

## Using Decision Trees to Classify Heart Disease Cases

In this notebook, we aim to address the classification problem of predicting whether a patient is likely to have heart disease, using the Heart Disease Dataset and Decision Tree algorithm.

### Import Modules

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid")
```

Importing the dataset and performing initial data exploration. I will use "copy path" of the required dataset to include file name in my code to avoid error as shown below.

```
df = pd.read_excel("/content/data.xlsx")
df.head(10)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
7	44	1	1	120	263	0	1	173	0	0.0	2	0	3	1
8	52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
9	57	1	2	150	168	0	1	174	0	1.6	2	0	2	1

Next steps:

[Generate code with df](#)[View recommended plots](#)[New interactive sheet](#)

```
df.shape
```

```
(303, 14)
```

```
duplicates = df.duplicated().sum()
print("Numbers of duplicate rows:",duplicates)
```


```
Numbers of duplicate rows: 1
```

```
df= df.drop_duplicates()
```

```
print("New shape after removing duplicates:", df.shape)
```

```
New shape after removing duplicates: (302, 14)
```

```
df.isnull().sum()
```



	0
age	0
sex	0
cp	0
trestbps	0
chol	0
fbs	0
restecg	0
thalach	0
exang	0
oldpeak	0
slope	0
ca	0
thal	0
target	0

