PROJECT REPORT

ON

Santander Customer Transaction Prediction

By : Rahul Prasad Sah

Submitted to : edwisor

Contents

1. Introduction:

1.1 Problem Statement
1.2 Data

2. Methodology:

2.1 Pre-Processing

2.1.1 Exploratory Data Analysis
2.1.2 Missing Values analysis
2.1.3 Outliers Analysis
2.1.4 Feature Selection
2.1.5 Feature Scaling

2.2 Modelling

2.2.1 Logistic Regression
2.2.2 Naïve Bayes
2.2.3 Random Forest

3. Conclusion:

3.1Model Evaluation

3.1.1 Confusion Matrix
3.1.2 Accuracy
3.1.3 Recall
3.1.4 Precision

3.2Model Selection

Chapter-1

INTRODUCTION

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.

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

The task is to develop a classification model that will predict which customers will make a specific transaction in the future, irrespective of the amount of money transacted. Here’s the data sets.

We have two data sets: -

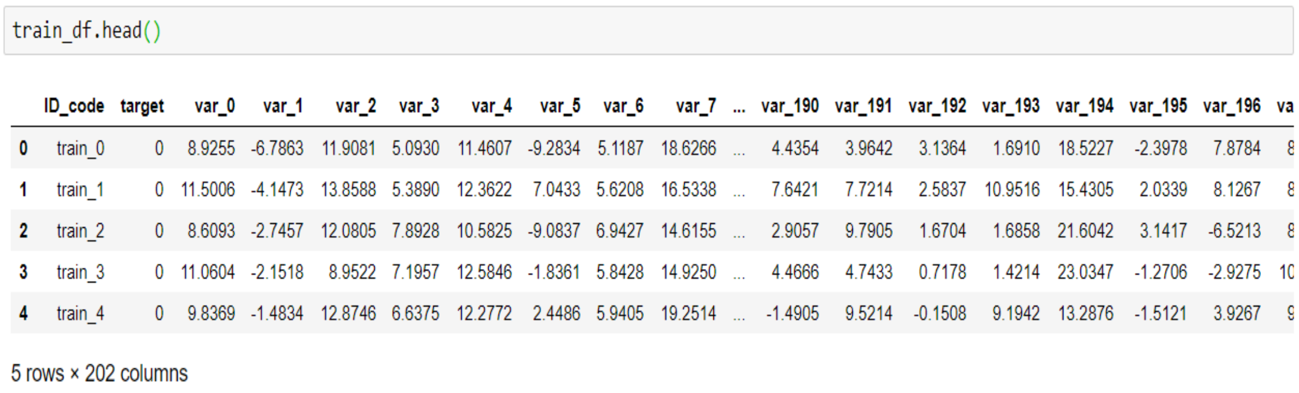
1)Train

2)Test

Number of attributes:

We have been 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.

Here's how training data looks like:

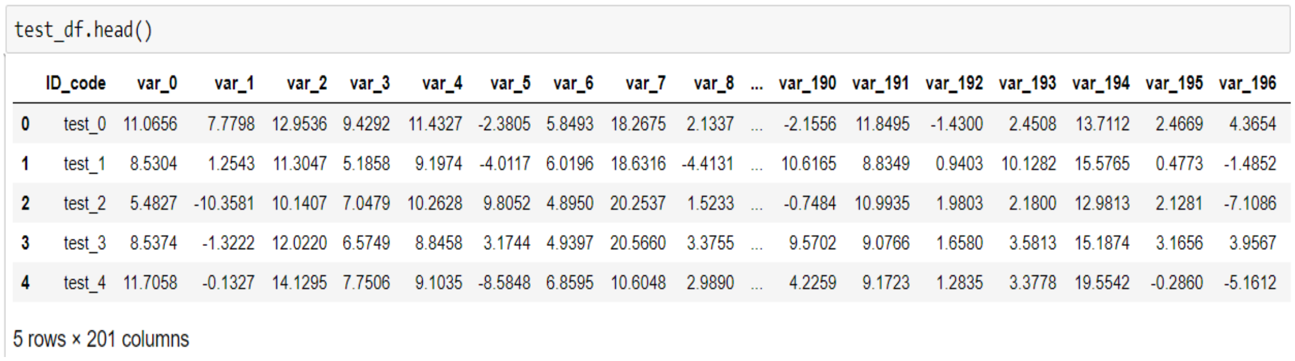


```
train_df.head()
```

	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	va
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
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
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
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
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

5 rows × 202 columns

& test data:



```
test_df.head()
```

	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
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
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
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
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
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

5 rows × 201 columns

Both the data sets have 200000 observations with 202 & 201 columns respectively.

Train contains:

- **ID_code** (string);
- **target**;
- **200** numerical variables, named from **var_0** to **var_199**;

Test contains:

- **ID_code** (string);
- **200** numerical variables, named from **var_0** to **var_199**;

Chapter-2

METHODOLOGY

2.1 Pre-Processing

Any classification model requires that we look at the data before we start modelling. 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**.

2.1.1 Exploratory Data Analysis:

Organizing, Plotting and Summarizing the data.

By visualising the data we can know how the data is distributed and then we can detect the pattern in the data. According to the data, as we have a target variable in the data set lets see the distribution of data in target variables.

