In [2]:
```
# Importing the required libraries
import pandas as pd, numpy as np
import matplotlib.pyplot as plt, seaborn as sns
%matplotlib inline
```

In [3]:
```
# Reading the csv file and putting it into 'df' object.
df = pd.read_csv('heart_v2.csv')
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['age', 'sex', 'BP', 'cholestrol', 'heart disease'], dtype='object')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 270 entries, 0 to 269
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   age            270 non-null    int64
 1   sex            270 non-null    int64
 2   BP             270 non-null    int64
 3   cholestrol     270 non-null    int64
 4   heart disease  270 non-null    int64
dtypes: int64(5)
memory usage: 10.7 KB
```

In [6]:
```
df.head()
```

Out[6]:

	age	sex	BP	cholestrol	heart disease
0	70	1	130	322	1
1	67	0	115	564	0
2	57	1	124	261	1
3	64	1	128	263	0
4	74	0	120	269	0

In [7]:
```
df['heart disease'].value_counts()
```

Out[7]:
```
0    150
1    120
Name: heart disease, dtype: int64
```

In [8]:
```
# Putting feature variable to X
X = df.drop('heart disease',axis=1)

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

In [9]:
```
from sklearn.model_selection import train_test_split
```

In [10]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, rand
X_train.shape, X_test.shape
```

Out[10]:
```
((189, 4), (81, 4))
```

Building the decision tree.

Use all default parameteres except depth

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

In [11]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [12]:
```
help(DecisionTreeClassifier)
```

```
 |
 |  .. [4] L. Breiman, and A. Cutler, "Random Forests",
 |         https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.ht
m (https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm)
 |
 |  Examples
 |  --------
 |  >>> from sklearn.datasets import load_iris
 |  >>> from sklearn.model_selection import cross_val_score
 |  >>> from sklearn.tree import DecisionTreeClassifier
 |  >>> clf = DecisionTreeClassifier(random_state=0)
 |  >>> iris = load_iris()
 |  >>> cross_val_score(clf, iris.data, iris.target, cv=10)
 |  ...                             # doctest: +SKIP
 |  ...
 |  array([ 1.     ,  0.93...,  0.86...,  0.93...,  0.93...,
 |          0.93...,  0.93...,  1.     ,  0.93...,  1.      ])
 |
 |  Method resolution order:
```

In [13]:
```
dt = DecisionTreeClassifier(max_depth=3)
dt.fit(X_train, y_train)
```

Out[13]:
```
DecisionTreeClassifier(max_depth=3)
```

In [14]:
```
#!pip install six
```

```
Requirement already satisfied: six in c:\users\asus\anaconda3\lib\site-packag
es (1.16.0)
```

In [15]:
```
#!pip install pydotplus
```

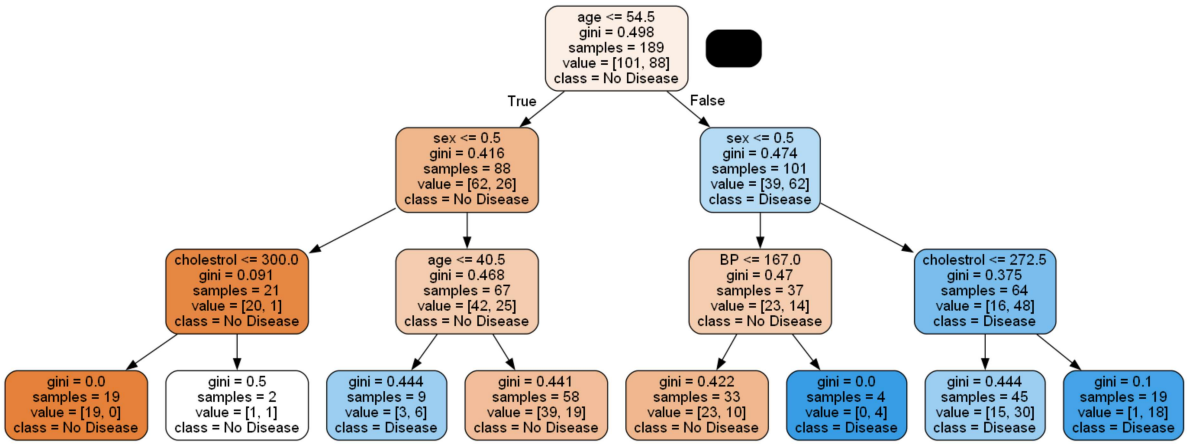
```
Requirement already satisfied: pydotplus in c:\users\asus\anaconda3\lib\site-
packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in c:\users\asus\anaconda3\li
b\site-packages (from pydotplus) (3.0.4)
```

In [24]:
```
#!pip install graphviz
```

In [25]:
```
# Importing required packages for visualization
from IPython.display import Image
from six import StringIO
from sklearn.tree import export_graphviz
import pydotplus, graphviz
```

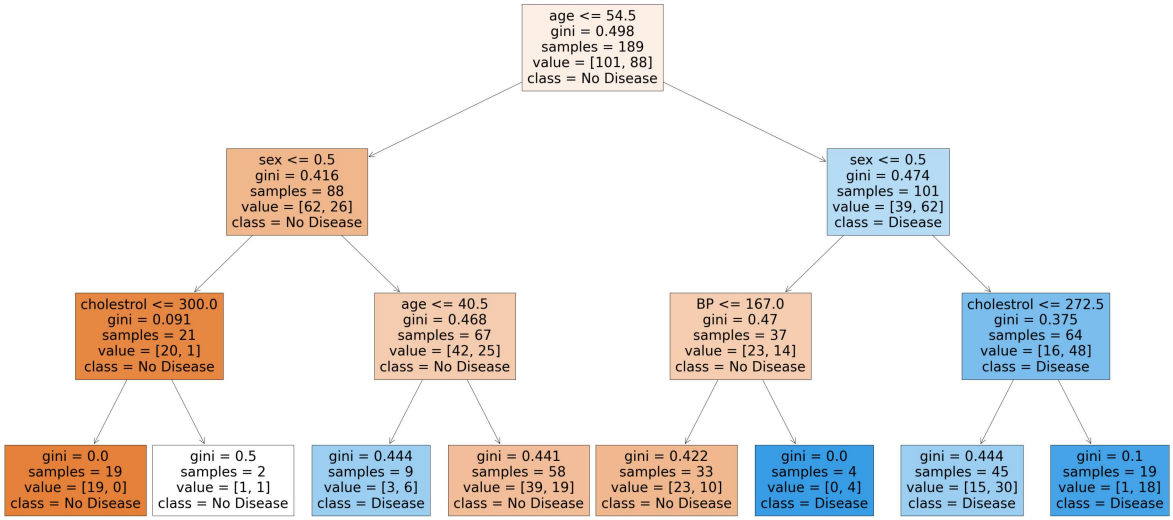
In [26]:
```
# plotting tree with max_depth=3
dot_data = StringIO()

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[26]:



Alternative method for Graphviz

In [27]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(60,30))
plot_tree(dt, feature_names = X.columns,class_names=['No Disease', "Disease"],
```



Evaluating model performance

In [28]:
```
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)
```

In [29]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [30]:
```
print(accuracy_score(y_train, y_train_pred))
confusion_matrix(y_train, y_train_pred)
```

```
0.7407407407407407
```

Out[30]:
```
array([[82, 19],
       [30, 58]], dtype=int64)
```

In [31]:
```
print("Test Performance")
print(accuracy_score(y_test, y_test_pred))
confusion_matrix(y_test, y_test_pred)
```

```
Test Performance
0.6049382716049383
```

Out[31]:
```
array([[35, 14],
       [18, 14]], dtype=int64)
```

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

In [32]:
```
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
```

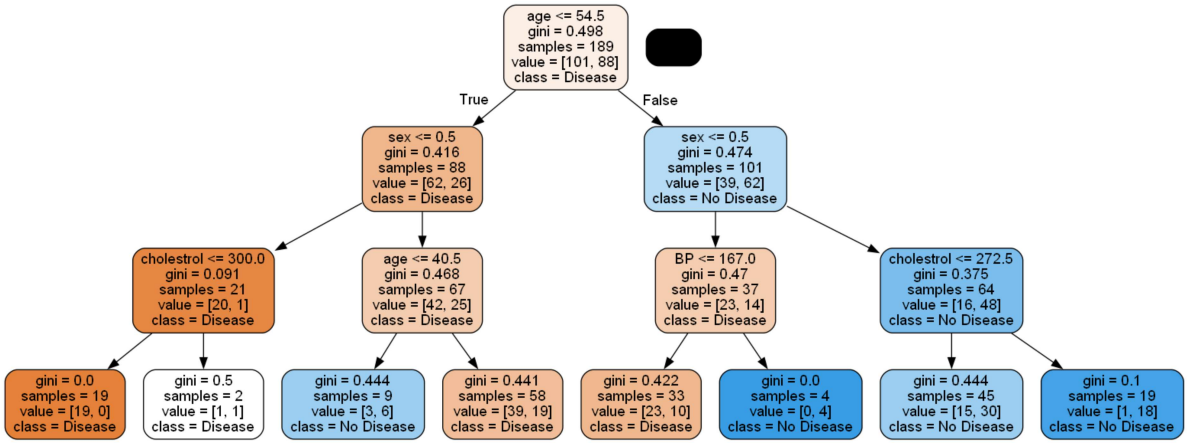
In [33]:
```
def evaluate_model(dt_classifier):
    y_train_pred=dt_classifier.predict(X_train)
    y_test_pred=dt_classifier.predict(X_test)
    print("Train Performance :", accuracy_score(y_train, y_train_pred))
    print("Train Confusion Matrix:")
    print(confusion_matrix(y_train, y_train_pred))
    print("-"*50)
    print("Test Performance :", accuracy_score(y_test, y_test_pred))
    print("Test Confusion Matrix:")
    print(confusion_matrix(y_test, y_test_pred))
```

In [34]:
```
evaluate_model(dt)
```

```
Train Performance : 0.7407407407407407
Train Confusion Matrix:
[[82 19]
 [30 58]]
--------------------------------------------------
Test Performance : 0.6049382716049383
Test Confusion Matrix:
[[35 14]
 [18 14]]
```

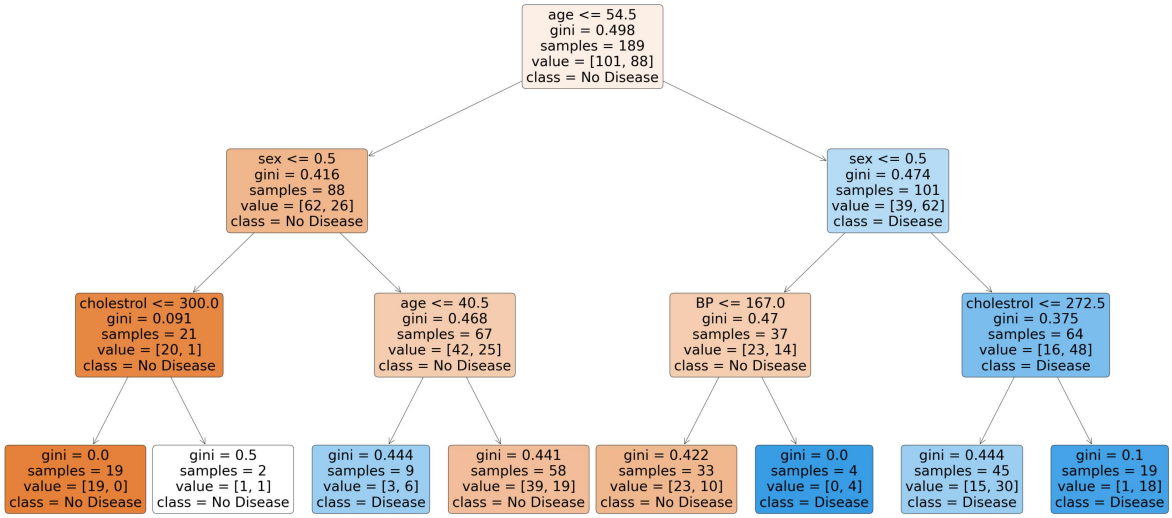
In [35]:
```
gph=get_dt_graph(dt)
Image(gph.create_png())
```

Out[35]:



In [36]:
```
def get_dt_graph2(dt_classifier):
    plt.figure(figsize=(60,30))
    plot_tree(dt_classifier, feature_names = X.columns,class_names=['No Diseas
    return # we do not need return for this, it will show graph even without i
```

In [37]:
```
get_dt_graph2(dt)
```



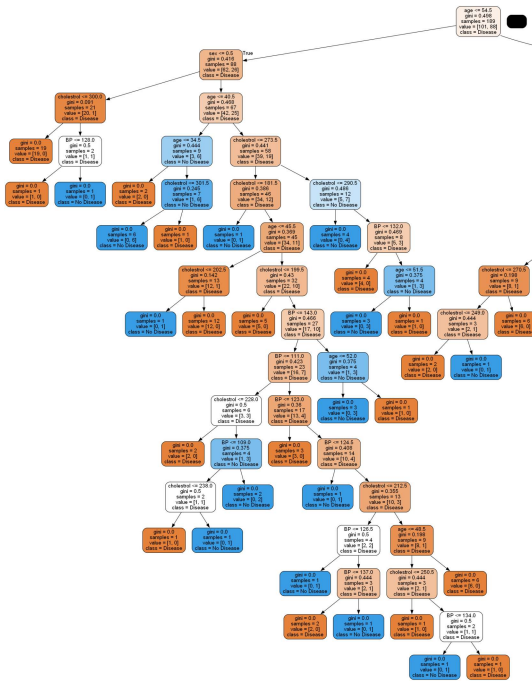
Without setting any hyper-parameters

In [38]:
```
dt_default = DecisionTreeClassifier(random_state=42)
dt_default.fit(X_train, y_train)
```

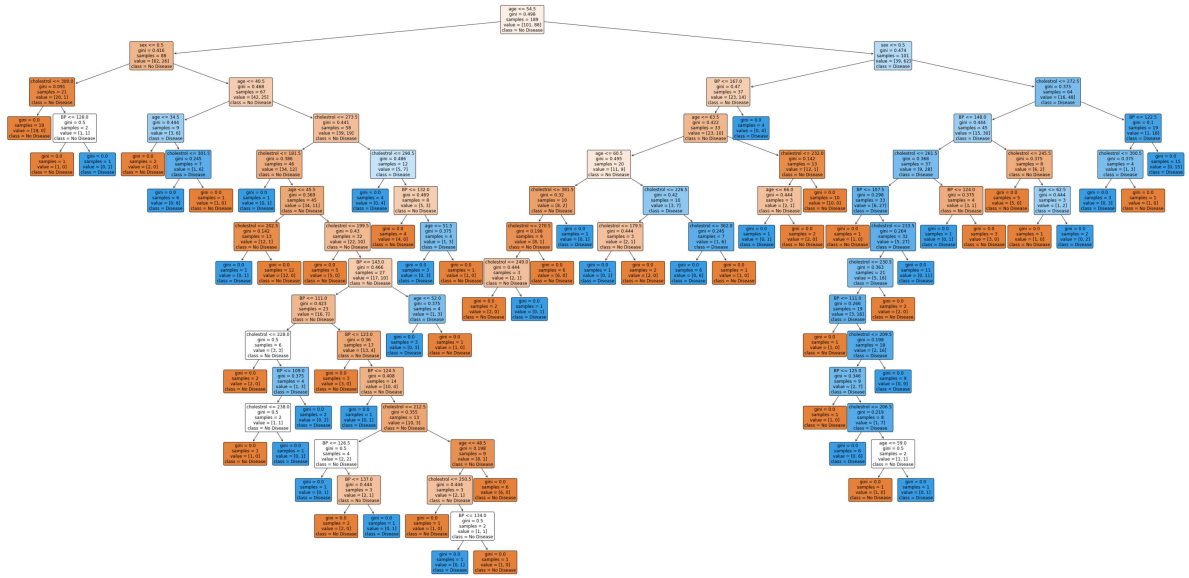
Out[38]:
```
DecisionTreeClassifier(random_state=42)
```

In [39]:
```
gph = get_dt_graph(dt_default)
Image(gph.create_png())
```

Out[39]:



In [40]:
```
get_dt_graph2(dt_default)
```



In [41]:
```
evaluate_model(dt_default)
```

```
Train Performance : 1.0
Train Confusion Matrix:
[[101   0]
 [  0  88]]
--------------------------------------------------
Test Performance : 0.6296296296296297
Test Confusion Matrix:
[[31 18]
 [12 20]]
```

Controlling the depth of the tree

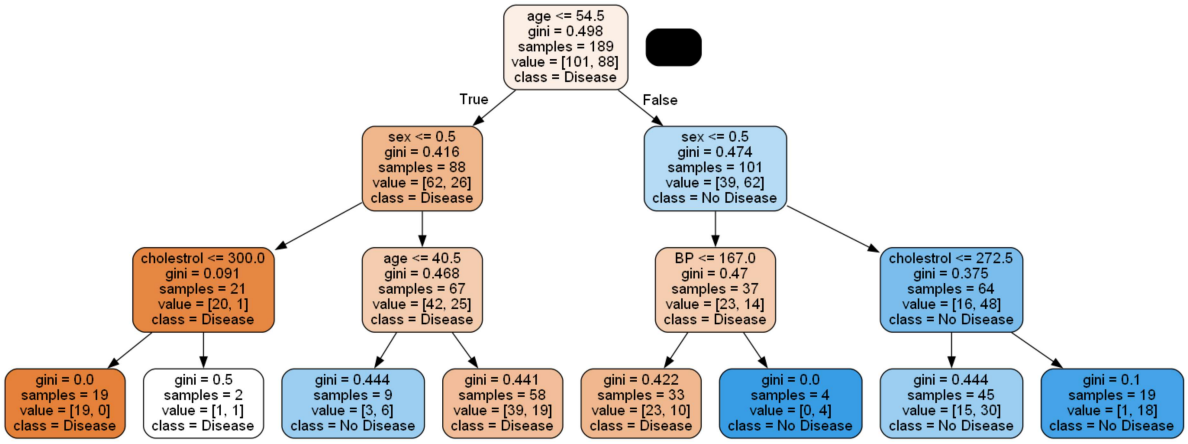
In [42]:
```
?DecisionTreeClassifier
```

In [43]:
```
dt_depth = DecisionTreeClassifier(max_depth=3,random_state=42)
dt_depth.fit(X_train, y_train)
```

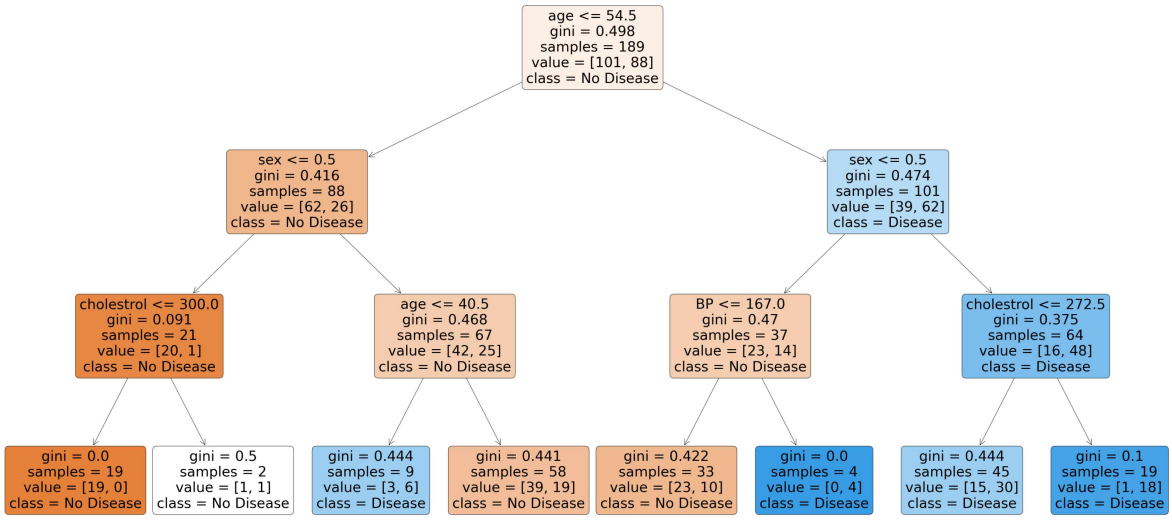
Out[43]:
```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [44]:
```
gph = get_dt_graph(dt_depth)
Image(gph.create_png())
```

Out[44]:



In [45]:
```
get_dt_graph2(dt_depth)
```



In [46]:
```
evaluate_model(dt_depth)
```

```
Train Performance : 0.7407407407407407
Train Confusion Matrix:
[[82 19]
 [30 58]]
--------------------------------------------------
Test Performance : 0.6049382716049383
Test Confusion Matrix:
[[35 14]
 [18 14]]
```

Specifying minimum samples before split

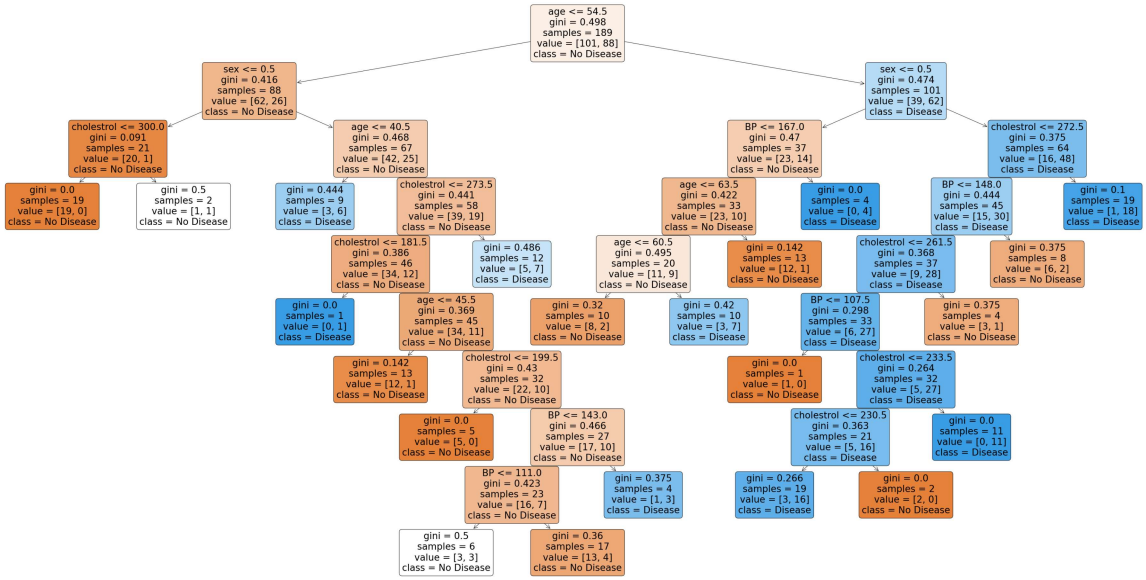
In [47]:
```
dt_min_split = DecisionTreeClassifier(min_samples_split=20)
dt_min_split.fit(X_train, y_train)
```

Out[47]:
```
DecisionTreeClassifier(min_samples_split=20)
```

In [48]:
```
gph = get_dt_graph(dt_min_split)
Image(gph.create_png())
```

Out[48]:

In [49]:
```
get_dt_graph2(dt_min_split)
```



In [50]:
```
evaluate_model(dt_min_split)
```

```
Train Performance : 0.8359788359788359
Train Confusion Matrix:
[[85 16]
 [15 73]]
--------------------------------------------------
Test Performance : 0.6419753086419753
Test Confusion Matrix:
[[32 17]
 [12 20]]
```

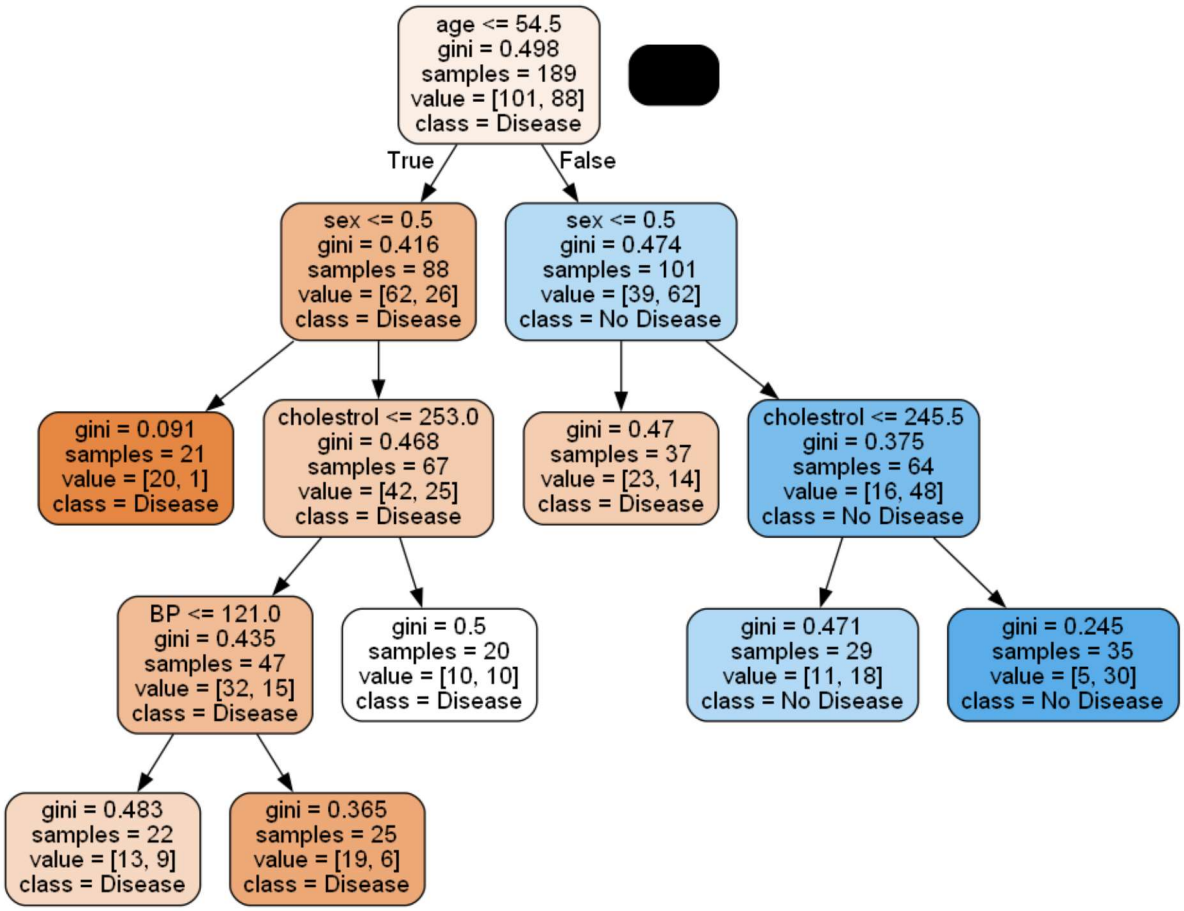
Specifying minimum samples in leaf node

In [51]:
```
dt_min_leaf = DecisionTreeClassifier(min_samples_leaf=20, random_state=42)
dt_min_leaf.fit(X_train, y_train)
```

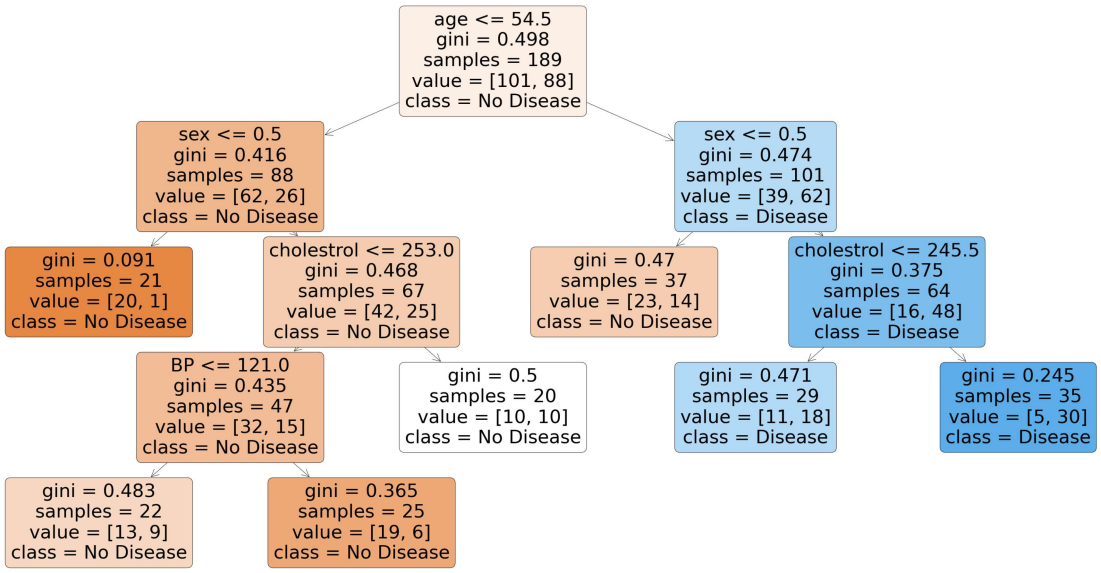
Out[51]:
```
DecisionTreeClassifier(min_samples_leaf=20, random_state=42)
```

In [52]:
```
gph = get_dt_graph(dt_min_leaf)
Image(gph.create_png())
```

Out[52]:



In [53]:
```
get_dt_graph2(dt_min_leaf)
```



In [54]:
```
evaluate_model(dt_min_leaf)
```

