Make a ANN Model of the given data set
predict its precision and accuracy and plot it

In [1]:
```
## Imports Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler , OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import f1_score,confusion_matrix ,accuracy_score,ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from imblearn.over_sampling import RandomOverSampler
from sklearn.ensemble import RandomForestClassifier
```

In [2]:
```
## 1- prepare the data set according to need (numeric)

data = pd.read_csv("../DataSets/heart.csv")
data.head()
```

Out[2]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisea
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	

In [3]:
```
## check for missing values to Clean and preprocess data
print("Duplicated",data.duplicated().sum())
data.isna().sum()
```

```
Duplicated 0
```

Out[3]:
```
Age             0
Sex             0
ChestPainType   0
RestingBP       0
Cholesterol     0
FastingBS       0
RestingECG      0
MaxHR           0
ExerciseAngina  0
Oldpeak         0
ST_Slope        0
HeartDisease    0
dtype: int64
```

In [4]:
```
## drop any missing values
df = data.dropna()

## Check for y_values
outLabel = df["HeartDisease"].value_counts()
outLabel
```

Out[4]:
```
1    508
0    410
Name: HeartDisease, dtype: int64
```

In [5]:
```
data.head(1)
```

Out[5]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisea
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	

In [6]:
```
##  Now do the preprocessing ( Label Encoding & StandardScaling)


X = df.drop("HeartDisease",axis=1)
Y = df["HeartDisease"]


scaling = StandardScaler()
encoder = OneHotEncoder()

category =["Sex","ChestPainType","RestingECG","ExerciseAngina","ST_Slope"]
numerical = ["Age","RestingBP","Cholesterol","MaxHR","Oldpeak"]


transform = ColumnTransformer([("numerical",scaling,numerical),
                               ("category",encoder,category)],remainder="passthrough")

trans_x =  transform.fit_transform(X)
trans_x.shape
```

Out[6]: (918, 20)

In [7]:
```
# train _test_split
np.random.seed(42)
x_train ,x_test ,y_train ,y_test = train_test_split(trans_x , Y,test_size =0.3 ,random_state = 42)
```

DeepLearning Model

In [26]:
```
 ## import necessary libraries/frameworks
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
```

In [41]:
```
model = Sequential()

model.add(Dense(units = 128 , activation = "relu" ,input_shape=(20,)))


model.add(Dense(units = 64 ,activation = "relu"))

model.add(Dense(2 ,activation = "softmax"))

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

model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_28 (Dense)            (None, 128)               2688

 dense_29 (Dense)            (None, 64)                8256

 dense_30 (Dense)            (None, 2)                 130

=================================================================
Total params: 11074 (43.26 KB)
Trainable params: 11074 (43.26 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [53]:
```
## Encode y_labels using onehotencoder

from keras.utils import to_categorical

y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)

model.fit(x_train ,y_train_encoded ,batch_size = 10 ,epochs = 100)
```

```
Epoch 1/100
65/65 [==============================] - 0s 4ms/step - loss: 0.0065 - accuracy: 0.9984
Epoch 2/100
65/65 [==============================] - 0s 4ms/step - loss: 0.0778 - accuracy: 0.9751
Epoch 3/100
65/65 [==============================] - 0s 4ms/step - loss: 0.2068 - accuracy: 0.9377
Epoch 4/100
65/65 [==============================] - 0s 4ms/step - loss: 0.0713 - accuracy: 0.9798
Epoch 5/100
65/65 [==============================] - 0s 4ms/step - loss: 0.0351 - accuracy: 0.9922
Epoch 6/100
65/65 [==============================] - 0s 5ms/step - loss: 0.0117 - accuracy: 0.9984
Epoch 7/100
65/65 [==============================] - 0s 4ms/step - loss: 0.0089 - accuracy: 1.0000
Epoch 8/100
65/65 [==============================] - 0s 5ms/step - loss: 0.0075 - accuracy: 1.0000
Epoch 9/100
65/65 [==============================] - 0s 5ms/step - loss: 0.0071 - accuracy: 1.0000
Epoch 10/100
```

In [54]:
```
## Predict the value
y_predict_encoded = model.predict(x_test)

y_pred = np.argmax(y_predict_encoded,axis = 1)
y_predict_encoded
print(y_pred)
print(y_test)
```

```
9/9 [==============================] - 0s 5ms/step
[0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 1 1 1 1
 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1 0 0 0 1 0 1 1 1 1 1 0 0 1 1
 0 1 0 1 0 1 1 1 1 0 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1
 1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 0 0 1 1 1 1 1 0
 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 0 1 0 1 1 0 0 1 0 1 0 1 0
 0 1 0 1 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 1 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 0
 1 0 1 0 0 1 0 0 1 1 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1
 1 0 1 0 0 1 0 1 1 1 1 1 1 0 1 1 0]
668    0
30     1
377    1
535    1
807    0
      ..
133    1
813    0
734    1
360    1
875    0
Name: HeartDisease, Length: 276, dtype: int64
```

In [55]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, roc_au
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

# Precision
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

# Recall
recall = recall_score(y_test, y_pred)
print("Recall:", recall)

# F1-Score
f1 = f1_score(y_test, y_pred)
print("F1-Score:", f1)

# ROC-AUC Score (only applicable for binary classification tasks)
roc_auc = roc_auc_score(y_test, y_pred)
print("ROC-AUC Score:", roc_auc)
```

```
Confusion Matrix:
 [[ 98  14]
 [ 32 132]]

Accuracy: 0.8333333333333334
Precision: 0.9041095890410958
Recall: 0.8048780487804879
F1-Score: 0.8516129032258065
ROC-AUC Score: 0.8399390243902439
```

Check the accuracy using Decision Tree

In [56]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(x_train ,y_train)

## Predict the value
y_preds = clf.predict(x_test)
clf.score(x_test ,y_test)
```

Out[56]:
```
0.7536231884057971
```

In [57]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_preds)
print("Confusion Matrix:\n", cm)

# Accuracy
accuracy = accuracy_score(y_test, y_preds)
print("\nAccuracy:", accuracy)

# Precision
precision = precision_score(y_test, y_preds)
print("Precision:", precision)

# Recall
recall = recall_score(y_test, y_preds)
print("Recall:", recall)

# F1-Score
f1 = f1_score(y_test, y_preds)
print("F1-Score:", f1)

# ROC-AUC Score (only applicable for binary classification tasks)
roc_auc = roc_auc_score(y_test, y_preds)
print("ROC-AUC Score:", roc_auc)
```

```
Confusion Matrix:
 [[ 91  21]
 [ 47 117]]

Accuracy: 0.7536231884057971
Precision: 0.8478260869565217
Recall: 0.7134146341463414
F1-Score: 0.7748344370860927
ROC-AUC Score: 0.7629573170731707
```

In [ ]:
```

```