In [1]:
```
from sklearn.datasets import load_boston
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
from sklearn import utils
from sklearn import tree
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
boston = load_boston()
X = boston.data
y = boston.target
```

```
/Users/ashleyxu/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function
load_boston is deprecated; `load_boston` is deprecated in 1.0 and will be removed in 1.2.

    The Boston housing prices dataset has an ethical problem. You can refer to
    the documentation of this function for further details.

    The scikit-learn maintainers therefore strongly discourage the use of this
    dataset unless the purpose of the code is to study and educate about
    ethical issues in data science and machine learning.

    In this special case, you can fetch the dataset from the original
    source::

        import pandas as pd
        import numpy as np


        data_url = "http://lib.stat.cmu.edu/datasets/boston"
        raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
        data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
        target = raw_df.values[1::2, 2]

    Alternative datasets include the California housing dataset (i.e.
    :func:`~sklearn.datasets.fetch_california_housing`) and the Ames housing
    dataset. You can load the datasets as follows::

        from sklearn.datasets import fetch_california_housing
        housing = fetch_california_housing()

    for the California housing dataset and::

        from sklearn.datasets import fetch_openml
        housing = fetch_openml(name="house_prices", as_frame=True)

    for the Ames housing dataset.

  warnings.warn(msg, category=FutureWarning)
```

1. Split the dataset into 70% training set and 30% test set.

In [3]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 25)
```

In [4]:
```
print("Shape of X_train: ",X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ",y_train.shape)
print("Shape of y_test",y_test.shape)
```

```
Shape of X_train:  (354, 13)
Shape of X_test:  (152, 13)
Shape of y_train:  (354,)
Shape of y_test (152,)
```

In [5]:
```
print(X_train)
```

```
[[  0.54011  20.        3.97    ...  13.      392.8       9.59   ]
 [  0.7258    0.        8.14    ...  21.      390.95     11.28   ]
 [  1.13081   0.        8.14    ...  21.      360.17     22.6    ]
 ...
 [  8.05579   0.       18.1     ...  20.2     352.58     18.14   ]
 [  0.40202   0.        9.9     ...  18.4     395.21     10.36   ]
 [  0.59005   0.       21.89    ...  21.2     385.76     11.12   ]]
```

In [6]:
```
print(y_test)
```

```
[21.4 19.1 25.  10.5 20.  28.1 32.4 13.6  9.5 35.2 14.5 14.  16.  22.7
 35.1 21.  28.4  8.8  5.  24.4 22.9 13.4 30.1 23.9 50.  23.  46.  24.7
 22.6 20.8 29.4 30.8 17.4 27.5 13.4 20.1 19.9 20.6 15.  20.1 18.7 20.2
 13.4 16.8  7.5 23.8 21.5 24.3 32.7 15.2 22.5 16.7 32.  18.3 17.1 11.7
 34.9 23.7 13.1 29.8 21.7 16.8 23.2 16.2 24.6 11.5 19.6 50.  23.9 15.
 19.3 22.6 20.  20.3 50.  22.8 16.1 29.  23.9 21.6 22.3 20.6 32.9 19.5
 10.4 22.  26.4 23.1 23.8 19.2 16.2 20.4 18.1 17.6 29.1 19.4 24.5 25.1
 14.4 29.9 15.6 21.4 18.3 16.1  8.3 14.5 15.1 33.4 48.3 20.5 17.2 22.3
 19.9 20.1 36.2 39.8 25.  38.7 22.9 23.4 21.5 13.1 22.  18.4 22.8 27.1
 19.4 18.8 27.5 50.   8.1 34.9 16.4 13.5 17.8 45.4 17.5 19.1 22.8 23.2
 33.2 29.  18.5 19.3 12.5 13.8 15.6 23.9 28.2 23.6 24.7 14.1]
```

2. Using scikit-learn's DecisionTreeClassifier, train a supervised learning model that can be used to generate predictions for your data

In [7]:
```
# convert y values to categorical values
lab = preprocessing.LabelEncoder()
y_train_transformed = lab.fit_transform(y_train)
y_test_transformed = lab.fit_transform(y_test)
```

In [8]:
```
clf = DecisionTreeClassifier(random_state=0)
clf = clf.fit(X_train,y_train_transformed)
```

3. Report the tree depth, number of leaves, feature importance, train score, and test score of the tree. Let the tree depth be Td.

In [9]:
```
Td = clf.get_depth()
leaves = clf.get_n_leaves()
feature_importance = clf.feature_importances_
train_score = clf.score(X_train, y_train_transformed)
test_score = clf.score(X_test, y_test_transformed)

print("tree depth: ", Td)
print("Number of leaves: ",leaves)
print("feature importance: ",feature_importance)
print("train score: ",train_score)
print("test score: ",test_score)
```

```
tree depth:  23
Number of leaves:  300
feature importance:  [0.12226168 0.02338521 0.05696511 0.00569379 0.07731762 0.13818823
 0.10759613 0.08120735 0.03900183 0.04378038 0.04498326 0.10818011
 0.1514393 ]
train score:  1.0
test score:  0.019736842105263157
```

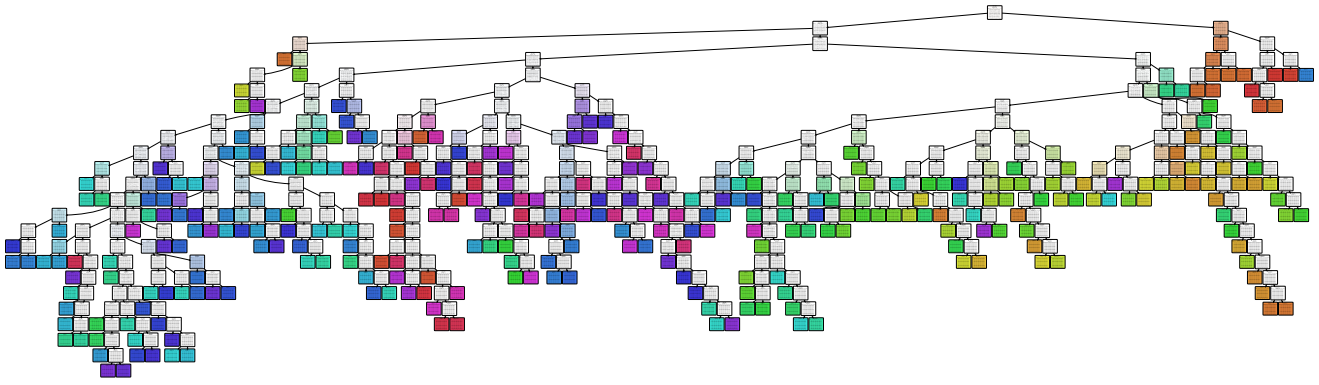
4. Show the visual output of the decision tree.

In [10]:
```
from sklearn.tree import export_graphviz
import graphviz

dot_data = tree.export_graphviz(clf, out_file=None,
                feature_names=boston.feature_names,
                filled=True, rounded=True,
                special_characters=True)
graph = graphviz.Source(dot_data)
```

In [11]:
```
graph
```

Out[11]:



5. Next, Generate (Td-1) decision trees on the same training set using fixed tree depths {1, 2, ...(T d – 1)}. The tree depth can be set using max=d, where d is the depth of the tree.

In [12]:
```
def generate_decision_trees(Td):
    clf = DecisionTreeClassifier(random_state=0, max_depth=Td)
    return clf
```

6. For each of the (Td-1) trees report, tree depth, number of leaves, feature importance, train score, and test score of the tree.

In [13]:
```
def generate_report(n):
    for i in range(1, n):
        clf = generate_decision_trees(i)
        clf = clf.fit(X_train,y_train_transformed)
        Td = clf.get_depth()
        leaves = clf.get_n_leaves()
        feature_importance = clf.feature_importances_
        train_score = clf.score(X_train, y_train_transformed)
        test_score = clf.score(X_test, y_test_transformed)
        print("======================")
        print("tree depth: ", Td)
        print("Number of leaves: ",leaves)
        print("feature importance: ",feature_importance)
        print("train score: ",train_score)
        print("test score: ",test_score)
```

In [14]:
```
generate_report(Td)
```

```
======================
tree depth:  1
Number of leaves:  2
feature importance:  [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
train score:  0.0423728813559322
test score:  0.006578947368421052
======================
tree depth:  2
Number of leaves:  4
feature importance:  [0.         0.         0.         0.         0.         0.56058587
 0.         0.21292597 0.22648815 0.         0.         0.
 0.        ]
train score:  0.05084745762711865
test score:  0.006578947368421052
======================
tree depth:  3
Number of leaves:  8
feature importance:  [0.         0.         0.         0.         0.16032157 0.39063399
 0.         0.13188129 0.14028138 0.         0.         0.
 0.17688178]
train score:  0.07062146892655367
test score:  0.006578947368421052
======================
tree depth:  4
Number of leaves:  14
feature importance:  [0.         0.         0.         0.         0.10657691 0.36098512
 0.08881409 0.20613922 0.0932548  0.         0.         0.
 0.14422986]
train score:  0.09322033898305085
test score:  0.013157894736842105
======================
tree depth:  5
Number of leaves:  21
feature importance:  [0.03201622 0.         0.         0.03201622 0.12365334 0.34774103
 0.         0.23732514 0.09925027 0.         0.         0.
 0.12799779]
train score:  0.12429378531073447
test score:  0.013157894736842105
======================
tree depth:  6
Number of leaves:  29
feature importance:  [0.06492848 0.         0.         0.         0.1502072  0.23579928
 0.02367129 0.18245577 0.0970523  0.         0.05972354 0.04734258
 0.13881956]
train score:  0.1638418079096045
test score:  0.006578947368421052
======================
tree depth:  7
Number of leaves:  41
feature importance:  [0.11052623 0.01122308 0.04892111 0.         0.09046516 0.21343539
 0.07424498 0.11582032 0.08114764 0.02589941 0.06082972 0.03813871
 0.12934824]
train score:  0.2175141242937853
test score:  0.006578947368421052
======================
tree depth:  8
Number of leaves:  56
feature importance:  [0.10891013 0.02580157 0.05776728 0.         0.08049333 0.14141068
 0.08501207 0.09943785 0.09718302 0.01290078 0.03254915 0.06074792
 0.19778623]
train score:  0.2711864406779661
test score:  0.006578947368421052
======================
tree depth:  9
Number of leaves:  79
feature importance:  [0.15623539 0.00698817 0.02464009 0.00931756 0.12628929 0.14146692
 0.11057576 0.10976495 0.05621386 0.         0.02350854 0.10133332
 0.13366616]
train score:  0.3474576271186441
test score:  0.013157894736842105
======================
tree depth:  10
Number of leaves:  106
feature importance:  [0.12402147 0.04725063 0.04105761 0.01431837 0.0928057  0.16955228
 0.06030194 0.06858817 0.05154455 0.01073878 0.01806289 0.11946272
 0.18229488]
train score:  0.4378531073446328
test score:  0.013157894736842105
======================
tree depth:  11
Number of leaves:  137
feature importance:  [0.12006497 0.02271813 0.03211535 0.01137938 0.0942393  0.1865875
 0.09757642 0.09888937 0.05518875 0.03275637 0.02004499 0.08261414
```

