```
#Dataset for bank·customer·churn·prediction : https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/JyothiAB/Deep-learning/1_Bank-Customer-Churn-Prediction/Churn_Modelling.
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
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	

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

```
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	E
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	

```
df.dtypes
```

```
CreditScore          int64
Geography           object
```

```
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

```
df.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	Ha
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	100
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	
std	96.653299	10.487806	2.892174	62397.405202	0.581654	
min	350.000000	18.000000	0.000000	0.000000	1.000000	
25%	584.000000	32.000000	3.000000	0.000000	1.000000	
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	

```
df.describe(include='object')
```

	Geography	Gender
top	France	Male

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df[['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary']] = scaler.fit_transform(df[['CreditScore','Age
```

```
df.describe()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	Has
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	1000(
mean	0.601058	0.282727	0.501280	0.304848	0.176733	(
std	0.193307	0.141727	0.289217	0.248696	0.193885	(
min	0.000000	0.000000	0.000000	0.000000	0.000000	(
25%	0.468000	0.189189	0.300000	0.000000	0.000000	(
50%	0.604000	0.256757	0.500000	0.387402	0.000000	1
75%	0.736000	0.351351	0.700000	0.508749	0.333333	1

```
df['Gender'].replace({'Male':0,'Female':1}, inplace=True)
```

```
df['Gender'].describe()
```

```
count    10000.000000
mean         0.454300
std          0.497932
min          0.000000
25%          0.000000
50%          0.000000
75%          1.000000
max          1.000000
Name: Gender, dtype: float64
```

```
df1 = pd.get_dummies(data = df, columns=['Geography'])
```

```
df1.describe()
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfP
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	1000(
mean	0.601058	0.454300	0.282727	0.501280	0.304848	(
std	0.193307	0.497932	0.141727	0.289217	0.248696	(
min	0.000000	0.000000	0.000000	0.000000	0.000000	(
25%	0.468000	0.000000	0.189189	0.300000	0.000000	(
50%	0.604000	0.000000	0.256757	0.500000	0.387402	(
75%	0.736000	1.000000	0.351351	0.700000	0.508749	(
max	1.000000	1.000000	1.000000	1.000000	1.000000	

```
X = df1.drop(columns = ['Exited'])
y = df1['Exited']
```

```
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=5)
```

```
len(x_train.columns)
```

```
12
```

```
import tensorflow as tf
from tensorflow import keras

model = keras.Sequential([
                          keras.layers.Dense(12, input_shape = (12,), activation='relu'),
                          keras.layers.Dense(12, input_shape = (6,), activation='relu'),
                          keras.layers.Dense(1, activation='sigmoid')
])

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

model.fit(x_train,y_train,epochs = 200)
```

```
Epoch 153/200
250/250 [==============================] - 1s 4ms/step - loss: 0.3226 - accuracy: 0.8664
Epoch 154/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3230 - accuracy: 0.8658
Epoch 155/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3235 - accuracy: 0.8659
Epoch 156/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3226 - accuracy: 0.8683
Epoch 157/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3230 - accuracy: 0.8680
Epoch 158/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3227 - accuracy: 0.8683
Epoch 159/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3231 - accuracy: 0.8661
Epoch 160/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3222 - accuracy: 0.8656
Epoch 161/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3216 - accuracy: 0.8674
Epoch 162/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3227 - accuracy: 0.8668
Epoch 163/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3225 - accuracy: 0.8694
Epoch 164/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3230 - accuracy: 0.8690
Epoch 165/200
250/250 [==============================] - 1s 4ms/step - loss: 0.3221 - accuracy: 0.8679
Epoch 166/200
```

```
Epoch 166/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3227 - accuracy: 0.8661
Epoch 167/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3218 - accuracy: 0.8668
Epoch 168/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3222 - accuracy: 0.8676
Epoch 169/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3226 - accuracy: 0.8664
Epoch 170/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3216 - accuracy: 0.8658
Epoch 171/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3226 - accuracy: 0.8650
Epoch 172/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3216 - accuracy: 0.8660
Epoch 173/200
250/250 [==============================] - 1s 4ms/step - loss: 0.3217 - accuracy: 0.8681
Epoch 174/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3223 - accuracy: 0.8646
Epoch 175/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3215 - accuracy: 0.8676
Epoch 176/200
250/250 [==============================] - 1s 4ms/step - loss: 0.3215 - accuracy: 0.8685
Epoch 177/200
250/250 [==============================] - 1s 4ms/step - loss: 0.3216 - accuracy: 0.8660
Epoch 178/200
250/250 [==============================] - 1s 4ms/step - loss: 0.3214 - accuracy: 0.8686
Epoch 179/200
250/250 [==============================] - 1s 4ms/step - loss: 0.3221 - accuracy: 0.8661
Epoch 180/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3217 - accuracy: 0.8661
Epoch 181/200
250/250 [==============================] - 1s 3ms/step - loss: 0.3215 - accuracy: 0.8684
```

```
model.compile(optimizer='SGD',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])

model.fit(x_train,y_train,epochs = 100)
```

```
Epoch 64/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3199 - accuracy: 0.8671

Epoch 65/100
```

```
250/250 [==============================] - 1s 3ms/step - loss: 0.3205 - accuracy: 0.8670
Epoch 66/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3204 - accuracy: 0.8690
Epoch 67/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3191 - accuracy: 0.8686
Epoch 68/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3195 - accuracy: 0.8680
Epoch 69/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3198 - accuracy: 0.8683
Epoch 70/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3201 - accuracy: 0.8674
Epoch 71/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3204 - accuracy: 0.8677
Epoch 72/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3198 - accuracy: 0.8670
Epoch 73/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3194 - accuracy: 0.8681
Epoch 74/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3194 - accuracy: 0.8689
Epoch 75/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3199 - accuracy: 0.8656
Epoch 76/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3198 - accuracy: 0.8691
Epoch 77/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3190 - accuracy: 0.8675
Epoch 78/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3194 - accuracy: 0.8668
Epoch 79/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3192 - accuracy: 0.8674
Epoch 80/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3196 - accuracy: 0.8669
Epoch 81/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3196 - accuracy: 0.8675
Epoch 82/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3194 - accuracy: 0.8691
Epoch 83/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3203 - accuracy: 0.8668
Epoch 84/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3199 - accuracy: 0.8677
Epoch 85/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3199 - accuracy: 0.8681

Epoch 86/100
```

```
250/250 [==============================] - 1s 3ms/step - loss: 0.3193 - accuracy: 0.8684
Epoch 87/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3199 - accuracy: 0.8680
Epoch 88/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3198 - accuracy: 0.8670
Epoch 89/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3197 - accuracy: 0.8680
Epoch 90/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3200 - accuracy: 0.8689
Epoch 91/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3198 - accuracy: 0.8679
Epoch 92/100
250/250 [==============================] - 1s 3ms/step - loss: 0.3194 - accuracy: 0.8675
```

```
model.evaluate(x_test,y_test)
```

```
63/63 [==============================] - 1s 5ms/step - loss: 0.3446 - accuracy: 0.8600
[0.3445870280265808, 0.8600000143051147]
```

```
y_pred = model.predict(x_test)
y_pred
```

```
array([[0.05764392],
       [0.06791589],
       [0.0512576 ],
       ...,
       [0.02203725],
       [0.03229055],
       [0.06185446]], dtype=float32)
```

```
y_predict = []
for i in y_pred:
  if i < 0.5:
    y_predict.append(0)
  else:
    y_predict.append(1)
```

```
from sklearn.metrics import confusion_matrix , classification_report
```

```
print(classification_report(y_test,y_predict))
```

```
                  precision    recall  f1-score   support

               0       0.88      0.96      0.92      1595
               1       0.75      0.46      0.57       405

        accuracy                           0.86      2000
       macro avg       0.81      0.71      0.74      2000
    weighted avg       0.85      0.86      0.85      2000
```

```
print(confusion_matrix(y_test,y_predict))
```

```
    [[1533   62]
     [ 218  187]]
```

✓ 0s completed at 4:12 PM