dtype: int64

df.dtypes

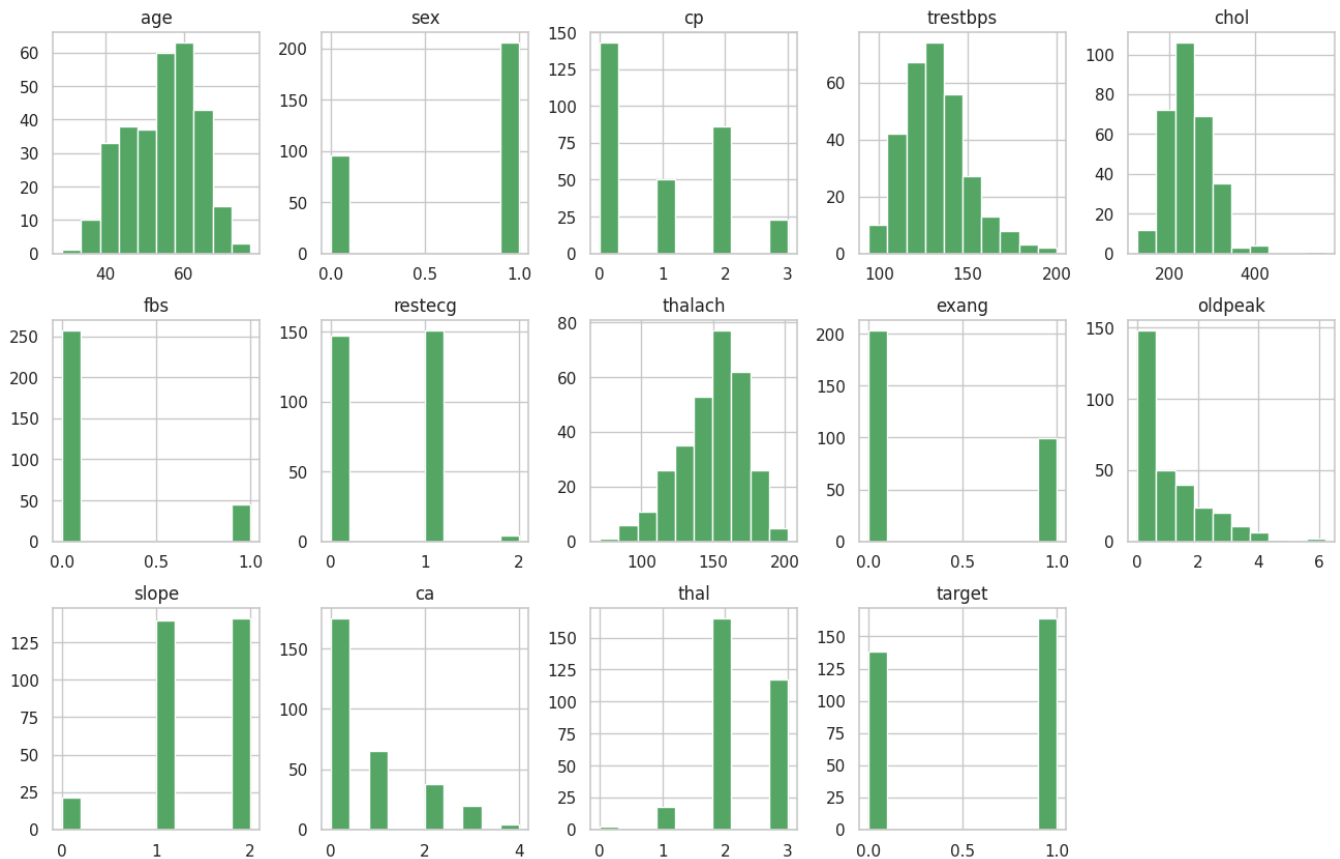


	0
age	int64
sex	int64
cp	int64
trestbps	int64
chol	int64
fbs	int64
restecg	int64
thalach	int64
exang	int64
oldpeak	float64
slope	int64
ca	int64
thal	int64
target	int64

dtype: object

```
df.hist(layout = (3,5), figsize=(16,10), color = 'g')
print('Data Distribution')
```

## Data Distribution



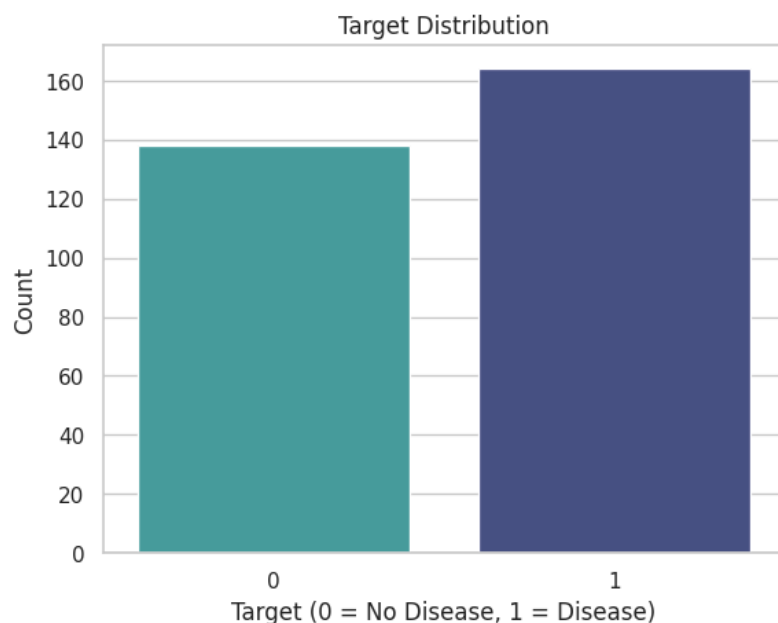
```
print("This looks like a fairly balanced dataset, as distribution of majority and minority class is around 55:45")
```

```
sns.countplot(x="target", data=df, palette="mako_r")
plt.title("Target Distribution")
plt.xlabel("Target (0 = No Disease, 1 = Disease)")
plt.ylabel("Count")
plt.show()
```

This looks like a fairly balanced dataset, as distribution of majority and minority class is around 55:45  
/tmp/ipython-input-111-441674168.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

```
sns.countplot(x="target", data=df, palette="mako_r")
```