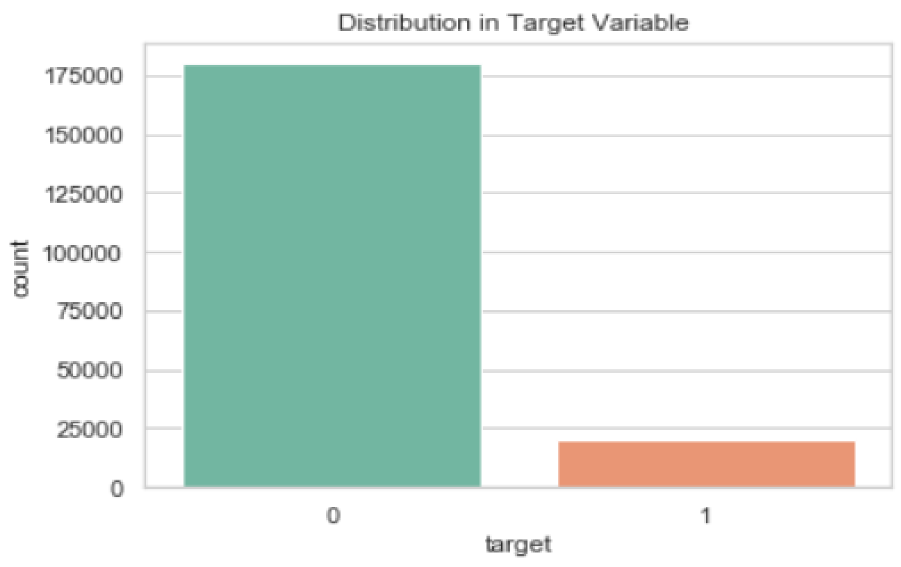


Fig. 2.1

There are 10.049% 1's in target variable & 89.951% 0's in target variable. With t his it is quite clear that we have a imbalanced data set.
As we have hundreds of variables so lets check the distribution of the data in th ose variables.

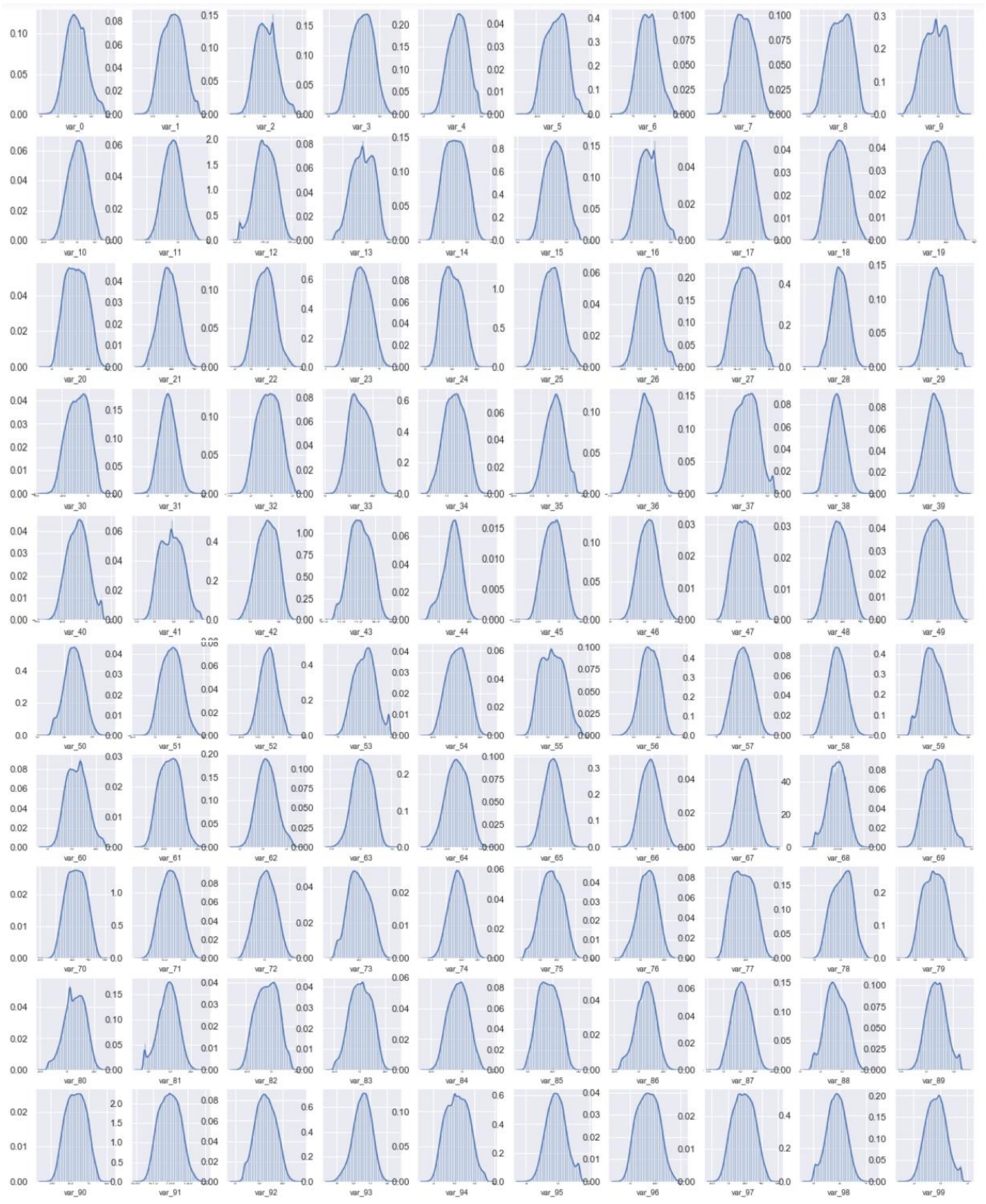
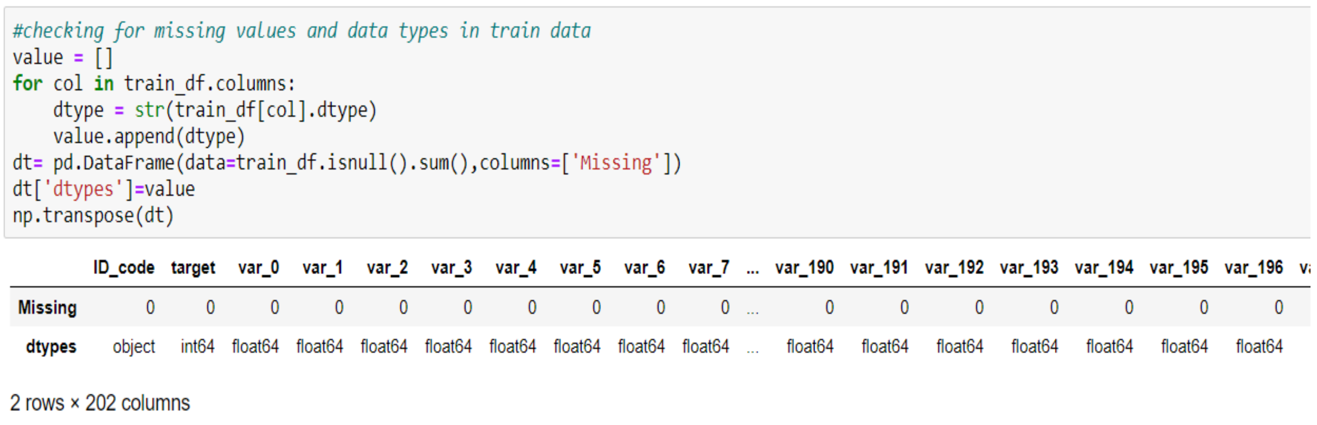


Fig. 2.2

from the above plots we can see that the data is well distributed though there were some exceptions here and there.

2.1.2 Missing Values Analysis:

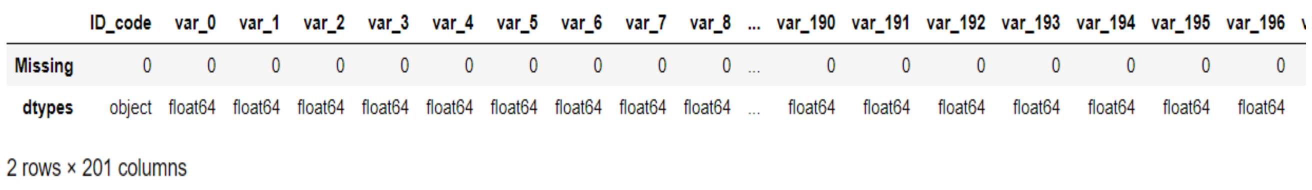
In classification modelling we have to look into data weather the data is clean, does it have any missing values. If it does have any missing then those values have to be replaced by mathematical methods if the number is insignificant they can be removed.



```
#checking for missing values and data types in train data
value = []
for col in train_df.columns:
    dtype = str(train_df[col].dtype)
    value.append(dtype)
dt= pd.DataFrame(data=train_df.isnull().sum(),columns=['Missing'])
dt['dtypes']=value
np.transpose(dt)
```

	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
Missing	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
dtypes	object	int64	float64	float64	float64	float64	float64	float64	float64	float64	...	float64	float64	float64	float64	float64	float64	float64

2 rows × 202 columns



	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
Missing	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0
dtypes	object	float64	float64	float64	float64	float64	float64	float64	float64	float64	...	float64	float64	float64	float64	float64	float64	float64

2 rows × 201 columns

As we can see there is no missing values in both train as well as in test data.

2.1.3 Outlier Analysis:

Outliers in the data may occur due to poor measurement quality or some external reasons. As they may effect in our prediction modelling we have to deal with it. In a simple way we can detect outliers by plotting box plots of the different variables in the data set.

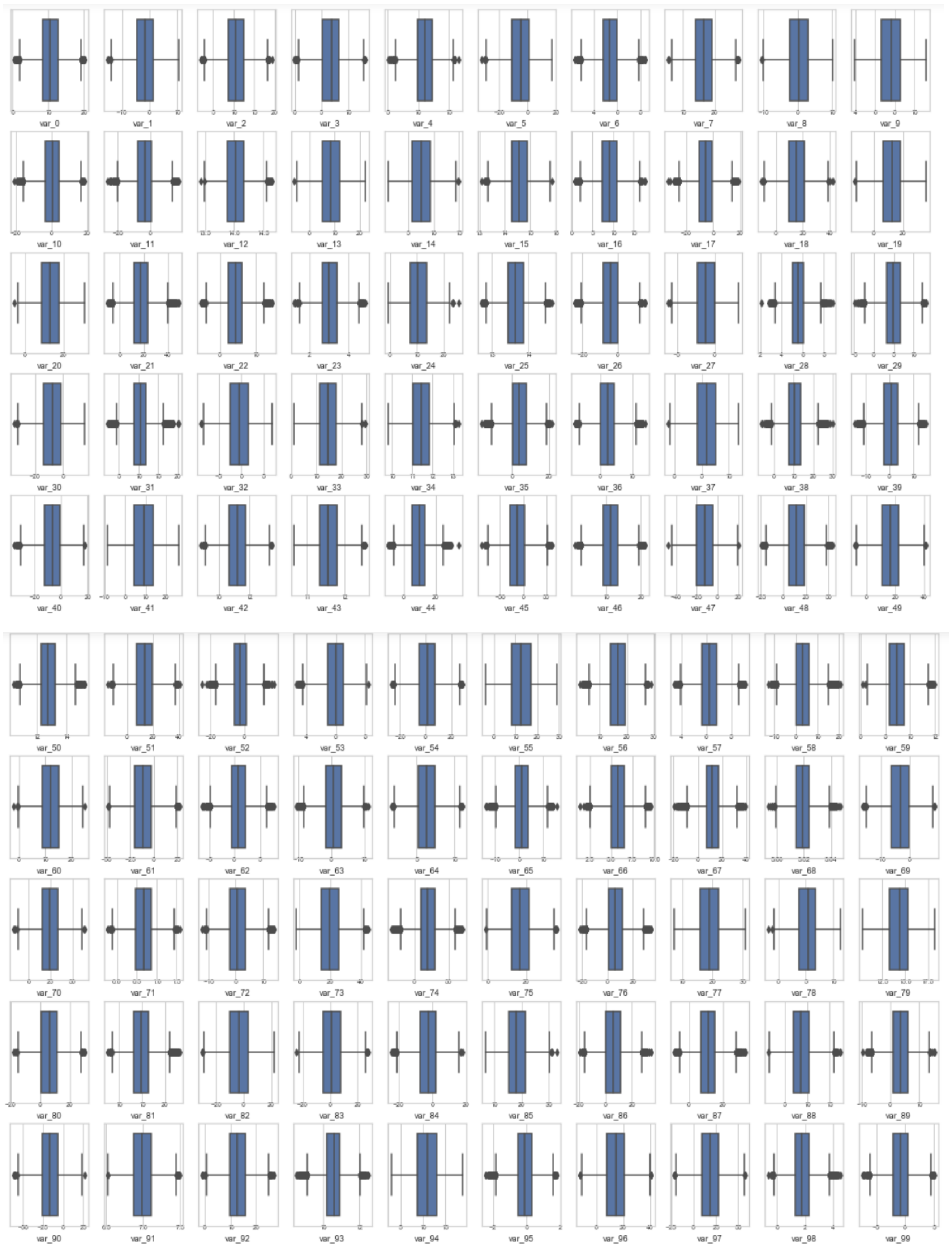
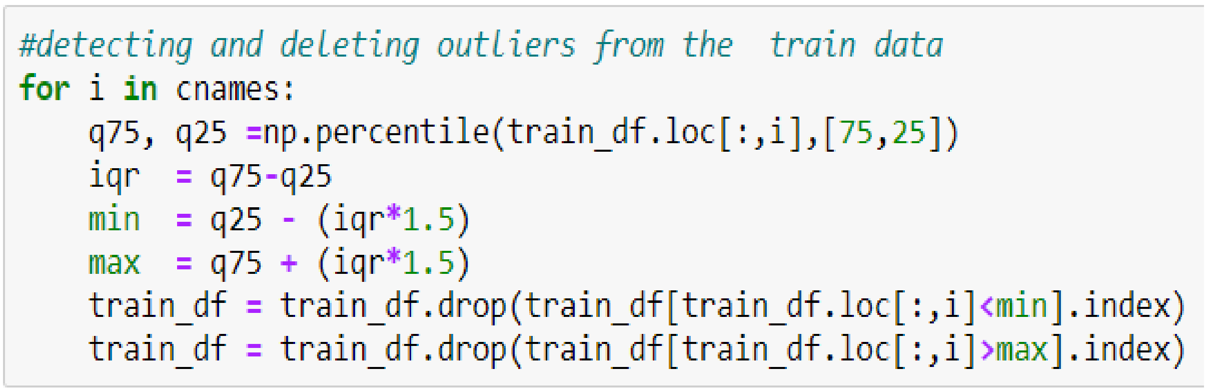

var_0
var_1
var_2
var_3
var_4
var_5
var_6
var_7
var_8
var_9
var_10
var_11
var_12
var_13
var_14
var_15
var_16
var_17
var_18
var_19
var_20
var_21
var_22
var_23
var_24
var_25
var_26
var_27
var_28
var_29
var_30
var_31
var_32
var_33
var_34
var_35
var_36
var_37
var_38
var_39
var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59
var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79
var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99

Fig. 2.3

An outlier is a data point that differs significantly from other observations. Box plot diagram is a graphical method typically depicted by quartiles and inter quartiles that helps in defining the upper limit and lower limit beyond which any data lying will be considered as outliers.

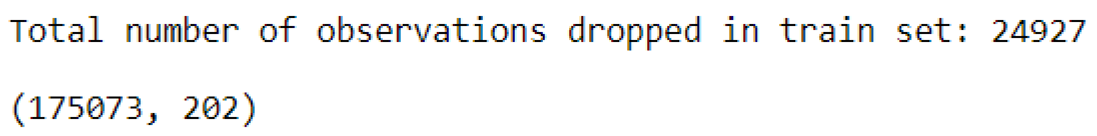
From this we can see that most of the variables have outliers so this may affect the model so we have to detect them as we have huge number of observation we can remove them.

Here is the method to remove the outliers from the data with the 25th and 75th percentile calculating the MIN and MAX then removing all the data points less than MIN and greater than MAX.



```
#detecting and deleting outliers from the  train data
for i in cnames:
    q75, q25 =np.percentile(train_df.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)
```

Even the test data have outliers the same process is done on the test data as well to deal with outliers.



```
Total number of observations dropped in train set: 24927

(175073, 202)
```

2.1.4 Feature Selection:

In the classification modelling feature selection is about selecting the independent variables which will be helpful in predicting the target variable. It is also know as Dimensionality Reduction. For numerical data we can use correlation plot

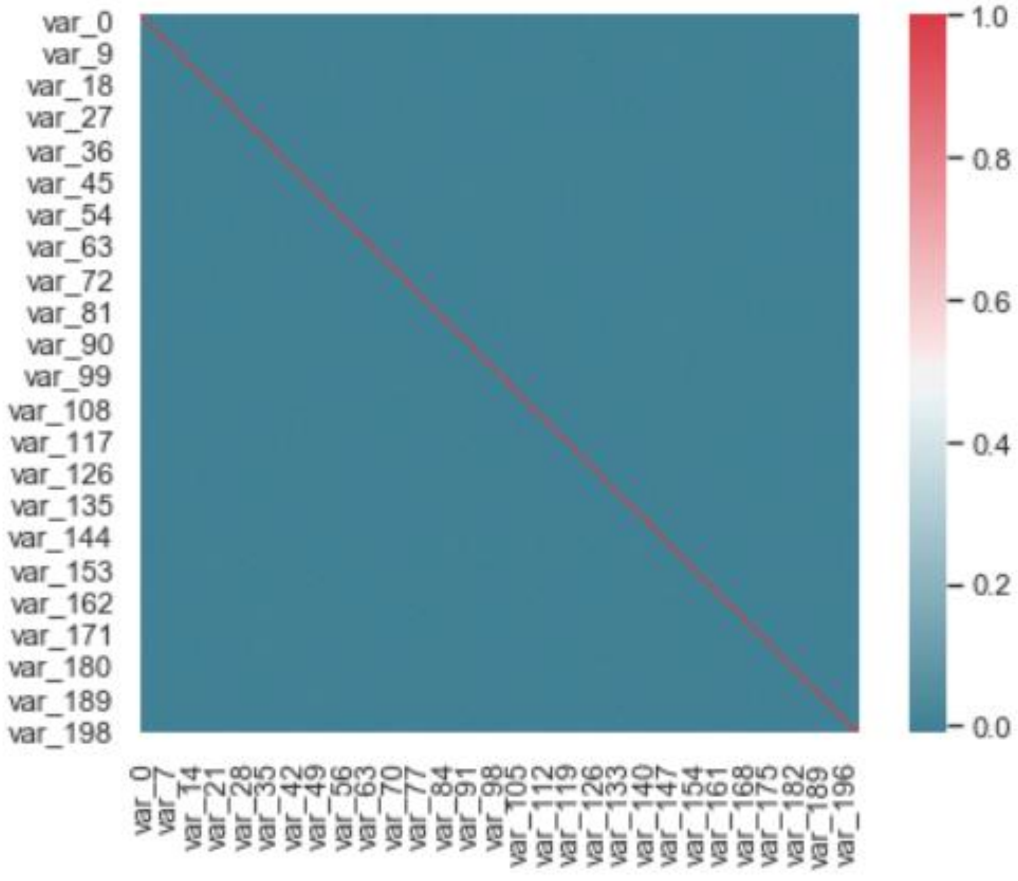


Fig. 2.4

Correlation:

1. It show whether and how strongly pairs of variables are related to each other.

2. Correlation takes values between -1 to +1, wherein values close to +1 represents strong positive correlation and values close to -1 represents strong negative correlation.

3. Value 0 represents no relation between variables. Since all the independent variables are of numeric type no need to perform chi square test as it is only applicable if there is any categorical variable in the dataset.

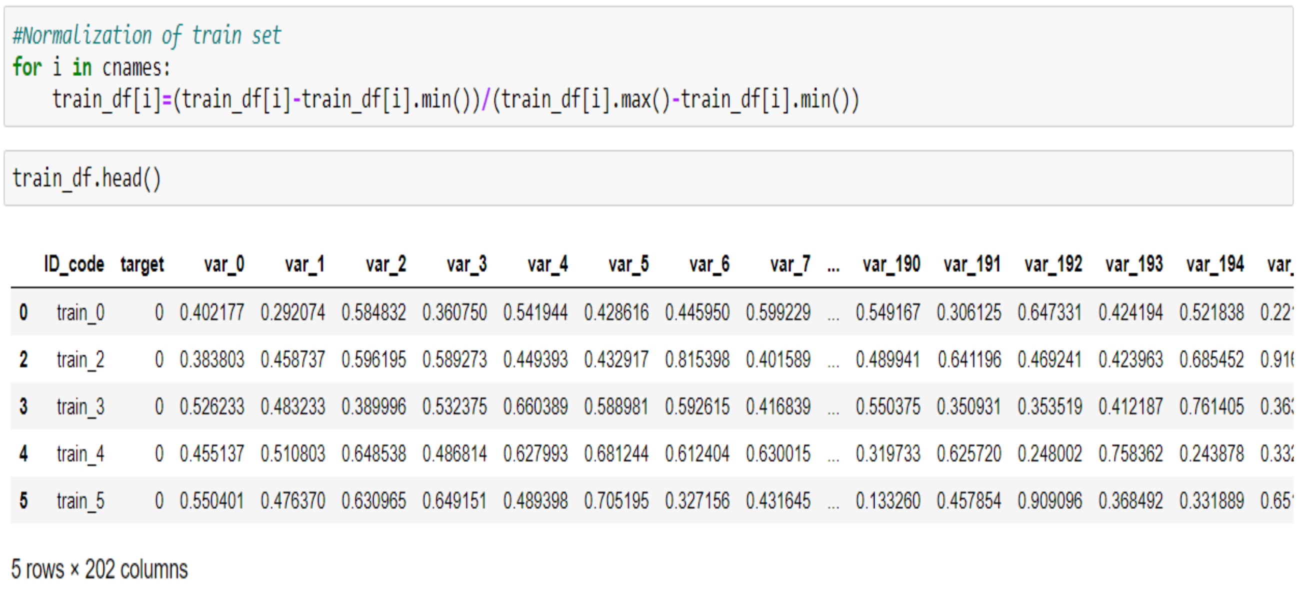
From the colour of the graph we can see that there isn't much correlation between the variables. So, we have to keep all the 200 columns.

2.1.5 Feature Selection

While working to learn about data it is important to scale the features to a range which are centered around zero. It is done to bring the feature on a same scale. This is done so that the variance of the features are in the same range. If a feature’s variance is orders of magnitude more than the variance of other features, that particular feature might dominate other features in the dataset, which is not something we want happening in our model. Before choosing the data scaling method, we need to check the distribution of data. If the data is uniformly distributed, then Standardization is the suitable method for the scaling purpose. On the other hand, if the data is not normally distributed, we go with Normalization scaling method. Normalization is bringing all the variables into proportion with one another. Normalization is the process of reducing unwanted variation either within or between variables and the range lies between 0 and 1.

As we saw in distribution of data there are few variables which are not normally distributed and as we want all the variables to be on same scale so we normalize the data.

Code:



```python
#Normalization of train set
for i in cnames:
    train_df[i]=(train_df[i]-train_df[i].min())/(train_df[i].max()-train_df[i].min())
```

```python
train_df.head()
```

```
   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_
0  train_0       0  0.402177  0.292074  0.584832  0.360750  0.541944  0.428616  0.445950  0.599229  ...  0.549167  0.306125  0.647331  0.424194  0.521838  0.22
2  train_2       0  0.383803  0.458737  0.596195  0.589273  0.449393  0.432917  0.815398  0.401589  ...  0.489941  0.641196  0.469241  0.423963  0.685452  0.91
3  train_3       0  0.526233  0.483233  0.389996  0.532375  0.660389  0.588981  0.592615  0.416839  ...  0.550375  0.350931  0.353519  0.412187  0.761405  0.36
4  train_4       0  0.455137  0.510803  0.648538  0.486814  0.627993  0.681244  0.612404  0.630015  ...  0.319733  0.625720  0.248002  0.758362  0.243878  0.33
5  train_5       0  0.550401  0.476370  0.630965  0.649151  0.489398  0.705195  0.327156  0.431645  ...  0.133260  0.457854  0.909096  0.368492  0.331889  0.65

5 rows × 202 columns
```

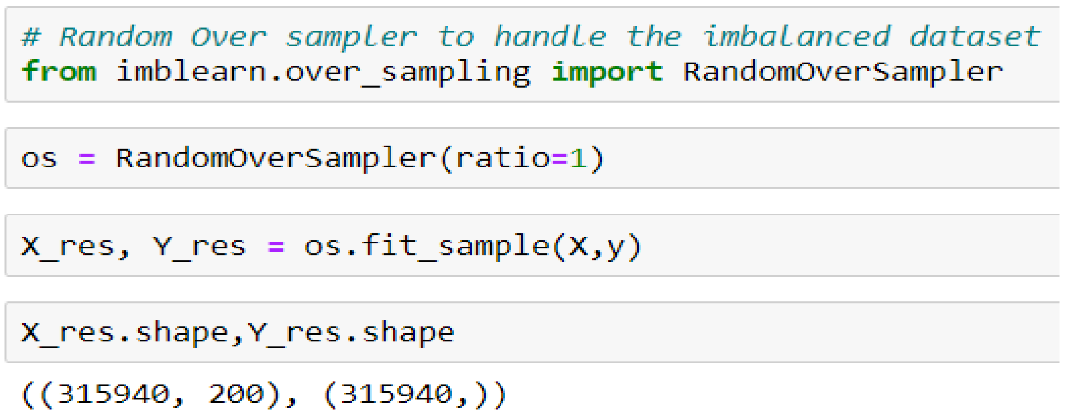
The same is done on the test data as well to scale the data.

As we have observed that the data set which is provided to us is an imbalanced data set the target variables in the target column are not equally distributed. So, we try resampling the target variable as we have 90% of 0’s and only 10% of 1’s we use over sampling technique so that 1’s match with the 0’s as well so that the model cannot be baised.

Even there are few disadvantages using sampling techniques on data sets as that might lead to the over fitting of the model so, tried sampling then classification in python and normal classification in R.

Resampling:

Code:



```
# Random Over sampler to handle the imbalanced dataset
from imblearn.over_sampling import RandomOverSampler
```

```
os = RandomOverSampler(ratio=1)
```

```
X_res, Y_res = os.fit_sample(X,y)
```

```
X_res.shape,Y_res.shape
```

```
((315940, 200), (315940,))
```

After sampling we have a balanced data set

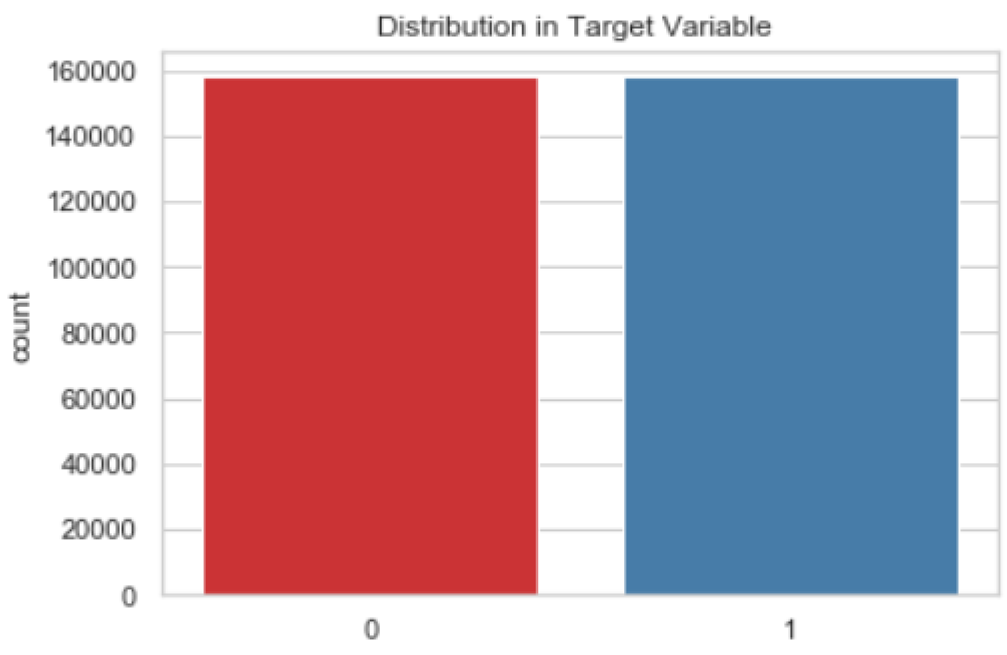


Fig. 2.5

2.2 Modelling:

In modelling we first have to split the clean dataset to train-set and test-set and then develop different models and evaluate them by metrics.

Code:

```
#dividing data_set into train & test
X_train, X_test, Y_train, Y_test= train_test_split(X_res, Y_res, test_size=0.2, random_state=82)
```

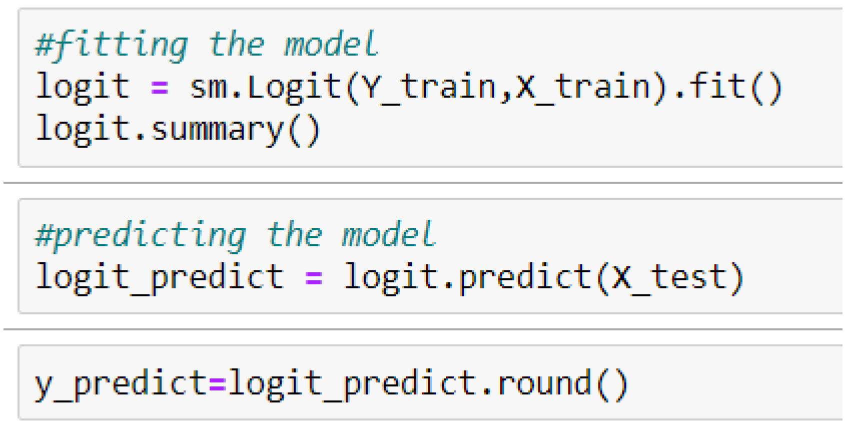
```
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
```

```
(252752, 200) (63188, 200) (252752,) (63188,)
```

2.2.1 Logistic Regression:

Logistic regression is a statistical model. It uses a logit model to model the probability of a certain class. It is used to explain the relationship between one dependent binary variable and one or more independent variable. If the target variable is categorical variable then go for logistic regression. Logistic regression is used only for the classification model. Input can be continuous or categorical. Output could be class(yes/no) and probabilities (0/1).

Code :



```
#fitting the model
logit = sm.Logit(Y_train,X_train).fit()
logit.summary()
```

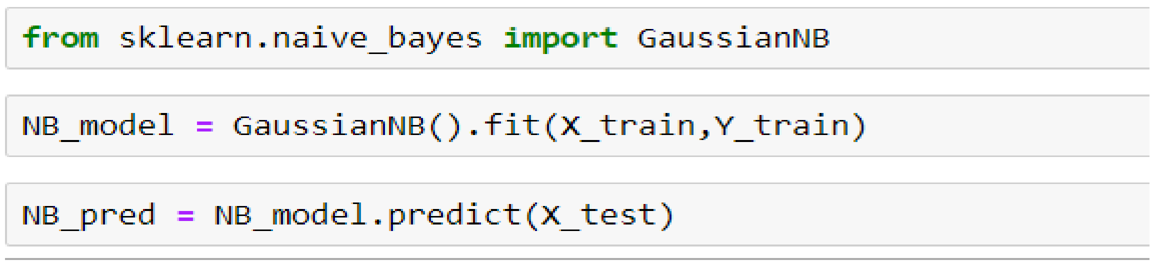
```
#predicting the model
logit_predict = logit.predict(X_test)
```

```
y_predict=logit_predict.round()
```

2.2.2 Naïve Bayes:

Naive Bayes classifiers are a family of simple "probabilistic classifier" based on applying bayes theorm with strong independence assumptions between the features. This is only use for the classification purpose. It is the one of the supervised machine learning algorithm which works based on the probability. Basically this allow us to predict a class for a set of features or predictors using the probability. So it is called as probabilistic algorithm for classifier.

Code:



```
from sklearn.naive_bayes import GaussianNB
```

```
NB_model = GaussianNB().fit(X_train,Y_train)
```

```
NB_pred = NB_model.predict(X_test)
```

2.2.3 Random Forest:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees. In the random forest classifier, the higher the number of trees in the forest gives the high accuracy results. As I have multiple no of features, I wanted a way to create a model that only includes the most important features. Random Forests are often used for feature selection in a data science. The reason is because the tree-based strategies used by random forests naturally ranks by how well they improve the purity of the node. This mean decrease in impurity over all trees (called gini impurity). Nodes with the greatest decrease in impurity happen at the start of the trees, while notes with the least decrease in impurity occur at the end of trees.

Code :

```
from sklearn.ensemble import RandomForestClassifier
```

```
#fitting the model
RF_model = RandomForestClassifier(n_estimators=5).fit(x_train,y_train)
```

```
#predicting
RF_pred = RF_model.predict(x_test)
```

Chapter-3

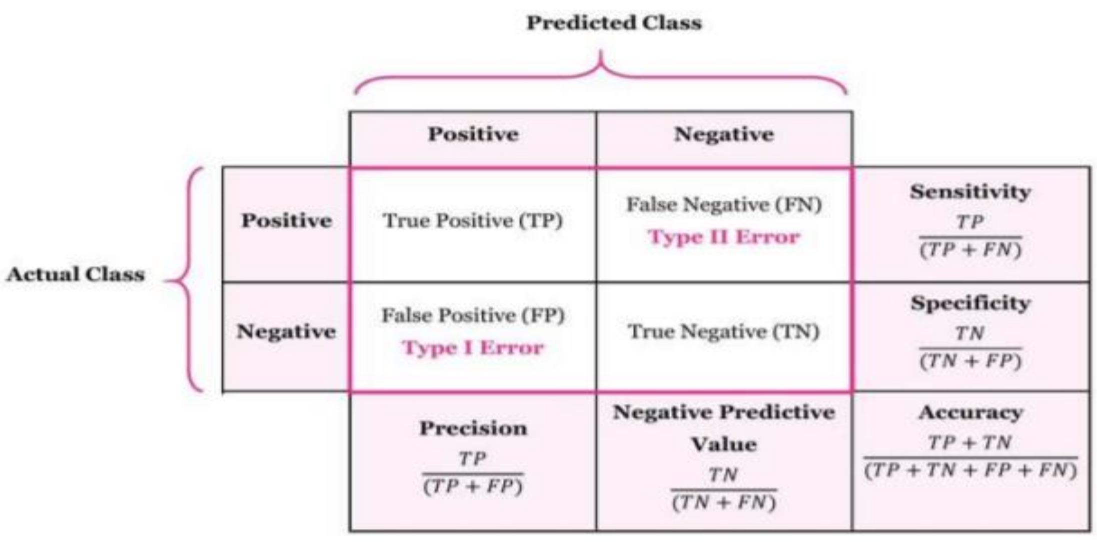
CONCLUSION

3.1 Model Evaluation:

For Evaluating a classification model we have regression metrics they are :-

3.1.1 Confusion Matrix

In machine learning, confusion matrix is one of the easiest ways to summarize the performance of your algorithm. At times, it is difficult to judge the accuracy of a model by just looking at the accuracy because of problems like unequal distribution. So, a better way to check how good your model is, is to use a confusion matrix.



Predicted Class

		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	**Negative Predictive Value** $\frac{TN}{(TN + FN)}$	**Accuracy** $\frac{TP + TN}{(TP + TN + FP + FN)}$

3.1.2 Accuracy

It is the most intuitive performance measure and it simply a ratio of correctly predicted to the total observations.

Accuracy = True Positive + True Negative / (True Positive +False Positive + False Negative + True Negative)

3.1.3 Recall

Recall we can also called as sensitivity or true positive rate. It is several positives that our model predicts compared to the actual number of positives in our data.

Recall = True Positives / (True Positives + False Positives) Recall = True Positives / Total Actual Positive

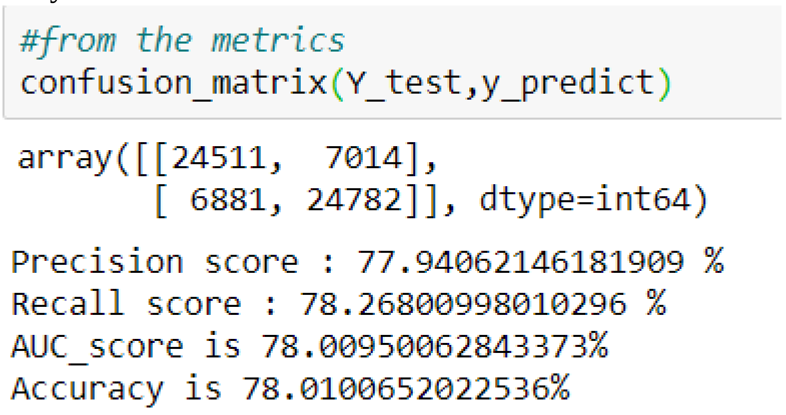
3.1.4 Precision

It is also called as the positive predictive value. Number of correct positives in your model that predicts compared to the total number of positives it predicts.

Precision = True Positives / (True Positives + False Positives) Precision = True Positives / Total predicted positive

3.2**Logistic Regression Metrics:**

In Python:



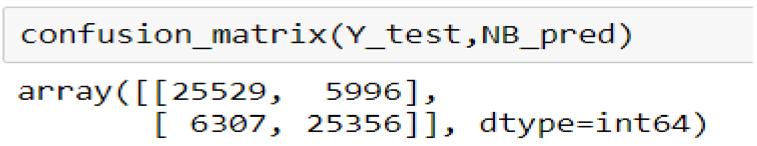
```
#from the metrics
confusion_matrix(Y_test,y_predict)
```

```
array([[24511,  7014],
       [ 6881, 24782]], dtype=int64)
```

```
Precision score : 77.94062146181909 %
Recall score : 78.26800998010296 %
AUC_score is 78.00950062843373%
Accuracy is 78.0100652022536%
```

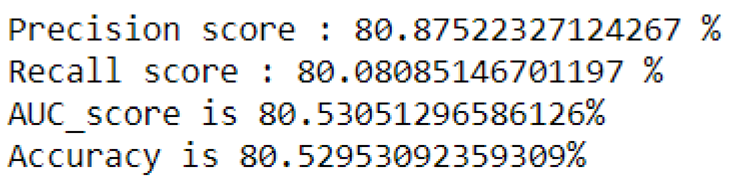
3.3**Naïve Bayes Metrics :**

Code:



```
confusion_matrix(Y_test,NB_pred)
```

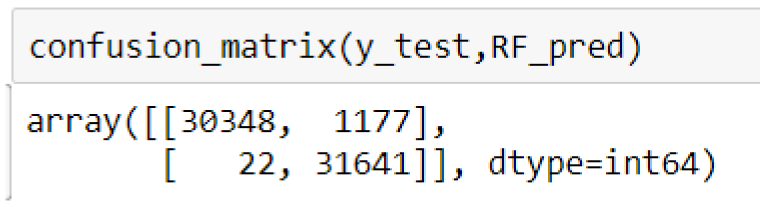
```
array([[25529,  5996],
       [ 6307, 25356]], dtype=int64)
```



```
Precision score : 80.87522327124267 %
Recall score : 80.08085146701197 %
AUC_score is 80.53051296586126%
Accuracy is 80.52953092359309%
```

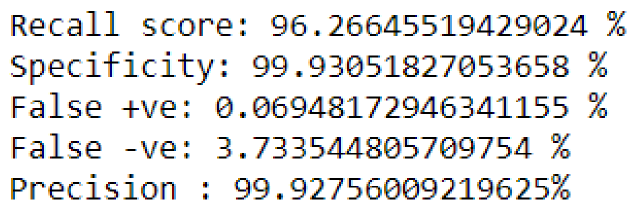
3.4**Random Forest Metrics:**

Code:

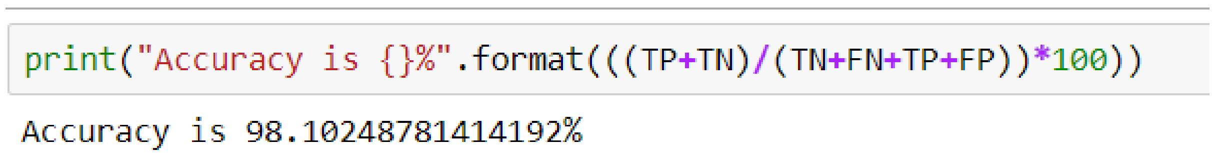


```
confusion_matrix(y_test,RF_pred)
```

```
array([[30348,  1177],
       [   22, 31641]], dtype=int64)
```



```
Recall score: 96.26645519429024 %
Specificity: 99.93051827053658 %
False +ve: 0.06948172946341155 %
False -ve: 3.733544805709754 %
Precision : 99.92756009219625%
```



```
print("Accuracy is {}%".format(((TP+TN)/(TN+FN+TP+FP))*100))
```

```
Accuracy is 98.10248781414192%
```

3.2 Model Selection:

From above all,

Random Forest is better for classifying customer transaction as it has better accuracy of 98.10% with recall score of 0.96.

……………..THANK YOU………………..