Name :

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

Assignment No .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:](https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling)

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 [46]:

**import pandas as pd import numpy as np**

**import seaborn as sns**

**import matplotlib.pyplot as plt** *#Importing the libraries*

In [47]:

**Preprocessing.**

df = pd.read\_csv("Churn\_Modelling.csv")

In [48]:

df.head()

Out[48]:

**RowNumber CustomerId Surname CreditScore Geography Gender Age Tenure Balance NumOfProducts HasCrCard IsActiveMember EstimatedSala**

**0** 1 15634602 Hargrave 619 France Female 42 2 0.00 1 1 1 101348.8

**1** 2 15647311 Hill 608 Spain Female 41 1 83807.86 1 0 1 112542.5

**2** 3 15619304 Onio 502 France Female 42 8 159660.80 3 1 0 113931.5

**3** 4 15701354 Boni 699 France Female 39 1 0.00 2 0 0 93826.6

**4** 5 15737888 Mitchell 850 Spain Female 43 2 125510.82 1 1 1 79084.1

In [49]:

df.shape

Out[49]: (10000, 14)

In [50]:

df.describe()

Out[50]:

**RowNumber CustomerId CreditScore Age Tenure Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary**

**count** 10000.00000 1.000000e+04 10000.000000 10000.000000 10000.000000 10000.000000 10000.000000 10000.00000 10000.000000 10000.000000

**mean** 5000.50000 1.569094e+07 650.528800 38.921800 5.012800 76485.889288 1.530200 0.70550 0.515100 100090.239881

**std** 2886.89568 7.193619e+04 96.653299 10.487806 2.892174 62397.405202 0.581654 0.45584 0.499797 57510.492818

**min** 1.00000 1.556570e+07 350.000000 18.000000 0.000000 0.000000 1.000000 0.00000 0.000000 11.580000

**25%** 2500.75000 1.562853e+07 584.000000 32.000000 3.000000 0.000000 1.000000 0.00000 0.000000 51002.110000

**50%** 5000.50000 1.569074e+07 652.000000 37.000000 5.000000 97198.540000 1.000000 1.00000 1.000000 100193.915000

**75%** 7500.25000 1.575323e+07 718.000000 44.000000 7.000000 127644.240000 2.000000 1.00000 1.000000 149388.247500

**max** 10000.00000 1.581569e+07 850.000000 92.000000 10.000000 250898.090000 4.000000 1.00000 1.000000 199992.480000

In [51]:

df.isnull()

Out[51]:

**RowNumber CustomerId Surname CreditScore Geography Gender Age Tenure Balance NumOfProducts HasCrCard IsActiveMember EstimatedS**

**0** False False False False False False False False False False False False

* 1. False False False False False False False False False False False False
  2. False False False False False False False False False False False False
  3. False False False False False False False False False False False False
  4. False False False False False False False False False False False False

**...** ... ... ... ... ... ... ... ... ... ... ... ...

**9995** False False False False False False False False False False False False **9996** False False False False False False False False False False False False **9997** False False False False False False False False False False False False **9998** False False False False False False False False False False False False **9999** False False False False False False False False False False False False

10000 rows × 14 columns

In [52]:

df.isnull().sum()

Out[52]: 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 [53]:

df.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 10000 entries, 0 to 9999 Data columns (total 14 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. RowNumber 10000 non-null int64
2. CustomerId 10000 non-null int64
3. Surname 10000 non-null object
4. CreditScore 10000 non-null int64
5. Geography 10000 non-null object
6. Gender 10000 non-null object
7. Age 10000 non-null int64
8. Tenure 10000 non-null int64
9. Balance 10000 non-null float64
10. NumOfProducts 10000 non-null int64
11. HasCrCard 10000 non-null int64
12. IsActiveMember 10000 non-null int64
13. EstimatedSalary 10000 non-null float64
14. Exited 10000 non-null int64 dtypes: float64(2), int64(9), object(3) memory usage: 1.1+ MB

In [54]:

df.dtypes

Out[54]: 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 [55]:

df.columns

Out[55]: Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Exited'],

dtype='object')

In [56]:

df.head()

df = df.drop(['RowNumber', 'Surname', 'CustomerId'], axis= 1) *#Dropping the unnecessary columns*

In [57]:

Out[57]:

**CreditScore Geography Gender Age Tenure Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary Exited 0** 619 France Female 42 2 0.00 1 1 1 101348.88 1

**1** 608 Spain Female 41 1 83807.86 1 0 1 112542.58 0

**2** 502 France Female 42 8 159660.80 3 1 0 113931.57 1

**3** 699 France Female 39 1 0.00 2 0 0 93826.63 0

**4** 850 Spain Female 43 2 125510.82 1 1 1 79084.10 0

# Visualization

In [101]:

**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 [102]:

df\_churn\_exited = df[df['Exited']==1]['Tenure'] df\_churn\_not\_exited = df[df['Exited']==0]['Tenure']

In [103]:

visualization(df\_churn\_exited, df\_churn\_not\_exited, "Tenure")

In [105]:

df\_churn\_exited2 = df[df['Exited']==1]['Age'] df\_churn\_not\_exited2 = df[df['Exited']==0]['Age']

In [106]:

visualization(df\_churn\_exited2, df\_churn\_not\_exited2, "Age")

# Converting the Categorical Variables

In [59]:

X = df[['CreditScore','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalar y']]

states = pd.get\_dummies(df['Geography'],drop\_first = **True**) gender = pd.get\_dummies(df['Gender'],drop\_first = **True**)

In [61]:

**Splitting the training and testing Dataset**

df = pd.concat([df,gender,states], axis = 1)

In [62]:

Out[62]:

df.head()

**CreditScore Geography Gender Age Tenure Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary Exited Male Germany Spain 0** 619 France Female 42 2 0.00 1 1 1 101348.88 1 0 0 0

**1** 608 Spain Female 41 1 83807.86 1 0 1 112542.58 0 0 0 1

**2** 502 France Female 42 8 159660.80 3 1 0 113931.57 1 0 0 0

**3** 699 France Female 39 1 0.00 2 0 0 93826.63 0 0 0 0

**4** 850 Spain Female 43 2 125510.82 1 1 1 79084.10 0 0 0 1

In [63]:

X = df[['CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','Male'

,'Germany','Spain']]

In [64]:

y = df['Exited']

In [65]:

**Normalizing the values with mean as 0 and Standard Deviation as 1**

**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 [66]:

**from sklearn.preprocessing import** StandardScaler sc = StandardScaler()

In [67]:

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

In [68]:

X\_train

Out[68]: array([[ 4.56838557e-01, -9.45594735e-01, 1.58341939e-03, ..., 9.13181783e-01, -5.81969145e-01, -5.73611200e-01],

[-2.07591864e-02, -2.77416637e-01, 3.47956411e-01, ...,

-1.09507222e+00, -5.81969145e-01, 1.74334114e+00],

[-1.66115021e-01, 1.82257167e+00, -1.38390855e+00, ...,

-1.09507222e+00, -5.81969145e-01, -5.73611200e-01],

...,

[-3.63383654e-01, -4.68324665e-01, 1.73344838e+00, ..., 9.13181783e-01, -5.81969145e-01, -5.73611200e-01],

[ 4.67221117e-01, -1.42286480e+00, 1.38707539e+00, ..., 9.13181783e-01, -5.81969145e-01, 1.74334114e+00],

[-8.82511636e-01, 2.95307447e-01, -6.91162564e-01, ..., 9.13181783e-01, -5.81969145e-01, -5.73611200e-01]])

In [69]:

X\_test

Out[69]: array([[ 3.63395520e-01, 1.99853433e-01, 1.58341939e-03, ..., 9.13181783e-01, -5.81969145e-01, -5.73611200e-01],

[-4.15243057e-02, 4.86215475e-01, 1.58341939e-03, ...,

-1.09507222e+00, -5.81969145e-01, 1.74334114e+00],

[-1.87923736e+00, -3.72870651e-01, -1.38390855e+00, ..., 9.13181783e-01, -5.81969145e-01, -5.73611200e-01],

...,

[-6.02182526e-01, -5.63778679e-01, -1.73028154e+00, ...,

-1.09507222e+00, -5.81969145e-01, -5.73611200e-01],

[ 1.51585964e+00, -6.59232693e-01, 1.73344838e+00, ..., 9.13181783e-01, -5.81969145e-01, -5.73611200e-01],

[-5.19122049e-01, 1.04399419e-01, 1.73344838e+00, ..., 9.13181783e-01, -5.81969145e-01, -5.73611200e-01]])

**Building the Classifier Model using Keras**

In [70]:

**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 [71]:

**from keras.models import** Sequential *#To create sequential neural network*

**from keras.layers import** Dense *#To create hidden layers*

In [72]:

classifier = Sequential()

In [74]:

*#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 = "uniform"))

In [75]:

classifier.add(Dense(activation = "relu",units = 6,kernel\_initializer = "uniform")) *#Adding second hidden layers*

In [76]:

classifier.add(Dense(activation = "sigmoid",units = 1,kernel\_initializer = "uniform")) *#Final neuron will be having siigmoid function*

In [77]:

classifier.compile(optimizer="adam",loss = 'binary\_crossentropy',metrics = ['accuracy']) *#To compile the Artificial Neural Network. Ussed Binary crossentropy as we just have only two output*

In [79]:

classifier.summary() *#3 layers created. 6 neurons in 1st,6neurons in 2nd layer and 1 neuron in last*

Model: "sequential\_1"

Layer (type) Output Shape Param #

=================================================================

dense\_3 (Dense) (None, 6) 72

dense\_4 (Dense) (None, 6) 42

dense\_5 (Dense) (None, 1) 7

=================================================================

Total params: 121

Trainable params: 121

Non-trainable params: 0

In [89]:

classifier.fit(X\_train,y\_train,batch\_size=10,epochs=50) *#Fitting the ANN to training dataset*

Epoch 1/50

700/700 [==============================] - 0s 674us/step - loss: 0.4293 - accuracy: 0.7947

Epoch 2/50

700/700 [==============================] - 0s 647us/step - loss: 0.4239 - accuracy: 0.7947

Epoch 3/50

700/700 [==============================] - 0s 657us/step - loss: 0.4203 - accuracy: 0.8067

Epoch 4/50

700/700 [==============================] - 0s 664us/step - loss: 0.4167 - accuracy: 0.8260

Epoch 5/50

700/700 [==============================] - 0s 674us/step - loss: 0.4153 - accuracy: 0.8287

Epoch 6/50

700/700 [==============================] - 0s 653us/step - loss: 0.4137 - accuracy: 0.8310

Epoch 7/50

700/700 [==============================] - 0s 658us/step - loss: 0.4125 - accuracy: 0.8317

Epoch 8/50

700/700 [==============================] - 1s 842us/step - loss: 0.4116 - accuracy: 0.8306

Epoch 9/50

700/700 [==============================] - 0s 671us/step - loss: 0.4103 - accuracy: 0.8331

Epoch 10/50

700/700 [==============================] - 0s 682us/step - loss: 0.4100 - accuracy: 0.8326

Epoch 11/50

700/700 [==============================] - 0s 690us/step - loss: 0.4093 - accuracy: 0.8337

Epoch 12/50

700/700 [==============================] - 0s 688us/step - loss: 0.4087 - accuracy: 0.8339

Epoch 13/50

700/700 [==============================] - 0s 675us/step - loss: 0.4081 - accuracy: 0.8341

Epoch 14/50

700/700 [==============================] - 1s 722us/step - loss: 0.4071 - accuracy: 0.8331

Epoch 15/50

700/700 [==============================] - 1s 811us/step - loss: 0.4065 - accuracy: 0.8341

Epoch 16/50

700/700 [==============================] - 0s 711us/step - loss: 0.4056 - accuracy: 0.8356

Epoch 17/50

700/700 [==============================] - 0s 702us/step - loss: 0.4046 - accuracy: 0.8366

Epoch 18/50

700/700 [==============================] - 0s 688us/step - loss: 0.4035 - accuracy: 0.8343

Epoch 19/50

700/700 [==============================] - 1s 715us/step - loss: 0.4024 - accuracy: 0.8363

Epoch 20/50

700/700 [==============================] - 0s 714us/step - loss: 0.4020 - accuracy: 0.8337

Epoch 21/50

700/700 [==============================] - 0s 705us/step - loss: 0.4010 - accuracy: 0.8374

Epoch 22/50

700/700 [==============================] - 1s 720us/step - loss: 0.4003 - accuracy: 0.8370

Epoch 23/50

700/700 [==============================] - 0s 692us/step - loss: 0.3993 - accuracy: 0.8374

Epoch 24/50

700/700 [==============================] - 0s 709us/step - loss: 0.3990 - accuracy: 0.8356

Epoch 25/50

700/700 [==============================] - 1s 871us/step - loss: 0.3984 - accuracy: 0.8366

Epoch 26/50

700/700 [==============================] - 1s 719us/step - loss: 0.3984 - accuracy: 0.8367

Epoch 27/50

700/700 [==============================] - 1s 719us/step - loss: 0.3980 - accuracy: 0.8366

Epoch 28/50

700/700 [==============================] - 0s 695us/step - loss: 0.3981 - accuracy: 0.8366

Epoch 29/50

700/700 [==============================] - 0s 667us/step - loss: 0.3976 - accuracy: 0.8374

Epoch 30/50

700/700 [==============================] - 0s 669us/step - loss: 0.3972 - accuracy: 0.8373

Epoch 31/50

700/700 [==============================] - 0s 670us/step - loss: 0.3970 - accuracy: 0.8370

Epoch 32/50

700/700 [==============================] - 1s 720us/step - loss: 0.3972 - accuracy: 0.8376

Epoch 33/50

700/700 [==============================] - 0s 675us/step - loss: 0.3965 - accuracy: 0.8367

Epoch 34/50

700/700 [==============================] - 0s 680us/step - loss: 0.3961 - accuracy: 0.8364

Epoch 35/50

700/700 [==============================] - 0s 685us/step - loss: 0.3962 - accuracy: 0.8379

Epoch 36/50

700/700 [==============================] - 1s 771us/step - loss: 0.3960 - accuracy: 0.8370

Epoch 37/50

700/700 [==============================] - 1s 1ms/step - loss: 0.3963 - accuracy: 0.8366

Epoch 38/50

700/700 [==============================] - 1s 764us/step - loss: 0.3962 - accuracy: 0.8373

Epoch 39/50

700/700 [==============================] - 1s 823us/step - loss: 0.3950 - accuracy: 0.8384

Epoch 40/50

700/700 [==============================] - 1s 759us/step - loss: 0.3956 - accuracy: 0.8361

Epoch 41/50

700/700 [==============================] - 1s 773us/step - loss: 0.3949 - accuracy: 0.8366

Epoch 42/50

700/700 [==============================] - 0s 695us/step - loss: 0.3953 - accuracy: 0.8369

Epoch 43/50

700/700 [==============================] - 0s 701us/step - loss: 0.3952 - accuracy: 0.8369

Epoch 44/50

700/700 [==============================] - 0s 707us/step - loss: 0.3952 - accuracy: 0.8366

Epoch 45/50

700/700 [==============================] - 0s 680us/step - loss: 0.3955 - accuracy: 0.8376

Epoch 46/50

700/700 [==============================] - 0s 665us/step - loss: 0.3947 - accuracy: 0.8373

Epoch 47/50

700/700 [==============================] - 0s 708us/step - loss: 0.3947 - accuracy: 0.8371

Epoch 48/50

700/700 [==============================] - 0s 681us/step - loss: 0.3944 - accuracy: 0.8371

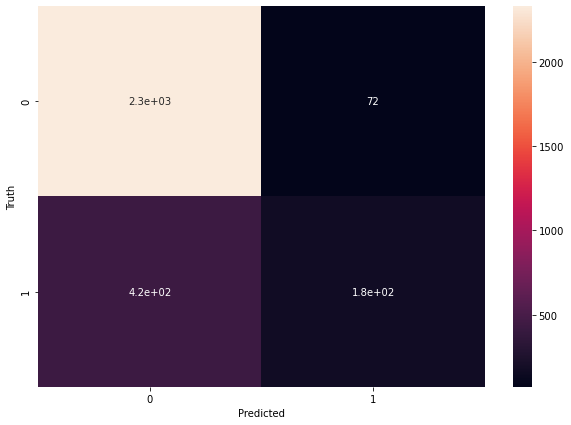
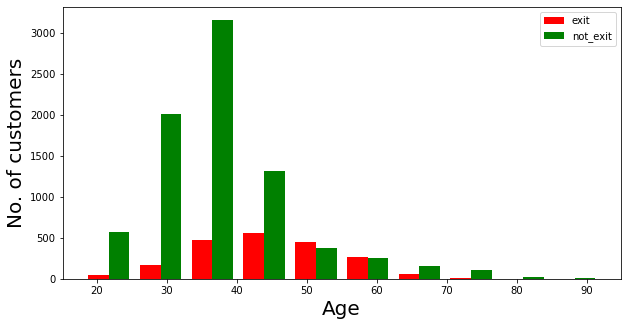
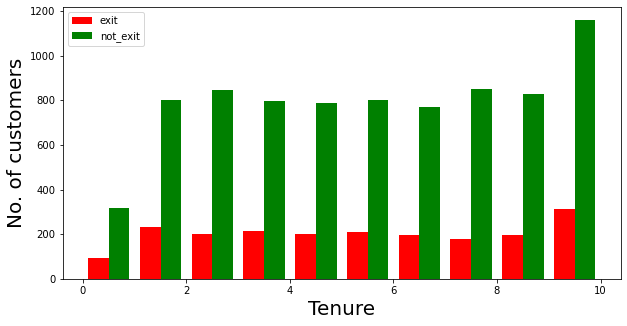
Epoch 49/50

700/700 [==============================] - 0s 678us/step - loss: 0.3947 - accuracy: 0.8383

Epoch 50/50

700/700 [==============================] - 1s 869us/step - loss: 0.3944 - accuracy: 0.8370

Out[89]: <tensorflow.python.keras.callbacks.History at 0x1fb1eb93df0>



In [90]:

y\_pred =classifier.predict(X\_test)

y\_pred = (y\_pred > 0.5) *#Predicting the result*

In [97]:

**from sklearn.metrics import** confusion\_matrix,accuracy\_score,classification\_report

In [92]:

cm = confusion\_matrix(y\_test,y\_pred)

In [93]:

cm

Out[93]: array([[2328, 72],

[ 425, 175]], dtype=int64)

In [94]:

accuracy = accuracy\_score(y\_test,y\_pred)

In [95]:

accuracy

Out[95]: 0.8343333333333334

In [98]:

plt.figure(figsize = (10,7)) sns.heatmap(cm,annot = **True**) plt.xlabel('Predicted') plt.ylabel('Truth')

Out[98]: Text(69.0, 0.5, 'Truth')

In [100]:

print(classification\_report(y\_test,y\_pred))

|  |  |  |
| --- | --- | --- |
|  | precision | recall f1-score support |
| 0 | 0.85 | 0.97 0.90 2400 |
| 1 | 0.71 | 0.29 0.41 600 |
| accuracy |  | 0.83 3000 |
| macro avg | 0.78 | 0.63 0.66 3000 |
| weighted avg | 0.82 | 0.83 0.81 3000 |