In [1]:
```
##### ANN model

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

In [2]:
```
### import data
data = pd.read_csv(r"C:\Users\shubham lokare\Downloads\Churn_Modelling.csv")
```

In [3]:
```
data
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard
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
...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1

10000 rows × 14 columns

In [4]:
```
data.head()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
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 [5]:
```
#### split the data

x =pd.DataFrame(data.iloc[:,3:13])
```

In [6]:
```
x
```

Out[6]:

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

10000 rows × 10 columns

In [7]:
```
y = pd.DataFrame(data.iloc[:,13])
```

In [8]:
```
y
```

Out[8]:

	Exited
0	1
1	0
2	1
3	0
4	0
...	...
9995	0
9996	0
9997	1
9998	1
9999	0

10000 rows × 1 columns

In [9]:
```
### the we do pre proprocessing and preprocessing only on input data
### gender and Geography have a categorical data then we need to convert into the numerical

#Create dummy variables
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder()
geography=pd.get_dummies(x["Geography"],drop_first=True)
gender=pd.get_dummies(x['Gender'],drop_first=True)
```

In [10]:
```
# Perform OneHotEncoding on both 'Geography' and 'Gender'
new = enc.fit_transform(x[['Geography', 'Gender']]).toarray()  # Convert to dense array

# Convert the array into DataFrame with proper column names
new_df = pd.DataFrame(new, columns=enc.get_feature_names_out(['Geography', 'Gender']))

# Concatenate the original DataFrame with the new one-hot encoded columns
X = pd.concat([x, new_df], axis=1)
```

In [11]:
```
##### the add into the data framedata = pd.concat([x,Geography ,Gender] ,axis =1)
## Concatenate the Data Frames

X
```

Out[11]:

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

10000 rows × 15 columns

In [12]:
```
#### drop

X = X.drop(['Geography' ,'Gender'], axis =1)
```

In [13]:
```
X
```

Out[13]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_France	Ge
0	619	42	2	0.00	1	1	1	101348.88	1.0	
1	608	41	1	83807.86	1	0	1	112542.58	0.0	
2	502	42	8	159660.80	3	1	0	113931.57	1.0	
3	699	39	1	0.00	2	0	0	93826.63	1.0	
4	850	43	2	125510.82	1	1	1	79084.10	0.0	
...	...	...	...	...	...	...	...	...	...	
9995	771	39	5	0.00	2	1	0	96270.64	1.0	
9996	516	35	10	57369.61	1	1	1	101699.77	1.0	
9997	709	36	7	0.00	1	0	1	42085.58	1.0	
9998	772	42	3	75075.31	2	1	0	92888.52	0.0	
9999	792	28	4	130142.79	1	1	0	38190.78	1.0	

10000 rows × 13 columns

In [14]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

In [15]:

```
X_train
```

Out[15]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_France	Ge
7389	667	34	5	0.00	2	1	0	163830.64	0.0	
9275	427	42	1	75681.52	1	1	1	57098.00	0.0	
2995	535	29	2	112367.34	1	1	0	185630.76	1.0	
5316	654	40	5	105683.63	1	1	0	173617.09	0.0	
356	850	57	8	126776.30	2	1	1	132298.49	0.0	
...	...	...	...	...	...	...	...	...	...	
9225	594	32	4	120074.97	2	1	1	162961.79	0.0	
4859	794	22	4	114440.24	1	1	1	107753.07	0.0	
3264	738	35	5	161274.05	2	1	0	181429.87	1.0	
9845	590	38	9	0.00	2	1	1	148750.16	0.0	
2732	623	48	1	108076.33	1	1	0	118855.26	0.0	

8000 rows × 13 columns

In [16]:

```
X_test
```

Out[16]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_France	Ge
9394	597	35	8	131101.04	1	1	1	192852.67	0.0	
898	523	40	2	102967.41	1	1	0	128702.10	1.0	
2398	706	42	8	95386.82	1	1	1	75732.25	0.0	
5906	788	32	4	112079.58	1	0	0	89368.59	1.0	
2343	706	38	5	163034.82	2	1	1	135662.17	0.0	
...	...	...	...	...	...	...	...	...	...	
1037	625	24	1	0.00	2	1	1	180969.55	1.0	
2899	586	35	7	0.00	2	1	0	70760.69	1.0	
9549	578	36	1	157267.95	2	1	0	141533.19	0.0	
2740	650	34	4	142393.11	1	1	1	11276.48	0.0	
6690	573	30	8	127406.50	1	1	0	192950.60	0.0	

2000 rows × 13 columns

In [17]:

```
#### 1st we apply scale
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [18]:
```
X_train= sc.fit_transform(X_train)
X_test =sc.fit_transform(X_test)
```

In [19]:
```
X_test
```

Out[19]:
```
array([[-0.56129438, -0.39401698,  0.9869706 , ..., -0.57427105,
         1.11339196, -1.11339196],
       [-1.33847768,  0.07611425, -1.08432132, ..., -0.57427105,
         1.11339196, -1.11339196],
       [ 0.58347561,  0.26416674,  0.9869706 , ...,  1.74133801,
         1.11339196, -1.11339196],
       ...,
       [-0.76084144, -0.29999074, -1.42953664, ...,  1.74133801,
        -0.8981563 ,  0.8981563 ],
       [-0.0046631 , -0.48804323, -0.39389068, ..., -0.57427105,
        -0.8981563 ,  0.8981563 ],
       [-0.81335383, -0.86414821,  0.9869706 , ..., -0.57427105,
        -0.8981563 ,  0.8981563 ]])
```

In [20]:
```
X_train
```

Out[20]:
```
array([[ 0.16958176, -0.46460796,  0.00666099, ...,  1.74309049,
         1.09168714, -1.09168714],
       [-2.30455945,  0.30102557, -1.37744033, ..., -0.57369368,
        -0.91601335,  0.91601335],
       [-1.19119591, -0.94312892, -1.031415  , ..., -0.57369368,
         1.09168714, -1.09168714],
       ...,
       [ 0.9015152 , -0.36890377,  0.00666099, ..., -0.57369368,
        -0.91601335,  0.91601335],
       [-0.62420521, -0.08179119,  1.39076231, ...,  1.74309049,
         1.09168714, -1.09168714],
       [-0.28401079,  0.87525072, -1.37744033, ..., -0.57369368,
         1.09168714, -1.09168714]])
```

In [21]:
```
##### apply ann model
import keras
```

In [24]:
```
from keras.models import Sequential
from keras.layers import Dropout
from keras.layers import LeakyReLU ,ReLU
from keras.layers import Dense
```

In [45]:
```
# Initialize the classifier
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units=6, kernel_initializer='he_uniform', activation='relu', input_dim=13))  # Updated inpu

# Adding the second hidden layer
classifier.add(Dense(units=6, kernel_initializer='he_uniform', activation='relu'))

###  update the ouput laye
classifier.add(Dense(units =1 ,kernel_initializer='glorot_uniform' ,activation='sigmoid'))
# Compile the model
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Fit the model
model_history = classifier.fit(X_train, y_train, validation_split=0.2, batch_size=10, epochs=100)
```

```
Epoch 1/100
C:\anaconda\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`inpu
t_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
640/640 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.7325 - loss: 0.6204 - val_accuracy: 0.8106 - val_loss: 0.
4630
Epoch 2/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8222 - loss: 0.4264 - val_accuracy: 0.8138 - val_loss: 0.
4425
Epoch 3/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8175 - loss: 0.4310 - val_accuracy: 0.8169 - val_loss: 0.
4331
Epoch 4/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8303 - loss: 0.4023 - val_accuracy: 0.8231 - val_loss: 0.
4266
Epoch 5/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8245 - loss: 0.4141 - val_accuracy: 0.8250 - val_loss: 0.
4212
Epoch 6/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8165 - loss: 0.4237 - val_accuracy: 0.8231 - val_loss: 0.
4169
```

```
Epoch 7/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8232 - loss: 0.4018 - val_accuracy: 0.8219 - val_loss: 0.
4112
Epoch 8/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8087 - loss: 0.4148 - val_accuracy: 0.8256 - val_loss: 0.
4084
Epoch 9/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8245 - loss: 0.3820 - val_accuracy: 0.8219 - val_loss: 0.
4048
Epoch 10/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8202 - loss: 0.3913 - val_accuracy: 0.8213 - val_loss: 0.
4026
Epoch 11/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8249 - loss: 0.3774 - val_accuracy: 0.8256 - val_loss: 0.
3997
Epoch 12/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8312 - loss: 0.3742 - val_accuracy: 0.8213 - val_loss: 0.
3984
Epoch 13/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8229 - loss: 0.3867 - val_accuracy: 0.8231 - val_loss: 0.
3969
Epoch 14/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8276 - loss: 0.3792 - val_accuracy: 0.8219 - val_loss: 0.
3948
Epoch 15/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8294 - loss: 0.3638 - val_accuracy: 0.8231 - val_loss: 0.
3930
Epoch 16/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8342 - loss: 0.3637 - val_accuracy: 0.8194 - val_loss: 0.
3904
Epoch 17/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8273 - loss: 0.3697 - val_accuracy: 0.8244 - val_loss: 0.
3876
Epoch 18/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8312 - loss: 0.3640 - val_accuracy: 0.8281 - val_loss: 0.
3853
Epoch 19/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8178 - loss: 0.3707 - val_accuracy: 0.8250 - val_loss: 0.
3818
Epoch 20/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8340 - loss: 0.3626 - val_accuracy: 0.8475 - val_loss: 0.
3814
Epoch 21/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8528 - loss: 0.3568 - val_accuracy: 0.8475 - val_loss: 0.
3778
Epoch 22/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8522 - loss: 0.3516 - val_accuracy: 0.8481 - val_loss: 0.
3764
Epoch 23/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8528 - loss: 0.3535 - val_accuracy: 0.8519 - val_loss: 0.
3733
Epoch 24/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8420 - loss: 0.3779 - val_accuracy: 0.8512 - val_loss: 0.
3731
Epoch 25/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8548 - loss: 0.3550 - val_accuracy: 0.8537 - val_loss: 0.
3750
Epoch 26/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8508 - loss: 0.3638 - val_accuracy: 0.8506 - val_loss: 0.
3696
Epoch 27/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8572 - loss: 0.3552 - val_accuracy: 0.8512 - val_loss: 0.
3715
Epoch 28/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8629 - loss: 0.3438 - val_accuracy: 0.8531 - val_loss: 0.
3693