```
 0.14582533]
train score:  0.5254237288135594
test score:  0.02631578947368421
======================
tree depth:  12
Number of leaves:  169
feature importance:  [0.0933083  0.00938765 0.05831532 0.         0.05141568 0.18301207
 0.1295223  0.12074805 0.02675374 0.03755058 0.04130946 0.10976526
 0.13891158]
train score:  0.6242937853107344
test score:  0.03289473684210526
======================
tree depth:  13
Number of leaves:  198
feature importance:  [0.1197566  0.01228414 0.04867524 0.02047357 0.07507455 0.18968072
 0.10725777 0.11497724 0.03357574 0.04367695 0.01442583 0.10009027
 0.12005139]
train score:  0.7062146892655368
test score:  0.019736842105263157
======================
tree depth:  14
Number of leaves:  219
feature importance:  [0.09285655 0.01874462 0.04981321 0.00374892 0.0887914  0.1870754
 0.06183552 0.12184622 0.04448639 0.04640527 0.05274982 0.08176569
 0.14988099]
train score:  0.768361581920904
test score:  0.013157894736842105
======================
tree depth:  15
Number of leaves:  234
feature importance:  [0.0951719  0.03742643 0.06302265 0.         0.08208012 0.17542297
 0.09370069 0.1080799  0.03354644 0.04858551 0.02412715 0.10286797
 0.13596828]
train score:  0.8135593220338984
test score:  0.019736842105263157
======================
tree depth:  16
Number of leaves:  246
feature importance:  [0.10753573 0.02695782 0.06096009 0.         0.07382594 0.15941053
 0.11163078 0.09252672 0.03437047 0.03834174 0.03037843 0.10789322
 0.15616852]
train score:  0.847457627118644
test score:  0.019736842105263157
======================
tree depth:  17
Number of leaves:  270
feature importance:  [0.12584566 0.01646147 0.04615693 0.00731621 0.06799216 0.1559432
 0.09258413 0.09887852 0.0429559  0.0652403  0.04329759 0.10465453
 0.1326734 ]
train score:  0.9096045197740112
test score:  0.019736842105263157
======================
tree depth:  18
Number of leaves:  273
feature importance:  [0.10437808 0.02835028 0.04327815 0.00925184 0.09963791 0.12795287
 0.12162667 0.08043067 0.02682964 0.04116982 0.04170432 0.14980429
 0.12558547]
train score:  0.923728813559322
test score:  0.013157894736842105
======================
tree depth:  19
Number of leaves:  288
feature importance:  [0.11297575 0.02844352 0.03897433 0.         0.09715901 0.14308446
 0.10122357 0.0995325  0.03765576 0.05935004 0.04306186 0.09197645
 0.14656276]
train score:  0.96045197740113
test score:  0.013157894736842105
======================
tree depth:  20
Number of leaves:  286
feature importance:  [0.13465073 0.01882821 0.04717299 0.         0.08991655 0.13992172
 0.10287083 0.07518896 0.03765576 0.06254102 0.04190877 0.11148474
 0.13785972]
train score:  0.96045197740113
test score:  0.02631578947368421
======================
tree depth:  21
Number of leaves:  294
feature importance:  [0.11682749 0.02794634 0.04237162 0.00582649 0.07647608 0.18068909
 0.08602094 0.0859779  0.04282405 0.05491384 0.03437866 0.08580467
 0.15994284]
train score:  0.9774011299435028
test score:  0.019736842105263157
```

```
======================
tree depth:  22
Number of leaves:  299
feature importance:  [0.12439963 0.02717593 0.03675882 0.0095714  0.06987456 0.15515837
 0.08717878 0.10133683 0.04077351 0.06343704 0.05302787 0.09307763
 0.13822964]
train score:  0.9915254237288136
test score:  0.006578947368421052
```

7. Show the visual output of the decision tree with highest test score from the (Td-1) trees.

From the report above, the decision tree with highest test score is when Td = 12, the test score is 0.0329

In [15]:
```
clf = DecisionTreeClassifier(random_state=0, max_depth=12)
clf = clf.fit(X_train,y_train_transformed)
```

In [16]:
```
dot_data = tree.export_graphviz(clf, out_file=None,
                feature_names=boston.feature_names,
                filled=True, rounded=True,
                special_characters=True)
graph = graphviz.Source(dot_data)
```

In [17]:
```
graph
```

Out[17]: