Santander

Customer Transaction Prediction

POOJA JOSHI

Table of Contents

INTRODUCTION

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 PROBLEM DESCRIPTION:

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?

According to past data and from the given problem the output is Classification and it comes under Supervised Machine Learning. We train the model with past data and when the new data is given we predict the outcome

1.3 DATA:

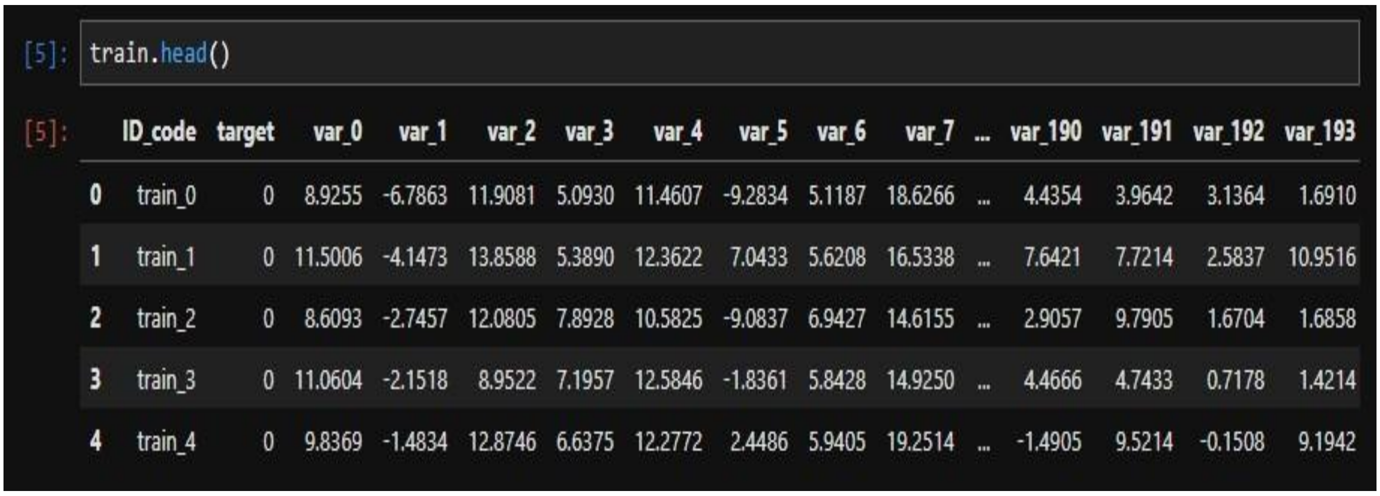
Given data contains numeric feature variables, the binary target column, and a string ID code column. The task is to predict the value of target column in the test set.

ID code (string); Target;

200 numerical variables, named from var_0 to var_199;

It has 201 predictors or independent variables and 1 target variable 'target'

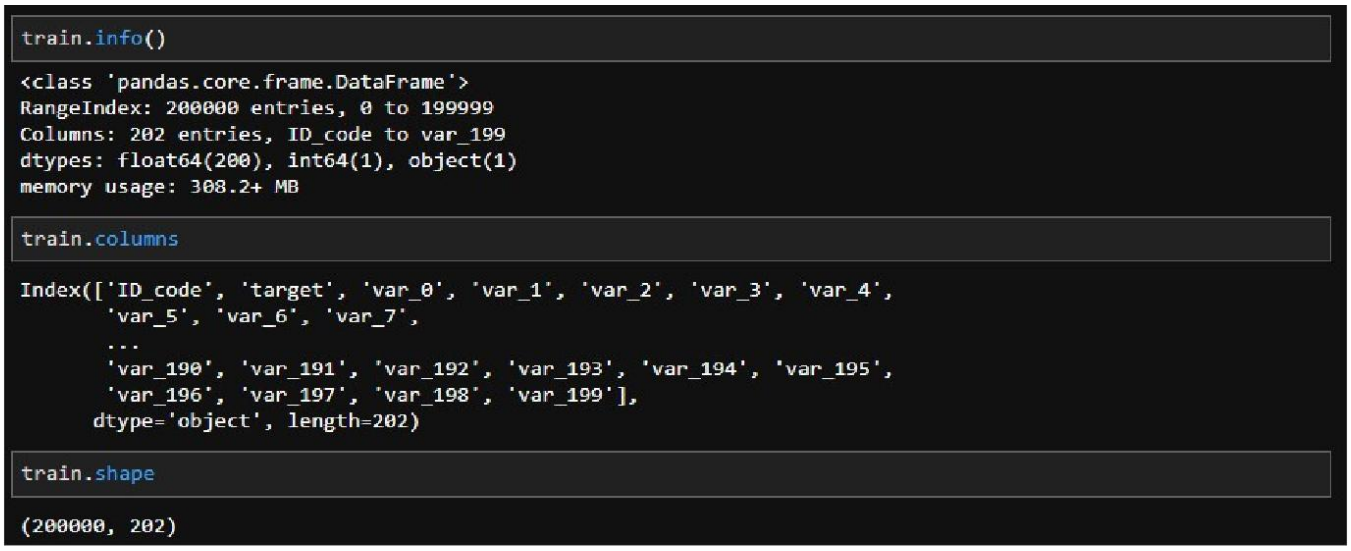
Exploratory Data Analysis



```
[5]: train.head()
```

[5]:

	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
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
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
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
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
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



```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 202 entries, ID_code to var_199
dtypes: float64(200), int64(1), object(1)
memory usage: 308.2+ MB
```

```
train.columns
```

```
Index(['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'],
      dtype='object', length=202)
```

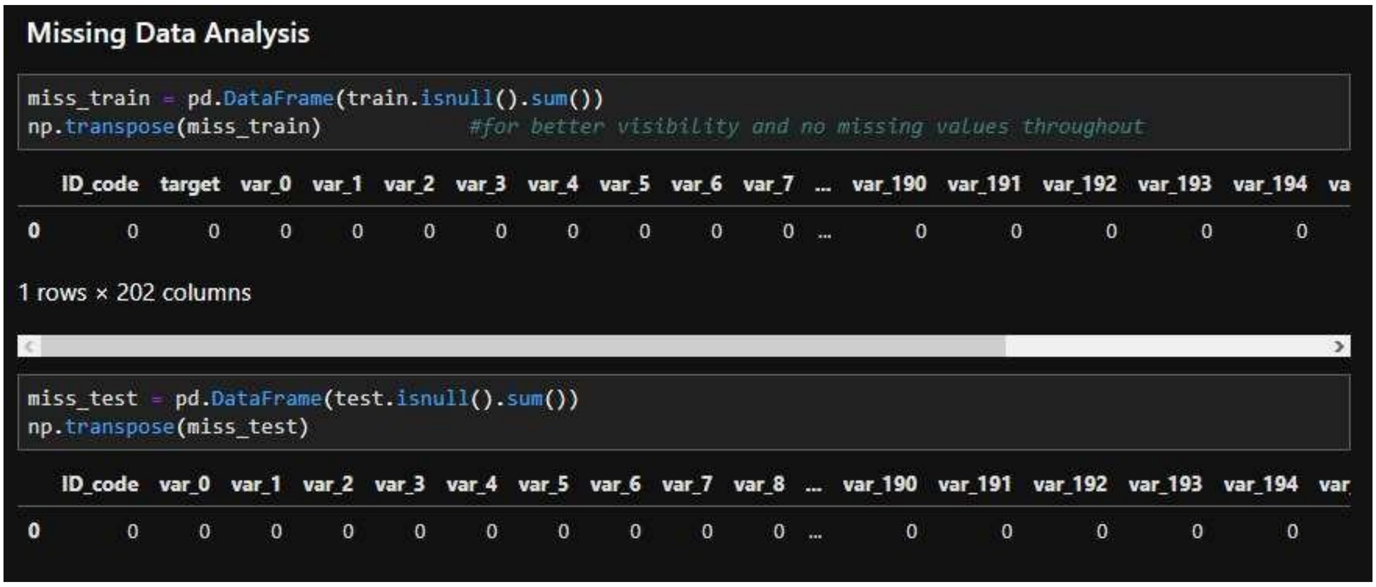
```
train.shape
```

```
(200000, 202)
```

Pre-processing

2.1 Missing Value Analysis:

Missing values are which, where the values are missing in an observation in the dataset. It can occur due to human errors, individuals refusing to answer while surveying, optional box in questionnaire Usually we only consider those variables for missing value imputation whose missing values is less than 30%, if it above this we will drop that variable in our analysis as imputing missing values which are more than 30% doesn't make any sense and the information would also be insensible to consider.



Missing Data Analysis

```
miss_train = pd.DataFrame(train.isnull().sum())
np.transpose(miss_train)          #for better visibility and no missing values throughout
```

```
   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  va
0        0       0      0      0      0      0      0      0      0      0  ...        0        0        0        0        0

1 rows × 202 columns
```

```
miss_test = pd.DataFrame(test.isnull().sum())
np.transpose(miss_test)
```

```
   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
0        0      0      0      0      0      0      0      0      0      0  ...        0        0        0        0        0
```

IN OUR GIVEN DATASET WE ARE NOT HAVING ANY MISSING VALUES....SO WE ARE NOT PROCEEDING WITH THIS STEP TO IMPUTE ANY MISSING VALUES.

2.2 DATA VISUALIZATIONS

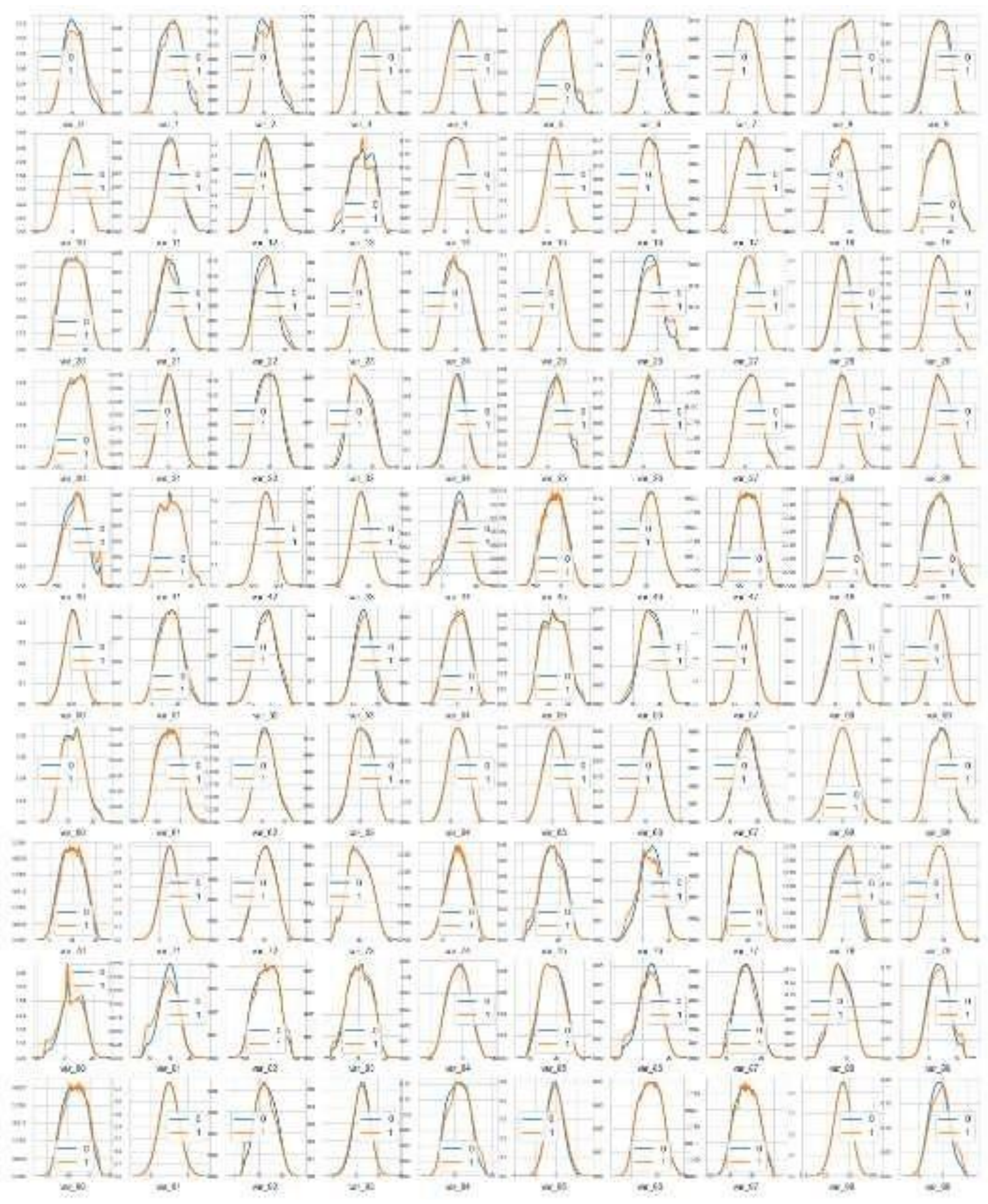
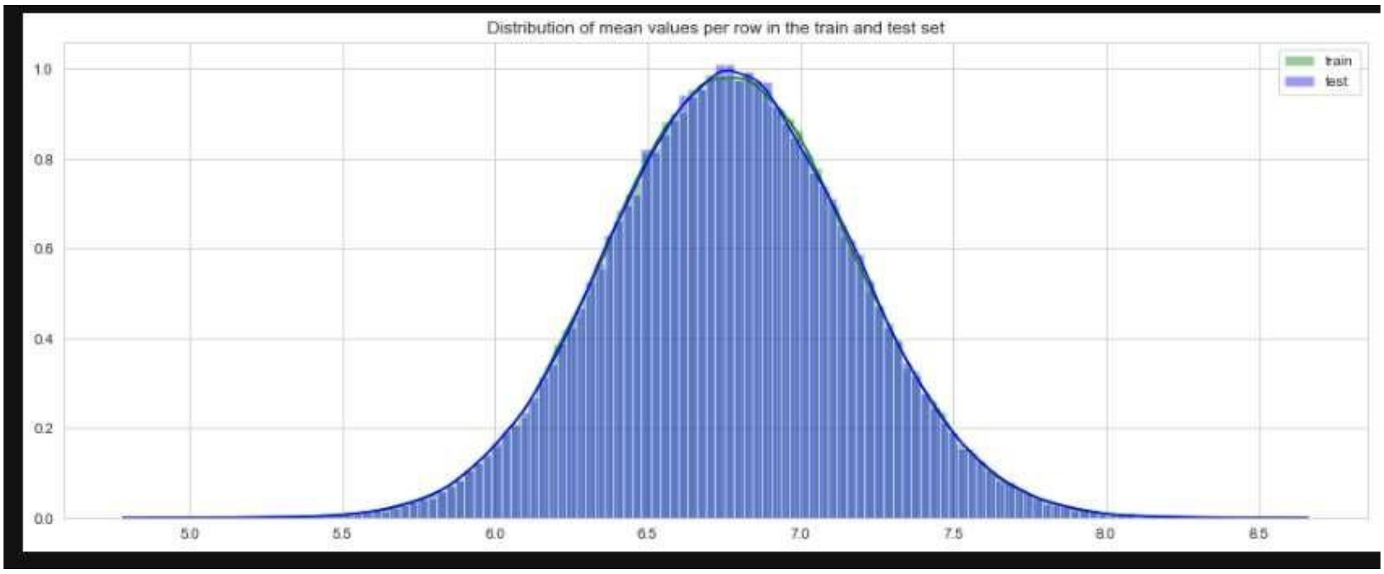
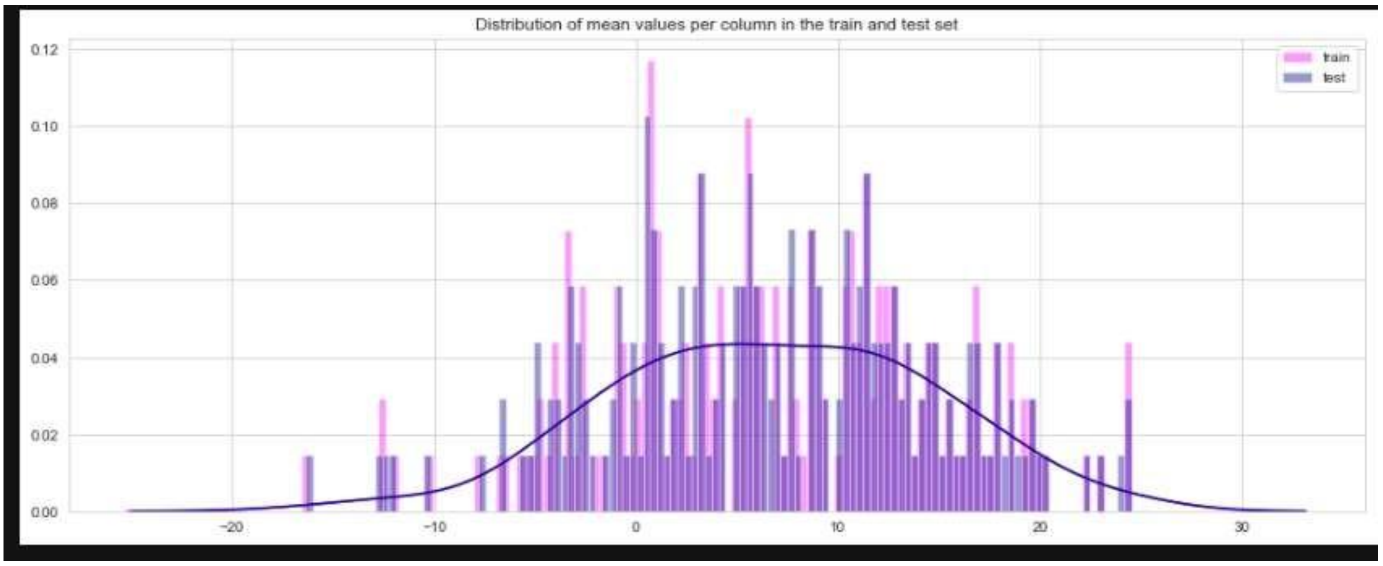


Fig: Density plots of features

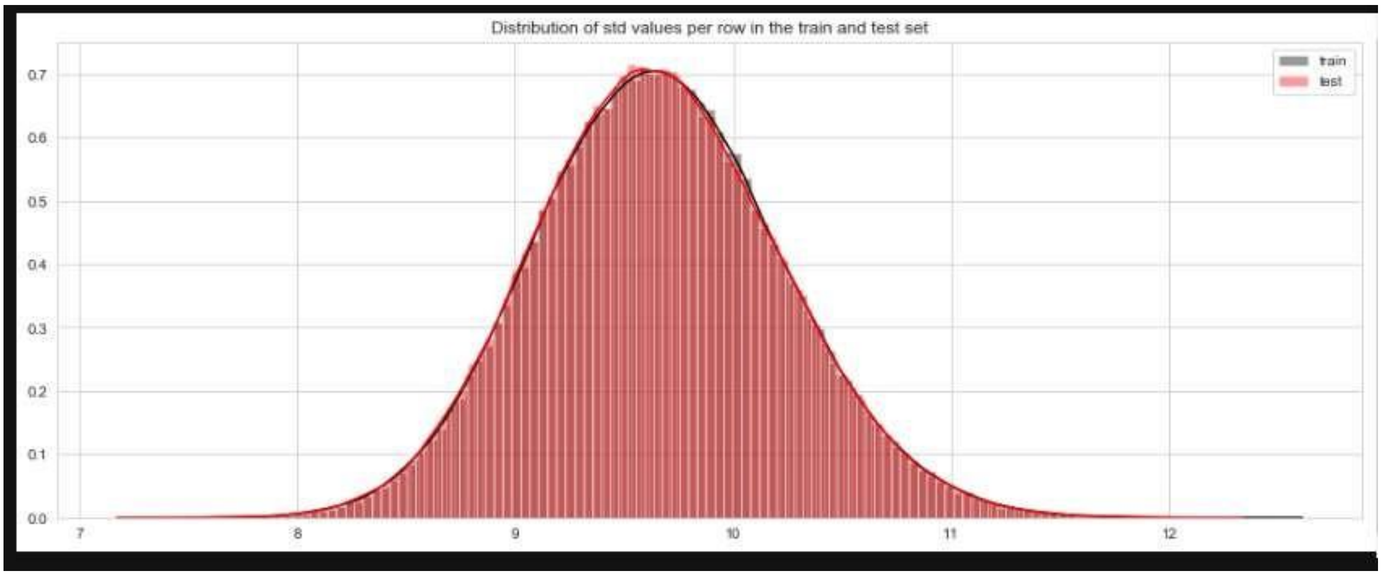
We can observe that there is a considerable number of features with significant different distribution for the two target values.



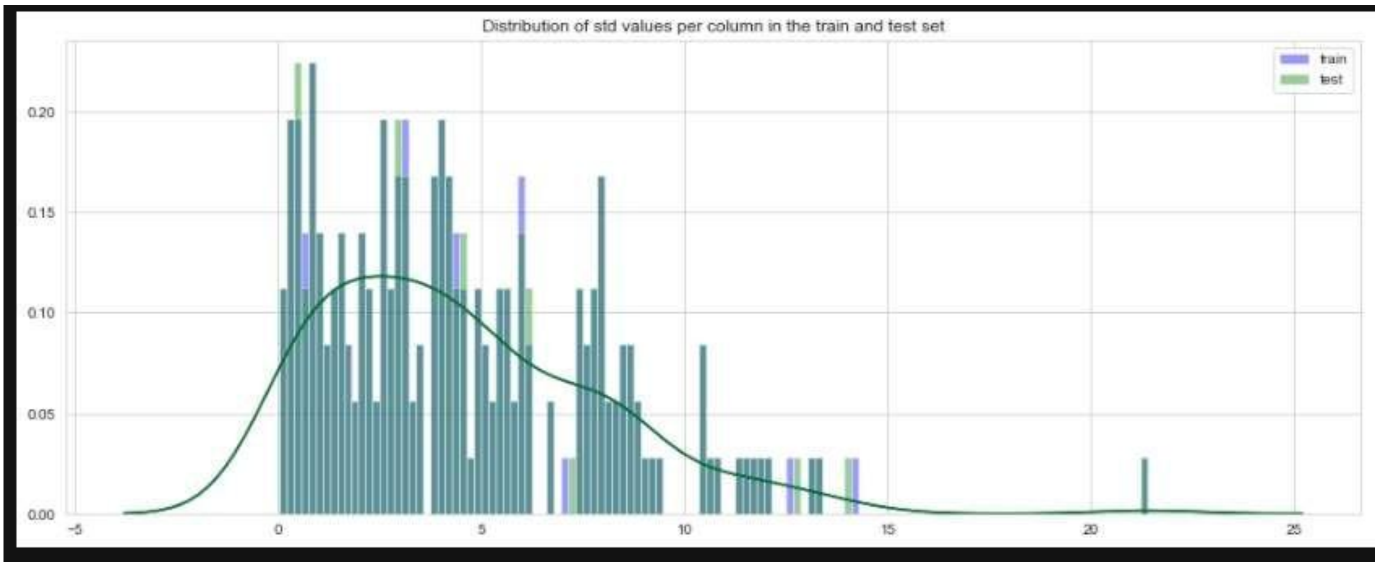
Distribution of the mean values per row in the train and test set.



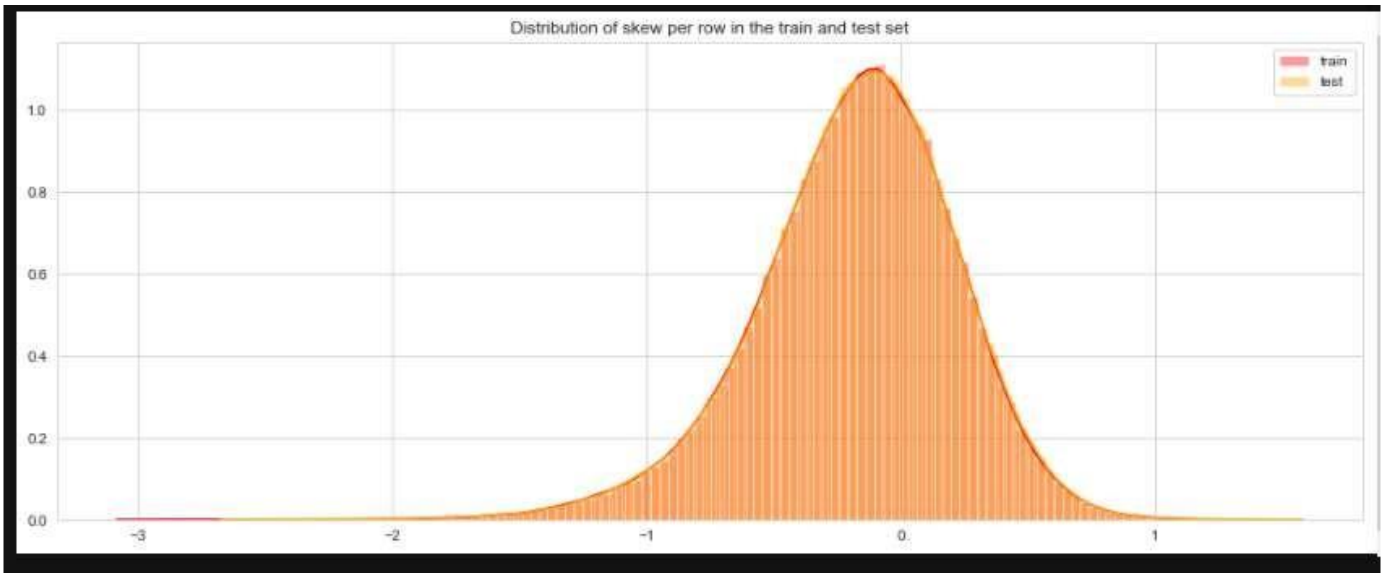
Distribution of the mean values per columns in the train and test set.



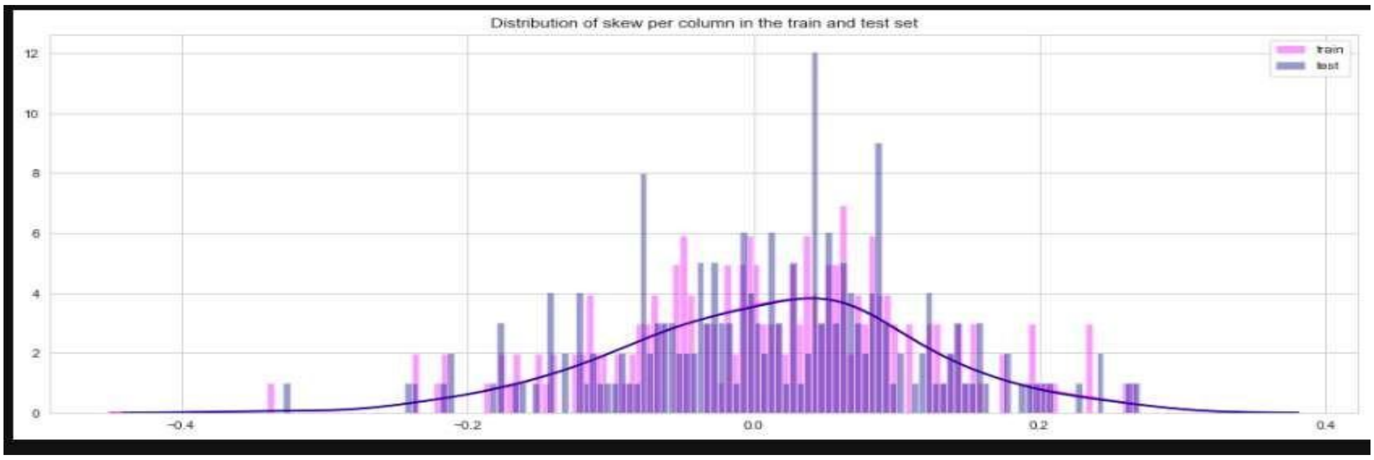
Distribution of standard deviation of values per row for train and test datasets.



Distribution of the standard deviation of values per columns in the train and test datasets.



Distribution of skew per row in the train and test set.

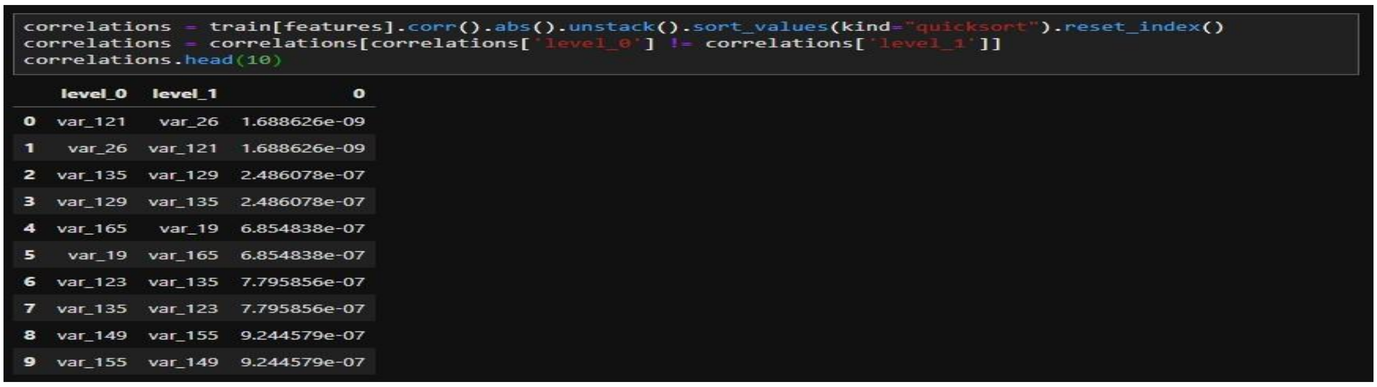


Distribution of skew per column in the train and test set.

2.3 FEATURE SELECTION

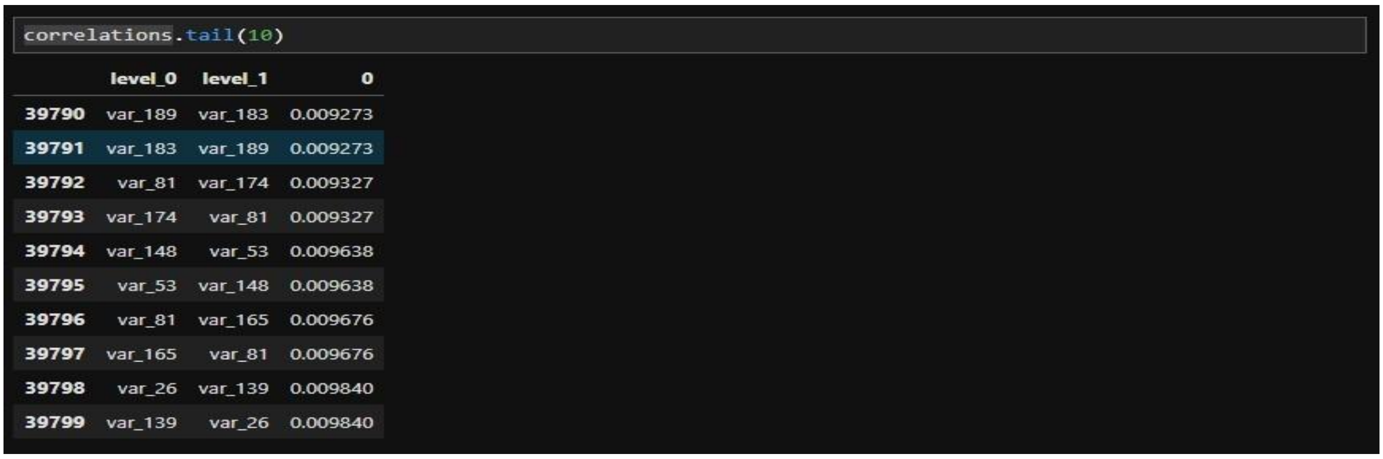
CORRELATION:

Before performing any type of modeling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. This process of selecting a subset of relevant features/variables is known as feature selection. There are several methods of doing feature selection. I have used correlation analysis.



```
correlations = train[features].corr().abs().unstack().sort_values(kind="quicksort").reset_index()
correlations = correlations[correlations['level_0'] != correlations['level_1']]
correlations.head(10)
```

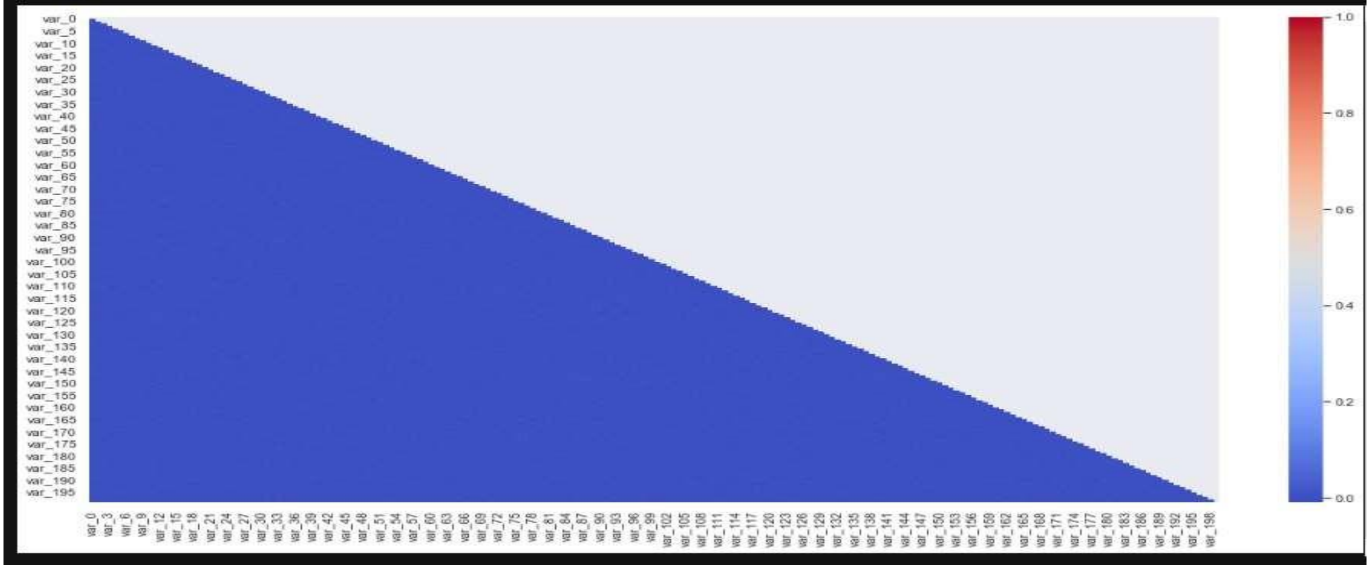
```
   level_0  level_1             0
0  var_121   var_26  1.688626e-09
1   var_26  var_121  1.688626e-09
2  var_135  var_129  2.486078e-07
3  var_129  var_135  2.486078e-07
4  var_165   var_19  6.854838e-07
5   var_19  var_165  6.854838e-07
6  var_123  var_135  7.795856e-07
7  var_135  var_123  7.795856e-07
8  var_149  var_155  9.244579e-07
9  var_155  var_149  9.244579e-07
```



```
correlations.tail(10)
```

```
       level_0  level_1         0
39790  var_189  var_183  0.009273
39791  var_183  var_189  0.009273
39792   var_81  var_174  0.009327
39793  var_174   var_81  0.009327
39794  var_148   var_53  0.009638
39795   var_53  var_148  0.009638
39796   var_81  var_165  0.009676
39797  var_165   var_81  0.009676
39798   var_26  var_139  0.009840
39799  var_139   var_26  0.009840
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x268a5eec0f0>
```



In our dataset, the correlation between the train attributes is very small. So, there is no need to remove variables.

2.4 HANDLING IMBALANCED DATA

Imbalanced classes are a common problem in machine learning classification where there is a disproportionate ratio of observations in each class. Class imbalance can be found in many different areas including medical diagnosis, spam filtering, and fraud detection. Some popular methods for dealing with class imbalance.

1. Change the performance metric

Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be very misleading. Metrics that can provide better insight include:

- Confusion Matrix: a table showing correct predictions and types of incorrect predictions.
- Precision: the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- Recall: the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- F1: Score: the weighted average of precision and recall.

Change the algorithm

While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets. Decision trees, Random Forests and Naive Bayes frequently perform well on imbalanced data. They work by learning a hierarchy of if/else questions and this can force both classes to be addressed.

2.5 Resampling Techniques

Oversample minority class

Our next method begins our resampling techniques. Oversampling can be defined as adding more copies of the minority class. Oversampling can be a good choice when you don't have a ton of data to work with. We will use the resampling module from Scikit-Learn to randomly replicate samples from the minority class.

Always split into test and train sets BEFORE trying oversampling techniques! Oversampling before splitting the data can allow the exact same observations to be present in both the test and train sets. This can allow our model to simply memorize specific data points and cause overfitting and poor generalization to the test data.

After resampling we have an equal ratio of data points for each class

Under sample majority class

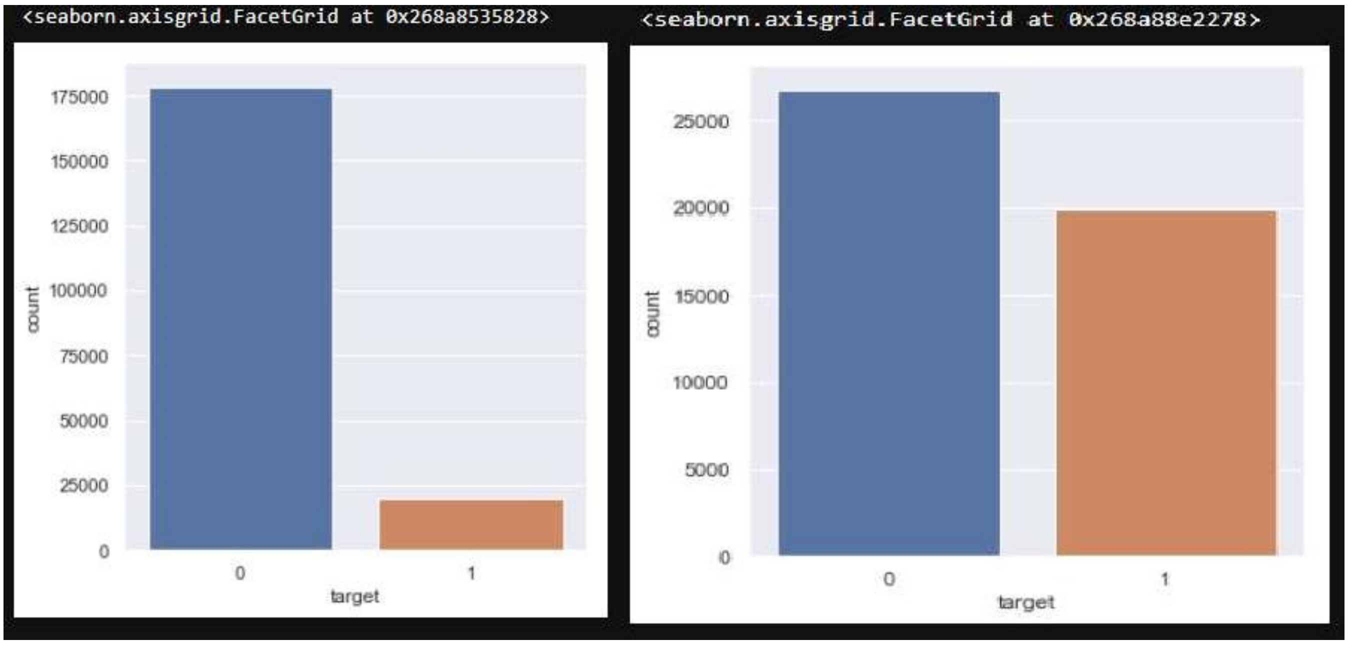
Under sampling can be defined as removing some observations of the majority class. Under sampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to under fitting and poor generalization to the test set.

Generate synthetic samples

A technique similar to up sampling is to create synthetic samples.

SMOTE (Synthetic Minority Oversampling Technique) uses a nearest neighbours algorithm to generate new and synthetic data we can use for training our model.

Again, it's important to generate the new samples only in the training set to ensure our model generalizes well to unseen data.



```
<seaborn.axisgrid.FacetGrid at 0x268a8535828>
```

```
<seaborn.axisgrid.FacetGrid at 0x268a88e2278>
```

IN OUR CASE GIVEN DATA IS IMBALANCED……WHERE 90% OF SAMPLES BELONGS TO CLASS 0 AND ONLY 10% BELONGS TO CLASS 1.

after applying under sampling technique, we get the balanced data as shown above.

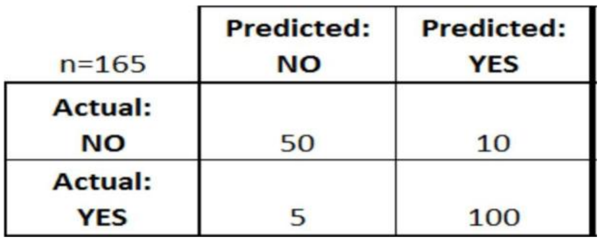
Model Development

This is the final phase of our project where we would build some machine learning models and will train our model on the data for future predictions. We would consider different machine learning algorithms to check which gives the best result.

3.1 Confusion Matrix

Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

Let's assume we have a binary classification problem. We have some samples belonging to two classes: YES, or NO. Also, we have our own classifier which predicts a class for a given input sample. On testing our model on 165 samples, we get the following result.



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

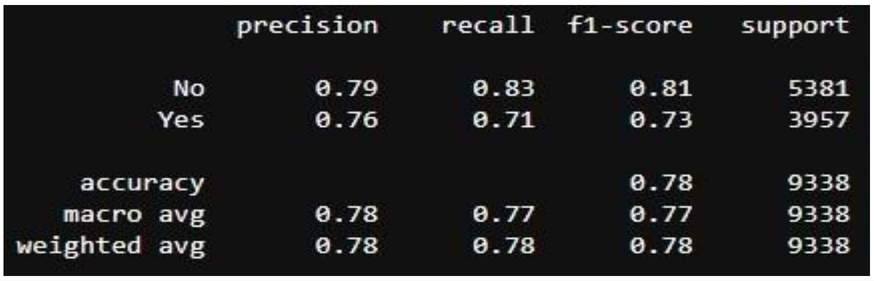
Confusion Matrix

There are 4 important terms:

- True Positives: The cases in which we predicted YES and the actual output was also YES.
- True Negatives: The cases in which we predicted NO and the actual output was NO.
- False Positives: The cases in which we predicted YES and the actual output was NO.
- False Negatives: The cases in which we predicted NO and the actual output was YES.

3.2 LOGISTIC REGRESSION:

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.



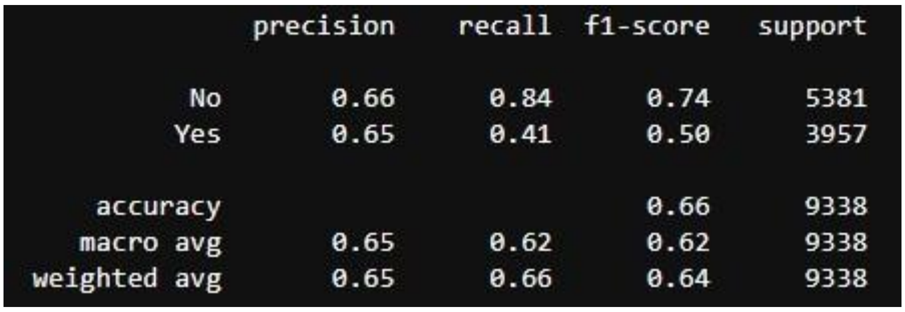
```
              precision    recall  f1-score   support

          No       0.79      0.83      0.81      5381
         Yes       0.76      0.71      0.73      3957

    accuracy                           0.78      9338
   macro avg       0.78      0.77      0.77      9338
weighted avg       0.78      0.78      0.78      9338
```

3.3 RANDOM FOREST

Random forests are based on a simple idea: 'the wisdom of the crowd'. Aggregate of the results of multiple predictors gives a better prediction than the best individual predictor. A group of predictors is called an ensemble. Thus, this technique is called Ensemble Learning.

To improve our technique, we can train a group of Decision Tree classifiers, each on a different random subset of the train set. To make a prediction, we just obtain the predictions of all individuals trees, then predict the class that gets the most votes. This technique is called Random Forest.

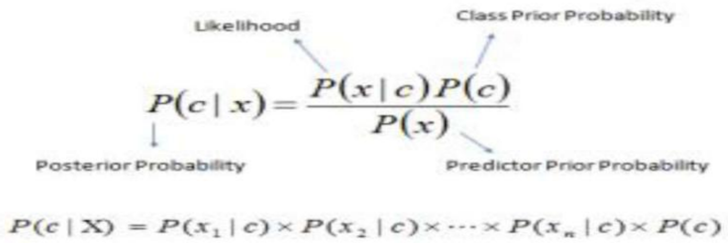
Random forest chooses a random subset of features and builds many Decision Trees. The model averages out all the predictions of the Decisions trees.



```
              precision    recall  f1-score   support

          No       0.66      0.84      0.74      5381
         Yes       0.65      0.41      0.50      3957

    accuracy                           0.66      9338
   macro avg       0.65      0.62      0.62      9338
weighted avg       0.65      0.66      0.64      9338
```

3.4 Naive Bayes

Naive Bayes is a Classification Algorithm. It is one of the most practical Machine Learning methods. It performs Probabilistic classification. It works on Bayes theorem of probability to predict the class of unknown data setBayes' theorem with strong (naive) independence assumptions between the features. Attribute values are conditionally independent given the target value. Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

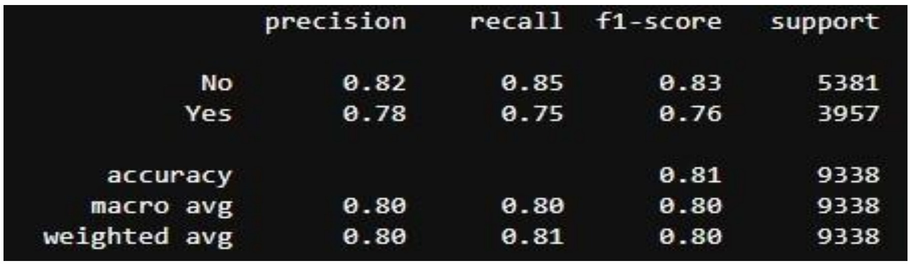
$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

P(c|x)is the posterior probability of class (target) given predictor (x,attributes).

P(c) is the prior probability of class.

P(x|c) is the likelihood which is the probability of predictor given class.

P(x) is the prior probability of predictor.



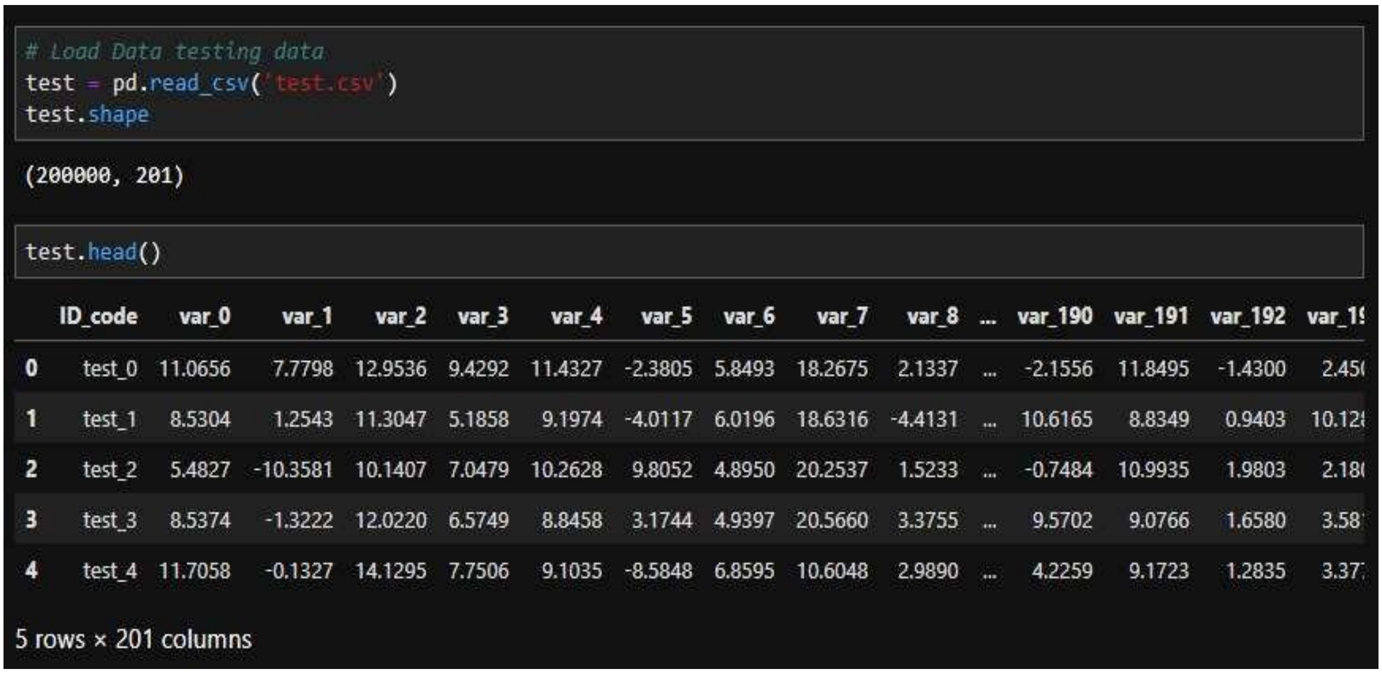
```
              precision    recall  f1-score   support

          No       0.82      0.85      0.83      5381
         Yes       0.78      0.75      0.76      3957

    accuracy                           0.81      9338
   macro avg       0.80      0.80      0.80      9338
weighted avg       0.80      0.81      0.80      9338
```

As per the project requirement we have to predict the results based Recall, Precision and Accuracy of all machine learning algorithm. "To get the most accurate model out of various models the value of Recall, Precision, AUC should be high", out of all the above developed Machine Learning algorithms we can deduce that "Naïve Bayes" is giving all the quantities highest among all other algorithms. Hence "Naïve Bayes" is selected to predict target variable from our given Test Data.

Model Deployment

After performing the machine learning algorithm and tested it for test data we have fetch out from train dataset the most accurate method. Now we are ready to predict any external data given to us. Now in the external data named as test.csv is given to us contain all variables of train data except "target" variable. Our task is to predict the target value in form of 0 and 1 where using the most appropriate algorithm we have formed in previous section. As we have developed in our previous section that NAÏVE BAYES gives most accurate model so we will use NAÏVE BAYES to predict the target value.

Steps followed for predicting target variable in test data.



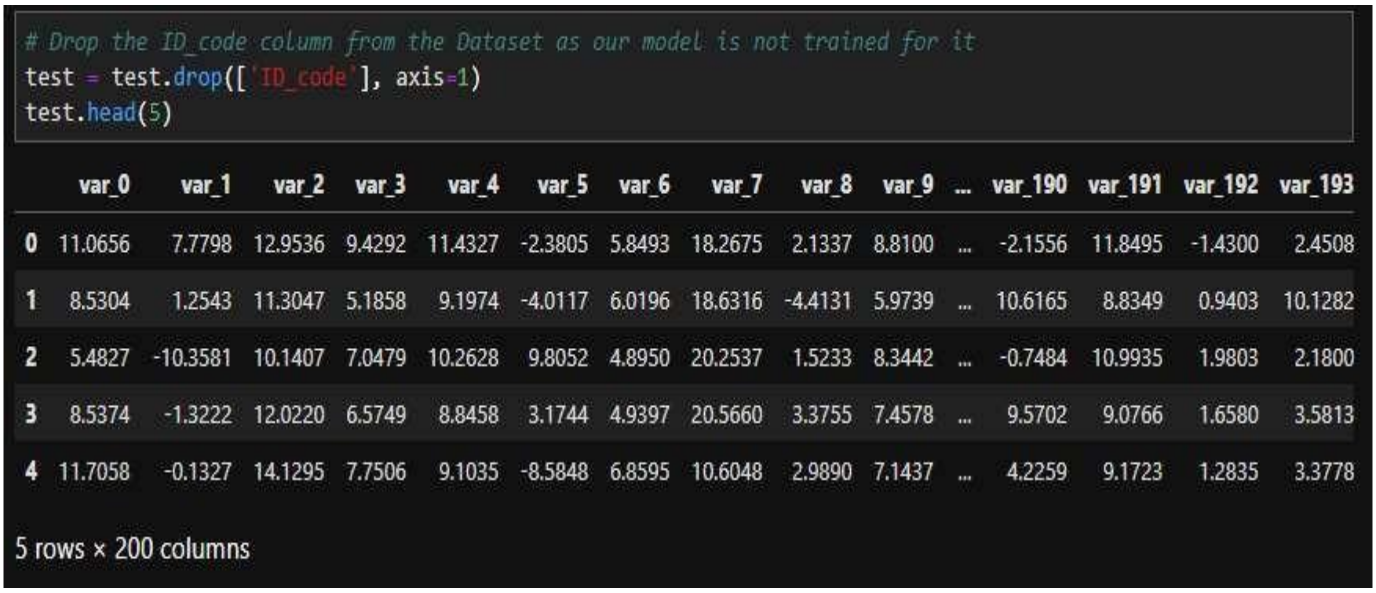
```
# Load Data testing data
test = pd.read_csv('test.csv')
test.shape
```

```
(200000, 201)
```

```
test.head()
```

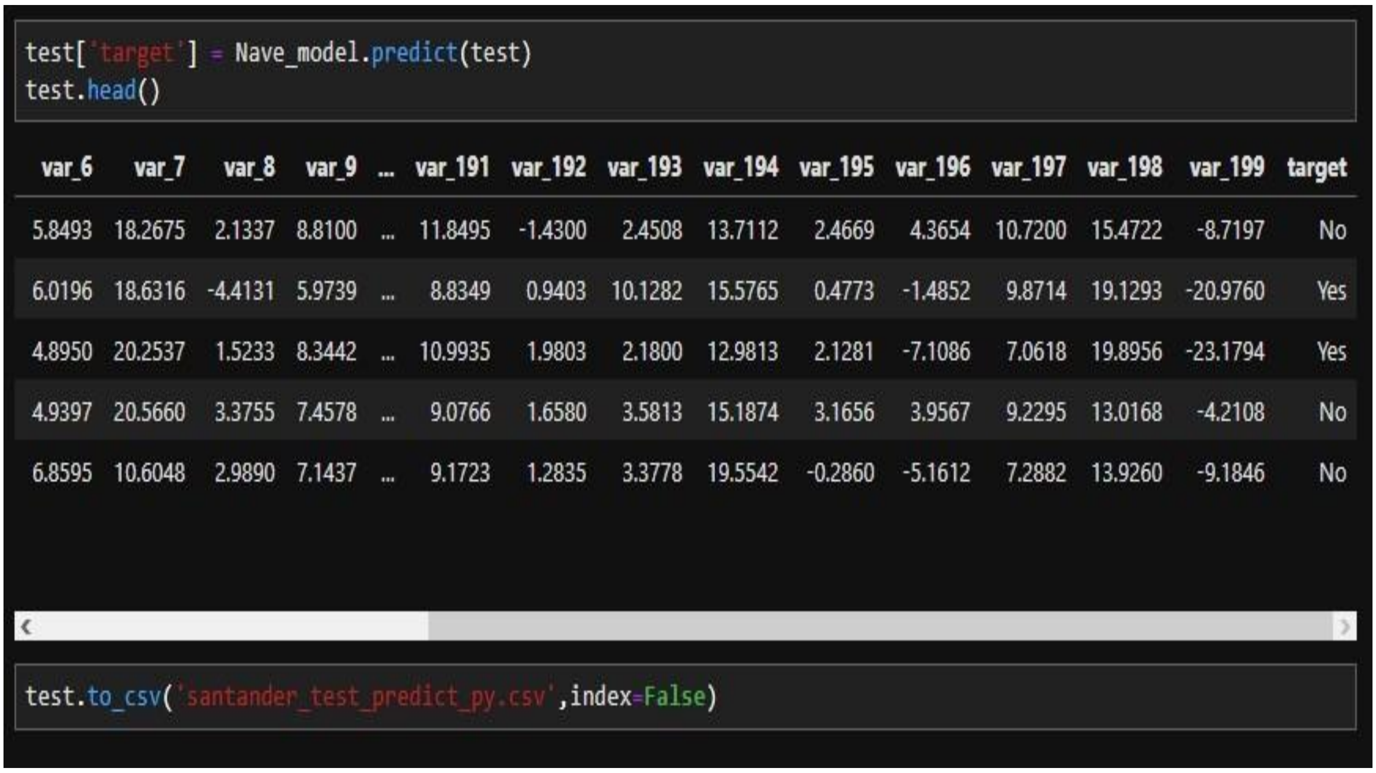
	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	...	var_190	var_191	var_192	var_19
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	...	-2.1556	11.8495	-1.4300	2.45
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	...	10.6165	8.8349	0.9403	10.12
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	2.18
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	3.58
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	3.37

5 rows × 201 columns



```
# Drop the ID_code column from the Dataset as our model is not trained for it
test = test.drop(['ID_code'], axis=1)
test.head(5)
```

```
      var_0     var_1    var_2   var_3    var_4   var_5   var_6    var_7   var_8   var_9  ...  var_190  var_191  var_192  var_193
0   11.0656    7.7798  12.9536  9.4292  11.4327 -2.3805  5.8493  18.2675  2.1337  8.8100  ...  -2.1556  11.8495  -1.4300   2.4508
1    8.5304    1.2543  11.3047  5.1858   9.1974 -4.0117  6.0196  18.6316 -4.4131  5.9739  ...  10.6165   8.8349   0.9403  10.1282
2    5.4827  -10.3581  10.1407  7.0479  10.2628  9.8052  4.8950  20.2537  1.5233  8.3442  ...  -0.7484  10.9935   1.9803   2.1800
3    8.5374   -1.3222  12.0220  6.5749   8.8458  3.1744  4.9397  20.5660  3.3755  7.4578  ...   9.5702   9.0766   1.6580   3.5813
4   11.7058   -0.1327  14.1295  7.7506   9.1035 -8.5848  6.8595  10.6048  2.9890  7.1437  ...   4.2259   9.1723   1.2835   3.3778

5 rows × 200 columns
```



```
test['target'] = Nave_model.predict(test)
test.head()
```

```
 var_6    var_7    var_8   var_9  ...  var_191  var_192  var_193  var_194  var_195  var_196  var_197  var_198   var_199  target
5.8493  18.2675   2.1337  8.8100  ...  11.8495  -1.4300   2.4508  13.7112   2.4669   4.3654  10.7200  15.4722   -8.7197      No
6.0196  18.6316  -4.4131  5.9739  ...   8.8349   0.9403  10.1282  15.5765   0.4773  -1.4852   9.8714  19.1293  -20.9760     Yes
4.8950  20.2537   1.5233  8.3442  ...  10.9935   1.9803   2.1800  12.9813   2.1281  -7.1086   7.0618  19.8956  -23.1794     Yes
4.9397  20.5660   3.3755  7.4578  ...   9.0766   1.6580   3.5813  15.1874   3.1656   3.9567   9.2295  13.0168   -4.2108      No
6.8595  10.6048   2.9890  7.1437  ...   9.1723   1.2835   3.3778  19.5542  -0.2860  -5.1612   7.2882  13.9260   -9.1846      No
```

```
test.to_csv('santander_test_predict_py.csv',index=False)
```

Conclusion

- Santander is interested in finding which customers will make a specific transaction in the future, irrespective of the amount of money transacted.
- Hence, it is interested in correctly identifying the customers with target label as 1, (i.e. customers who will make a specific transaction in the future)
- Since our dataset is an imbalance class dataset, where the proportion of positive samples is low **(around 10%)**, we should aim for **higher precision since it does not include True negatives in calculation, and hence it will not affected by class imbalance.**

PYTHON CODE

```
import numpy as np
import pandas as pd
import csv
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
import os
#set working directory-
os.chdir("H:\data scientist\projects\Santender_Project-master")

#check current working directory-
os.getcwd()
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
train.head()
train.info()
train.columns
train.shape
train.describe()
miss_train = pd.DataFrame(train.isnull().sum())
np.transpose(miss_train)           #for better visibility and no missing values throughout
miss_test = pd.DataFrame(test.isnull().sum())
np.transpose(miss_test)
#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
numerical_features=train.columns[2:]
from scipy.special import erfc
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, 20), kind='barh').set_xlabel('Number of outliers');
print('Total number of outliers in training set: {} ({:.2f}%)'.format(sum(train_outliers.values), (sum(train_outliers.values) /
train.shape[0]) * 100))
#outliers in each variable in test data
test_outliers = dict()
for col in [col for col in numerical_features]:
    test_outliers[col] = test[chauvenet(test[col].values)].shape[0]
test_outliers = pd.Series(test_outliers)

test_outliers.sort_values().plot(figsize=(14, 40), kind='barh').set_xlabel('Number of outliers');
print('Total number of outliers in testing set: {} ({:.2f}%)'.format(sum(test_outliers.values), (sum(test_outliers.values) /
test.shape[0]) * 100))
#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))]
def plot_feature_distribution(df1, df2, label1, label2, features):
    i = 0
    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(10,10,figsize=(18,22))

    for feature in features:
        i += 1
        plt.subplot(10,10,i)
        sns.kdeplot(df1[feature], bw=0.5,label=label1)
        sns.kdeplot(df2[feature], bw=0.5,label=label2)
        plt.xlabel(feature, fontsize=9)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', which='major', labelsize=6, pad=-6)
        plt.tick_params(axis='y', which='major', labelsize=6)
    plt.show()
import seaborn as sns
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
features = train.columns.values[2:102]
plot_feature_distribution(t0, t1, '0', '1', features)
# distribution of the mean values per row in the train and test set.
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of mean values per row in the train and test set")
sns.distplot(train[features].mean(axis=1),color="green", kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=1),color="blue", kde=True,bins=120, label='test')
plt.legend()
plt.show()
#distribution of the mean values per columns in the train and test set.

plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per column in the train and test set")
sns.distplot(train[features].mean(axis=0),color="black",kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=0),color="darkblue", kde=True,bins=120, label='test')
plt.legend()
plt.show()
# distribution of standard deviation of values per row for train and test datasets.

plt.figure(figsize=(16,6))
plt.title("Distribution of std values per row in the train and test set")
sns.distplot(train[features].std(axis=1),color="black", kde=True,bins=120, label='train')
sns.distplot(test[features].std(axis=1),color="red", kde=True,bins=120, label='test')
plt.legend();plt.show()
# distribution of the standard deviation of values per columns in the train and test datasets.

plt.figure(figsize=(16,6))
plt.title("Distribution of std values per column in the train and test set")
sns.distplot(train[features].std(axis=0),color="blue",kde=True,bins=120, label='train')
sns.distplot(test[features].std(axis=0),color="green", kde=True,bins=120, label='test')
plt.legend(); plt.show()
plt.figure(figsize=(16,6))
plt.title("Distribution of skew per row in the train and test set")
sns.distplot(train[features].skew(axis=1),color="red", kde=True,bins=120, label='train')
sns.distplot(test[features].skew(axis=1),color="orange", kde=True,bins=120, label='test')
plt.legend()
plt.show()
plt.figure(figsize=(16,6))
plt.title("Distribution of skew per column in the train and test set")
sns.distplot(train[features].skew(axis=0),color="magenta", kde=True,bins=120, label='train')
sns.distplot(test[features].skew(axis=0),color="darkblue", kde=True,bins=120, label='test')
plt.legend()
plt.show()
correlations = train[features].corr().abs().unstack().sort_values(kind="quicksort").reset_index()
correlations = correlations[correlations['level_0'] != correlations['level_1']]
correlations.head(10)
correlations.tail(10)
sns.set(rc={'figure.figsize':(20,10)})

# Compute the correlation matrix
correlations = train[features].corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(correlations, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

sns.heatmap(correlations, mask=mask,
            #annot=True,
            #fmt=".2f",
            cmap='coolwarm')
import seaborn as sns
#count of both class(number of classes)
train['target'].value_counts()
sns.factorplot('target', data=train, kind='count')
#IN OUR CASE GIVEN DATA IS IMBALANCED……WHERE 90% OF SAMPLES BELONGS TO CLASS 0 AND ONLY
10% BELONGS TO CLASS 1
train.shape
#WE seperate the dataset whose target class is belong to class 0
data=train.loc[train['target'] == 0]
#choose starting 30000 rows
data2=data.loc[:30000]
data2
#WE seperate the dataset whose target class is belong to class 1
data1=train.loc[train['target'] == 1]
data1
#Add both Dataframe data1 and data2 in one dataframe
newdata=pd.concat([data1, data2], ignore_index=True)
newdata
#Shuffle the Dataframe
newdata=newdata.sample(frac=1)
newdata
sns.factorplot('target', data=newdata, kind='count')
#After applying undersampling technique,we get the balanced data as shown above
#import libraries
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
plt.rc("font", size=14)
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.model_selection import cross_val_score
# Replace target variable categories with Yes or No
newdata['target'] = newdata['target'].replace(0, 'No')
newdata['target'] = newdata['target'].replace(1, 'Yes')

# Converted entries of 'target' variable from o/1 to No/Yes
newdata.head(5)
# Divide data into train and test for model Development
```

```
X = newdata.values[:, 2:202]
Y = newdata.values[:,1]

X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.2)
# Decision tree

C50_model = tree.DecisionTreeClassifier(criterion='entropy').fit(X_train, y_train)

#test scores

C50_predictions = C50_model.predict(X_test)
C50_predictions
#confusion matrix

CM = pd.crosstab(y_test, C50_predictions)

#let check for belw values
TN = CM.iloc[0,0]
FN = CM.iloc[1,0]
TP = CM.iloc[1,1]
FP = CM.iloc[0,1]

CM
from sklearn.metrics import classification_report
print(classification_report(y_test,C50_predictions))

from sklearn import metrics

print('Accuracy:', metrics.accuracy_score(y_test, C50_predictions)*100)
# Random Forest
# Import Libraries
from sklearn.ensemble import RandomForestClassifier

# Develop and train random forest model
RF_model = RandomForestClassifier(n_estimators = 10).fit(X_train, y_train)

# Predict new test cases
RF_Predictions = RF_model.predict(X_test)
RF_Predictions
# Build confusion matrix
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test, RF_Predictions)

# To get confusion matrix in tabular form
CM = pd.crosstab(y_test, RF_Predictions)

CM
from sklearn.metrics import classification_report
print(classification_report(y_test,RF_Predictions))
from sklearn import metrics

print('Accuracy:', metrics.accuracy_score(y_test, RF_Predictions)*100)
# logistic regression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

#instantiate the model

logreg = LogisticRegression()

logreg.fit(X_train, y_train)
# Predict new test cases
```

```
log_Pred = logreg.predict(X_test)
log_Pred
# Build confusion matrix
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test, log_Pred)

# To get confusion matrix in tabular form
CM = pd.crosstab(y_test, log_Pred)

CM
from sklearn.metrics import classification_report
print(classification_report(y_test,log_Pred))
from sklearn import metrics

print('Accuracy:', metrics.accuracy_score(y_test, log_Pred)*100)
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
#implementation
Nave_model = GaussianNB().fit(X_train, y_train)
NB_Predictions= Nave_model.predict(X_test)
NB_Predictions
# Build confusion matrix
from sklearn.metrics import confusion_matrix
CM = confusion_matrix(y_test, NB_Predictions)
CM = pd.crosstab(y_test, NB_Predictions)
CM
from sklearn.metrics import classification_report
print(classification_report(y_test,NB_Predictions))
from sklearn import metrics
print('Accuracy:', metrics.accuracy_score(y_test, NB_Predictions)*100)
```

# SUMMARY
"To get the most accurate model out of various models the value of recall, precision, AUC should be high".
As per the directions of our project we have to predict the results based on recall, precision and accuracy
of all machine learning algorithm. Out of all the above developed Machine Learning algorithms we can deduce
that "Naive Bayes" is giving all the quantities highest among all other algorithms. Hence "Naive Bayes" is
selected to predict target variable from our given test data.

#### FINDING THE TARGET VALUE OF TEST DATA
* Now since we have tested all the Machine Learning Algorithms and Statistical Models on our Training Data and retrieved the accuracy from each model
* So we choose Naive Bayes method of Supervised Learning to predict the value of our target variable on our test Data

```
# Load Data testing data
test = pd.read_csv('test.csv')
test.shape

test.head()
# Drop the ID_code column from the Dataset as our model is not trained for it
test = test.drop(['ID_code'], axis=1)
test.head(5)
test['target'] = Nave_model.predict(test)
test.head()
test.to_csv('santander_test_predict_py.csv',index=False)
```