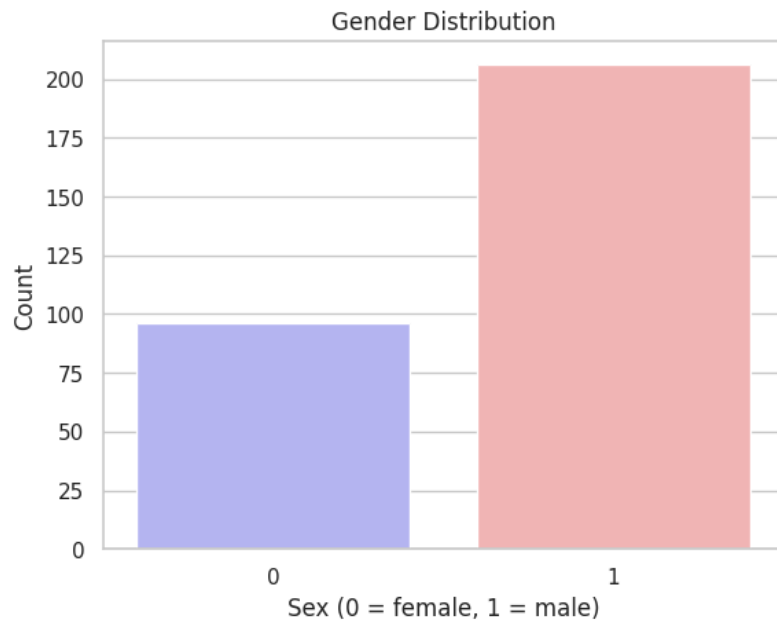


```
sns.countplot(x='sex', data=df, palette="bwr")
plt.xlabel("Sex (0 = female, 1 = male)")
plt.title("Gender Distribution")
plt.ylabel("Count")
plt.show()
```

↗ /tmp/ipython-input-112-2583002801.py:1: FutureWarning:

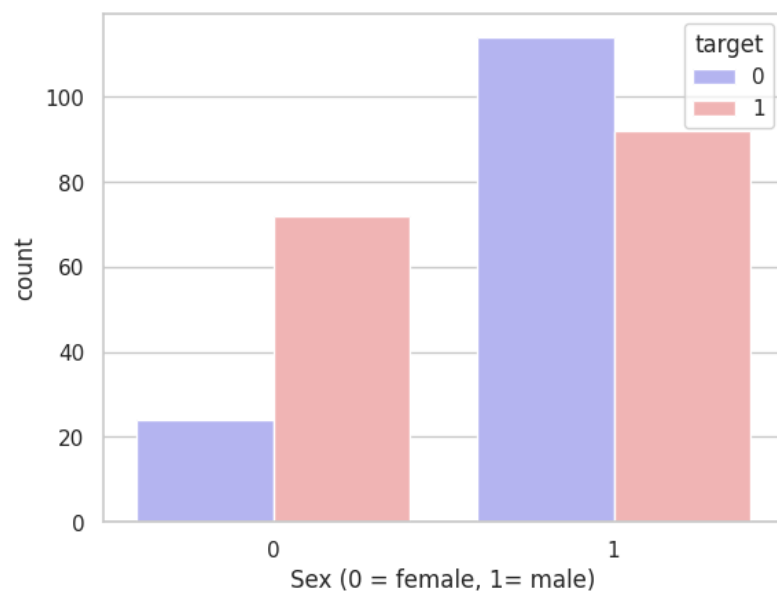
Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `l

```
sns.countplot(x='sex', data=df, palette="bwr")
```

