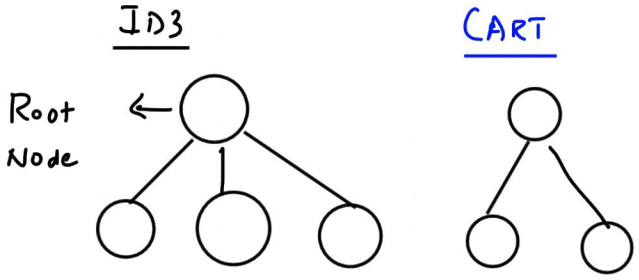
Decision Tree

① Decision Tree Classifier [classification]

② Decision Tree Regressor [Regression]



Decision Tree Classifier

Two techniques

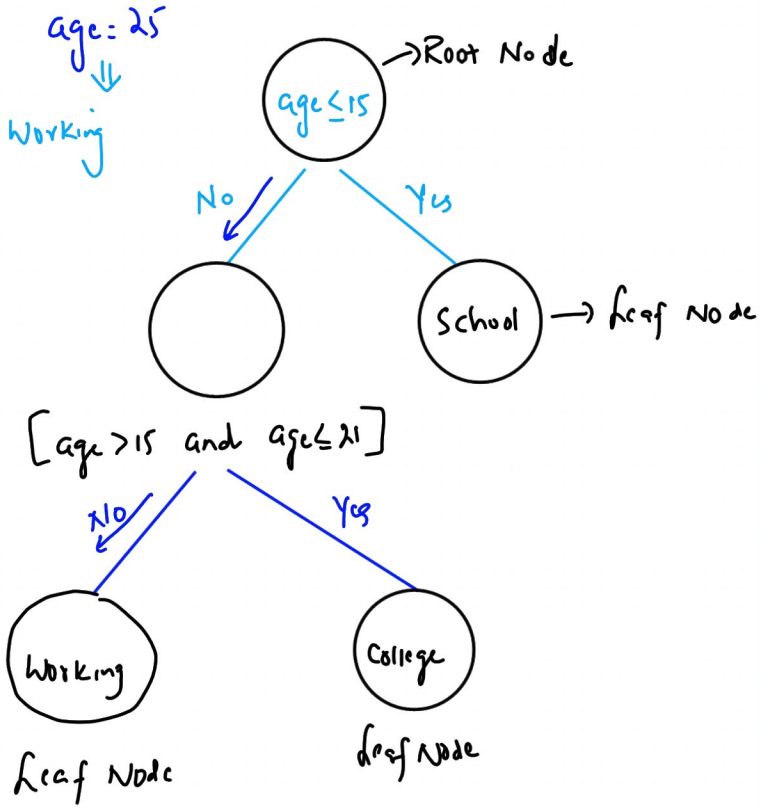
① ID3 [Iterative Dichotomiser 3]

② CART [Classification And Regression Tree]

Multinested if else clause

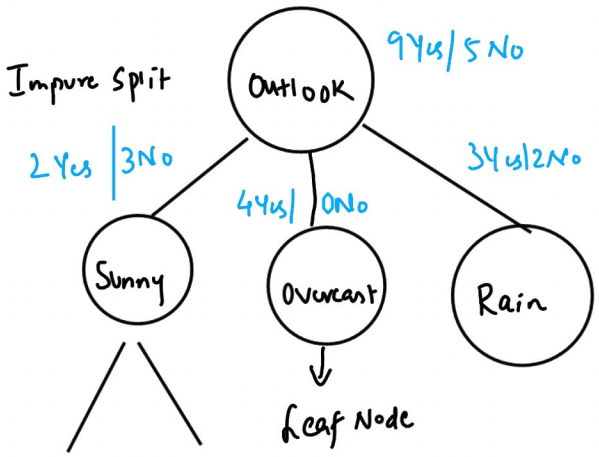
age = 14

```
if (age ≤ 15):
    Print("School")
elif (age>15 and age ≤21):
    Print("College")
else:
    Print("Working").
```



Dataset → Problem statement

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No



① Purity Split check – Pure Split or Impure Split

→ Entropy
→ Gini Impurity
} Measure of Purity

② What feature you need to select to start the split – Information Gain.

① Purity Check

Binary classification

① Entropy

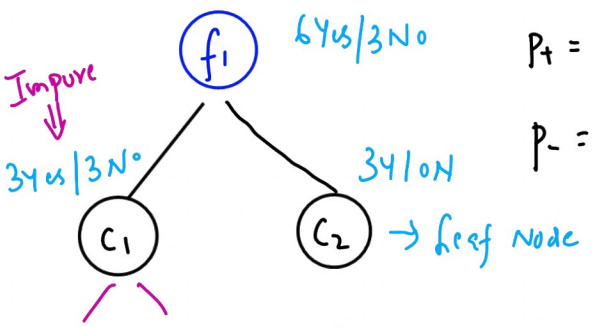
$$H(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

p_+ = probability of positive category

p_- = " " negative category

② Gini Impurity

$$GI = 1 - \sum_{i=1}^{n} (p)^2$$



$p_+ = \frac{3}{6} = \frac{1}{2}$

$p_- = \frac{3}{6} = \frac{1}{2}$

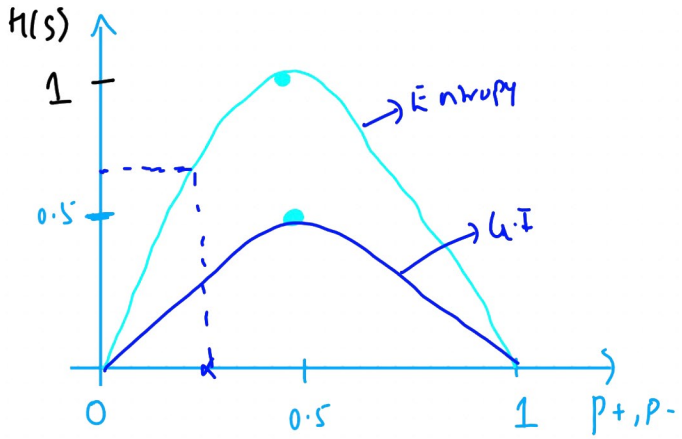
$$H(C_1) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

$= -\frac{3}{6}\log_2 3/6 - 3/6 \log 3/6.$

$H(C_1) = 1$ ⇒ Impure Split

$H(C_2) = -\frac{3}{3}\log_2 3/3 - \frac{0}{3}\log_2 0/3$

$H(C_2) = 0$ ⇒ Pure Split



② Gini Impurity

$G.I = 1 - \sum_{i=1}^{n} (p)^2$

34Yes/0No

$G.I(C_1) = 1 - \left[(p_+)^2 + (p_-)^2\right]$

$= 1 - \left[\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right]$

$= 1 - \frac{1}{2} = 0.5$ ⇒ Impure Split

$G.I(C_2) = 1 - \left[(3/3)^2 + (0/3)^2\right]$

$= 1 - 1 = 0$ ⇒ Pure Split.

Multiclass Classification Problem : 3 categories In O/P

$H(S) = -P_{C_1}\log_2 P_{C_1} - P_{C_2}\log_2 P_{C_2} - P_{C_3}\log_2 P_{C_3}$

$G.I = 1 - \left[(P_{C_1})^2 + (P_{C_2})^2 + (P_{C_3})^2\right]$

② Information Gain → Which feature to select to start the split?

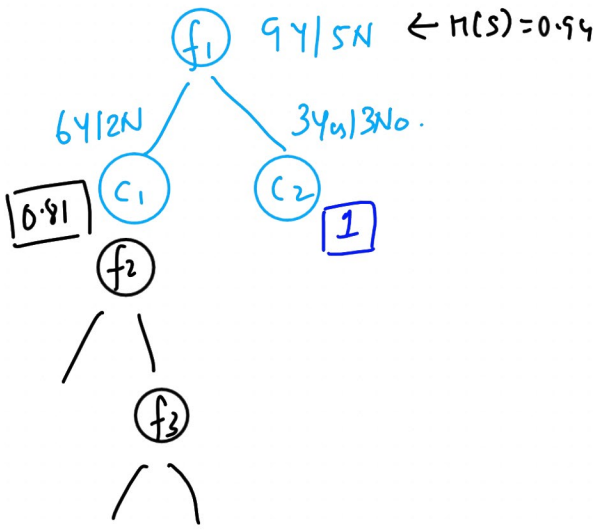
$Gain(S, f_1) = H(S) - \sum_{v \in val} \frac{|S_v|}{|S|} H(S_v)$ → Entropy of Categories

$H(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$

$= -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 (5/14)$

≈ 0.94

f1 f2 f3 O/P



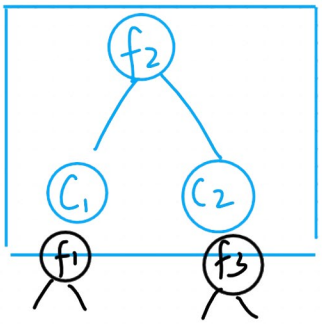
$H(C_1) = -\frac{6}{8} \log_2 \left(\frac{6}{8}\right) - \frac{2}{8} \log_2 \frac{2}{8} \approx \boxed{0.81}$

$H(C_2) = \underline{1}$

$Gain(S, f_1) = H(S) - \sum_{v \in Val} \frac{|S_v|}{|S|} H(S_v)$ → Entropy of Categories

$= 0.94 - \left[\frac{8}{14} \times 0.81 + \frac{6}{14} \times 1\right]$

$\boxed{Gain(S, f_1) = 0.049}$



⇒ Information Gain = 0.051

$\boxed{Gain(S, f_2) = 0.051} > \boxed{Gain(S, f_1) = 0.049}$

We need to splitting by using f2 features

Entropy vs Gini Impurity

When dataset is small → Entropy [log formula]

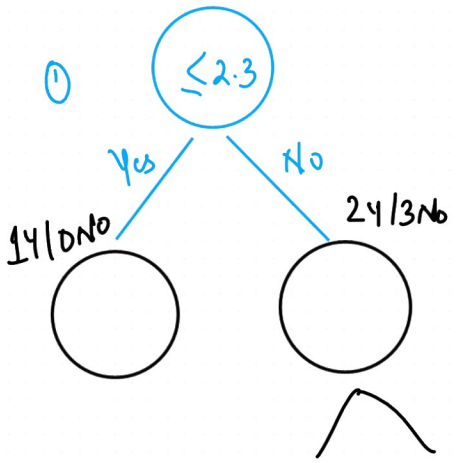
When dataset is huge → Gini Impurity [Simple Maths]

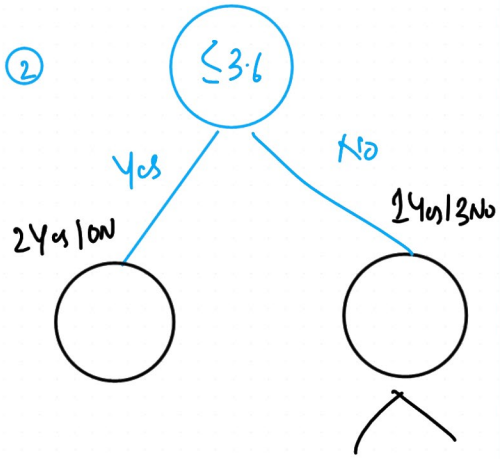
(6) What if my feature is continuous.

f_1	O/P
→ 2.3	Yes
→ 3.6	Yes
4	No
5.2	No
6.7	Yes
7.8	No

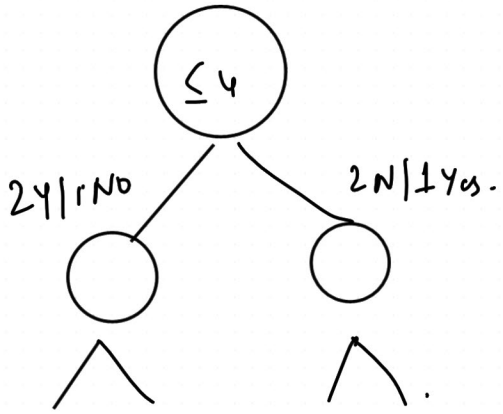
① Sort the feature f_1

① Threshold = 2.3





③ Threshold = 4



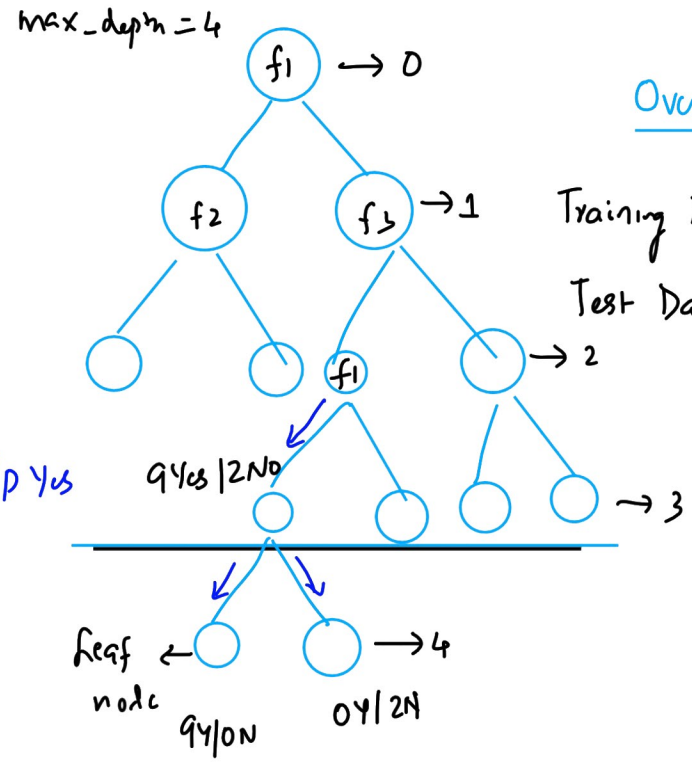
Time Complexity ↑↑

DATASET ↑↑.

Decision Tree Post Pruning And Pre Pruning [Reduce Overfitting]

Training Data

f1 f2 f3 O/P



Overfitting

Training Data Acc ↑↑

Test Data Acc ↓↓

Generalized Model

Reduce Overfitting

max_depth = 3

① Post Pruning

(i) Construct Entire Decision Tree to complete Leaf Node

(ii) Pruning the Decision Tree

(iii) Smaller Dataset Suitable.

② Pre Pruning

(i) Hyperparameter Tunning to Select the Best parameter

Decision Tree Regressor

Dataset

O/P (continuous)

Exp	Career Gap	Salary
2	Yes	40K
2.5	Yes	42K
3	No	52K
4	No	60K
4.5	Yes	56K

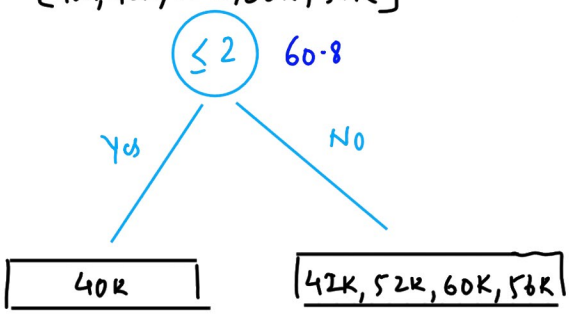
$\bar{y}$ = 50K [Mean]

DT Classifier
1. Entropy
2. G.I
3. Information Gain

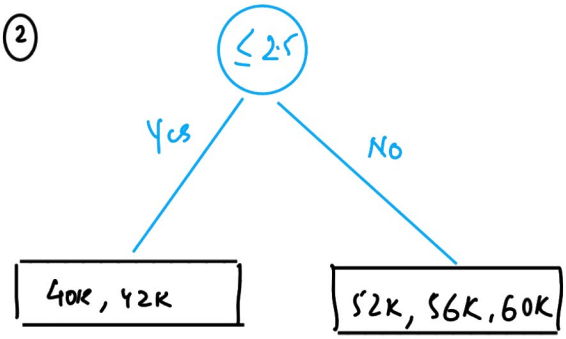
DT Regressor
1. Variance Reduction
2. Variance

0.10

① [40K, 42K, 52K, 60K, 56K]



②



Final aim Variance Reduction

$$\text{Variance or Error} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2$$ [Mean Squared Error]

$$\text{Variance (Root)} = \frac{1}{5}\left[(40-50)^2+(42-50)^2+(52-50)^2 + (60-50)^2+(56-50)^2\right]$$

$$= \frac{1}{5}\left[100+64+4+100+36\right]$$

$$= 60.8$$

Var(Root) = 60.8

$$\text{Var(left)} = \frac{1}{2}\left[100+64\right]$$

$$= 82$$

$$\text{Var(Right)} = \frac{1}{3}\left[4+36+100\right]$$

$$= 46.66$$

$$\text{Variance (Left)} = \frac{1}{1}\left[(40-50)^2\right]$$

$$= 100$$

$$\text{Variance (Right)} = \frac{1}{4}\left[(64) + 4 + 100 + 36\right]$$

$$= 51$$

Variance Reduction : $\text{Var(Root)} - \sum w_i \text{Var(child)}$

$$= 60.8 - \left[\frac{1}{5} * 100 + \frac{4}{5} * 51\right]$$

Variance Reduction $= 0$

Variance Reduction :

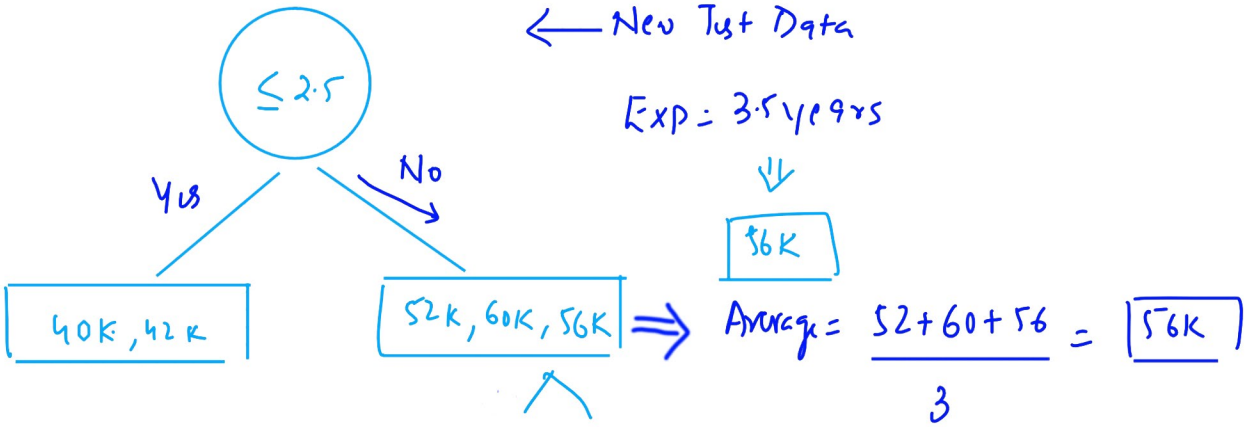
$$60.8 - \left[\frac{2}{5} * 82 + \frac{3}{5} * 46.66\right]$$

$$= 0.004$$

Variance Reduction < Variance Reduction

Select this split wit feature.

Selected Split



← New Test Data

Exp = 3.5 years

Average $= \frac{52+60+56}{3} = 56K$

ML_classification_Decision_Tree_Classifier_iris_dataset

1 Decision Tree Classifier Implementation With Post Prunning And Preprunning

[44]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[45]:
```
from sklearn.datasets import load_iris
## for multi-class classification
```

[46]:
```
dataset = load_iris()
```

[47]:
```
print(dataset.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
```

```
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988
```

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken from Fisher's paper. Note that it's the same as in R, but not as in the UCI Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

.. topic:: References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
- See also: 1988 MLC Proceedings, 54-64. Cheeseman et al"s AUTOCLASS II conceptual clustering system finds 3 classes in the data.
- Many, many more …

[48]:
```
## we can also import this dataset from seaborn library
import seaborn as sns
sns.load_dataset('iris')
```

[48]:
```
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            …            …             …            …        …
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]
```

[49]:
```
df = sns.load_dataset('iris')
```

[50]:
```
## in seaborn dataset is already in a dataframe
## but target/dependent variable is in words, so we can
## take target variable from sklearn-dataset it's in numerical
### form to train model
dataset.target
```

[50]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

[51]:
```
#Independant and Dependant features
X = df.iloc[:,:-1] ## store all columns expect last column
y = dataset.target ## from sklearn-iris-dataset
```

1.0.1 we don't use feature-scaling in Decision tree, because it's of no use, because it try to split the nodes from one-feature to another to it's complete depth, so feature-scaling will only increase time complexity (even if we perform it)

[52]:
```
### train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

1.1 Post prunning : first we'll construct decision tree and then we decide the parameter

- good for small datasets

[53]:
```
from sklearn.tree import DecisionTreeClassifier
```

[54]:
```
classifier =DecisionTreeClassifier() ## default parameter got selected
```

[55]:
```
classifier.fit(X_train,y_train)
## internally performed information-gain, entropy etc..
## in documentation we can see by-default criterion='gini'
## so it used gini, we know it's a small data set and if
### we want to use 'entropy' we can pass it as parameter on
## time of initialization
```

[55]:
```
DecisionTreeClassifier()
```

[56]:
```
classifier =DecisionTreeClassifier(criterion='entropy') ## for small datasets␣
↪because entropy uses 'log'
```

[57]:
```
classifier.fit(X_train,y_train)
```

[57]:
```
DecisionTreeClassifier(criterion='entropy')
```

[66]:
```
X_train.head()
```

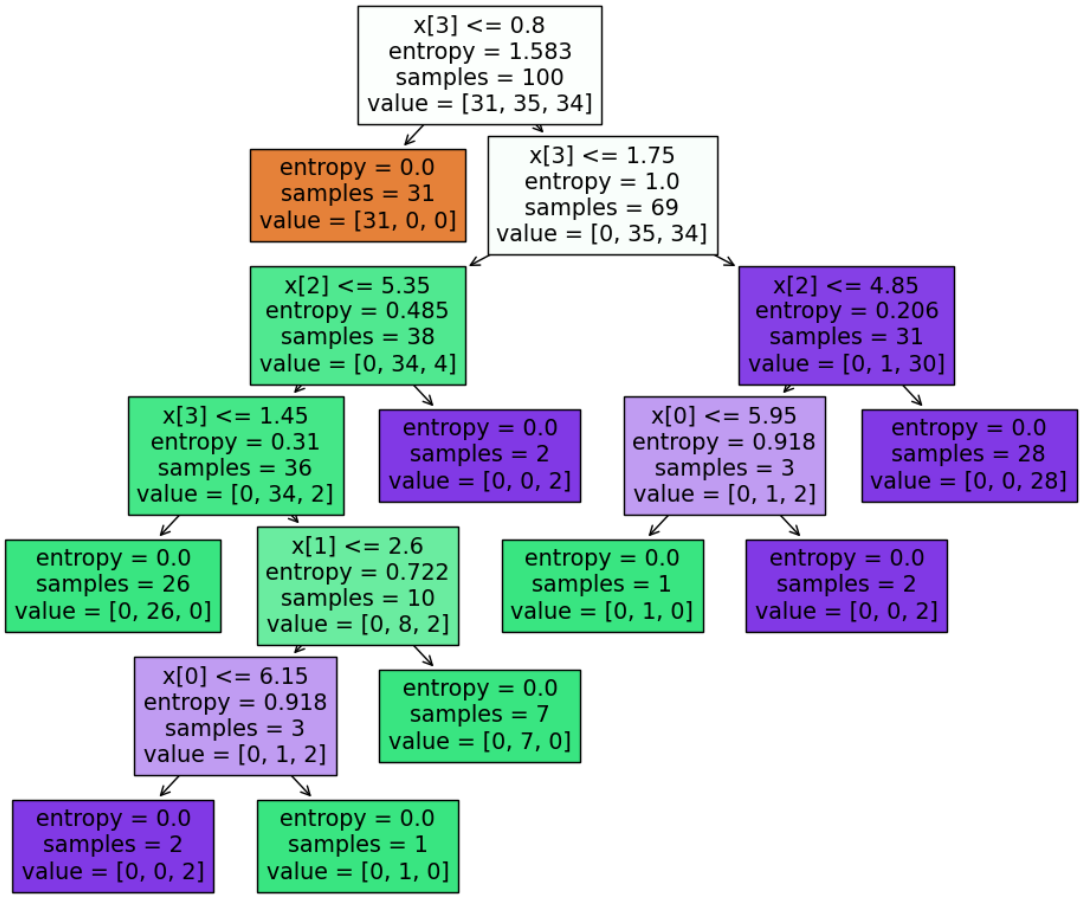
[66]:

	sepal_length	sepal_width	petal_length	petal_width
96	5.7	2.9	4.2	1.3
105	7.6	3.0	6.6	2.1
66	5.6	3.0	4.5	1.5
0	5.1	3.5	1.4	0.2
122	7.7	2.8	6.7	2.0

[58]:
```
## to display all decision tree splits
from sklearn import tree
plt.figure(figsize=(12,10)) ## inside this figure we'll display decision tree
tree.plot_tree(classifier, filled=True)##filled=true to put color inside splits
## this plot_tree() function plots the split details
```

[58]:
```
[Text(0.4444444444444444, 0.9285714285714286, 'x[3] <= 0.8\nentropy =
1.583\nsamples = 100\nvalue = [31, 35, 34]'),
 Text(0.3333333333333333, 0.7857142857142857, 'entropy = 0.0\nsamples =
31\nvalue = [31, 0, 0]'),
 Text(0.5555555555555556, 0.7857142857142857, 'x[3] <= 1.75\nentropy =
1.0\nsamples = 69\nvalue = [0, 35, 34]'),
 Text(0.3333333333333333, 0.6428571428571429, 'x[2] <= 5.35\nentropy =
0.485\nsamples = 38\nvalue = [0, 34, 4]'),
```

```
 Text(0.2222222222222222, 0.5, 'x[3] <= 1.45\nentropy = 0.31\nsamples =
36\nvalue = [0, 34, 2]'),
 Text(0.1111111111111111, 0.35714285714285715, 'entropy = 0.0\nsamples =
26\nvalue = [0, 26, 0]'),
 Text(0.3333333333333333, 0.35714285714285715, 'x[1] <= 2.6\nentropy =
0.722\nsamples = 10\nvalue = [0, 8, 2]'),
 Text(0.2222222222222222, 0.21428571428571427, 'x[0] <= 6.15\nentropy =
0.918\nsamples = 3\nvalue = [0, 1, 2]'),
 Text(0.1111111111111111, 0.07142857142857142, 'entropy = 0.0\nsamples =
2\nvalue = [0, 0, 2]'),
 Text(0.3333333333333333, 0.07142857142857142, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1, 0]'),
 Text(0.4444444444444444, 0.21428571428571427, 'entropy = 0.0\nsamples =
7\nvalue = [0, 7, 0]'),
 Text(0.4444444444444444, 0.5, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0, 2]'),
 Text(0.7777777777777778, 0.6428571428571429, 'x[2] <= 4.85\nentropy =
0.206\nsamples = 31\nvalue = [0, 1, 30]'),
 Text(0.6666666666666666, 0.5, 'x[0] <= 5.95\nentropy = 0.918\nsamples =
3\nvalue = [0, 1, 2]'),
 Text(0.5555555555555556, 0.35714285714285715, 'entropy = 0.0\nsamples =
1\nvalue = [0, 1, 0]'),
 Text(0.7777777777777778, 0.35714285714285715, 'entropy = 0.0\nsamples =
2\nvalue = [0, 0, 2]'),
 Text(0.8888888888888888, 0.5, 'entropy = 0.0\nsamples = 28\nvalue = [0, 0,
28]')]
```



[59]:
```
## we can see in plot that, at level-2 (max-depth) most(3) categories
## are already split, and after that model is overfitting so we
## will set parameter max-depth=2
```

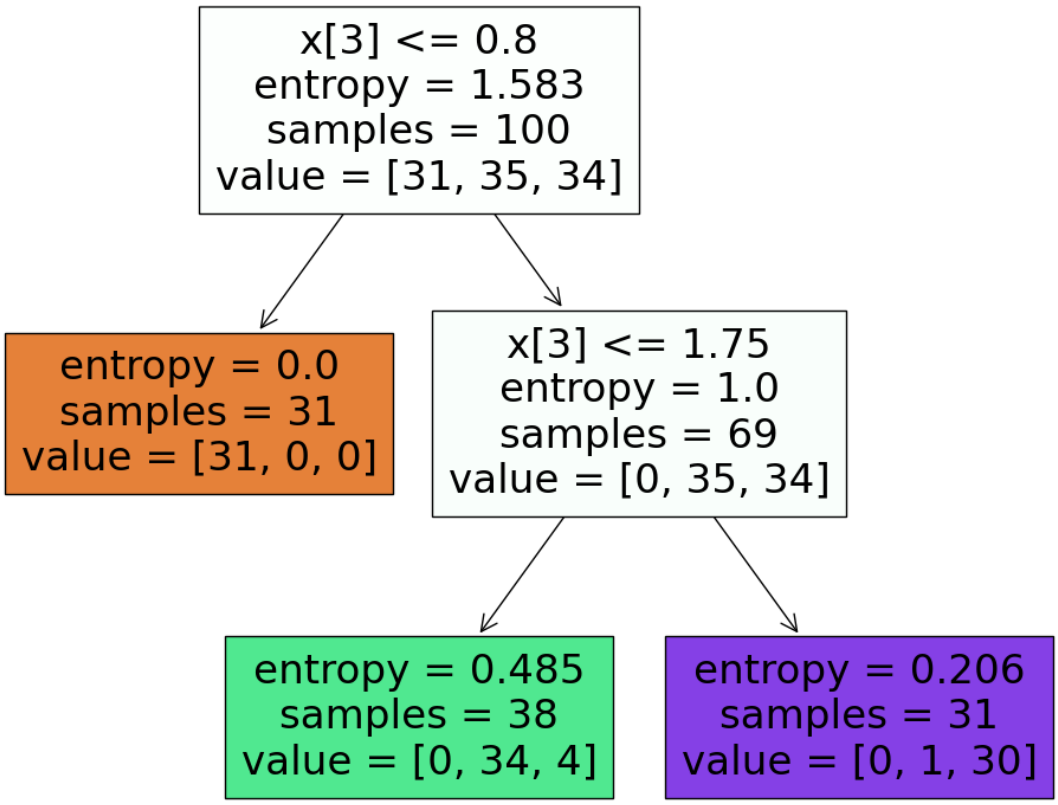
[60]:
```
## Post Prunning
classifier = DecisionTreeClassifier(criterion='entropy', max_depth=2)
classifier.fit(X_train,y_train)
```

[60]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=2)
```

[61]:
```
## again plotting
from sklearn import tree
plt.figure(figsize=(12,10))
tree.plot_tree(classifier, filled=True)
```

[61]:
```
[Text(0.4, 0.8333333333333334, 'x[3] <= 0.8\nentropy = 1.583\nsamples =
100\nvalue = [31, 35, 34]'),
 Text(0.2, 0.5, 'entropy = 0.0\nsamples = 31\nvalue = [31, 0, 0]'),
```

```
 Text(0.6, 0.5, 'x[3] <= 1.75\nentropy = 1.0\nsamples = 69\nvalue = [0, 35,
34]'),
 Text(0.4, 0.16666666666666666, 'entropy = 0.485\nsamples = 38\nvalue = [0, 34,
4]'),
 Text(0.8, 0.16666666666666666, 'entropy = 0.206\nsamples = 31\nvalue = [0, 1,
30]')]
```



[62]:
```
## Prediction
y_pred = classifier.predict(X_test)
```

[63]:
```
y_pred
```

[63]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0, 1, 1, 2, 1, 2])
```

[64]:
```
## check accuracy
from sklearn.metrics import accuracy_score, classification_report
```

```
score = accuracy_score(y_pred,y_test)
print(score)
print(classification_report(y_pred,y_test))
```

```
0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.94      0.97        16
           2       0.94      1.00      0.97        15

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

[ ]:

1.2 DecisionTree Prepruning and Hyperparameter Tuning for Huge Data

[67]:
```
## with grid search CV we get many errors because some of the
## parameter gets deprecated , so removing/ignore warnings
import warnings
warnings.filterwarnings('ignore') ## it'll ignore whatever warning gridsearchCV␣
 ↪throw
```

[68]:
```
parameter={
    'criterion':['gini', 'entropy', 'log_loss'],
    'splitter':['best','random'],
    'max_depth':[1,2,3,4,5],
    'max_features':['auto','sqrt','log2']
}
## passing parameter option/combination available for␣
 ↪classification-DecisionTree
## to select best amoung them to GridSearchCV
```

[69]:
```
from sklearn.model_selection import GridSearchCV
```

[70]:
```
classifier=DecisionTreeClassifier()
clf=GridSearchCV(classifier,param_grid=parameter, cv=5, scoring='accuracy')
## for classification we use 'accuracy' for scoring
```

[71]:
```
clf.fit(X_train,y_train)
```

[71]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy', 'log_loss'],
                         'max_depth': [1, 2, 3, 4, 5],
```

```
                        'max_features': ['auto', 'sqrt', 'log2'],
                        'splitter': ['best', 'random']},
                scoring='accuracy')
```

[72]:
```
clf.best_params_ ##return best parameter combination
```

[72]:
```
{'criterion': 'gini',
 'max_depth': 3,
 'max_features': 'auto',
 'splitter': 'best'}
```

[73]:
```
y_pred = clf.predict(X_test)
```

[74]:
```
from sklearn.metrics import accuracy_score, classification_report
score = accuracy_score(y_pred,y_test)
print(score)
print(classification_report(y_pred,y_test))
```

```
0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      0.94      0.97        16
           2       0.94      1.00      0.97        15

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

[ ]:
```

```

ML_regression_Decision_Tree_Regressor_california_house_pricing

0.1 Decision Tree Regressor Implementation

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
## California House Pricing Dataset
from sklearn.datasets import fetch_california_housing
california_df = fetch_california_housing()
```

[3]:
```
california_df
```

[3]:
```
{'data': array([[   8.3252    ,   41.        ,    6.98412698, …,
2.55555556,
           37.88      , -122.23      ],
        [   8.3014    ,   21.        ,    6.23813708, …,    2.10984183,
           37.86      , -122.22      ],
        [   7.2574    ,   52.        ,    8.28813559, …,    2.80225989,
           37.85      , -122.24      ],
        …,
        [   1.7       ,   17.        ,    5.20554273, …,    2.3256351 ,
           39.43      , -121.22      ],
        [   1.8672    ,   18.        ,    5.32951289, …,    2.12320917,
           39.43      , -121.32      ],
        [   2.3886    ,   16.        ,    5.25471698, …,    2.61698113,
           39.37      , -121.24      ]]),
 'target': array([4.526, 3.585, 3.521, …, 0.923, 0.847, 0.894]),
 'frame': None,
 'target_names': ['MedHouseVal'],
 'feature_names': ['MedInc',
  'HouseAge',
  'AveRooms',
  'AveBedrms',
  'Population',
  'AveOccup',
  'Latitude',
```

```
 'Longitude'],
 'DESCR': '.. _california_housing_dataset:\n\nCalifornia Housing
dataset\n--------------------------\n\n**Data Set Characteristics:**\n\n
:Number of Instances: 20640\n\n    :Number of Attributes: 8 numeric, predictive
attributes and the target\n\n    :Attribute Information:\n        - MedInc
median income in block group\n        - HouseAge      median house age in block
group\n        - AveRooms      average number of rooms per household\n        -
AveBedrms     average number of bedrooms per household\n        - Population
block group population\n        - AveOccup      average number of household
members\n        - Latitude      block group latitude\n        - Longitude
block group longitude\n\n    :Missing Attribute Values: None\n\nThis dataset was
obtained from the StatLib
repository.\nhttps://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html\n\nThe
target variable is the median house value for California districts,\nexpressed
in hundreds of thousands of dollars ($100,000).\n\nThis dataset was derived from
the 1990 U.S. census, using one row per census\nblock group. A block group is
the smallest geographical unit for which the U.S.\nCensus Bureau publishes
sample data (a block group typically has a population\nof 600 to 3,000
people).\n\nAn household is a group of people residing within a home. Since the
average\nnumber of rooms and bedrooms in this dataset are provided per
household, these\ncolumns may take surpinsingly large values for block groups
with few households\nand many empty houses, such as vacation resorts.\n\nIt can
be downloaded/loaded using
the\n:func:`sklearn.datasets.fetch_california_housing` function.\n\n.. topic::
References\n\n    - Pace, R. Kelley and Ronald Barry, Sparse Spatial
Autoregressions,\n      Statistics and Probability Letters, 33 (1997)
291-297\n'}
```

[5]:
```
df = pd.DataFrame(california_df.data,columns=california_df.feature_names)
df['Target'] = california_df.target
```

[6]:
```
df.shape
```

[6]:
```
(20640, 9)
```

[7]:
```
## Taking Sample Data
df=df.sample(frac=0.25)
```

[8]:
```
df.shape
```

[8]:
```
(5160, 9)
```

[10]:
```
df ## now rows are less, decreasing it just for this example
```

[10]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	\
11002	5.3903	29.0	6.451477	1.029536	1197.0	2.525316	33.75	
16877	6.2210	52.0	6.571429	0.966667	530.0	2.523810	37.60	

```
8838   2.8889      30.0  3.639582   1.055511      1910.0  1.536605     34.08
14080  2.7928      48.0  5.106599   1.071066       893.0  2.266497     32.77
13145  1.6458      52.0  4.292453   1.000000       429.0  2.023585     38.26
…         …         …         …          …           …         …          …
3085   1.5957      18.0  6.327586   1.559387       986.0  1.888889     35.70
264    2.3125      44.0  4.399491   1.053435      1240.0  3.155216     37.78
7314   1.7562      30.0  3.442308   1.069231      1827.0  3.513462     33.97
20220  4.1250      52.0  5.639798   1.057935       941.0  2.370277     34.28
18915  1.7560      33.0  5.309322   1.008475       753.0  3.190678     38.15

       Longitude  Target
11002    -117.80   2.866
16877    -122.40   4.203
8838     -118.38   3.353
14080    -117.12   1.750
13145    -121.51   0.528
…            …      …
3085     -118.50   1.014
264      -122.21   1.028
7314     -118.19   1.813
20220    -119.27   3.490
18915    -122.23   0.864

[5160 rows x 9 columns]
```

[11]:
```
#independent features
X=df.iloc[:,:-1]
#dependent feature
y=df.iloc[:,-1]
```

[12]:
```
X.shape , y.shape
```

[12]:
```
((5160, 8), (5160,))
```

[13]:
```
X.head()
```

[13]:
```
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
11002  5.3903      29.0  6.451477   1.029536      1197.0  2.525316     33.75
16877  6.2210      52.0  6.571429   0.966667       530.0  2.523810     37.60
8838   2.8889      30.0  3.639582   1.055511      1910.0  1.536605     34.08
14080  2.7928      48.0  5.106599   1.071066       893.0  2.266497     32.77
13145  1.6458      52.0  4.292453   1.000000       429.0  2.023585     38.26

       Longitude
11002    -117.80
16877    -122.40
8838     -118.38
```

```
14080    -117.12
13145    -121.51
```

[32]:
```
### train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(
    X,y,test_size=0.33,random_state=42)
```

[33]:
```
from sklearn.tree import DecisionTreeRegressor
## because it's a regression problem
regressor = DecisionTreeRegressor()
```

[34]:
```
regressor.fit(X_train,y_train)
```

[34]:
```
DecisionTreeRegressor()
```

[35]:
```
y_pred = regressor.predict(X_test)
```

[36]:
```
y_pred
```

[36]:
```
array([2.656, 1.127, 0.575, …, 2.218, 1.528, 3.897])
```

[37]:
```
from sklearn.metrics import r2_score
score = r2_score(y_pred,y_test)
```

[38]:
```
score
```

[38]:
```
0.5348569503027532
```

[41]:
```
## Hyperparameter Tunning to get better pridiction
parameter ={
    'criterion':['squared_error','friedman_mse','absolute_error','poisson'],
    'splitter':['best','random'],
    'max_depth':[1,2,3,4,5,6,7,8,10,11,12],
    'max_features':['auto','sqrt','log2']
}
regressor = DecisionTreeRegressor()
```

[42]:
```
#https://scikit-learn.org/stable/modules/model_evaluation.html
import warnings ## to ignore warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import GridSearchCV
regressorcv =␣
 ↪GridSearchCV(regressor,param_grid=parameter,cv=2,scoring='neg_mean_squared_error')
## scoring parameter will be different for regressor, cluster and␣
 ↪classification so it's important to use correct
```

```
## we can check which scoring parameter to use in grid-search-cv documentation␣
 ↪on sklearn site
## we use it to find residuals(error)
```

[43]:
```
regressorcv.fit(X_train,y_train)
## it'll try all the combination and give us best combination for this fit
```

[43]:
```
GridSearchCV(cv=2, estimator=DecisionTreeRegressor(),
             param_grid={'criterion': ['squared_error', 'friedman_mse',
                                       'absolute_error', 'poisson'],
                         'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'splitter': ['best', 'random']},
             scoring='neg_mean_squared_error')
```

[44]:
```
regressorcv.best_params_
```

[44]:
```
{'criterion': 'squared_error',
 'max_depth': 7,
 'max_features': 'auto',
 'splitter': 'best'}
```

[ ]:
```

```

[45]:
```
y_pred = regressorcv.predict(X_test)
```

[46]:
```
r2_score(y_pred,y_test)
```

[46]:
```
0.5132502886920713
```

[ ]:
```

```

[57]:
```
regressor_new = DecisionTreeRegressor(criterion= 'squared_error',
 max_depth= 7,
 max_features= 'auto',
 splitter= 'best')
```

[58]:
```
regressor_new.fit(X_train,y_train)
```

[58]:
```
DecisionTreeRegressor(max_depth=7, max_features='auto')
```

[59]:
```
y_pred = regressor_new.predict(X_test)
```

[60]:
```
r2_score(y_pred,y_test)
```

[60]:
```
0.5125979930862274
```

[ ]:

[52]:
```
regressor_new =␣
↪DecisionTreeRegressor(criterion='absolute_error',max_depth=7,max_features='auto',splitter='
```

[53]:
```
regressor_new.fit(X_train,y_train)
```

[53]:
```
DecisionTreeRegressor(criterion='absolute_error', max_depth=7,
                      max_features='auto')
```

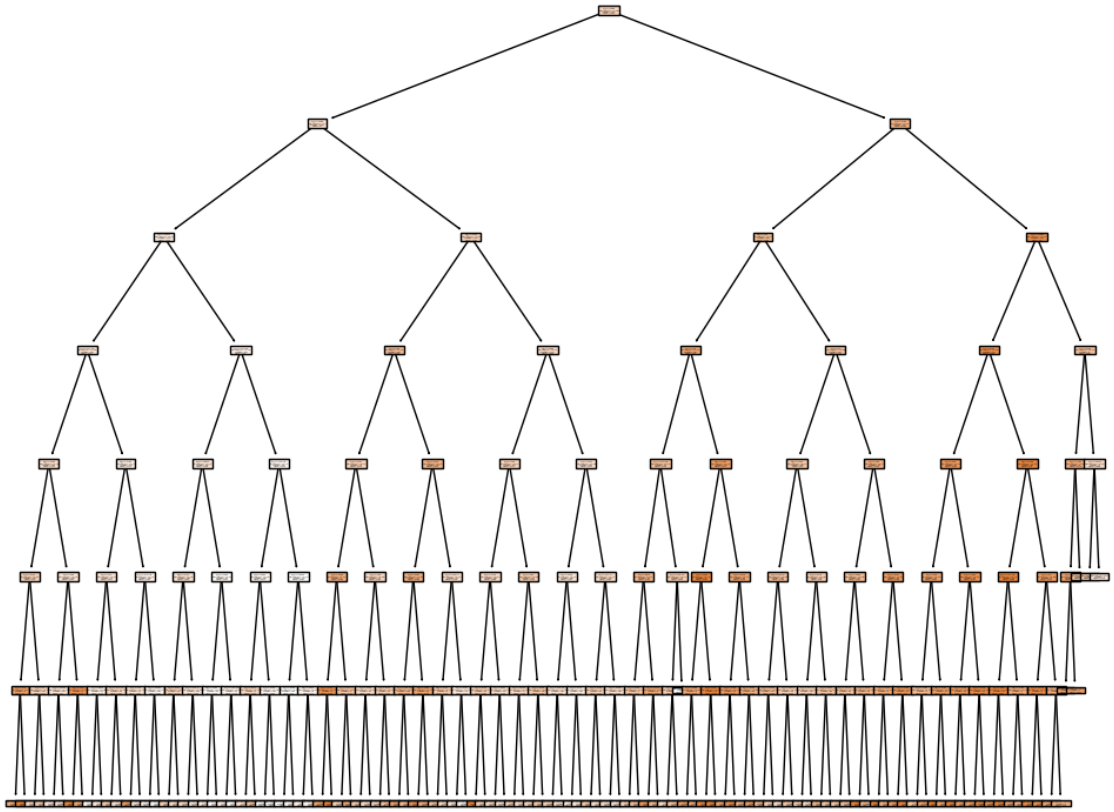
[56]:
```
y_pred = regressor_new.predict(X_test)
r2_score(y_pred,y_test)
```

[56]:
```
0.49413101910219626
```

[ ]:

[62]:
```
%matplotlib inline
```

[64]:
```
from sklearn import tree
plt.figure(figsize=(12,10))
tree.plot_tree(regressor_new, filled=True)
plt.show()
```



[ ]:

[ ]: