In [1]:

```
#Examining Factors Responsible for Heart Attacks
```

In [2]:

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

In [3]:

```
#Section 1 - Loading the data and exploratory data analysis
```

In [4]:

```
#Load the data
```

In [7]:

```
data = pd.read_csv("heart_Data.csv")
data.head()
```

Out[7]:

	Unnamed: 0	age	sex	chest_pain_type	resting_bp	cholestoral	fasting_blood_sugar	restecg	max_hr	exang	oldpeak	slop
0	0	63	1	3	145	233	1	0	150	0	2.3	
1	1	37	1	2	130	250	0	1	187	0	3.5	
2	2	41	0	1	130	204	0	0	172	0	1.4	
3	3	56	1	1	120	236	0	1	178	0	0.8	
4	4	57	0	0	120	354	0	1	163	1	0.6	

In [12]:

```
#data.to_csv("heart_Data")
```

Performing EDA: 4. Identify the data variables which might be categorical in nature. Describe and explore these variables using appropriate tools. For example: count plot.

In [9]:

```
data.shape
```

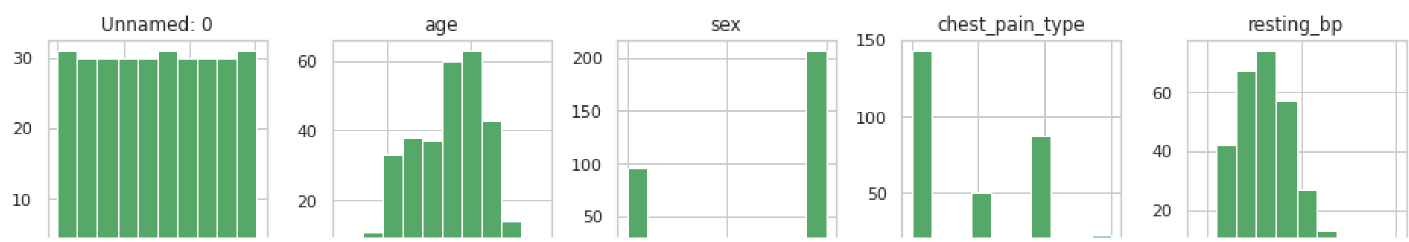
Out[9]:

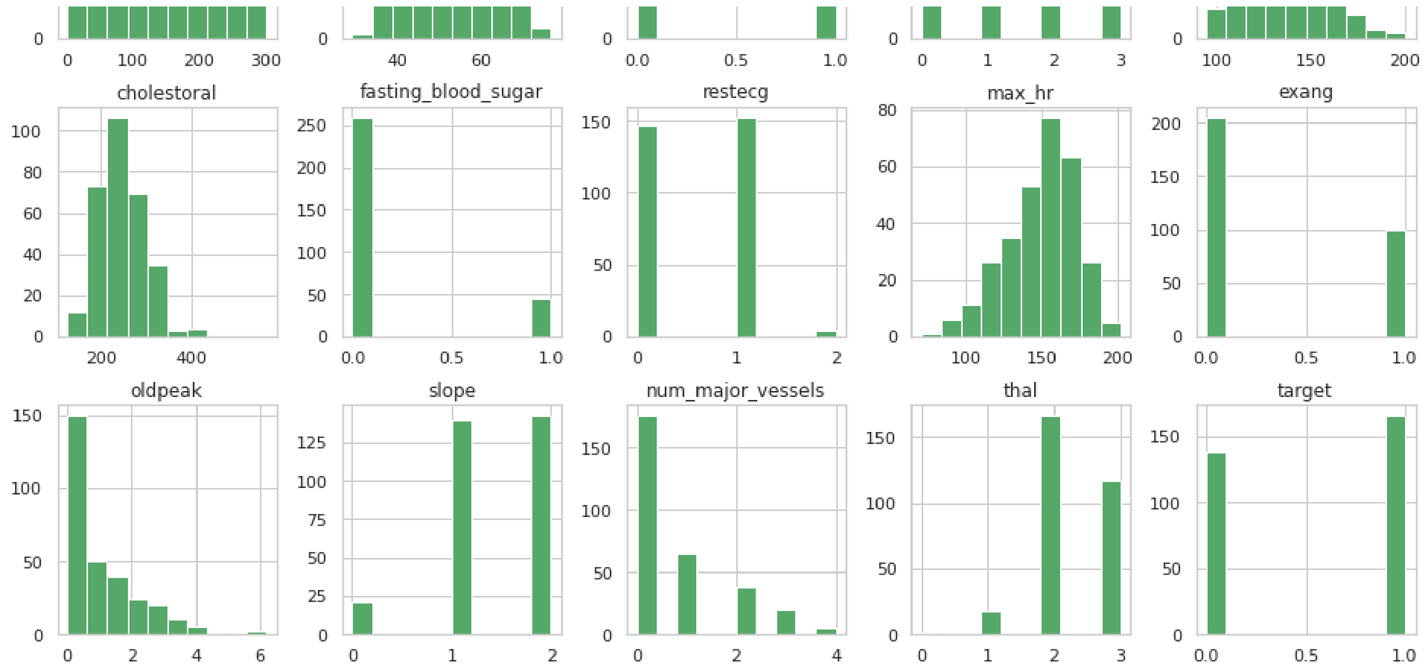
```
(303, 15)
```

In [10]:

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

```
Data Distribution
```

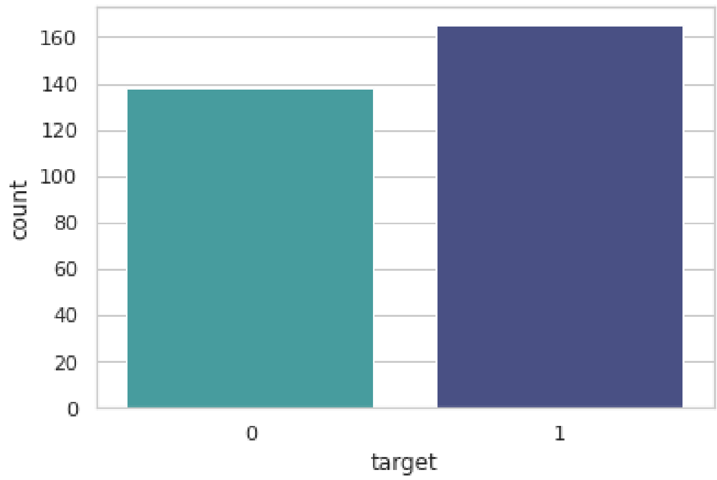




In [11]:

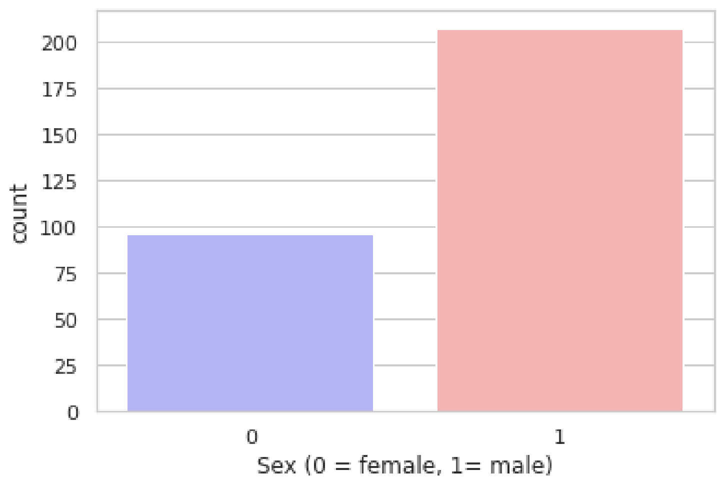
```
print('This looks like a fairly balanced dataset, as distribution of majority and minorit
y class is around 55:45')
sns.countplot(x="target", data=data, palette="mako_r")
plt.show()
```

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



In [13]:

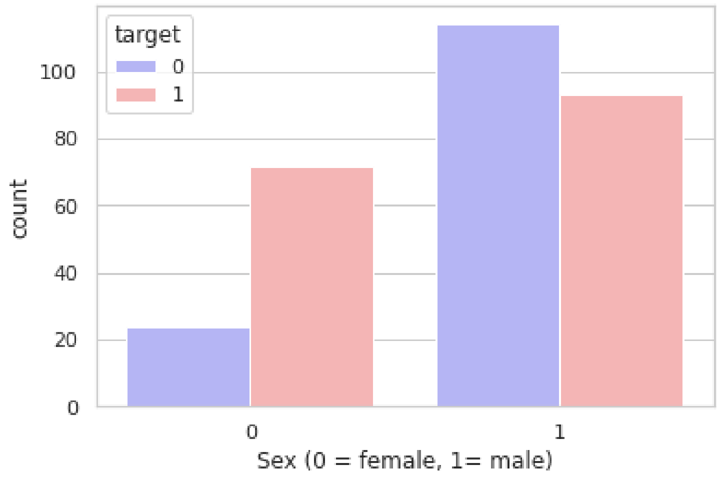
```
sns.countplot(x='sex', data=data, palette="bwr")
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```



In [14]:

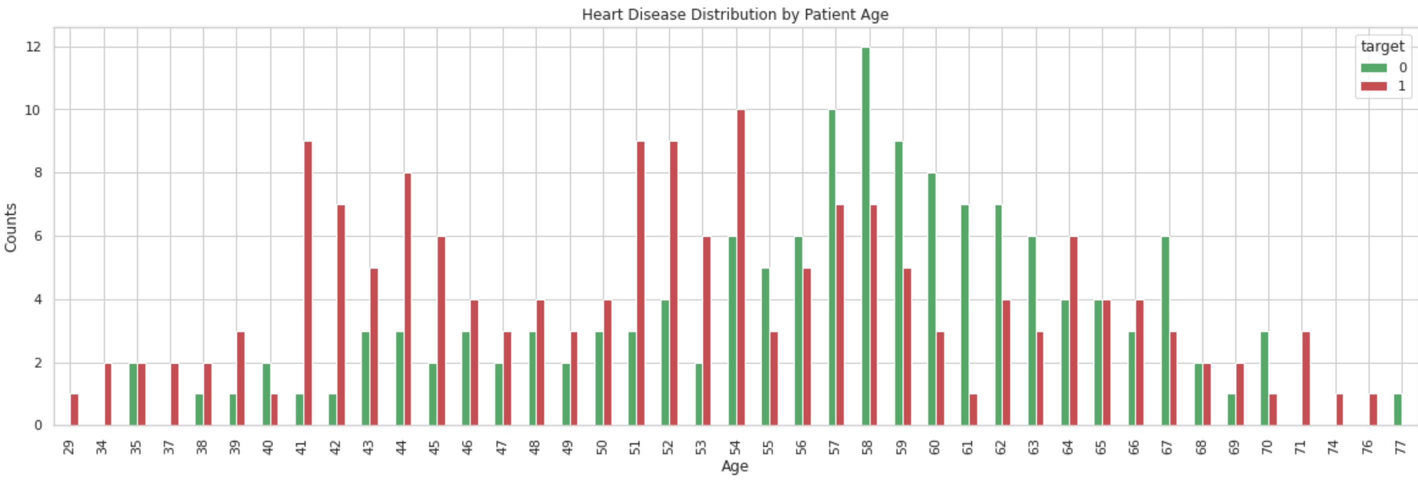
```
print('Analysing distribution of target and sex (0-female 1-male)')
sns.countplot(x = data['sex'], hue = data['target'], palette='bwr')
plt.xlabel("Sex (0 = female, 1= male)")
plt.show()
```

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



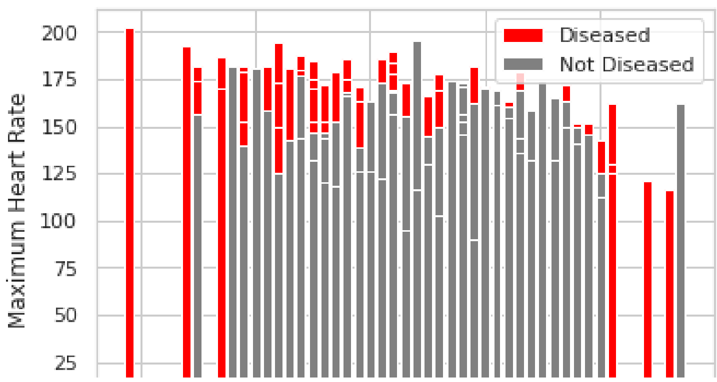
In [15]:

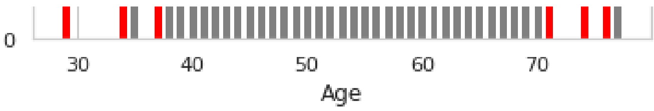
```
pd.crosstab(data.age,data.target).plot(kind="bar",figsize=(20,6), color = ['g','r'])
plt.title('Heart Disease Distribution by Patient Age')
plt.xlabel('Age')
plt.ylabel('Counts')
plt.show()
```



In [16]:

```
plt.bar(data.age[data.target==1], data.max_hr[(data.target==1)], color="red")
plt.bar(data.age[data.target==0], data.max_hr[(data.target==0)], color="grey")
plt.legend(["Diseased", "Not Diseased"])
plt.xlabel("Age")
plt.ylabel("Maximum Heart Rate")
plt.show()
```





In [17]:

```
## Section 2 - Data Pre-processing
```

In [18]:

```
#Null or missing value check
```

In [19]:

```
data.isnull().sum()
```

Out[19]:

```
Unnamed: 0             0
age                    0
sex                    0
chest_pain_type        0
resting_bp             0
cholestoral            0
fasting_blood_sugar    0
restecg                0
max_hr                 0
exang                  0
oldpeak                0
slope                  0
num_major_vessels      0
thal                   0
target                 0
dtype: int64
```

We do not see any missing values for this dataset.

In [20]:

```
#Duplicate inspection
```

In [21]:

```
data.duplicated().any()
```

Out[21]:

```
False
```

Looks like the dataset has some duplicates. Let's remove the duplicates

In [22]:

```
data.drop_duplicates(subset=None, inplace=True)
data.duplicated().any()
```

Out[22]:

```
False
```

In [23]:

```
data.shape
```

Out[23]:

```
(303, 15)
```

So, we can see that there was one duplicate row

In [24]:

```
#One Hot Encoding
```

In [25]:

```
def encode_features(df, features):
    '''
    Method for one-hot encoding all selected categorical fields
    '''
    for f in features:
        if(f in df.columns):
            encoded = pd.get_dummies(df[f])
            encoded = encoded.add_prefix(f + '_')
            df = pd.concat([df, encoded], axis=1)
        else:
            print('Feature not found')
            return df

        df.drop(columns=features, inplace = True)

    return df
```

In [26]:

```
features_to_encode = ['thal', 'slope', 'chest_pain_type', 'restecg']
encoded = encode_features(data, features_to_encode)
data = encoded.copy()
print(data.shape)
```

```
Feature not found
(303, 15)
```

In [27]:

```
data.columns
```

Out[27]:

```
Index(['Unnamed: 0', 'age', 'sex', 'resting_bp', 'cholestoral',
       'fasting_blood_sugar', 'max_hr', 'exang', 'oldpeak',
       'num_major_vessels', 'target', 'thal_0', 'thal_1', 'thal_2', 'thal_3'],
      dtype='object')
```

In [28]:

```
#Outlier Inspection
```

In [30]:

```
data.describe()
```

Out[30]:

	Unnamed: 0	age	sex	resting_bp	cholestoral	fasting_blood_sugar	max_hr	exang	oldpeak
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	151.000000	54.366337	0.683168	131.623762	246.264026	0.148515	149.646865	0.326733	1.039604
std	87.612784	9.082101	0.466011	17.538143	51.830751	0.356198	22.905161	0.469794	1.161075
min	0.000000	29.000000	0.000000	94.000000	126.000000	0.000000	71.000000	0.000000	0.000000
25%	75.500000	47.500000	0.000000	120.000000	211.000000	0.000000	133.500000	0.000000	0.000000
50%	151.000000	55.000000	1.000000	130.000000	240.000000	0.000000	153.000000	0.000000	0.800000
75%	226.500000	61.000000	1.000000	140.000000	274.500000	0.000000	166.000000	1.000000	1.600000
max	302.000000	77.000000	1.000000	200.000000	564.000000	1.000000	202.000000	1.000000	6.200000

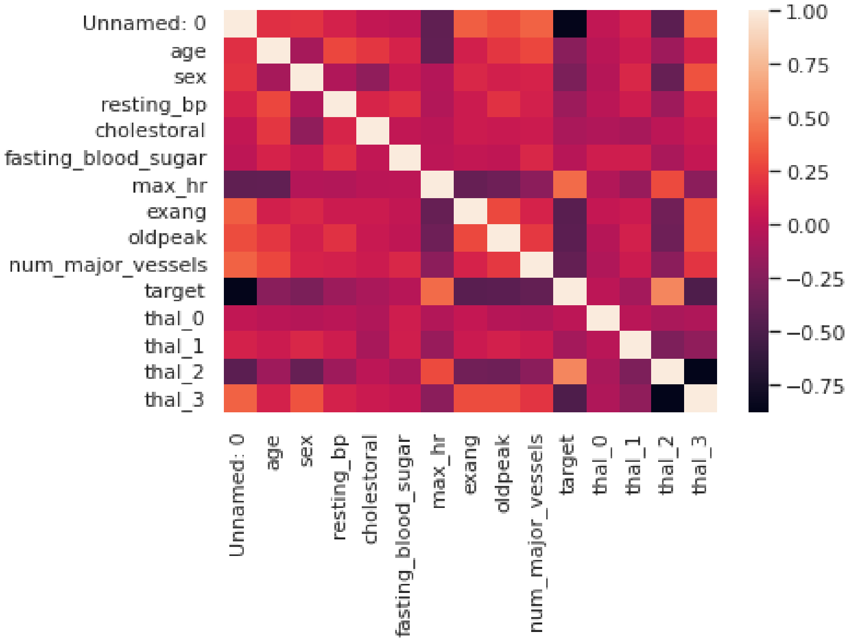
In [31]:

```
# Data Correlation
```

In [32]:

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

```
Unnamed: 0            -0.862585
age                   -0.225439
sex                   -0.280937
resting_bp            -0.144931
cholestoral           -0.085239
fasting_blood_sugar   -0.028046
max_hr                 0.421741
exang                 -0.436757
oldpeak               -0.430696
num_major_vessels     -0.391724
target                 1.000000
thal_0                -0.007293
thal_1                -0.106589
thal_2                 0.527334
thal_3                -0.486112
Name: target, dtype: float64
```



In [34]:

```
#Section-3 : Feature Engineering
```

In [35]:

```
feature_engg_data = data.copy()
outlier_data = data.copy()
target_index = data.columns.get_loc("target") # Use this for separating the target label
s from data
factor = 3
# Include this only for columns with suspected outliers
# Using a factor of 3, following Nelson's rule 1 to remove outliers - https://en.wikipedi
a.org/wiki/Nelson_rules
# Only for non-categorical fields
columns_to_include = ['age', 'resting_bp', 'cholestoral', 'max_hr', 'oldpeak', 'num_major
_vessels']
for column in columns_to_include:
    upper_lim = feature_engg_data[column].mean () + feature_engg_data[column].std () * f
actor
    lower_lim = feature_engg_data[column].mean () - feature_engg_data[column].std () * f
actor
    feature_engg_data = feature_engg_data[(feature_engg_data[column] < upper_lim) & (fea
ture_engg_data[column] > lower_lim)]

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

In [36]:

```
print(feature_engg_data.shape)
print(outlier_data.shape)
```

```
(289, 15)
(14, 15)
```

In [37]:

```
#Data Normalization
```

In [38]:

```
from sklearn import preprocessing

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)

    return df2

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

In [39]:

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

#Data Splits Splitting Feature Engineered Data into train-valid-test dataset in 70:20:10 Ratio, the choice of selecting this splitting ratio is to ensure we have sufficient training data, sufficient validation data for mainly hyper parameter tuning and sufficient testing data to ensure model generalization.

In [40]:

```
from sklearn.model_selection import train_test_split

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

x, x_test, y, y_test = train_test_split(input_data,targets,test_size=0.1,train_size=0.9,
random_state=5)
x_train, x_valid, y_train, y_valid = train_test_split(x,y,test_size = 0.22,train_size =0
.78, random_state=5)
```

In [41]:

```
#Section 4 - Building the Model
```

In [42]:

```
#Model Evaluation Metrics
```

In [43]:

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

In [44]:

```
def evaluatation_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()
```

In [45]:

```
import time

def ml_model(classifier, classifier_name, **kwargs):
    '''
    Generic method to train the selected classification algorithm on train, validation an
d 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('******************************************************')
        evaluatation_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('******************************************************')
        evaluatation_metrics(kwargs['y_valid'], y_pred_valid, classifier_name)

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

In [46]:

```
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_me
an, 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 scor
e")
    plt.legend(loc="best")
    plt.show()

def plot_model_scalability_curves(train_sizes, training_time_mean, training_time_std, mod
el_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 = 'pur
ple')
    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 impro
ve 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_learnin
g_curve.html#sphx-glr-auto-examples-model-selection-plot-learning-curve-py
    '''
    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=epoch
s, 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_m
ean, test_scores_std, model_name)

    # Plot scalability curve
    plot_model_scalability_curves(train_sizes, training_time_mean, training_time_std, mod
el_name)

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

In [47]:

```
#Classification Algorithms
```

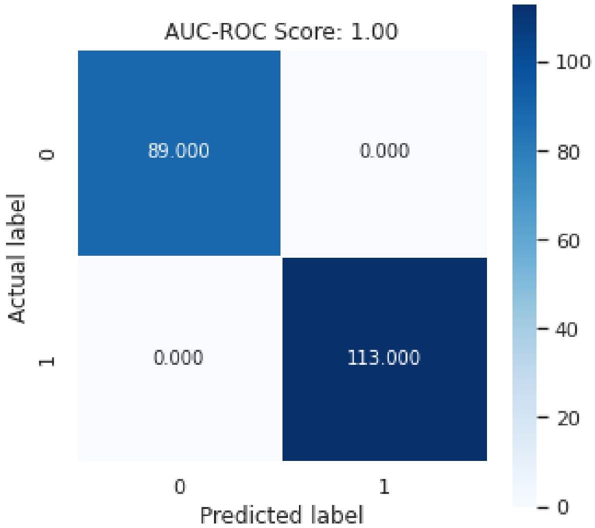
In [48]:

```
#Baseline - Decision Tree
```

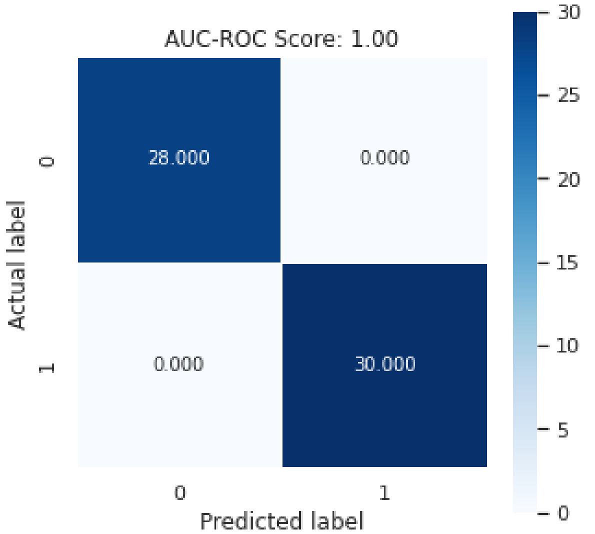
In [49]:

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

DTC = DecisionTreeClassifier(criterion='entropy', random_state=3) # Baseline model witho
ut 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 :
```

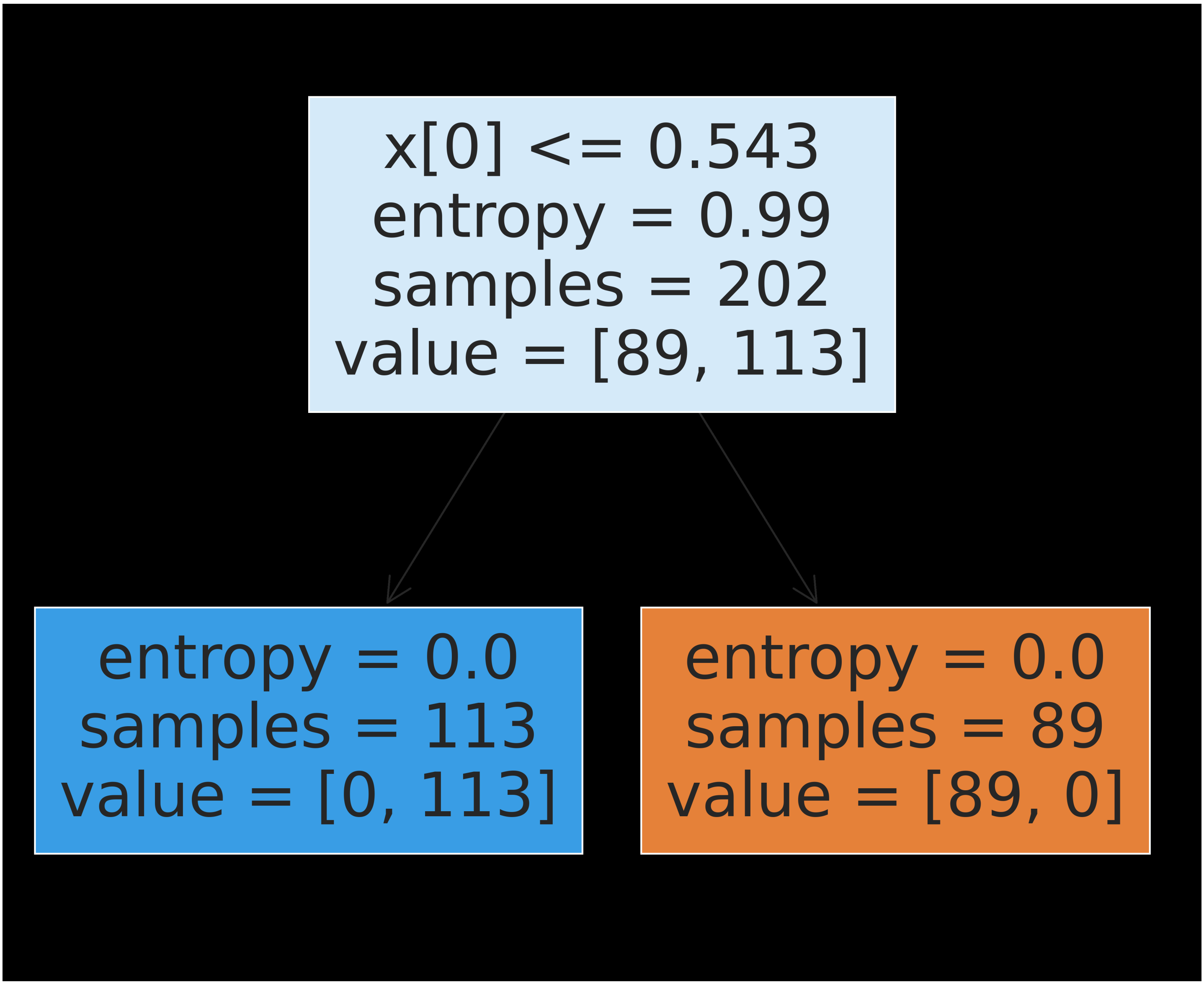



```
*****************************************************
Validation Set Performance:
*****************************************************
Accuracy of Decision Tree : 1.00
ROC AUC Score of Decision Tree : 1.00
Confusion Matrix of Decision Tree :
```



