Name: MD Ekram Ullah
Batch: ML-DS 102

1. Problem Statement:

Given the details of customers in a bank, the aim is to improve bank marketing s strategies by analyzing their past marketing strategies and recommending which customers to target.

2. Appproach:

First of all, I loaded the data in jupyter notebook and put a detailed look into the data. I performed EDA vastly to look into categorical and numerical feature distribution, relationship between categorical features and label, relationship between discrete numerical features and label and also continuous numerical features and label, pair plot and so on. After getting a clear idea of my data, I performed Feature engineering into the dataset. Where I dropped the unwanted features, handled the missing values, handles categorical features, feature scaling and removed the outliers. Then after getting a clean dataset, I splitted the dataset into training and test set and then I trained the models.

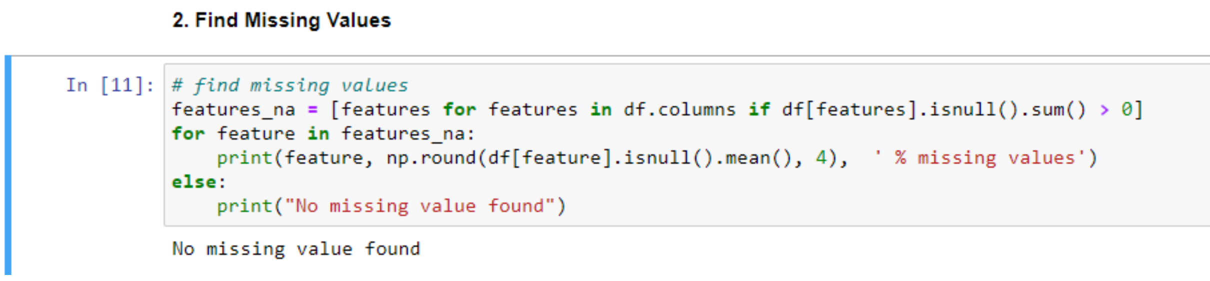
3. Phases:

i) Exploratory Data Analysis:

I did my EDA task vastly with so many steps in it which are described below:

a) **Find Missing Values:**

Here I tried to find the missing values feature by feature. But found no missing values.

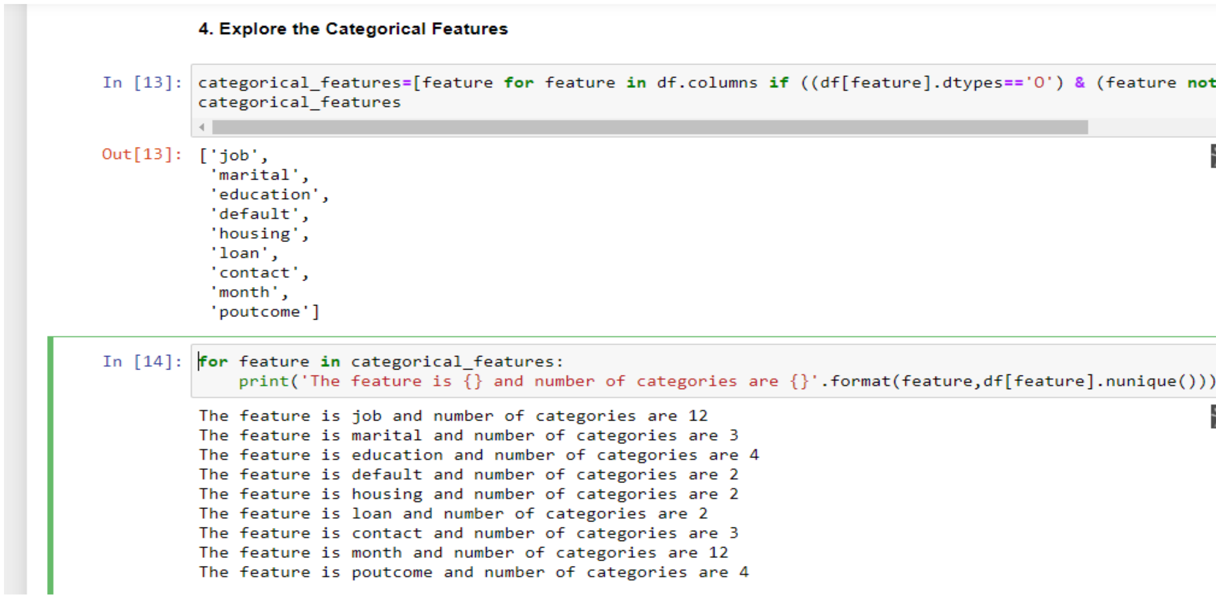


2. Find Missing Values

```
In [11]: # find missing values
         features_na = [features for features in df.columns if df[features].isnull().sum() > 0]
         for feature in features_na:
             print(feature, np.round(df[feature].isnull().mean(), 4),  ' % missing values')
         else:
             print("No missing value found")

         No missing value found
```

b) **Explore the Categorical Features:**

Here I looked into details of categorical features.



4. Explore the Categorical Features

```
In [13]: categorical_features=[feature for feature in df.columns if ((df[feature].dtypes=='O') & (feature not
         categorical_features
```

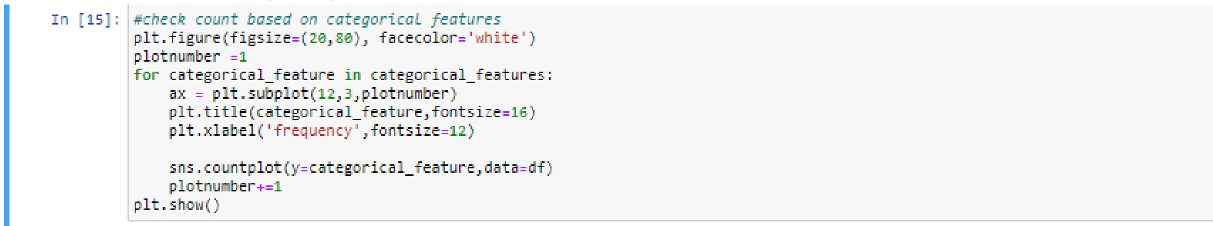
```
Out[13]: ['job',
          'marital',
          'education',
          'default',
          'housing',
          'loan',
          'contact',
          'month',
          'poutcome']
```

```
In [14]: for feature in categorical_features:
             print('The feature is {} and number of categories are {}'.format(feature,df[feature].nunique()))
```

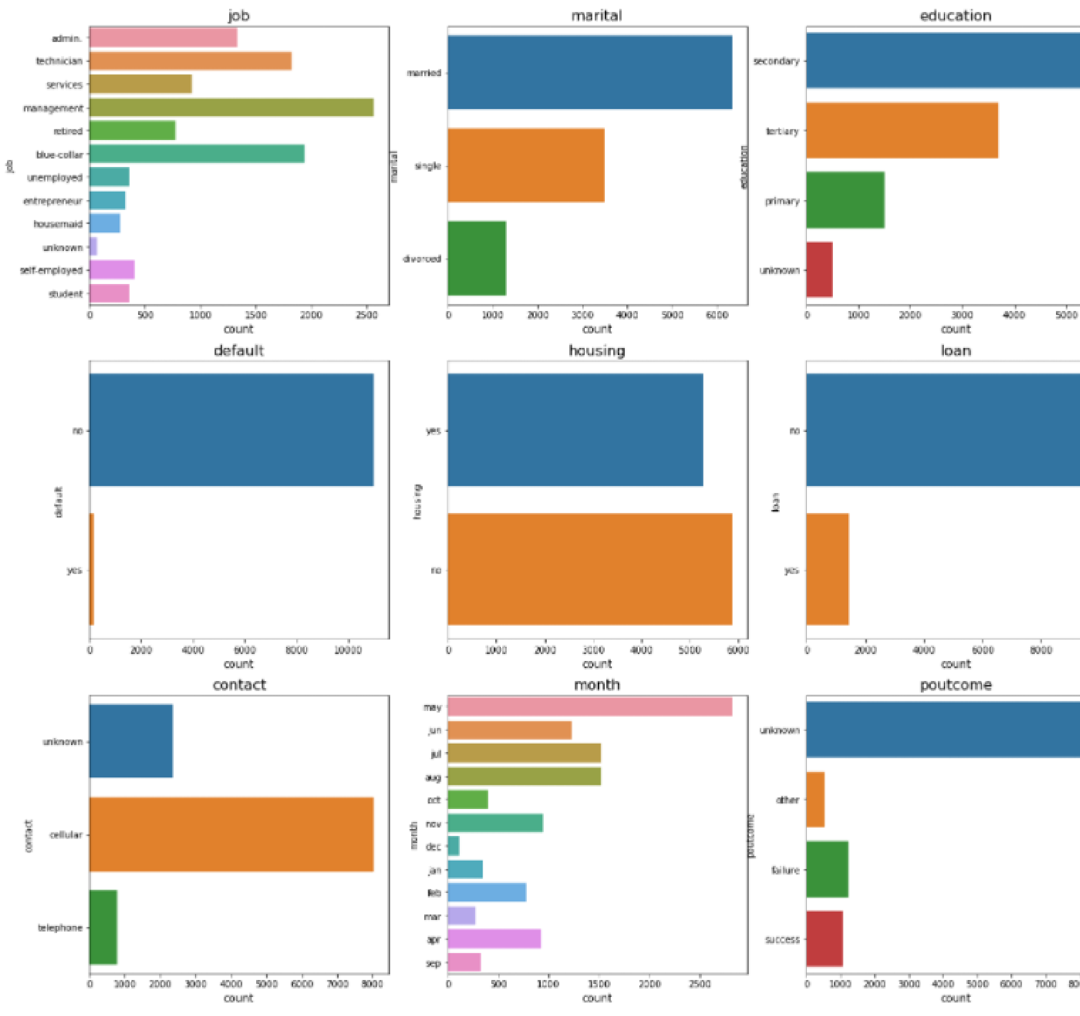
```
The feature is job and number of categories are 12
The feature is marital and number of categories are 3
The feature is education and number of categories are 4
The feature is default and number of categories are 2
The feature is housing and number of categories are 2
The feature is loan and number of categories are 2
The feature is contact and number of categories are 3
The feature is month and number of categories are 12
The feature is poutcome and number of categories are 4
```

c) **Find Categorical Feature Distribution:**

I looked into the distribution of every categorical feature , how there unique values are distributed and the frequency.

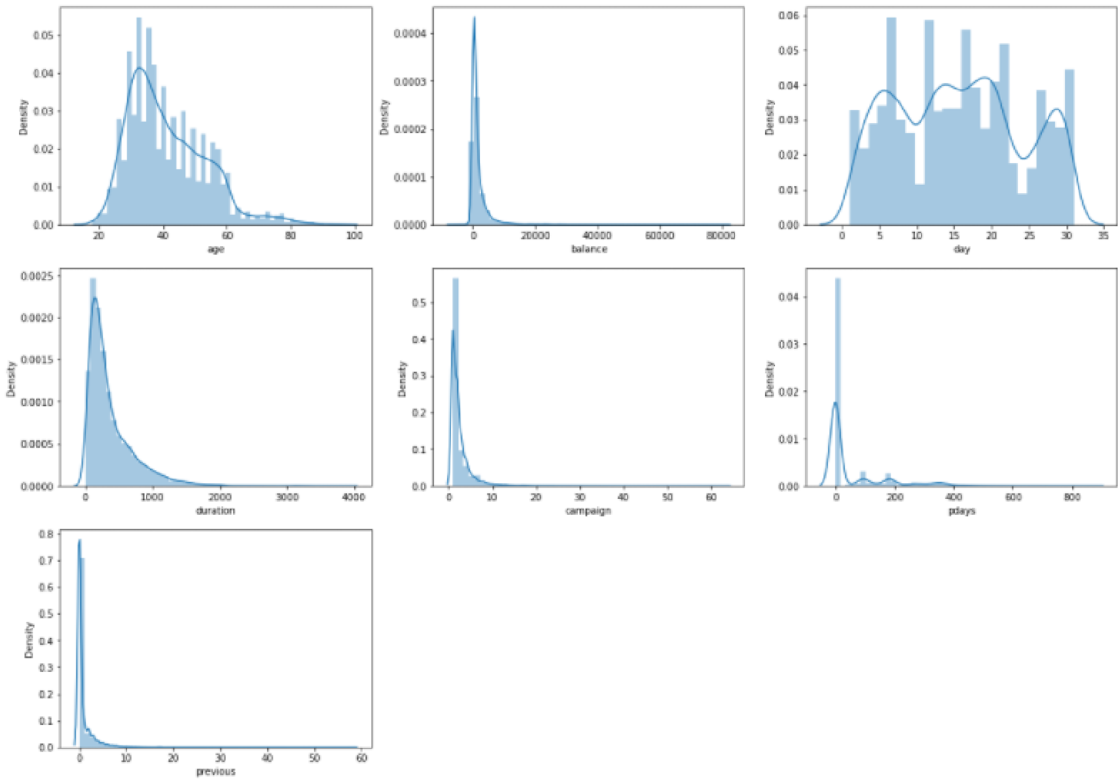


```
In [15]: #check count based on categorical features
         plt.figure(figsize=(20,80), facecolor='white')
         plotnumber =1
         for categorical_feature in categorical_features:
             ax = plt.subplot(12,3,plotnumber)
             plt.title(categorical_feature,fontsize=16)
             plt.xlabel('frequency',fontsize=12)

             sns.countplot(y=categorical_feature,data=df)
             plotnumber+=1
         plt.show()
```



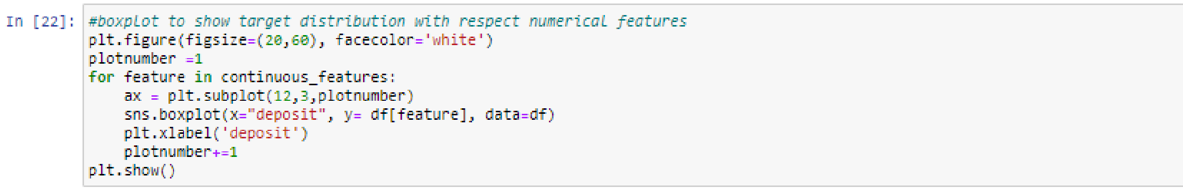
d) **Distribution of Continuous Numerical Features:**

```
In [21]: #plot a univariate distribution of continues observations
         plt.figure(figsize=(20,60), facecolor='white')
         plotnumber =1
         for continuous_feature in continuous_features:
             ax = plt.subplot(12,3,plotnumber)
             sns.distplot(df[continuous_feature])
             plt.xlabel(continuous_feature)
             plotnumber+=1
         plt.show()
```

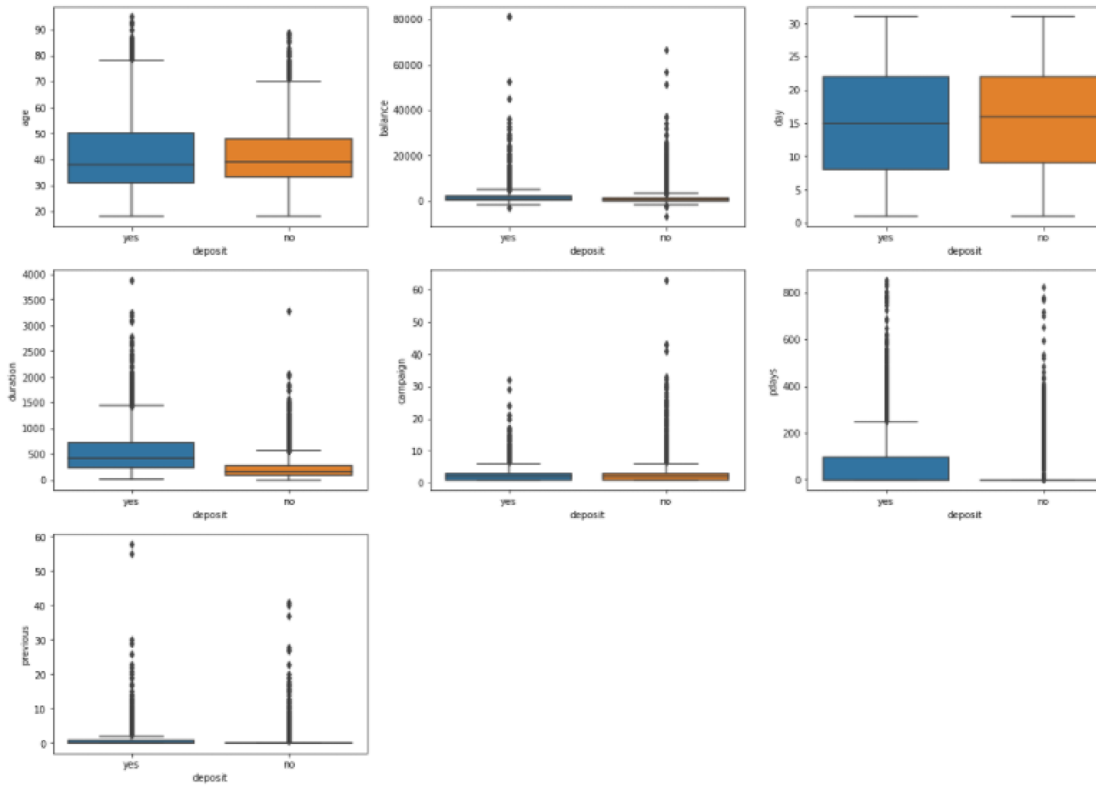


e) **Relation between Continous numerical Features and Labels:**

I tried to look the relation between each feature and label.

