ASSIGNEMNT-3: Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt #Importing the libraries
```

In [2]:
```
df = pd.read_csv("Churn_Modelling.csv")
```

Preprocessing.

In [3]:
```
df.head()
```

Out[3]:

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

In [4]:
```
df.shape
```

Out[4]:
```
(10000, 14)
```

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

Out[5]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	Has(
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	Has(
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1

In [6]:

```
df.isnull()
```

Out[6]:

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

10000 rows × 14 columns

In [7]:

```
df.isnull().sum()
```

Out[7]:

```
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 [8]:

```
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 [9]:
```
df.dtypes
```

Out[9]:
```
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 [10]:
```
df.columns
```

Out[10]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [11]:
```
df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) #Dropping the unnecessary co
```

In [12]:
```
df.head()
```

Out[12]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estimate
0	619	France	Female	42	2	0.00	1	1	1	1(
1	608	Spain	Female	41	1	83807.86	1	0	1	1'
2	502	France	Female	42	8	159660.80	3	1	0	1'
3	699	France	Female	39	1	0.00	2	0	0	(
4	850	Spain	Female	43	2	125510.82	1	1	1	7

Visualization

In [13]:

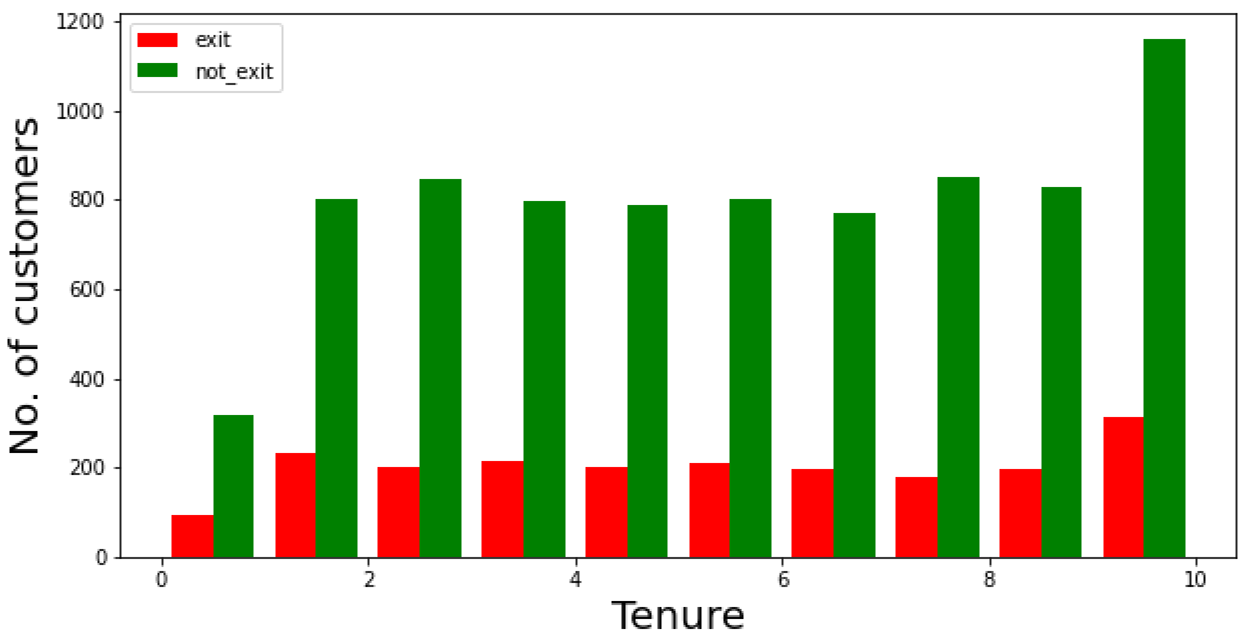
```
def visualization(x, y, xlabel):
    plt.figure(figsize=(10,5))
    plt.hist([x, y], color=['red', 'green'], label = ['exit', 'not_exit'])
    plt.xlabel(xlabel,fontsize=20)
    plt.ylabel("No. of customers", fontsize=20)
    plt.legend()
```

In [14]:

```
df_churn_exited = df[df['Exited']==1]['Tenure']
df_churn_not_exited = df[df['Exited']==0]['Tenure']
```

In [15]:

```
visualization(df_churn_exited, df_churn_not_exited, "Tenure")
```

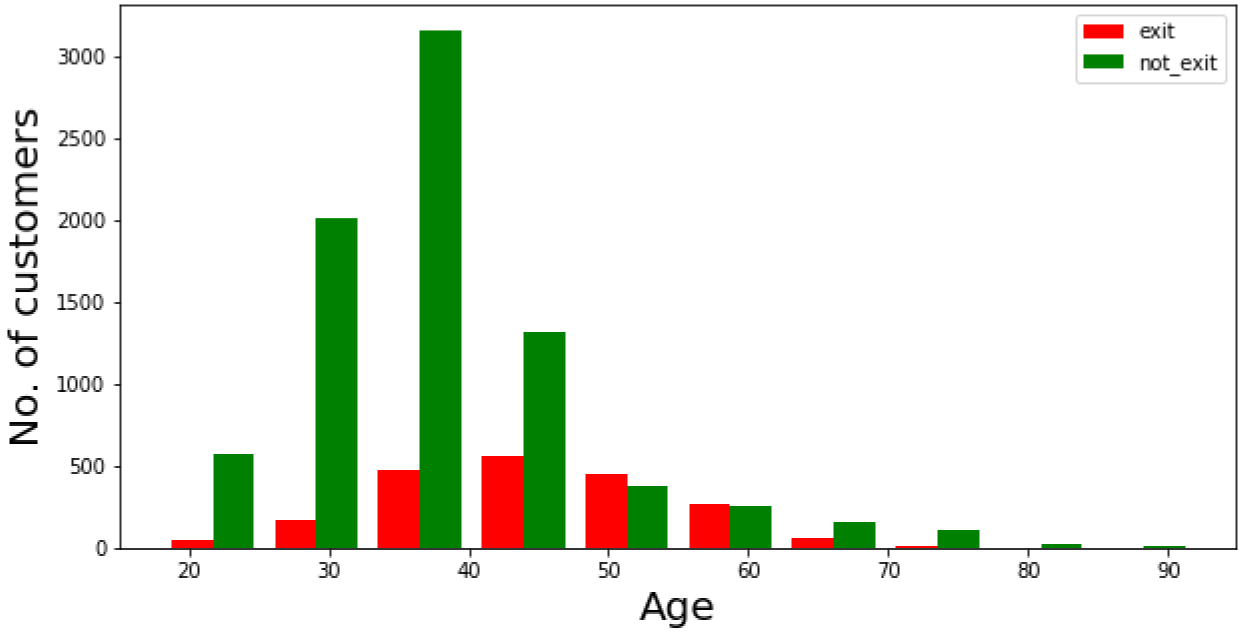


In [16]:

```
df_churn_exited2 = df[df['Exited']==1]['Age']
df_churn_not_exited2 = df[df['Exited']==0]['Age']
```

In [17]:

```
visualization(df_churn_exited2, df_churn_not_exited2, "Age")
```



Converting the Categorical Variables

In [18]:
```
X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsAct
states = pd.get_dummies(df['Geography'],drop_first = True)
gender = pd.get_dummies(df['Gender'],drop_first = True)
```

In [19]:
```
df = pd.concat([df,gender,states], axis = 1)
```

Splitting the training and testing Dataset

In [20]:
```
df.head()
```

Out[20]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estimate
0	619	France	Female	42	2	0.00	1	1	1	1(
1	608	Spain	Female	41	1	83807.86	1	0	1	1'
2	502	France	Female	42	8	159660.80	3	1	0	1'
3	699	France	Female	39	1	0.00	2	0	0	(
4	850	Spain	Female	43	2	125510.82	1	1	1	7

In [21]:
```
X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember
```

In [22]:
```
y = df['Exited']
```

In [23]:
```
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)
```

Normalizing the values with mean as 0 and Standard

Deviation as 1

In [24]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [25]:
```
X_train  = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [26]:
```
X_train
```

Out[26]:
```
array([[-1.59854167, -0.94412505, -1.0440418 , ...,  0.90375116,
        -0.58174919,  1.73073215],
       [ 0.39192495, -0.94412505, -1.39017672, ...,  0.90375116,
        -0.58174919, -0.57779016],
       [-0.88321773, -0.65980305,  1.37890261, ..., -1.10649927,
         1.71895383, -0.57779016],
       ...,
       [ 0.17421766, -0.47025505, -0.00563705, ..., -1.10649927,
        -0.58174919,  1.73073215],
       [ 0.06018051, -0.47025505,  0.34049786, ..., -1.10649927,
        -0.58174919,  1.73073215],
       [ 1.20055202, -0.09115905,  0.68663278, ..., -1.10649927,
         1.71895383, -0.57779016]])
```

In [27]:
```
X_test
```

Out[27]:
```
array([[ 0.60963224, -0.28070705,  0.68663278, ..., -1.10649927,
        -0.58174919, -0.57779016],
       [-1.02835592, -0.37548105,  0.68663278, ...,  0.90375116,
        -0.58174919, -0.57779016],
       [-0.9350528 ,  1.70954695, -1.73631163, ...,  0.90375116,
        -0.58174919, -0.57779016],
       ...,
       [ 1.52192944, -0.94412505, -1.39017672, ..., -1.10649927,
        -0.58174919, -0.57779016],
       [ 0.49559509,  0.09838895, -1.73631163, ..., -1.10649927,
        -0.58174919,  1.73073215],
       [ 0.18458468,  0.66703295, -1.73631163, ...,  0.90375116,
        -0.58174919, -0.57779016]])
```

Building the Classifier Model using Keras

In [28]:
```
import keras #Keras is the wrapper on the top of tenserflow
#Can use Tenserflow as well but won't be able to understand the errors initially.
```

In [29]:
```
from keras.models import Sequential #To create sequential neural network
from keras.layers import Dense #To create hidden layers
```

In [30]:
```
classifier = Sequential()
```

In [31]:
```
#To add the layers
#Dense helps to contruct the neurons
#Input Dimension means we have 11 features
# Units is to create the hidden layers
```

```
#Uniform helps to distribute the weight uniformly
classifier.add(Dense(activation = "relu",input_dim = 11,units = 6,kernel_initializer = "ur
```

In [32]:

```
classifier.add(Dense(activation = "relu",units = 6,kernel_initializer = "uniform"))    #Add
```

In [33]:

```
classifier.add(Dense(activation = "sigmoid",units = 1,kernel_initializer = "uniform")) #F
```

In [34]:

```
classifier.compile(optimizer="adam",loss = 'binary_crossentropy',metrics = ['accuracy']) #
```

In [35]:

```
classifier.summary() #3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuro
```

```
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 [36]:

```
classifier.fit(X_train,y_train,batch_size=10,epochs=50) #Fitting the ANN to training datas
```

```
Epoch 1/50
700/700 [==============================] - 7s 7ms/step - loss: 0.4930 - accuracy: 0.7970
Epoch 2/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4309 - accuracy: 0.7971
Epoch 3/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4260 - accuracy: 0.7971
Epoch 4/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4215 - accuracy: 0.8020
Epoch 5/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4185 - accuracy: 0.8210
Epoch 6/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4162 - accuracy: 0.8239
Epoch 7/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4145 - accuracy: 0.8300
Epoch 8/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4133 - accuracy: 0.8287
Epoch 9/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4121 - accuracy: 0.8311
Epoch 10/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4105 - accuracy: 0.8313
Epoch 11/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4103 - accuracy: 0.8327
Epoch 12/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4099 - accuracy: 0.8319
Epoch 13/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4090 - accuracy: 0.8340
Epoch 14/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4084 - accuracy: 0.8336
Epoch 15/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4080 - accuracy: 0.8339
```

```
Epoch 16/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4075 - accuracy: 0.8356
Epoch 17/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4067 - accuracy: 0.8343
Epoch 18/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4066 - accuracy: 0.8354
Epoch 19/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4061 - accuracy: 0.8347
Epoch 20/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4061 - accuracy: 0.8333
Epoch 21/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4056 - accuracy: 0.8347
Epoch 22/50
700/700 [==============================] - 4s 6ms/step - loss: 0.4048 - accuracy: 0.8350
Epoch 23/50
700/700 [==============================] - 4s 6ms/step - loss: 0.4054 - accuracy: 0.8349
Epoch 24/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4038 - accuracy: 0.8346
Epoch 25/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4025 - accuracy: 0.8359
Epoch 26/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4015 - accuracy: 0.8350
Epoch 27/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4017 - accuracy: 0.8347
Epoch 28/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4009 - accuracy: 0.8341
Epoch 29/50
700/700 [==============================] - 5s 8ms/step - loss: 0.4006 - accuracy: 0.8343
Epoch 30/50
700/700 [==============================] - 5s 7ms/step - loss: 0.4000 - accuracy: 0.8359
Epoch 31/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3994 - accuracy: 0.8351
Epoch 32/50
700/700 [==============================] - 5s 8ms/step - loss: 0.3992 - accuracy: 0.8354
Epoch 33/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3989 - accuracy: 0.8353
Epoch 34/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3982 - accuracy: 0.8343
Epoch 35/50
700/700 [==============================] - 5s 8ms/step - loss: 0.3982 - accuracy: 0.8336
Epoch 36/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3972 - accuracy: 0.8354
Epoch 37/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3970 - accuracy: 0.8354
Epoch 38/50
700/700 [==============================] - 5s 8ms/step - loss: 0.3967 - accuracy: 0.8366
Epoch 39/50
700/700 [==============================] - 5s 8ms/step - loss: 0.3968 - accuracy: 0.8349
Epoch 40/50
700/700 [==============================] - 5s 8ms/step - loss: 0.3967 - accuracy: 0.8361
Epoch 41/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3959 - accuracy: 0.8364
Epoch 42/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3959 - accuracy: 0.8361
Epoch 43/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3960 - accuracy: 0.8356
Epoch 44/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3958 - accuracy: 0.8357
Epoch 45/50
700/700 [==============================] - 6s 8ms/step - loss: 0.3959 - accuracy: 0.8341
Epoch 46/50
700/700 [==============================] - 5s 8ms/step - loss: 0.3955 - accuracy: 0.8356
Epoch 47/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3957 - accuracy: 0.8353
Epoch 48/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3952 - accuracy: 0.8369
```

```
Epoch 49/50
700/700 [==============================] - 4s 6ms/step - loss: 0.3956 - accuracy: 0.8357
Epoch 50/50
700/700 [==============================] - 5s 7ms/step - loss: 0.3948 - accuracy: 0.8363
```

Out[36]:

```
<keras.callbacks.History at 0x1a7b365d0a0>
```

In [37]:

```
y_pred =classifier.predict(X_test)
y_pred = (y_pred > 0.5) #Predicting the result
```

In [38]:

```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
```

In [39]:

```
cm = confusion_matrix(y_test,y_pred)
```

In [40]:

```
cm
```

Out[40]:

```
array([[2304,   79],
       [ 399,  218]], dtype=int64)
```

In [41]:

```
accuracy = accuracy_score(y_test,y_pred)
```

In [42]:

```
accuracy
```

Out[42]:

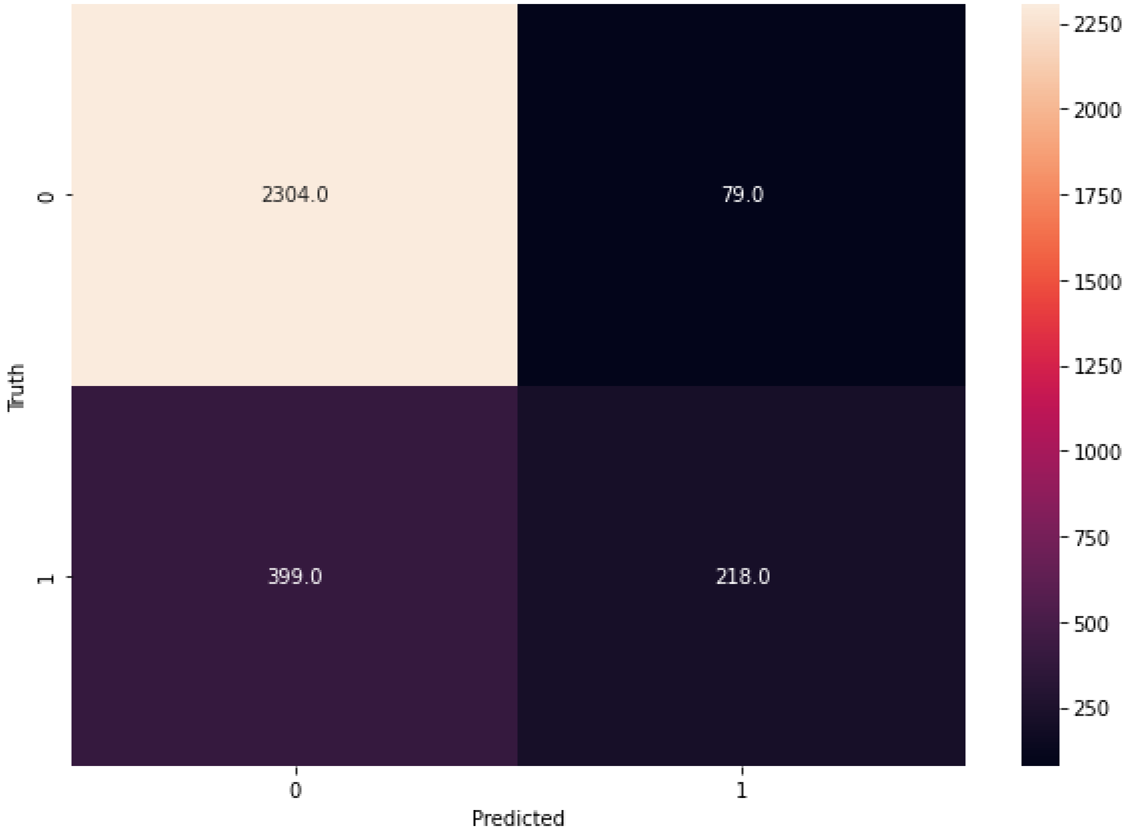
```
0.8406666666666667
```

In [45]:

```
plt.figure(figsize = (10,7))
sns.heatmap(cm,annot = True,fmt = ".1f")
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[45]:

```
Text(69.0, 0.5, 'Truth')
```



In [44]:

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.97      0.91      2383
           1       0.73      0.35      0.48       617

    accuracy                           0.84      3000
   macro avg       0.79      0.66      0.69      3000
weighted avg       0.83      0.84      0.82      3000
```