```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
from tensorflow.keras import Sequential # used to build ANN
from tensorflow.keras.layers import Dense # used to add hidden layers
from sklearn.metrics import classification_report
```

```
# read the dataset
df = pd.read_csv('/content/heart_failure_clinical_records_dataset.csv')
df.head()
```

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_bloo
0	75.0	0	582	0	20	
1	55.0	0	7861	0	38	
2	65.0	0	146	0	20	
3	50.0	1	111	0	20	
4	65.0	1	160	1	20	

```
df.info()
```

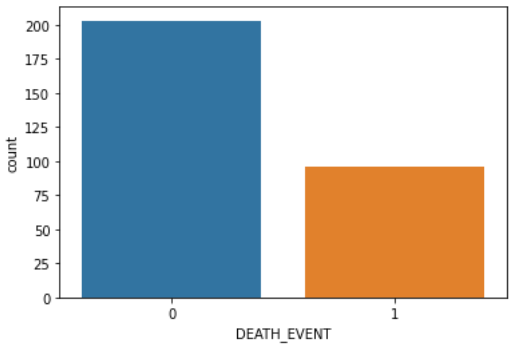
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

```
df['DEATH_EVENT'].value_counts()
```

```
0    203
1     96
Name: DEATH_EVENT, dtype: int64
```

```
sns.countplot(df['DEATH_EVENT'])
```

```
<AxesSubplot:xlabel='DEATH_EVENT', ylabel='count'>
```



```
#assigning values to features as X and target as y
X=df.drop(["DEATH_EVENT"],axis=1)
y=df["DEATH_EVENT"]
```

```
#spliting test and training sets
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=71)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(239, 12)
(60, 12)
(239,)
(60,)
```

```
#Set up a standard scaler for the features

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test  = scaler.transform(X_test)
```

```
X_train
```

```
array([[-0.20329345,  1.16890403, -0.20625801, ..., -1.46449201,
        -0.69604026, -0.44441423],
       [-1.36539901, -0.85550223, -0.50867365, ...,  0.68283063,
        -0.69604026, -0.59704027],
       [-1.53141409, -0.85550223,  4.3960104 , ...,  0.68283063,
         1.43669849, -0.49528958],
       ...,
       [ 2.86798552,  1.16890403, -0.23401134, ...,  0.68283063,
        -0.69604026, -0.96588652],
       [ 0.12873671, -0.85550223, -0.36799295, ...,  0.68283063,
         1.43669849, -1.47463998],
       [-1.28239147,  1.16890403,  1.206291  , ...,  0.68283063,
        -0.69604026, -0.48257074]])
```

```
# step 1: initialize model
ann = Sequential()

# step 2: add layers into model
ann.add(Dense(units= 10, activation = 'relu')) # create  hidden layers
ann.add(Dense(units= 10, activation = 'relu')) # create  hidden layers

ann.add(Dense(units = 1, activation = 'sigmoid')) # output layer
```

```
# step 3: establish connection between the layers
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
# step 4: train the model
history = ann.fit(X_train, y_train, batch_size = 32, epochs = 150, validation_split = 0.2)
```

```
6/6 [==============================] - 0s 10ms/step - loss: 0.3171 - accuracy: 0.8743 - val_loss: 0.3134 - val_accuracy: 0.8750
Epoch 89/150
6/6 [==============================] - 0s 12ms/step - loss: 0.3156 - accuracy: 0.8743 - val_loss: 0.3128 - val_accuracy: 0.8750
Epoch 90/150
6/6 [==============================] - 0s 10ms/step - loss: 0.3141 - accuracy: 0.8743 - val_loss: 0.3117 - val_accuracy: 0.8750
Epoch 91/150
6/6 [==============================] - 0s 10ms/step - loss: 0.3127 - accuracy: 0.8743 - val_loss: 0.3113 - val_accuracy: 0.8750
Epoch 92/150
6/6 [==============================] - 0s 9ms/step - loss: 0.3114 - accuracy: 0.8796 - val_loss: 0.3106 - val_accuracy: 0.8750
Epoch 93/150
6/6 [==============================] - 0s 9ms/step - loss: 0.3099 - accuracy: 0.8848 - val_loss: 0.3101 - val_accuracy: 0.8750
Epoch 94/150
6/6 [==============================] - 0s 8ms/step - loss: 0.3084 - accuracy: 0.8848 - val_loss: 0.3095 - val_accuracy: 0.8750
Epoch 95/150
6/6 [==============================] - 0s 9ms/step - loss: 0.3075 - accuracy: 0.8901 - val_loss: 0.3085 - val_accuracy: 0.8750
Epoch 96/150
6/6 [==============================] - 0s 9ms/step - loss: 0.3060 - accuracy: 0.8901 - val_loss: 0.3080 - val_accuracy: 0.8750
Epoch 97/150
6/6 [==============================] - 0s 8ms/step - loss: 0.3054 - accuracy: 0.8848 - val_loss: 0.3060 - val_accuracy: 0.8750
Epoch 98/150
6/6 [==============================] - 0s 9ms/step - loss: 0.3039 - accuracy: 0.8848 - val_loss: 0.3062 - val_accuracy: 0.8750
Epoch 99/150
6/6 [==============================] - 0s 12ms/step - loss: 0.3027 - accuracy: 0.8848 - val_loss: 0.3056 - val_accuracy: 0.8750
Epoch 100/150
6/6 [==============================] - 0s 10ms/step - loss: 0.3018 - accuracy: 0.8953 - val_loss: 0.3051 - val_accuracy: 0.8750
Epoch 101/150
6/6 [==============================] - 0s 12ms/step - loss: 0.3006 - accuracy: 0.8953 - val_loss: 0.3056 - val_accuracy: 0.8750
Epoch 102/150
6/6 [==============================] - 0s 12ms/step - loss: 0.2993 - accuracy: 0.8953 - val_loss: 0.3057 - val_accuracy: 0.8750
Epoch 103/150
6/6 [==============================] - 0s 8ms/step - loss: 0.2983 - accuracy: 0.8953 - val_loss: 0.3057 - val_accuracy: 0.8750
Epoch 104/150
6/6 [==============================] - 0s 11ms/step - loss: 0.2973 - accuracy: 0.8953 - val_loss: 0.3062 - val_accuracy: 0.8750
Epoch 105/150
6/6 [==============================] - 0s 8ms/step - loss: 0.2960 - accuracy: 0.8953 - val_loss: 0.3066 - val_accuracy: 0.8542
Epoch 106/150
6/6 [==============================] - 0s 9ms/step - loss: 0.2950 - accuracy: 0.8901 - val_loss: 0.3068 - val_accuracy: 0.8542
Epoch 107/150
6/6 [==============================] - 0s 8ms/step - loss: 0.2939 - accuracy: 0.8901 - val_loss: 0.3060 - val_accuracy: 0.8542
Epoch 108/150
```

```
# step 5: make predictions
y_pred = ann.predict(X_test)
y_pred
```

```
       [1.0792710e-02],
       [3.4195152e-01],
       [1.1708825e-02],
       [5.2020136e-02],
       [6.2205382e-02],
       [3.1800143e-02],
       [2.3907950e-02],
       [8.0514497e-01],
       [7.9514199e-01],
       [4.8968709e-01],
       [8.0889231e-01],
       [4.2456616e-02],
       [2.0562400e-01],
       [1.9417398e-01],
       [8.3172190e-01],
       [2.9036936e-01],
       [2.0525673e-01],
       [6.0600036e-01],
       [1.6603772e-02],
       [6.2161712e-03],
       [3.8612181e-01],
       [2.1511118e-01],
       [5.3075773e-01],
       [6.5496795e-02],
       [3.5690423e-02],
       [5.9423786e-01],
       [5.0989735e-01],
       [6.6816825e-01],
       [2.1678555e-01],
       [1.9325352e-01],
       [9.7599748e-04],
       [2.1181139e-03],
       [3.7843511e-01],
       [1.5000390e-02],
       [2.7494353e-01],
       [6.7775510e-02],
       [2.4826832e-02],
       [2.2854013e-03],
       [4.4204746e-03],
       [3.9427146e-01],
       [1.8947221e-01],
```

```
       [2.2958335e-03],
       [9.1191649e-01],
       [4.3970287e-02],
       [4.7416048e-04],
       [7.3146284e-01],
       [2.4568458e-04],
       [9.4857715e-02],
       [1.3643870e-01],
       [7.7790475e-01],
       [2.7104491e-01],
       [7.3600608e-01],
       [4.4750604e-01],
       [7.9199350e-01],
       [8.4212637e-03]], dtype=float32)
```

```
# step 6: set the threshold
y_pred = np.where(y_pred<0.5, 0, 1)
y_pred
```

```
array([[1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [0],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
       [1],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0],
       [1],
       [0],
```

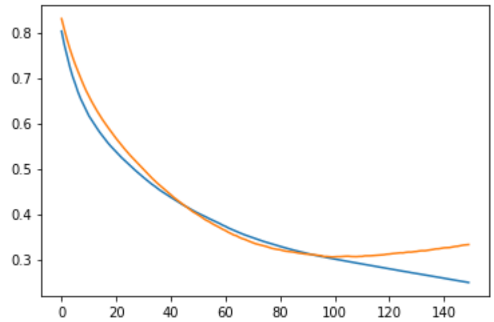
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        43
           1       0.60      0.53      0.56        17

    accuracy                           0.77        60
   macro avg       0.71      0.69      0.70        60
```

```
weighted avg       0.76      0.77      0.76        60
```

```
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
```

```
[<matplotlib.lines.Line2D at 0x7f0798e73b20>]
```



✓ 0s completed at 8:37 PM