Epoch 29/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8569 - loss: 0.3557 - val_accuracy: 0.8500 - val_loss: 0.
3727
Epoch 30/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8545 - loss: 0.3509 - val_accuracy: 0.8519 - val_loss: 0.
3684
Epoch 31/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8609 - loss: 0.3419 - val_accuracy: 0.8512 - val_loss: 0.
3678
Epoch 32/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - accuracy: 0.8652 - loss: 0.3376 - val_accuracy: 0.8575 - val_loss: 0.
3658
Epoch 33/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8631 - loss: 0.3430 - val_accuracy: 0.8537 - val_loss: 0.
3638
Epoch 34/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8553 - loss: 0.3556 - val_accuracy: 0.8550 - val_loss: 0.
```

```
3641
Epoch 35/100
640/640 4s 6ms/step - accuracy: 0.8611 - loss: 0.3595 - val_accuracy: 0.8512 - val_loss: 0.
3660
Epoch 36/100
640/640 4s 6ms/step - accuracy: 0.8569 - loss: 0.3526 - val_accuracy: 0.8537 - val_loss: 0.
3657
Epoch 37/100
640/640 3s 4ms/step - accuracy: 0.8679 - loss: 0.3311 - val_accuracy: 0.8512 - val_loss: 0.
3628
Epoch 38/100
640/640 3s 5ms/step - accuracy: 0.8551 - loss: 0.3476 - val_accuracy: 0.8525 - val_loss: 0.
3629
Epoch 39/100
640/640 3s 4ms/step - accuracy: 0.8625 - loss: 0.3430 - val_accuracy: 0.8512 - val_loss: 0.
3626
Epoch 40/100
640/640 2s 3ms/step - accuracy: 0.8668 - loss: 0.3305 - val_accuracy: 0.8519 - val_loss: 0.
3638
Epoch 41/100
640/640 2s 3ms/step - accuracy: 0.8643 - loss: 0.3342 - val_accuracy: 0.8475 - val_loss: 0.
3622
Epoch 42/100
640/640 4s 5ms/step - accuracy: 0.8627 - loss: 0.3455 - val_accuracy: 0.8531 - val_loss: 0.
3617
Epoch 43/100
640/640 2s 4ms/step - accuracy: 0.8581 - loss: 0.3454 - val_accuracy: 0.8506 - val_loss: 0.
3618
Epoch 44/100
640/640 4s 6ms/step - accuracy: 0.8605 - loss: 0.3383 - val_accuracy: 0.8525 - val_loss: 0.
3619
Epoch 45/100
640/640 4s 6ms/step - accuracy: 0.8696 - loss: 0.3246 - val_accuracy: 0.8475 - val_loss: 0.
3625
Epoch 46/100
640/640 4s 6ms/step - accuracy: 0.8590 - loss: 0.3415 - val_accuracy: 0.8506 - val_loss: 0.
3605
Epoch 47/100
640/640 3s 5ms/step - accuracy: 0.8621 - loss: 0.3333 - val_accuracy: 0.8500 - val_loss: 0.
3607
Epoch 48/100
640/640 4s 6ms/step - accuracy: 0.8643 - loss: 0.3428 - val_accuracy: 0.8494 - val_loss: 0.
3596
Epoch 49/100
640/640 4s 6ms/step - accuracy: 0.8580 - loss: 0.3407 - val_accuracy: 0.8512 - val_loss: 0.
3604
Epoch 50/100
640/640 4s 6ms/step - accuracy: 0.8671 - loss: 0.3342 - val_accuracy: 0.8537 - val_loss: 0.
3604
Epoch 51/100
640/640 4s 6ms/step - accuracy: 0.8603 - loss: 0.3432 - val_accuracy: 0.8506 - val_loss: 0.
3620
Epoch 52/100
640/640 3s 5ms/step - accuracy: 0.8658 - loss: 0.3307 - val_accuracy: 0.8525 - val_loss: 0.
3594
Epoch 53/100
640/640 3s 5ms/step - accuracy: 0.8530 - loss: 0.3511 - val_accuracy: 0.8531 - val_loss: 0.
3590
Epoch 54/100
640/640 3s 5ms/step - accuracy: 0.8684 - loss: 0.3300 - val_accuracy: 0.8494 - val_loss: 0.
3622
Epoch 55/100
640/640 3s 5ms/step - accuracy: 0.8619 - loss: 0.3396 - val_accuracy: 0.8525 - val_loss: 0.
3600
Epoch 56/100
640/640 3s 5ms/step - accuracy: 0.8610 - loss: 0.3380 - val_accuracy: 0.8500 - val_loss: 0.
3584
Epoch 57/100
640/640 3s 5ms/step - accuracy: 0.8689 - loss: 0.3299 - val_accuracy: 0.8544 - val_loss: 0.
3586
Epoch 58/100
640/640 4s 5ms/step - accuracy: 0.8621 - loss: 0.3309 - val_accuracy: 0.8500 - val_loss: 0.
3587
Epoch 59/100
640/640 3s 5ms/step - accuracy: 0.8649 - loss: 0.3323 - val_accuracy: 0.8531 - val_loss: 0.
3592
Epoch 60/100
640/640 4s 5ms/step - accuracy: 0.8707 - loss: 0.3216 - val_accuracy: 0.8506 - val_loss: 0.
3626
Epoch 61/100
640/640 4s 5ms/step - accuracy: 0.8625 - loss: 0.3371 - val_accuracy: 0.8500 - val_loss: 0.
3596
Epoch 62/100
```

```
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8621 - loss: 0.3369 - val_accuracy: 0.8531 - val_loss: 0.
3602
Epoch 63/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8643 - loss: 0.3384 - val_accuracy: 0.8525 - val_loss: 0.
3594
Epoch 64/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8624 - loss: 0.3296 - val_accuracy: 0.8544 - val_loss: 0.
3571
Epoch 65/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8692 - loss: 0.3251 - val_accuracy: 0.8519 - val_loss: 0.
3579
Epoch 66/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8680 - loss: 0.3324 - val_accuracy: 0.8531 - val_loss: 0.
3589
Epoch 67/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8596 - loss: 0.3369 - val_accuracy: 0.8481 - val_loss: 0.
3598
Epoch 68/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8678 - loss: 0.3260 - val_accuracy: 0.8537 - val_loss: 0.
3594
Epoch 69/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8664 - loss: 0.3284 - val_accuracy: 0.8556 - val_loss: 0.
3603
Epoch 70/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8727 - loss: 0.3294 - val_accuracy: 0.8525 - val_loss: 0.
3574
Epoch 71/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8660 - loss: 0.3317 - val_accuracy: 0.8512 - val_loss: 0.
3587
Epoch 72/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8604 - loss: 0.3424 - val_accuracy: 0.8556 - val_loss: 0.
3574
Epoch 73/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8749 - loss: 0.3147 - val_accuracy: 0.8556 - val_loss: 0.
3580
Epoch 74/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8653 - loss: 0.3283 - val_accuracy: 0.8562 - val_loss: 0.
3595
Epoch 75/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8658 - loss: 0.3254 - val_accuracy: 0.8525 - val_loss: 0.
3600
Epoch 76/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8572 - loss: 0.3478 - val_accuracy: 0.8525 - val_loss: 0.
3585
Epoch 77/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8543 - loss: 0.3477 - val_accuracy: 0.8544 - val_loss: 0.
3588
Epoch 78/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8642 - loss: 0.3309 - val_accuracy: 0.8512 - val_loss: 0.
3595
Epoch 79/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8671 - loss: 0.3336 - val_accuracy: 0.8556 - val_loss: 0.
3582
Epoch 80/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8618 - loss: 0.3354 - val_accuracy: 0.8575 - val_loss: 0.
3566
Epoch 81/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.8630 - loss: 0.3391 - val_accuracy: 0.8544 - val_loss: 0.
3575
Epoch 82/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8574 - loss: 0.3460 - val_accuracy: 0.8519 - val_loss: 0.
3569
Epoch 83/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8632 - loss: 0.3337 - val_accuracy: 0.8556 - val_loss: 0.
3566
Epoch 84/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.8615 - loss: 0.3324 - val_accuracy: 0.8525 - val_loss: 0.
3580
Epoch 85/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8690 - loss: 0.3248 - val_accuracy: 0.8544 - val_loss: 0.
3572
Epoch 86/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8625 - loss: 0.3296 - val_accuracy: 0.8512 - val_loss: 0.
3571
Epoch 87/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8664 - loss: 0.3220 - val_accuracy: 0.8569 - val_loss: 0.
3589
Epoch 88/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8605 - loss: 0.3422 - val_accuracy: 0.8525 - val_loss: 0.
3599
Epoch 89/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8602 - loss: 0.3387 - val_accuracy: 0.8531 - val_loss: 0.
3578
```

```
Epoch 90/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8641 - loss: 0.3357 - val_accuracy: 0.8544 - val_loss: 0.
3582
Epoch 91/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8656 - loss: 0.3224 - val_accuracy: 0.8537 - val_loss: 0.
3567
Epoch 92/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8684 - loss: 0.3273 - val_accuracy: 0.8544 - val_loss: 0.
3576
Epoch 93/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8655 - loss: 0.3332 - val_accuracy: 0.8550 - val_loss: 0.
3573
Epoch 94/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8633 - loss: 0.3246 - val_accuracy: 0.8550 - val_loss: 0.
3591
Epoch 95/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8636 - loss: 0.3229 - val_accuracy: 0.8556 - val_loss: 0.
3569
Epoch 96/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8709 - loss: 0.3213 - val_accuracy: 0.8525 - val_loss: 0.
3584
Epoch 97/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8593 - loss: 0.3351 - val_accuracy: 0.8537 - val_loss: 0.
3568
Epoch 98/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8679 - loss: 0.3347 - val_accuracy: 0.8550 - val_loss: 0.
3576
Epoch 99/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8624 - loss: 0.3318 - val_accuracy: 0.8550 - val_loss: 0.
3571
Epoch 100/100
640/640 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.8688 - loss: 0.3215 - val_accuracy: 0.8512 - val_loss: 0.
3591
```

In [46]:
```
y_pred = classifier.predict(X_test)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

In [49]:
```
y_pred
y_pred = (y_pred > 0.5)
```

In [50]:
```
#### then we find accuracy of model

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Calculate the Accuracy
from sklearn.metrics import accuracy_score
score=accuracy_score(y_pred,y_test)
```

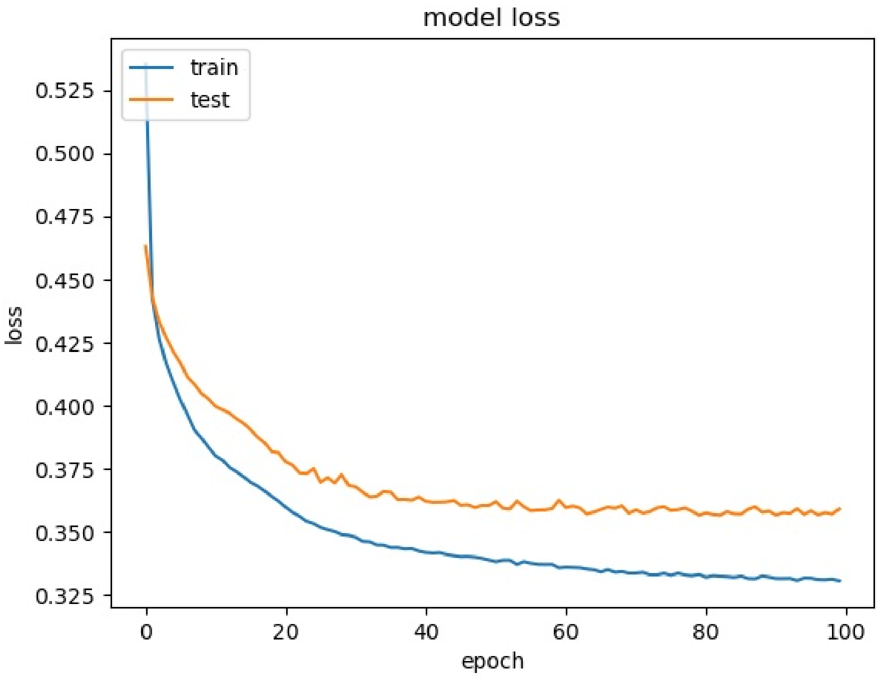
In [51]:
```
cm
```

Out[51]:
```
array([[1496,   99],
       [ 190,  215]], dtype=int64)
```

In [52]:
```
score
```

Out[52]:
```
0.8555
```

In [53]:
```
# summarize history for loss
plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js