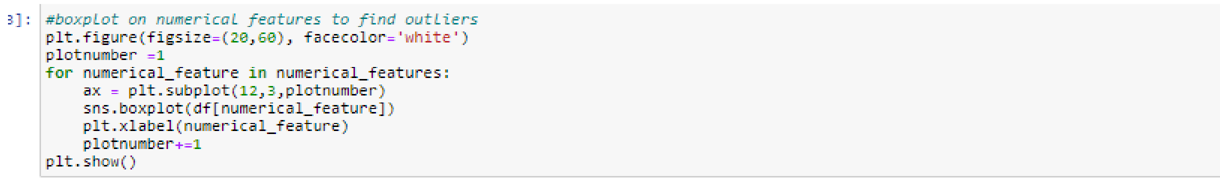


```
In [22]: #boxplot to show target distribution with respect numerical features
         plt.figure(figsize=(20,60), facecolor='white')
         plotnumber =1
         for feature in continuous_features:
             ax = plt.subplot(12,3,plotnumber)
             sns.boxplot(x="deposit", y= df[feature], data=df)
             plt.xlabel('deposit')
             plotnumber+=1
         plt.show()
```

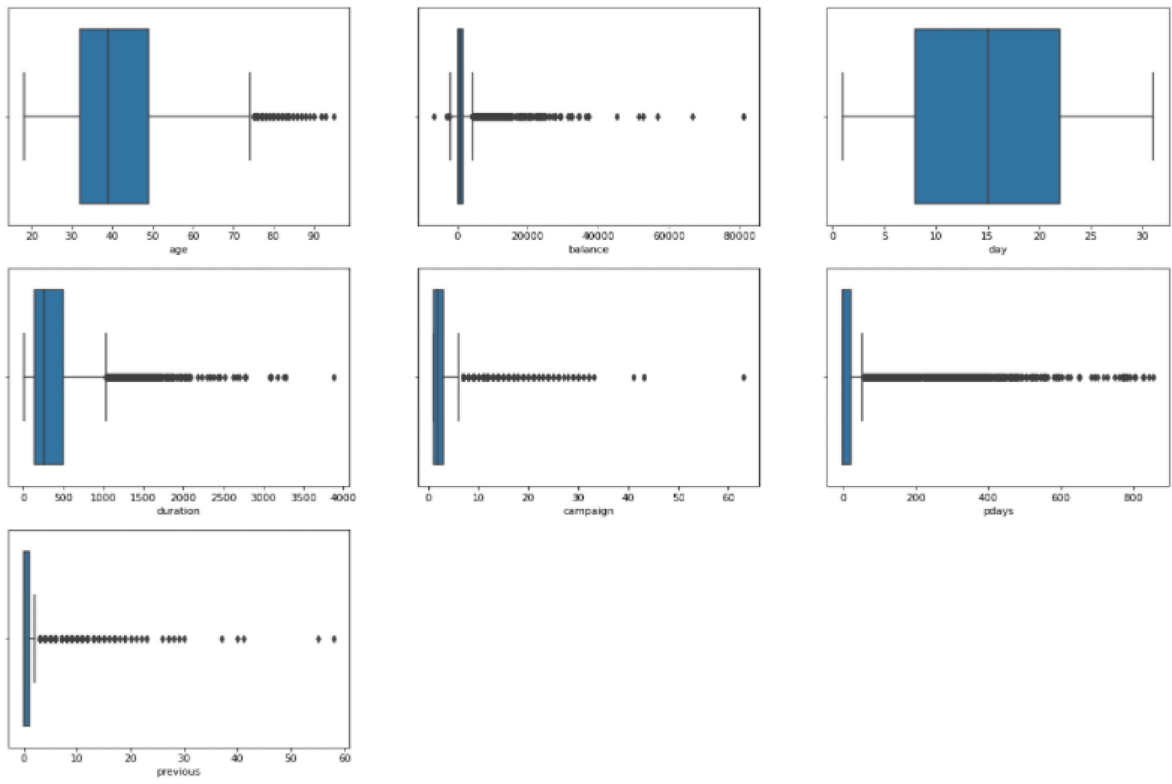


f) **Find Outliers in numerical features:**

Here I have plotted the outliers of the numerical features through boxplot.



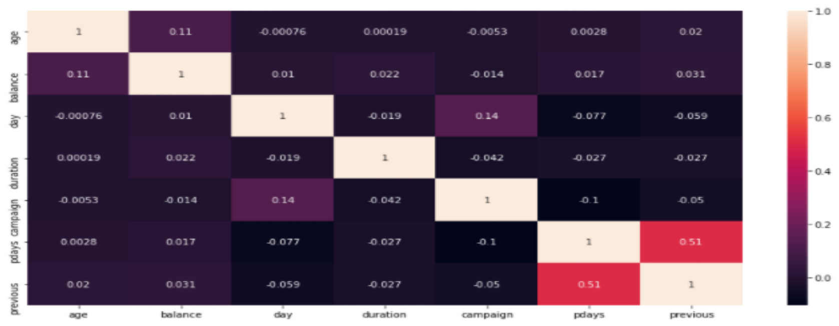
```
3]: #boxplot on numerical features to find outliers
    plt.figure(figsize=(20,60), facecolor='white')
    plotnumber =1
    for numerical_feature in numerical_features:
        ax = plt.subplot(12,3,plotnumber)
        sns.boxplot(df[numerical_feature])
        plt.xlabel(numerical_feature)
        plotnumber+=1
    plt.show()
```



g) **Explore the Correlation between numerical features:**

Here I have plotted the correlation between the numerical features through heatmap

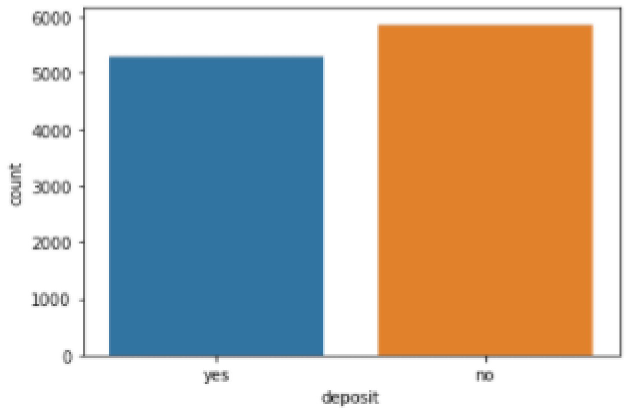
```
In [24]: ## Checking for correlation
         cor_mat=df.corr()
         fig = plt.figure(figsize=(15,7))
         sns.heatmap(cor_mat,annot=True)

Out[24]: <AxesSubplot:>
```



	age	balance	day	duration	campaign	pdays	previous
age	1	0.11	-0.00076	0.00019	-0.0053	0.0028	0.02
balance	0.11	1	0.01	0.022	-0.014	0.017	0.031
day	-0.00076	0.01	1	-0.019	0.14	-0.077	-0.059
duration	0.00019	0.022	-0.019	1	-0.042	-0.027	-0.027
campaign	-0.0053	-0.014	0.14	-0.042	1	-0.1	-0.05
pdays	0.0028	0.017	-0.077	-0.027	-0.1	1	0.51
previous	0.02	0.031	-0.059	-0.027	-0.05	0.51	1

h) **Check the Data set is balanced or not based on target values in classification:**

```
In [25]: #total patient count based on cardio_results
         sns.countplot(x='deposit',data=df)
         plt.show()
```

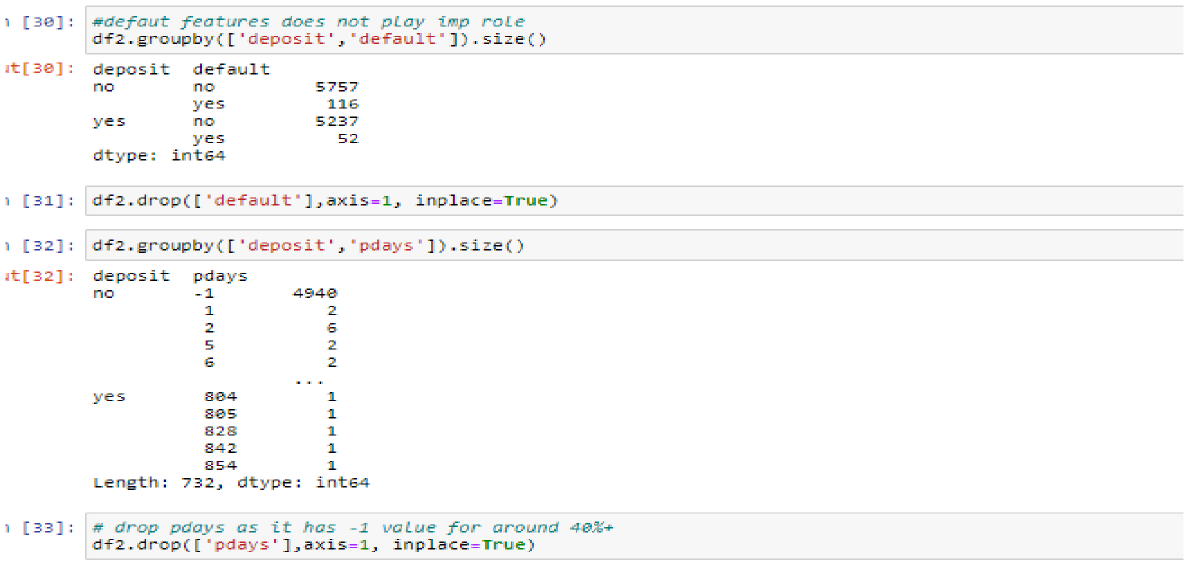


```
In [26]: df['deposit'].groupby(df['deposit']).count()

Out[26]: deposit
         no     5873
         yes    5289
         Name: deposit, dtype: int64
```

ii) Feature Engineering:

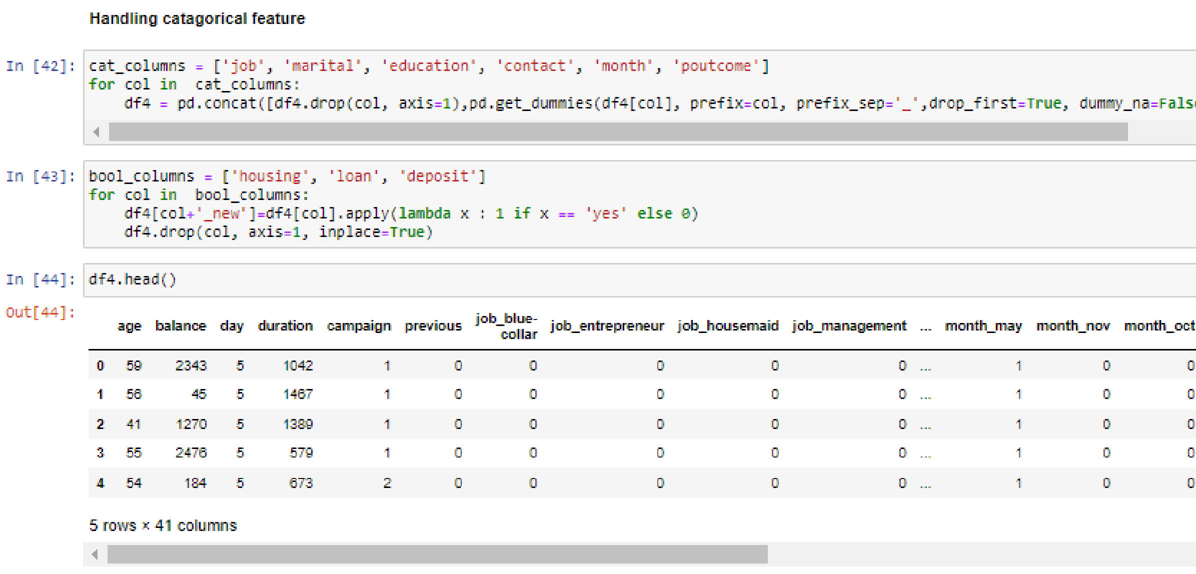
In this stage , I dropped unwanted features , handled the missing values,categorical features, feature scaling and also removed the outliers.

a) Drop unwanted Features:



```
[30]: #defaut features does not play imp role
      df2.groupby(['deposit','default']).size()

t[30]: deposit  default
       no       no         5757
                yes         116
       yes      no         5237
                yes          52
       dtype: int64

[31]: df2.drop(['default'],axis=1, inplace=True)

[32]: df2.groupby(['deposit','pdays']).size()

t[32]: deposit  pdays
       no       -1       4940
                 1          2
                 2          6
                 5          2
                 6          2
                         ...
       yes       804        1
                 805        1
                 828        1
                 842        1
                 854        1
       Length: 732, dtype: int64

[33]: # drop pdays as it has -1 value for around 40%+
      df2.drop(['pdays'],axis=1, inplace=True)
```

b) Handle Categorical Features



Handling catagorical feature

```
In [42]: cat_columns = ['job', 'marital', 'education', 'contact', 'month', 'poutcome']
         for col in  cat_columns:
             df4 = pd.concat([df4.drop(col, axis=1),pd.get_dummies(df4[col], prefix=col, prefix_sep='_',drop_first=True, dummy_na=Fals
```

```
In [43]: bool_columns = ['housing', 'loan', 'deposit']
         for col in  bool_columns:
             df4[col+'_new']=df4[col].apply(lambda x : 1 if x == 'yes' else 0)
             df4.drop(col, axis=1, inplace=True)
```

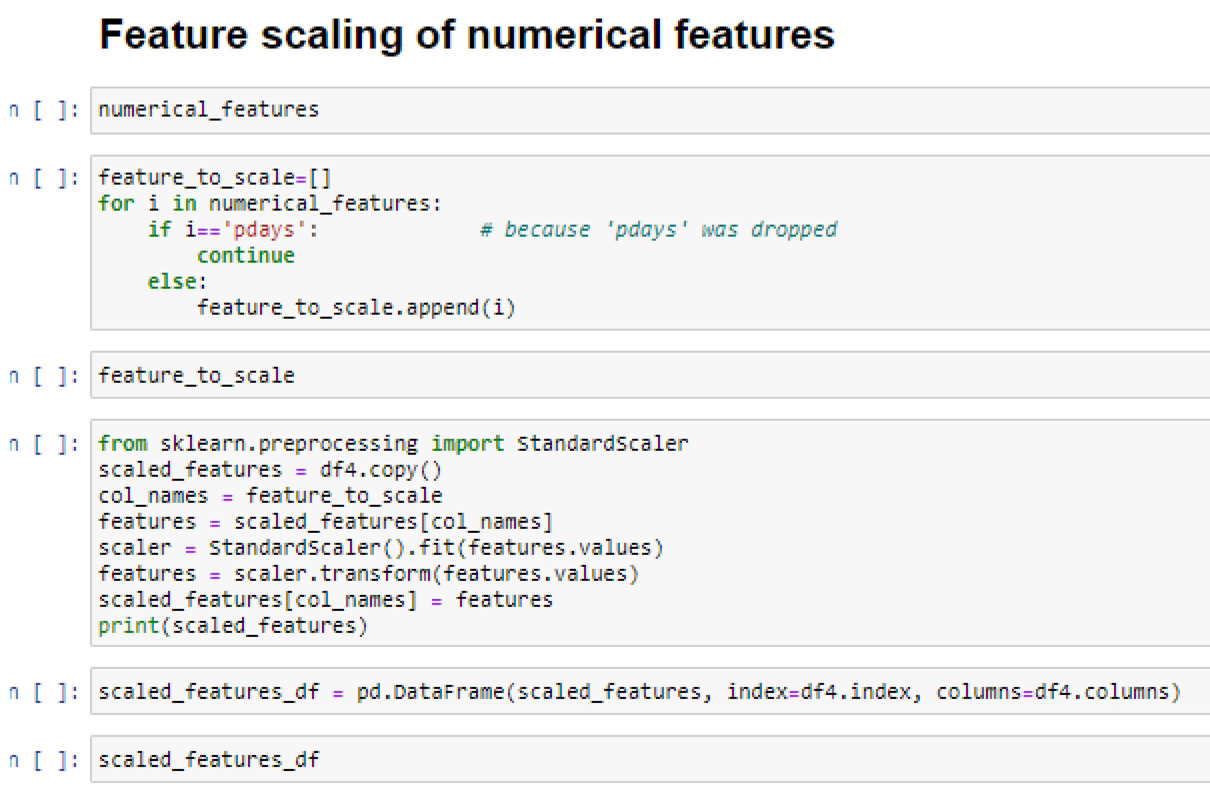
```
In [44]: df4.head()
```

Out[44]:

	age	balance	day	duration	campaign	previous	job_blue-collar	job_entrepreneur	job_housemaid	job_management	...	month_may	month_nov	month_oct
0	59	2343	5	1042	1	0	0	0	0	0	...	1	0	0
1	56	45	5	1467	1	0	0	0	0	0	...	1	0	0
2	41	1270	5	1389	1	0	0	0	0	0	...	1	0	0
3	55	2476	5	579	1	0	0	0	0	0	...	1	0	0
4	54	184	5	673	2	0	0	0	0	0	...	1	0	0

5 rows × 41 columns

c) Handle Feature Scaling:



Feature scaling of numerical features

```
n [ ]: numerical_features
```

```
n [ ]: feature_to_scale=[]
       for i in numerical_features:
           if i=='pdays':              # because 'pdays' was dropped
               continue
           else:
               feature_to_scale.append(i)
```

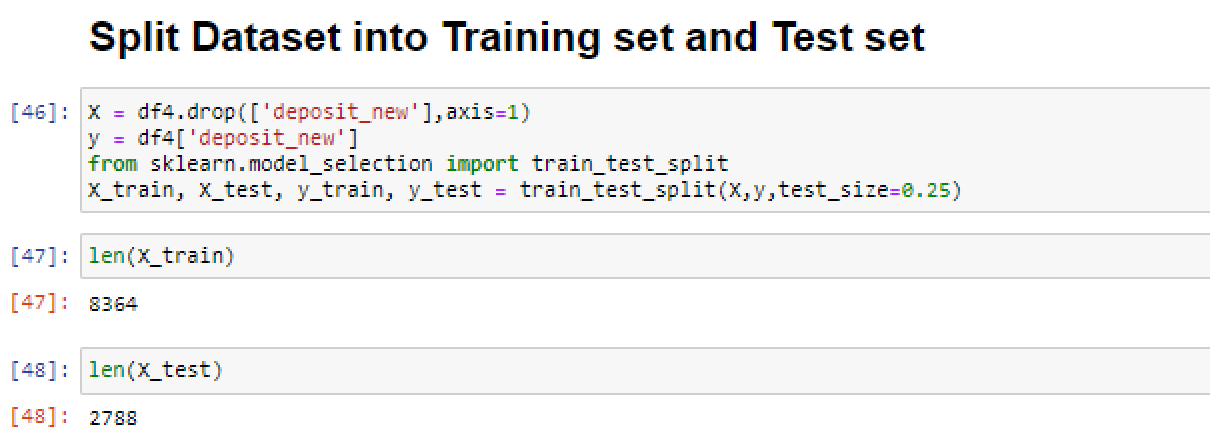
```
n [ ]: feature_to_scale
```

```
n [ ]: from sklearn.preprocessing import StandardScaler
       scaled_features = df4.copy()
       col_names = feature_to_scale
       features = scaled_features[col_names]
       scaler = StandardScaler().fit(features.values)
       features = scaler.transform(features.values)
       scaled_features[col_names] = features
       print(scaled_features)
```

```
n [ ]: scaled_features_df = pd.DataFrame(scaled_features, index=df4.index, columns=df4.columns)
```

```
n [ ]: scaled_features_df
```

iii) Train test split:



