In [60]:
```
#Import the Library

import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [61]:
```
#Import the Dataset

dataset = pd.read_csv('Dataset.csv')
dataset.head(1)
```

Out[61]:

	Patient ID	Age	Blood Pressure	Cholesterol Levels	Heart Rate	BMI	Diagnosis	Treatment Plan	Recovery Status	M
0	101	65	130	250	72	28.0	Hypertension with high cholesterol.	Medication: Lisinopril (blood pressure), Stati...	Active Recovery	

In [94]:
```
dataset['Diagnosis'] = dataset['Diagnosis'].apply(
    lambda x: 1 if any(condition in x for condition in ['Hypertension', 'Obesity',
)
```

In [95]:
```
print(dataset['Diagnosis'].value_counts())
```

```
Diagnosis
0    63
1    37
Name: count, dtype: int64
```

In [96]:
```
#Encoding Categorical Data

#Level Encoding the “Gender” Column

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for column in ['Treatment Plan', 'Recovery Status', 'Medication Type', 'Follow-up R
    dataset[column] = le.fit_transform(dataset[column])
print(dataset)
```

```
    Patient ID  Age  Blood Pressure  Cholesterol Levels  Heart Rate   BMI  \
0          101   65             130                 250          72  28.0
1          102   42             110                 150          76  24.0
2          103   58             140                 200          80  30.0
3          104   71             160                 220          88  32.0
4          105   35             120                 180          74  27.0
..         ...  ...             ...                 ...         ...   ...
95         110   45             140                 260          85  32.0
96         111   70             125                 200          70  25.0
97         112   55             115                 180          80  24.0
98         113   30             120                 190          75  23.0
99         101   45             120                 240          80  28.0

    Diagnosis  Treatment Plan  Recovery Status  Medication Type  \
0           1              62                1               50
1           0              15               10               59
2           0              63                1               58
3           0              61                1               32
4           1              15               10               59
..        ...             ...              ...              ...
95          1               2               13               15
96          0              82               12                1
97          0              13               13               45
98          0              84                9               66
99          1              60               13               16

    Follow-up Requirement
0                      26
1                      28
2                      26
3                      21
4                      30
..                    ...
95                     18
96                      2
97                      0
98                     20
99                     26

[100 rows x 11 columns]
```

In [97]:
```
dataset.head(1)
```

Out[97]:

	Patient ID	Age	Blood Pressure	Cholesterol Levels	Heart Rate	BMI	Diagnosis	Treatment Plan	Recovery Status	Med
0	101	65	130	250	72	28.0	1	62	1	

In [98]:
```
meddata=dataset[['Age','Blood Pressure','Cholesterol Levels','Heart Rate','BMI','Tr
```

In [99]:
```
X = meddata.drop('Diagnosis', axis=1)
Y = meddata['Diagnosis']
```

```
print(X)
print(Y)
```

```
    Age  Blood Pressure  Cholesterol Levels  Heart Rate   BMI  Treatment Plan  \
0    65             130                 250          72  28.0              62
1    42             110                 150          76  24.0              15
2    58             140                 200          80  30.0              63
3    71             160                 220          88  32.0              61
4    35             120                 180          74  27.0              15
..  ...             ...                 ...         ...   ...             ...
95   45             140                 260          85  32.0               2
96   70             125                 200          70  25.0              82
97   55             115                 180          80  24.0              13
98   30             120                 190          75  23.0              84
99   45             120                 240          80  28.0              60

    Recovery Status  Medication Type  Follow-up Requirement
0                 1               50                     26
1                10               59                     28
2                 1               58                     26
3                 1               32                     21
4                10               59                     30
..              ...              ...                    ...
95               13               15                     18
96               12                1                      2
97               13               45                      0
98                9               66                     20
99               13               16                     26

[100 rows x 9 columns]
0     1
1     0
2     0
3     0
4     1
     ..
95    1
96    0
97    0
98    0
99    1
Name: Diagnosis, Length: 100, dtype: int64
```

In [100…

```
#Use Sklearn to split user data into a training set and a testing set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_s
```

In [101…

```
#Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [102…

```
#Initializing the ANN
ann = tf.keras.models.Sequential()
```

In [103…

```
#Adding the Input layer and first hidden layer
ann.add(tf.keras.layers.Dense(units = 6, activation = 'relu'))
```

```
#Adding the second Hidden Layer
ann.add(tf.keras.layers.Dense(units = 6, activation = 'relu'))

#Adding the Output Layer
ann.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))
```

In [104…

```
#Compiling the ANN
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'
```

In [105…

```
#Training the ANN on the Training Set
ann.fit(X_train, Y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 4s 431ms/step - accuracy: 0.4480 - loss: 0.7140
Epoch 2/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.4896 - loss: 0.7046
Epoch 3/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5000 - loss: 0.7055
Epoch 4/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4890 - loss: 0.7067
Epoch 5/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5071 - loss: 0.6967
Epoch 6/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4759 - loss: 0.6991
Epoch 7/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5501 - loss: 0.6912
Epoch 8/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5117 - loss: 0.6869
Epoch 9/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5533 - loss: 0.6891
Epoch 10/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5377 - loss: 0.6828
Epoch 11/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.5722 - loss: 0.6816
Epoch 12/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.5787 - loss: 0.6802
Epoch 13/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6054 - loss: 0.6766
Epoch 14/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.5748 - loss: 0.6829
Epoch 15/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5741 - loss: 0.6822
Epoch 16/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6047 - loss: 0.6748
Epoch 17/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6118 - loss: 0.6762
Epoch 18/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6307 - loss: 0.6703
Epoch 19/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6033 - loss: 0.6791
Epoch 20/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6385 - loss: 0.6732
Epoch 21/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6385 - loss: 0.6685
Epoch 22/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6535 - loss: 0.6696
Epoch 23/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6730 - loss: 0.6623
Epoch 24/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.5916 - loss: 0.6774
Epoch 25/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6502 - loss: 0.6675
Epoch 26/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7075 - loss: 0.6574
Epoch 27/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6535 - loss: 0.6624
Epoch 28/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6411 - loss: 0.6669
```

```
Epoch 29/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6717 - loss: 0.6590
Epoch 30/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6834 - loss: 0.6540
Epoch 31/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6365 - loss: 0.6662
Epoch 32/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6912 - loss: 0.6575
Epoch 33/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6677 - loss: 0.6652
Epoch 34/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6756 - loss: 0.6576
Epoch 35/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6756 - loss: 0.6557
Epoch 36/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6632 - loss: 0.6587
Epoch 37/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6717 - loss: 0.6551
Epoch 38/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7022 - loss: 0.6512
Epoch 39/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6983 - loss: 0.6463
Epoch 40/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6905 - loss: 0.6561
Epoch 41/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7140 - loss: 0.6407
Epoch 42/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6788 - loss: 0.6491
Epoch 43/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6554 - loss: 0.6514
Epoch 44/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6944 - loss: 0.6491
Epoch 45/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6671 - loss: 0.6521
Epoch 46/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6749 - loss: 0.6516
Epoch 47/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6866 - loss: 0.6464
Epoch 48/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6788 - loss: 0.6435
Epoch 49/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6866 - loss: 0.6355
Epoch 50/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7022 - loss: 0.6363
Epoch 51/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7022 - loss: 0.6418
Epoch 52/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7100 - loss: 0.6258
Epoch 53/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6632 - loss: 0.6493
Epoch 54/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6554 - loss: 0.6534
Epoch 55/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7022 - loss: 0.6297
Epoch 56/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6905 - loss: 0.6336
```

```
Epoch 57/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6866 - loss: 0.6364
Epoch 58/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6977 - loss: 0.6341
Epoch 59/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6820 - loss: 0.6361
Epoch 60/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6703 - loss: 0.6400
Epoch 61/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7016 - loss: 0.6345
Epoch 62/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7094 - loss: 0.6252
Epoch 63/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6977 - loss: 0.6192
Epoch 64/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6938 - loss: 0.6259
Epoch 65/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6859 - loss: 0.6291
Epoch 66/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7094 - loss: 0.6291
Epoch 67/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.6820 - loss: 0.6343
Epoch 68/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6898 - loss: 0.6248
Epoch 69/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6827 - loss: 0.6224
Epoch 70/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6710 - loss: 0.6294
Epoch 71/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7022 - loss: 0.6122
Epoch 72/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6905 - loss: 0.6197
Epoch 73/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6983 - loss: 0.6080
Epoch 74/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6898 - loss: 0.6166
Epoch 75/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7094 - loss: 0.6080
Epoch 76/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7133 - loss: 0.6062
Epoch 77/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6898 - loss: 0.6216
Epoch 78/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6508 - loss: 0.6308
Epoch 79/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6977 - loss: 0.6088
Epoch 80/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6938 - loss: 0.6160
Epoch 81/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7243 - loss: 0.6054
Epoch 82/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.6931 - loss: 0.6147
Epoch 83/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7478 - loss: 0.5921
Epoch 84/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7087 - loss: 0.6070
```

```
Epoch 85/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7165 - loss: 0.6041
Epoch 86/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6977 - loss: 0.6065
Epoch 87/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7211 - loss: 0.5888
Epoch 88/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.6938 - loss: 0.6007
Epoch 89/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7172 - loss: 0.5857
Epoch 90/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7055 - loss: 0.6014
Epoch 91/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7094 - loss: 0.5830
Epoch 92/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6938 - loss: 0.6093
Epoch 93/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6938 - loss: 0.6048
Epoch 94/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.6898 - loss: 0.6023
Epoch 95/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7094 - loss: 0.5794
Epoch 96/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7133 - loss: 0.5838
Epoch 97/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.6859 - loss: 0.5980
Epoch 98/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.7282 - loss: 0.5714
Epoch 99/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7400 - loss: 0.5745
Epoch 100/100
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.7243 - loss: 0.5763
```

Out[105…

```
<keras.src.callbacks.history.History at 0x7c6dbc50bf70>
```

In [107…

```
print(ann.predict(sc.transform([[79, 140, 250, 69,  28.0,61,1,50,26]] )) > 0.5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
[[ True]]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: UserWarning: X does not
have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

In [110…

```
#Predicting the Test Set Results

y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred), 1), Y_test.values.reshape(len(Y_t
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
[[0 1]
 [0 0]
 [1 0]
 [0 1]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 1]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [1 1]
 [0 0]
 [0 0]
 [1 0]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [0 0]
 [0 0]]
```

In [111…

```python
#Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(Y_test, y_pred)
print(cm)
accuracy_score(Y_test, y_pred)
```

```
[[17  3]
 [ 7  3]]
```

Out[111…

```
0.6666666666666666
```