Santander Customer Transaction

**Contents:**

1. **Introduction**
   1. **Problem Statement**
   2. **Data**
2. **Methodology**
   1. **Pre Processing**
      1. **Missing Value**
      2. **Plots for all the statistical parameters**
   2. **Modelling**
      1. **Logistic regression**
      2. **Logistic Regression with SMOTE /ROSE**
      3. **Random Forests**

**3 Conclusion**

**3.1 Model Evaluation**

**3.1.1 Confusion Matrix.**

**Appendix A – Python code**

**Appendix B – R Code**

**Chapter 1**

**Introduction**

**1.1 Problem Statement**

In this [competition](https://www.kaggle.com/c/santander-customer-transaction-prediction/), Kagglers are supposed to identify which customers will make a specific transaction in the future. You are provided with an anonymized dataset containing 200 numeric feature variables, the binary target column (with imbalanced classes), and a string ID\_code column. The task is to predict the value of target column in the test set. Submissions are evaluated on [area under the ROC curve](http://en.wikipedia.org/wiki/Receiver_operating_characteristic) between the predicted probability and the observed target.

**1.2 Data**

We are going to predict the Customer transaction through the historical metrics. Following are the files used in this project.

**Train Data**

| **ID\_code** | **target** | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **var\_4** | **var\_5** | **var\_6** | **var\_7** | **...** |  | **var\_190** | **var\_191** | **var\_192** | **var\_193** | **var\_194** | **var\_195** | **var\_196** | **var\_197** | **var\_198** | **var\_199** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | train\_0 | 0 | 8.9255 | -6.7863 | 11.9081 | 5.0930 | 11.4607 | -9.2834 | 5.1187 | 18.6266 | ... | 4.4354 | 3.9642 | 3.1364 | 1.6910 | 18.5227 | -2.3978 | 7.8784 | 8.5635 | 12.7803 | -1.0914 |
| **1** | train\_1 | 0 | 11.5006 | -4.1473 | 13.8588 | 5.3890 | 12.3622 | 7.0433 | 5.6208 | 16.5338 | ... | 7.6421 | 7.7214 | 2.5837 | 10.9516 | 15.4305 | 2.0339 | 8.1267 | 8.7889 | 18.3560 | 1.9518 |
| **2** | train\_2 | 0 | 8.6093 | -2.7457 | 12.0805 | 7.8928 | 10.5825 | -9.0837 | 6.9427 | 14.6155 | ... | 2.9057 | 9.7905 | 1.6704 | 1.6858 | 21.6042 | 3.1417 | -6.5213 | 8.2675 | 14.7222 | 0.3965 |
| **3** | train\_3 | 0 | 11.0604 | -2.1518 | 8.9522 | 7.1957 | 12.5846 | -1.8361 | 5.8428 | 14.9250 | ... | 4.4666 | 4.7433 | 0.7178 | 1.4214 | 23.0347 | -1.2706 | -2.9275 | 10.2922 | 17.9697 | -8.9996 |
| **4** | train\_4 | 0 | 9.8369 | -1.4834 | 12.8746 | 6.6375 | 12.2772 | 2.4486 | 5.9405 | 19.2514 | ... | -1.4905 | 9.5214 | -0.1508 | 9.1942 | 13.2876 | -1.5121 | 3.9267 | 9.5031 | 17.9974 | -8.8104 |
| **5** | train\_5 | 0 | 11.4763 | -2.3182 | 12.6080 | 8.6264 | 10.9621 | 3.5609 | 4.5322 | 15.2255 | ... | -6.3068 | 6.6025 | 5.2912 | 0.4403 | 14.9452 | 1.0314 | -3.6241 | 9.7670 | 12.5809 | -4.7602 |
| **6** | train\_6 | 0 | 11.8091 | -0.0832 | 9.3494 | 4.2916 | 11.1355 | -8.0198 | 6.1961 | 12.0771 | ... | 8.7830 | 6.4521 | 3.5325 | 0.1777 | 18.3314 | 0.5845 | 9.1104 | 9.1143 | 10.8869 | -3.2097 |
| **7** | train\_7 | 0 | 13.5580 | -7.9881 | 13.8776 | 7.5985 | 8.6543 | 0.8310 | 5.6890 | 22.3262 | ... | 13.1700 | 6.5491 | 3.9906 | 5.8061 | 23.1407 | -0.3776 | 4.2178 | 9.4237 | 8.6624 | 3.4806 |
| **8** | train\_8 | 0 | 16.1071 | 2.4426 | 13.9307 | 5.6327 | 8.8014 | 6.1630 | 4.4514 | 10.1854 | ... | 1.4298 | 14.7510 | 1.6395 | 1.4181 | 14.8370 | -1.9940 | -1.0733 | 8.1975 | 19.5114 | 4.8453 |
| **9** | train\_9 | 0 | 12.5088 | 1.9743 | 8.8960 | 5.4508 | 13.6043 | -16.2859 | 6.0637 | 16.8410 | ... | 0.5543 | 6.3160 | 1.0371 | 3.6885 | 14.8344 | 0.4467 | 14.1287 | 7.9133 | 16.2375 | 14.25 |

**Test Data**

| **ID\_code** | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **var\_4** | **var\_5** | **var\_6** | **var\_7** | **var\_8** | **...** |  | **var\_190** | **var\_191** | **var\_192** | **var\_193** | **var\_194** | **var\_195** | **var\_196** | **var\_197** | **var\_198** | **var\_199** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | test\_0 | 11.0656 | 7.7798 | 12.9536 | 9.4292 | 11.4327 | -2.3805 | 5.8493 | 18.2675 | 2.1337 | ... | -2.1556 | 11.8495 | -1.4300 | 2.4508 | 13.7112 | 2.4669 | 4.3654 | 10.7200 | 15.4722 | -8.7197 |
| **1** | test\_1 | 8.5304 | 1.2543 | 11.3047 | 5.1858 | 9.1974 | -4.0117 | 6.0196 | 18.6316 | -4.4131 | ... | 10.6165 | 8.8349 | 0.9403 | 10.1282 | 15.5765 | 0.4773 | -1.4852 | 9.8714 | 19.1293 | -20.9760 |
| **2** | test\_2 | 5.4827 | -10.3581 | 10.1407 | 7.0479 | 10.2628 | 9.8052 | 4.8950 | 20.2537 | 1.5233 | ... | -0.7484 | 10.9935 | 1.9803 | 2.1800 | 12.9813 | 2.1281 | -7.1086 | 7.0618 | 19.8956 | -23.1794 |
| **3** | test\_3 | 8.5374 | -1.3222 | 12.0220 | 6.5749 | 8.8458 | 3.1744 | 4.9397 | 20.5660 | 3.3755 | ... | 9.5702 | 9.0766 | 1.6580 | 3.5813 | 15.1874 | 3.1656 | 3.9567 | 9.2295 | 13.0168 | -4.2108 |
| **4** | test\_4 | 11.7058 | -0.1327 | 14.1295 | 7.7506 | 9.1035 | -8.5848 | 6.8595 | 10.6048 | 2.9890 | ... | 4.2259 | 9.1723 | 1.2835 | 3.3778 | 19.5542 | -0.2860 | -5.1612 | 7.2882 | 13.9260 | -9.1846 |
| **5** | test\_5 | 5.9862 | -2.2913 | 8.6058 | 7.0685 | 14.2465 | -8.6761 | 4.2467 | 14.7632 | 1.8790 | ... | -2.1115 | 7.1178 | -0.4249 | 8.8781 | 14.9438 | -2.2151 | -6.0233 | 9.8117 | 17.1127 | 10.8240 |
| **6** | test\_6 | 8.4624 | -6.1065 | 7.3603 | 8.2627 | 12.0104 | -7.2073 | 4.1670 | 13.0809 | -4.3004 | ... | 12.3609 | 6.8661 | 4.0971 | 8.8484 | 17.5010 | 0.0295 | 7.7443 | 9.1509 | 18.4736 | 5.1499 |
| **7** | test\_7 | 17.3035 | -2.4212 | 13.3989 | 8.3998 | 11.0777 | 9.6449 | 5.9596 | 17.8477 | -4.8068 | ... | 4.4676 | 4.4214 | 0.9303 | 1.4994 | 15.2648 | -1.7931 | 6.5316 | 10.4855 | 23.4631 | 0.7283 |
| **8** | test\_8 | 6.9856 | 0.8402 | 13.7161 | 4.7749 | 8.6784 | -13.7607 | 4.3386 | 14.5843 | 2.5883 | ... | -3.4657 | 7.8754 | 2.4698 | -0.0362 | 16.7144 | 0.1221 | -1.4328 | 9.9207 | 16.9865 | -3.3304 |

**Chapter 2 :**

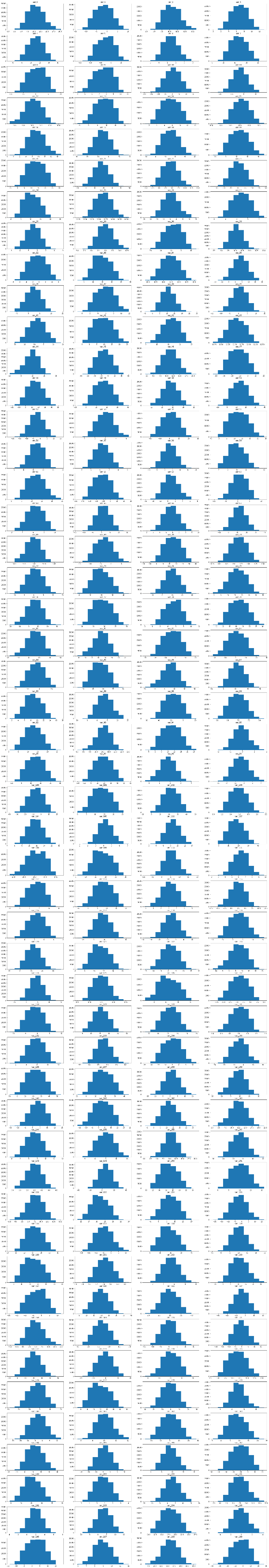
**2.1 Pre processing:**

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms *looking at data* refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**.

1. We check for null values in train data

Code: train\_df.isna().sum().sum() #checking for train missing values

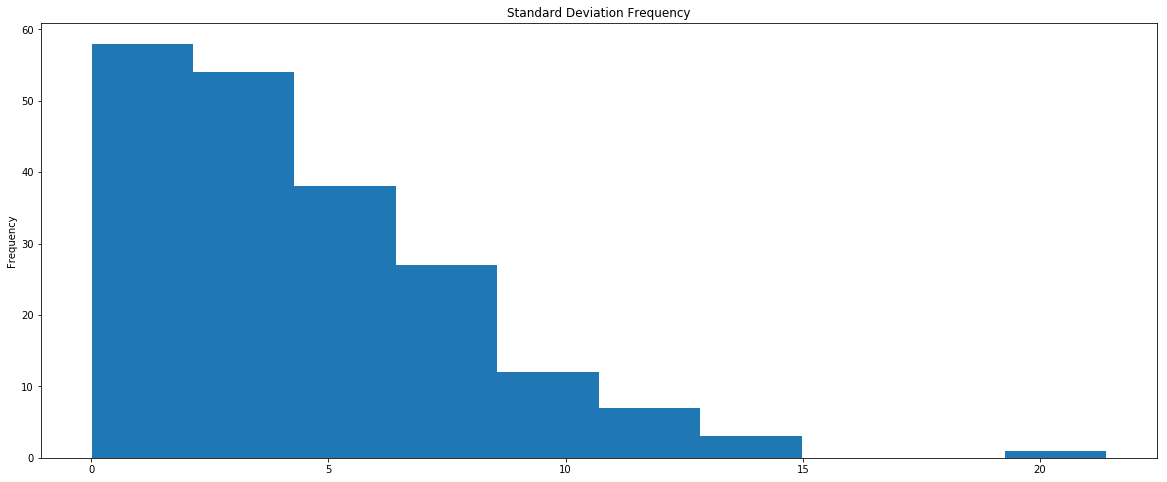
1. Check the distribution of coloumns so that its easier for us to understand the data better .



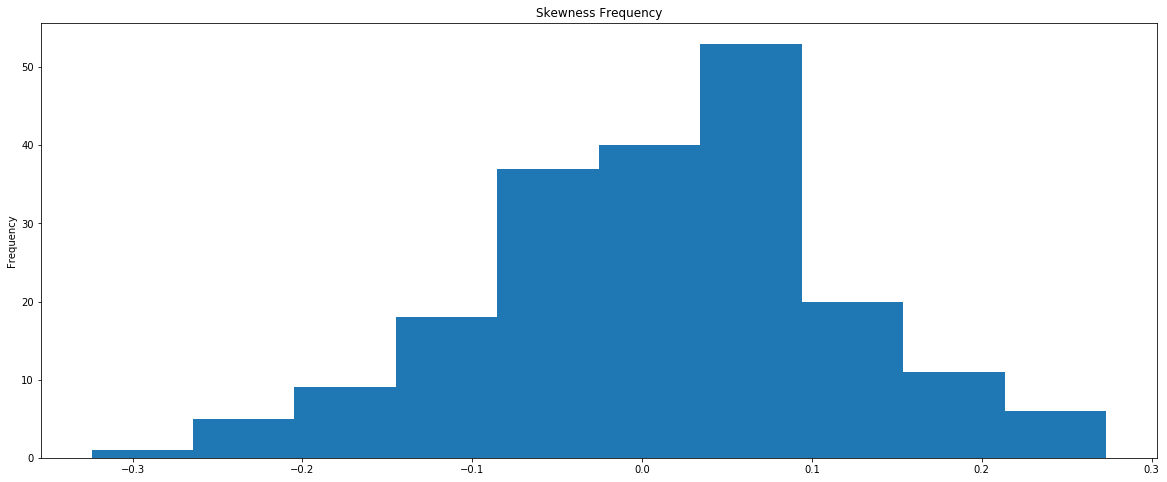
1. Check the mean distribution.



1. check the Standard deviation distribution



1. Check the frequesncy of skewness



1. Check Kurtosis frequency
2. 
3. Check the correlation of variables .

corr = train\_df[numerical\_features].corr()

s = corr.unstack().drop\_duplicates()

so = s.sort\_values(kind="quicksort")

so = so.drop\_duplicates()

print("Top most highly positive correlated features:")

print(so[(so<1) & (so>0.5)].sort\_values(ascending=False))

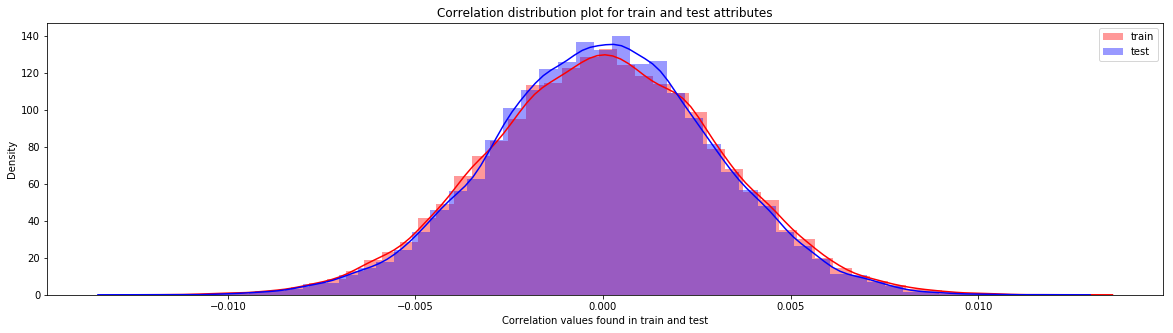
print()

print("Top most highly megative correlated features:")

print(so[(so < - 0.005)])

We see that we have majority of the variables mostly negatively correlated with each other .

1. Checking the correlation of variables with numerical features. And plotting the same.



The above graph incluses all the data on fare amount even when it has negative and null values . Hence we need to set a boundary . In this case , I have set the boundary for fare amount between 0 to 200. Below is the histogram distribution of the fare amount. The distribution looksmuch better.

**2.2.1> Model Selection:**

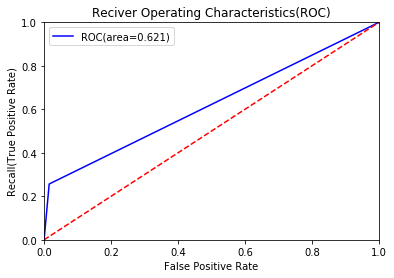
As this is a cla]ssification problem as the output is to predict if the cx will perform any transaction with us or not . Hence since the answer is in “yes” or “ no “ . We will go with a simple model which is logistic regresion.

lr\_model=LogisticRegression(random\_state=42)

Accuracy of the lr\_model : 0.9146363058718139

cross\_val\_score : 0.9117879853441689

we will build the confusion matrix and the plot :



We know that this model is not giving enough insights on the accuracy hence we wil lgo for the second model . SMOTE

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

#Generating synthetic data points

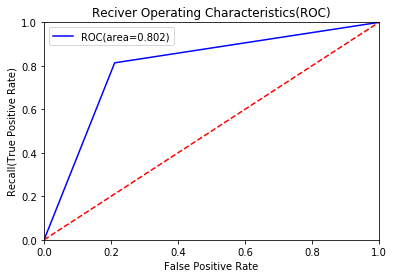
X\_smote,y\_smote=sm.fit\_sample(X\_train,y\_train)

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

Accuracy of the smote\_model : 0.8000456925508793

cross\_val\_score : 0.8015370370370369

We will again biuld the confusion matrix .



The above model is worse than logistic regression . Hence we wil try Random forests now .

#Random forest classifier

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

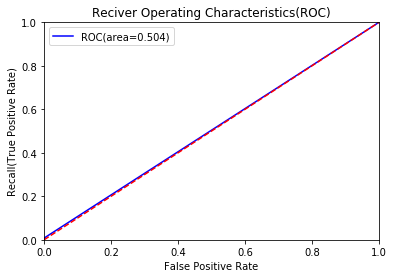
#fitting the model

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

Accuracy of the rf\_model : 0.9850555555555556

cross\_val\_score : 0.9850555555555556

Confusion matrix and plot :

****

Out of all the 3 models , random forest is the best hence we will submit this too the test data .

#final submission

sub\_df=pd.DataFrame({'ID\_code':test\_df['ID\_code'].values})

sub\_df['Random\_Forest\_predict']=rf1.predict

sub\_df.to\_csv('submission.csv',index=False)

sub\_df.head()

**Appendix A**

**# importing libraries**

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,auc

from sklearn.model\_selection import StratifiedKFold

from sklearn import tree

import time

import gc

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

import warnings

warnings.filterwarnings('ignore')

import scikitplot as skplt

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

import os

SEED = 13

np.random.seed(SEED)

from sklearn import metrics

from imblearn.over\_sampling import SMOTE, RandomOverSampler

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

start\_time = time.time() **# loading train data**

train\_df=train=pd.read\_csv("C:/Users/AnushaSanthosh/Desktop/train\_1.csv" , nrows=120000) # the first column 'key' is not useful

print("%s seconds" % (time.time() - start\_time))

start\_time = time.time() **# loading test data**

test\_df=train=pd.read\_csv("C:/Users/AnushaSanthosh/Desktop/test\_1.csv" , nrows=120000) # the first column 'key' is not useful

print("%s seconds" % (time.time() - start\_time))

train\_df.shape, test\_df.shape **# checking the shape**

train.columns **# checking the col names**

train\_df.target.value\_counts() **# knowing the binary data**

train\_df.head(10)

train\_df.isna().sum().sum() **#checking for train missing values**

test\_df.head(10)

test\_df.isna().sum().sum() **# checking for test missing values**

gc.collect();

train\_df.describe() **# chk statistical values in train**

numerical\_features = train\_df.columns[2:]

train\_df.columns[2:]

**# checking the distribution of each variables.**

print('Distributions columns')

plt.figure(figsize=(30, 185))

for i, col in enumerate(numerical\_features):

plt.subplot(50, 4, i + 1)

plt.hist(train\_df[col])

plt.title(col)

gc.collect();

plt.figure(figsize=(20, 8)) **# checkig the distribution for mean**

train\_df[numerical\_features].median().plot('hist');

plt.title('Median Frequency');

plt.figure(figsize=(20, 8)) **# checkig the distribution for SD**

train[numerical\_features].std().plot('hist');

plt.title('Standard Deviation Frequency');

plt.figure(figsize=(20, 8)) **# checkig the skewness frequency**

train[numerical\_features].skew().plot('hist');

plt.title('Skewness Frequency');

plt.figure(figsize=(20, 8)) **# checking kurtosis frequency**

train[numerical\_features].kurt().plot('hist');

plt.title('Kurtosis Frequency');

**# checking the correlation of numerical freatures with all the variables**.

corr = train\_df[numerical\_features].corr()

s = corr.unstack().drop\_duplicates()

so = s.sort\_values(kind="quicksort")

so = so.drop\_duplicates()

print("Top most highly positive correlated features:")

print(so[(so<1) & (so>0.5)].sort\_values(ascending=False))

print()

print("Top most highly megative correlated features:")

print(so[(so < - 0.005)])

train\_df.shape, test\_df.shape

**# checking correlation of numerical features with all the test variables.**

corr = test\_df[numerical\_features].corr()

s = corr.unstack().drop\_duplicates()

so = s.sort\_values(kind="quicksort")

so = so.drop\_duplicates()

print("Top most highly positive correlated features:")

print(so[(so<1) & (so>0.5)].sort\_values(ascending=False))

print()

print("Top most highly megative correlated features:")

print(so[(so < - 0.005)])

**# plotting the correlation of test and train**

**#Correlations in train data**

train\_correlations=train\_df[numerical\_features].corr()

train\_correlations=train\_correlations.values.flatten()

train\_correlations=train\_correlations[train\_correlations!=1]

**#Correlations in test data**

test\_correlations=test\_df[numerical\_features].corr()

test\_correlations=test\_correlations.values.flatten()

test\_correlations=test\_correlations[test\_correlations!=1]

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

**#Distribution plot for correlations in train data**

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

**#Distribution plot for correlations in test data**

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

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

plt.ylabel("Density")

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

plt.legend()

**# splitting the data for ML**

X=train\_df.drop(columns=['ID\_code','target'],axis=1)

test=test\_df.drop(columns=['ID\_code'],axis=1)

Y=train\_df['target']

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)

**# using logistic regression**

**#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)

**#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)

**#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)

**#Random forest classifier**

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

**#fitting the model**

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

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

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

cm1=confusion\_matrix(y\_valid,rf1\_predict)

**#Plot the confusion matrix**

plot\_confusion\_matrix(y\_valid,rf1\_predict,normalize=False,figsize=(15,8))

**#ROC\_AUC score**

roc\_score=roc\_auc\_score(y\_valid,rf1\_predict)

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

**#ROC\_AUC curve**

plt.figure()

false\_positive\_rate,recall,thresholds=roc\_curve(y\_valid,rf1\_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,rf1\_predict)

print(scores)

**#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)

%%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)

**#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))

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)

**#final submission**

sub\_df=pd.DataFrame({'ID\_code':test\_df['ID\_code'].values})

sub\_df['Random\_Forest\_predict']=rf1.predict

sub\_df.to\_csv('submission.csv',index=False)

sub\_df.head()

**Appendix B:**

**Code in R :**

**# claring the environment**

rm(list=ls())

**# installing libraries**

library(tidyverse)

library(caret)

library(Matrix)

library(caTools)

library(randomForest)

library(glmnet)

install.packages("moments")

library(moments)

install.packages("glmnet")

library(glmnet)

install.packages("e1071")

library(e1071)

install.packages("pROC")

library(pROC)

install.packages("DMwR")

library(DMwR)

install.packages("ROSE")

library(ROSE)

library(dplyr)

install.packages("randomForest" , repos = "http://cran.us.r-project.org")

***# setting working directory***

setwd("C:/Users/AnushaSanthosh/Desktop")

**# reading train data**

train\_df<-read.csv("C:/Users/AnushaSanthosh/Desktop/train\_1.csv",nrows=50000)

head(train\_df) # seeing the data

dim(train\_df)

str(train\_df)

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

table(train\_df$target)

**# plotting the distribution of data for eavh variable**

for (var in names(train\_df)[c(3:202)]){

target<-train\_df$target

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

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

print(plot)

}

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

target<-train\_df$target

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

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

print(plot)

}

**# loading test data**

test\_df<-read.csv("C:/Users/AnushaSanthosh/Desktop/test\_1.csv",nrows=50000)

head(test\_df)

dim(test\_df)

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

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

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

ggplot()+

**#Distribution of mean values per row in train data**

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

#Distribution of mean values per row in test data

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

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

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

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

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

ggplot()+

**#Distribution of mean values per column in train data**

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

**#Distribution of mean values per column in test data**

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

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

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

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

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

ggplot()+

**#Distribution of sd values per row in train data**

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

**#Distribution of mean values per row in test data**

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

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

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

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

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

ggplot()+

**#Distribution of sd values per column in train data**

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

**#Distribution of sd values per column in test data**

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

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

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

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

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

ggplot()+

**#Distribution of skewness values per row in train data**

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

**#Distribution of skewness values per column in test data**

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

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

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

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

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

ggplot()+

**#Distribution of skewness values per column in train data**

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

**#Distribution of skewness values per column in test data**

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

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

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

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

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

ggplot()+

**#Distribution of sd values per column in train data**

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

**#Distribution of sd values per column in test data**

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

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

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

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

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

ggplot()+

**#Distribution of sd values per column in train data**

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

#Distribution of sd values per column in test data

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

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

**#Finding the missing values in train data**

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

missing\_val<-sum(missing\_val)

missing\_val

**#Finding the missing values in test data**

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

missing\_val<-sum(missing\_val)

missing\_val

**#Correlations in train data**

#convert factor to int

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

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

train\_correlations

**#Correlations in test data**

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

test\_correlations

**#Split the data using CreateDataPartition**

set.seed(689)

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

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

**#train data**

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

**#validation data**

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

#dimension of train data

dim(train.data)

**#dimension of validation data**

dim(valid.data)

**#target classes in train data**

table(train.data$target)

**#target classes in validation data**

table(valid.data$target)

**#Training dataset**

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

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

**#validation dataset**

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

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

**#test dataset**

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

set.seed(667) # to reproduce results

lr\_model <-glmnet(X\_t,y\_t, family = "binomial")

summary(lr\_model)

set.seed(8909)

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

cv\_lr

**#Minimum lambda**

cv\_lr$lambda.min

**#plot the auc score vs log(lambda)**

plot(cv\_lr)

set.seed(5363)

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

cv\_predict.lr

**#Confusion matrix**

set.seed(689)

**#actual target variable**

target<-valid.data$target

**#convert to factor**

target<-as.factor(target)

**#predicted target variable**

**#convert to factor**

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

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

set.seed(892)

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

roc(response=target,predictor=cv\_predict.lr,auc=TRUE,plot=TRUE)

**#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)

set.seed(462)

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

summary(lr\_rose)

**#Cross validation prediction**

set.seed(473)

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

cv\_rose

**#Minimum lambda**

cv\_rose$lambda.min

#plot the auc score vs log(lambda)

plot(cv\_rose)

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(response=target,predictor=cv\_predict.rose,auc=TRUE,plot=TRUE)

**#Split the training data using simple random sampling**

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

**#train data**

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

**#validation data**

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

**#dimension of train and validation data**

dim(train\_data)

dim(valid\_data)

**#Training the Random forest classifier**

set.seed(2732)

**#convert to int to factor**

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

valid\_data$target<-as.factor(valid\_data$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)

pred.rf<-predict(rf,newdata=valid\_data)

cmu<-confusionMatrix(pred.rf,valid\_data$target)

cmu

pred.rf<-as.numeric(pred.rf)

roc(response=valid\_data$target,predictor=pred.rf,auc=TRUE,plot=TRUE)

**# Final submission**

submission<-data.frame(ID=test\_df , target=pred.rf)