Edwisor

Santander Customer Transaction Prediction

"Identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted."

Submitted by:

Kambhampati Venkata Datta,

23-01-2020

Contents

1. Introduction

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?

1.1 Problem Statement

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

1.2 Data

Our task is to build an classification model to predict the target variable which is binomial 1 or 0 using different classification techniques.

ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7
train_0	0	8.9255	-6.7863	11.9081	5.093	11.4607	-9.2834	5.1187	18.6266
train_1	0	11.5006	-4.1473	13.8588	5.389	12.3622	7.0433	5.6208	16.5338
train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155
train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.925
train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514
train_5	0	11.4763	-2.3182	12.608	8.6264	10.9621	3.5609	4.5322	15.2255
train_6	0	11.8091	-0.0832	9.3494	4.2916	11.1355	-8.0198	6.1961	12.0771
train_7	0	13.558	-7.9881	13.8776	7.5985	8.6543	0.831	5.689	22.3262
train_8	0	16.1071	2.4426	13.9307	5.6327	8.8014	6.163	4.4514	10.1854
train_9	0	12.5088	1.9743	8.896	5.4508	13.6043	-16.2859	6.0637	16.841
train_10	0	5.0702	-0.5447	9.59	4.2987	12.391	-18.8687	6.0382	14.3797

The variable present in this data are

They provided with an anonymized dataset containing numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

2.Methodology

2.1 Pre Processing

The real-world data has many errors and more inconsistent in format. In pre-processing we transform the raw data into understandable format. Real-world data is incomplete and/or lacking in certain behaviors or trends, and it contain many errors. Data preprocessing is a proven method of resolving such Looking the data refers to exploring the data, cleaning data as well as visualizing the data through graphs and plots.This is often called as Exploratory Data Analysis.

2.1.1 Exploratory Data Analysis

In exploring the data we have

- Feature Engineering
- Deleted ID_Code variable as it is nothing but an index.
- Some visualizations on data distribution to see kurtosis and skewness of data

2.1.2 Missing Value Analysis

Missing value analysis is done to check is there any missing value present in given dataset. There missing values can be treated by replacing it by using several imputation techniques like mean, median and KNN imputation for continuous variable and mode for categorical variable. These techniques are used to impute missing value.

In R function(x){sum(is.na(x))} is the function used to check the sum of missing values.

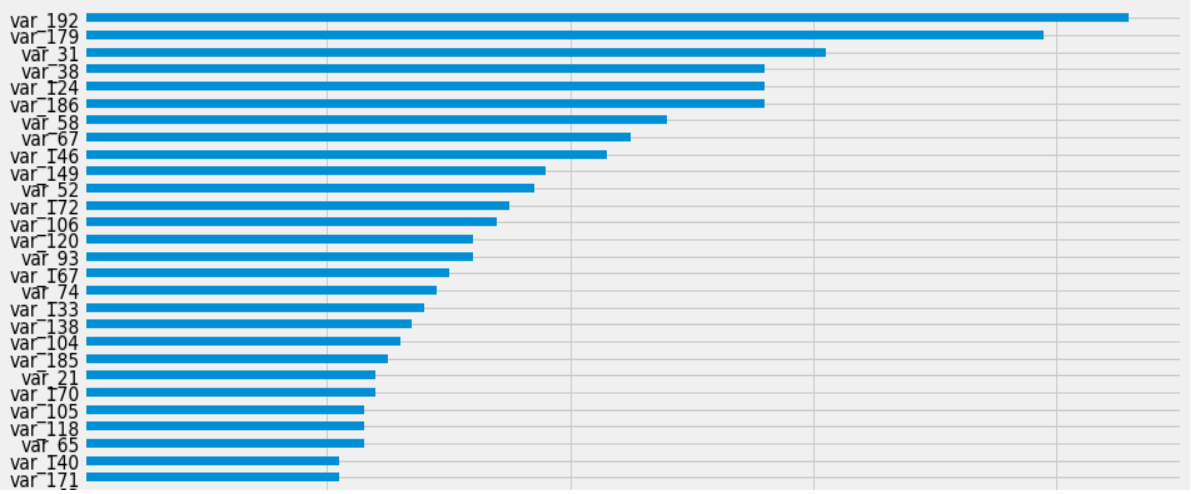
In python df.isnull().sum() is used to detect any missing value

dteday	0
season	0
yr	0
mnth	0
holiday	0
weekday	0
workingday	0
weathersit	0
temp	0
hum	0
windspeed	0
cnt	0

From above we can observe that there is no missing value in given dataset.

2.1.3 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.



#remove these outliers in train and test data

for col in numerical_features:

    train=train.loc[(~chauvenet(train[col].values))]

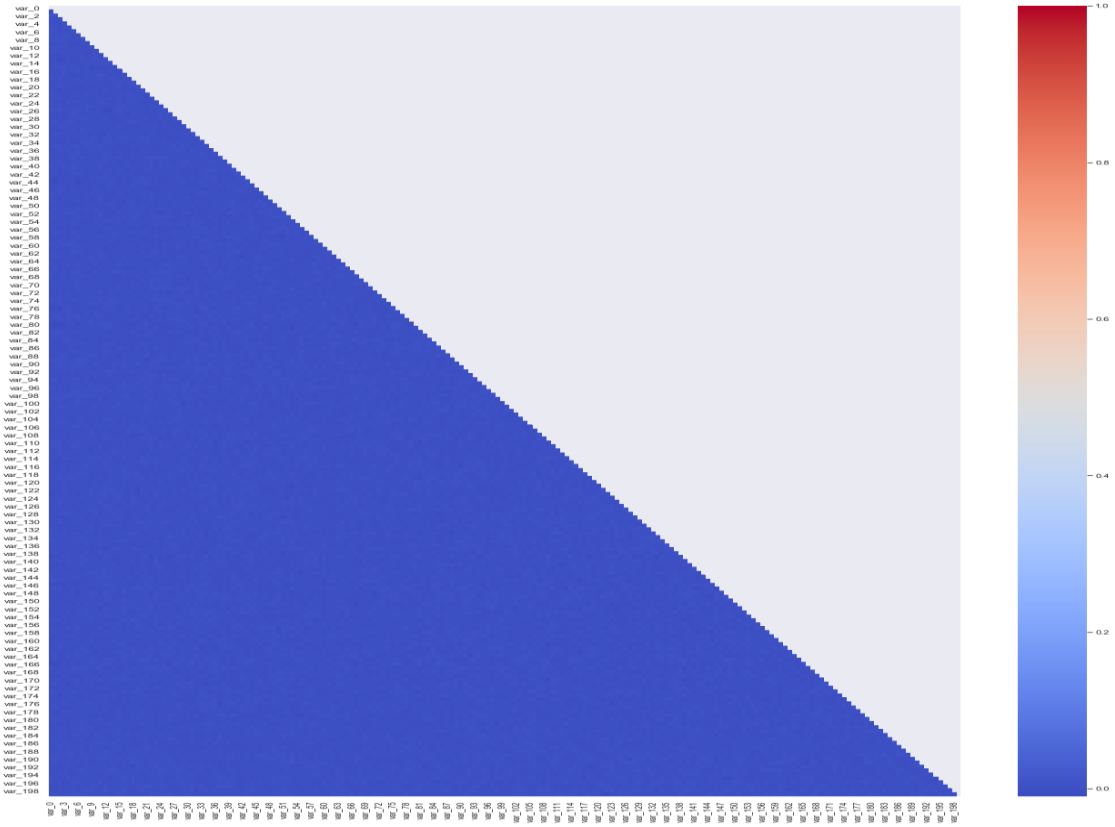
for col in numerical_features:

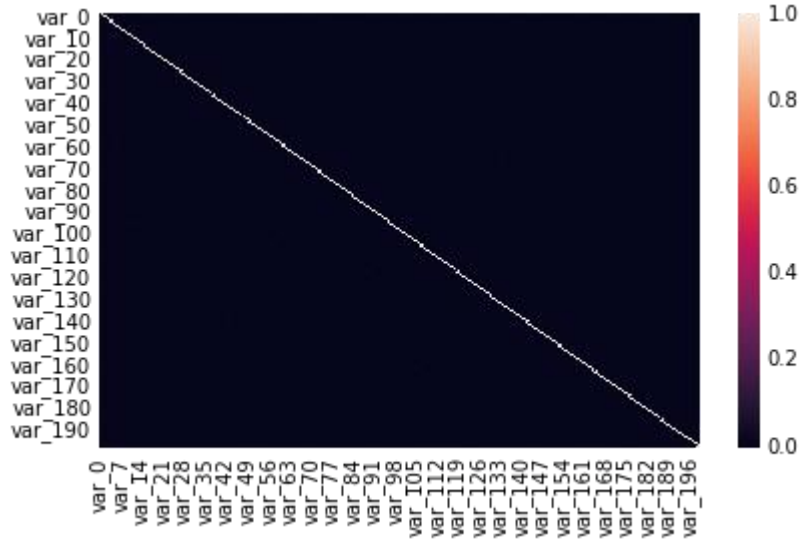
test=test.loc[(~chauvenet(test[col].values))]

To remove outliers I used this technique.

2.1.4 Feature Selection

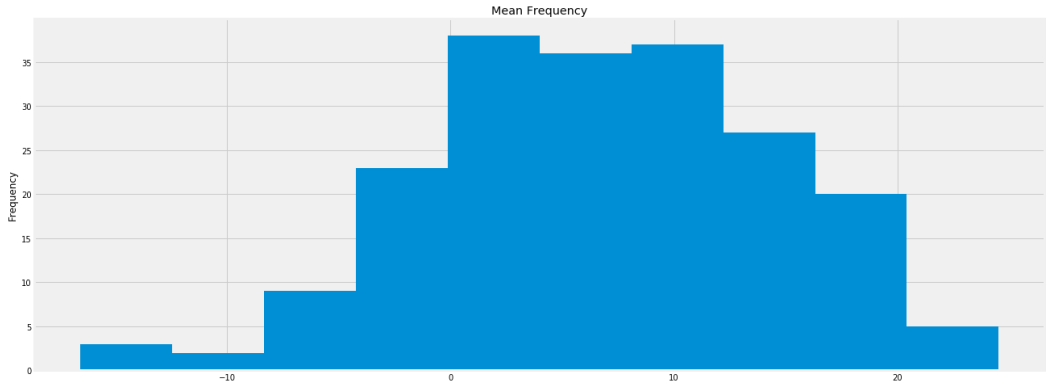
Minimum correlation among variables is -0.009844361358419677
Maximum correlation between variables is 0.009713658349534146
We have 200 features that are mostly uncorrelated between them.



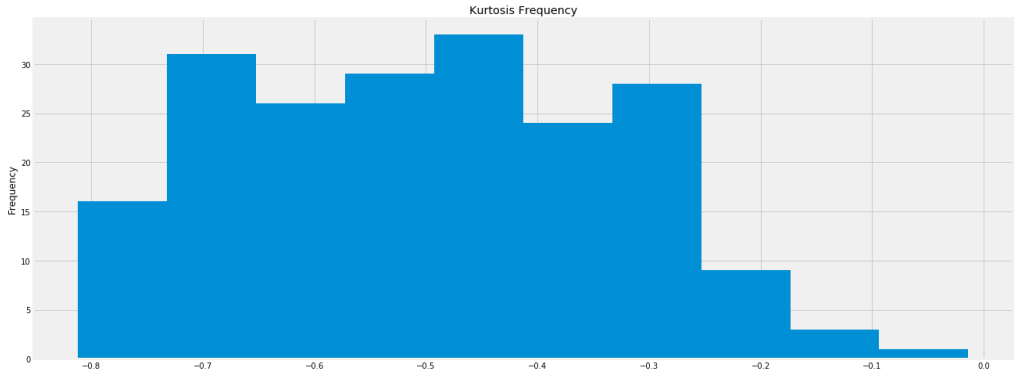


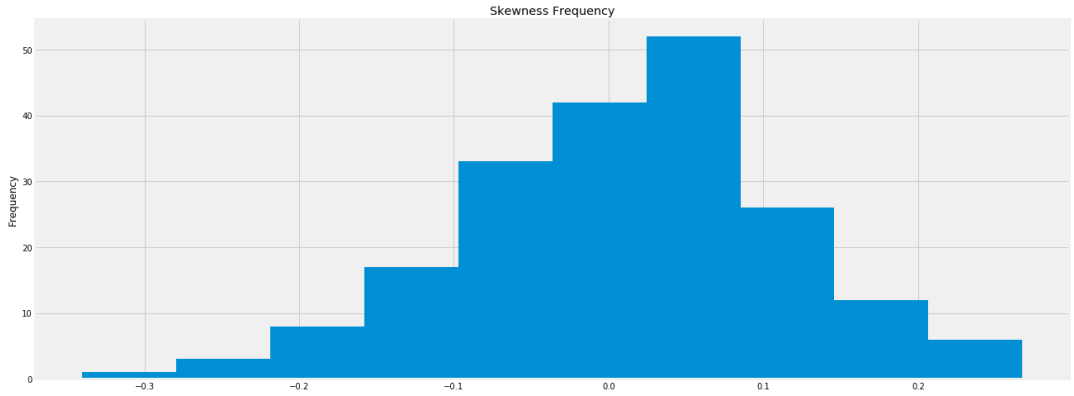
2.1.4 Feature Scaling

Feature scaling includes two functions normalization and standardization. It is done reduce unwanted variation either within or between variables and to bring all of the variables into proportion with one another.
In given dataset all numeric values are already present in normalized form.



From above graph we can interpret that the data is in normalised form.





Most of the distributions show small std. deviations. Both skew and kurtosis graph shoes that each feature distribution is normal one.

2.2 Modeling

2.2.1 Model Selection

The main objective of this case was to predict the target variable based on customer transactions. After applying many models we will evaluate based on some error matrices and select the model which fits the data and predicts more accurately. The three classification models I used are

1. Random Forest classifier
2. Logistic regression
3. LightGBM

2.2.2 Random Forest Classifier

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.

```
# check area under curve
y_pred_prob = model.predict_proba(X_test)[:, 1]
print("AUC:    \t", round(roc_auc_score(y_test, y_pred_prob),2))
```

```
[[44928    48]
 [ 4985    39]]
AUC:              0.65
```

After selecting important features and performing Randomforest Classifier.

```
print("ROC:    \t", round(roc_auc_score(y_test, y_pred_prob),2))
ac=accuracy_score(y_test,y_pred)
print("accuracy:",ac)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    5.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
```

```
[[44589     0]
 [ 4977     0]]
Sensitivity:    0.0
Specificity:    1.0
ROC:             0.78
accuracy: 0.8995884275511439
```

```
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    5.1s finished
```

So the AUC curve generated value 0.78 which is far better than previous RFC without important features. But the concern here was the sensitivity and specificity which shown as it can't able to classify it properly this is due to data imbalance as I used to split the data in random sampling format. It's better to use stratified sampling method to split the data. We can use ROSE method to balance the data by over sampling or under sampling methods.

2.2.3 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.

```
: # print best hyperparameters
  print("Best AUC: ", model.best_score_)
  print("Best hyperparameters: ", model.best_params_)
```

```
Best AUC:  0.8590988473272022
Best hyperparameters:  {'logistic__C': 0.5, 'logistic__penalty': 'l1'}
```

After doing hyper parameter testing we generated adequate AUC score as well as good penalty and logistic lambda value.

```
[[34944  9645]
 [ 1128  3849]]
Sensitivity:    0.77
Specificity:    0.78
ROC:             0.86
Accuracy: 0.7826534317879191
```

The AUC-ROC value is 0.86 which was better than Random Forest Classifier abut accuracy value is not adequate enough.

2.2.3 Light GBM

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.

```
Training until validation scores don't improve for 4000 rounds
[5000]  training's auc: 0.925048        valid_1's auc: 0.894416
[10000] training's auc: 0.940618        valid_1's auc: 0.895758
Early stopping, best iteration is:
[10456] training's auc: 0.941861        valid_1's auc: 0.895927
CV score: 0.90046
Wall time: 6h 17min 38s
```

Tuning LightGBm Model

Parameter Tuning:

Following set of practices **can be used to improve your model efficiency.**

1. **num_leaves:** This is the main parameter to control the complexity of the tree model. Ideally, the value of num_leaves should be less than or equal to 2^(max_depth). Value more than this will result in overfitting.
2. **min_data_in_leaf:** Setting it to a large value can avoid growing too deep a tree, but may cause under-fitting. In practice, setting it to hundreds or thousands is enough for a large dataset.
3. **max_depth:** You also can use max_depth to limit the tree depth explicitly.

```
Accuracy Score for LightGBM is  0.86031
Precision Score for  LightGBM is  0.777291272763459
Recall Score for LightGBM is  0.39970320335687237
f1 Score for LightGBM is  0.527930789767159
```

3. Conclusion

3.1 Model Evaluation and Selection

Random Forest Classifier-

```
print("ROC:    \t", round(roc_auc_score(y_test, y_pred_prob),2))
ac=accuracy_score(y_test,y_pred)
print("accuracy:",ac)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    5.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
```

```
[[44589     0]
 [ 4977     0]]
Sensitivity:    0.0
Specificity:    1.0
ROC:             0.78
accuracy: 0.8995884275511439
```

```
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    5.1s finished
```

Logistic Regression:

```
[[34944  9645]
 [ 1128  3849]]
Sensitivity:    0.77
Specificity:    0.78
ROC:             0.86
Accuracy: 0.7826534317879191
```

LIGHTGBM-

```
Accuracy Score for LightGBM is  0.86031
Precision Score for  LightGBM is  0.777291272763459
Recall Score for LightGBM is  0.39970320335687237
f1 Score for LightGBM is  0.527930789767159
```

From above result you can see that the lightgbm has adequate accuracy and AUC curve value. The random forest generated good accuracy value compared to lightGBM but the thing here is it didn't have viable AUC value. Considering both perception I choosed LightGBM as best model

4. Appendix file

R-code:

```
rm(list=ls(all=T))
setwd("D:/Edwisor/santender project")
getwd()

library(tidyverse)
library(moments)
install.packages("DataExplorer")
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)

train=read.csv("train.csv",header = T)
test=read.csv("test.csv",header = T)
head(train)
```

```
dim(train)
str(train)


require(gridExtra)
#Count of target classes
table(train$target)
#Percenatge counts of target classes
table(train$target)/length(train$target)*100
#Bar plot for count of target classes
plot1<-
ggplot(train,aes(target))+theme_bw()+geom_bar(stat='count',fill='lightgreen')
#Violin with jitter plots for target classes
plot2<-
ggplot(train,aes(x=target,y=1:nrow(train)))+theme_bw()+geom_violin(fill='ligh
tblue')+
  facet_grid(train$target)+geom_jitter(width=0.02)+labs(y='Index')
grid.arrange(plot1,plot2, ncol=2)


#Distribution of train attributes from 3 to 102
for (var in names(train)[c(3:102)]){
  target<-train$target
  plot<-ggplot(train, aes(x=train[[var]],fill=target)) +
    geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}


train_mean<-apply(train[,-c(1,2)],MARGIN=1,FUN=mean)
test_mean<-apply(test[,-c(1)],MARGIN=1,FUN=mean)
ggplot()+
  #Distribution of mean values per row in train data
```

```
  geom_density(data=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=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(train[,-c(1,2)],MARGIN=2,FUN=mean)

test_mean<-apply(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 row in
train and test dataset")


##Missing value analysis

#Finding the missing values in train data

missing_val<-
data.frame(missing_val=apply(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(test,2,function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
missing_val

#Correlations in train data
#convert factor to int
train$target<-as.numeric(train$target)
train_correlations<-cor(train[,c(2:202)])
train_correlations

#Split the training data using simple random sampling
train_index<-sample(1:nrow(train),0.75*nrow(train))
#train data
train_data<-train[train_index,]
#validation data
valid_data<-train[-train_index,]
#dimension of train and validation data
dim(train_data)
dim(valid_data)

#Training the Random forest classifier
set.seed(2732)
#convert to int to factor
train_data$target<-as.factor(train_data$target)
#setting the mtry
mtry<-floor(sqrt(200))
#setting the tunegrid
tuneGrid<-expand.grid(.mtry=mtry)
#fitting the ranndom forest
```

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

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

#We will plot "var_13"
par.var_13 <- partial(rf, pred.var = c("var_13"), chull = TRUE)
plot.var_13 <- autoplot(par.var_13, contour = TRUE)
plot.var_13

#Split the data using CreateDataPartition
set.seed(689)
#train.index<-createDataPartition(train_df$target,p=0.8,list=FALSE)
train.index<-sample(1:nrow(train),0.8*nrow(train))
#train data
train.data<-train[train.index,]
#validation data
valid.data<-train[-train.index,]
#dimension of train data
dim(train.data)
#dimension of validation data
dim(valid.data)
#target classes in train data
table(train.data$target)
#target classes in validation data
table(valid.data$target)
```

```
#Training 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[,-c(1)])

#Logistic regression model
set.seed(667) # to reproduce results
lr_model <-glm(target~.,data=train.data[,-c(1,2)],family="binomial")
summary(lr_model)

#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
cv_lr

#Minimum lambda
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)

#Model performance on validation dataset
set.seed(5363)
cv_predict.lr<-predict(cv_lr,X_v,s = "lambda.min", type = "class")
cv_predict.lr
```

```
#Confusion matrix
set.seed(689)
#actual target variable
target<-valid.data$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)

#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
roc(data=valid.data[,-
c(1,2)],response=target,predictor=cv_predict.lr,auc=TRUE,plot=TRUE)

#Random Oversampling Examples(ROSE)
set.seed(699)
train.rose <- ROSE(target~., data =train.data[,-c(1)],seed=32)$data
#target classes in balanced train data
table(train.rose$target)
valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=42)$data
#target classes in balanced valid data
table(valid.rose$target)

#Logistic regression model
set.seed(462)
```

```
lr_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family = "binomial")
summary(lr_rose)


#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),family = "binomial", type.measure = "class")
cv_rose


#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)


#Model performance on validation dataset
set.seed(442)
cv_predict.rose<-predict(cv_rose,as.matrix(valid.rose),s = "lambda.min", type = "class")
cv_predict.rose


#Confusion matrix
set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.rose<-as.factor(cv_predict.rose)
```

```
#Confusion matrix
confusionMatrix(data=cv_predict.rose,reference=target)

#ROC 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)

#Convert data frame to matrix
set.seed(5432)
X_train<-as.matrix(train.data[,-c(1,2)])
y_train<-as.matrix(train.data$target)
X_valid<-as.matrix(valid.data[,-c(1,2)])
y_valid<-as.matrix(valid.data$target)
test_data<-as.matrix(test_df[,-c(1)])

#training data
lgb.train <- lgb.Dataset(data=X_train, label=y_train)
#Validation data
lgb.valid <- lgb.Dataset(data=X_valid,label=y_valid)

set.seed(653)
lgb.grid = list(objective = "binary",
                metric = "auc",
                boost='gbdt',
                max_depth=-1,
                boost_from_average='false',
```

```
        min_sum_hessian_in_leaf = 12,
        feature_fraction = 0.05,
        bagging_fraction = 0.45,
        bagging_freq = 5,
        learning_rate=0.02,
        tree_learner='serial',
        num_leaves=20,
        num_threads=5,
        min_data_in_bin=150,
        min_gain_to_split = 30,
        min_data_in_leaf = 90,
        verbosity=-1,
        is_unbalance = TRUE)


set.seed(7663)
lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,eval_freq =1000,
                valids=list(val1=lgb.train,val2=lgb.valid),early_stopping_rounds = 5000)


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)


sub_df<-
data.frame(ID_code=test_df$ID_code,lgb_predict_prob=lgbm_pred_prob,lgb_predict=lgbm_pred)
write.csv(sub_df,'submission.CSV',row.names=F)
```

head(sub_df)

5.2 Python code

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy.stats import chi2_contingency
import seaborn as sns
from random import randrange,uniform
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
from pandas import Series
import datetime

bikerental=pd.read_csv("day.csv")
bikerental.head(5)
Target variable  analysis
#descriptive statistics summary
bikerental['cnt'].describe()
#Check whether target variable is normal or not
sns.distplot(bikerental['cnt']);

bikerental['season']=bikerental['season'].astype('category')
bikerental['yr']=bikerental['yr'].astype('category')
bikerental['mnth']=bikerental['mnth'].astype('category')
bikerental['holiday']=bikerental['holiday'].astype('category')
```

```
bikerental['workingday']=bikerental['workingday'].astype('category')
bikerental['weekday']=bikerental['weekday'].astype('category')
bikerental['weathersit']=bikerental['weathersit'].astype('category')
d1=bikerental['dteday'].copy()
for i in range(0,d1.shape[0]):
    d1[i]=datetime.datetime.strptime(d1[i],'%Y-%m-%d').strftime('%d')
bikerental['dteday']=d1
bikerental['dteday']=bikerental['dteday'].astype('category')
bikerental=bikerental.drop(['instant','casual','registered'],axis=1)
missing_value=pd.DataFrame(bikerental.isnull().sum())
missing_value=missing_value.reset_index()
missing_value = missing_value.rename(columns = {'index': 'Variables', 0: 'Missing_percentage'})
missing_value['Missing_percentage']=(missing_value['Missing_percentage']/len(bikerental))*100
#descending order
missing_value = missing_value.sort_values('Missing_percentage', ascending = False).reset_index(drop
= True)

#save output results
missing_value.to_csv("Missing_perc.csv", index = False)

################Outlier Analysis###################

#saving numeric values#
cnames=["temp","atemp","hum","windspeed",]
#ploting boxplotto visualize outliers#
plt.boxplot(bikerental['temp'])

bikerental=bikerental.drop(['atemp'],axis=1)

train, test = train_test_split(bikerental, test_size=0.3)

######c50#######
fit_DT = DecisionTreeRegressor(max_depth=2).fit(train.iloc[:,0:11], train.iloc[:,11])
predictions_DT = fit_DT.predict(test.iloc[:,0:11])
```

```
print(predictions_DT)

#defining MAPE function
def MAPE(y_true, y_pred):
   mape = np.mean(np.abs((y_true - y_pred) / y_true))*100
   return mape

#random forest
RFmodel = RandomForestRegressor(n_estimators = 200).fit(train.iloc[:,0:11], train.iloc[:,11])
RF_Predictions = RFmodel.predict(test.iloc[:,0:11])

####svr
svr=SVR(kernel='poly')
lm=LinearRegression()
svr.fit(train.iloc[:,0:11],train.iloc[:,11])
lm.fit(train.iloc[:,0:11],train.iloc[:,11])

####Linear regression
trainlr, testlr = train_test_split(data_lr, test_size=0.2)
model = sm.OLS(trainlr.iloc[:,63].astype(float), trainlr.iloc[:,0:63].astype(float)).fit()
predictions_LR = model.predict(testlr.iloc[:,0:63])

MAPE(test.iloc[:,11],pred_SVR)
result=pd.DataFrame(test.iloc[:,0:11])
result['pred_cnt'] = (RF_Predictions)

result.to_csv("Random forest output python.csv",index=True)
```

Python code:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory
import os
from scipy.special import erfc
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import gc
# Any results you write to the current directory are saved as output.
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
train.shape, test.shape
train.columns
train.isna().sum().sum()
test.isna().sum().sum()
numerical_features = train.columns[2:]
print('Distributions columns')
plt.figure(figsize=(30, 185))
for i, col in enumerate(numerical_features):
    plt.subplot(50, 4, i + 1)
    plt.hist(train[col])
    plt.title(col)
gc.collect();
print('Distributions columns')
plt.figure(figsize=(30, 185))
for i, col in enumerate(numerical_features):
    plt.subplot(50, 4, i + 1)
    plt.hist(train[train["target"] == 0][col], alpha=0.5, label='0', color='b')
```

```
    plt.hist(train[train["target"] == 1][col], alpha=0.5, label='1', color='r')
    plt.title(col)
gc.collect();
plt.figure(figsize=(20, 8))
train[numerical_features].kurt().plot('hist');
plt.title('Kurtosis Frequency');
#checking outliers using Chauvenet's criterion
def chauvenet(array):
    mean = array.mean()          # Mean of incoming array
    stdv = array.std()           # Standard deviation
    N = len(array)               # Lenght of incoming array
    criterion = 1.0/(2*N)        # Chauvenet's criterion
    d = abs(array-mean)/stdv     # Distance of a value to mean in stdv's
    prob = erfc(d)               # Area normal dist.
    return prob < criterion      # Use boolean array outside this function
#outliers in each variable in train data
train_outliers = dict()
for col in [col for col in numerical_features]:
    train_outliers[col] = train[chauvenet(train[col].values)].shape[0]
train_outliers = pd.Series(train_outliers)

train_outliers.sort_values().plot(figsize=(14, 40), kind='barh').set_xlabel('Number of outliers');
#remove these outliers in train and test data
for col in numerical_features:
    train=train.loc[(~chauvenet(train[col].values))]
for col in numerical_features:
    test=test.loc[(~chauvenet(test[col].values))]
sns.heatmap(train[numerical_features].corr())
# import required libraries
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from imblearn.metrics import sensitivity_specificity_support
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
# divide data into train and test
X = train[numerical_features]
y = train.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 4, stratify = y)
# random forest - the class weight is used to handle class imbalance - it adjusts the cost function
forest = RandomForestClassifier(class_weight={0:0.1, 1: 0.9}, n_jobs = -1)

# hyperparameter space
params = {"criterion": ['gini', 'entropy'], "max_features": ['auto', 0.4]}

# create 5 folds
folds = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 4)

# create gridsearch object
model = GridSearchCV(estimator=forest, cv=folds, param_grid=params, scoring='roc_auc', n_jobs=-1, verbose=1)
```

```
# predict target on test data
y_pred = model.predict(X_test)

# create confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# check area under curve
y_pred_prob = model.predict_proba(X_test)[:, 1]
print("AUC:    \t", round(roc_auc_score(y_test, y_pred_prob),2))
num_folds = 15
folds = StratifiedKFold(n_splits=num_folds, shuffle=False, random_state=2319)
oof = np.zeros(len(train))
predictions = np.zeros(len(test))
features = [c for c in train.columns if c not in ['ID_code', 'target']]
%%time
print('Starting cross-validation:')
for fold_, (trn_idx, val_idx) in enumerate(folds.split(train.values, target.values)):
    print("Fold idx:{}".format(fold_ + 1))
    trn_data = lgb.Dataset(train.iloc[trn_idx][features], label=target.iloc[trn_idx])
    val_data = lgb.Dataset(train.iloc[val_idx][features], label=target.iloc[val_idx])
    clf = lgb.train(param, trn_data, 1000000, valid_sets = [trn_data, val_data], verbose_eval=5000, early_stopping_rounds = 4000)
    oof[val_idx] = clf.predict(train.iloc[val_idx][features], num_iteration=clf.best_iteration)
    predictions += clf.predict(test[features], num_iteration=clf.best_iteration) / folds.n_splits
print("CV score: {:<8.5f}".format(roc_auc_score(target, oof)))
lgb_acu=accuracy_score(lgb_pred2,target)
lgb_pre=precision_score(lgb_pred2, target)
lgb_rec=recall_score(lgb_pred2,target)
lgb_f1=f1_score(lgb_pred2, target)

print("Accuracy Score for LightGBM is ",lgb_acu)
```

```
print("Precision Score for  LightGBM is ",lgb_pre)
print("Recall Score for LightGBM is ",lgb_rec)
print("f1 Score for LightGBM is ",lgb_f1)
print('Writing results to a text file')
sub = pd.DataFrame({"ID_code": test.ID_code.values})
sub["target"] = predictions
sub.to_csv('lgbm_submission.csv', index=False)
```