Assignment No: 03

Title: Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

**Name**: Vasant Kumar

**Class :B.E** **Div:B** **Batch:C**

**Roll No:405B091**

In [1]:
```
# Importing header files and csv file
#Importing the libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# loading file
df = pd.read_csv("Churn_Modelling.csv")
```

## Preprocessing

In [2]:
```
df.head()
```

Out[2]:

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProd |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 15634602 | Hargrave | 619 | France | Female | 42 | 2 | 0.00 | |
| **1** | 2 | 15647311 | Hill | 608 | Spain | Female | 41 | 1 | 83807.86 | |
| **2** | 3 | 15619304 | Onio | 502 | France | Female | 42 | 8 | 159660.80 | |
| **3** | 4 | 15701354 | Boni | 699 | France | Female | 39 | 1 | 0.00 | |
| **4** | 5 | 15737888 | Mitchell | 850 | Spain | Female | 43 | 2 | 125510.82 | |

In [3]:
```
df.shape
```

Out[3]:
```
(10000, 14)
```

In [4]:
```
df.describe()
```

Out[4]:

| | RowNumber | CustomerId | CreditScore | Age | Tenure | Balance | NumOfProducts |
|---|---|---|---|---|---|---|---|
| **count** | 10000.00000 | 1.000000e+04 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 |
| **mean** | 5000.50000 | 1.569094e+07 | 650.528800 | 38.921800 | 5.012800 | 76485.889288 | 1.530200 |
| **std** | 2886.89568 | 7.193619e+04 | 96.653299 | 10.487806 | 2.892174 | 62397.405202 | 0.581654 |
| **min** | 1.00000 | 1.556570e+07 | 350.000000 | 18.000000 | 0.000000 | 0.000000 | 1.000000 |
| **25%** | 2500.75000 | 1.562853e+07 | 584.000000 | 32.000000 | 3.000000 | 0.000000 | 1.000000 |
| **50%** | 5000.50000 | 1.569074e+07 | 652.000000 | 37.000000 | 5.000000 | 97198.540000 | 1.000000 |
| **75%** | 7500.25000 | 1.575323e+07 | 718.000000 | 44.000000 | 7.000000 | 127644.240000 | 2.000000 |
| **max** | 10000.00000 | 1.581569e+07 | 850.000000 | 92.000000 | 10.000000 | 250898.090000 | 4.000000 |

In [5]:
```
df.isnull()
```

Out[5]:

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfPr |
|---|---|---|---|---|---|---|---|---|---|---|

| | RowNumber | CustomerId | Surname | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfPr |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | False | False | False | False | |
| **1** | False | False | False | False | False | False | False | False | False | |
| **2** | False | False | False | False | False | False | False | False | False | |
| **3** | False | False | False | False | False | False | False | False | False | |
| **4** | False | False | False | False | False | False | False | False | False | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **9995** | False | False | False | False | False | False | False | False | False | |
| **9996** | False | False | False | False | False | False | False | False | False | |
| **9997** | False | False | False | False | False | False | False | False | False | |
| **9998** | False | False | False | False | False | False | False | False | False | |
| **9999** | False | False | False | False | False | False | False | False | False | |

10000 rows × 14 columns

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [8]:
```
df.dtypes
```

Out[8]:
```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [9]:
```
df.columns
```

Out[9]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [10]:
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary colum
df.head()
```

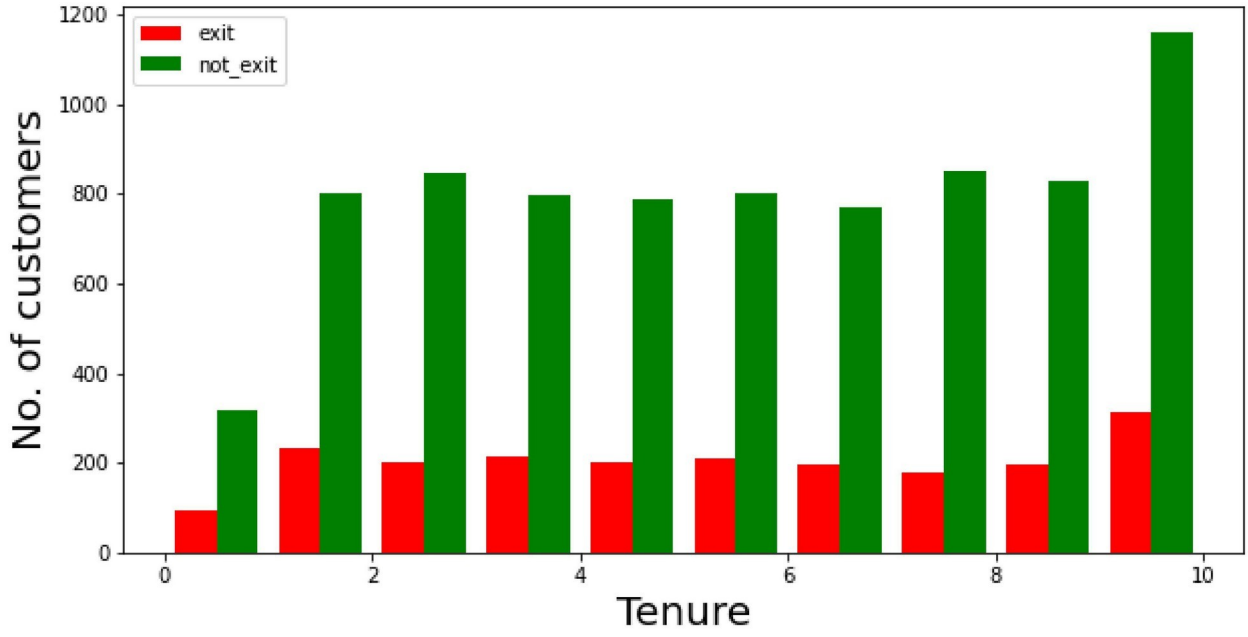
Out[10]:

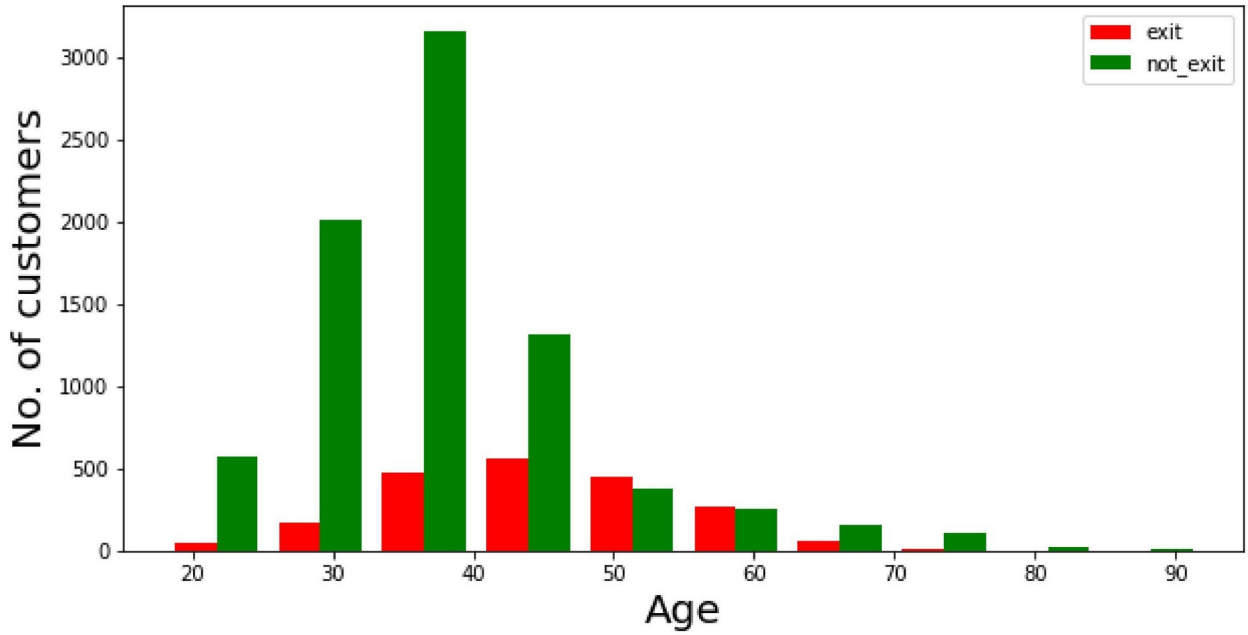
| | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | Es |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | |
| **1** | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | |
| **2** | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | |
| **3** | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | |
| **4** | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 | 1 | |

## Visualization

In [11]:
```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/fromnumeric.py:3208: VisibleDeprecationWarn
ing: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-t
uples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, y
ou must specify 'dtype=object' when creating the ndarray.
  return asarray(a).size
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDeprecationW
arning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-o
r-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do thi
s, you must specify 'dtype=object' when creating the ndarray.
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```




In [12]:
```
# Converting the Categorical Variables
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActive
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
df = pd.concat([df,gender,states], axis = 1)
df.head()
```

Out[12]:

| | CreditScore | Geography | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | Es |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 619 | France | Female | 42 | 2 | 0.00 | 1 | 1 | 1 | |
| **1** | 608 | Spain | Female | 41 | 1 | 83807.86 | 1 | 0 | 1 | |
| **2** | 502 | France | Female | 42 | 8 | 159660.80 | 3 | 1 | 0 | |
| **3** | 699 | France | Female | 39 | 1 | 0.00 | 2 | 0 | 0 | |
| **4** | 850 | Spain | Female | 43 | 2 | 125510.82 | 1 | 1 | 1 | |

In [13]:
```
# Splitting the training and testing Dataset
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','
y = df['Exited']
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.30)
```

In [14]:
```
# Normalizing the values with mean as 0 and Standard Deviation as 1
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print("Train \n",X_train)
print("Test \n",X_test)
```

```
Train
 [[ 0.62819435 -0.9478278   1.04124445 ... -1.08847096 -0.57031183
  -0.57339125]
 [ 0.349612    1.83232831 -1.36892977 ...  0.91871996 -0.57031183
  -0.57339125]
 [ 0.32897627 -0.85196035  1.72986565 ...  0.91871996 -0.57031183
  -0.57339125]
 ...
 [ 1.15440545  1.35299105 -0.33599796 ... -1.08847096 -0.57031183
  -0.57339125]
 [-1.86872892 -0.37262309 -1.36892977 ...  0.91871996 -0.57031183
  -0.57339125]
 [ 0.48374424  1.2571236   1.04124445 ...  0.91871996  1.75342671
  -0.57339125]]
Test
 [[ 0.80359805  0.58605143  0.69693385 ... -1.08847096 -0.57031183
   1.74400987]
 [ 0.41151919 -1.71476742 -0.68030856 ...  0.91871996 -0.57031183
   1.74400987]
 [ 0.61787649  0.20258162 -0.68030856 ...  0.91871996 -0.57031183
  -0.57339125]
 ...
 [-0.10437405 -0.18088819  0.69693385 ... -1.08847096  1.75342671
  -0.57339125]
 [-1.7552324   0.10671417 -0.68030856 ...  0.91871996 -0.57031183
  -0.57339125]
 [-0.03214899 -0.08502073  1.72986565 ...  0.91871996 -0.57031183
  -0.57339125]]
```

In [15]:
```
# Building the Classifier Model using Keras
import keras
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
```

In [16]:
```
classifier = Sequential()
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializer = "unifo
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform")) #Adding s
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform")) #Final
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy']) #To
```

In [17]:
```
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron i
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6)                 72

 dense_1 (Dense)             (None, 6)                 42

 dense_2 (Dense)             (None, 1)                 7

=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training dataset
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
```

```
Epoch 1/50
700/700 [==============================] - 2s 1ms/step - loss: 0.5023 - accuracy: 0.7934
Epoch 2/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4347 - accuracy: 0.7931
Epoch 3/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4309 - accuracy: 0.7931
Epoch 4/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4267 - accuracy: 0.8071
Epoch 5/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4230 - accuracy: 0.8174
Epoch 6/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4187 - accuracy: 0.8246
Epoch 7/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4152 - accuracy: 0.8287
Epoch 8/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4124 - accuracy: 0.8304
Epoch 9/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4102 - accuracy: 0.8291
Epoch 10/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4086 - accuracy: 0.8313
Epoch 11/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4070 - accuracy: 0.8320
Epoch 12/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4059 - accuracy: 0.8317
Epoch 13/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4050 - accuracy: 0.8317
Epoch 14/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4039 - accuracy: 0.8330
Epoch 15/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4033 - accuracy: 0.8309
Epoch 16/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4018 - accuracy: 0.8326
Epoch 17/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4026 - accuracy: 0.8317
Epoch 18/50
700/700 [==============================] - 1s 2ms/step - loss: 0.4019 - accuracy: 0.8321
Epoch 19/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4016 - accuracy: 0.8337
Epoch 20/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4012 - accuracy: 0.8320
Epoch 21/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4010 - accuracy: 0.8341
Epoch 22/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4006 - accuracy: 0.8334
Epoch 23/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4012 - accuracy: 0.8321
Epoch 24/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4003 - accuracy: 0.8341
Epoch 25/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4001 - accuracy: 0.8324
Epoch 26/50
700/700 [==============================] - 1s 998us/step - loss: 0.3994 - accuracy: 0.8336
Epoch 27/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3999 - accuracy: 0.8343
Epoch 28/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3997 - accuracy: 0.8324
Epoch 29/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3990 - accuracy: 0.8334
Epoch 30/50
700/700 [==============================] - 1s 993us/step - loss: 0.3985 - accuracy: 0.8351
Epoch 31/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3989 - accuracy: 0.8339
Epoch 32/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3991 - accuracy: 0.8347
Epoch 33/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3987 - accuracy: 0.8346
Epoch 34/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3983 - accuracy: 0.8349
```

```
Epoch 35/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3985 - accuracy: 0.8351
Epoch 36/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3984 - accuracy: 0.8349
Epoch 37/50
700/700 [==============================] - 1s 2ms/step - loss: 0.3984 - accuracy: 0.8350
Epoch 38/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3980 - accuracy: 0.8351
Epoch 39/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3980 - accuracy: 0.8361
Epoch 40/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3984 - accuracy: 0.8331
Epoch 41/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3976 - accuracy: 0.8347
Epoch 42/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3975 - accuracy: 0.8360
Epoch 43/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3977 - accuracy: 0.8349
Epoch 44/50
700/700 [==============================] - 1s 991us/step - loss: 0.3974 - accuracy: 0.8369
Epoch 45/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3972 - accuracy: 0.8347
Epoch 46/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3975 - accuracy: 0.8347
Epoch 47/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3969 - accuracy: 0.8347
Epoch 48/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3971 - accuracy: 0.8346
Epoch 49/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3969 - accuracy: 0.8353
Epoch 50/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3970 - accuracy: 0.8349
94/94 [==============================] - 0s 853us/step
```

In [19]:
```
y_pred
```

Out[19]:
```
array([[False],
       [False],
       [False],
       ...,
       [False],
       [False],
       [False]])
```

In [20]:
```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
cm = confusion_matrix(y_test,y_pred)
cm
```

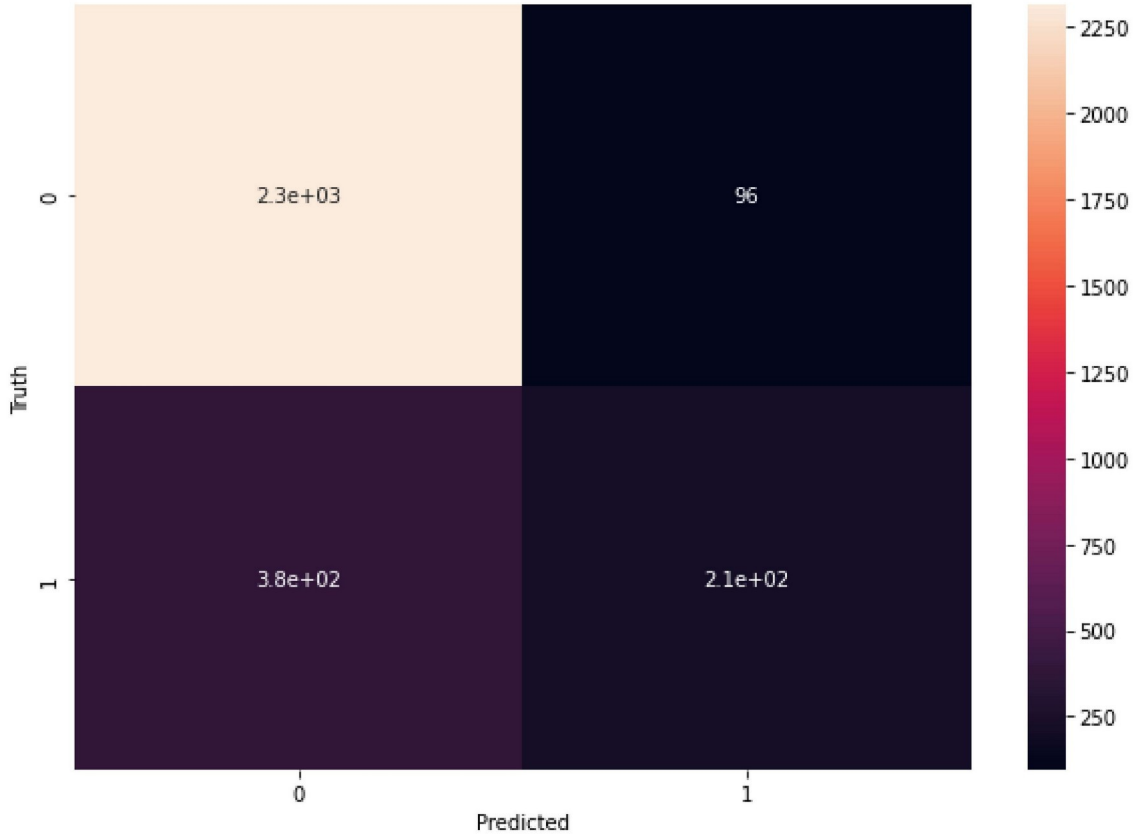
Out[20]:
```
array([[2315,   96],
       [ 375,  214]])
```

In [21]:
```
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

Out[21]:
```
0.843
```

In [22]:
```
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[22]:
```
Text(69.0, 0.5, 'Truth')
```




In [23]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      2411
           1       0.69      0.36      0.48       589

    accuracy                           0.84      3000
   macro avg       0.78      0.66      0.69      3000
weighted avg       0.83      0.84      0.82      3000
```