Decision Tree Classification on Hepatitis Dataset

In [1]:
```
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, r
from sklearn.metrics import classification_report
```

Data Loading

In [2]:
```
url = 'https://github.com/rashakil-ds/Public-Datasets/raw/main/hepatitis.csv'
df = pd.read_csv(url)
df.head()
```

Out[2]:

	Class	AGE	SEX	STEROID	ANTIVIRALS	FATIGUE	MALAISE	ANOREXIA	LIVER BIG	LIVER FIRM	SPLEEN PALPABLE	SF
0	0	30	2	1.0	2	2	2	2	1.0	2.0	2.0	
1	0	50	1	1.0	2	1	2	2	1.0	2.0	2.0	
2	0	78	1	2.0	2	1	2	2	2.0	2.0	2.0	
3	0	31	1	NaN	1	2	2	2	2.0	2.0	2.0	
4	0	34	1	2.0	2	2	2	2	2.0	2.0	2.0	

In [3]:
```
df.isnull().sum()
```

Out[3]:
```
Class              0
AGE                0
SEX                0
STEROID            1
ANTIVIRALS         0
FATIGUE            0
MALAISE            0
ANOREXIA           0
LIVER BIG          9
LIVER FIRM        10
SPLEEN PALPABLE    4
SPIDERS            4
ASCITES            4
VARICES            4
BILIRUBIN          5
ALK PHOSPHATE     28
SGOT               3
ALBUMIN           15
PROTIME           66
HISTOLOGY          0
dtype: int64
```

Check the missing values

In [4]:
```
missing_values = df.isnull().sum()
print(missing_values)
```

```
Class              0
AGE                0
SEX                0
STEROID            1
ANTIVIRALS         0
FATIGUE            0
MALAISE            0
ANOREXIA           0
LIVER BIG          9
LIVER FIRM        10
SPLEEN PALPABLE    4
SPIDERS            4
ASCITES            4
VARICES            4
BILIRUBIN          5
ALK PHOSPHATE     28
SGOT               3
ALBUMIN           15
PROTIME           66
HISTOLOGY          0
dtype: int64
```

In [5]:
```
df.dropna(inplace = True)
```

Split the dataset into features (X) and Target Variable (y)

In [6]:
```
X = df.drop('Class', axis = 1)
y = df['Class']
```

Convert the target variable to binary(1 for positive, 0 for negative)

In [7]:
```
y = y.apply(lambda x: 1 if x == 1 else 0)
```

Decision Tree Model

Split the dataset into training and testing sets

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
```

Build a Decision Tree classifier

In [9]:
```
clf = DecisionTreeClassifier()
```

Train the model on the training set

In [10]:
```
clf.fit(X_train, y_train)
```

Out[10]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Make predictions on the testing set

In [11]:
```
clf.score(X_test, y_test)
```

Out[11]:
```
0.8333333333333334
```

Evaluate the model

In [12]:
```
y_pred = clf.predict(X_test)
```

Evaluate the model using various matrics

In [13]:
```
conf_matrix = confusion_matrix(y_test, y_pred)
prec_score = precision_score(y_test, y_pred)
rec_score = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)
```

Print the evaluate results

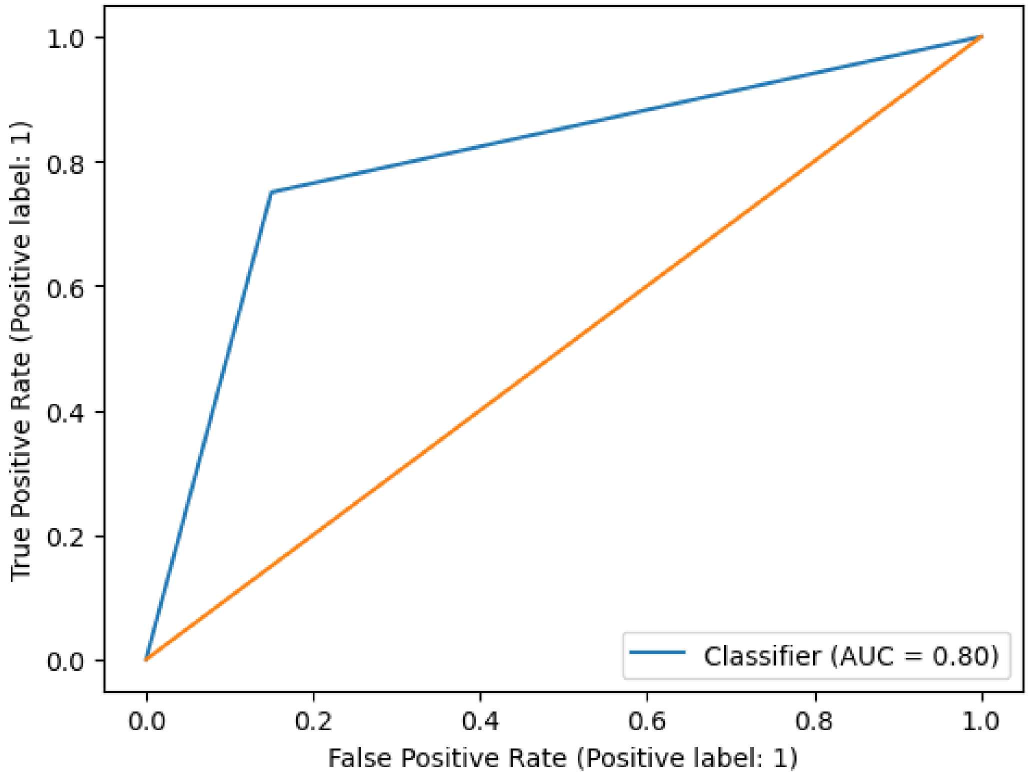
In [14]:
```
print('Confusion Matrix:\n', conf_matrix)
print('Precision Score: ', prec_score)
print('Recall Score: ', rec_score)
print('F1 Score: ', f1)
print('AUC-ROC Score: ', roc_auc)
```

```
Confusion Matrix:
 [[17  3]
 [ 1  3]]
Precision Score:  0.5
Recall Score:  0.75
F1 Score:  0.6
AUC-ROC Score:  0.8
```

Classifier for AUC

In [15]:
```
from sklearn.metrics import RocCurveDisplay
import matplotlib.pyplot as plt
RocCurveDisplay.from_predictions(y_test, y_pred)
plt.plot([0,1],[0,1])
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x2068a6a1510>]
```



Results and Analysis

Summary of Evaluation Metrics

The model exhibits a decent performance with a Precision of 0.5, indicating a balanced prediction of positive outcomes. A Recall of 0.75 suggests that the model captures a substantial portion of the actual positive cases. The F1 Score of 0.6 signifies a reasonable balance between Precision and Recall. An AUC-ROC Score of 0.8 indicates a good ability to discriminate between positive and negative instances.

Strengths and Weaknesses of the Decision Tree Model:

Strengths: Decision trees are interpretable and suitable for simple decision-making scenarios. They handle both numerical and categorical data well. Decision trees provide insights into feature importance.

Weaknesses: The model may be sensitive to noisy data and prone to overfitting. Decision trees might not capture complex relationships in the data as effectively as more sophisticated models. They may struggle with class imbalances.

Recommendations: Consider tuning hyperparameters to potentially improve model performance. Explore ensemble methods or alternative algorithms to address weaknesses. Evaluate the model's performance on different subsets of the data to ensure robustness.