```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as seaborn
import warnings
warnings.filterwarnings("ignore")
```

```
df=pd.read_excel("Churn_Modelling.xlsx")
```

```
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	I
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8
2	3	15619304	Onio	502	France	Female	42	8	15
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12

```
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
```

```
df.drop(columns=["RowNumber","CustomerId","Surname"],inplace=True)
```

```
df
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCr(
0	619	France	Female	42	2	0.00	1	
1	608	Spain	Female	41	1	83807.86	1	
2	502	France	Female	42	8	159660.80	3	
3	699	France	Female	39	1	0.00	2	
4	850	Spain	Female	43	2	125510.82	1	
...	...	...	...	...	...	...	...	
9995	771	France	Male	39	5	0.00	2	
9996	516	France	Male	35	10	57369.61	1	
9997	709	France	Female	36	7	0.00	1	
9998	772	Germany	Male	42	3	75075.31	2	
9999	792	France	Female	28	4	130142.79	1	

10000 rows × 11 columns

```
from sklearn.preprocessing import LabelEncoder


le=LabelEncoder()
df["Geography"]=le.fit_transform(df["Geography"])


le1=LabelEncoder()
df["Gender"]=le1.fit_transform(df["Gender"])


df
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCr(
0	619	0	0	42	2	0.00	1	
1	608	2	0	41	1	83807.86	1	
2	502	0	0	42	8	159660.80	3	
3	699	0	0	39	1	0.00	2	
4	850	2	0	43	2	125510.82	1	
...	...	...	...	...	...	...	...	
9995	771	0	1	39	5	0.00	2	
9996	516	0	1	35	10	57369.61	1	
9997	709	0	0	36	7	0.00	1	
9998	772	1	1	42	3	75075.31	2	
9999	792	0	0	28	4	130142.79	1	

10000 rows × 11 columns

```
x=df.iloc[:,:-1]
x
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCr(
0	619	0	0	42	2	0.00	1	
1	608	2	0	41	1	83807.86	1	
2	502	0	0	42	8	159660.80	3	
3	699	0	0	39	1	0.00	2	
4	850	2	0	43	2	125510.82	1	
...	...	...	...	...	...	...	...	
9995	771	0	1	39	5	0.00	2	
9996	516	0	1	35	10	57369.61	1	
9997	709	0	0	36	7	0.00	1	
9998	772	1	1	42	3	75075.31	2	
9999	792	0	0	28	4	130142.79	1	

10000 rows × 10 columns

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x)


x
```

```
array([[-0.32622142, -0.90188624, -1.09598752, ...,  0.64609167,
         0.97024255,  0.02188649],
       [-0.44003595,  1.51506738, -1.09598752, ..., -1.54776799,
         0.97024255,  0.21653375],
       [-1.53679418, -0.90188624, -1.09598752, ...,  0.64609167,
        -1.03067011,  0.2406869 ],
       ...,
       [ 0.60498839, -0.90188624, -1.09598752, ..., -1.54776799,
         0.97024255, -1.00864308],
       [ 1.25683526,  0.30659057,  0.91241915, ...,  0.64609167,
```

```
       -1.03067011, -0.12523071],
      [ 1.46377078, -0.90188624, -1.09598752, ...,  0.64609167,
       -1.03067011, -1.07636976]])
```

```
y=df["Exited"].values
y
```

```
array([1, 0, 1, ..., 1, 1, 0])
```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.30,random_state=1)
```

```
import tensorflow as tf
from tensorflow.keras import Sequential # it is used to build NN
from tensorflow.keras.layers import Dense # it is used to add hidden layers
from sklearn.metrics import classification_report #evaulation
```

```
#builind NN
```

```
#step 1-: init a model
ann=Sequential()

#step2-: add layers to model
ann.add(Dense(units=12,activation="relu"))   #HL
ann.add(Dense(units=1,activation="sigmoid"))

#step3-: establish connection with layers
ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])

#step4-: fit the model
ann.fit(xtrain,ytrain,epochs=100,batch_size=32)

#step5-: make predictions
ypred=ann.predict(xtest)
```

```
Epoch 1/100
219/219 [==============================] - 1s 2ms/step - loss: 0.6460 - accuracy: 0.6351
Epoch 2/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4658 - accuracy: 0.7999
Epoch 3/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4357 - accuracy: 0.8109
Epoch 4/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4221 - accuracy: 0.8203
Epoch 5/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4114 - accuracy: 0.8274
Epoch 6/100
219/219 [==============================] - 0s 2ms/step - loss: 0.4014 - accuracy: 0.8324
Epoch 7/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3922 - accuracy: 0.8397
Epoch 8/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3839 - accuracy: 0.8457
Epoch 9/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3770 - accuracy: 0.8499
Epoch 10/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3714 - accuracy: 0.8506
Epoch 11/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3672 - accuracy: 0.8519
Epoch 12/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3639 - accuracy: 0.8526
Epoch 13/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3613 - accuracy: 0.8536
Epoch 14/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3597 - accuracy: 0.8551
Epoch 15/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3580 - accuracy: 0.8541
Epoch 16/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3568 - accuracy: 0.8559
Epoch 17/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3559 - accuracy: 0.8546
Epoch 18/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3550 - accuracy: 0.8543
Epoch 19/100
219/219 [==============================] - 1s 2ms/step - loss: 0.3544 - accuracy: 0.8567
Epoch 20/100
219/219 [==============================] - 1s 2ms/step - loss: 0.3537 - accuracy: 0.8561
Epoch 21/100
219/219 [==============================] - 1s 2ms/step - loss: 0.3531 - accuracy: 0.8556
```

```
Epoch 22/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3527 - accuracy: 0.8556
Epoch 23/100
219/219 [==============================] - 1s 2ms/step - loss: 0.3523 - accuracy: 0.8571
Epoch 24/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3516 - accuracy: 0.8566
Epoch 25/100
219/219 [==============================] - 0s 1ms/step - loss: 0.3509 - accuracy: 0.8576
Epoch 26/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3508 - accuracy: 0.8566
Epoch 27/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3501 - accuracy: 0.8576
Epoch 28/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3497 - accuracy: 0.8560
Epoch 29/100
219/219 [==============================] - 0s 2ms/step - loss: 0.3495 - accuracy: 0.8571
```

```
ypred
```

```
array([[0.03889292],
       [0.07804814],
       [0.07518935],
       ...,
       [0.03322417],
       [0.06540847],
       [0.03827015]], dtype=float32)
```

```
#step6-: Set the threshold
ypred=np.where(ypred>0.5,1,0)
ypred
```

```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [0]])
```

```
print(classification_report(ytest,ypred))
```

```
                  precision    recall  f1-score   support

               0       0.88      0.96      0.92      2373
               1       0.76      0.48      0.59       627

        accuracy                           0.86      3000
       macro avg       0.82      0.72      0.75      3000
    weighted avg       0.85      0.86      0.85      3000
```