In [2]:

```python


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

In [3]:

```python
df = pd.read_csv('Churn_Modelling.csv')
df.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	838
2	3	15619304	Onio	502	France	Female	42	8	1596
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1255

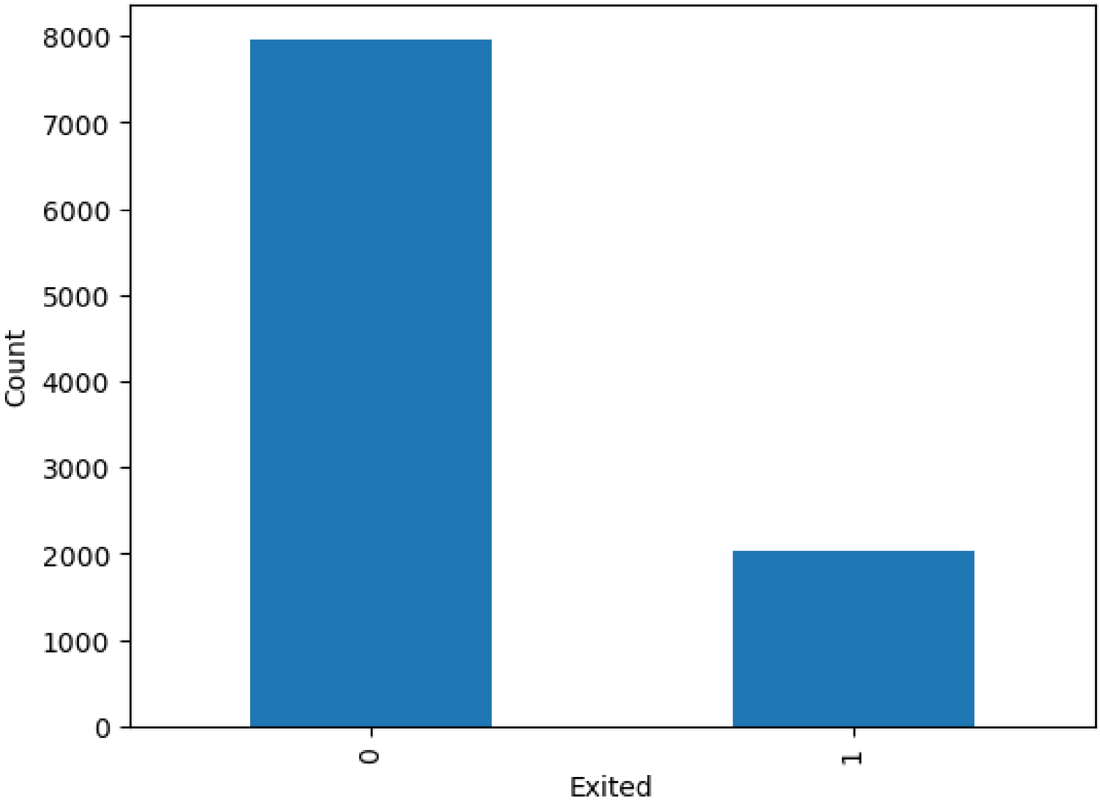
In [4]:

```python
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
```

In [5]:

```
plt.xlabel('Exited')
plt.ylabel('Count')
df['Exited'].value_counts().plot.bar()
plt.show()
```



In [6]:

```
df['Geography'].value_counts()
```

Out[6]:

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [7]:

```
df = pd.concat([df,pd.get_dummies(df['Geography'],prefix='Geo')],axis=1)
```

In [8]:

```
df = pd.concat([df,pd.get_dummies(df['Gender'])],axis=1)
```

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 19 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  
 14  Geo_France       10000 non-null  uint8  
 15  Geo_Germany      10000 non-null  uint8  
 16  Geo_Spain        10000 non-null  uint8  
 17  Female           10000 non-null  uint8  
 18  Male             10000 non-null  uint8  
dtypes: float64(2), int64(9), object(3), uint8(5)
memory usage: 1.1+ MB
```

In [10]:

```
df.drop(columns=['RowNumber','CustomerId','Surname','Geography','Gender'],
```

In [11]:

```
df.head()
```

Out[11]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estin
0	619	42	2	0.00	1	1	1	
1	608	41	1	83807.86	1	0	1	
2	502	42	8	159660.80	3	1	0	
3	699	39	1	0.00	2	0	0	
4	850	43	2	125510.82	1	1	1	

Splitting Data

In [12]:

```
y = df['Exited'].values
x = df.loc[:,df.columns != 'Exited'].values
```

In [13]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=20,test_
```

Scaling Data

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

In [15]:
```
x_train.shape
```

Out[15]:
```
(7500, 13)
```

Tensorflow Model - Neural Network Classifier

In [16]:
```
import tensorflow as tf
from tensorflow.keras.layers import Dense,Conv1D,Flatten
from tensorflow.keras.models import Sequential, Model
```

In [17]:
```
model=Sequential()
model.add(Flatten(input_shape=(13,)))
model.add(Dense(100,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

In [18]:
```
model.compile(optimizer='adam',metrics=['accuracy'],loss='BinaryCrossentro
```

In [19]:
```
model.fit(x_train,y_train,batch_size=64,validation_split=0.1,epochs=100)
```

```
Epoch 1/100
106/106 [==============================] - 1s 3ms/step - loss: 0.5127
- accuracy: 0.7621 - val_loss: 0.4269 - val_accuracy: 0.8240
Epoch 2/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4306
- accuracy: 0.8119 - val_loss: 0.4042 - val_accuracy: 0.8280
Epoch 3/100
106/106 [==============================] - 0s 2ms/step - loss: 0.4129
- accuracy: 0.8204 - val_loss: 0.3825 - val_accuracy: 0.8400
Epoch 4/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3970
- accuracy: 0.8323 - val_loss: 0.3681 - val_accuracy: 0.8560
Epoch 5/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3826
- accuracy: 0.8413 - val_loss: 0.3493 - val_accuracy: 0.8653
Epoch 6/100
106/106 [==============================] - 0s 2ms/step - loss: 0.3712
- accuracy: 0.8428 - val_loss: 0.3389 - val_accuracy: 0.8640
Epoch 7/100
```

In [20]:
```
pred = model.predict(x_test)
```

```
79/79 [==============================] - 0s 3ms/step
```

In [21]:
```
y_pred = []
for val in pred:
    if val > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

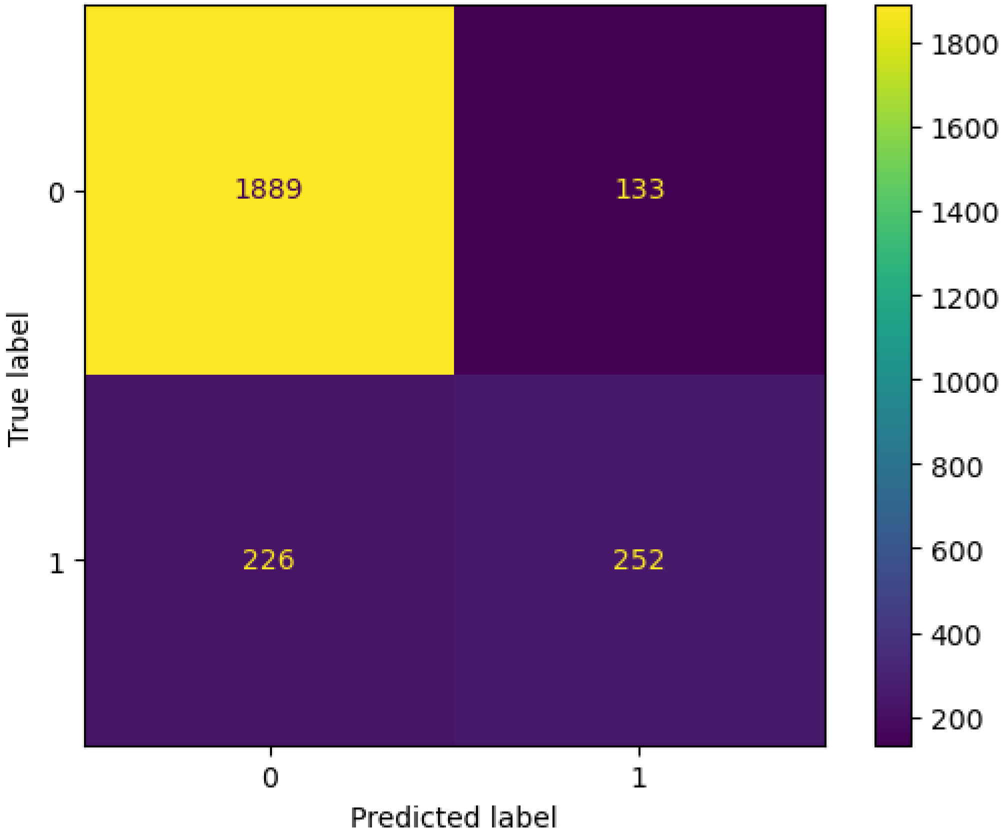
In [22]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,ConfusionMatri
```

In [23]:
```
accuracy_score(y_test,y_pred)
```

Out[23]:
```
0.8564
```

In [24]:
```
cm = confusion_matrix(y_test,y_pred)
display = ConfusionMatrixDisplay(cm)
display.plot()
```

Out[24]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2a6b0
bb7070>
```



In [25]:
```
from sklearn.neural_network import MLPClassifier
```

In [26]:

```
nn_classifier = MLPClassifier(hidden_layer_sizes=(100),activation='logisti
nn_classifier.fit(x_train,y_train)
```

```
c:\Users\hp\anaconda3\lib\site-packages\sklearn\neural_network\_multilaye
r_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum it
erations (300) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[26]:

```
MLPClassifier(activation='logistic', hidden_layer_sizes=100, max_iter=30
0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [27]:

```
y_pred2 = nn_classifier.predict(x_test)
```

In [28]:

```
accuracy_score(y_pred=y_pred2,y_true=y_test)
```

Out[28]:

```
0.8648
```

In [29]:

```
nn_classifier.score(x_test,y_test)
```

Out[29]:

```
0.8648
```

In [ ]: