```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")

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

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

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	1.0	15634602.0	Hargrave	619.0	France	Female	42.0	2.0	0
1	2.0	15647311.0	Hill	608.0	Spain	Female	41.0	1.0	83807
2	3.0	15619304.0	Onio	502.0	France	Female	42.0	8.0	159660
3	4.0	15701354.0	Boni	699.0	France	Female	39.0	1.0	0
4	5.0	15737888.0	Mitchell	850.0	Spain	Female	43.0	2.0	125510

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

```
x = df.iloc[:, 3:-1].values
x
```

```
array([[619.0, 'France', 'Female', ..., 1.0, 1.0, 101348.88],
       [608.0, 'Spain', 'Female', ..., 0.0, 1.0, 112542.58],
       [502.0, 'France', 'Female', ..., 1.0, 0.0, 113931.57],
       ...,
       [709.0, 'France', 'Female', ..., 0.0, 1.0, 42085.58],
       [772.0, 'Germany', 'Male', ..., 1.0, 0.0, 92888.52],
       [792.0, 'France', 'Female', ..., 1.0, 0.0, 38190.78]], dtype=object)
```

```
y = df.iloc[:, -1].values
y
```

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

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
x[:,1] = le.fit_transform(x[:,1])

le_1 = LabelEncoder()
x[:,2] = le_1.fit_transform(x[:,2])
```

```
x
```

```
array([[619.0, 0, 0, ..., 1.0, 1.0, 101348.88],
       [608.0, 2, 0, ..., 0.0, 1.0, 112542.58],
       [502.0, 0, 0, ..., 1.0, 0.0, 113931.57],
       ...,
       [709.0, 0, 0, ..., 0.0, 1.0, 42085.58],
       [772.0, 1, 1, ..., 1.0, 0.0, 92888.52],
       [792.0, 0, 0, ..., 1.0, 0.0, 38190.78]], dtype=object)
```

```
le.classes_
```

```
array(['France', 'Germany', 'Spain'], dtype=object)
```

```
le_1.classes_
```

```
array(['Female', 'Male'], dtype=object)
```

```
from scipy.sparse.construct import rand
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3, random_state=123)
```

```
xtrain
```

```
array([[648.0, 2, 1, ..., 0.0, 1.0, 181534.04],
       [693.0, 2, 0, ..., 1.0, 1.0, 135502.77],
       [586.0, 2, 0, ..., 1.0, 1.0, 168261.4],
       ...,
       [685.0, 0, 1, ..., 1.0, 0.0, 38691.34],
       [643.0, 0, 1, ..., 1.0, 1.0, 165614.4],
       [686.0, 0, 1, ..., 1.0, 0.0, 8816.37]], dtype=object)
```

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

xtest = sc.transform(xtest)
```

```
# STEP 1: Initiate the model
ann = Sequential()

# STEP 2: add layers into the model
ann.add(Dense(units=6, activation="relu")) # Created one hidden layer
ann.add(Dense(units=1, activation="sigmoid"))

# STEP 3: establish connection between the layers
ann.compile(optimizer="adam", loss="binary_crossentropy", metrics=["accuracy"])

# STEP 4: train the model
ann.fit(xtrain, ytrain, batch_size=30, epochs=100)

# STEP 5: make prediction
ypred = ann.predict(xtest)
```

```
Epoch 1/100
234/234 [==============================] - 1s 2ms/step - loss: 0.5972 - accuracy: 0.7453
Epoch 2/100
234/234 [==============================] - 0s 2ms/step - loss: 0.4991 - accuracy: 0.7966
Epoch 3/100
234/234 [==============================] - 0s 2ms/step - loss: 0.4641 - accuracy: 0.7974
Epoch 4/100
234/234 [==============================] - 0s 2ms/step - loss: 0.4430 - accuracy: 0.8060
Epoch 5/100
234/234 [==============================] - 0s 2ms/step - loss: 0.4304 - accuracy: 0.8146
Epoch 6/100
234/234 [==============================] - 1s 2ms/step - loss: 0.4219 - accuracy: 0.8194
Epoch 7/100
234/234 [==============================] - 1s 2ms/step - loss: 0.4142 - accuracy: 0.8247
Epoch 8/100
234/234 [==============================] - 1s 3ms/step - loss: 0.4072 - accuracy: 0.8297
Epoch 9/100
234/234 [==============================] - 0s 2ms/step - loss: 0.4006 - accuracy: 0.8334
Epoch 10/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3937 - accuracy: 0.8349
Epoch 11/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3880 - accuracy: 0.8393
Epoch 12/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3825 - accuracy: 0.8427
Epoch 13/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3778 - accuracy: 0.8449
Epoch 14/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3736 - accuracy: 0.8467
Epoch 15/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3695 - accuracy: 0.8479
Epoch 16/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3664 - accuracy: 0.8503
Epoch 17/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3640 - accuracy: 0.8514
Epoch 18/100
234/234 [==============================] - 1s 2ms/step - loss: 0.3617 - accuracy: 0.8513
Epoch 19/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3599 - accuracy: 0.8537
Epoch 20/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3586 - accuracy: 0.8530
Epoch 21/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3573 - accuracy: 0.8541
Epoch 22/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3561 - accuracy: 0.8547
Epoch 23/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3554 - accuracy: 0.8561
Epoch 24/100
234/234 [==============================] - 0s 1ms/step - loss: 0.3546 - accuracy: 0.8551
Epoch 25/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3539 - accuracy: 0.8559
Epoch 26/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3536 - accuracy: 0.8566
Epoch 27/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3529 - accuracy: 0.8561
Epoch 28/100
234/234 [==============================] - 0s 2ms/step - loss: 0.3525 - accuracy: 0.8559
Epoch 29/100
234/234 [==============================] - 0s 1ms/step - loss: 0.3524 - accuracy: 0.8550
```

```
# STEP 6: Set the threshold
ypred = np.where(ypred<0.5,0,1)
```

```
ypred
```

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

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

```
              precision    recall  f1-score   support

         0.0       0.88      0.97      0.92      2395
         1.0       0.80      0.47      0.59       605

    accuracy                           0.87      3000
   macro avg       0.84      0.72      0.76      3000
weighted avg       0.86      0.87      0.86      3000
```

```
df["Exited"].value_counts()
```

```
0.0    7963
1.0    2037
Name: Exited, dtype: int64
```

Checking the accuracy with Logistic regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(xtrain, ytrain)
ypred_lr = lr.predict(xtest)
```

```
print(classification_report(ytest, ypred_lr))
```

```
              precision    recall  f1-score   support

         0.0       0.82      0.97      0.89      2395
         1.0       0.61      0.16      0.25       605

    accuracy                           0.81      3000
   macro avg       0.72      0.57      0.57      3000
weighted avg       0.78      0.81      0.76      3000
```

✓ 0s completed at 15:25