Implement a Simple Neural Network Model and predict if a person will buy insurance or not for the given dataset.

In [85]:

```
import pandas as pd
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
```

In [86]:

```
data = pd.read_csv("insurance_data.csv")
data.head()
```

Out[86]:

	age	Affordability	bought_insurance
0	22	1	0
1	25	1	0
2	47	0	1
3	52	1	0
4	46	1	1

In [87]:

```
x = data.iloc[:, :-1]
y = data.iloc[:, -1]
print(x.shape)
```

```
(27, 2)
```

In [88]:

```
train_x, test_x, train_y, test_y = train_test_split(x, y, train_size=0.8, shuffle=True)
```

In [89]:

```
scaling = StandardScaler()
```

In [90]:

```
train_x = scaling.fit_transform(train_x)
test_x = scaling.fit_transform(test_x)
```

In [91]:

```
model = Sequential()
model.add(Dense(2,input_dim=2, activation='relu'))
model.add(Dense(2,input_dim=2, activation='relu'))
model.add(Dense(1,activation='softmax'))
```

In [92]:

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

In [93]:

```
model.fit(train_x, train_y, epochs=100, batch_size=40)
model.evaluate(test_x, test_y)
predict_y = model.predict(test_x)
actual_result = test_y.tolist()
predicted_result = predict_y.tolist()
```

```
Epoch 79/100
1/1 [==============================] - 0s 5ms/step - loss: 0.7161 - accura
cy: 0.4762
Epoch 80/100
1/1 [==============================] - 0s 4ms/step - loss: 0.7156 - accura
cy: 0.4762
Epoch 81/100
1/1 [==============================] - 0s 6ms/step - loss: 0.7151 - accura
cy: 0.4762
Epoch 82/100
1/1 [==============================] - 0s 4ms/step - loss: 0.7146 - accura
cy: 0.4762
Epoch 83/100
1/1 [==============================] - 0s 4ms/step - loss: 0.7141 - accura
cy: 0.4762
Epoch 84/100
1/1 [==============================] - 0s 4ms/step - loss: 0.7136 - accura
cy: 0.4762
Epoch 85/100
1/1 [==============================] - 0s 3ms/step - loss: 0.7131 - accura
```

In [94]:

```
accuracy_score(predicted_result, actual_result)*100
```

Out[94]:

```
66.66666666666666
```

In [ ]:

```
```

Use the given dataset and build a customer churn prediction model using artificial neural network.

[Concept used: ANN Classification, Use Multiple Hidden Layers, Confusion Matrix to check the accuracy rate of the model]

In [15]:

```
import pandas as pd
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split as split_data
from sklearn.metrics import accuracy_score as accuracy
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
```

In [3]:

```
dataset = pd.read_csv("churn_modelling.csv")
dataset.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balanc
0	1	15634602	Hargrave	619	France	Female	42	2	0.0
1	2	15647311	Hill	608	Spain	Female	41	1	83807.8
2	3	15619304	Onio	502	France	Female	42	8	159660.8
3	4	15701354	Boni	699	France	Female	39	1	0.0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.8

In [4]:

```
# row number, surname, customerid are irrelevant to the model we are tying to create
dataset.drop(dataset.columns[[0,1,2]], axis=1, inplace=True)
```

In [5]:

```
dataset.head()
```

Out[5]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAc
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

In [8]:

```
def convert(vector):
    visited = []
    ind = 0
    result = []
    for i in vector:
        if i in visited:
            result.append(visited.index(i))
        else:
            visited.append(i)
            result.append(len(visited)-1)
    return result
```

In [9]:

```
dataset.iloc[:, 1] = np.array(convert(dataset.iloc[:, 1].tolist()))
dataset.iloc[:, 2] = np.array(convert(dataset.iloc[:, 2].tolist()))
```

In [10]:

```
dataset.head()
```

Out[10]:

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

In [11]:

```
x = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
x.shape, y.shape
```

Out[11]:

```
((10000, 10), (10000,))
```

In [13]:

```
train_x, test_x, train_y, test_y = split_data(x, y, train_size=0.75, shuffle=True)
```

In [16]:

```
scaling = StandardScaler()
```

In [17]:

```
train_x = scaling.fit_transform(train_x)
test_x = scaling.fit_transform(test_x)
```

In [19]:

```
model = Sequential()
model.add(Dense(10, input_dim=10, activation='relu', name='input'))
model.add(Dense(10, input_dim=10, activation='relu', name='layer1'))
model.add(Dense(5, input_dim=10, activation='relu', name='layer2'))
model.add(Dense(3, input_dim=5, activation='relu', name='layer3'))
model.add(Dense(1, activation='sigmoid', name='output'))
```

In [20]:

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (Dense)               (None, 10)                110

 layer1 (Dense)              (None, 10)                110

 layer2 (Dense)              (None, 5)                 55

 layer3 (Dense)              (None, 3)                 18

 output (Dense)              (None, 1)                 4

=================================================================
Total params: 297
Trainable params: 297
Non-trainable params: 0
_________________________________________________________________
```

In [21]:

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

In [28]:

```
def sigmoid_correction(predict):
    for i in range(len(predict)):
        predict[i] = 1 if predict[i][0] >=0.5 else 0
    return predict
```

In [32]:

```
model.fit(train_x, train_y, epochs=500, batch_size=450)
prediction = model.predict(test_x)
# model.evaluate(test_x, test_y)
actual_y = test_y.tolist()
predicted_y = sigmoid_correction(prediction.tolist())
```

```
17/17 [==============================] - 0s 2ms/step - loss: 0.2973 - accu
racy: 0.8777
Epoch 408/500
17/17 [==============================] - 0s 2ms/step - loss: 0.2969 - accu
racy: 0.8780
Epoch 409/500
17/17 [==============================] - 0s 3ms/step - loss: 0.2976 - accu
racy: 0.8763
Epoch 410/500
17/17 [==============================] - 0s 3ms/step - loss: 0.2980 - accu
racy: 0.8777
Epoch 411/500
17/17 [==============================] - 0s 3ms/step - loss: 0.2980 - accu
racy: 0.8771
Epoch 412/500
17/17 [==============================] - 0s 3ms/step - loss: 0.2972 - accu
racy: 0.8771
Epoch 413/500
```

In [33]:

```
accuracy(predicted_y, actual_y)*100
```

Out[33]:

```
84.68
```

In [ ]: