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

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

Out[2]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	0
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	1
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0

10000 rows × 14 columns

In [3]:
```
df.drop(columns = ['RowNumber','CustomerId','Surname'], inplace=True)
```

In [4]:
```
df
```

Out[4]:

	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
...	...	...	...	...	...	...	...	...	...	...	...
9995	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 11 columns

In [5]:
```
geography = pd.get_dummies(df['Geography'])
gender = pd.get_dummies(df['Gender'])
```

In [6]:
```
df = pd.concat([df,geography,gender],axis=1)
df
```

Out[6]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	France	Ger
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1	True	
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0	False	
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1	True	
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0	True	
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	771	France	Male	39	5	0.00	2	1	0	96270.64	0	True	
9996	516	France	Male	35	10	57369.61	1	1	1	101699.77	0	True	
9997	709	France	Female	36	7	0.00	1	0	1	42085.58	1	True	
9998	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1	False	
9999	792	France	Female	28	4	130142.79	1	1	0	38190.78	0	True	

10000 rows × 16 columns

In [7]:
```
df.drop(columns=['Geography','Gender'],inplace=True)
```

In [8]:
```
df
```

Out[8]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	France	Germany	Spain	Female
0	619	42	2	0.00	1	1	1	101348.88	1	True	False	False	True
1	608	41	1	83807.86	1	0	1	112542.58	0	False	False	True	True
2	502	42	8	159660.80	3	1	0	113931.57	1	True	False	False	True
3	699	39	1	0.00	2	0	0	93826.63	0	True	False	False	True
4	850	43	2	125510.82	1	1	1	79084.10	0	False	False	True	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64	0	True	False	False	False
9996	516	35	10	57369.61	1	1	1	101699.77	0	True	False	False	False
9997	709	36	7	0.00	1	0	1	42085.58	1	True	False	False	True
9998	772	42	3	75075.31	2	1	0	92888.52	1	False	True	False	False
9999	792	28	4	130142.79	1	1	0	38190.78	0	True	False	False	True

10000 rows × 14 columns

In [12]:
```
X = df.drop(['Exited'],axis=1)
y = df['Exited']
```

In [13]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,y, test_size=0.25)
```

In [14]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)
```

In [18]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

In [23]:
```
classifier = Sequential()
classifier.add(Dense(activation='relu', input_dim=13, units=6, kernel_initializer='uniform')) #input layer
classifier.add(Dense(activation='relu', units=6, kernel_initializer='uniform')) #hidden layer
classifier.add(Dense(activation='sigmoid', units=1)) #output layer
#sigmoid for binary classification and softmax for more than 2 classification
classifier.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
classifier.summary()
```

```
/home/student/.local/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `inpu
t_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the f
irst layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_8 (Dense)	(None, 6)	84
dense_9 (Dense)	(None, 6)	42
dense_10 (Dense)	(None, 1)	7

Total params: 133 (532.00 B)

Trainable params: 133 (532.00 B)

Non-trainable params: 0 (0.00 B)

In [24]:
```
classifier.fit(x_train,y_train, epochs=50, batch_size=10)
```

```
Epoch 1/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.8007 - loss: 0.5538
Epoch 2/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 938us/step - accuracy: 0.7970 - loss: 0.4323
Epoch 3/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 838us/step - accuracy: 0.8052 - loss: 0.4281
Epoch 4/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 938us/step - accuracy: 0.8292 - loss: 0.4173
Epoch 5/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 863us/step - accuracy: 0.8389 - loss: 0.4041
Epoch 6/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 940us/step - accuracy: 0.8341 - loss: 0.4087
Epoch 7/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 905us/step - accuracy: 0.8393 - loss: 0.4012
Epoch 8/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 986us/step - accuracy: 0.8385 - loss: 0.3932
Epoch 9/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 938us/step - accuracy: 0.8393 - loss: 0.3978
Epoch 10/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 919us/step - accuracy: 0.8305 - loss: 0.4063
Epoch 11/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 855us/step - accuracy: 0.8408 - loss: 0.3871
Epoch 12/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 941us/step - accuracy: 0.8375 - loss: 0.3966
Epoch 13/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 891us/step - accuracy: 0.8388 - loss: 0.3909
Epoch 14/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8398 - loss: 0.3895
Epoch 15/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 927us/step - accuracy: 0.8387 - loss: 0.3992
Epoch 16/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 820us/step - accuracy: 0.8304 - loss: 0.4065
Epoch 17/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 789us/step - accuracy: 0.8404 - loss: 0.3908
Epoch 18/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 842us/step - accuracy: 0.8449 - loss: 0.3904
Epoch 19/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8431 - loss: 0.3826
Epoch 20/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 847us/step - accuracy: 0.8473 - loss: 0.3813
Epoch 21/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8414 - loss: 0.3871
Epoch 22/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 961us/step - accuracy: 0.8372 - loss: 0.3966
Epoch 23/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 857us/step - accuracy: 0.8426 - loss: 0.3888
Epoch 24/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 861us/step - accuracy: 0.8444 - loss: 0.3826
Epoch 25/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 932us/step - accuracy: 0.8392 - loss: 0.3850
Epoch 26/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 869us/step - accuracy: 0.8469 - loss: 0.3842
Epoch 27/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 884us/step - accuracy: 0.8391 - loss: 0.3882
Epoch 28/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 851us/step - accuracy: 0.8368 - loss: 0.3933
Epoch 29/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 900us/step - accuracy: 0.8362 - loss: 0.3954
Epoch 30/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 967us/step - accuracy: 0.8428 - loss: 0.3944
Epoch 31/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.8395 - loss: 0.3887
Epoch 32/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8415 - loss: 0.3821
Epoch 33/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 923us/step - accuracy: 0.8412 - loss: 0.3858
Epoch 34/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 826us/step - accuracy: 0.8487 - loss: 0.3741
Epoch 35/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 808us/step - accuracy: 0.8410 - loss: 0.3911
Epoch 36/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 945us/step - accuracy: 0.8431 - loss: 0.3765
Epoch 37/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8430 - loss: 0.3798
Epoch 38/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 859us/step - accuracy: 0.8411 - loss: 0.3843
Epoch 39/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 890us/step - accuracy: 0.8404 - loss: 0.3883
Epoch 40/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 953us/step - accuracy: 0.8400 - loss: 0.3900
Epoch 41/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 880us/step - accuracy: 0.8420 - loss: 0.3792
Epoch 42/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 850us/step - accuracy: 0.8364 - loss: 0.3936
Epoch 43/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 889us/step - accuracy: 0.8419 - loss: 0.3843
Epoch 44/50
```

```
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 982us/step - accuracy: 0.8416 - loss: 0.3866
Epoch 45/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 915us/step - accuracy: 0.8441 - loss: 0.3882
Epoch 46/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 866us/step - accuracy: 0.8446 - loss: 0.3845
Epoch 47/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.8435 - loss: 0.3821
Epoch 48/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 911us/step - accuracy: 0.8439 - loss: 0.3855
Epoch 49/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 894us/step - accuracy: 0.8469 - loss: 0.3837
Epoch 50/50
750/750 ━━━━━━━━━━━━━━━━━━━━ 1s 866us/step - accuracy: 0.8414 - loss: 0.3850
```

Out[24]:
```
<keras.src.callbacks.history.History at 0x7b685676bbb0>
```

In [27]:
```
F_pred = classifier.predict(x_test)
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 784us/step
```

In [28]:
```
y_pred = (F_pred>0.5)
```

In [29]:
```
print(y_pred)
```

```
[[False]
 [False]
 [False]
 ...
 [False]
 [False]
 [False]]
```

In [30]:
```
from sklearn.metrics import confusion_matrix,precision_score,recall_score,f1_score,accuracy_score
```

In [31]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[31]:
```
array([[1918,   45],
       [ 391,  146]])
```

In [33]:
```
acc = accuracy_score(y_test,y_pred)
acc
```

Out[33]:
```
0.8256
```

In [34]:
```
precision = precision_score(y_test,y_pred)
precision
```

Out[34]:
```
0.7643979057591623
```

In [35]:
```
recall = recall_score(y_test,y_pred)
recall
```

Out[35]:
```
0.2718808193668529
```

In [36]:
```
f1= f1_score(y_test,y_pred)
f1
```

Out[36]:
```
0.4010989010989011
```

In [ ]:
```

```