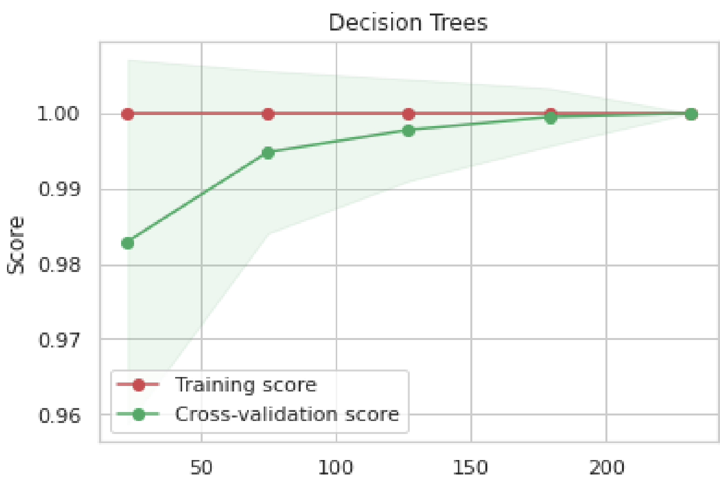
In [50]:

```
# Visualizing Decision Trees
fig, axes = plt.subplots(figsize = (12,10), dpi=500)
plot_tree(DTC,filled = True)
plt.show()
```

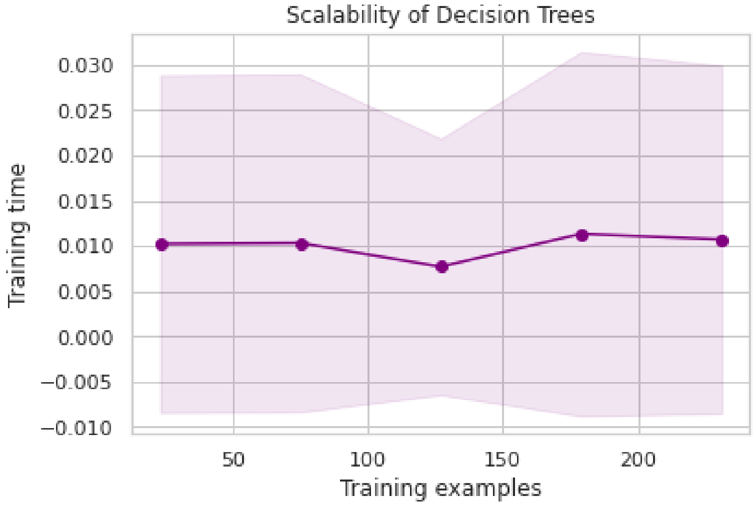


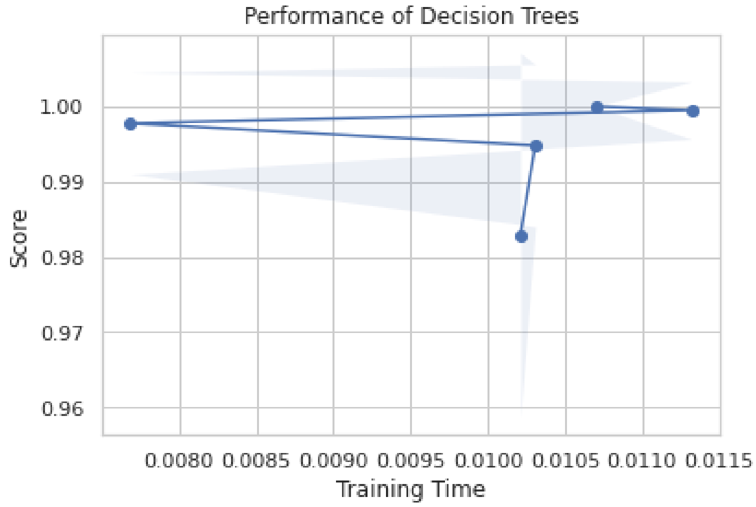
In [52]:

```
# Generation model curves on training and cross validation data
generate_learning_curves(
    model = DTC,
    model_name = "Decision Trees",
    X = input_data,
    y = targets,
    ylim=(0.7, 1.01),
    epochs=5)
```



Training examples





In [53]:

```
#Hyper-parameter tuning
```

In [54]:

```
# We will apply grid search to find the best possible set of hyper parameters
def DTGridSearch(X,y,param_lim):
    '''
    Decision Tree Grid Search to find the initial best guess of hyper-parameters
    '''
    param_grid = {
                'min_samples_leaf':np.linspace(param_lim[0],param_lim[1],20).astype(
'int'),
                'max_depth':np.arange(1,param_lim[2]),
                'criterion' : ['entropy' , 'gini']
    }

    best_DT = GridSearchCV(estimator = DecisionTreeClassifier(random_state=3), param_gri
d=param_grid, cv=10)
    best_DT.fit(X, y)

    print("Best Decision Tree Hyper-Parameters are:")
    print(best_DT.best_params_)

    return best_DT.best_params_['min_samples_leaf'], best_DT.best_params_['max_depth'],
best_DT.best_params_['criterion']
```

In [56]:

```
min_samples_leaf_lim = int(0.005 * len(x_train)) # 0.5% of length of training size
max_samples_leaf_lim = int(0.1 * len(x_train)) # 10% of length of training size
max_depth = 10
```

```
best_min_sample_leaf, best_max_depth, best_criterion = DTGridSearch( x_train,
                                                                      y_train,
                                                                      (min_samples_leaf_
lim, max_samples_leaf_lim, max_depth)
                                                                    )
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_162/1327311137.py in <cell line: 5>()
      3 max_depth = 10
      4
----> 5 best_min_sample_leaf, best_max_depth, best_criterion = DTGridSearch( x_train,
      6                                                                       y_train,
      7                                                                       (min_sampl
es_leaf_lim, max_samples_leaf_lim, max_depth)

/tmp/ipykernel_162/3665654205.py in DTGridSearch(X, y, param_lim)
     10     }
     11
---> 12     best_DT = GridSearchCV(estimator = DecisionTreeClassifier(random_state=3), pa
ram_grid=param_grid, cv=10)
     13     best_DT.fit(X, y)
     14

NameError: name 'GridSearchCV' is not defined
```

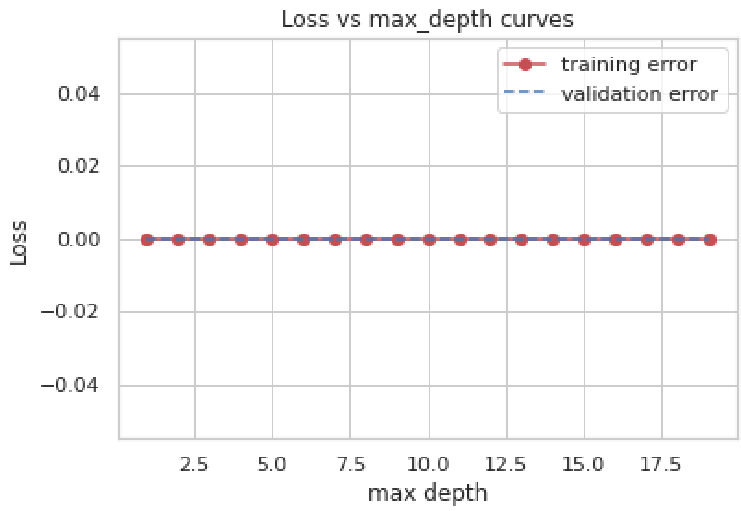
Best Decision Tree Hyper-Parameters are: {'criterion': 'entropy', 'max_depth': 1, 'min_samples_leaf': 1}

In [58]:

```
train=[]
valid=[]
for i in range(1,20):
    dec = DecisionTreeClassifier(criterion = 'gini', min_samples_leaf = 12, random_state
=3, max_depth=i)
    dec.fit(x_train, y_train)
    train.append(1- accuracy_score(dec.predict(x_train), y_train))
    valid.append(1- accuracy_score(dec.predict(x_valid), y_valid))

plt.title('Loss vs max_depth curves')
depth=[i for i in range(1,20)]
plt.plot(depth,train,'o-', color = 'r', label = 'training error')
plt.plot(depth,valid, '--', color = 'b', label = 'validation error')
plt.xlabel('max depth')
plt.ylabel('Loss')
plt.legend()
```

Out[58]:

```
<matplotlib.legend.Legend at 0x7f949a9c0d60>
```

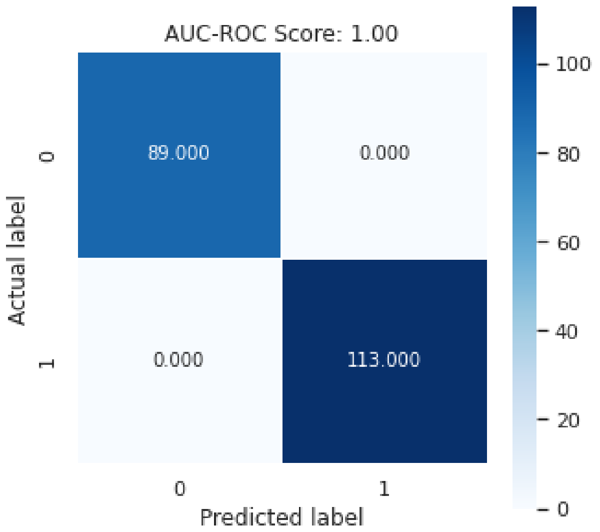


The loss curve also shows some over-fitting, but the lowest loss is obtained for a max_depth = 3. Now, let's apply these hyper-parameter values and let's see the model performance and generalization.

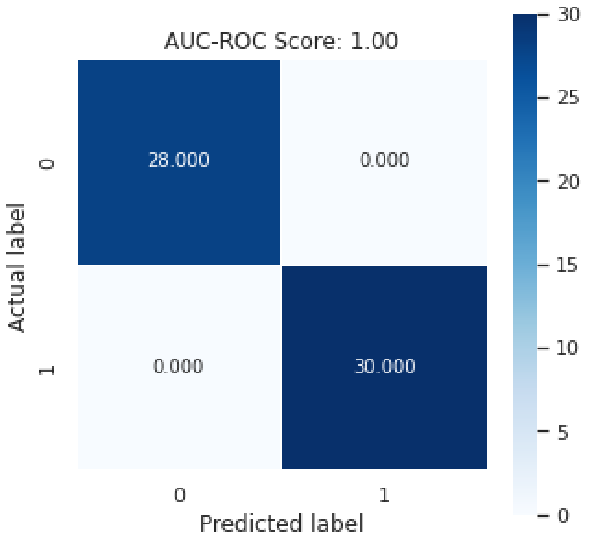
In [59]:

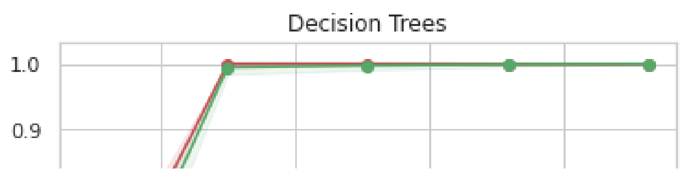
```
tuned_DTC = DecisionTreeClassifier(criterion='gini', max_depth = 3, min_samples_leaf = 1
2, random_state=3)
ml_model(tuned_DTC, 'Decision Tree', x_train = x_train, y_train = y_train, x_valid = x_v
alid, y_valid = y_valid, x_test = None)

generate_learning_curves(
    model = tuned_DTC,
    model_name = "Decision Trees",
    X = input_data,
    y = targets,
    ylim=(0.7, 1.01),
    epochs=5)
```

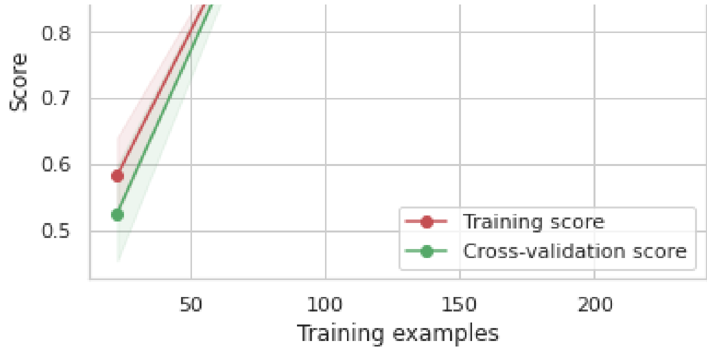
```
*****************************************************
Training Set Performance:
*****************************************************
Accuracy of Decision Tree : 1.00
ROC AUC Score of Decision Tree : 1.00
Confusion Matrix of Decision Tree :
```

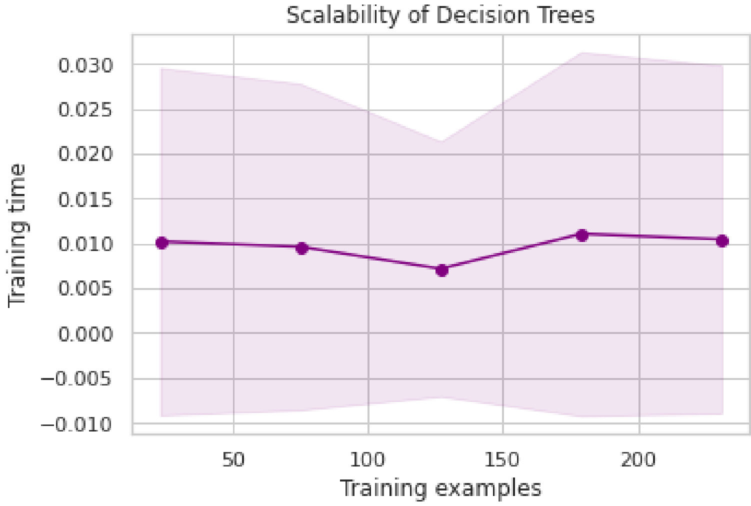


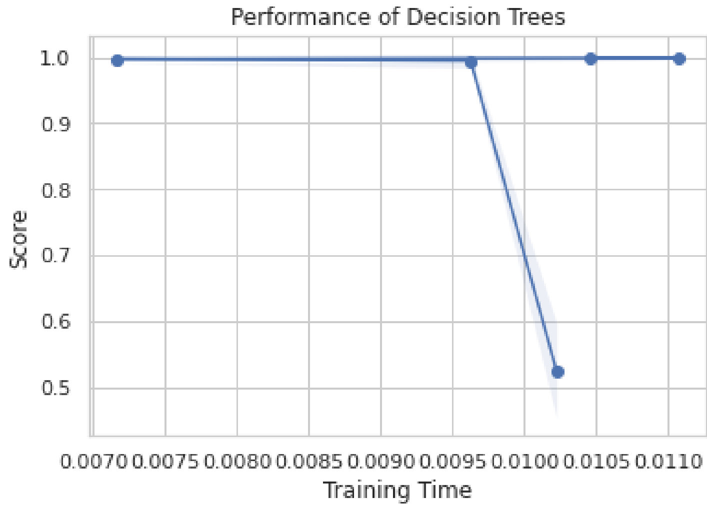

```
*****************************************************
Validation Set Performance:
*****************************************************
Accuracy of Decision Tree : 1.00
ROC AUC Score of Decision Tree : 1.00
Confusion Matrix of Decision Tree :
```











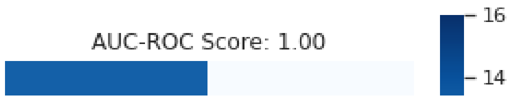
In [60]:

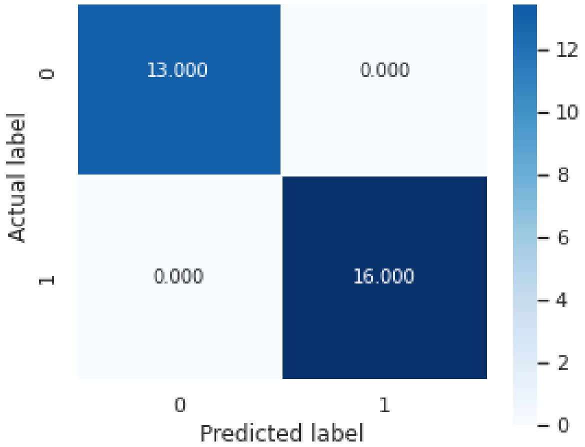
```
#Section 5 : Model Performance on Test Data
```

In [61]:

```
print('For the tuned Decision Tree model :\n')
ml_model(tuned_DTC, 'tuned Decision Tree', x_train = None, x_valid = None, x_test = x_test, y_test = y_test.values.ravel() )
```

```
For the tuned Decision Tree model :

*****************************************************
Test Set Performance:
*****************************************************
Model Time Complexity on Test Data: 1.455 milli seconds
Accuracy of tuned Decision Tree : 1.00
ROC AUC Score of tuned Decision Tree : 1.00
Confusion Matrix of tuned Decision Tree :
```





13.000
0.000
0.000
16.000
Actual label
0
1
Predicted label
0
1
12
10
8
6
4
2
0

In [ ]: