



```
In [1]: import pandas as pd  
import numpy as np
```

```
In [2]: df=pd.read_csv('Churn_Modelling.csv')  
print(df.head())  
print(df.info())
```

```
      RowNumber CustomerId Surname CreditScore Geography Gender Age \\\n0           1   15634602 Hargrave        619    France Female  42\n1           2   15647311    Hill        608     Spain Female  41\n2           3   15619304   Onio        502    France Female  42\n3           4   15701354   Boni        699    France Female  39\n4           5   15737888 Mitchell       850     Spain Female  43\n\n      Tenure    Balance NumOfProducts HasCrCard IsActiveMember \\\n0       2      0.00             1          1                 1\n1       1    83807.86            1          0                 1\n2       8   159660.80            3          1                 0\n3       1      0.00             2          0                 0\n4       2   125510.82            1          1                 1\n\n      EstimatedSalary Exited\n0           101348.88     1\n1           112542.58     0\n2           113931.57     1\n3           93826.63      0\n4           79084.10      0\n<class 'pandas.core.frame.DataFrame'\nRangeIndex: 10000 entries, 0 to 9999\nData columns (total 14 columns):\n #   Column           Non-Null Count  Dtype \n---\n  0   RowNumber        10000 non-null   int64 \n  1   CustomerId       10000 non-null   int64 \n  2   Surname          10000 non-null   object \n  3   CreditScore      10000 non-null   int64 \n  4   Geography         10000 non-null   object \n  5   Gender            10000 non-null   object \n  6   Age               10000 non-null   int64 \n  7   Tenure            10000 non-null   int64 \n  8   Balance           10000 non-null   float64\n  9   NumOfProducts     10000 non-null   int64 \n  10  HasCrCard         10000 non-null   int64 \n  11  IsActiveMember    10000 non-null   int64 \n  12  EstimatedSalary   10000 non-null   float64\n  13  Exited            10000 non-null   int64 \n dtypes: float64(2), int64(9), object(3)\nmemory usage: 1.1+ MB\nNone
```

```
In [3]: df.isnull().sum()
```

```
Out[3]: 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 [10]: X = df.drop(['RowNumber', 'CustomerId', 'Surname', 'Exited'], axis=1)  
Y=df['Exited']
```

```
In [11]: #Encode categorical data  
# Encode 'Gender' and 'Geography' directly using pandas  
X = pd.get_dummies(X, columns=['Geography', 'Gender'], drop_first=True)
```

```
In [14]: X = X.astype(int)
```

```
In [16]: X.head(10)
```

```
Out[16]: CreditScore  Age  Tenure  Balance  NumOfProducts  HasCrCard  IsActiveMem  
0           619    42       2        0            1           1           1  
1           608    41       1      83807            1           1           0  
2           502    42       8     159660            3           1           1  
3           699    39       1        0            2           0           0  
4           850    43       2     125510            1           1           1  
5           645    44       8     113755            2           1           1  
6           822    50       7        0            2           1           1  
7           376    29       4     115046            4           1           1  
8           501    44       4     142051            2           0           0  
9           684    27       2     134603            1           1           1
```

```
In [17]: from sklearn.model_selection import train_test_split
```

```
In [18]: X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=
```

```
In [24]: #Normalize the data
```

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

```
In [25]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
In [26]: # Step 7: Build Neural Network Model
model = Sequential()
model.add(Dense(units=6, activation='relu', input_dim=X_train.shape[1])) # In
model.add(Dense(units=6, activation='relu')) # Second hidden layer
model.add(Dense(units=1, activation='sigmoid')) # Output layer (binary classi
```

```
C:\Users\suraj\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\core\dense.py:95: UserWarning: Do not pass an `input_shape`/`input_d
im` argument to a layer. When using Sequential models, prefer using an `Input(s
hape)` object as the first layer in the model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
In [27]: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
In [29]: model.fit(X_train, Y_train, batch_size=10, epochs=100, verbose=1)
```

Epoch 1/100  
**800/800** 1s 832us/step - accuracy: 0.7919 - loss: 0.5012  
Epoch 2/100  
**800/800** 1s 777us/step - accuracy: 0.8062 - loss: 0.4352  
Epoch 3/100  
**800/800** 1s 801us/step - accuracy: 0.8119 - loss: 0.4177  
Epoch 4/100  
**800/800** 1s 876us/step - accuracy: 0.8235 - loss: 0.4013  
Epoch 5/100  
**800/800** 1s 799us/step - accuracy: 0.8431 - loss: 0.3746  
Epoch 6/100  
**800/800** 1s 879us/step - accuracy: 0.8535 - loss: 0.3601  
Epoch 7/100  
**800/800** 1s 856us/step - accuracy: 0.8579 - loss: 0.3538  
Epoch 8/100  
**800/800** 1s 851us/step - accuracy: 0.8585 - loss: 0.3497  
Epoch 9/100  
**800/800** 1s 859us/step - accuracy: 0.8583 - loss: 0.3470  
Epoch 10/100  
**800/800** 1s 847us/step - accuracy: 0.8600 - loss: 0.3458  
Epoch 11/100  
**800/800** 1s 799us/step - accuracy: 0.8599 - loss: 0.3454  
Epoch 12/100  
**800/800** 1s 808us/step - accuracy: 0.8565 - loss: 0.3442  
Epoch 13/100  
**800/800** 1s 933us/step - accuracy: 0.8604 - loss: 0.3432  
Epoch 14/100  
**800/800** 1s 803us/step - accuracy: 0.8589 - loss: 0.3429  
Epoch 15/100  
**800/800** 1s 813us/step - accuracy: 0.8597 - loss: 0.3419  
Epoch 16/100  
**800/800** 1s 797us/step - accuracy: 0.8616 - loss: 0.3421  
Epoch 17/100  
**800/800** 1s 882us/step - accuracy: 0.8600 - loss: 0.3416  
Epoch 18/100  
**800/800** 1s 864us/step - accuracy: 0.8615 - loss: 0.3416  
Epoch 19/100  
**800/800** 1s 881us/step - accuracy: 0.8608 - loss: 0.3410  
Epoch 20/100  
**800/800** 1s 826us/step - accuracy: 0.8619 - loss: 0.3407  
Epoch 21/100  
**800/800** 1s 826us/step - accuracy: 0.8604 - loss: 0.3401  
Epoch 22/100  
**800/800** 1s 870us/step - accuracy: 0.8625 - loss: 0.3402  
Epoch 23/100  
**800/800** 1s 904us/step - accuracy: 0.8622 - loss: 0.3393  
Epoch 24/100  
**800/800** 1s 807us/step - accuracy: 0.8608 - loss: 0.3393  
Epoch 25/100  
**800/800** 1s 894us/step - accuracy: 0.8616 - loss: 0.3392  
Epoch 26/100  
**800/800** 1s 889us/step - accuracy: 0.8627 - loss: 0.3380  
Epoch 27/100  
**800/800** 1s 813us/step - accuracy: 0.8622 - loss: 0.3384

Epoch 28/100  
**800/800** 1s 990us/step - accuracy: 0.8627 - loss: 0.3382  
Epoch 29/100  
**800/800** 1s 900us/step - accuracy: 0.8624 - loss: 0.3378  
Epoch 30/100  
**800/800** 1s 894us/step - accuracy: 0.8618 - loss: 0.3371  
Epoch 31/100  
**800/800** 1s 801us/step - accuracy: 0.8626 - loss: 0.3379  
Epoch 32/100  
**800/800** 1s 809us/step - accuracy: 0.8621 - loss: 0.3373  
Epoch 33/100  
**800/800** 1s 906us/step - accuracy: 0.8601 - loss: 0.3374  
Epoch 34/100  
**800/800** 1s 1ms/step - accuracy: 0.8620 - loss: 0.3368  
Epoch 35/100  
**800/800** 1s 905us/step - accuracy: 0.8618 - loss: 0.3366  
Epoch 36/100  
**800/800** 1s 830us/step - accuracy: 0.8636 - loss: 0.3370  
Epoch 37/100  
**800/800** 1s 874us/step - accuracy: 0.8624 - loss: 0.3370  
Epoch 38/100  
**800/800** 1s 863us/step - accuracy: 0.8609 - loss: 0.3365  
Epoch 39/100  
**800/800** 1s 850us/step - accuracy: 0.8630 - loss: 0.3364  
Epoch 40/100  
**800/800** 1s 942us/step - accuracy: 0.8616 - loss: 0.3363  
Epoch 41/100  
**800/800** 1s 777us/step - accuracy: 0.8618 - loss: 0.3367  
Epoch 42/100  
**800/800** 1s 945us/step - accuracy: 0.8629 - loss: 0.3358  
Epoch 43/100  
**800/800** 1s 951us/step - accuracy: 0.8639 - loss: 0.3360  
Epoch 44/100  
**800/800** 1s 919us/step - accuracy: 0.8619 - loss: 0.3360  
Epoch 45/100  
**800/800** 1s 879us/step - accuracy: 0.8650 - loss: 0.3357  
Epoch 46/100  
**800/800** 1s 926us/step - accuracy: 0.8621 - loss: 0.3357  
Epoch 47/100  
**800/800** 1s 879us/step - accuracy: 0.8631 - loss: 0.3355  
Epoch 48/100  
**800/800** 1s 1ms/step - accuracy: 0.8619 - loss: 0.3349  
Epoch 49/100  
**800/800** 1s 990us/step - accuracy: 0.8627 - loss: 0.3355  
Epoch 50/100  
**800/800** 1s 1ms/step - accuracy: 0.8640 - loss: 0.3345  
Epoch 51/100  
**800/800** 1s 1ms/step - accuracy: 0.8612 - loss: 0.3353  
Epoch 52/100  
**800/800** 1s 1ms/step - accuracy: 0.8621 - loss: 0.3343  
Epoch 53/100  
**800/800** 1s 954us/step - accuracy: 0.8625 - loss: 0.3353  
Epoch 54/100  
**800/800** 1s 1ms/step - accuracy: 0.8626 - loss: 0.3353

Epoch 55/100  
**800/800** 1s 1ms/step - accuracy: 0.8644 - loss: 0.3349  
Epoch 56/100  
**800/800** 1s 1ms/step - accuracy: 0.8641 - loss: 0.3346  
Epoch 57/100  
**800/800** 1s 1ms/step - accuracy: 0.8618 - loss: 0.3344  
Epoch 58/100  
**800/800** 1s 1ms/step - accuracy: 0.8611 - loss: 0.3353  
Epoch 59/100  
**800/800** 1s 935us/step - accuracy: 0.8636 - loss: 0.3347  
Epoch 60/100  
**800/800** 1s 1ms/step - accuracy: 0.8624 - loss: 0.3347  
Epoch 61/100  
**800/800** 1s 1ms/step - accuracy: 0.8620 - loss: 0.3347  
Epoch 62/100  
**800/800** 1s 1ms/step - accuracy: 0.8635 - loss: 0.3348  
Epoch 63/100  
**800/800** 1s 1ms/step - accuracy: 0.8620 - loss: 0.3344  
Epoch 64/100  
**800/800** 1s 1ms/step - accuracy: 0.8637 - loss: 0.3340  
Epoch 65/100  
**800/800** 1s 1ms/step - accuracy: 0.8635 - loss: 0.3337  
Epoch 66/100  
**800/800** 1s 981us/step - accuracy: 0.8627 - loss: 0.3344  
Epoch 67/100  
**800/800** 1s 1ms/step - accuracy: 0.8605 - loss: 0.3347  
Epoch 68/100  
**800/800** 1s 1ms/step - accuracy: 0.8602 - loss: 0.3337  
Epoch 69/100  
**800/800** 1s 1ms/step - accuracy: 0.8634 - loss: 0.3346  
Epoch 70/100  
**800/800** 1s 1ms/step - accuracy: 0.8629 - loss: 0.3340  
Epoch 71/100  
**800/800** 1s 1ms/step - accuracy: 0.8621 - loss: 0.3342  
Epoch 72/100  
**800/800** 1s 1ms/step - accuracy: 0.8636 - loss: 0.3343  
Epoch 73/100  
**800/800** 1s 1ms/step - accuracy: 0.8649 - loss: 0.3340  
Epoch 74/100  
**800/800** 1s 1ms/step - accuracy: 0.8631 - loss: 0.3341  
Epoch 75/100  
**800/800** 1s 948us/step - accuracy: 0.8633 - loss: 0.3341  
Epoch 76/100  
**800/800** 1s 1ms/step - accuracy: 0.8620 - loss: 0.3338  
Epoch 77/100  
**800/800** 1s 1ms/step - accuracy: 0.8621 - loss: 0.3344  
Epoch 78/100  
**800/800** 1s 1ms/step - accuracy: 0.8640 - loss: 0.3343  
Epoch 79/100  
**800/800** 1s 1ms/step - accuracy: 0.8637 - loss: 0.3335  
Epoch 80/100  
**800/800** 1s 1ms/step - accuracy: 0.8630 - loss: 0.3341  
Epoch 81/100  
**800/800** 1s 1ms/step - accuracy: 0.8624 - loss: 0.3330

```
Epoch 82/100
800/800 1s 1ms/step - accuracy: 0.8622 - loss: 0.3339
Epoch 83/100
800/800 1s 1ms/step - accuracy: 0.8611 - loss: 0.3340
Epoch 84/100
800/800 1s 1ms/step - accuracy: 0.8629 - loss: 0.3336
Epoch 85/100
800/800 1s 1ms/step - accuracy: 0.8630 - loss: 0.3338
Epoch 86/100
800/800 1s 1ms/step - accuracy: 0.8611 - loss: 0.3337
Epoch 87/100
800/800 1s 1ms/step - accuracy: 0.8639 - loss: 0.3335
Epoch 88/100
800/800 1s 1ms/step - accuracy: 0.8641 - loss: 0.3336
Epoch 89/100
800/800 1s 1ms/step - accuracy: 0.8619 - loss: 0.3337
Epoch 90/100
800/800 1s 1ms/step - accuracy: 0.8608 - loss: 0.3335
Epoch 91/100
800/800 1s 1ms/step - accuracy: 0.8619 - loss: 0.3339
Epoch 92/100
800/800 1s 1ms/step - accuracy: 0.8612 - loss: 0.3340
Epoch 93/100
800/800 1s 1ms/step - accuracy: 0.8631 - loss: 0.3337
Epoch 94/100
800/800 1s 1ms/step - accuracy: 0.8621 - loss: 0.3336
Epoch 95/100
800/800 1s 1ms/step - accuracy: 0.8631 - loss: 0.3333
Epoch 96/100
800/800 1s 1ms/step - accuracy: 0.8631 - loss: 0.3334
Epoch 97/100
800/800 1s 1ms/step - accuracy: 0.8620 - loss: 0.3338
Epoch 98/100
800/800 1s 1ms/step - accuracy: 0.8618 - loss: 0.3338
Epoch 99/100
800/800 1s 1ms/step - accuracy: 0.8627 - loss: 0.3336
Epoch 100/100
800/800 1s 1ms/step - accuracy: 0.8625 - loss: 0.3332
```

Out[29]: <keras.src.callbacks.history.History at 0x21efea99e80>

In [32]: `Y_pred=(model.predict(X_test) > 0.5)`

```
63/63 0s 3ms/step
```

In [33]: `from sklearn.metrics import confusion_matrix,accuracy_score`

In [34]: `print(confusion_matrix(Y_test,Y_pred))  
print(accuracy_score(Y_test,Y_pred))`

```
[[1540  67]  
 [ 208 185]]  
0.8625
```

In [ ]: