In [1]:

```
# Importing the required libraries
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

In [2]:

```
# Reading the csv file and putting it into 'df' object.
df = pd.read_csv('heart.csv')
```

In [3]:

```
df.columns
```

Out[3]:

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

In [4]:

```
df.head()
```

Out[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

In [5]:

```
df = df.drop_duplicates(ignore_index = True)
df.shape
```

Out[5]:

```
(302, 14)
```

In [6]:

```
# Putting feature variable to X
X = df.drop('target',axis=1)

# Putting response variable to y
y = df['target']
```

In [7]:

```
from sklearn.model_selection import train_test_split
```

In [8]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=42)
X_train.shape, X_test.shape
```

Out[8]:

```
((211, 13), (91, 13))
```

Fitting the decision tree with default hyperparameters, apart from max_depth which is 3 so that we can plot and read the tree.

In [9]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [10]:

```
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X_train, y_train)
```

Out[10]:

```
DecisionTreeClassifier(max_depth=3)
```

In [11]:

```
!pip install six
```

```
Requirement already satisfied: six in c:\users\nikki\anaconda3\lib\site-pack
ages (1.15.0)
```

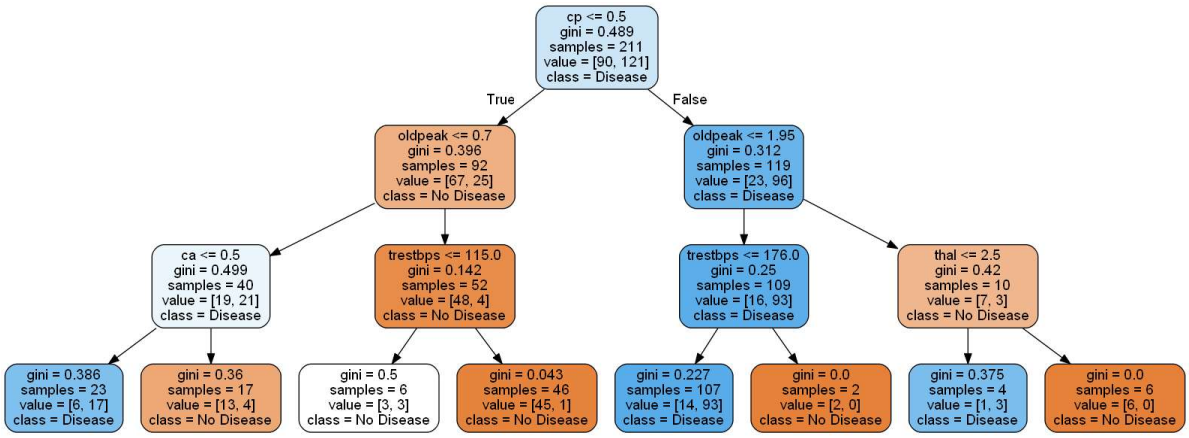
In [12]:

```
# Importing required packages for visualization
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus, graphviz
```

In [13]:

```
# plotting tree with max_depth=3
dot_data = StringIO()

export_graphviz(dt, out_file=dot_data, filled=True, rounded=True,
                feature_names=X.columns,
                class_names=['No Disease', "Disease"])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
#Image(graph.create_png(),width=800,height=900)
#graph.write_pdf("dt_heartdisease.pdf")
```

Out[13]:



Evaluating model performance

In [14]:

```
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)
```

In [15]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [16]:

```
print(accuracy_score(y_train, y_train_pred))
confusion_matrix(y_train, y_train_pred)
```

```
0.8625592417061612
```

Out[16]:

```
array([[ 69,  21],
       [  8, 113]], dtype=int64)
```

In [17]:

```
print(accuracy_score(y_test, y_test_pred))
confusion_matrix(y_test, y_test_pred)
```

```
0.7692307692307693
```

Out[17]:

```
array([[35, 13],
       [ 8, 35]], dtype=int64)
```

Creating helper functions to evaluate model performance and help plot the decision tree

In [18]:

```
def get_dt_graph(dt_classifier):
    dot_data = StringIO()
    export_graphviz(dt_classifier, out_file=dot_data, filled=True,rounded=True,
                    feature_names=X.columns,
                    class_names=['Disease', "No Disease"])
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph
```

In [19]:

```
def evaluate_model(dt_classifier):
    print("Train Accuracy :", accuracy_score(y_train, dt_classifier.predict(X_train)))
    print("Train Confusion Matrix:")
    print(confusion_matrix(y_train, dt_classifier.predict(X_train)))
    print("-"*50)
    print("Test Accuracy :", accuracy_score(y_test, dt_classifier.predict(X_test)))
    print("Test Confusion Matrix:")
    print(confusion_matrix(y_test, dt_classifier.predict(X_test)))
```

Without setting any hyper-parameters

In [20]:

```
dt_default = DecisionTreeClassifier(random_state=42)
dt_default.fit(X_train, y_train)
```

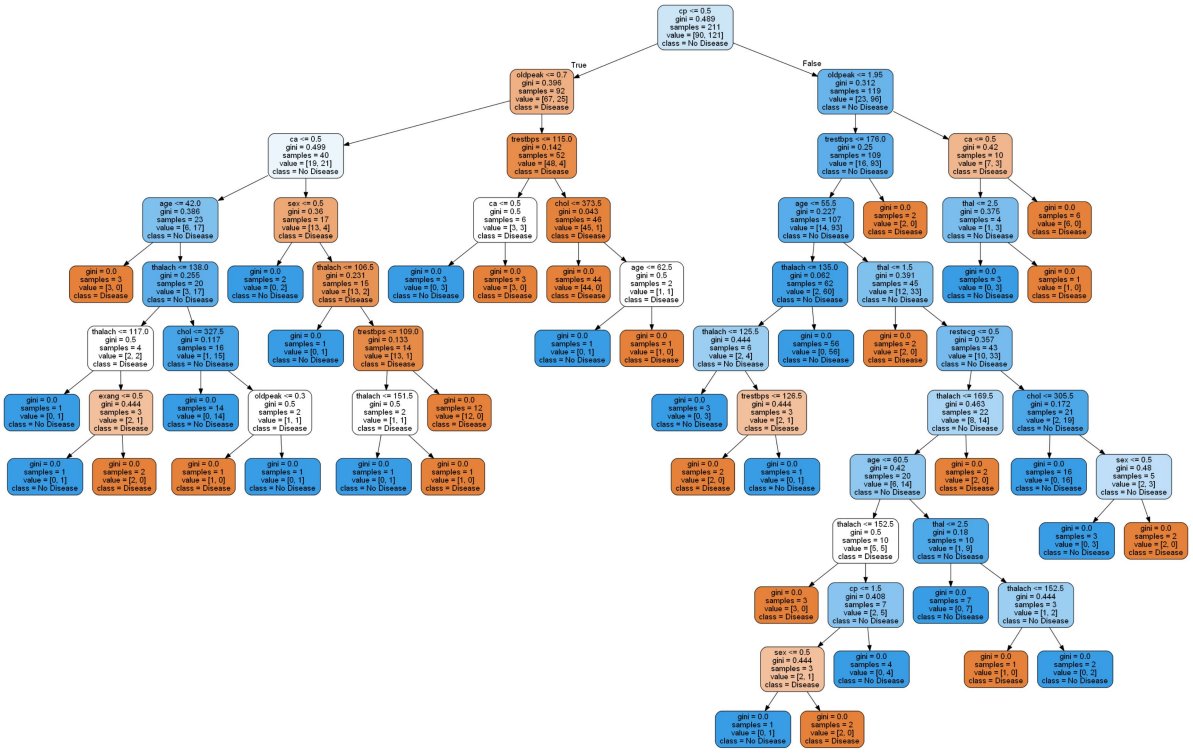
Out[20]:

```
DecisionTreeClassifier(random_state=42)
```

In [21]:

```
gph = get_dt_graph(dt_default)
Image(gph.create_png())
```

Out[21]:



In [22]:

```
evaluate_model(dt_default)
```

```
Train Accuracy : 1.0
Train Confusion Matrix:
[[ 90   0]
 [  0 121]]
--------------------------------------------------
Test Accuracy : 0.7472527472527473
Test Confusion Matrix:
[[35 13]
 [10 33]]
```

Controlling the depth of the tree

In [23]:

```
?DecisionTreeClassifier
```

In [24]:

```
dt_depth = DecisionTreeClassifier(max_depth=3)
dt_depth.fit(X_train, y_train)
```

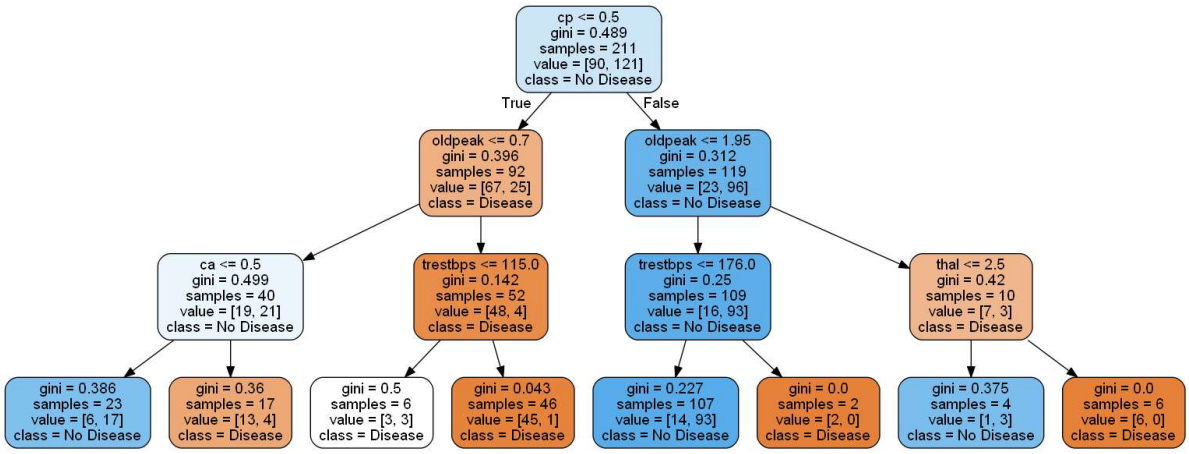
Out[24]:

```
DecisionTreeClassifier(max_depth=3)
```

In [25]:

```
gph = get_dt_graph(dt_depth)
Image(gph.create_png())
```

Out[25]:



In [26]:

```
evaluate_model(dt_depth)
```

```
Train Accuracy : 0.8625592417061612
Train Confusion Matrix:
[[ 69  21]
 [  8 113]]
--------------------------------------------------
Test Accuracy : 0.7692307692307693
Test Confusion Matrix:
[[35 13]
 [ 8 35]]
```

Specifying minimum samples before split

In [27]:

```
dt_min_split = DecisionTreeClassifier(min_samples_split=20)
dt_min_split.fit(X_train, y_train)
```

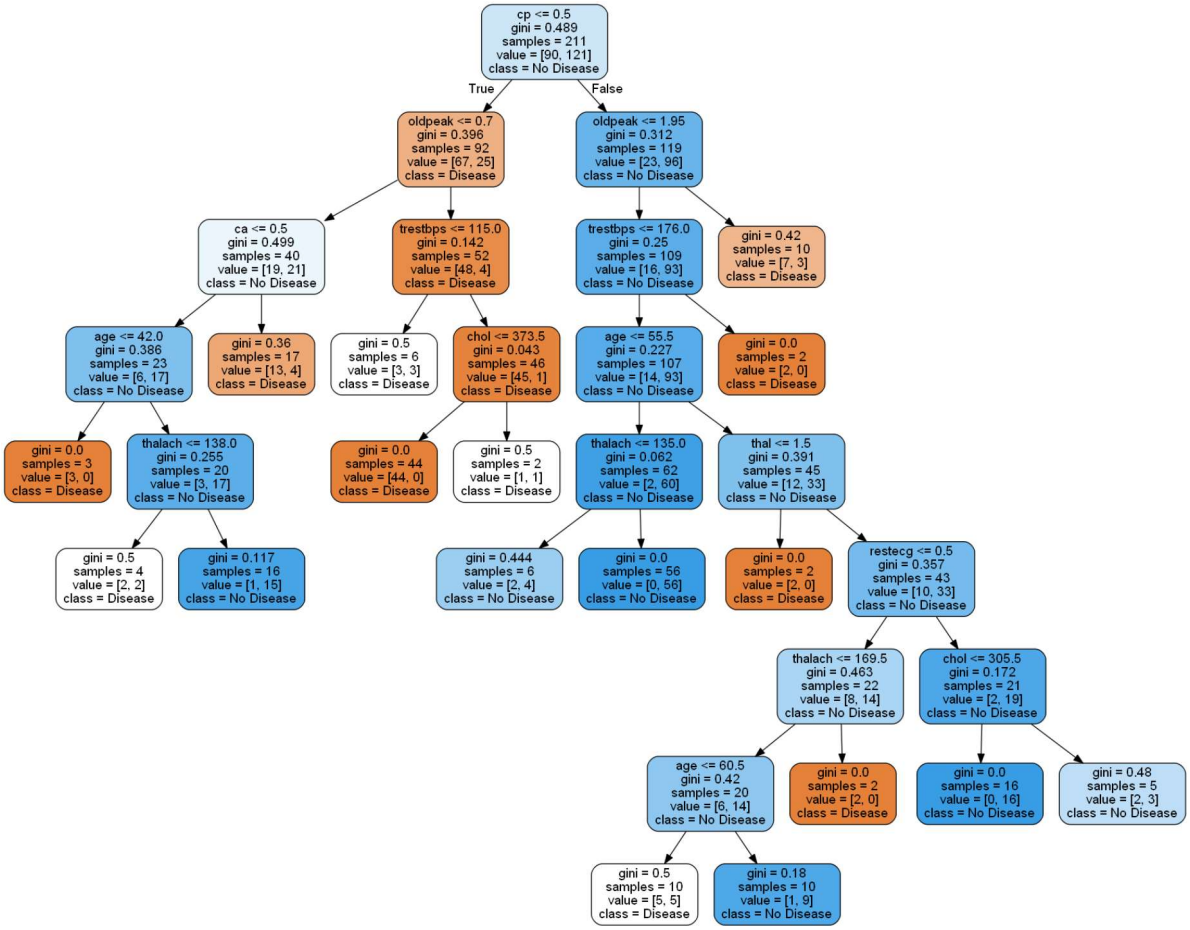
Out[27]:

```
DecisionTreeClassifier(min_samples_split=20)
```

In [28]:

```
gph = get_dt_graph(dt_min_split)
Image(gph.create_png())
```

Out[28]:



In [29]:

```
evaluate_model(dt_min_split)
```

```
Train Accuracy : 0.8862559241706162
Train Confusion Matrix:
[[ 84   6]
 [ 18 103]]
--------------------------------------------------
Test Accuracy : 0.7472527472527473
Test Confusion Matrix:
[[39  9]
 [14 29]]
```

Specifying minimum samples in leaf node

In [30]:

```
dt_min_leaf = DecisionTreeClassifier(min_samples_leaf=20, random_state=42)
dt_min_leaf.fit(X_train, y_train)
```

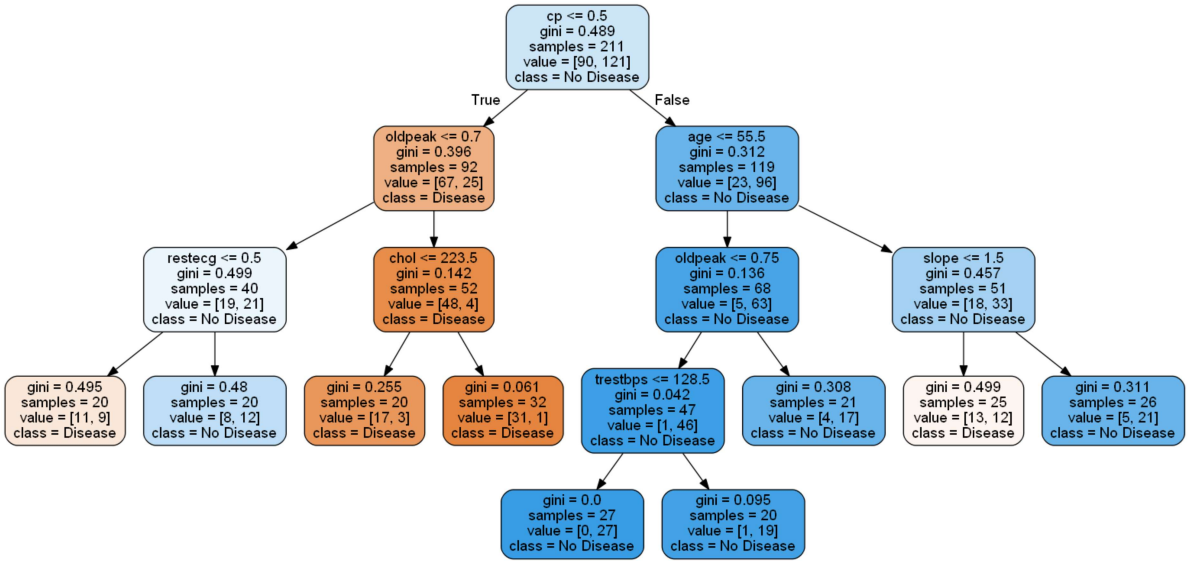
Out[30]:

```
DecisionTreeClassifier(min_samples_leaf=20, random_state=42)
```

In [31]:

```
gph = get_dt_graph(dt_min_leaf)
Image(gph.create_png())
```

Out[31]:



In [32]:

```
evaluate_model(dt_min_leaf)
```

```
Train Accuracy : 0.7962085308056872
Train Confusion Matrix:
[[72 18]
 [25 96]]
--------------------------------------------------
Test Accuracy : 0.7362637362637363
Test Confusion Matrix:
[[40  8]
 [16 27]]
```

Using Entropy instead of Gini

In [33]:

```
dt_min_leaf_entropy = DecisionTreeClassifier(min_samples_leaf=20, random_state=42, criterio
dt_min_leaf_entropy.fit(X_train, y_train)
```

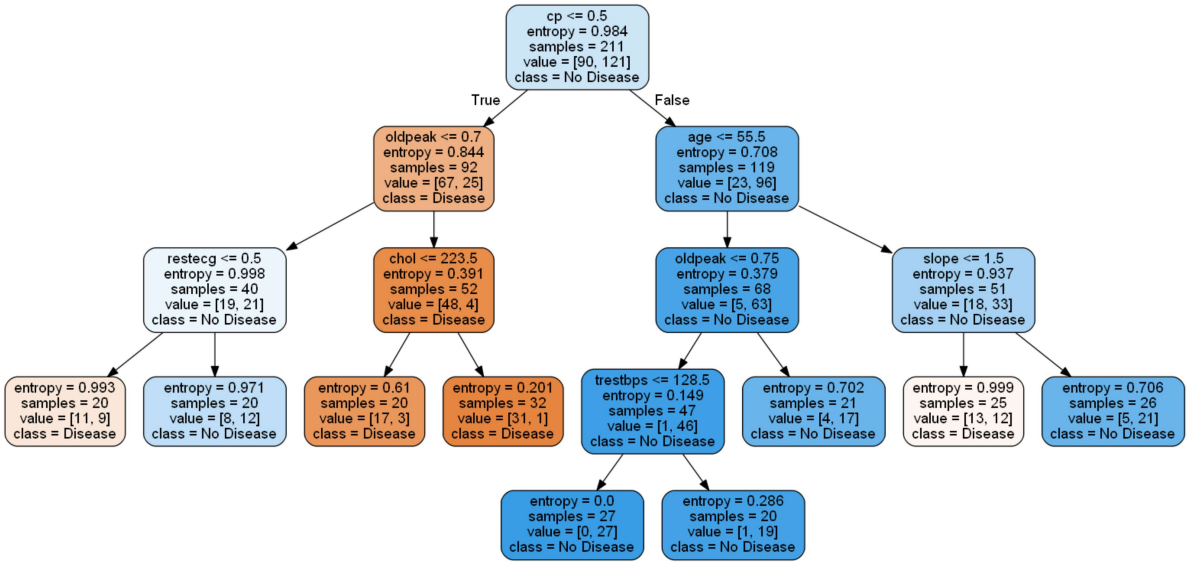
Out[33]:

```
DecisionTreeClassifier(criterion='entropy', min_samples_leaf=20,
                       random_state=42)
```

In [34]:

```
gph = get_dt_graph(dt_min_leaf_entropy)
Image(gph.create_png())
```

Out[34]:



In [35]:

```
evaluate_model(dt_min_leaf_entropy)
```

```
Train Accuracy : 0.7962085308056872
Train Confusion Matrix:
[[72 18]
 [25 96]]
--------------------------------------------------
Test Accuracy : 0.7362637362637363
Test Confusion Matrix:
[[40  8]
 [16 27]]
```

Hyper-parameter tuning

In [36]:

```
dt = DecisionTreeClassifier(random_state=42)
```

In [37]:

```
from sklearn.model_selection import GridSearchCV
```

In [38]:

```
# Create the parameter grid based on the results of random search
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'criterion': ["gini", "entropy"]
}
```

In [39]:

```
# grid_search = GridSearchCV(estimator=dt,
#                            param_grid=params,
#                            cv=4, n_jobs=-1, verbose=1, scoring = "f1")
```

In [40]:

```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=dt,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

In [41]:

```
%%time
grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 50 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.8s

Wall time: 3.24 s

[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:    3.1s finished
```

Out[41]:

```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=42), n_jobs
=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100]},
             scoring='accuracy', verbose=1)
```

In [42]:

```
score_df = pd.DataFrame(grid_search.cv_results_)
score_df.head()
```

Out[42]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_d
0	0.010825	0.001987	0.001948	0.001955	gini	
1	0.007088	0.002869	0.005555	0.001667	gini	
2	0.006181	0.002551	0.005351	0.002118	gini	
3	0.009481	0.002277	0.005200	0.002112	gini	
4	0.008041	0.001705	0.002615	0.001677	gini	

In [43]:

```
score_df.nlargest(5,"mean_test_score")
```

Out[43]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_
40	0.006551	0.000381	0.003360	0.000466	entropy	
45	0.006284	0.000501	0.002678	0.001695	entropy	
35	0.005707	0.003625	0.005207	0.003451	entropy	
6	0.006716	0.000771	0.004701	0.001362	gini	
11	0.005917	0.000627	0.004657	0.001109	gini	

In [44]:

```
grid_search.best_estimator_
```

Out[44]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_leaf=
5,
                       random_state=42)
```

In [45]:

```
dt_best = grid_search.best_estimator_
```

In [46]:

```
evaluate_model(dt_best)
```

```
Train Accuracy : 0.8957345971563981
Train Confusion Matrix:
[[ 82   8]
 [ 14 107]]
--------------------------------------------------
Test Accuracy : 0.7252747252747253
Test Confusion Matrix:
[[39  9]
 [16 27]]
```

In [47]:

```
from sklearn.metrics import classification_report
```

In [48]:

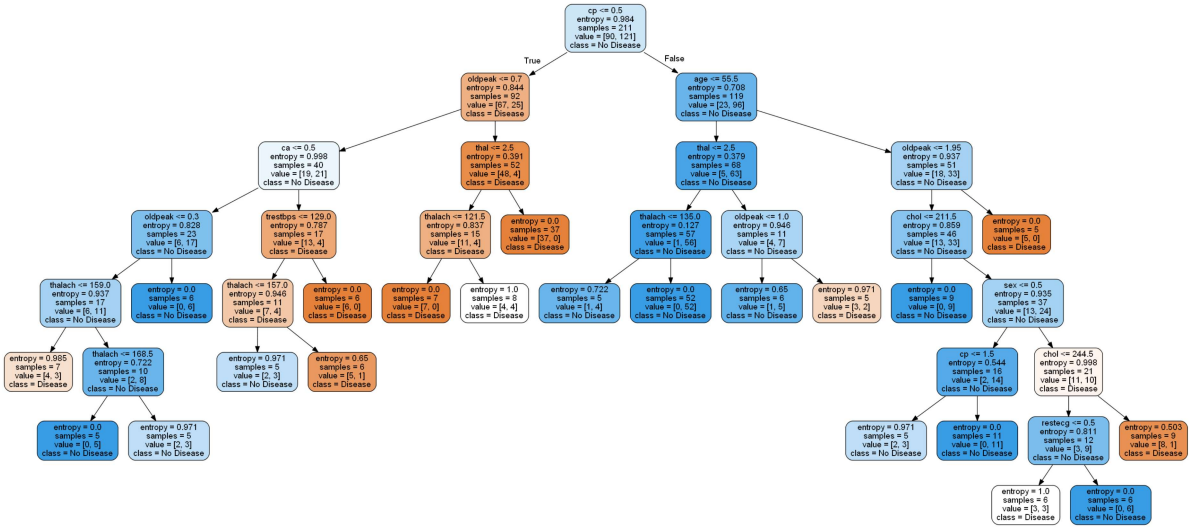
```
print(classification_report(y_test, dt_best.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.71      0.81      0.76        48
           1       0.75      0.63      0.68        43

    accuracy                           0.73        91
   macro avg       0.73      0.72      0.72        91
weighted avg       0.73      0.73      0.72        91
```

In [49]:

```
gph = get_dt_graph(dt_best)
Image(gph.create_png())
```

Out[49]:



Random Forest

In [50]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [51]:

```
rf = RandomForestClassifier(random_state=42, n_estimators=10, max_depth=3)
```

In [52]:

```
rf.fit(X_train, y_train)
```

Out[52]:

```
RandomForestClassifier(max_depth=3, n_estimators=10, random_state=42)
```

In [53]:

```
rf.estimators_[0]
```

Out[53]:

```
DecisionTreeClassifier(max_depth=3, max_features='auto',
                       random_state=1608637542)
```

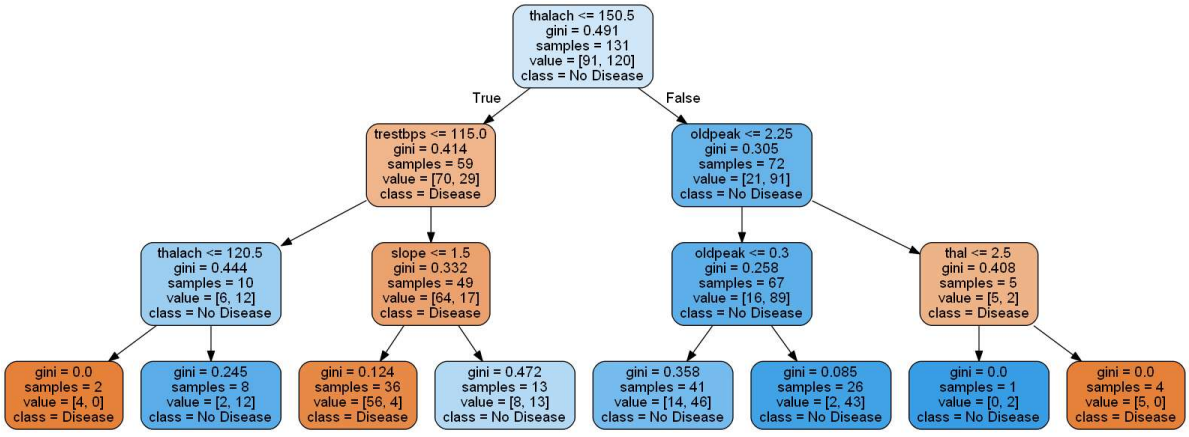
In [54]:

```
sample_tree = rf.estimators_[4]
```

In [55]:

```
gph = get_dt_graph(sample_tree)
Image(gph.create_png(), width=700, height=700)
```

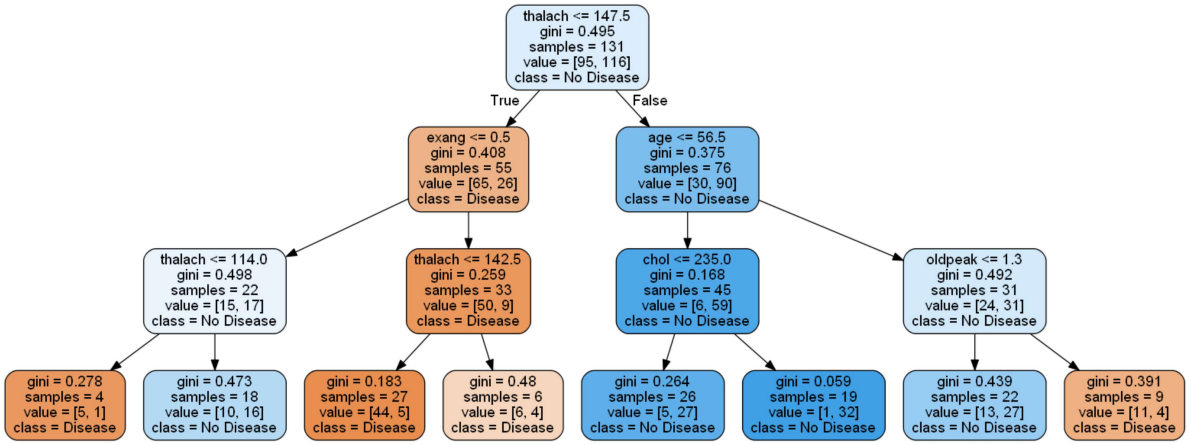
Out[55]:



In [56]:

```
gph = get_dt_graph(rf.estimators_[2])
Image(gph.create_png(), width=700, height=700)
```

Out[56]:



In [57]:

```
evaluate_model(rf)
```

```
Train Accuracy : 0.8530805687203792
Train Confusion Matrix:
[[ 67  23]
 [  8 113]]
--------------------------------------------------
Test Accuracy : 0.8351648351648352
Test Confusion Matrix:
[[34 14]
 [ 1 42]]
```

Grid search for hyper-parameter tuning

In [58]:

```
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

In [59]:

```
# Create the parameter grid based on the results of random search
params = {
    'max_depth': [1, 2, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'max_features': [2,3,4],
    'n_estimators': [10, 30, 50, 100, 200]
}
```

In [60]:

```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=classifier_rf, param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

In [61]:

```
%%time
grid_search.fit(X,y)
```

```
Fitting 4 folds for each of 375 candidates, totalling 1500 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    2.4s
[Parallel(n_jobs=-1)]: Done 264 tasks      | elapsed:   14.5s
[Parallel(n_jobs=-1)]: Done 514 tasks      | elapsed:   29.0s
[Parallel(n_jobs=-1)]: Done 864 tasks      | elapsed:   48.7s
[Parallel(n_jobs=-1)]: Done 1314 tasks      | elapsed:  1.2min

Wall time: 1min 25s

[Parallel(n_jobs=-1)]: Done 1500 out of 1500 | elapsed:  1.4min finished
```

Out[61]:

```
GridSearchCV(cv=4, estimator=RandomForestClassifier(n_jobs=-1, random_state=
42),
             n_jobs=-1,
             param_grid={'max_depth': [1, 2, 5, 10, 20],
                         'max_features': [2, 3, 4],
                         'min_samples_leaf': [5, 10, 20, 50, 100],
                         'n_estimators': [10, 30, 50, 100, 200]},
             scoring='accuracy', verbose=1)
```

In [62]:

```
rf_best = grid_search.best_estimator_
```

In [63]:

```
rf_best
```

Out[63]:

```
RandomForestClassifier(max_depth=10, max_features=4, min_samples_leaf=5,
                       n_estimators=10, n_jobs=-1, random_state=42)
```

In [64]:

```
evaluate_model(rf_best)
```

```
Train Accuracy : 0.8767772511848341
Train Confusion Matrix:
[[ 79  11]
 [ 15 106]]
--------------------------------------------------
Test Accuracy : 0.8901098901098901
Test Confusion Matrix:
[[41  7]
 [ 3 40]]
```

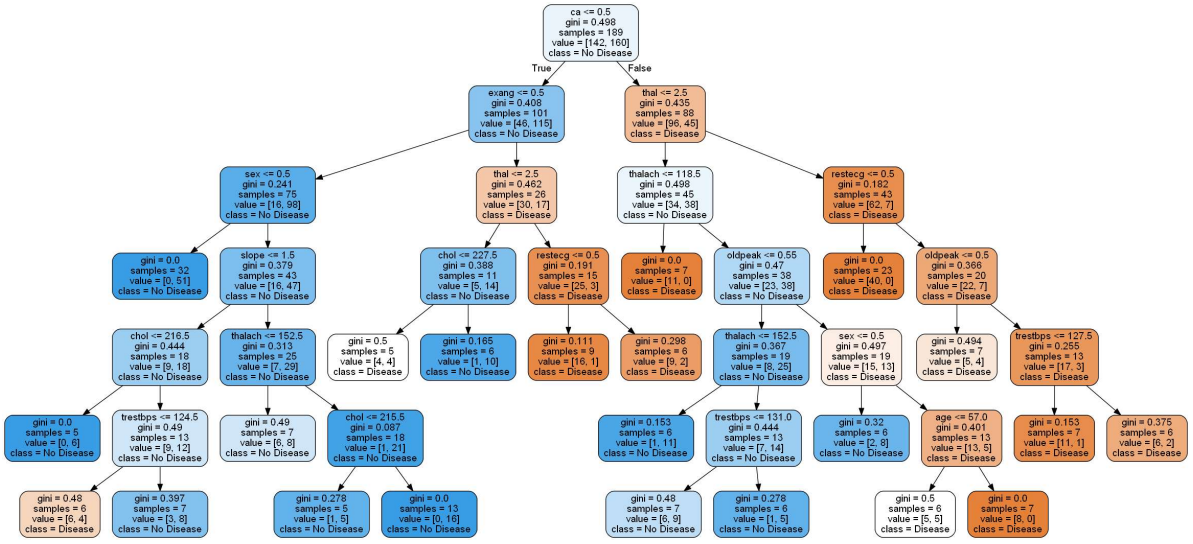
In [65]:

```
sample_tree = rf_best.estimators_[0]
```

In [66]:

```
gph = get_dt_graph(sample_tree)
Image(gph.create_png())
```

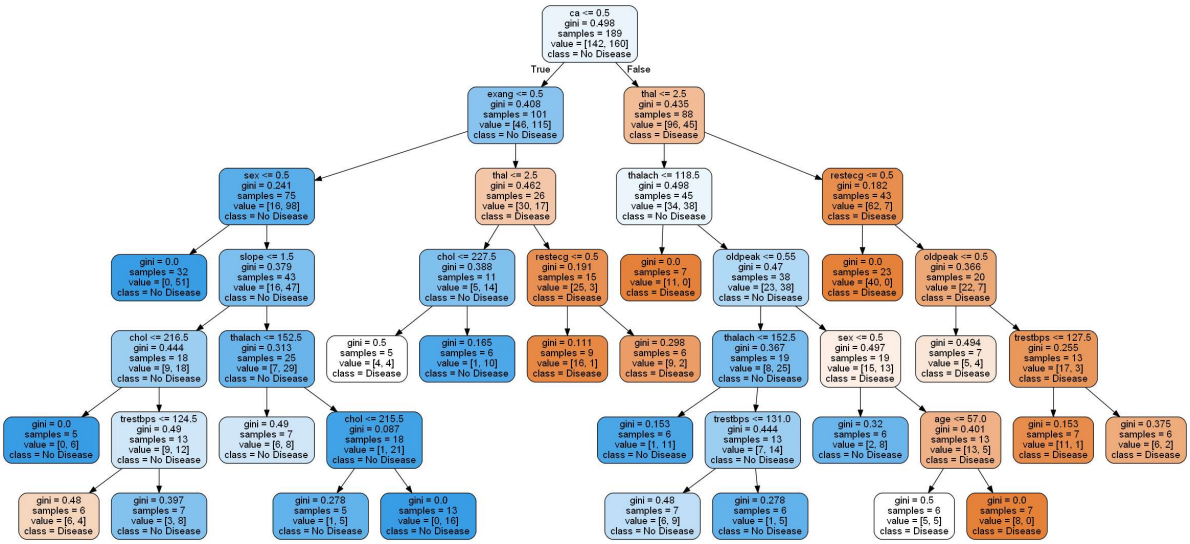
Out[66]:



In [67]:

```
gph = get_dt_graph(rf_best.estimators_[0])
Image(gph.create_png(), height=600, width=600)
```

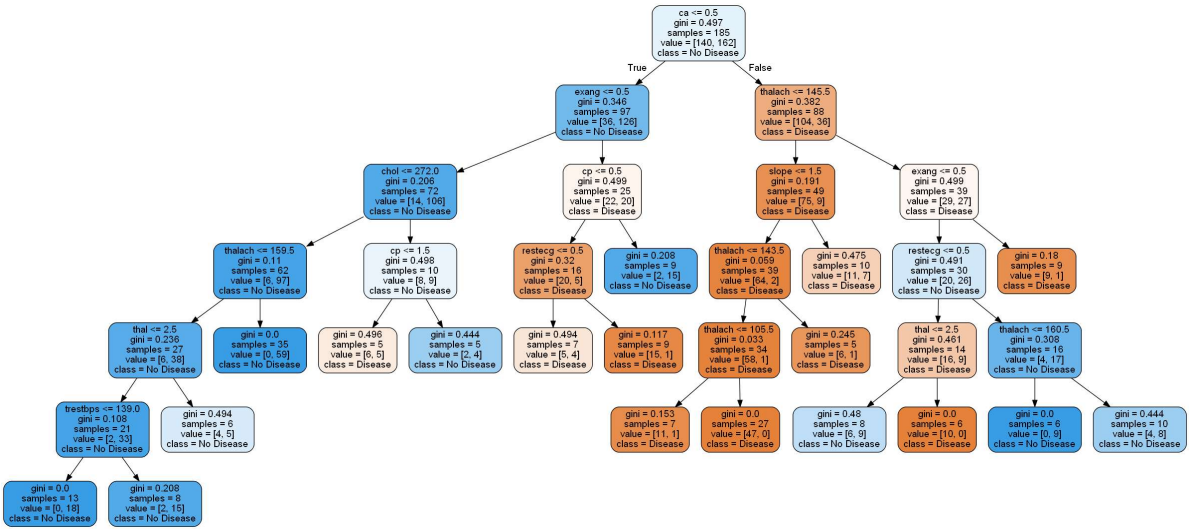
Out[67]:



In [74]:

```
gph = get_dt_graph(rf_best.estimators_[1])
Image(gph.create_png(), height=600, width=600)
```

Out[74]:



Variable importance in RandomForest and Decision trees

In [80]:

```
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=10, max_featur
                       n_estimators=10, oob_score=True)
```

In [81]:

```
classifier_rf.fit(X_train, y_train)
```

Out[81]:

```
RandomForestClassifier(max_depth=10, max_features=4, min_samples_leaf=5,
                       n_estimators=10, n_jobs=-1, oob_score=True,
                       random_state=42)
```

In [83]:

```
classifier_rf.oob_score_
```

Out[83]:

```
0.8104265402843602
```

In [84]:

```
classifier_rf.feature_importances_
```

Out[84]:

```
array([0.04395741, 0.03538842, 0.2480013 , 0.02708451, 0.03040023,
       0.00456046, 0.02340153, 0.20943237, 0.03997417, 0.14859533,
       0.03194401, 0.1265283 , 0.03073195])
```

In [86]:

```
imp_df = pd.DataFrame({
    "Varname": X_train.columns,
    "Imp": classifier_rf.feature_importances_
})
```

In [87]:

```
imp_df.sort_values(by="Imp", ascending=False)
```

Out[87]:

	Varname	Imp
2	cp	0.248001
7	thalach	0.209432
9	oldpeak	0.148595
11	ca	0.126528
0	age	0.043957
8	exang	0.039974
1	sex	0.035388
10	slope	0.031944
12	thal	0.030732
4	chol	0.030400
3	trestbps	0.027085
6	restecg	0.023402
5	fbs	0.004560

In [85]:

```
# Create the parameter grid based on the results of random search
params = {
    'max_depth': [1, 2, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'max_features': [2,3,4],
    'n_estimators': [10, 30, 50, 100, 200]
}
```

In [88]:

```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=classifier_rf, param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

In [89]:

```
%%time
grid_search.fit(X,y)
```

```
Fitting 4 folds for each of 375 candidates, totalling 1500 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    5.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   13.8s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   31.1s
[Parallel(n_jobs=-1)]: Done 784 tasks      | elapsed:   55.3s
[Parallel(n_jobs=-1)]: Done 1234 tasks      | elapsed:  1.5min

Wall time: 1min 48s

[Parallel(n_jobs=-1)]: Done 1500 out of 1500 | elapsed:  1.8min finished
```

Out[89]:

```
GridSearchCV(cv=4,
             estimator=RandomForestClassifier(max_depth=10, max_features=4,
                                              min_samples_leaf=5,
                                              n_estimators=10, n_jobs=-1,
                                              oob_score=True, random_state=4
2),
             n_jobs=-1,
             param_grid={'max_depth': [1, 2, 5, 10, 20],
                         'max_features': [2, 3, 4],
                         'min_samples_leaf': [5, 10, 20, 50, 100],
                         'n_estimators': [10, 30, 50, 100, 200]},
             scoring='accuracy', verbose=1)
```

In [90]:

```
rf_best = grid_search.best_estimator_
rf_best
```

Out[90]:

```
RandomForestClassifier(max_depth=10, max_features=4, min_samples_leaf=5,
                       n_estimators=10, n_jobs=-1, oob_score=True,
                       random_state=42)
```

In [91]:

```
evaluate_model(rf_best)
```

```
Train Accuracy : 0.8767772511848341
Train Confusion Matrix:
[[ 79  11]
 [ 15 106]]
--------------------------------------------------
Test Accuracy : 0.8901098901098901
Test Confusion Matrix:
[[41  7]
 [ 3 40]]
```

In [ ]:

```
```