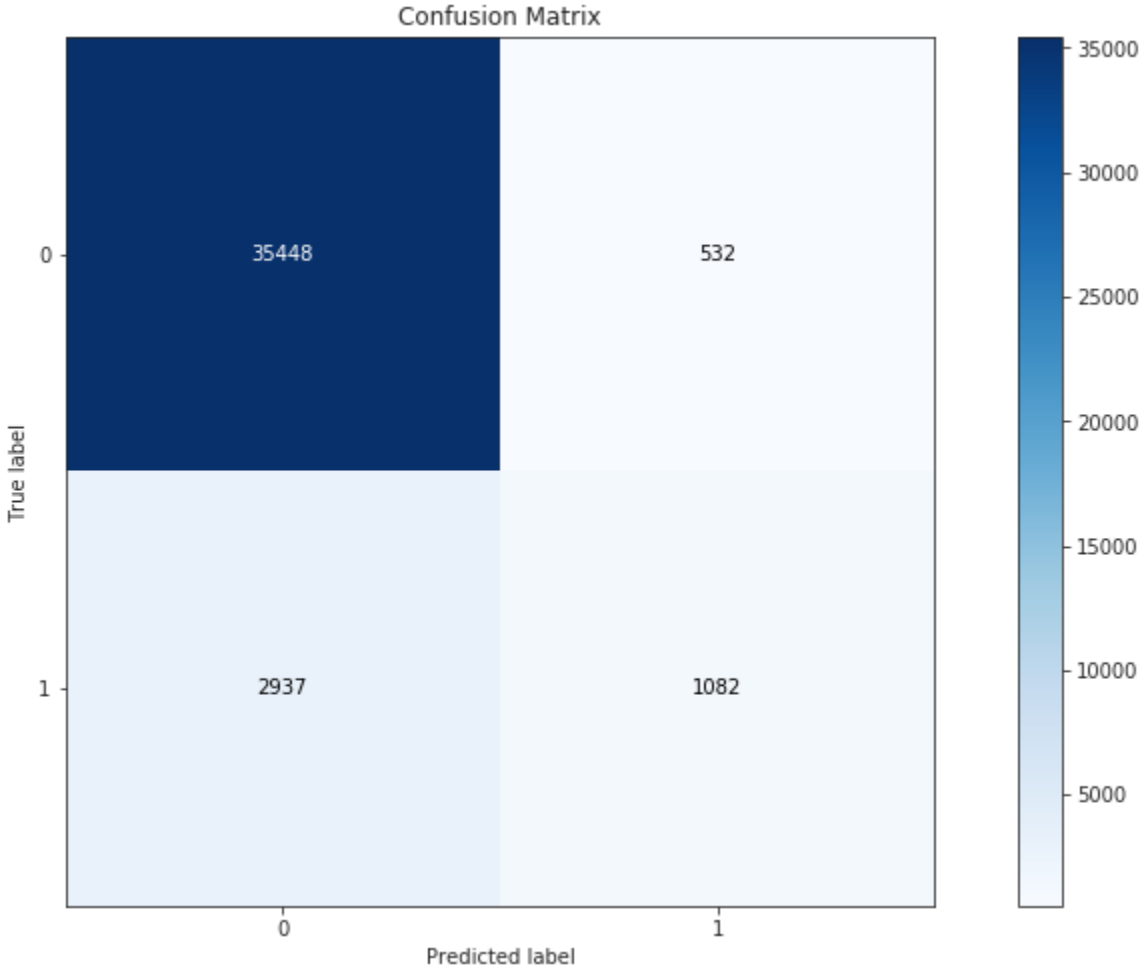
roc_auc_score :- It is a metric that computes the area under the Roc curve and also used metric for imbalanced data.

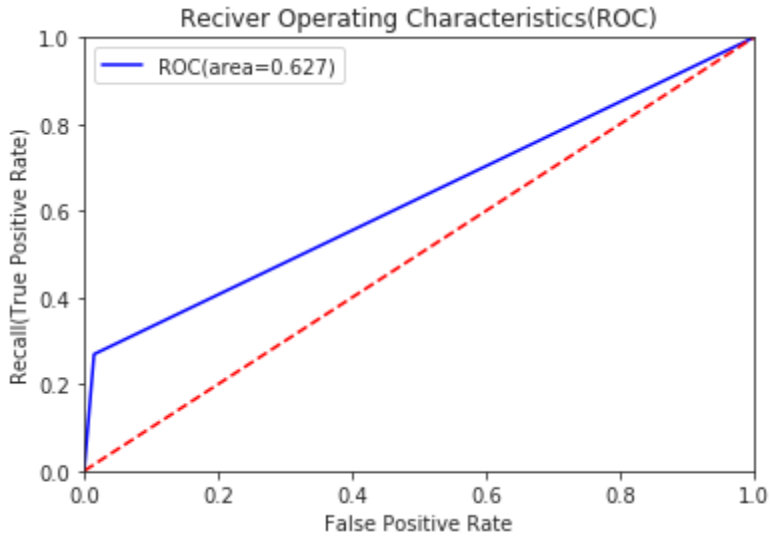
Roc curve is plotted true positive rate or Recall on y axis against false positive rate or specificity on x axis. The larger the area under the roc curve better the performance of the model.



An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity.

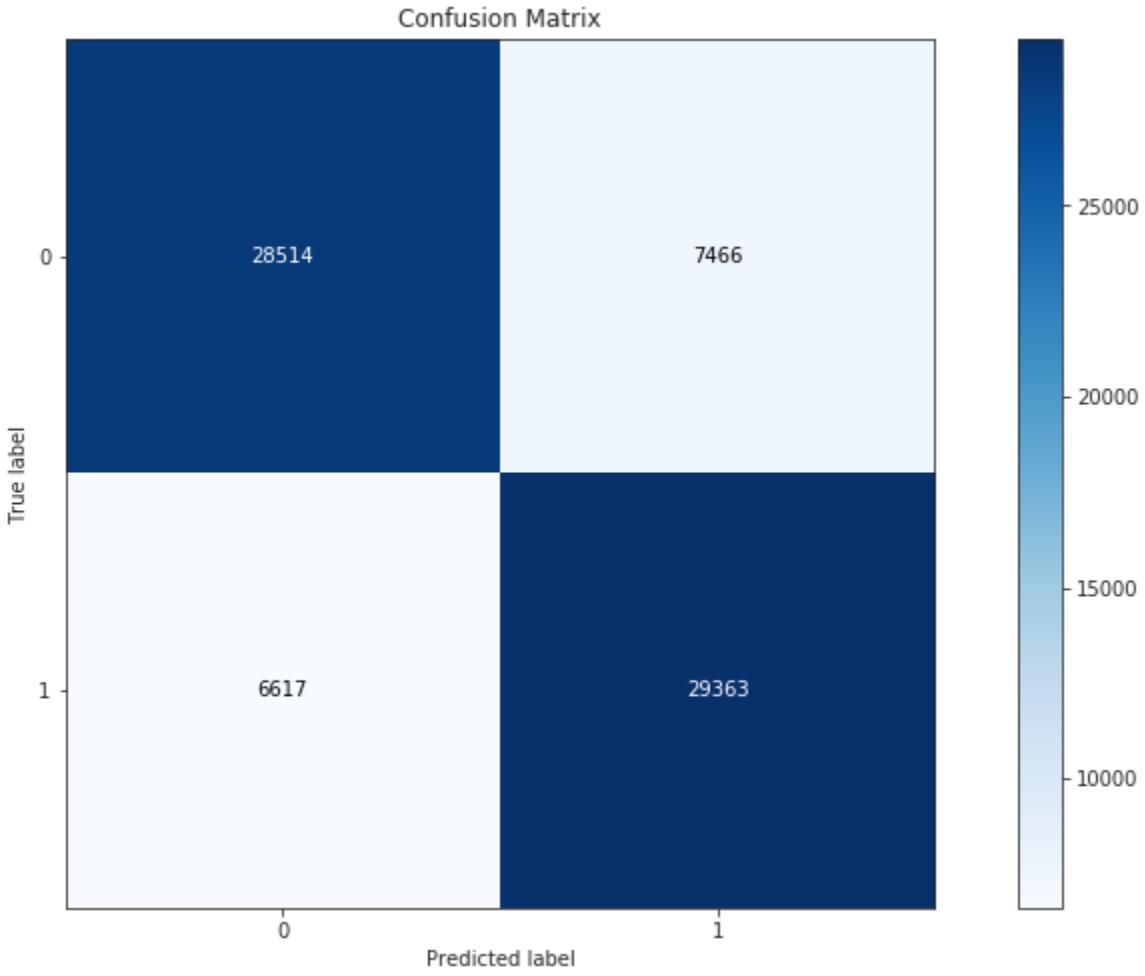
- **Logistic Regression**

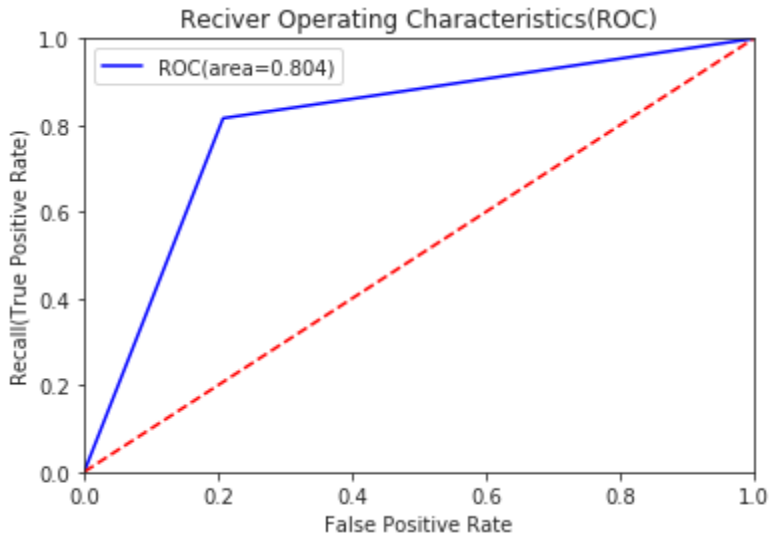




- When we compare the roc_auc_score and confusion matrix, we conclude that model is not performing well on imbalanced data.

- **LightGBM**





- When we compare the roc_auc_score and confusion matrix, we conclude that model is performing well on imbalanced data.

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 – Complete Python and R Code

Python Code

Exploratory Data Analysis

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import
train_test_split,cross_val_predict,cross_val_score
from sklearn.metrics import
roc_auc_score,confusion_matrix,make_scorer,classification_report,roc_curve,a
uc
from sklearn.model_selection import StratifiedKFold
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import ClusterCentroids,NearMiss,
RandomUnderSampler
import lightgbm as lgb
import eli5
from eli5.sklearn import PermutationImportance
from sklearn import tree
import graphviz
from pdpbox import pdp, get_dataset, info_plots
import scikitplot as skplt
from scikitplot.metrics import
plot_confusion_matrix,plot_precision_recall_curve
from scipy.stats import randint as sp_randint
import warnings
warnings.filterwarnings('ignore')
random_state=42
np.random.seed(random_state)
#############working directory##################
os.chdir("C:/Users/lenovo-pc/Desktop/Data Science Project 2")
#importing the train dataset
train_df=pd.read_csv('train.csv')
train_df.head()
#Shape of the train dataset
train_df.shape
#Summary of the dataset
```

```
train_df.describe()
```

Target classes count

```
%%time
#target classes count
target_class=train_df['target'].value_counts()
print('Count of target classes :\n',target_class)
#Percentage of target classes count
per_target_class=train_df['target'].value_counts()/len(train_df)*100
print('percentage of count of target classes :\n',per_target_class)
#Countplot and violin plot for target classes
fig,ax=plt.subplots(1,2,figsize=(20,5))
sns.countplot(train_df.target.values,ax=ax[0],palette='husl')
sns.violinplot(x=train_df.target.values,y=train_df.index.values,ax=ax[1],
palette='husl')
sns.stripplot(x=train_df.target.values,y=train_df.index.values,jitter=True,
color=' black',linewidth=0.5,size=0.5,alpha=0.5,ax=ax[1],palette='husl')
ax[0].set_xlabel('Target')
ax[1].set_xlabel('Target')
ax[1].set_ylabel('Index')
#plotting pie chart for target class
train['target'].value_counts().plot(kind='pie', figsize=(8,8))
```

Distribution of columns

```
#hisograms are used to check distribution of data
#draw histograms of numeric data in training set
print("Distribution of Columns")
plt.figure(figsize=(40,200))
for i,col in enumerate(numerical_features):
    plt.subplot(50,4,i+1)
    plt.hist(train[col])
    plt.title(col)
```

Distribution of train attributes

```
%%time
def plot_train_attribute_distribution(t0,t1,label1,label2,train_attributes):
i=0
sns.set_style('whitegrid')
fig=plt.figure()
ax=plt.subplots(10,10,figsize=(22,18))
for attribute in train_attributes:
i+=1
plt.subplot(10,10,i)
sns.distplot(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()
```
Let us see first 100 train attributes
```
%%time
t0=train_df[train_df.target.values==0]
t1=train_df[train_df.target.values==1]
train_attributes=train_df.columns.values[2:102]
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)
```
Next 100 train attributes
```
train_attributes=train_df.columns.values[102:203]
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)
```

Distribution of test attributes

```
#importing the test dataset
############working directory##################
os.chdir("C:/Users/lenovo-pc/Desktop/Data Science Project 2")
test_df=pd.read_csv('test.csv')
test_df.head()
#Shape of the test dataset
test_df.shape
def plot_test_attribute_distribution(test_attributes):
i=0
sns.set_style('whitegrid')
fig=plt.figure()
ax=plt.subplots(10,10,figsize=(22,18))
for attribute in test_attributes:
i+=1
plt.subplot(10,10,i)
sns.distplot(test_df[attribute],hist=False)
plt.xlabel('Attribute',)
sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
plt.show()
#Let us see first 100 test attributes
test_attributes=test_df.columns.values[1:101]
plot_test_attribute_distribution(test_attributes)
#Next 100 test attributes
test_attributes=test_df.columns.values[101:202]
plot_test_attribute_distribution(test_attributes)
```

Distribution of mean values in train and test dataset

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

```
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for mean values per column in train attributes
sns.distplot(train_df[train_attributes].mean(axis=0),color='blue',kde=True,bins=
150,label='train')
#Distribution plot for mean values per column in test attributes
sns.distplot(test_df[test_attributes].mean(axis=0),color='green',kde=True,bins=1
50,label='test')
plt.title('Distribution of mean values per column in train and test dataset')
plt.legend()
plt.show()



#Distribution of mean values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for mean values per row in train attributes
sns.distplot(train_df[train_attributes].mean(axis=1),color='blue',kde=True,bins=
150,label='train')
#Distribution plot for mean values per row in test attributes
sns.distplot(test_df[test_attributes].mean(axis=1),color='green',kde=True,
bins=150, label='test')
plt.title('Distribution of mean values per row in train and test dataset')
plt.legend()
plt.show()
```

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

```
%%time
#Distribution of std values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for std values per column in train attributes
sns.distplot(train_df[train_attributes].std(axis=0),color='red',kde=True,
bins=150,label='train')
#Distribution plot for std values per column in test attributes
sns.distplot(test_df[test_attributes].std(axis=0),color='blue',kde=True,bins=150,
label='test')
plt.title('Distribution of std values per column in train and test dataset')
```

```
plt.legend()
plt.show()
#Distribution of std values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for std values per row in train attributes
sns.distplot(train_df[train_attributes].std(axis=1),color='red',kde=True,bins=150
, label='train')
#Distribution plot for std values per row in test attributes
sns.distplot(test_df[test_attributes].std(axis=1),color='blue',kde=True, bins=150
, label='test')
plt.title('Distribution of std values per row in train and test dataset')
plt.legend()
```

Distribution of skewness in train and test dataset

```
%%time
#Distribution of skew values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for skew values per column in train attributes
sns.distplot(train_df[train_attributes].skew(axis=0),color='green',kde=True,

bins=150,label='train')
#Distribution plot for skew values per column in test attributes
sns.distplot(test_df[test_attributes].skew(axis=0),color='blue',kde=True,bins=15
0
,label='test')
plt.title('Distribution of skewness values per column in train and test dataset')
plt.legend()
plt.show()
#Distribution of skew values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for skew values per row in train attributes
sns.distplot(train_df[train_attributes].skew(axis=1),color='green',kde=True,
bins=150,label='train')
#Distribution plot for skew values per row in test attributes
sns.distplot(test_df[test_attributes].skew(axis=1),color='blue',kde=True,
bins=150, label='test')
plt.title('Distribution of skewness values per row in train and test dataset')
plt.legend()
plt.show()
```

Distribution of kurtosis values in train and test dataset

```
%%time
#Distribution of kurtosis values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train_df.columns.values[2:202]
#test attributes
test_attributes=test_df.columns.values[1:201]
#Distribution plot for kurtosis values per column in train attributes
sns.distplot(train_df[train_attributes].kurtosis(axis=0),color='blue',kde=True,
bins=150,label='train')
#Distribution plot for kurtosis values per column in test attributes
sns.distplot(test_df[test_attributes].kurtosis(axis=0),color='red',kde=True,
bins=150,label='test')
plt.title('Distribution of kurtosis values per column in train and test dataset')
plt.legend()
plt.show()
#Distribution of kutosis values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for kurtosis values per row in train attributes
sns.distplot(train_df[train_attributes].kurtosis(axis=1),color='blue',kde=True,
bins=150,label='train')
#Distribution plot for kurtosis values per row in test attributes
sns.distplot(test_df[test_attributes].kurtosis(axis=1),color='red',kde=True,
bins=150, label='test')
plt.title('Distribution of kurtosis values per row in train and test dataset')
plt.legend()
```

Missing value analysis and Correlations

```
%%time
#Finding the missing values in train and test data
train_missing=train_df.isnull().sum().sum()
test_missing=test_df.isnull().sum().sum()
print('Missing values in train data :',train_missing)
print('Missing values in test data :',test_missing
%%time
#Correlations in train attributes
train_attributes=train_df.columns.values[2:202]
train_correlations=train_df[train_attributes].corr().abs().unstack().sort_values(ki
nd='quicksort').reset_index()
train_correlations=train_correlations[train_correlations['level_0']!=train_correla
tions['level_1']]
print(train_correlations.head(10))
```

```
print(train_correlations.tail(10))
%%time
#Correlations in test attributes
test_attributes=test_df.columns.values[1:201]
test_correlations=test_df[test_attributes].corr().abs().unstack().sort_values(kind
='quicksort').reset_index()
test_correlations=test_correlations[test_correlations['level_0']!=test_correlations
['level_1']]
print(test_correlations.head(10))
print(test_correlations.tail(10))
```

correlations between numerical data

```
sns.set(rc={'figure.figsize':(20,28)})
# Compute the correlation matrix
corr = train[numerical_features].corr()
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

sns.heatmap(corr, mask=mask,
            #annot=True,
            #fmt=".2f",
            cmap='coolwarm')
```

Feature engineering

```
#training data
X=train_df.drop(columns=['ID_code','target'],axis=1)
test=test_df.drop(columns=['ID_code'],axis=1)
y=train_df['target']
#Split the training data
X_train,X_valid,y_train,y_valid=train_test_split(X,y,random_state=42)
print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)
%%time
#Random forest classifier
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)
#fitting the model
rf_model.fit(X_train,y_train)
#Permutation importance
%%time
from eli5.sklearn import PermutationImportance
```

```
perm_imp=PermutationImportance(rf_model,random_state=42)
#fitting the model
perm_imp.fit(X_valid,y_valid)
%%time
#Important features
eli5.show_weights(perm_imp,feature_names=X_valid.columns.tolist(),top=200
)
#partial dependence plots
%%time
#Create the data we will plot 'var_81'
features=[v for v in X_valid.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,f
eature='var_81')
#plot feature "var_81"
pdp.pdp_plot(pdp_data,'var_81')
plt.show()
%%time
#Create the data we will plot
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,f
eature=' var_109')
#plot feature "var_109"
pdp.pdp_plot(pdp_data,'var_109')
plt.show()
```

Handling of imbalanced data

```
#Training data
X=train_df.drop(['ID_code','target'],axis=1)
Y=train_df['target']
#StratifiedKFold cross validator
cv=StratifiedKFold(n_splits=5,random_state=42,shuffle=True)
for train_index,valid_index in cv.split(X,Y):
X_train, X_valid=X.iloc[train_index], X.iloc[valid_index]
y_train, y_valid=Y.iloc[train_index], Y.iloc[valid_index]
print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)
%%time
#Logistic regression model
lr_model=LogisticRegression(random_state=42)
#fitting the lr model
lr_model.fit(X_train,y_train)
```

```
#Accuracy of the model
lr_score=lr_model.score(X_train,y_train)
print('Accuracy of the lr_model :',lr_score)
%%time
#Cross validation prediction
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)
#Plot the confusion matrix
plot_confusion_matrix(y_valid,cv_predict,normalize=False,figsize=(15,8))
#ROC_AUC score
roc_score=roc_auc_score(y_valid,cv_predict)
print('ROC score :',roc_score)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_valid,cv_predict)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')

plt.show()
print('AUC:',roc_auc)
#Classification report
scores=classification_report(y_valid,cv_predict)
print(scores)
%%time
#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
lr_pred=lr_model.predict(X_test)
print(lr_pred)
#Synthetic Minority Oversampling Technique
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
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score :',np.average(cv_score))
#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
#Plot the confusion matrix
plot_confusion_matrix(y_smote_v,cv_pred,normalize=False,figsize=(15,8))
#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)
%%time
#Predicting the model
X_test=test_df.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
print(smote_pred)
```

LightGBM

```
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)
#Selecting best hyperparameters by tuning of different parameters
params={'boosting_type': 'gbdt',
'max_depth' : -1, #no limit for max_depth if <0
'objective': 'binary',
'boost_from_average':False,
'nthread': 8,
'metric':'auc',
'num_leaves': 100,
'learning_rate': 0.03,
'max_bin': 950, #default 255
'subsample_for_bin': 200,
'subsample': 1,
'subsample_freq': 1,
'colsample_bytree': 0.8,
'reg_alpha': 1.2, #L1 regularization(>0)
'reg_lambda': 1.2,#L2 regularization(>0)
'min_split_gain': 0.5, #>0
'min_child_weight': 1,
'min_child_samples': 5,
'is_unbalance':True,
}
num_rounds=3000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],
verbose_eval=100,early_stopping_rounds = 1000)
X_test=test_df.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)
#plot the important features
lgb.plot_importance(lgbm,max_num_features=150,importance_type="split",fig
size=(20,50))
```

```
#final submission
sub_df=pd.DataFrame({'ID_code':test_df['ID_code'].values})
sub_df['lgbm_predict_prob']=lgbm_predict_prob
sub_df['lgbm_predict']=lgbm_predict
sub_df['smote_predict']=smote_pred
sub_df.to_csv('submission.csv',index=False)
sub_df.head()
```

R code

```
#Project Title:Santander Customer Transaction Prediction using R

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

#Clearing Objects:-
rm(list=ls(all=T))

#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("C:/Users/lenovo-pc/Desktop/Data Science Project 2")
```

```
#Importing the training Data:-
df_train=read.csv("train.csv")
anahead(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='ligh
tgreen')

#Violin with jitter plots for target classes
plot2=ggplot(df_train,aes(x=target,y=1:nrow(df_train)))+theme_bw()+geom_vi
olin(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')+t
heme_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='g
reen')+
  labs(x='mean values per column',title="Distribution of mean values per
column in train and test dataset")


#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')+them
e_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")

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

#Applying the function to find kurtosis values per row in train and test data.
train_kurtosis<-apply(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,colo
r='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,colo
r='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)