Name : Aditya R. Kotame

Div : A

Roll No : 49

In [7]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [8]:
```
df = pd.read_csv('/home/student/Downloads/Churn_Modelling.csv')
df.head()
```

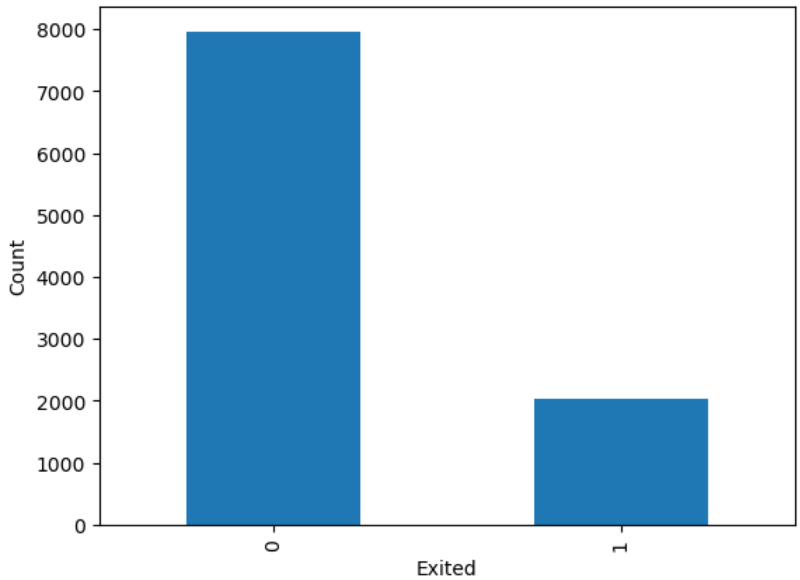
Out[8]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	

In [9]:
```
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 [10]:
```
plt.xlabel('Exited')
plt.ylabel('Count')
df['Exited'].value_counts().plot.bar()
plt.show()
```



In [11]:
```
df['Geography'].value_counts()
```

Out[11]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [12]:
```
df = pd.concat([df,pd.get_dummies(df['Geography'],prefix='Geo')],axis=1)
df = pd.concat([df,pd.get_dummies(df['Gender'])],axis=1)
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 [13]:
```
df.drop(columns=['RowNumber','CustomerId','Surname','Geography','Gender'],inplace=True)
df.head()
```

Out[13]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geo_France	Geo_Germany	G
0	619	42	2	0.00	1	1	1	101348.88	1	1	0	
1	608	41	1	83807.86	1	0	1	112542.58	0	0	0	
2	502	42	8	159660.80	3	1	0	113931.57	1	1	0	
3	699	39	1	0.00	2	0	0	93826.63	0	1	0	
4	850	43	2	125510.82	1	1	1	79084.10	0	0	0	

In [15]:
```
y = df['Exited'].values
x = df.loc[:,df.columns != 'Exited'].values
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_size=0.25)
```

In [16]:
```
from sklearn.preprocessing import StandardScaler
std_x = StandardScaler()
x_train = std_x.fit_transform(x_train)
x_test = std_x.transform(x_test)
```

In [17]:
```
x_train.shape
```

Out[17]:
```
(7500, 13)
```

In [ ]:

In [18]:
```
import tensorflow as tf
from tensorflow.keras.layers import Dense,Conv1D,Flatten
from tensorflow.keras.models import Sequential, Model

model=Sequential()
model.add(Flatten(input_shape=(13,)))
model.add(Dense(100,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',metrics=['accuracy'],loss='BinaryCrossentropy')

model.fit(x_train,y_train,batch_size=64,validation_split=0.1,epochs=100)
```

```
Epoch 1/100
/home/student/anaconda3/lib/python3.10/site-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do
not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input
(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
106/106 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.6244 - loss: 0.6341 - val_accuracy: 0.8227 - val_loss:
0.4371
Epoch 2/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 601us/step - accuracy: 0.8080 - loss: 0.4434 - val_accuracy: 0.8280 - val_loss:
0.4091
Epoch 3/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 687us/step - accuracy: 0.8188 - loss: 0.4148 - val_accuracy: 0.8360 - val_loss:
0.3886
Epoch 4/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 559us/step - accuracy: 0.8206 - loss: 0.4051 - val_accuracy: 0.8480 - val_loss:
0.3706
Epoch 5/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 518us/step - accuracy: 0.8352 - loss: 0.3880 - val_accuracy: 0.8600 - val_loss:
0.3566
Epoch 6/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 647us/step - accuracy: 0.8425 - loss: 0.3742 - val_accuracy: 0.8680 - val_loss:
0.3447
Epoch 7/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 564us/step - accuracy: 0.8395 - loss: 0.3802 - val_accuracy: 0.8640 - val_loss:
0.3381
Epoch 8/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8509 - loss: 0.3583 - val_accuracy: 0.8760 - val_loss:
0.3278
Epoch 9/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 660us/step - accuracy: 0.8478 - loss: 0.3555 - val_accuracy: 0.8707 - val_loss:
0.3264
Epoch 10/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 907us/step - accuracy: 0.8550 - loss: 0.3467 - val_accuracy: 0.8747 - val_loss:
0.3203
Epoch 11/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 606us/step - accuracy: 0.8554 - loss: 0.3520 - val_accuracy: 0.8707 - val_loss:
0.3202
Epoch 12/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 573us/step - accuracy: 0.8561 - loss: 0.3442 - val_accuracy: 0.8800 - val_loss:
0.3184
Epoch 13/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 794us/step - accuracy: 0.8627 - loss: 0.3365 - val_accuracy: 0.8787 - val_loss:
0.3184
Epoch 14/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 516us/step - accuracy: 0.8649 - loss: 0.3363 - val_accuracy: 0.8827 - val_loss:
0.3176
Epoch 15/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 855us/step - accuracy: 0.8474 - loss: 0.3563 - val_accuracy: 0.8827 - val_loss:
0.3139
Epoch 16/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8599 - loss: 0.3399 - val_accuracy: 0.8773 - val_loss:
0.3145
Epoch 17/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 573us/step - accuracy: 0.8607 - loss: 0.3377 - val_accuracy: 0.8760 - val_loss:
0.3177
Epoch 18/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 793us/step - accuracy: 0.8585 - loss: 0.3421 - val_accuracy: 0.8840 - val_loss:
0.3147
Epoch 19/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 999us/step - accuracy: 0.8578 - loss: 0.3437 - val_accuracy: 0.8787 - val_loss:
0.3171
Epoch 20/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 805us/step - accuracy: 0.8580 - loss: 0.3436 - val_accuracy: 0.8773 - val_loss:
0.3157
Epoch 21/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 546us/step - accuracy: 0.8595 - loss: 0.3417 - val_accuracy: 0.8813 - val_loss:
0.3118
Epoch 22/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 538us/step - accuracy: 0.8597 - loss: 0.3330 - val_accuracy: 0.8853 - val_loss:
0.3128
Epoch 23/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 766us/step - accuracy: 0.8699 - loss: 0.3201 - val_accuracy: 0.8853 - val_loss:
0.3136
Epoch 24/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 785us/step - accuracy: 0.8527 - loss: 0.3471 - val_accuracy: 0.8880 - val_loss:
0.3098
Epoch 25/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 882us/step - accuracy: 0.8600 - loss: 0.3425 - val_accuracy: 0.8800 - val_loss:
0.3129
Epoch 26/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 571us/step - accuracy: 0.8642 - loss: 0.3355 - val_accuracy: 0.8747 - val_loss:
0.3150
Epoch 27/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 585us/step - accuracy: 0.8576 - loss: 0.3434 - val_accuracy: 0.8787 - val_loss:
0.3130
Epoch 28/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 673us/step - accuracy: 0.8679 - loss: 0.3269 - val_accuracy: 0.8733 - val_loss:
0.3153
Epoch 29/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - accuracy: 0.8631 - loss: 0.3369 - val_accuracy: 0.8813 - val_loss:
0.3156
Epoch 30/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8629 - loss: 0.3355 - val_accuracy: 0.8800 - val_loss:
0.3121
Epoch 31/100
```

```
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8639 - loss: 0.3266 - val_accuracy: 0.8867 - val_loss:
0.3079
Epoch 32/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 971us/step - accuracy: 0.8656 - loss: 0.3354 - val_accuracy: 0.8813 - val_loss:
0.3107
Epoch 33/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 520us/step - accuracy: 0.8649 - loss: 0.3294 - val_accuracy: 0.8760 - val_loss:
0.3198
Epoch 34/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8612 - loss: 0.3262 - val_accuracy: 0.8760 - val_loss:
0.3181
Epoch 35/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 604us/step - accuracy: 0.8667 - loss: 0.3189 - val_accuracy: 0.8787 - val_loss:
0.3132
Epoch 36/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 567us/step - accuracy: 0.8647 - loss: 0.3263 - val_accuracy: 0.8773 - val_loss:
0.3117
Epoch 37/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 711us/step - accuracy: 0.8671 - loss: 0.3280 - val_accuracy: 0.8773 - val_loss:
0.3157
Epoch 38/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 545us/step - accuracy: 0.8720 - loss: 0.3199 - val_accuracy: 0.8760 - val_loss:
0.3151
Epoch 39/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 502us/step - accuracy: 0.8587 - loss: 0.3290 - val_accuracy: 0.8827 - val_loss:
0.3127
Epoch 40/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 727us/step - accuracy: 0.8677 - loss: 0.3182 - val_accuracy: 0.8667 - val_loss:
0.3210
Epoch 41/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 730us/step - accuracy: 0.8692 - loss: 0.3229 - val_accuracy: 0.8760 - val_loss:
0.3156
Epoch 42/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 510us/step - accuracy: 0.8631 - loss: 0.3295 - val_accuracy: 0.8787 - val_loss:
0.3111
Epoch 43/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 484us/step - accuracy: 0.8642 - loss: 0.3244 - val_accuracy: 0.8800 - val_loss:
0.3089
Epoch 44/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 640us/step - accuracy: 0.8722 - loss: 0.3179 - val_accuracy: 0.8813 - val_loss:
0.3124
Epoch 45/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 552us/step - accuracy: 0.8655 - loss: 0.3259 - val_accuracy: 0.8760 - val_loss:
0.3130
Epoch 46/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 629us/step - accuracy: 0.8644 - loss: 0.3296 - val_accuracy: 0.8800 - val_loss:
0.3071
Epoch 47/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 581us/step - accuracy: 0.8698 - loss: 0.3134 - val_accuracy: 0.8853 - val_loss:
0.3088
Epoch 48/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 521us/step - accuracy: 0.8615 - loss: 0.3311 - val_accuracy: 0.8733 - val_loss:
0.3152
Epoch 49/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 486us/step - accuracy: 0.8708 - loss: 0.3164 - val_accuracy: 0.8747 - val_loss:
0.3109
Epoch 50/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 515us/step - accuracy: 0.8577 - loss: 0.3352 - val_accuracy: 0.8773 - val_loss:
0.3160
Epoch 51/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 546us/step - accuracy: 0.8649 - loss: 0.3210 - val_accuracy: 0.8787 - val_loss:
0.3116
Epoch 52/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 729us/step - accuracy: 0.8690 - loss: 0.3223 - val_accuracy: 0.8800 - val_loss:
0.3088
Epoch 53/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 580us/step - accuracy: 0.8634 - loss: 0.3200 - val_accuracy: 0.8840 - val_loss:
0.3077
Epoch 54/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 551us/step - accuracy: 0.8691 - loss: 0.3208 - val_accuracy: 0.8813 - val_loss:
0.3102
Epoch 55/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 521us/step - accuracy: 0.8693 - loss: 0.3114 - val_accuracy: 0.8800 - val_loss:
0.3113
Epoch 56/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 532us/step - accuracy: 0.8753 - loss: 0.3102 - val_accuracy: 0.8773 - val_loss:
0.3233
Epoch 57/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8713 - loss: 0.3168 - val_accuracy: 0.8787 - val_loss:
0.3115
Epoch 58/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 898us/step - accuracy: 0.8606 - loss: 0.3366 - val_accuracy: 0.8827 - val_loss:
0.3072
Epoch 59/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 538us/step - accuracy: 0.8661 - loss: 0.3152 - val_accuracy: 0.8787 - val_loss:
0.3102
Epoch 60/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 590us/step - accuracy: 0.8624 - loss: 0.3256 - val_accuracy: 0.8773 - val_loss:
0.3109
Epoch 61/100
```

```
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 682us/step - accuracy: 0.8604 - loss: 0.3277 - val_accuracy: 0.8787 - val_loss:
0.3086
Epoch 62/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 581us/step - accuracy: 0.8664 - loss: 0.3198 - val_accuracy: 0.8813 - val_loss:
0.3075
Epoch 63/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 769us/step - accuracy: 0.8705 - loss: 0.3198 - val_accuracy: 0.8853 - val_loss:
0.3116
Epoch 64/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 518us/step - accuracy: 0.8730 - loss: 0.3167 - val_accuracy: 0.8800 - val_loss:
0.3079
Epoch 65/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 680us/step - accuracy: 0.8684 - loss: 0.3170 - val_accuracy: 0.8800 - val_loss:
0.3122
Epoch 66/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 663us/step - accuracy: 0.8728 - loss: 0.3117 - val_accuracy: 0.8880 - val_loss:
0.3069
Epoch 67/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 651us/step - accuracy: 0.8672 - loss: 0.3183 - val_accuracy: 0.8720 - val_loss:
0.3091
Epoch 68/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 549us/step - accuracy: 0.8661 - loss: 0.3215 - val_accuracy: 0.8800 - val_loss:
0.3077
Epoch 69/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 552us/step - accuracy: 0.8664 - loss: 0.3156 - val_accuracy: 0.8747 - val_loss:
0.3122
Epoch 70/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 608us/step - accuracy: 0.8681 - loss: 0.3206 - val_accuracy: 0.8800 - val_loss:
0.3072
Epoch 71/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 602us/step - accuracy: 0.8701 - loss: 0.3229 - val_accuracy: 0.8800 - val_loss:
0.3109
Epoch 72/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8672 - loss: 0.3198 - val_accuracy: 0.8760 - val_loss:
0.3118
Epoch 73/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - accuracy: 0.8717 - loss: 0.3115 - val_accuracy: 0.8800 - val_loss:
0.3092
Epoch 74/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 878us/step - accuracy: 0.8609 - loss: 0.3184 - val_accuracy: 0.8733 - val_loss:
0.3145
Epoch 75/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - accuracy: 0.8674 - loss: 0.3228 - val_accuracy: 0.8787 - val_loss:
0.3089
Epoch 76/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 538us/step - accuracy: 0.8646 - loss: 0.3247 - val_accuracy: 0.8840 - val_loss:
0.3080
Epoch 77/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 569us/step - accuracy: 0.8612 - loss: 0.3289 - val_accuracy: 0.8733 - val_loss:
0.3120
Epoch 78/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 921us/step - accuracy: 0.8734 - loss: 0.3083 - val_accuracy: 0.8773 - val_loss:
0.3113
Epoch 79/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 564us/step - accuracy: 0.8722 - loss: 0.3060 - val_accuracy: 0.8733 - val_loss:
0.3116
Epoch 80/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 619us/step - accuracy: 0.8708 - loss: 0.3155 - val_accuracy: 0.8800 - val_loss:
0.3125
Epoch 81/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 648us/step - accuracy: 0.8741 - loss: 0.3080 - val_accuracy: 0.8800 - val_loss:
0.3066
Epoch 82/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 562us/step - accuracy: 0.8628 - loss: 0.3183 - val_accuracy: 0.8707 - val_loss:
0.3124
Epoch 83/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 577us/step - accuracy: 0.8714 - loss: 0.3088 - val_accuracy: 0.8813 - val_loss:
0.3067
Epoch 84/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 977us/step - accuracy: 0.8753 - loss: 0.3072 - val_accuracy: 0.8853 - val_loss:
0.3062
Epoch 85/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 508us/step - accuracy: 0.8665 - loss: 0.3210 - val_accuracy: 0.8813 - val_loss:
0.3104
Epoch 86/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 548us/step - accuracy: 0.8769 - loss: 0.3047 - val_accuracy: 0.8773 - val_loss:
0.3091
Epoch 87/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 531us/step - accuracy: 0.8688 - loss: 0.3157 - val_accuracy: 0.8747 - val_loss:
0.3121
Epoch 88/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 870us/step - accuracy: 0.8727 - loss: 0.3030 - val_accuracy: 0.8667 - val_loss:
0.3203
Epoch 89/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 858us/step - accuracy: 0.8583 - loss: 0.3241 - val_accuracy: 0.8827 - val_loss:
0.3099
Epoch 90/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 736us/step - accuracy: 0.8697 - loss: 0.3122 - val_accuracy: 0.8840 - val_loss:
0.3046
Epoch 91/100
```

```
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 610us/step - accuracy: 0.8764 - loss: 0.2977 - val_accuracy: 0.8800 - val_loss:
0.3126
Epoch 92/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 716us/step - accuracy: 0.8776 - loss: 0.2998 - val_accuracy: 0.8813 - val_loss:
0.3057
Epoch 93/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 597us/step - accuracy: 0.8760 - loss: 0.3002 - val_accuracy: 0.8680 - val_loss:
0.3157
Epoch 94/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 540us/step - accuracy: 0.8829 - loss: 0.2984 - val_accuracy: 0.8800 - val_loss:
0.3070
Epoch 95/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 507us/step - accuracy: 0.8789 - loss: 0.2993 - val_accuracy: 0.8773 - val_loss:
0.3125
Epoch 96/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 720us/step - accuracy: 0.8761 - loss: 0.3058 - val_accuracy: 0.8813 - val_loss:
0.3109
Epoch 97/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 931us/step - accuracy: 0.8714 - loss: 0.3079 - val_accuracy: 0.8800 - val_loss:
0.3108
Epoch 98/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 578us/step - accuracy: 0.8724 - loss: 0.3036 - val_accuracy: 0.8787 - val_loss:
0.3138
Epoch 99/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 614us/step - accuracy: 0.8784 - loss: 0.2993 - val_accuracy: 0.8720 - val_loss:
0.3150
Epoch 100/100
106/106 ━━━━━━━━━━━━━━━━━━━━ 0s 580us/step - accuracy: 0.8632 - loss: 0.3138 - val_accuracy: 0.8747 - val_loss:
0.3126
```

Out[18]:
```
<keras.src.callbacks.history.History at 0x79df07062230>
```

In [23]:
```python
pred = model.predict(x_test)
```

```
79/79 ━━━━━━━━━━━━━━━━━━━━ 0s 518us/step
```

In [25]:
```python
y_pred = []
for val in pred:
    if val > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)

from sklearn.metrics import accuracy_score,confusion_matrix,ConfusionMatrixDisplay
accuracy_score(y_test,y_pred)
```

Out[25]:
```
0.8624
```

In [27]:
```python
cm = confusion_matrix(y_test,y_pred)
display = ConfusionMatrixDisplay(cm)
display.plot()
```

Out[27]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x79df08535a50>
```