LAB 4 - CLASSIFICATION (DECISION TREE)

SUBMITTED BY:

Name : Kavitha.S
Reg no : 21122033
Class : 2MSDS

LAB OVERVIEW:

Perform Classification using Decision Trees.
Demonstrate Multiple Datasets, do the necessary EDA and show various evaluation metrics.

PROBLEM DEFINITION:

To import multiple datasets and perform various exploratory data analysis on it. Split the dataset into train and test. Perform Classification using Decision Trees and evaluate the model.

APPROACH:

Use Pandas to Import the Datasets.
Performing necessary Exploratory Data Analysis. Visualizing the dataset using various plots from matplotlib and seaborn.
Use the train_test_split method available in SCIKIT to split the dataset into Train Dataset and Test Dataset.
Perform classification on the datasets using Decision tree with various parameters.
Finding the accuracy of the model adn evaluation the model based on various evaluation metrics.

Importing the necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

Importing a dataset which contains various attributes about a car based on which the car is being evaluated for its safety.

In [2]:
```
df = pd.read_csv('car_evaluation.csv', header=None)
```

Exploratory data analysis

In [3]:
```
df.head()
```

Out[3]:

	0	1	2	3	4	5	6
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

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

Out[4]:
```
Int64Index([0, 1, 2, 3, 4, 5, 6], dtype='int64')
```

Rename column names

We can see that the dataset does not have proper column names. The columns are merely labelled as 0,1,2.... and so on. WE are assigning them names based on their nature as follows:

In [5]:
```
col_names = ['buying_price', 'maint', 'doors', 'persons', 'lug_boot', 'safet
df.columns = col_names
col_names
```

Out[5]:
```
['buying_price', 'maint', 'doors', 'persons', 'lug_boot', 'safety', 'class']
```

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

Out[6]:

	buying_price	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

In [7]:
```
df.describe()
```

Out[7]:

	buying_price	maint	doors	persons	lug_boot	safety	class
count	1728	1728	1728	1728	1728	1728	1728
unique	4	4	4	3	3	3	4
top	vhigh	vhigh	3	more	small	high	unacc
freq	432	432	432	576	576	576	1210

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   buying_price  1728 non-null   object
 1   maint         1728 non-null   object
 2   doors         1728 non-null   object
 3   persons       1728 non-null   object
 4   lug_boot      1728 non-null   object
 5   safety        1728 non-null   object
 6   class         1728 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

The dataframe contains 7 columns and 1728 entries in each column.

Frequency distribution of values in variables¶

In [9]:
```
for col in col_names:

    print(df[col].value_counts())
```

```
vhigh    432
high     432
low      432
med      432
Name: buying_price, dtype: int64
vhigh    432
high     432
low      432
med      432
Name: maint, dtype: int64
3        432
5more    432
2        432
4        432
Name: doors, dtype: int64
more    576
2       576
4       576
Name: persons, dtype: int64
small    576
big      576
med      576
Name: lug_boot, dtype: int64
high    576
low     576
med     576
Name: safety, dtype: int64
unacc    1210
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

INFERENCE:

There are 7 variables in the dataset. All the variables are of categorical data type.
These are given by buying, maint, doors, persons, lug_boot, safety and class.
Where buying, maint, doors, persons, lug_boot, safety are the feature variables.
Class is the target variable.

Checking for null values in the dataframe

In [10]:
```
plt.figure(figsize=(12,4))
sns.heatmap(df.isnull(),cmap='viridis')
plt.title('Missing value in the dataset');
```

There are no null values

Encode categorical variables

Machine learning algorithms cannot work with categorical data directly, categorical data must be converted to number.
Since the variables are mostly categorical data type, converting them into numerical form using label encoder.
Label encoding refers to transforming the word labels into numerical form so that the algorithms can understand how to operate on them.

In [11]:
```
label_encoder = preprocessing.LabelEncoder()
```

In [12]:
```
list = ['buying_price', 'maint', 'doors', 'persons', 'lug_boot', 'safety', '
for i in list:
    print(df[i].unique())
    df[i]= label_encoder.fit_transform(df[i])
df.head()
```

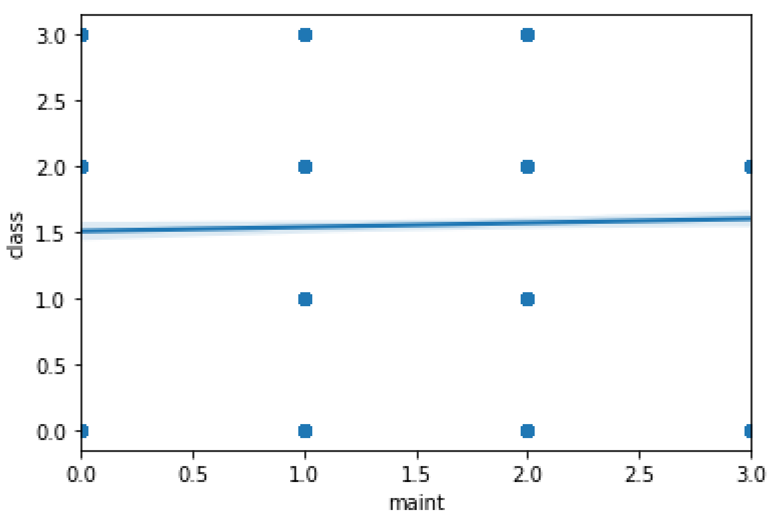
```
['vhigh' 'high' 'med' 'low']
['vhigh' 'high' 'med' 'low']
['2' '3' '4' '5more']
['2' '4' 'more']
['small' 'med' 'big']
['low' 'med' 'high']
['unacc' 'acc' 'vgood' 'good']
```

Out[12]:

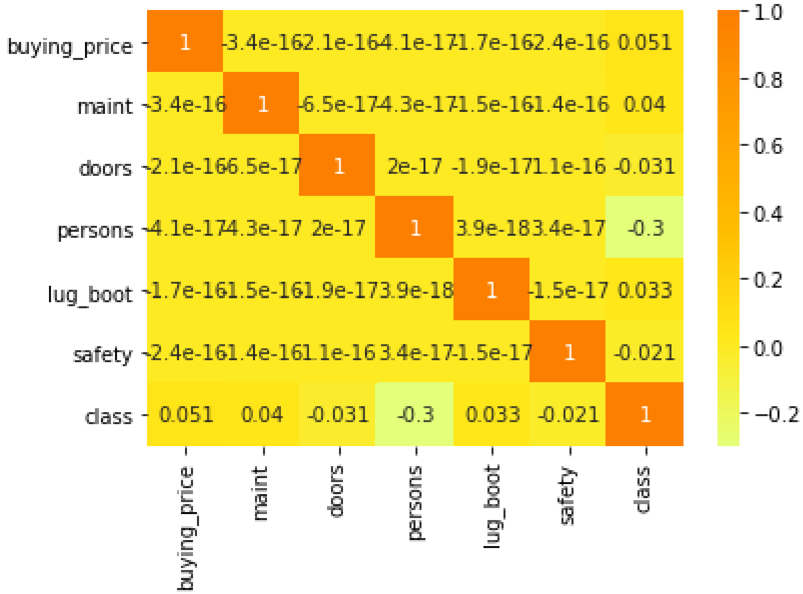
	buying_price	maint	doors	persons	lug_boot	safety	class
0	3	3	0	0	2	1	2
1	3	3	0	0	2	2	2
2	3	3	0	0	2	0	2
3	3	3	0	0	1	1	2
4	3	3	0	0	1	2	2

In [13]:
```
sns.regplot(x = 'maint', y = 'class',data=df)
```

Out[13]: <AxesSubplot:xlabel='maint', ylabel='class'>

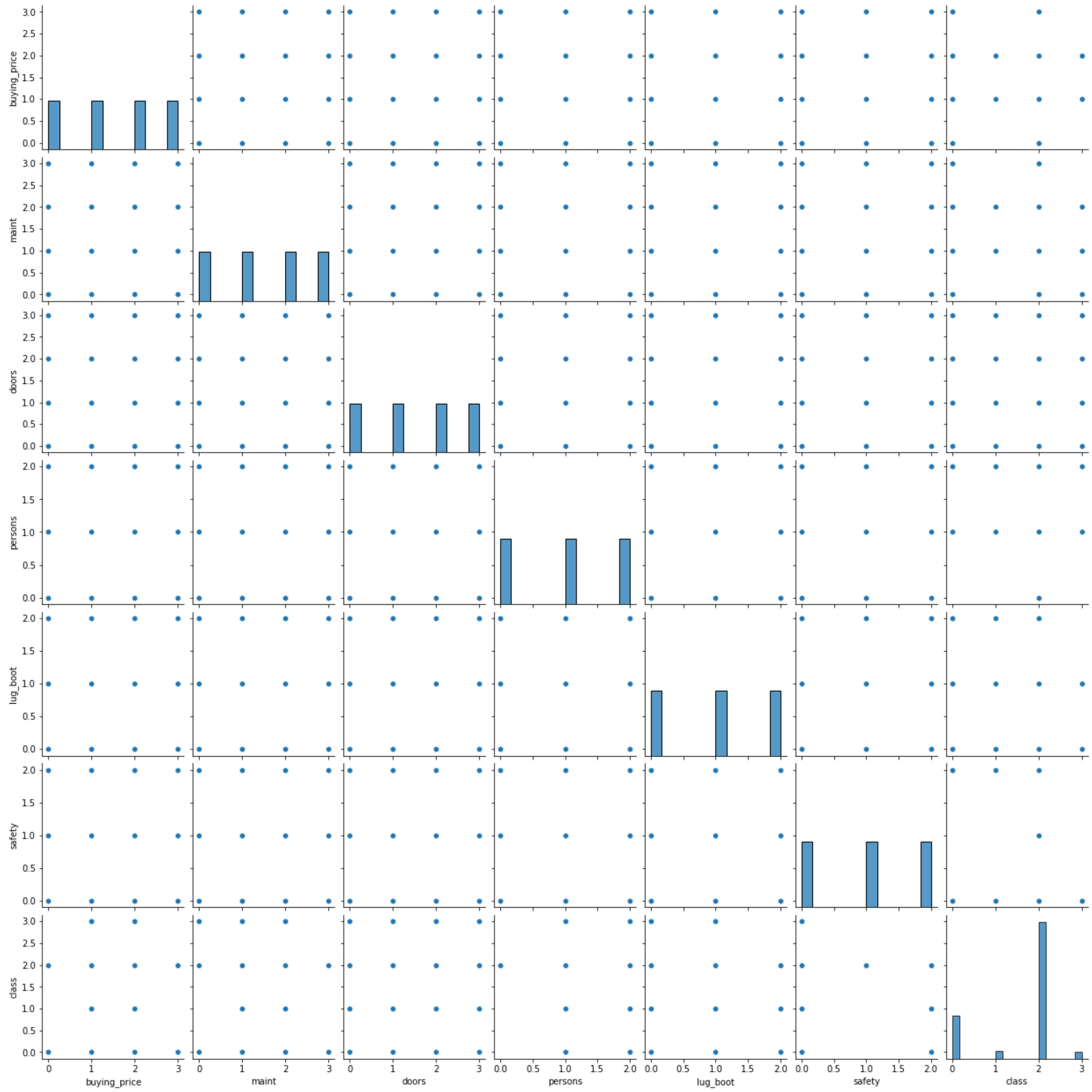


In [14]:
```
corr = df.corr()
sns.heatmap(corr, cmap = 'Wistia', annot= True);
```



In [15]:
```
sns.pairplot(df)
```

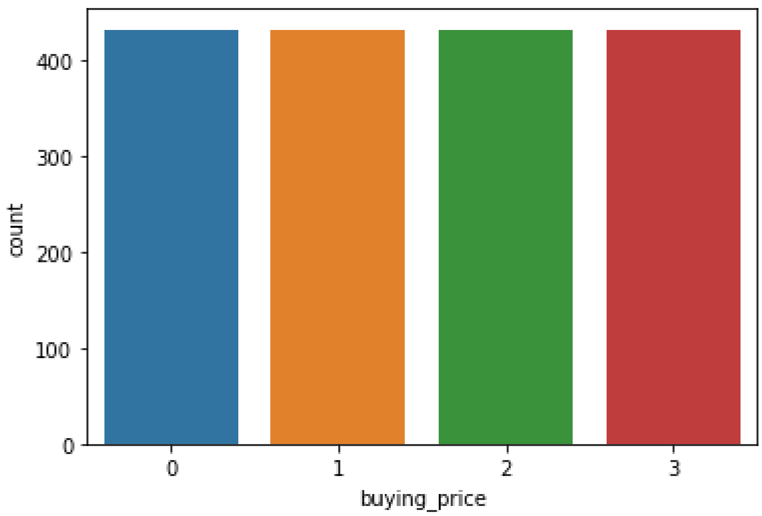
Out[15]: <seaborn.axisgrid.PairGrid at 0x14b32e763a0>



In [16]:
```
sns.countplot("buying_price",data=df)
```

```
C:\Users\SRIDHAR\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureW
arning: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[16]: <AxesSubplot:xlabel='buying_price', ylabel='count'>



In [17]:
```
f= plt.figure(figsize=(12,4))

ax=f.add_subplot(121)
sns.distplot(df['buying_price'],bins=100,color='b',ax=ax)
ax.set_title('Distribution of Buying price')
```

```
C:\Users\SRIDHAR\anaconda3\lib\site-packages\seaborn\distributions.py:2557: Fut
ureWarning: `distplot` is a deprecated function and will be removed in a future
version. Please adapt your code to use either `displot` (a figure-level functio
n with similar flexibility) or `histplot` (an axes-level function for histogram
s).
  warnings.warn(msg, FutureWarning)
```

Out[17]:
```
Text(0.5, 1.0, 'Distribution of Buying price')
```

Declare feature vector and target variable

In [18]:
```
X = df.drop(['class'], axis=1)

y = df['class']
```

Split data into separate training and test set

In [19]:
```
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,
```

In [20]:
```
X_train.shape, X_test.shape
```

Out[20]:
```
((1157, 6), (571, 6))
```

In [21]:
```
X_train.head()
```

Out[21]:

	buying_price	maint	doors	persons	lug_boot	safety
48	3	3	1	2	1	1
468	0	3	1	1	2	1
155	3	0	1	2	2	0
1721	1	1	3	2	2	0
1208	2	1	0	2	2	0

In [22]:
```
y_train.head()
```

Out[22]:
```
48      2
468     2
155     2
1721    1
1208    2
Name: class, dtype: int32
```

Decision Tree Classifier with criterion gini index

In [23]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [24]:
```
# instantiate the DecisionTreeClassifier model with criterion gini index
clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_stat

# fit the model
clf_gini.fit(X_train, y_train)
```

Out[24]:
```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

Predict the Test set results with criterion gini index

In [25]:
```
y_pred_gini = clf_gini.predict(X_test)
```

Check accuracy score

In [26]:
```
print('Model accuracy score with criterion gini index: {0:0.4f}'. format(acc
```

```
Model accuracy score with criterion gini index: 0.7653
```

Compare the train-set and test-set accuracy

In [27]:
```
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
```

Out[27]:
```
array([2, 2, 0, ..., 0, 2, 2])
```

In [28]:
```
print('Training-set accuracy score: {:f}'. format(accuracy_score(y_train, y_
```

```
Training-set accuracy score: 0.774417
```

Print the scores on training and test set

In [29]:
```
print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
```

```
Training set score: 0.7744
Test set score: 0.7653
```

EVALUATION METRICS

In [30]:
```
print('Mean Absolute Error : ', metrics.mean_absolute_error(y_test, y_pred_g
print('Mean Squared Error : ', metrics.mean_squared_error(y_test, y_pred_gin
print('Root Mean Squared Error : ', np.sqrt(metrics.mean_squared_error(y_tes
```