```
Train Performance : 0.7037037037037037
Train Confusion Matrix:
[[85 16]
 [40 48]]
--------------------------------------------------
Test Performance : 0.6419753086419753
Test Confusion Matrix:
[[38 11]
 [18 14]]
```

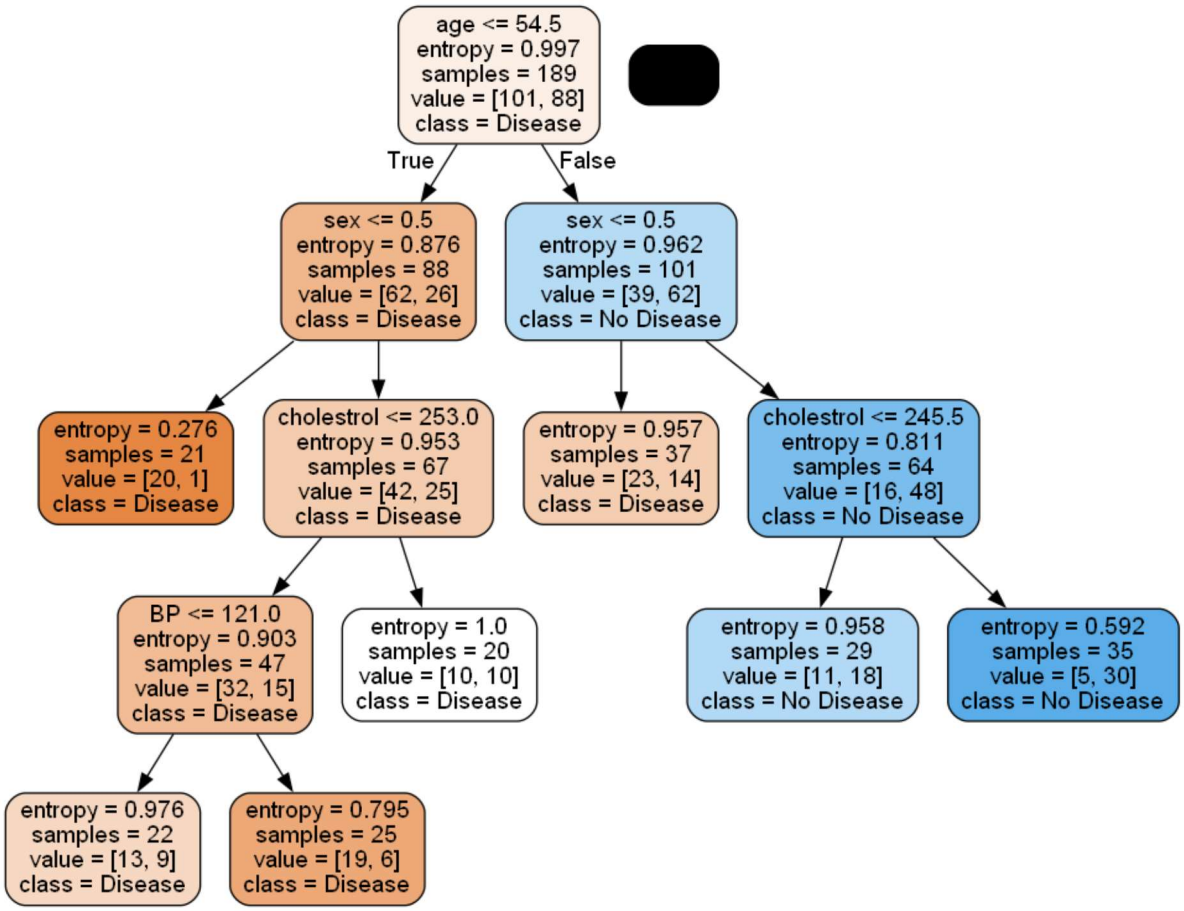
Using Entropy instead of Gini

In [55]:
```
dt_min_leaf_entropy = DecisionTreeClassifier(min_samples_leaf=20, random_state
dt_min_leaf_entropy.fit(X_train, y_train)
```

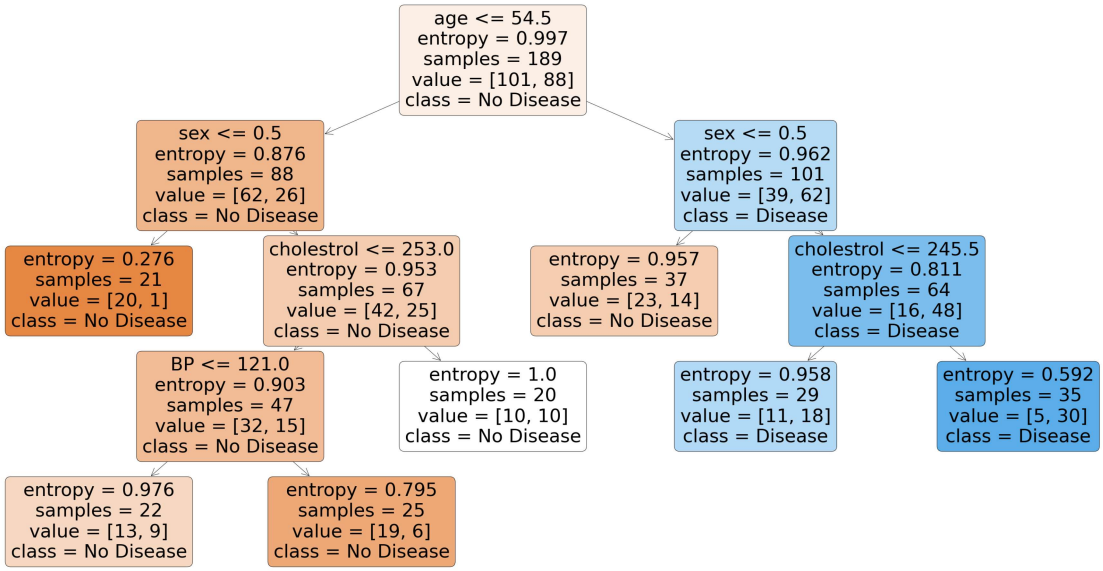
Out[55]:
```
DecisionTreeClassifier(criterion='entropy', min_samples_leaf=20,
                       random_state=42)
```

In [56]:
```
gph = get_dt_graph(dt_min_leaf_entropy)
Image(gph.create_png())
```

Out[56]:



In [57]:
```
get_dt_graph2(dt_min_leaf_entropy)
```



In [58]:
```
evaluate_model(dt_min_leaf_entropy)
```

```
Train Performance : 0.7037037037037037
Train Confusion Matrix:
[[85 16]
 [40 48]]
--------------------------------------------------
Test Performance : 0.6419753086419753
Test Confusion Matrix:
[[38 11]
 [18 14]]
```

Hyper-parameter tuning using GridSearchCV

In [59]:
```
dt = DecisionTreeClassifier(random_state=42)
```

In [60]:
```
from sklearn.model_selection import GridSearchCV
```

In [52]:
```
?GridSearchCV
```

In [53]:
```
range(1, 40)
```

Out[53]:
```
range(1, 40)
```

In [61]:
```
# Create the parameter grid based on the results of random search
params = {
    'max_depth': [*range(3,10,3)],
    'min_samples_split':[*range(5,10,3)],
    'min_samples_leaf': [*range(3,10,3)],
    'criterion': ["gini", "entropy"]
}
```

In [62]:
```
# grid_search = GridSearchCV(estimator=dt,
#                            param_grid=params,
#                            cv=4, n_jobs=-1, verbose=1, scoring = "f1")
```

In [63]:
```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=dt,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

In [64]:
```
%%time
grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 36 candidates, totalling 144 fits
CPU times: total: 406 ms
Wall time: 6.26 s
```

Out[64]:
```
GridSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=42), n_jobs=
-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 6, 9], 'min_samples_leaf': [3, 6,
9],
                         'min_samples_split': [5, 8]},
             scoring='accuracy', verbose=1)
```

In [65]:
```
grid_search.cv_results_
```

Out[65]:
```
{'mean_fit_time': array([0.00390583, 0.00781298, 0.00390631, 0.00781274,
0.00419605,
        0.0039078 , 0.00390613, 0.00390607, 0.00390702, 0.00390702,
        0.00390559, 0.        , 0.00781202, 0.        , 0.00390631,
        0.00390631, 0.00390673, 0.        , 0.00390625, 0.00614679,
        0.00353891, 0.00155389, 0.0039075 , 0.0039075 , 0.00781316,
        0.        , 0.00390679, 0.00781333, 0.00781286, 0.00781286,
        0.00390607, 0.0117166 , 0.00781178, 0.00390625, 0.00781202,
        0.        ]),
 'std_fit_time': array([0.0067651 , 0.00781298, 0.00676593, 0.00781274, 0.
00726777,
        0.00676851, 0.00676562, 0.00676551, 0.00676717, 0.00676717,
        0.00676469, 0.        , 0.00781202, 0.        , 0.00676593,
        0.00676593, 0.00676665, 0.        , 0.00676582, 0.00577309,
        0.00409662, 0.00172325, 0.00676799, 0.00676799, 0.00781316,
        0.        , 0.00676675, 0.00781333, 0.00781286, 0.00781286,
        0.00676551, 0.00676458, 0.00781178, 0.00676582, 0.00781202,
        0.        ]),
 'mean_score_time': array([0.        , 0.        , 0.00390542, 0.        ,
```

In [66]:
```
score_df = pd.DataFrame(grid_search.cv_results_)
score_df.head()
```

Out[66]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_de
0	0.003906	0.006765	0.000000	0.000000	gini	
1	0.007813	0.007813	0.000000	0.000000	gini	
2	0.003906	0.006766	0.003905	0.006764	gini	
3	0.007813	0.007813	0.000000	0.000000	gini	
4	0.004196	0.007268	0.004196	0.007268	gini	

In [67]:
```
score_df.nlargest(5,"mean_test_score")
```

Out[67]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_d
20	0.003539	0.004097	0.005461	0.006052	entropy	
21	0.001554	0.001723	0.000252	0.000436	entropy	
18	0.003906	0.006766	0.003906	0.006766	entropy	
19	0.006147	0.005773	0.006402	0.005330	entropy	
2	0.003906	0.006766	0.003905	0.006764	gini	

In [68]:
```
grid_search.best_estimator_
```

Out[68]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=6,
                       min_samples_split=5, random_state=42)
```

In [69]:
```
dt_best = grid_search.best_estimator_
```

In [70]:
```
evaluate_model(dt_best)
```

```
Train Performance : 0.7195767195767195
Train Confusion Matrix:
[[82 19]
 [34 54]]
--------------------------------------------------
Test Performance : 0.6172839506172839
Test Confusion Matrix:
[[36 13]
 [18 14]]
```

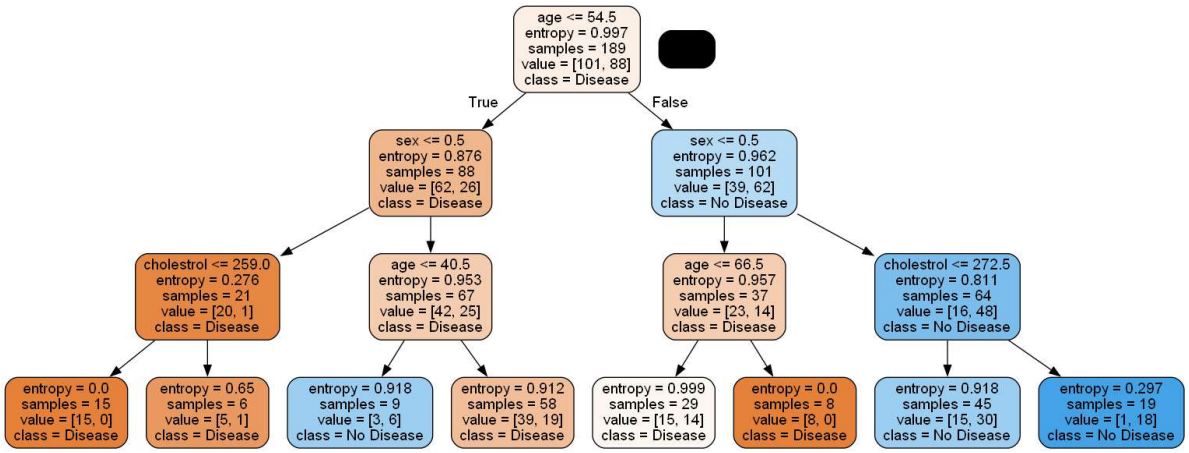
In [71]:
```
from sklearn.metrics import classification_report
```

In [72]:
```
print(classification_report(y_test, dt_best.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.67      0.73      0.70        49
           1       0.52      0.44      0.47        32

    accuracy                           0.62        81
   macro avg       0.59      0.59      0.59        81
weighted avg       0.61      0.62      0.61        81
```

In [73]:
```
gph = get_dt_graph(dt_best)
Image(gph.create_png())
```

Out[73]:



Using Random Forest

In [74]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [75]:
```
?RandomForestClassifier
```

In [76]:
```
df = pd.read_csv('heart_v2.csv')
# Putting feature variable to X
X = df.drop('heart disease',axis=1)

# Putting response variable to y
y = df['heart disease']
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, rand
X_train.shape, X_test.shape
```

Out[76]:
```
((189, 4), (81, 4))
```

In [77]:
```
rf = RandomForestClassifier(max_depth=5, min_samples_leaf=3, min_samples_split
```

In [78]:
```
rf.fit(X_train, y_train)
```

Out[78]:
```
RandomForestClassifier(max_depth=5, min_samples_leaf=3, min_samples_split=5,
                       n_jobs=-1, random_state=42)
```

In [79]:
```
rf.base_estimator_
```

Out[79]:
```
DecisionTreeClassifier()
```

In [80]:
```
rf.n_features_
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:103: F
utureWarning: Attribute `n_features_` was deprecated in version 1.0 and will
be removed in 1.2. Use `n_features_in_` instead.
  warnings.warn(msg, category=FutureWarning)
```

Out[80]:
```
4
```

In [81]:
```
rf.feature_names_in_
```

Out[81]:
```
array(['age', 'sex', 'BP', 'cholestrol'], dtype=object)
```

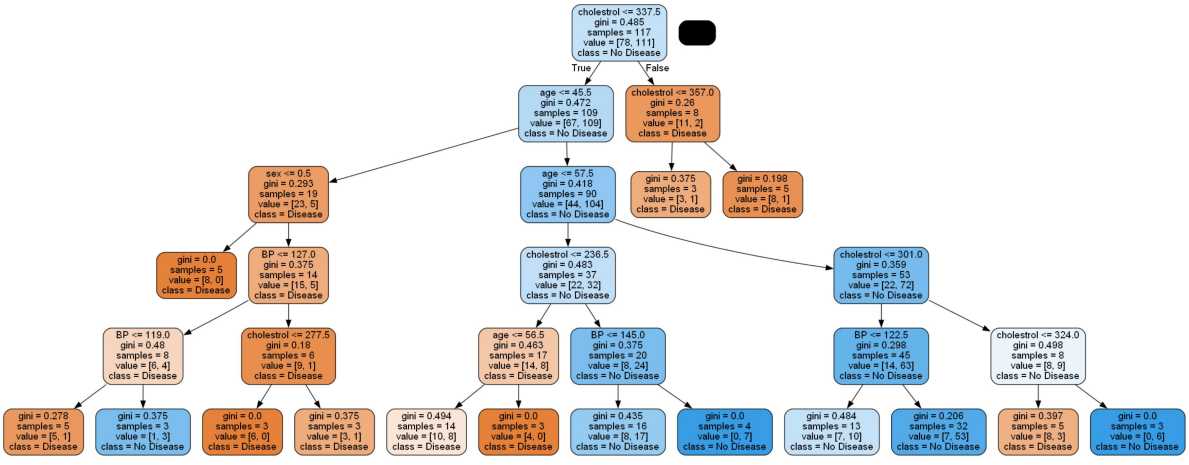
In [82]:
```
rf.n_outputs_
```

Out[82]:
```
1
```

In [83]:
```
sample_tree = rf.estimators_[20]
```

In [84]:
```
gph = get_dt_graph(sample_tree)
Image(gph.create_png(), width=700, height=700)
```

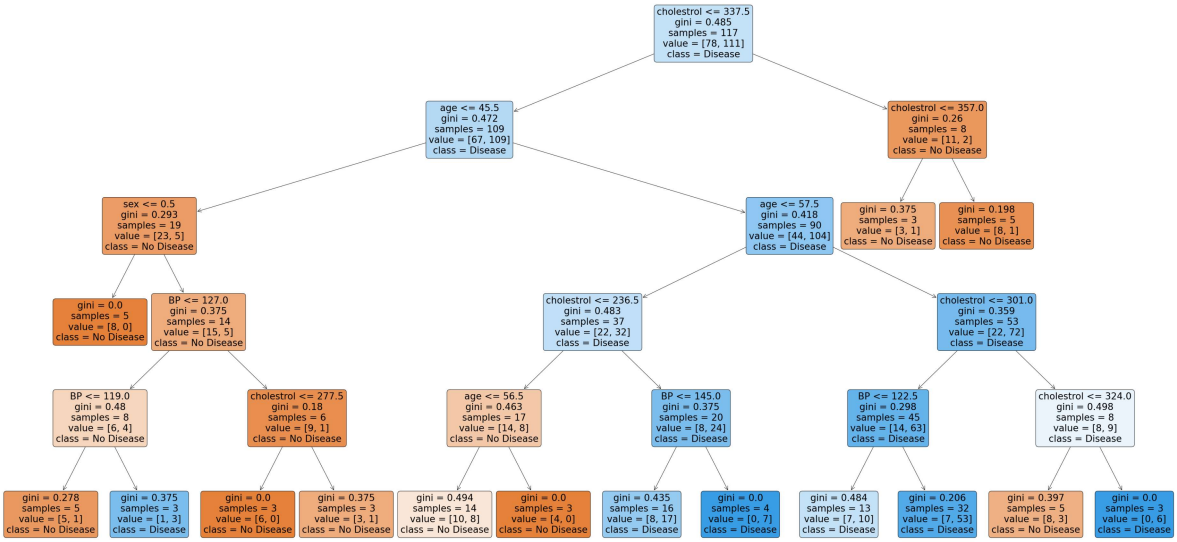
Out[84]:



In [85]:
```
evaluate_model(rf)
```

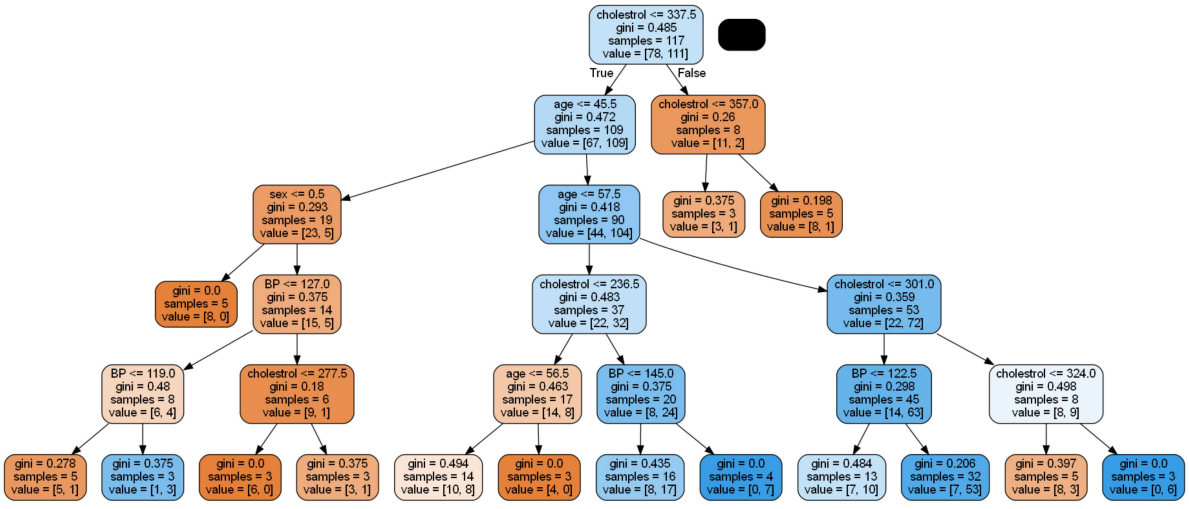
```
Train Performance : 0.8306878306878307
Train Confusion Matrix:
[[87 14]
 [18 70]]
--------------------------------------------------
Test Performance : 0.6790123456790124
Test Confusion Matrix:
[[37 12]
 [14 18]]
```

In [86]:
```
gph = get_dt_graph2(sample_tree)
```



In [277]:
```
dot_data = StringIO()
export_graphviz(sample_tree, out_file=dot_data, filled=True, rounded=True,
                feature_names=X_train.columns)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[277]:



Grid search for hyper-parameter tuning

In [87]:
```
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

In [88]:
```
# 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 [89]:
```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=classifier_rf, param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

In [90]:
```
%%time
grid_search.fit(X,y)
```

```
Fitting 4 folds for each of 375 candidates, totalling 1500 fits
CPU times: total: 4.55 s
Wall time: 53.8 s
```

Out[90]:
```
GridSearchCV(cv=4, estimator=RandomForestClassifier(n_jobs=-1, 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 [91]:
```
rf_best = grid_search.best_estimator_
```

In [92]:
```
rf_best
```

Out[92]:
```
RandomForestClassifier(max_depth=5, max_features=3, min_samples_leaf=5,
                       n_estimators=30, n_jobs=-1, random_state=42)
```

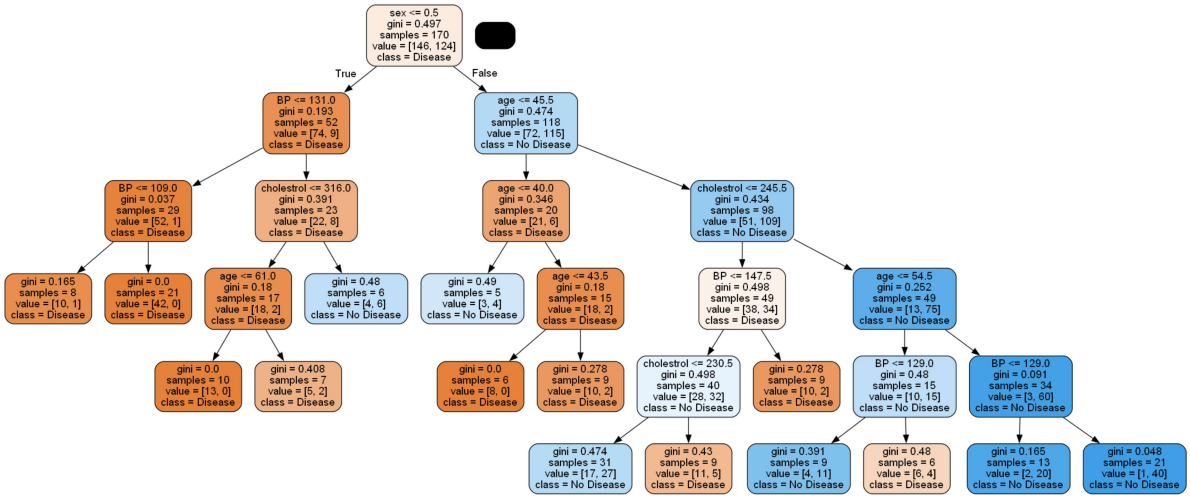
In [93]:
```
evaluate_model(rf_best)
```

```
Train Performance : 0.8042328042328042
Train Confusion Matrix:
[[87 14]
 [23 65]]
--------------------------------------------------
Test Performance : 0.8024691358024691
Test Confusion Matrix:
[[42  7]
 [ 9 23]]
```

In [94]:
```
sample_tree = rf_best.estimators_[0]
```

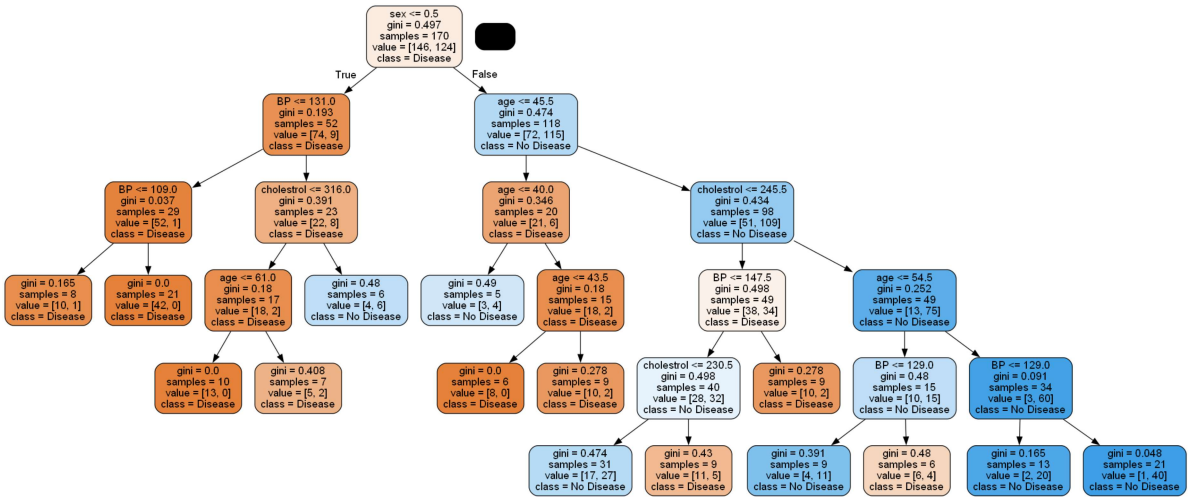
In [95]:
```
gph = get_dt_graph(sample_tree)
Image(gph.create_png())
```

Out[95]:



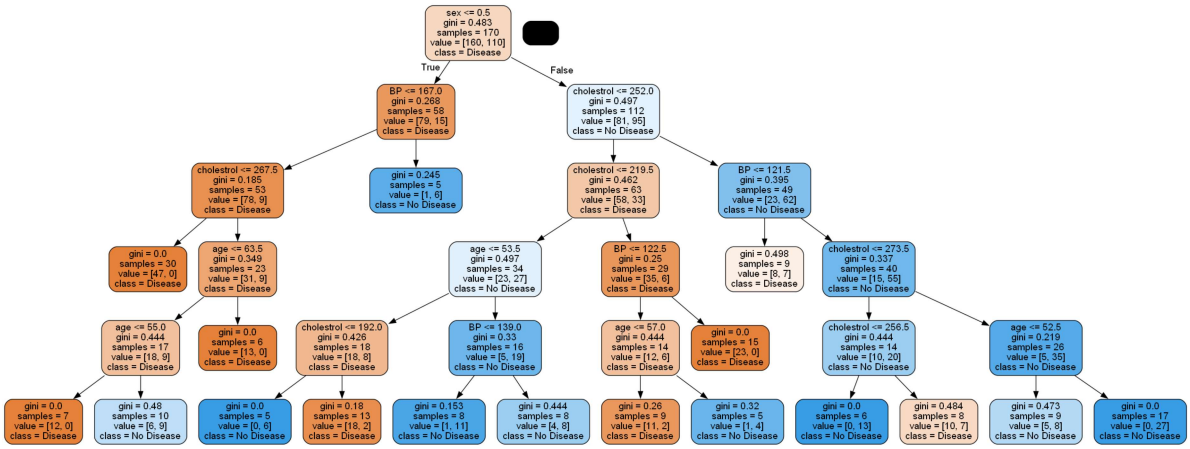
In [96]:
```
gph = get_dt_graph(rf_best.estimators_[0])
Image(gph.create_png(), height=600, width=600)
```

Out[96]:



In [97]:
```
gph = get_dt_graph(rf_best.estimators_[10])
Image(gph.create_png(), height=600, width=600)
```

Out[97]:



Variable importance in RandomForest and Decision trees

In [98]:
```
classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5
```

In [99]:
```
classifier_rf.fit(X_train, y_train)
```

Out[99]:
```
RandomForestClassifier(max_depth=5, n_jobs=-1, oob_score=True, random_state=4
2)
```

In [100]:
```
classifier_rf.feature_importances_
```

Out[100]:
```
array([0.37539743, 0.13780788, 0.20834585, 0.27844883])
```

In [101]:
```
imp_df = pd.DataFrame({
    "Varname": X_train.columns,
    "Imp": classifier_rf.feature_importances_
})
```

In [102]:
```
imp_df.sort_values(by="Imp", ascending=False)
```

Out[102]:

	Varname	Imp
0	age	0.375397
3	cholestrol	0.278449
2	BP	0.208346
1	sex	0.137808

In [ ]:
```

```

In [ ]:

Decision Tree Another example

In [67]:
```
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import KFold
```

In [68]:
```
# read training data
bank_train = pd.read_csv("bank_train.csv")
# read test data
bank_test = pd.read_csv("bank_train.csv")
```

In [69]:
```
bank_train.head(3)
```

Out[69]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor
0	49	284	1	999	0	1.4	93.918	-42.7	4.
1	58	374	1	999	0	1.4	93.918	-42.7	4.
2	36	529	1	999	0	1.4	93.444	-36.1	4.

3 rows × 22 columns

In [70]:
```
# build the model

# # train the model
print(bank_train.columns)
x_train = bank_train.drop(['purchased', 'id'], axis=1)
y_train = bank_train[['purchased']]
```

```
Index(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed',
       'purchased', 'id', 'job', 'marital', 'education', 'default', 'housin
g',
       'loan', 'contact', 'month', 'day_of_week', 'poutcome'],
      dtype='object')
```

In [71]:
```
y_train.value_counts()
```

Out[71]:
```
purchased
0            2944
1             351
dtype: int64
```

In [72]:
```
# Hyperparameter tuning: maxdepth
# specify number of folds for k-fold CV
n_folds = 5
```

In [73]:
```
# parameters to build the model on
parameters = {'max_depth': range(1, 50)}
```

In [74]:
```
# instantiate the model
dtree = DecisionTreeClassifier(random_state = 100)
```

In [75]:
```
# fit tree on training data
tree = GridSearchCV(dtree, parameters,
                    cv=n_folds, n_jobs=-1, verbose=1,
                   scoring="accuracy",
                   return_train_score=True)
tree.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
```

Out[75]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=100),
             n_jobs=-1, param_grid={'max_depth': range(1, 50)},
             return_train_score=True, scoring='accuracy', verbose=1)
```

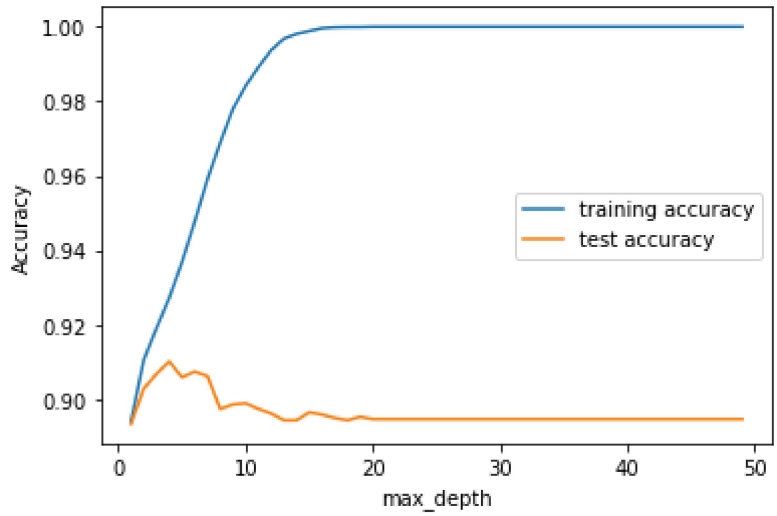
In [76]:
```
# scores of GridSearch CV
scores = tree.cv_results_
pd.DataFrame(scores).head()
```

Out[76]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params
0	0.006201	0.007595	0.003101	0.006201	1	{'max_depth': 1}
1	0.006248	0.007653	0.000000	0.000000	2	{'max_depth': 2}
2	0.003124	0.006249	0.003124	0.006249	3	{'max_depth': 3}
3	0.006249	0.007653	0.000000	0.000000	4	{'max_depth': 4}
4	0.006248	0.007653	0.003124	0.006249	5	{'max_depth': 5}

5 rows × 21 columns

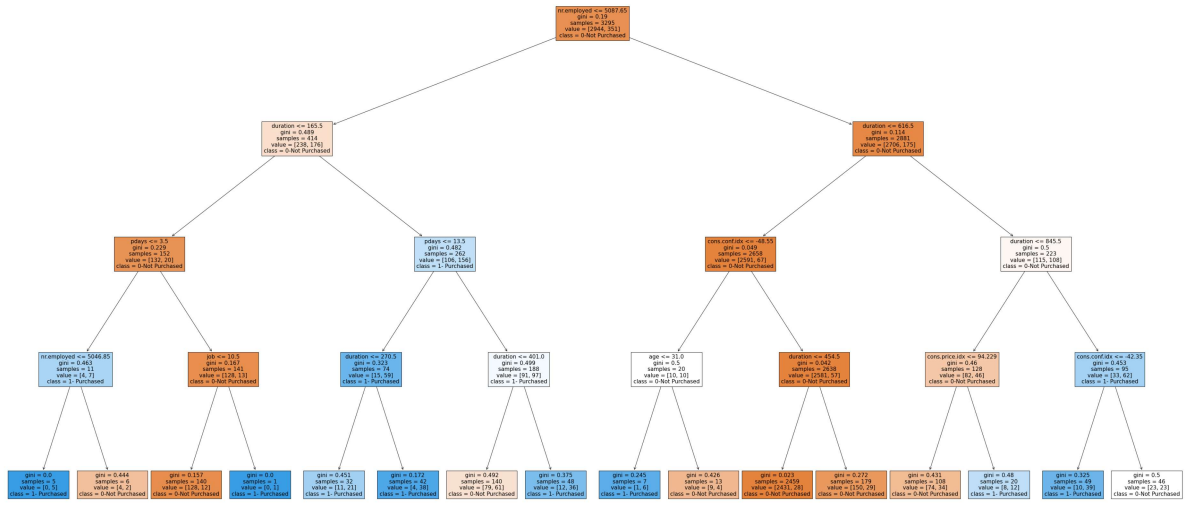
In [77]:
```
# plotting accuracies with max_depth
plt.figure()
plt.plot(scores["param_max_depth"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_max_depth"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [78]:
```
# create DT with optimal max_depth
best_tree = DecisionTreeClassifier(max_depth=4)
best_tree.fit(x_train, y_train)
```

Out[78]:
```
DecisionTreeClassifier(max_depth=4)
```

In [79]:
```
from sklearn.tree import plot_tree
plt.figure(figsize=(60,30))
plot_tree(best_tree, feature_names = x_train.columns,class_names=['0-Not Purch
```



In [80]:
```
bank_test.head()
```

Out[80]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor
0	49	284	1	999	0	1.4	93.918	-42.7	4.9
1	58	374	1	999	0	1.4	93.918	-42.7	4.9
2	36	529	1	999	0	1.4	93.444	-36.1	4.9
3	47	216	1	999	0	1.4	94.465	-41.8	4.9
4	38	191	1	999	0	-1.8	92.893	-46.2	1.3

5 rows × 22 columns

In [81]:
```
# make predictions
predictions = best_tree.predict(bank_test.drop(['purchased', 'id'], axis=1))
print(predictions[:15])
```

```
[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
```

In [82]:
```
# write columns id, predictions into the output file
d = pd.DataFrame({'id': bank_test['id'], 'bank_predicted': predictions})
print("\n", "d", "\n", d.head())
# # write the output
#d.to_csv('/code/output/bank_predictions.csv', sep=",")
```

```
 d
      id  bank_predicted
0   648               0
1  2186               0
2  1645               0
3  2751               0
4   104               0
```

In [83]:
```
d.head(15)
```

Out[83]:

	id	bank_predicted
0	648	0
1	2186	0
2	1645	0
3	2751	0
4	104	0
5	2677	0
6	4026	0
7	643	0
8	3595	1
9	2938	0
10	3703	0
11	240	0
12	2453	0
13	134	0
14	2572	0

Random Forest Another example

In [292]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics, preprocessing
from sklearn.ensemble import RandomForestClassifier

# read training data
bank_train = pd.read_csv("bank_train.csv")

# read test data
bank_test = pd.read_csv("bank_train.csv")

print(bank_train.head())
print(bank_test.head())

############################
### WRITE YOUR CODE HERE ###
############################

# Build the model
# Create a random forest object rf (use default hyperparameters)
rf = RandomForestClassifier()

# Train the model

# Create x_train: Drop the columns 'purchased' (target) and 'id'
print(bank_train.columns)
x_train = bank_train.drop(['purchased', 'id'], axis=1)

# Create y_train
y_train = bank_train['purchased']

# Fit the model
rf.fit(x_train, y_train)

#  Make predictions using test data
print(bank_test.head())

# remember to drop 'id' from the test dataset
predictions = rf.predict(bank_test.drop(['purchased', 'id'], axis=1))
print(predictions[:15])

# Write the columns 'id' and 'predictions' into the output file
d = pd.DataFrame({'id': bank_test['id'], 'bank_predicted': predictions})

# Write the output
#d.to_csv('/code/output/bank_predictions.csv', sep=",")
```

```
   age  duration  campaign  pdays  previous  emp.var.rate  cons.price.idx  \
0   49       284         1    999         0           1.4          93.918
1   58       374         1    999         0           1.4          93.918
2   36       529         1    999         0           1.4          93.444
3   47       216         1    999         0           1.4          94.465
4   38       191         1    999         0          -1.8          92.893

   cons.conf.idx  euribor3m  nr.employed  ...  job  marital  education  \
0          -42.7      4.957       5228.1  ...    1        0          7
1          -42.7      4.963       5228.1  ...    5        1          0
2          -36.1      4.965       5228.1  ...    0        3          6
3          -41.8      4.961       5228.1  ...    4        1          0
4          -46.2      1.313       5099.1  ...    0        2          7

   default  housing  loan  contact  month  day_of_week  poutcome
0        1        2     0        0      3            4         1
1        1        2     0        0      3            4         1
2        0        2     0        0      1            4         1
3        0        0     2        1      4            2         1
4        0        2     0        0      6            0         1

[5 rows x 22 columns]
   age  duration  campaign  pdays  previous  emp.var.rate  cons.price.idx  \
0   49       284         1    999         0           1.4          93.918
1   58       374         1    999         0           1.4          93.918
2   36       529         1    999         0           1.4          93.444
3   47       216         1    999         0           1.4          94.465
4   38       191         1    999         0          -1.8          92.893

   cons.conf.idx  euribor3m  nr.employed  ...  job  marital  education  \
0          -42.7      4.957       5228.1  ...    1        0          7
1          -42.7      4.963       5228.1  ...    5        1          0
2          -36.1      4.965       5228.1  ...    0        3          6
3          -41.8      4.961       5228.1  ...    4        1          0
4          -46.2      1.313       5099.1  ...    0        2          7

   default  housing  loan  contact  month  day_of_week  poutcome
0        1        2     0        0      3            4         1
1        1        2     0        0      3            4         1
2        0        2     0        0      1            4         1
3        0        0     2        1      4            2         1
4        0        2     0        0      6            0         1

[5 rows x 22 columns]
Index(['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
       'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.employed',
       'purchased', 'id', 'job', 'marital', 'education', 'default', 'housin
g',
       'loan', 'contact', 'month', 'day_of_week', 'poutcome'],
      dtype='object')
   age  duration  campaign  pdays  previous  emp.var.rate  cons.price.idx  \
0   49       284         1    999         0           1.4          93.918
1   58       374         1    999         0           1.4          93.918
2   36       529         1    999         0           1.4          93.444
3   47       216         1    999         0           1.4          94.465
4   38       191         1    999         0          -1.8          92.893
```

```
   cons.conf.idx  euribor3m  nr.employed  ...  job  marital  education  \
0          -42.7      4.957       5228.1  ...    1        0          7
1          -42.7      4.963       5228.1  ...    5        1          0
2          -36.1      4.965       5228.1  ...    0        3          6
3          -41.8      4.961       5228.1  ...    4        1          0
4          -46.2      1.313       5099.1  ...    0        2          7

   default  housing  loan  contact  month  day_of_week  poutcome
0        1        2     0        0      3            4         1
1        1        2     0        0      3            4         1
2        0        2     0        0      1            4         1
3        0        0     2        1      4            2         1
4        0        2     0        0      6            0         1

[5 rows x 22 columns]
[0 0 1 0 0 0 0 0 0 0 0 0 0 1]
```

In [290]:
```
from sklearn.metrics import r2_score
r2_score(bank_train['purchased'], predictions)
```

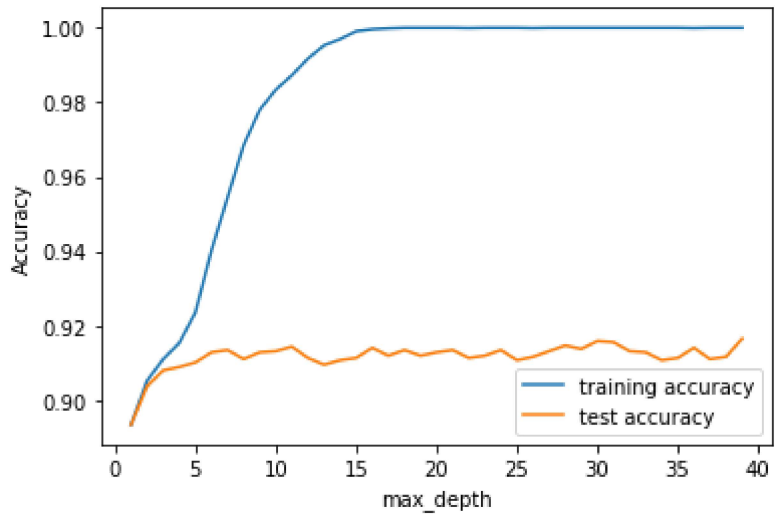
Out[290]:
```
0.7512832125603865
```

In [291]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics, preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

# read training data
bank_train = pd.read_csv("bank_train.csv")

# read test data
bank_test = pd.read_csv("bank_train.csv")

print(bank_train.head())
print(bank_test.head())

##########################
## WRITE YOUR CODE HERE ##
##########################

# create x_train and y_train
x_train = bank_train.drop(['purchased', 'id'], axis=1)
y_train = bank_train['purchased']

#######################################################
## Implement GridSearchCV to find optimal max_depth
#######################################################

# specify number of folds for k-fold CV
n_folds = 5

# specify range of the hyperparameter max_depth
parameters = {'max_depth': range(1, 40)}

# instantiate the model
rf = RandomForestClassifier()

# fit tree on training data
rf = GridSearchCV(rf, parameters,
                    cv=n_folds,
                   scoring="accuracy",
                   return_train_score = True)

# fit the rf model
rf.fit(x_train, y_train)

# store scores/results of GridSearch CV in a df
scores = rf.cv_results_
print(pd.DataFrame(scores).head())

#######################################################
## Plot mean_train_score and mean_test_score (accuracies) on the x-axis
# and param_max_depth on the y-axis
#######################################################

plt.figure()
```

```
plt.plot(scores["param_max_depth"],
         scores["mean_train_score"],
         label="training accuracy")
plt.plot(scores["param_max_depth"],
         scores["mean_test_score"],
         label="test accuracy")
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
#plt.savefig('/code/output/max_depth.png')


# from the plot, observe the optimal value of max_depth
# and store in max_depth_optimal
max_depth_optimal = 9

########################################
# Build the model with optimal max_depth
########################################
rf = RandomForestClassifier(max_depth = max_depth_optimal)
rf.fit(x_train, y_train)

## Make predictions
predictions = rf.predict(bank_test.drop(['purchased', 'id'], axis=1))
print(predictions[:15])

# Write columns id, predictions into the output file
d = pd.DataFrame({'id': bank_test['id'], 'bank_predicted': predictions})

# Write the output
#d.to_csv('/code/output/bank_predictions.csv', sep=",")
```

```
   age  duration  campaign  pdays  previous  emp.var.rate  cons.price.idx  \
0   49       284         1    999         0           1.4          93.918
1   58       374         1    999         0           1.4          93.918
2   36       529         1    999         0           1.4          93.444
3   47       216         1    999         0           1.4          94.465
4   38       191         1    999         0          -1.8          92.893

   cons.conf.idx  euribor3m  nr.employed  ...  job  marital  education  \
0          -42.7      4.957       5228.1  ...    1        0          7
1          -42.7      4.963       5228.1  ...    5        1          0
2          -36.1      4.965       5228.1  ...    0        3          6
3          -41.8      4.961       5228.1  ...    4        1          0
4          -46.2      1.313       5099.1  ...    0        2          7

   default  housing  loan  contact  month  day_of_week  poutcome
0        1        2     0        0      3            4         1
1        1        2     0        0      3            4         1
2        0        2     0        0      1            4         1
3        0        0     2        1      4            2         1
4        0        2     0        0      6            0         1

[5 rows x 22 columns]
   age  duration  campaign  pdays  previous  emp.var.rate  cons.price.idx  \
0   49       284         1    999         0           1.4          93.918
1   58       374         1    999         0           1.4          93.918
2   36       529         1    999         0           1.4          93.444
3   47       216         1    999         0           1.4          94.465
4   38       191         1    999         0          -1.8          92.893

   cons.conf.idx  euribor3m  nr.employed  ...  job  marital  education  \
0          -42.7      4.957       5228.1  ...    1        0          7
1          -42.7      4.963       5228.1  ...    5        1          0
2          -36.1      4.965       5228.1  ...    0        3          6
3          -41.8      4.961       5228.1  ...    4        1          0
4          -46.2      1.313       5099.1  ...    0        2          7

   default  housing  loan  contact  month  day_of_week  poutcome
0        1        2     0        0      3            4         1
1        1        2     0        0      3            4         1
2        0        2     0        0      1            4         1
3        0        0     2        1      4            2         1
4        0        2     0        0      6            0         1

[5 rows x 22 columns]
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       0.108544      0.004389         0.013792        0.003647
1       0.139122      0.012527         0.011283        0.006117
2       0.134981      0.007053         0.007360        0.007644
3       0.143653      0.006117         0.009371        0.007651
4       0.158192      0.009283         0.009203        0.007064

  param_max_depth            params  split0_test_score  split1_test_score  \
0               1  {'max_depth': 1}           0.893778           0.893778
1               2  {'max_depth': 2}           0.899848           0.905918
2               3  {'max_depth': 3}           0.904401           0.901366
3               4  {'max_depth': 4}           0.904401           0.904401
4               5  {'max_depth': 5}           0.904401           0.902883
```

```
   split2_test_score  split3_test_score  ...  mean_test_score  std_test_score
\
0           0.893778           0.893778  ...         0.893475        0.000607
1           0.905918           0.899848  ...         0.903794        0.003269
2           0.908953           0.915023  ...         0.908042        0.004760
3           0.905918           0.918058  ...         0.908953        0.005343
4           0.907436           0.922610  ...         0.910167        0.007207

   rank_test_score  split0_train_score  split1_train_score  \
0               39            0.893399            0.893399
1               38            0.908194            0.905539
2               37            0.912747            0.912367
3               36            0.917299            0.915781
4               34            0.926404            0.924127

   split2_train_score  split3_train_score  split4_train_score  \
0            0.893399            0.893399            0.893778
1            0.905918            0.902124            0.904780
2            0.911988            0.909332            0.908953
3            0.916161            0.913505            0.914264
4            0.924507            0.922231            0.920713

   mean_train_score  std_train_score
0          0.893475         0.000152
1          0.905311         0.001958
2          0.911077         0.001602
3          0.915402         0.001357
4          0.923596         0.001958

[5 rows x 21 columns]
```




```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

In [289]:
```
from sklearn.metrics import r2_score
r2_score(bank_test['purchased'], predictions)
```

Out[289]:
```
0.7512832125603865
```

In [ ]: