```
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier,plot_tree

data = load_boston()
X = data.data
y = data.target

/usr/local/lib/python3.7/dist-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)
```

```
import pandas as pd

y = list(y)
for i in range(len(y)):
  idx = y.index(min(y))
  if i < len(y)/3:
    y[idx] = 100
  elif i > len(y)/3 and i < 2*(len(y)/3):
    y[idx] = 200
  else:
    y[idx] = 300

def split_equally(y):
  if y == 100:
    return 0
  elif y == 200:
    return 1
  else:
    return 2

y_data = list(map(split_equally, y))

X = pd.DataFrame(X)
y = pd.DataFrame(y_data)

#1. Split the data in 70/30
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.3)

#2. Use Decision Tree Classifier. Train a supervised learning model to
generate predictions.
dt = DecisionTreeClassifier()
dt = dt.fit(X_train, y_train)
predict = dt.predict(X_test)

#3. Report the tree depth, number of leaves, feature importance, train
score and test score
Td = dt.get_depth()
print("The depth of the tree : ", Td)
print("\nNumber of leaves : ", dt.get_n_leaves())
print("\nFeature Importance : ", dt.feature_importances_)
print("\nTrain Score : ", dt.score(X_train, y_train))
print("\nTest Score : ", dt.score(X_test, y_test))
```

```
The depth of the tree :  13
```

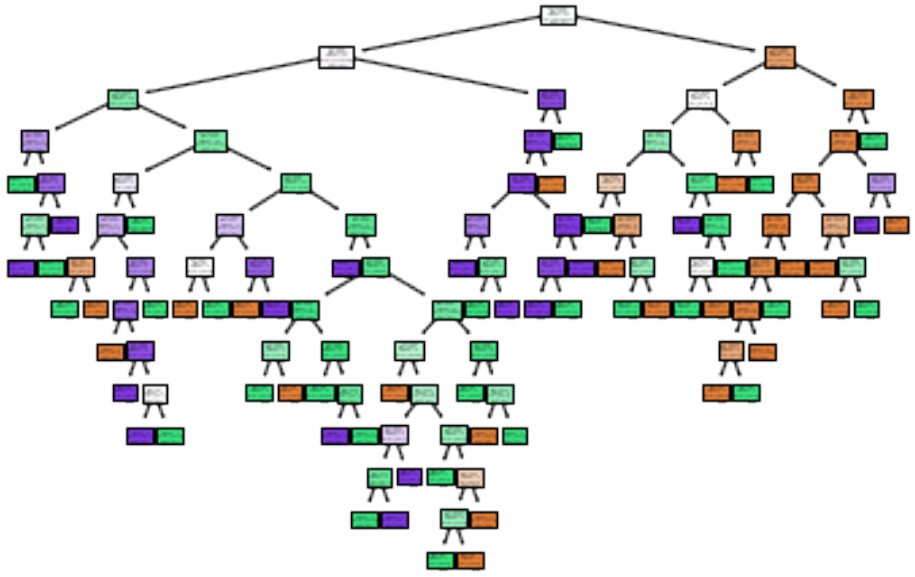
```
Number of leaves :  62

Feature Importance :  [0.07152736 0.         0.05491654 0.00636614
0.05510525 0.26071412
 0.07746326 0.03779531 0.02088058 0.01450709 0.01480504 0.06238938
 0.32352992]

Train Score :  1.0

Test Score :  0.6776315789473685
```

```
#4. Show visual output
plt.figure()
plot_tree(dt, feature_names= y_data, class_names=True, filled=True)
plt.show()
```



```
#5. Generate Td-1 decision tree on same training set.
#6. For each (Td-1) report the scores

td_max = 0
dt_max = dt.fit(X_train, y_train)
for d in range(1, Td):
  dt = DecisionTreeClassifier(max_depth=d)
  dt = dt.fit(X_train, y_train)
  dt_predict = dt.predict(X_test)
  print("The depth of the tree : ", d)
  print("\nNumber of leaves : ", dt.get_n_leaves())
  print("\nFeature Importance : ", dt.feature_importances_)
  print("\nTrain Score : ", dt.score(X_train, y_train))
  print("\nTest Score : ", dt.score(X_test, y_test))
  if td_max < dt.score(X_test, y_test):
```

```
        td_max=dt.score(X_test, y_test)
        dt_max=dt

The depth of the tree :  1

Number of leaves :  2

Feature Importance :  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]

Train Score :  0.5932203389830508

Test Score :  0.631578947368421
The depth of the tree :  2

Number of leaves :  4

Feature Importance :  [0.         0.         0.         0.
0.07577166 0.38065911
 0.         0.         0.         0.         0.         0.
 0.54356923]

Train Score :  0.7824858757062146

Test Score :  0.7236842105263158
The depth of the tree :  3

Number of leaves :  8

Feature Importance :  [0.         0.         0.         0.
0.06658292 0.33449703
 0.07117803 0.02511351 0.         0.         0.02497727 0.
 0.47765124]

Train Score :  0.8305084745762712

Test Score :  0.756578947368421
The depth of the tree :  4

Number of leaves :  14

Feature Importance :  [0.01371437 0.         0.01850522 0.
0.05978052 0.31588107
 0.06390617 0.03465203 0.         0.01168839 0.02242548 0.03059443
 0.42885232]

Train Score :  0.8531073446327684

Test Score :  0.7171052631578947
The depth of the tree :  5
```

```
Number of leaves :  22

Feature Importance :  [0.01231164 0.         0.05045395 0.
0.06443873 0.29859543
 0.06775766 0.03620758 0.         0.01049288 0.02013176 0.02746517
 0.41214522]

Train Score :  0.8870056497175142

Test Score :  0.743421052631579
The depth of the tree :  6

Number of leaves :  32

Feature Importance :  [0.02930049 0.         0.06101882 0.
0.07370269 0.27201969
 0.06172706 0.02308595 0.00080569 0.02287783 0.0341123  0.03748064
 0.38386884]

Train Score :  0.9209039548022598

Test Score :  0.7236842105263158
The depth of the tree :  7

Number of leaves :  42

Feature Importance :  [0.04125785 0.         0.05609103 0.00724929
0.08329401 0.2623536
 0.07777233 0.02122156 0.00879538 0.00878701 0.03135744 0.04411947
 0.35770102]

Train Score :  0.9519774011299436

Test Score :  0.7039473684210527
The depth of the tree :  8

Number of leaves :  47

Feature Importance :  [0.07158943 0.         0.05453565 0.
0.072137   0.27700657
 0.05525407 0.03223444 0.01262382 0.01559162 0.02422279 0.04172135
 0.34308326]

Train Score :  0.9548022598870056

Test Score :  0.7105263157894737
The depth of the tree :  9
```

```
Number of leaves :  52

Feature Importance :  [0.03862637 0.         0.06250939 0.
0.06543243 0.27412317
 0.06024086 0.03188948 0.01554409 0.00827201 0.03428585 0.0700652
 0.33901115]

Train Score :  0.9661016949152542

Test Score :  0.7039473684210527
The depth of the tree :  10

Number of leaves :  56

Feature Importance :  [0.03559486 0.         0.0608364  0.00664176
0.07012338 0.25763571
 0.0775699  0.03000842 0.01512807 0.01926782 0.03336823 0.04595686
 0.3478686 ]

Train Score :  0.9774011299435028

Test Score :  0.6973684210526315
The depth of the tree :  11

Number of leaves :  59

Feature Importance :  [0.06505819 0.         0.05010841 0.
0.08225974 0.27068523
 0.07880101 0.04097849 0.02052162 0.01504545 0.01506072 0.04624959
 0.31523155]

Train Score :  0.9915254237288136

Test Score :  0.7039473684210527
The depth of the tree :  12

Number of leaves :  60

Feature Importance :  [0.05585735 0.00640237 0.06504604 0.00640237
0.05569495 0.26049054
 0.0756277  0.0511773  0.00748329 0.00776045 0.02712494 0.05932984
 0.32160283]

Train Score :  0.9971751412429378

Test Score :  0.6842105263157895
```

```
#7. Visualize
import graphviz
from sklearn import tree
from io import StringIO
import pydotplus
from IPython.display import Image

def visualize_tree(dt, feature_name):
  dot_data = StringIO()
  tree.export_graphviz(dt, out_file=dot_data, feature_names =
feature_name,
                       class_names=True, filled = True,
                       rounded = True, special_characters=True)
  graph = pydotplus.graph_from_dot_data(dot_data(dot_data.getvalue()))
  return Image(graph.create_png())


visualize_tree(dt, y_data)

---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-31-b439c89ed8bb> in <module>()
----> 1 visualize_tree(dt, y_data)

<ipython-input-28-c3685632e5f0> in visualize_tree(dt, feature_name)
     10   tree.export_graphviz(dt, out_file=dot_data, feature_names =
feature_name,
     11                        class_names=True, filled = True,
---> 12                        rounded = True,
special_characters=True)
     13   graph =
pydotplus.graph_from_dot_data(dot_data(dot_data.getvalue()))
     14   return Image(graph.create_png())

/usr/local/lib/python3.7/dist-packages/sklearn/tree/_export.py in
export_graphviz(decision_tree, out_file, max_depth, feature_names,
class_names, label, filled, leaves_parallel, impurity, node_ids,
proportion, rotate, rounded, special_characters, precision, fontname)
    887             fontname=fontname,
    888         )
--> 889         exporter.export(decision_tree)
    890
    891         if return_string:

/usr/local/lib/python3.7/dist-packages/sklearn/tree/_export.py in
export(self, decision_tree)
    452                 raise ValueError(
    453                     "Length of feature_names, %d does not
```

```
match number of features, %d"
--> 454                      % (len(self.feature_names),
decision_tree.n_features_in_)
    455                  )
    456         # each part writes to out_file

ValueError: Length of feature_names, 506 does not match number of
features, 13
```

```
print("Highest test score with depth",dt_max.get_depth())
plt.figure()
plot_tree(dt_max, feature_names=y_data,class_names=True,filled=True)
plt.show()
```

```
Highest test score with depth 3
```