Santander Customer Transaction Prediction

Akshay Prabhu

14/11/2019

Table of Contents

1. Background
2. Problem statement
3. Data details
4. Attributes
5. Metrics
6. Exploratory Data Analysis
7. Data modeling
8. Model evaluation
9. Conclusion

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 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?

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.

Data -

1) test.csv
2) train.csv

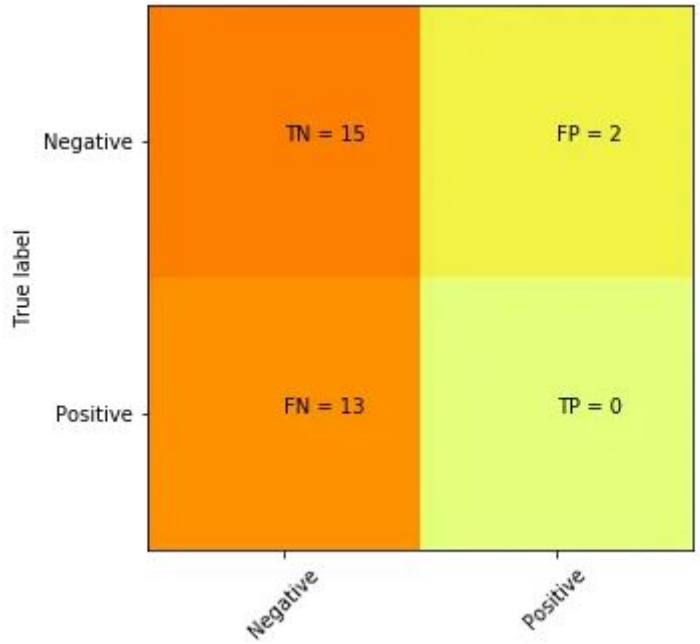
Attributes -

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 valueof target column in the test set.

Metrics -

This is a classification problem and we need to understand confusion matrix forgetting evaluation metrics. It is a performance measurement for machine learning classification problem where output can be two or

more classes. It is a table with 4 different combinations of predicted and actual values. It is extremely useful for measuring Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC Curve. The accuracy metrics we are going to consider are AUC,Precision & Recall.

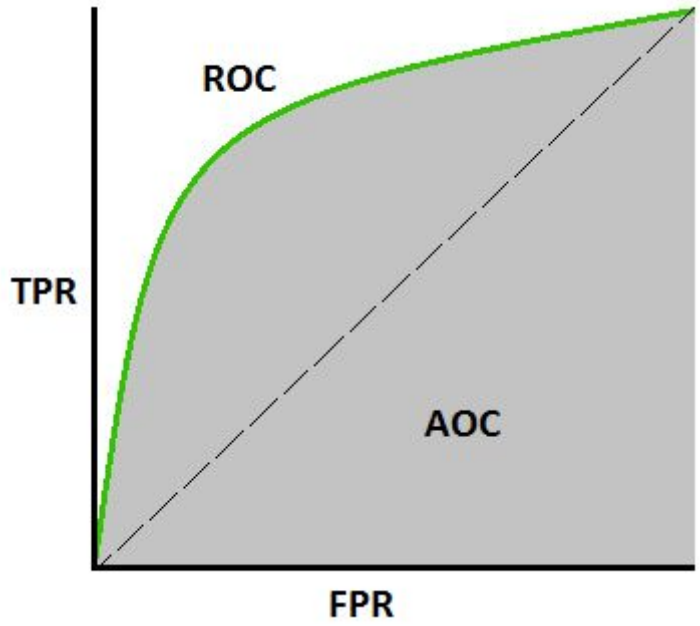


True Positive: You predicted positive and it's true.
True Negative: You predicted negative and it's true.
False Positive: (Type 1 Error) You predicted positive and it's false.
False Negative: (Type 2 Error) You predicted negative and it's false.

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.
Recall = TP/TP+FP

Precision- Out of all the classes,how much we predicted correctly. It should be high as possible.
Precision = TP/TP+FN

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. The ROC curve is plotted with TPR(True Positive Rate) against the FPR(False Positive Rate) where TPR is on y-axis and FPR is on the x-axis.



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.

Exploratory Data Analysis -

Check for missing values

missing_val_train=pd.DataFrame(train_df.isnull().sum())

missing_val_test=pd.DataFrame(test_df.isnull().sum())

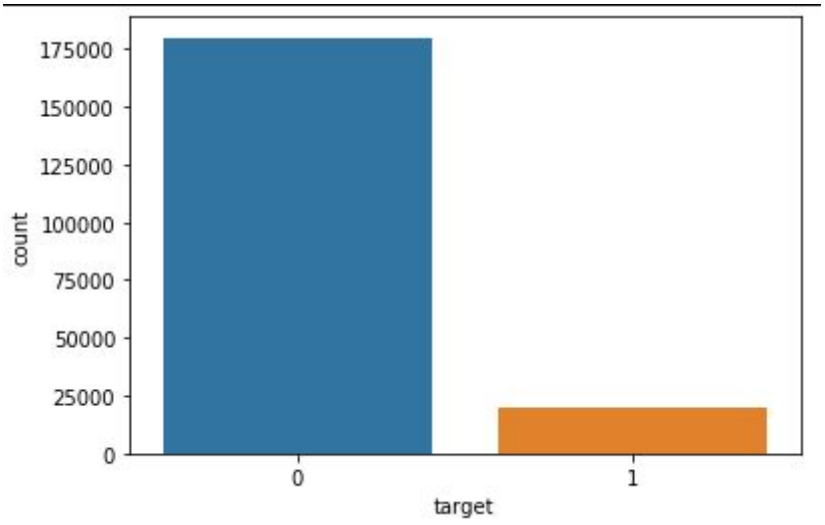
No missing values.

Shape of data

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

Check Balance of target column

sns.countplot(train_df['target'])



This is an imbalanced Dataset.

Outlier Removal

An Outlier is a rare chance of occurrence within a given data set. In Data Science, an Outlier is an observation point that is distant from other observations. An Outlier may be due to variability in the measurement or it may indicate experimental error. We have to remove outliers as it can cause deviation in our analysis.

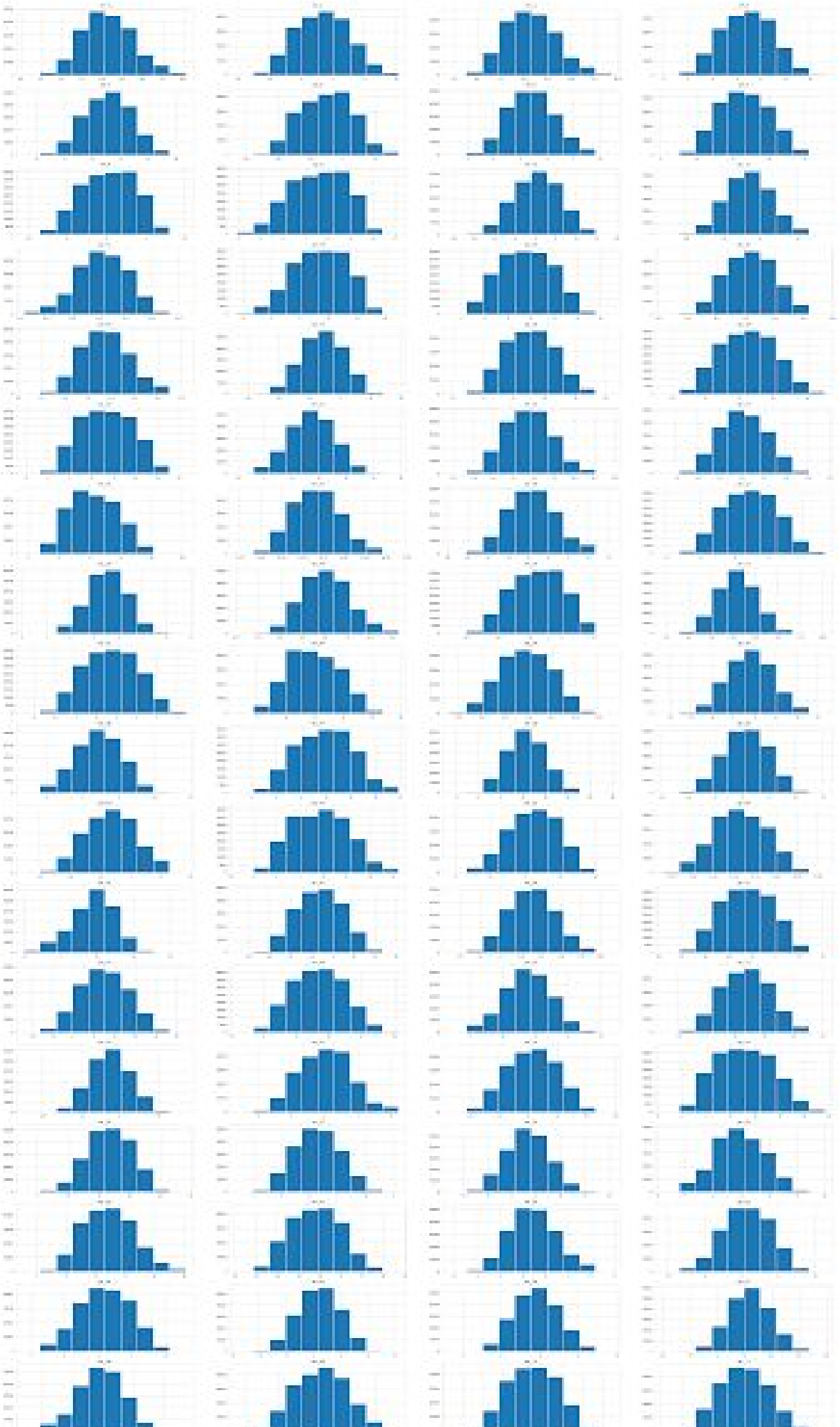
```
features=train_df.iloc[:,2:]
for i in features:
    q75,q25 = np.percentile(features.loc[:,i],[75,25])
    iqr = q75-q25

    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)

    train_df = train_df.drop(train_df[train_df.loc[:,i]<min].index)
    train_df = train_df.drop(train_df[train_df.loc[:,i]>max].index)
```

Distribution of numerical features

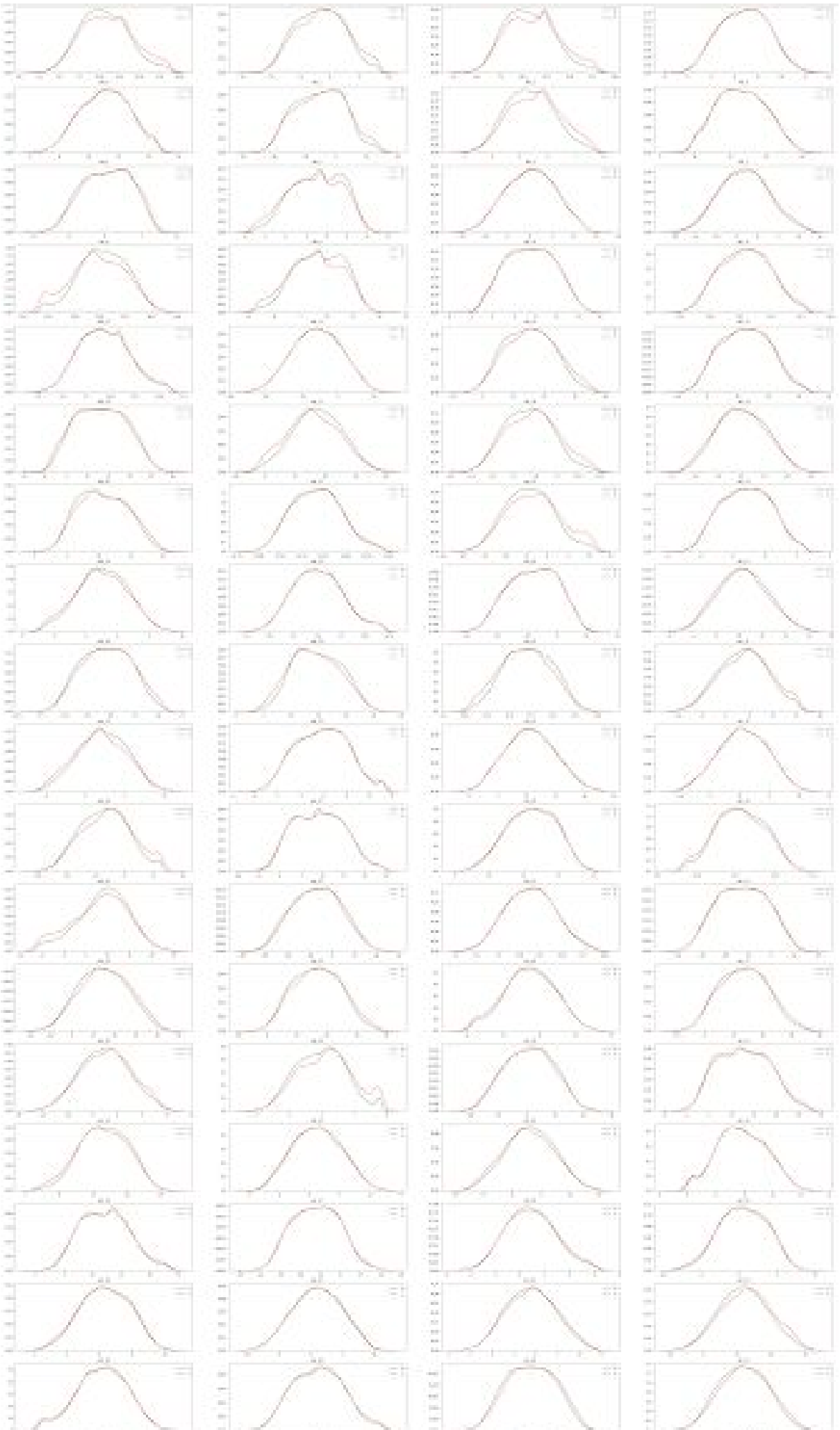
```
plt.figure(figsize=(40,200))
for i,j in enumerate(features):
    plt.subplot(50,4,i+1)
    plt.hist(train_df[j])
    plt.title(j)
```



Features are normally distributed.

Distribution of all numerical features per each class.

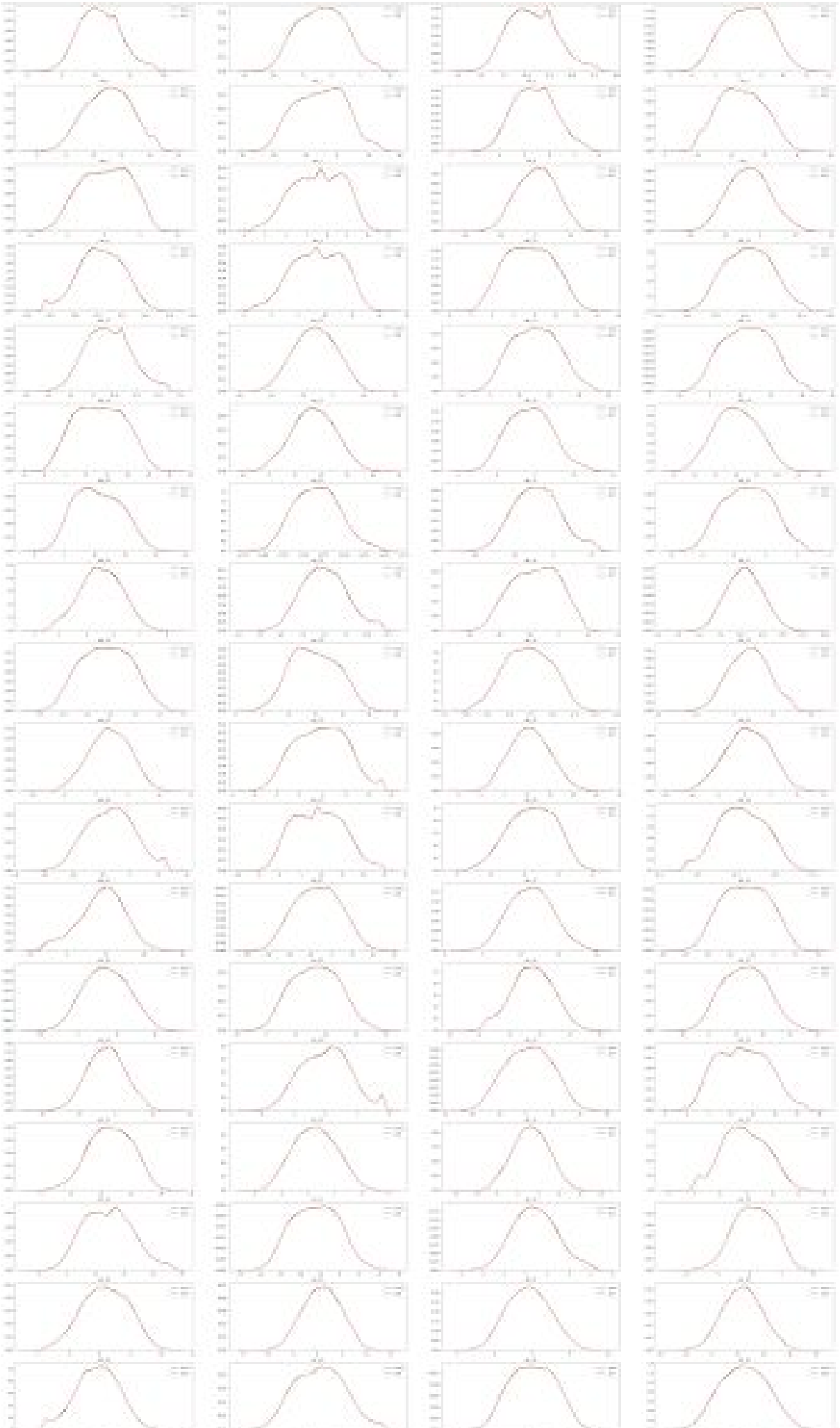
```
plt.figure(figsize=(40,200))
for i,j in enumerate(features):
        plt.subplot(50,4,i+1)
sns.distplot(train_df[train_df['target']==0][j],hist=False,label='0',color='green')
sns.distplot(train_df[train_df['target']==1][j],hist=False,label='1',color='red')
```



We can observe that there is a considerable number of features with significant different distribution for the two target values.
For example, var_0, var_1, var_2, var_5, var_9, var_13, var_106, var_109, var_139 and many others.
Also some features, like var_2, var_13, var_26, var_55, var_175, var_184, var_196 shows a distribution that resembles to a bivariate distribution

Distribution of features with respect to train and test data:

```
plt.figure(figsize=(40,200))
for i,j in enumerate(features):
    plt.subplot(50,4,i+1)
    sns.distplot(train_df[j],hist=False,label='train',color='green')
    sns.distplot(test_df[j],hist=False,label='test',color='red')
```

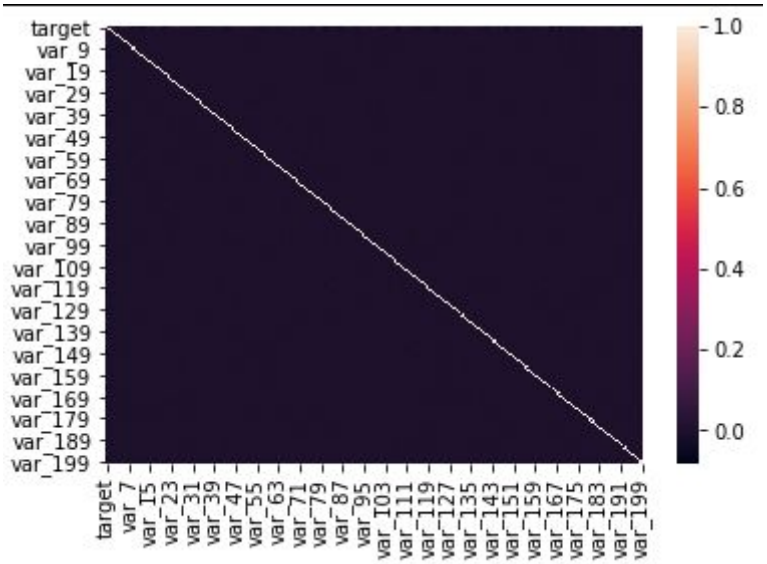


The train and test are well balanced with respect to distribution of the numeric variables.

Correlation analysis

Correlation is a statistical term which in common usage refers to how close two variables are to having a linear relationship with each other. Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two independent features have high correlation, we can drop one of the two features.

corr_df=train_df.corr()
sns.heatmap(corr_df)



None of the features have correlation between them.
Hence we have to go with all 200 variables.

Splitting data into train and test:

It is important to split data into train data and test data for model preparation.

X = train_df.iloc[:,2:]
y = train_df[["target"]]
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=0)

Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step.

Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly without normalization. For example, many classifiers calculate the distance between two points by the Euclidean distance. If one of the features has a broad range of values, the distance will be governed by this particular feature. Therefore, the range of all features should be normalized so that each feature contributes approximately proportionately to the final distance.

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Dealing with imbalanced data

Random Over Sampling: Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample. Unlike under sampling this method leads to no information loss. It also outperforms under sampling.

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
x_train_over, y_train_over = ros.fit_resample(x_train, y_train)
x_test_over, y_test_over = ros.fit_resample(x_test, y_test)
```

Data Modeling-

Logistic regression

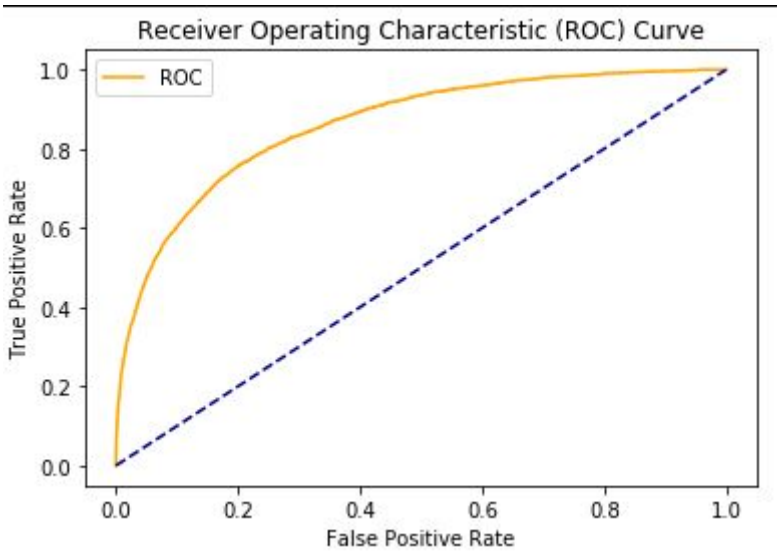
This is the classification problem. We can use logistic regression for this problem. Logistic Regression is used when the dependent variable(target) is categorical. It has a bias towards classes which have large number of instances.It tends to only predict the majority class data. The features of the 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.

```
from sklearn.linear_model import LogisticRegression
lrc = LogisticRegression(random_state=42)
lrc.fit(x_train_over, y_train_over)
y_pred_prob = lrc.predict_proba(x_test_over)
```

AUC score for logistic regression:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test_over, y_pred_pos)
print('AUC for LR: %.2f' % auc)
fpr, tpr, thresholds = roc_curve(y_test_over, y_pred_pos)
```

AUC for LR: 0.86



To calculate precision and recall we need to convert probability into target class

y_pred_class = lrc.predict(x_test_over)

#confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_over, y_pred_class)
print(cm)
TN,FP,FN,TP = confusion_matrix(y_test_over, y_pred_class).ravel()

#precision and recall
*lr_precision = TP*100/float(TP + FP)*
*lr_recall = TP*100/float(TP+FN)*
print("precision and recall for LR: ",lr_precision, lr_recall)
precision: 77.94620300193193
Recall: 77.18261893249806

Random Forest

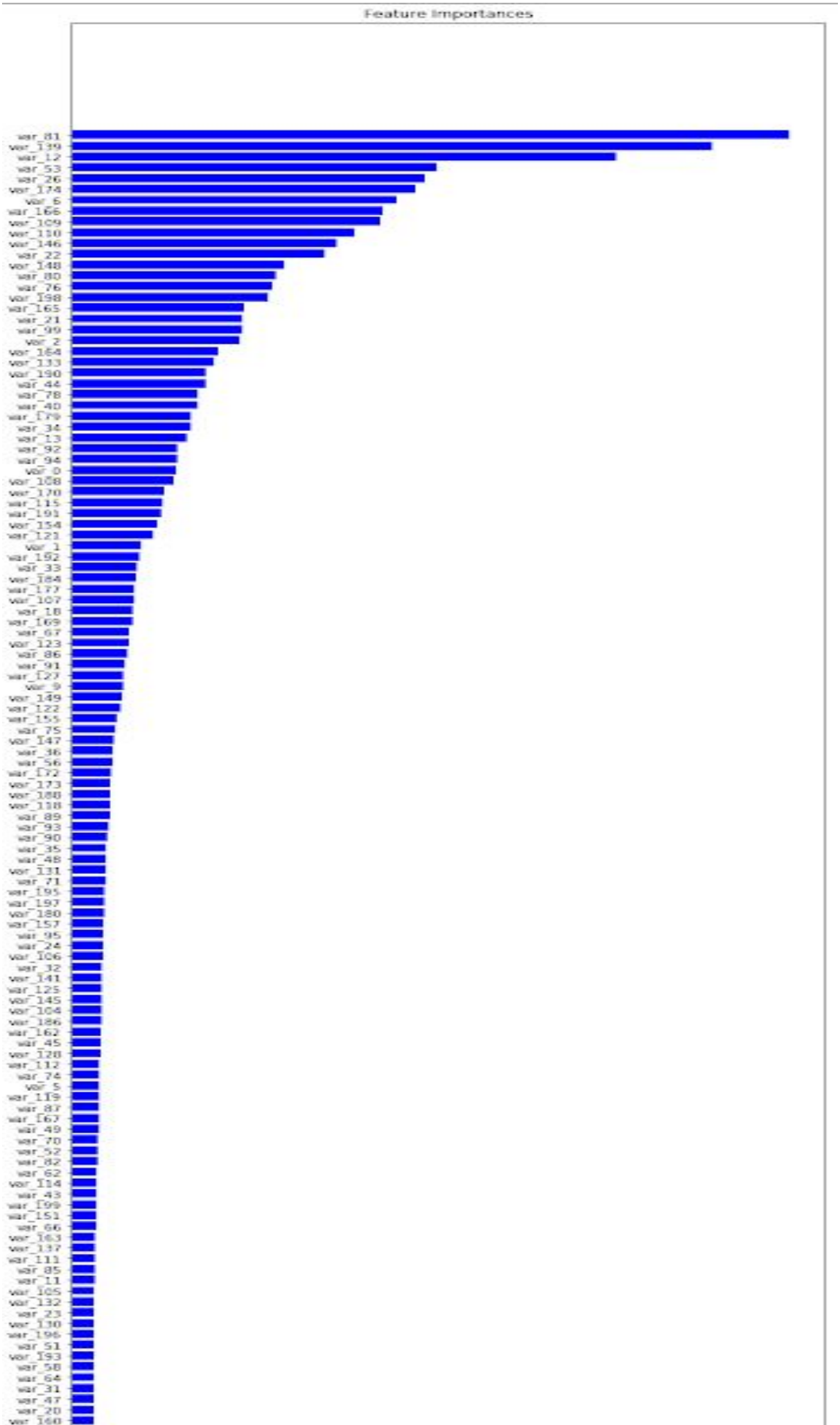
A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap Aggregation, commonly known as bagging.

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(max_depth=10, random_state=0, n_estimators=100)
rfc.fit(x_train_over, y_train_over)
y_rfc_prob = rfc.predict_proba(x_test_over)

Lets plot the important features directed by random forest algorithm

imp_col=train_df.columns[2:]
*importances = rfc.feature_importances_*
indices = np.argsort(importances)
plt.figure(figsize=(10,50))

```
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), imp_col[indices])
plt.xlabel('Relative Importance')
```

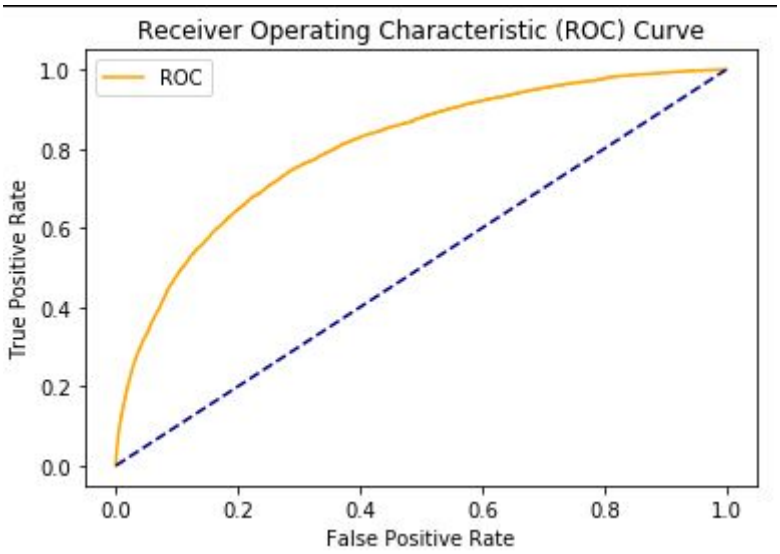

Feature Importances
var_81
var_139
var_12
var_53
var_26
var_174
var_6
var_166
var_109
var_110
var_146
var_22
var_148
var_80
var_76
var_198
var_165
var_21
var_99
var_2
var_184
var_133
var_190
var_44
var_78
var_40
var_179
var_34
var_13
var_92
var_94
var_0
var_108
var_170
var_115
var_191
var_154
var_121
var_1
var_192
var_33
var_184
var_177
var_107
var_18
var_169
var_67
var_123
var_86
var_91
var_127
var_9
var_149
var_122
var_155
var_75
var_147
var_36
var_56
var_172
var_173
var_188
var_118
var_89
var_93
var_90
var_35
var_48
var_131
var_71
var_195
var_197
var_180
var_157
var_95
var_24
var_106
var_32
var_141
var_125
var_145
var_104
var_186
var_162
var_45
var_128
var_112
var_74
var_5
var_119
var_87
var_167
var_49
var_70
var_52
var_82
var_62
var_114
var_43
var_199
var_151
var_66
var_163
var_137
var_111
var_85
var_11
var_105
var_132
var_23
var_130
var_196
var_51
var_193
var_58
var_64
var_31
var_47
var_20
var_160

AUC score for random forest:

auc = roc_auc_score(y_test_over, y_nb_pos)

print('AUC for RF: %.2f' % auc)

AUC for RF: 0.80



#to calculate precision and recall we need to convert probability into target class

y_rfc_class = rfc.predict(x_test_over)

#confusion matrix

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test_over, y_rfc_class)

print(cm)

TN,FP,FN,TP = confusion_matrix(y_test_over, y_rfc_class).ravel()

#precision and recall

*lr_precision = TP*100/float(TP + FP)*

*lr_recall = TP*100/float(TP+FN)*

print("precision and recall for RF: ",lr_precision, lr_recall)

Precision for RF: 81.00549731526463

Recall for RF: 53.28049780319115

Naive Bayes

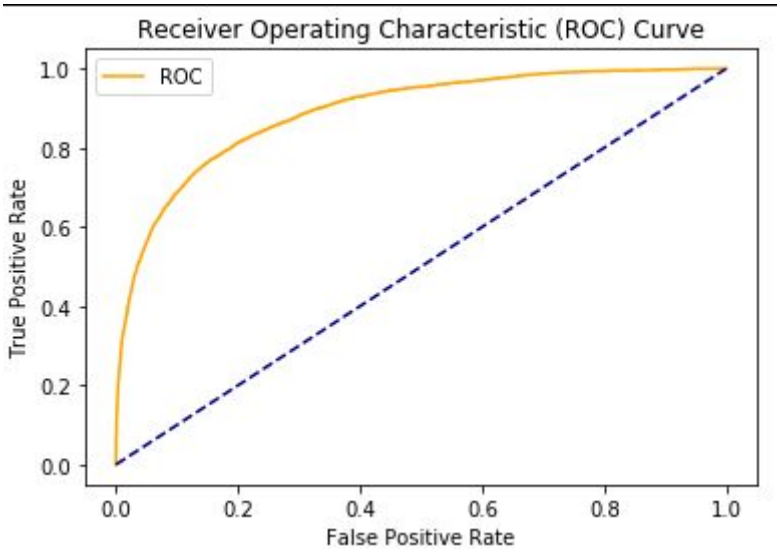
Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. In machine learning, naïve Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. They are among the simplest Bayesian network models.

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train_over, y_train_over)
y_nb_prob = nb.predict_proba(x_test_over)
```

AUC score for Naive bayes:

```
auc = roc_auc_score(y_test_over, y_nb_pos)
print('AUC for NB: %.2f' % auc)
```

AUC for NB: 0.89




```
#to calculate precision and recall we need to convert probability into target class
y_nb_class = nb.predict(x_test_over)
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_over, y_nb_class)
print(cm)
TN,FP,FN,TP = confusion_matrix(y_test_over, y_nb_class).ravel()

#precision and recall
nb_precision = TP*100/float(TP + FP)
nb_recall = TP*100/float(TP+FN)
print("precision and recall for NB: ",nb_precision, nb_recall)
```

Precision for NB: 81.09899508231773
Recall for NB: 79.73680338035275

Model Evaluation-

We are going to evaluate our models by selecting the best model on the basis of AUC score, Recall and Precision.

	AUC	Precision	Recall
Logistic Regression	0.86	77.94	77.18
Random Forest	0.80	81.00	53.28
Naive Bayes	0.89	81.09	79.73

We can see that, from the above scores, Naive Bayes has performed well. Hence we select Naive Bayes algorithm to predict the test data.

```
#removing outliers from test data
features_test=test_df.iloc[:,1:]

for i in features_test:
        q75,q25 = np.percentile(features_test.loc[:,i],[75,25])
```

```
    iqr = q75-q25

    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)

    test_df = test_df.drop(test_df[test_df.loc[:,i]<min].index)
    test_df = test_df.drop(test_df[test_df.loc[:,i]>max].index)


#scale
test_scale = sc.transform(test_df.drop(['ID_code'],axis=1))

#impute naive bayes model into the test data as it has the highest AUC,precision,recall score
test_model=nb.predict(test_scale)
test_df["predicted_class"]=test_model
```

We create a new csv file ‘predicted_python.csv’ which contains the test data with the predicted class.

```
test_df.to_csv("predicted_python.csv", index=False)
```

Conclusion-

This was a classification problem on a typically unbalanced dataset with no missing values. Predictor variables are anonymous and numeric and target variable is categorical. By visualising descriptive features we finally got to know that these variables are not correlated among themselves. After that we treated imbalanced dataset using over sampling, applied feature scaling and built different models and chose Naive Bayes as our final model. For Naive Bayes we got an AUC score of 0.89, precision of 89.09 and recall of 79.73.

R code for the project:

```
install.packages('caret',repos = 'http://cran.rstudio.com/')
library(caret)
library(e1071)
library(ROSE)
library(dplyr)
library(DMwR)
library(tidyr)
library(caTools)
library(Hmisc)
library(Metrics)
library(OutlierDetection)
library(randomForest)
library(caret)
library(ggplot2)

setwd("C:/Users/Ak/Desktop/santander")

train_df=read.csv("train.csv", stringsAsFactors = FALSE, header = TRUE)
test_df=read.csv("test.csv", stringsAsFactors = FALSE, header = TRUE)
train_df$target = as.factor(train_df$target)

num_features = subset(train_df, select = -c(1,2))

#outlier removal
outliers=OutlierDetection(num_features, k = 0.05 * nrow(variable1), cutoff = 0.95,
                Method = "euclidean", rnames = FALSE, depth = FALSE,
                dense = FALSE, distance = FALSE, dispersion = FALSE)[2]

for(i in outliers){
  train_df = train_df[-c(i),]
}
```

```
train_df = subset(train_df, select = -c(1))

#correlation analysis
corr_df=cor(num_features)
heatmap(corr_df, scale = 'row')
#no variables are correlated with each other

#split into train and test
set.seed(123)
data_model=sample.split(train_df, SplitRatio = 0.7)
data_train=subset(train_df, data_model==TRUE)
data_test=subset(train_df, data_model==FALSE)

prop.table(table(train_df$target))
#we can see that the target class is highly imbalanced
#class '0' ~91% and class '1'~9%

#feature scaling
#normality check
par(mfrow=c(3,3))
colnames<-dimnames(num_features)[[2]]
for (i in 1:200) {
  hist(num_features[,i],main = colnames[i],probability = TRUE)
}
#data is normally distributed
# feature scaling
data_train[-1]=scale(data_train[-1])
data_test[-1]=scale(data_test[-1])

table(train_df$target)
#we can see that the data is imbalanced
#balanced sampling to handle imbalanced data
data_train_sampled = ROSE(target ~ ., data = data_train, seed =
1)$data
```

```
table(data_train_sampled$target)
data_test_sampled = ROSE(target ~ ., data = data_test, seed = 1)$data
table(data_test_sampled$target)

#logistic regression
logit_model = glm(target~.,data = data_train_sampled, family = 'binomial')
logit_pred = predict(logit_model, newdata = data_test_sampled, type = 'response')
logit_pred=ifelse(logit_pred>0.5,1,0)

#Auc precision recall for logistic regression
roc.curve(data_test_sampled$target, logit_pred)
conf_matrix=confusionMatrix(factor(round(logit_pred)),factor(data_test_over$target))
recall(factor(logit_pred),factor(data_test_sampled$target))
precision(factor(logit_pred),factor(data_test_sampled$target))

#random forest
rf_model=randomForest(target ~ .,data = data_train_sampled, ntree=100)
rf_pred=predict(rf_model, data_test_sampled)

#auc recall precision for random forest
roc.curve(data_test_sampled$target, rf_pred)
conf_matrix=confusionMatrix(factor(round(rf_pred)),factor(data_test_sampled$target))
recall(factor(rf_pred),factor(data_test_sampled$target))
precision(factor(rf_pred),factor(data_test_sampled$target))


#naive bayes
nm_model = naiveBayes(target~., data=data_train_sampled)
nb_pred = predict(nm_model, data_test_sampled[,2:201], type = 'class')
```

```
#auc recall precision for naive bayes
roc.curve(data_test_sampled$target, nb_pred)
conf_matrix=confusionMatrix(factor(round(nb_pred)),factor(data_test_sampled$target))
recall(factor(nb_pred),factor(data_test_sampled$target))
precision(factor(nb_pred),factor(data_test_sampled$target))

#hence we conclude that the naive bayes gives us the highest auc precision and recall score
```