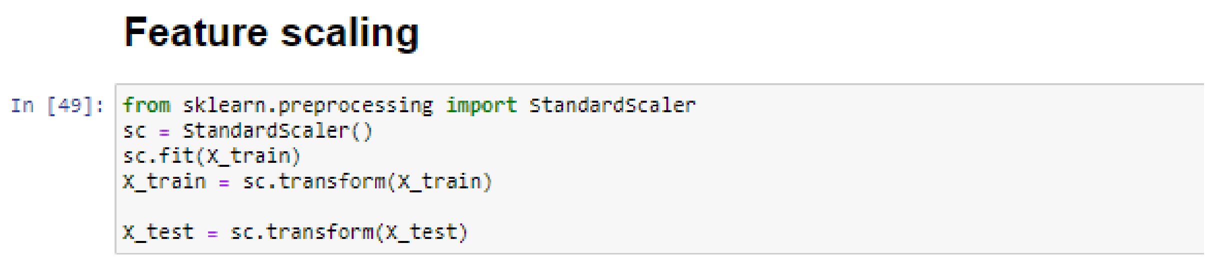
Split Dataset into Training set and Test set

```
[46]: X = df4.drop(['deposit_new'],axis=1)
      y = df4['deposit_new']
      from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25)
```

```
[47]: len(X_train)

[47]: 8364
```

```
[48]: len(X_test)

[48]: 2788
```

iv) Feature Scaling:



Feature scaling

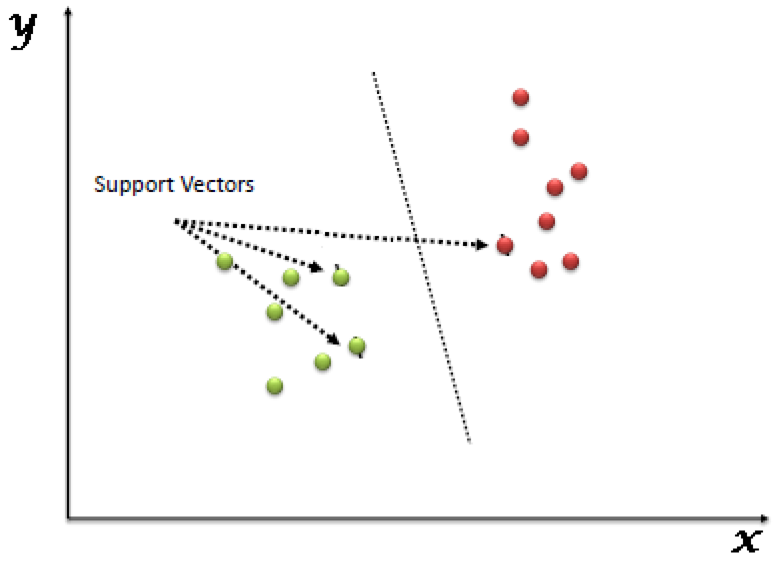
```
In [49]: from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         sc.fit(X_train)
         X_train = sc.transform(X_train)

         X_test = sc.transform(X_test)
```

4.Modelling:

i) SVM:

Definition:

“Support Vector Machine” (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.



Performance Metrics:

Accuracy: The accuracy of a machine learning classification algorithm is one way to measure how often the algorithm classifies a data point correctly. Accuracy is the number of correctly predicted data points out of all the data points.

```
Here in SVM model, Accuracy is 0.8238880918220947
```

Precision: Precision attempts to answer what proportion of positive identifications was actually correct. Precision is defined as follows:

$$precision = \frac{tp}{tp + fp}$$

```
Here in SVM model, Precision  is 0.8262518968133535
```

Recall: Recall attempts to answer what proportion of actual positives was identified correctly. Mathematically, recall is defined as follows:

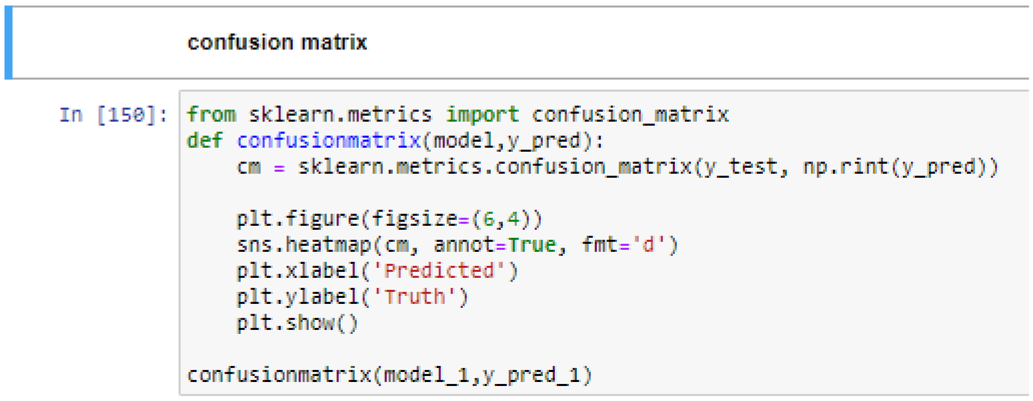
$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

```
 Here in SVM model, Recall  is 0.8060695780903034
```

Confusion Matrix:

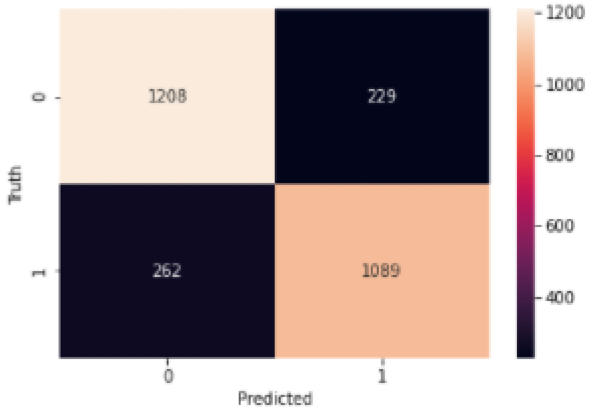
A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model.

Here in SVM Model confusion matrix is:



confusion matrix

```
In [150]: from sklearn.metrics import confusion_matrix
          def confusionmatrix(model,y_pred):
              cm = sklearn.metrics.confusion_matrix(y_test, np.rint(y_pred))

              plt.figure(figsize=(6,4))
              sns.heatmap(cm, annot=True, fmt='d')
              plt.xlabel('Predicted')
              plt.ylabel('Truth')
              plt.show()

          confusionmatrix(model_1,y_pred_1)
```

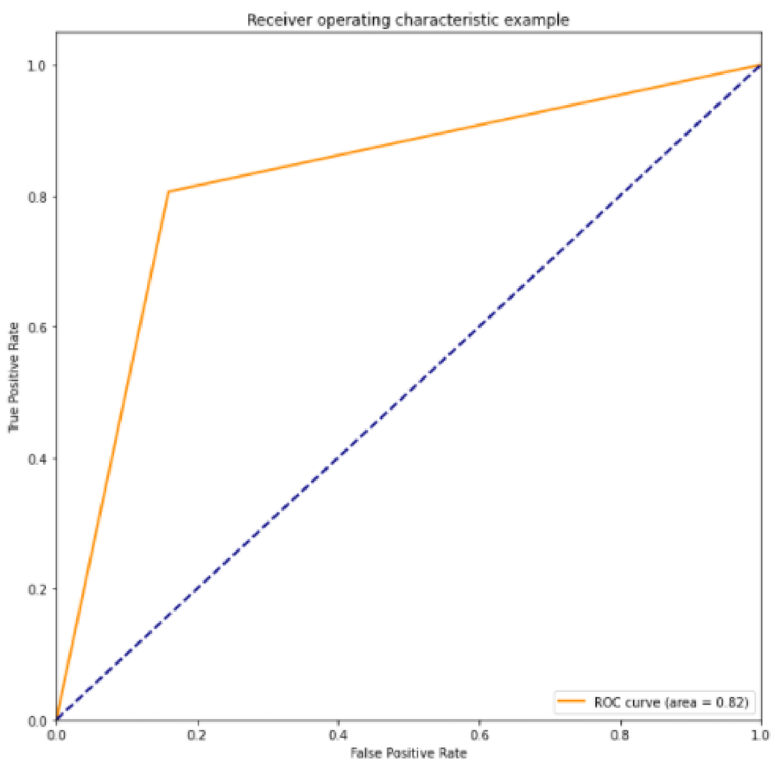


<u>ROC Curve</u>:

An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

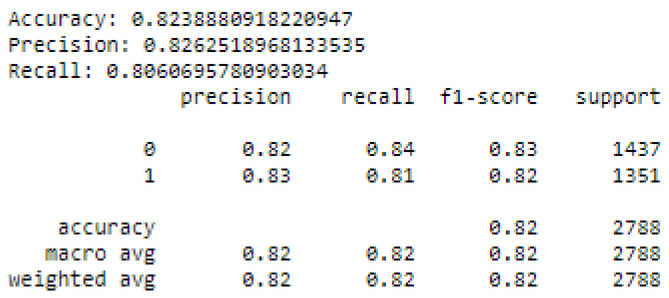
- True Positive Rate
- False Positive Rate

Here in SVM model, ROC Curve is represented as:



Where area under curve (AUC) is .82 which is quite good.

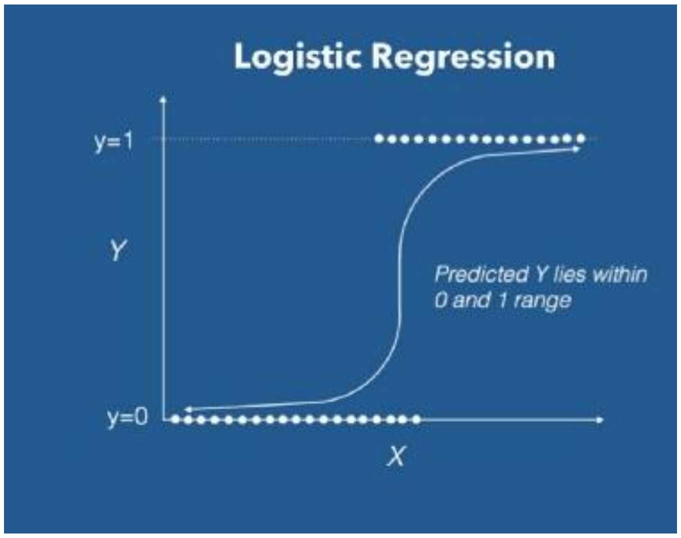
Screenshot of the output:



```
Accuracy: 0.8238880918220947
Precision: 0.8262518968133535
Recall: 0.8060695780903034
              precision    recall  f1-score   support

           0       0.82      0.84      0.83      1437
           1       0.83      0.81      0.82      1351

    accuracy                           0.82      2788
   macro avg       0.82      0.82      0.82      2788
weighted avg       0.82      0.82      0.82      2788
```

ii) Logistic Regression:

Definition: Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability. We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function**' or also known as the 'logistic function' instead of a linear function.



Performance Metrics:

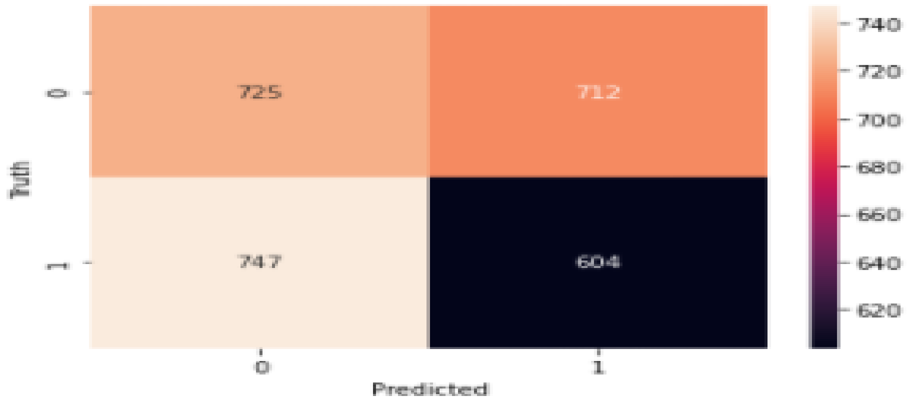
Accuracy : `The accuracy of this model is: 0.833931133428`

Precision: `The precision of this model is: 0.8404255319`

Recall: `The recall of this model is: 0.8138337012`

Confusion Matrix:

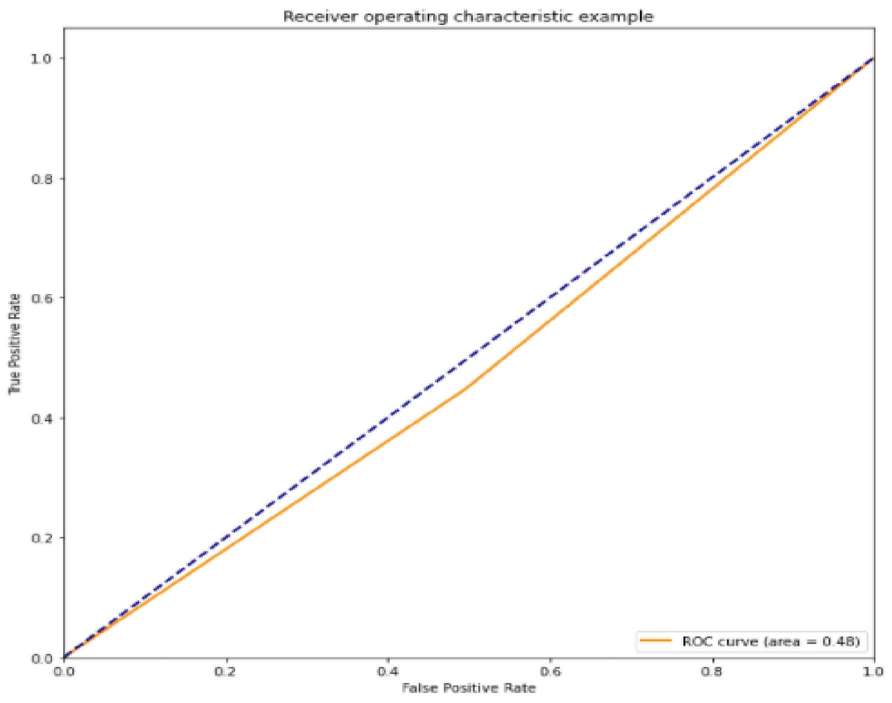
```
[151]:  confusionmatrix(model_2,y_pred_2)
```



ROC Curve:

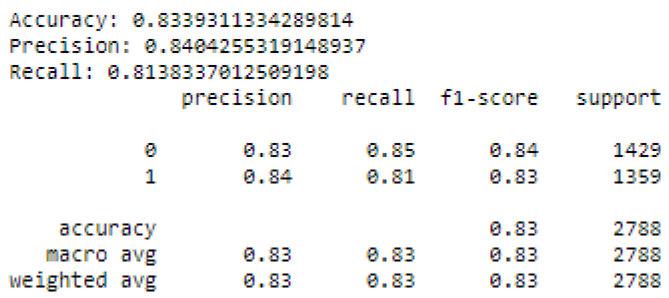
```
ROC_curve(y_pred_2)
```

```
[0.         0.49547669 1.        ]
[0.         0.44707624 1.        ]
[2 1 0]

<Figure size 432x288 with 0 Axes>
```



The AUC here is .48 which is a poor value

Screenshot of the output:


```
Accuracy: 0.8339311334289814
Precision: 0.8404255319148937
Recall: 0.8138337012509198
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      1429
           1       0.84      0.81      0.83      1359

    accuracy                           0.83      2788
   macro avg       0.83      0.83      0.83      2788
weighted avg       0.83      0.83      0.83      2788
```

iii) <u>K-nearest neighbors:</u>

Definition:

The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand, but has a major drawback of becoming significantly slows as the size of that data in use grows.
KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification)

Performance Metrics:

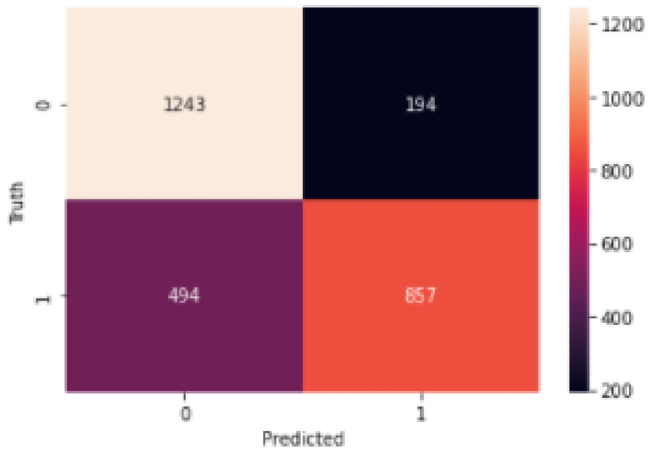
<u>Accuracy</u> : The accuracy of this model is : 0.75322812

<u>Precision</u>: The precision of this model is : 0.81541389

<u>Recall</u>: The recall of this model is : 0.63434492

Confusion Matrix:

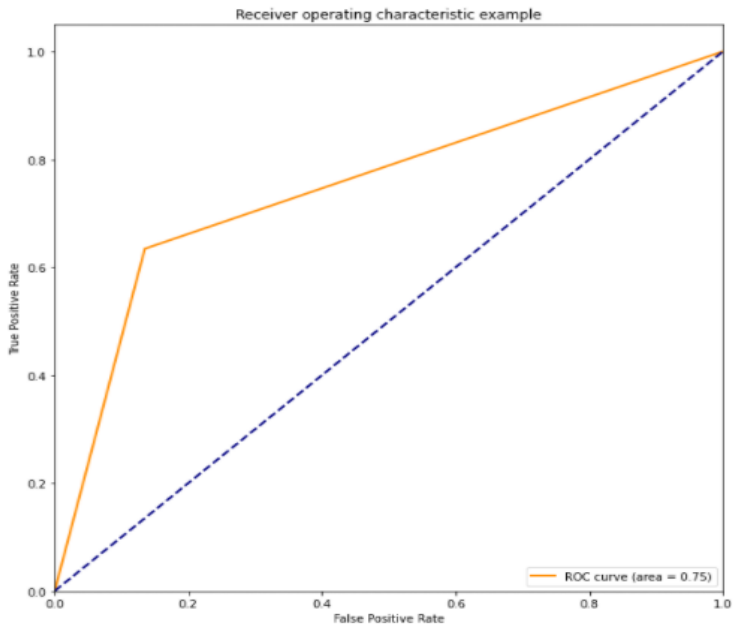
```
In [160]: confusionmatrix(model_3,y_pred_3)
```



ROC Curve:

```
]: ROC_curve(y_pred_3)
```

```
[0.         0.13500348 1.        ]
[0.         0.63434493 1.        ]
[2 1 0]

<Figure size 432x288 with 0 Axes>
```



AUC is .75 ,which is quite good.

Tuning Details:

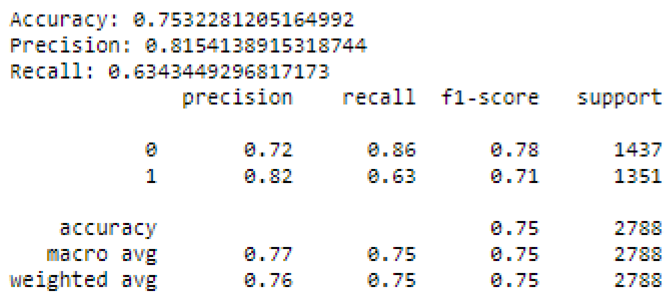
i) n_neighbors = 3:

```
Accuracy: 0.7553802008608321
```

ii) n_neighbors = 21:

```
Accuracy: 0.75322812051
```

Screenshot of the output:



```
Accuracy: 0.7532281205164992
Precision: 0.8154138915318744
Recall: 0.6343449296817173
              precision    recall  f1-score   support

           0       0.72      0.86      0.78      1437
           1       0.82      0.63      0.71      1351

    accuracy                           0.75      2788
   macro avg       0.77      0.75      0.75      2788
weighted avg       0.76      0.75      0.75      2788
```

iv) Decision Tree:

Definition:

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

Performance Metrics:

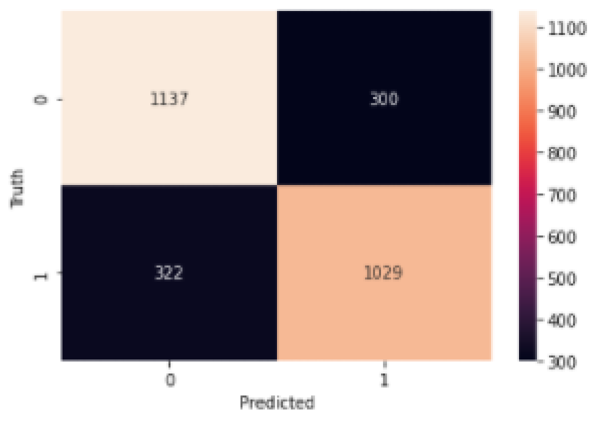
Accuracy : The accuracy of this model is : 0.77690100

Precision: The precision of this model is : 0.7742663

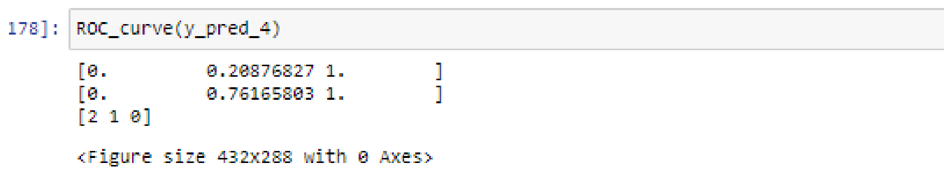
Recall: The recall of this model is : 0.7616580

Confusion Matrix:

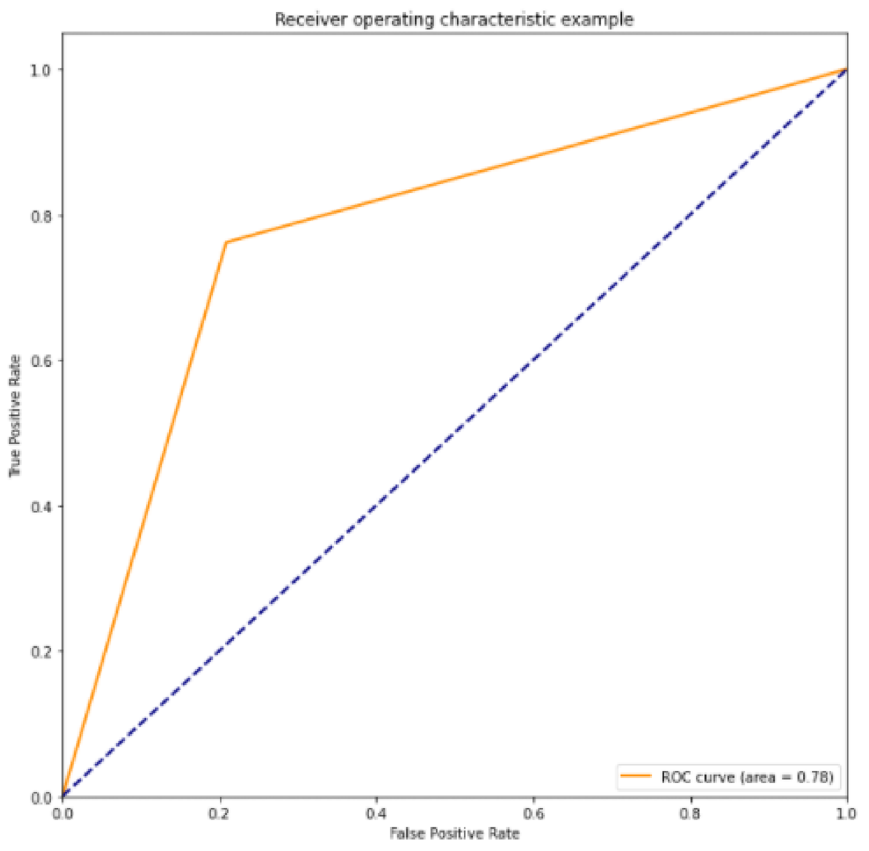
```
In [167]: confusionmatrix(model_4,y_pred_4)
```



ROC Curve:

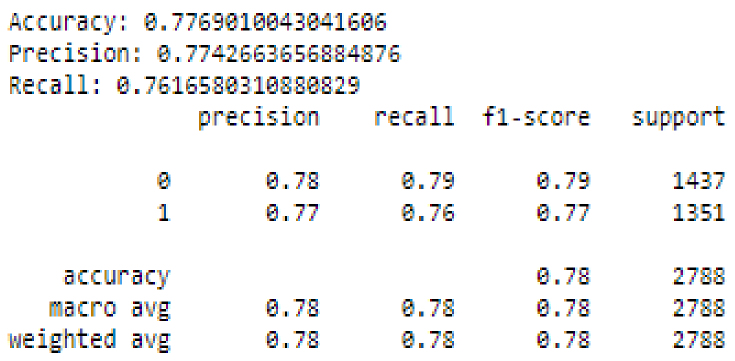


```
178]: ROC_curve(y_pred_4)

[0.         0.20876827 1.        ]
[0.         0.76165803 1.        ]
[2 1 0]

<Figure size 432x288 with 0 Axes>
```



The AUC is .78 , which is a good value.

Screenshot of the output:



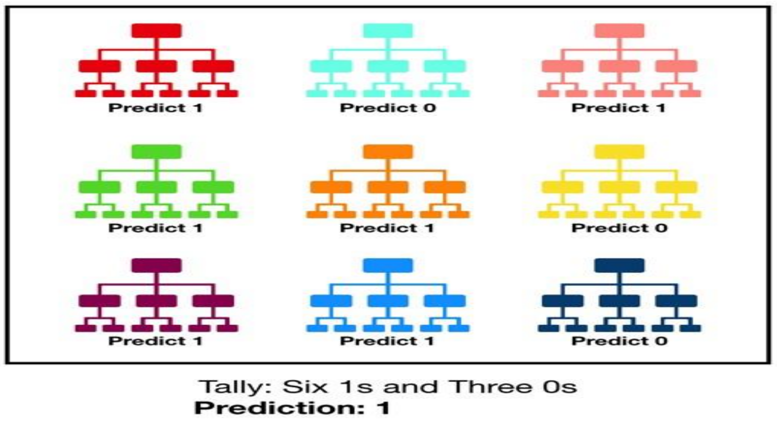
```
Accuracy: 0.7769010043041606
Precision: 0.7742663656884876
Recall: 0.7616580310880829
              precision    recall  f1-score   support

           0       0.78      0.79      0.79      1437
           1       0.77      0.76      0.77      1351

    accuracy                           0.78      2788
   macro avg       0.78      0.78      0.78      2788
weighted avg       0.78      0.78      0.78      2788
```

v) Random Forest:

Definition:

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.



Performance Metrics:

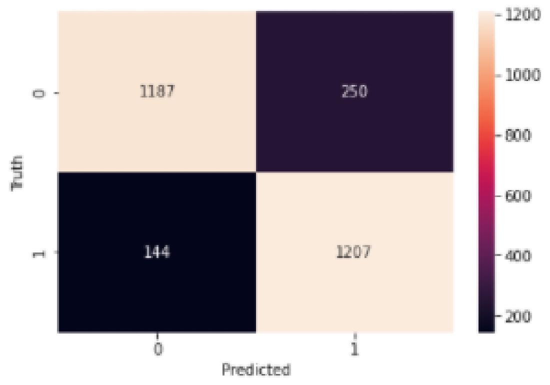
Accuracy : The accuracy of this model is : 0.858680

Precision: The precision of this model is : 0.82841

Recall: The recall of this model is : 0.893412

Confusion Matrix:

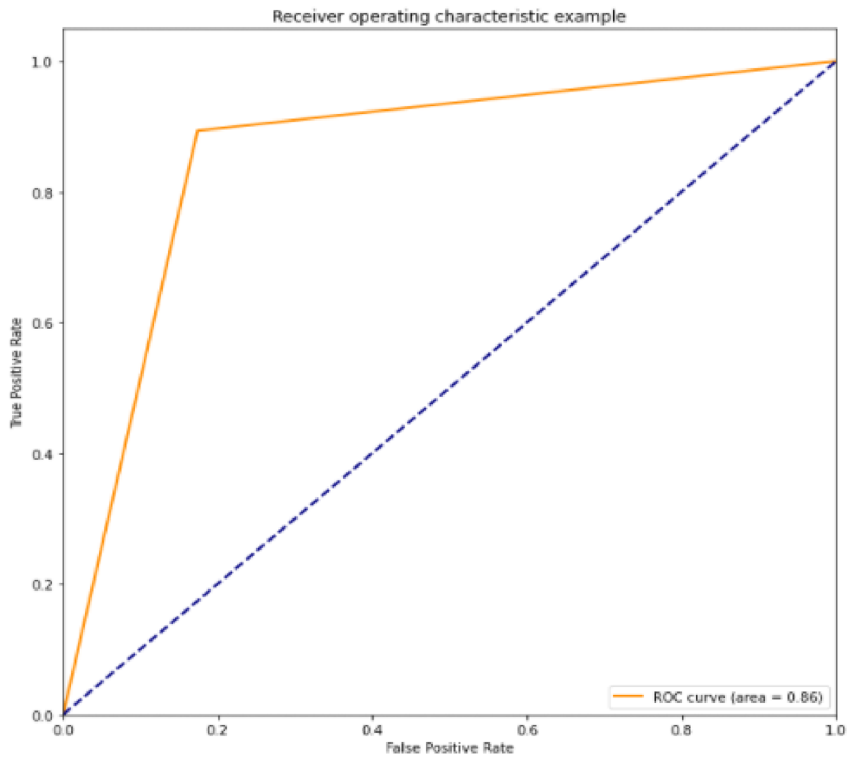
```
In [171]: confusionmatrix(model_5,y_pred_5)
```



ROC Curve:

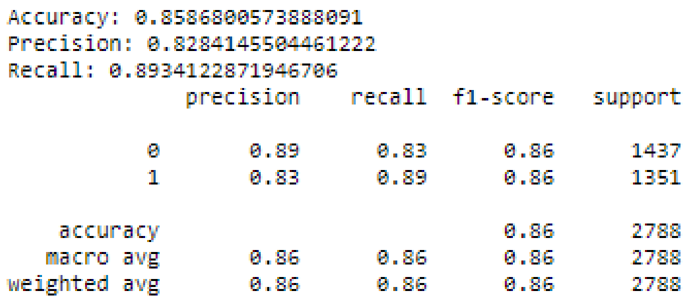
```
ROC_curve(y_pred_5)
```

```
[0.         0.17397356 1.        ]
[0.         0.89341229 1.        ]
[2 1 0]

<Figure size 432x288 with 0 Axes>
```



The AUC is .86 ,which is quite good value.

Screenshot of the output:



```
Accuracy: 0.8586800573888091
Precision: 0.8284145504461222
Recall: 0.8934122871946706
              precision    recall  f1-score   support

           0       0.89      0.83      0.86      1437
           1       0.83      0.89      0.86      1351

    accuracy                           0.86      2788
   macro avg       0.86      0.86      0.86      2788
weighted avg       0.86      0.86      0.86      2788
```

vi) Neural Network:

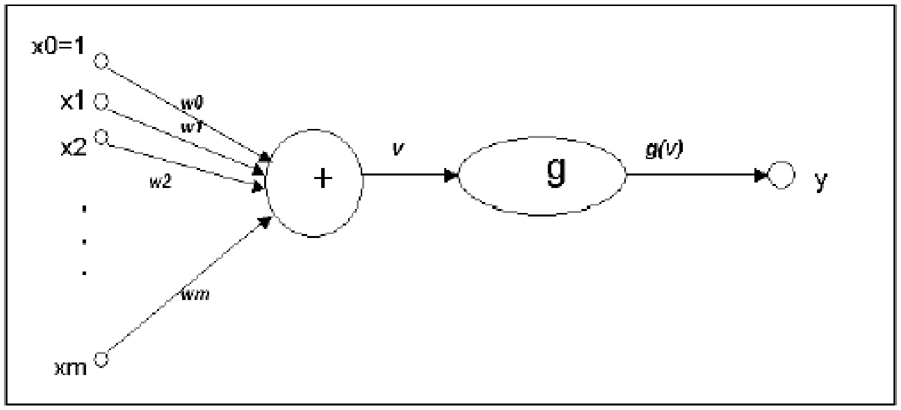
Definition:

Artificial neural networks are relatively crude electronic networks of neurons based on the neural structure of the brain. They process records one at a time, and learn by comparing their classification of the record (i.e., largely arbitrary) with the known actual classification of the record. The errors from the initial classification of the first record is fed back into the network, and used to modify the networks algorithm for further iterations.

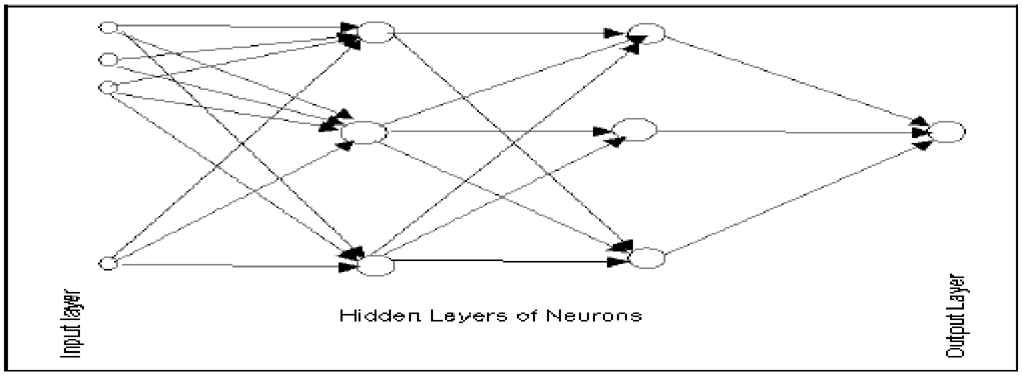
A neuron in an artificial neural network is

1. A set of input values (xi) and associated weights (wi).

2. A function (g) that sums the weights and maps the results to an output (y).



Neurons are organized into layers: input, hidden and output. The input layer is composed not of full neurons, but rather consists simply of the record's values that are inputs to the next layer of neurons. The next layer is the hidden layer. Several hidden layers can exist in one neural network. The final layer is the output layer, where there is one node for each class. A single sweep forward through the network results in the assignment of a value to each output node, and the record is assigned to the class node with the highest value.



Performance Metrics:

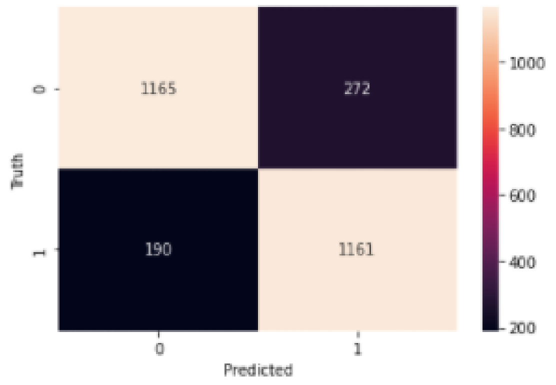
Accuracy : The accuracy of this model is : 0.83

Precision: The precision of this model is : 0.83

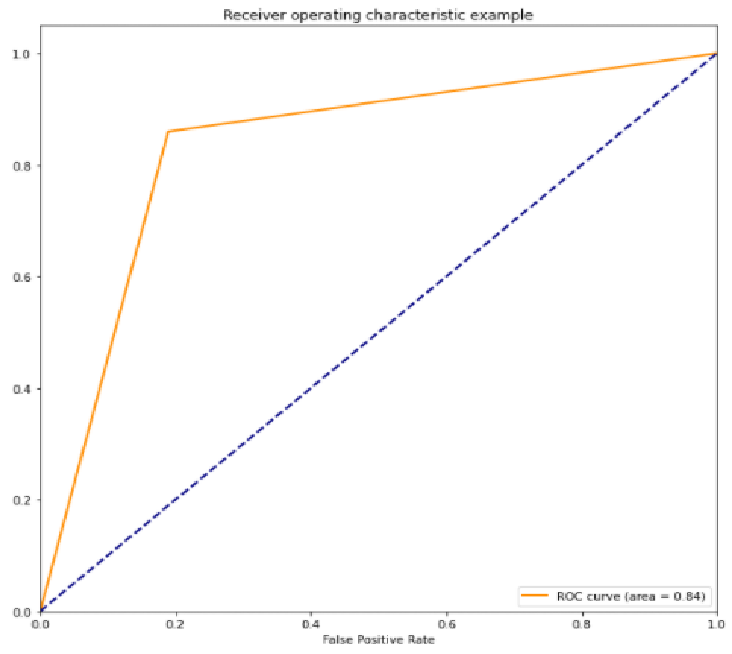
Recall: The recall of this model is : 0.83

Confusion Matrix:

```
n [184]: confusionmatrix(model_6,y_pred_6)
```

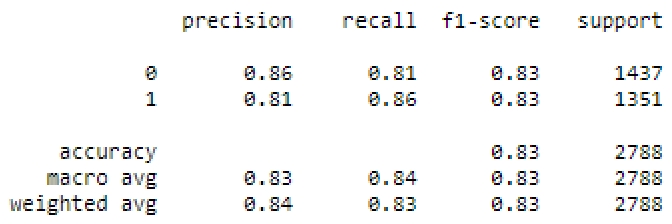


ROC Curve:



The AUC is .84, which is a good value.

Screenshot of the output:



```
              precision    recall  f1-score   support

           0       0.86      0.81      0.83      1437
           1       0.81      0.86      0.83      1351

    accuracy                           0.83      2788
   macro avg       0.83      0.84      0.83      2788
weighted avg       0.84      0.83      0.83      2788
```