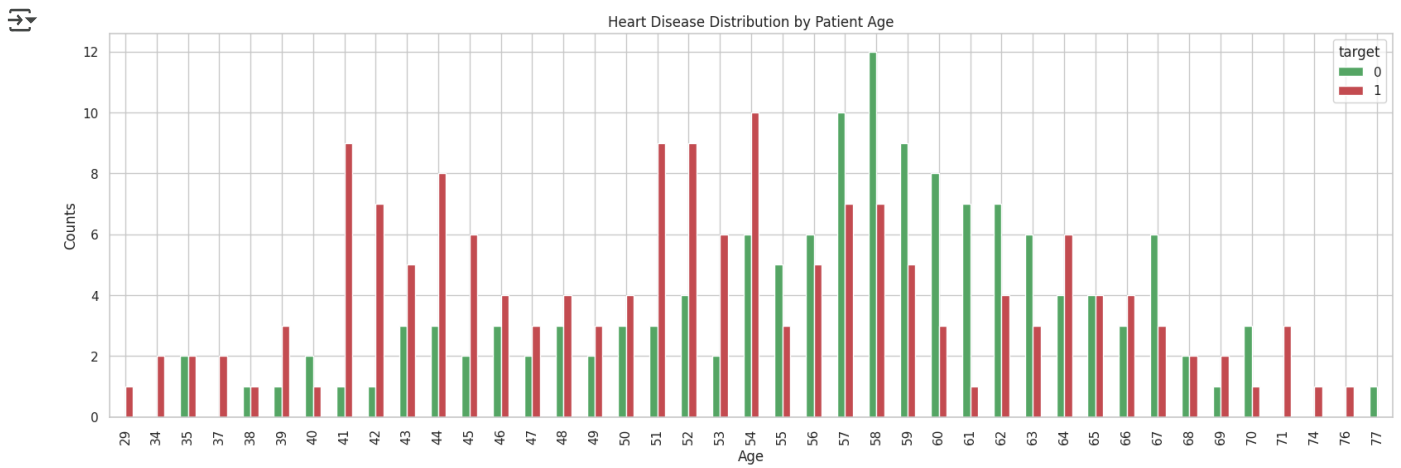


```
print('Analysing distribution of target and sex (0-female 1-male)')
sns.countplot(x = df['sex'], hue = df['target'], palette='bwr')
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```

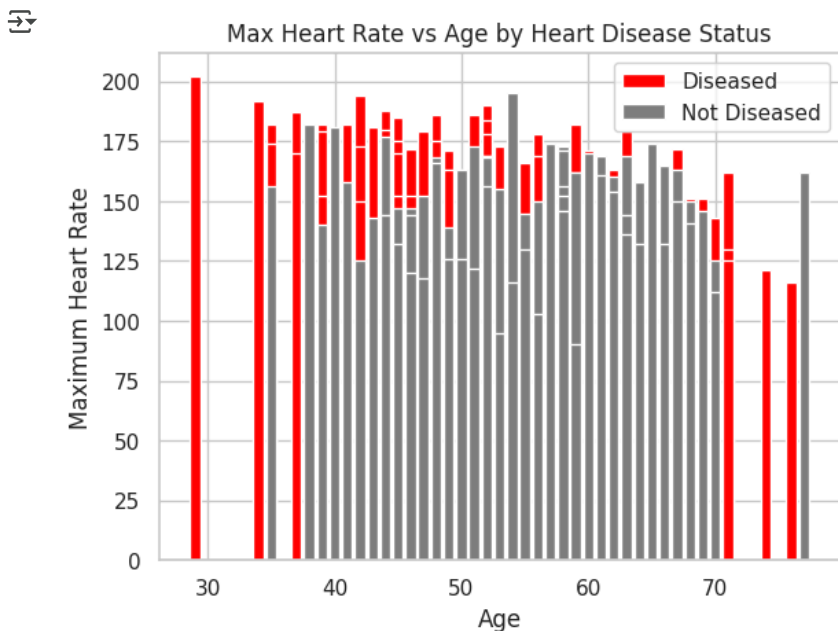
↗ Analysing distribution of target and sex (0-female 1-male)



```
pd.crosstab(df.age,df.target).plot(kind="bar",figsize=(20,6), color = ['g','r'])
plt.title('Heart Disease Distribution by Patient Age')
plt.xlabel('Age')
plt.ylabel('Counts')
plt.show()
```



```
plt.bar(df.age[df.target==1], df.thalach[df.target==1], color="red")
plt.bar(df.age[df.target==0], df.thalach[df.target==0], color="grey")
plt.legend(["Diseased", "Not Diseased"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.title("Max Heart Rate vs Age by Heart Disease Status")
plt.show()
```



```
df.describe()
```

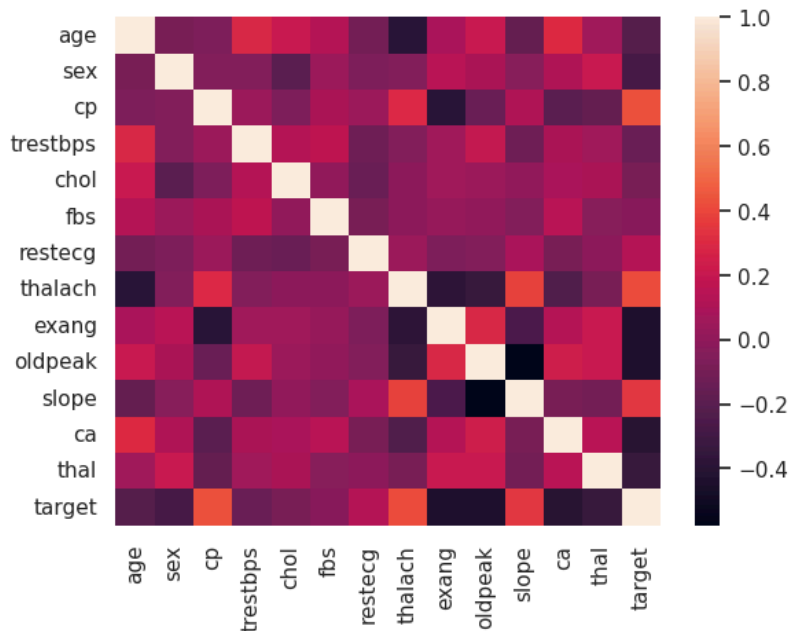
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	sl
count	302.00000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000000	302.000
mean	54.42053	0.682119	0.963576	131.602649	246.500000	0.149007	0.526490	149.569536	0.327815	1.043046	1.397
std	9.04797	0.466426	1.032044	17.563394	51.753489	0.356686	0.526027	22.903527	0.470196	1.161452	0.616
min	29.00000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000
25%	48.00000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.250000	0.000000	0.000000	1.000
50%	55.50000	1.000000	1.000000	130.000000	240.500000	0.000000	1.000000	152.500000	0.000000	0.800000	1.000
75%	61.00000	1.000000	2.000000	140.000000	274.750000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000
max	77.00000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000

```
print(df.corr()['target'])
sns.heatmap(df.corr())
plt.show()
```

```

↗ age      -0.221476
sex       -0.283609
cp        0.432080
trestbps  -0.146269
chol      -0.081437
fbs       -0.026826
restecg   0.134874
thalach   0.419955
exang     -0.435601
oldpeak   -0.429146
slope     0.343940
ca        -0.408992
thal      -0.343101
target    1.000000
Name: target, dtype: float64

```



The features do not exhibit strong correlation (greater than 0.6) with the target variable. If any features had shown high positive or negative correlation, they would have been considered for removal to avoid potential bias during model training.

### Feature Engineering

```

feature_engg_data = df.copy()
outlier_data = df.copy()

factor = 3
columns_to_include = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak', 'ca']

for column in columns_to_include:
    upper_lim = feature_engg_data[column].mean() + feature_engg_data[column].std() * factor
    lower_lim = feature_engg_data[column].mean() - feature_engg_data[column].std() * factor
    feature_engg_data = feature_engg_data[
        (feature_engg_data[column] < upper_lim) & (feature_engg_data[column] > lower_lim)
    ]

outlier_data = pd.concat([outlier_data, feature_engg_data]).drop_duplicates(keep=False)

print("Cleaned dataset shape:", feature_engg_data.shape)
print("Outlier data shape:", outlier_data.shape)

```

```

↗ Cleaned dataset shape: (289, 14)
Outlier data shape: (13, 14)

```

In the next step, we will normalize the data and split it into training, validation, and test sets to build the model. The outlier data will be kept separate to later evaluate how well the model performs on data points that were excluded during outlier removal.

### Normalise Data

```

from sklearn import preprocessing
import pandas as pd

```

```
def normalize_data(df):
    val = df.values
    min_max_normalizer = preprocessing.MinMaxScaler()
    norm_val = min_max_normalizer.fit_transform(val)
    df2 = pd.DataFrame(norm_val, columns=df.columns) # Keep original column names
    return df2
```

```
norm_feature_engg_data = normalize_data(feature_engg_data)
norm_outlier_data = normalize_data(outlier_data)
```

## Data Splits

We split the feature-engineered data into training, validation, and test sets using a 70:20:10 ratio. This helps ensure we have enough data to train the model, tune it using validation data, and then check how well it performs on unseen test data."

```
from sklearn.model_selection import train_test_split

input_data = norm_feature_engg_data.drop(['target'], axis='columns')
targets = norm_feature_engg_data[['target']]

x, x_test, y, y_test = train_test_split(input_data, targets, test_size=0.1, random_state=5)
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.22, random_state=5)
```

## Model Building

To evaluate the model, we will use AUC-ROC scores, accuracy, and the confusion matrix to measure how well the model generalizes, especially on the validation and test sets. Initially, these metrics will be applied to the test set to check the model's fit and identify any potential bias from the training phase.

```
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix
import math
import seaborn as sns

def evaluation_metrics(y_true, y_pred, model):

    accuracy = accuracy_score(y_true, y_pred)
    roc_auc = roc_auc_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)

    print("Accuracy of", model, ": {:.2f}".format(accuracy))
    print("ROC AUC Score of", model, ": {:.2f}".format(roc_auc))
    print("Confusion Matrix of", model, ": \n")

    plt.figure(figsize=(5,5))
    sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues');
    plt.ylabel('Actual label');
    plt.xlabel('Predicted label');
    title = 'AUC-ROC Score: {:.2f}'.format(roc_auc)
    plt.title(title)
    plt.show()
```

```
import time
```

```
def ml_model(classifier, classifier_name, **kwargs):
    """
    Generic method to train the selected classification algorithm on train, validation and test dataset.
    """
    # Fit model
    if kwargs['x_train'] is not None:
        model = classifier.fit(kwargs['x_train'], kwargs['y_train'])
        y_pred_train= model.predict(kwargs['x_train'])
        print('*****')
        print('Training Set Performance:')
        print('*****')
        evaluation_metrics(kwargs['y_train'], y_pred_train, classifier_name)

    if kwargs['x_valid'] is not None:
        y_pred_valid = model.predict(kwargs['x_valid'])
        print('*****')
        print('Validation Set Performance:')
```

```

    print('*****')
    evaluation_metrics(kwarg['y_valid'], y_pred_valid, classifier_name)

if kwarg['x_test'] is not None:
    start = time.time()
    y_pred_test= classifier.predict(kwarg['x_test'])
    end = time.time()
    print('*****')
    print('Test Set Performance:')
    print('*****')
    print('Model Time Complexity on Test Data: {:.3f} milli seconds'.format((end - start) * 1000))
    evaluation_metrics(kwarg['y_test'], y_pred_test, classifier_name)

from sklearn.model_selection import learning_curve
from sklearn.model_selection import ShuffleSplit

def plot_learning_curves(train_sizes, train_scores_mean, train_scores_std, test_scores_mean, test_scores_std, model_name):
    """
    Method to generate learning curves for using training and cross validation scores
    """
    plt.title(model_name)

    plt.xlabel("Training examples")
    plt.ylabel("Score")

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
                     color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",label="Cross-validation score")
    plt.legend(loc="best")
    plt.show()

def plot_model_scalability_curves(train_sizes, training_time_mean, training_time_std, model_name):
    """
    Method to generate scalability curve to see the model complexity
    """
    plt.plot(train_sizes, training_time_mean, 'o-', color = 'purple')
    plt.fill_between(train_sizes, training_time_mean - training_time_std,
                     training_time_mean + training_time_std, alpha=0.1, color = 'purple')
    plt.xlabel("Training examples")
    plt.ylabel("Training time")
    plt.title("Scalability of "+ model_name)
    plt.show()

def plot_model_performance_curves(training_time_mean, test_scores_mean, test_scores_std, model_name):
    """
    Method to generate performance curves to see if increase model complexity would improve score or not
    """
    plt.plot(training_time_mean, test_scores_mean, 'o-')
    plt.fill_between(training_time_mean, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1)
    plt.xlabel("Training Time")
    plt.ylabel("Score")
    plt.title("Performance of "+ model_name)
    plt.show()

def generate_learning_curves(model, model_name, X, y, xlim = None, ylim=None,
                             epochs =None, figsize = (20,5)):
    """
    Generic method to generate Learning Curves, Scalability curves and Performance curves
    Referred - https://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_learning\_curve.html#sphx-glr-auto-exam
    """
    cross_valid = ShuffleSplit(n_splits=100, test_size=0.2, random_state=0)

    train_sizes=np.linspace(.1, 1.0, 5)
    train_sizes, train_scores, test_scores, training_time, _ = learning_curve(model, X, y, cv=cross_valid,
                                                                              n_jobs=epochs, train_sizes=train_sizes,
                                                                              return_times=True)

    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)

    test_scores_mean = np.mean(test_scores, axis=1)

```



```
test_scores_std = np.std(test_scores, axis=1)

training_time_mean = np.mean(training_time, axis=1)
training_time_std = np.std(training_time, axis=1)

# Plot learning curve
plot_learning_curves(train_sizes, train_scores_mean, train_scores_std, test_scores_mean, test_scores_std, model_name)

# Plot scalability curve
plot_model_scalability_curves(train_sizes, training_time_mean, training_time_std, model_name)

# Plot model performance score
plot_model_performance_curves(training_time_mean, test_scores_mean, test_scores_std, model_name)
```

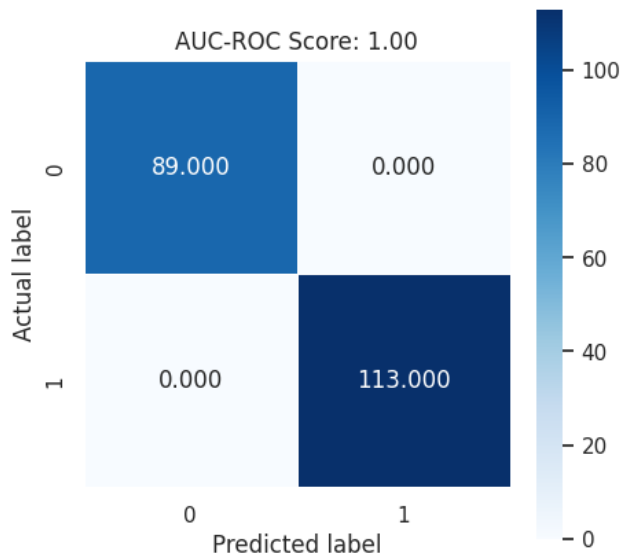
## Classification Algorithms

### Baseline - Decision Tree

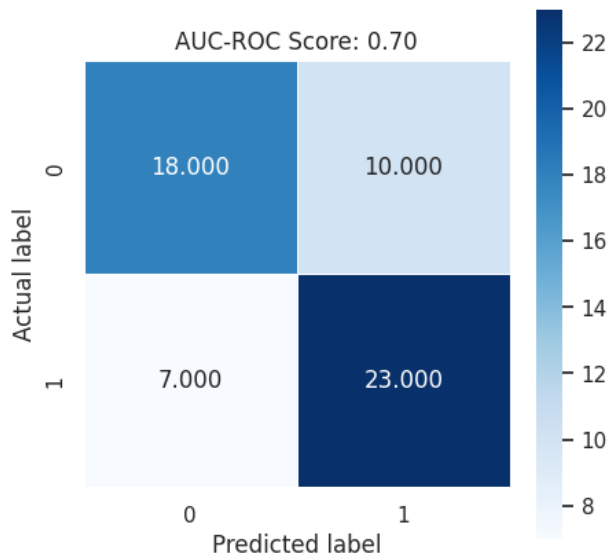
```
from sklearn.tree import DecisionTreeClassifier, plot_tree

DTC = DecisionTreeClassifier(criterion='entropy', random_state=3) # Baseline model without any form of pruning
ml_model(DTC, 'Decision Tree', x_train = x_train, y_train = y_train, x_valid = x_valid, y_valid = y_valid, x_test = None)
```

```
*****  
Training Set Performance:  
*****  
Accuracy of Decision Tree : 1.00  
ROC AUC Score of Decision Tree : 1.00  
Confusion Matrix of Decision Tree :
```



```
*****  
Validation Set Performance:  
*****  
Accuracy of Decision Tree : 0.71  
ROC AUC Score of Decision Tree : 0.70  
Confusion Matrix of Decision Tree :
```



```
import matplotlib.pyplot as plt  
from sklearn.tree import plot_tree
```

```
plt.figure(figsize=(20,10))  
plot_tree(DTC, filled=True, feature_names=x_train.columns, class_names=['No Disease', 'Disease'])  
plt.title("Decision Tree Structure")  
plt.show()
```