```
Mean Absolute Error :  0.47810858143607704
Mean Squared Error :  1.052539404553415
Root Mean Squared Error :  1.0259334308586572
```

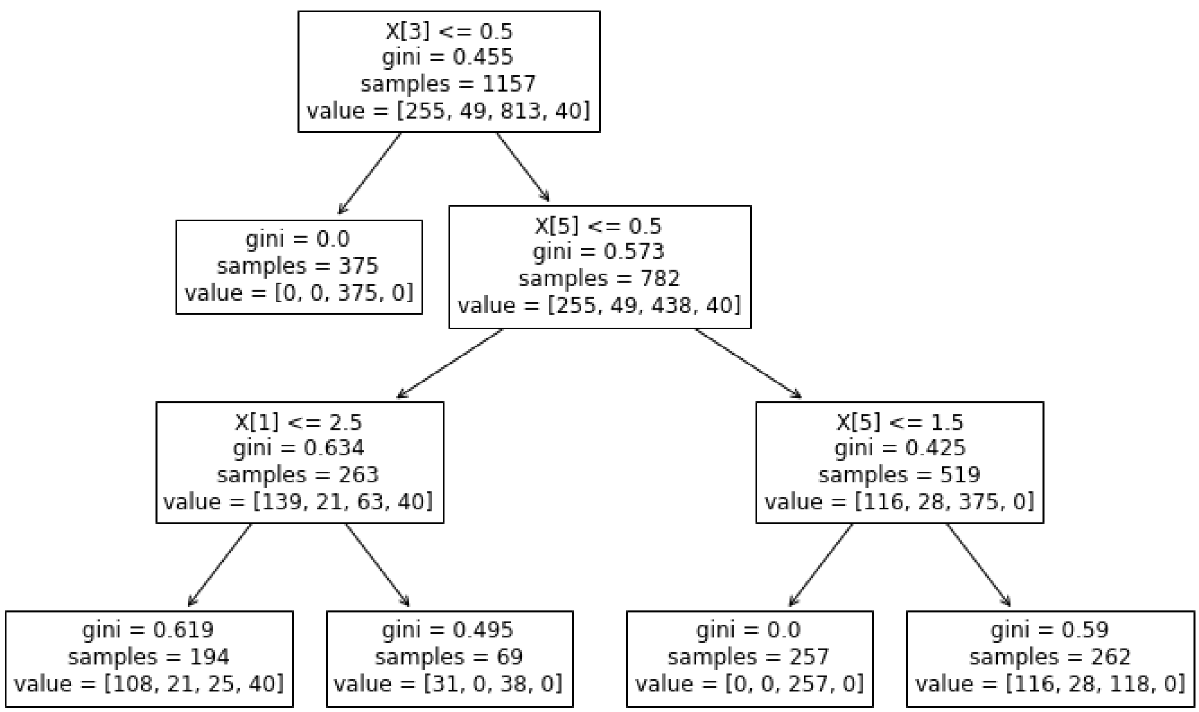
Visualize decision-trees

In [31]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_gini.fit(X_train, y_train))
```

Out[31]:
```
[Text(251.10000000000002, 380.52, 'X[3] <= 0.5\ngini = 0.455\nsamples = 1157\nv
alue = [255, 49, 813, 40]'),
 Text(167.4, 271.8, 'gini = 0.0\nsamples = 375\nvalue = [0, 0, 375, 0]'),
 Text(334.8, 271.8, 'X[5] <= 0.5\ngini = 0.573\nsamples = 782\nvalue = [255, 4
9, 438, 40]'),
 Text(167.4, 163.07999999999998, 'X[1] <= 2.5\ngini = 0.634\nsamples = 263\nval
ue = [139, 21, 63, 40]'),
 Text(83.7, 54.360000000000014, 'gini = 0.619\nsamples = 194\nvalue = [108, 21,
25, 40]'),
 Text(251.10000000000002, 54.360000000000014, 'gini = 0.495\nsamples = 69\nvalu
e = [31, 0, 38, 0]'),
 Text(502.20000000000005, 163.07999999999998, 'X[5] <= 1.5\ngini = 0.425\nsampl
es = 519\nvalue = [116, 28, 375, 0]'),
 Text(418.5, 54.360000000000014, 'gini = 0.0\nsamples = 257\nvalue = [0, 0, 25
7, 0]'),
 Text(585.9, 54.360000000000014, 'gini = 0.59\nsamples = 262\nvalue = [116, 28,
118, 0]')]
```



Decision Tree Classifier with criterion entropy

In [32]:
```
clf_en = DecisionTreeClassifier(criterion='entropy')
```

In [33]:
```
clf_en.fit(X_train, y_train)
```

Out[33]:
```
DecisionTreeClassifier(criterion='entropy')
```

Predict the Test set results and Check accuracy score with criterion entropy

In [34]:
```
y_pred_en = clf_en.predict(X_test)
print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accura
```

```
Model accuracy score with criterion entropy: 0.9720
```

Compare the train-set and test-set score

In [35]:
```
y_pred_train_en = clf_en.predict(X_train)

y_pred_train_en
```

Out[35]:
```
array([2, 2, 2, ..., 0, 2, 0])
```

In [36]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train
```

```
Training-set accuracy score: 1.0000
```

Print the scores on training and test set

In [37]:
```
print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.9720
```

INFERENCE:

Now, based on the above analysis we can conclude that our classification model accuracy is very good. Our model is doing a very good job in terms of predicting the class labels. In case of classification with criterion entropy, The training accuracy score is 1 showing the excellent training of the model on the training data, because we dint mention any parameter and the dataset is very small, the model trained well on all the data. Whereas in the Classification with criterion gingi index the training accuracy was a bit less,because the max_depth was 3. Showing the the branching stooped at 3rd step and the model has been trained until only that.

Confusion matrix

In [38]:
```
cm = confusion_matrix(y_test, y_pred_en)

print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

 [[118   9   2   0]
 [  0  20   0   0]
 [  2   0 395   0]
 [  3   0   0  22]]
```

In [39]:
```
print(classification_report(y_test, y_pred_en))
```

```
              precision    recall  f1-score   support

           0       0.96      0.91      0.94       129
           1       0.69      1.00      0.82        20
           2       0.99      0.99      0.99       397
           3       1.00      0.88      0.94        25

    accuracy                           0.97       571
   macro avg       0.91      0.95      0.92       571
weighted avg       0.98      0.97      0.97       571
```

EVALUATION METRICS

In [41]:
```
print('Mean Absolute Error : ', metrics.mean_absolute_error(y_test, y_pred_e
print('Mean Squared Error : ', metrics.mean_squared_error(y_test, y_pred_en)
print('Root Mean Squared Error : ', np.sqrt(metrics.mean_squared_error(y_tes
```

```
Mean Absolute Error :  0.04553415061295972
Mean Squared Error :  0.09106830122591944
Root Mean Squared Error :  0.30177524952507195
```

Visualize decision-trees

In [42]:
```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_en.fit(X_train, y_train))
```

Out[42]:
```
[Text(357.78319672131147, 420.384, 'X[3] <= 0.5\nentropy = 1.2\nsamples = 1157
\nvalue = [255, 49, 813, 40]'),
 Text(346.8061475409836, 391.392, 'entropy = 0.0\nsamples = 375\nvalue = [0, 0,
375, 0]'),
 Text(368.76024590163934, 391.392, 'X[5] <= 0.5\nentropy = 1.465\nsamples = 782
\nvalue = [255, 49, 438, 40]'),
 Text(198.6159836065574, 362.4, 'X[1] <= 2.5\nentropy = 1.684\nsamples = 263\nv
alue = [139, 21, 63, 40]'),
 Text(111.82868852459016, 333.408, 'X[0] <= 0.5\nentropy = 1.668\nsamples = 194
\nvalue = [108, 21, 25, 40]'),
 Text(32.93114754098361, 304.416, 'X[2] <= 0.5\nentropy = 0.232\nsamples = 53\n
value = [51, 0, 2, 0]'),
 Text(21.95409836065574, 275.424, 'X[4] <= 1.5\nentropy = 0.65\nsamples = 12\nv
alue = [10, 0, 2, 0]'),
 Text(10.97704918032787, 246.432, 'entropy = 0.0\nsamples = 8\nvalue = [8, 0,
0, 0]'),
 Text(32.93114754098361, 246.432, 'X[3] <= 1.5\nentropy = 1.0\nsamples = 4\nval
ue = [2, 0, 2, 0]'),
 Text(21.95409836065574, 217.44, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0, 0,
0]'),
 Text(43.90819672131148, 217.44, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0, 2,
0]'),
 Text(43.90819672131148, 275.424, 'entropy = 0.0\nsamples = 41\nvalue = [41, 0,
0, 0]'),
 Text(190.72622950819672, 304.416, 'X[0] <= 2.5\nentropy = 1.88\nsamples = 141
\nvalue = [57, 21, 23, 40]'),
 Text(145.44590163934427, 275.424, 'X[4] <= 1.5\nentropy = 1.753\nsamples = 92
\nvalue = [26, 21, 5, 40]'),
 Text(87.81639344262295, 246.432, 'X[1] <= 0.5\nentropy = 1.196\nsamples = 59\n
value = [13, 6, 0, 40]'),
 Text(65.86229508196722, 217.44, 'X[0] <= 1.5\nentropy = 0.988\nsamples = 23\nv
alue = [13, 0, 0, 10]'),
 Text(54.885245901639344, 188.44799999999998, 'X[4] <= 0.5\nentropy = 0.779\nsa
mples = 13\nvalue = [3, 0, 0, 10]'),
 Text(43.90819672131148, 159.45599999999996, 'entropy = 0.0\nsamples = 8\nvalue
= [0, 0, 0, 8]'),
 Text(65.86229508196722, 159.45599999999996, 'X[2] <= 2.0\nentropy = 0.971\nsam
ples = 5\nvalue = [3, 0, 0, 2]'),
 Text(54.885245901639344, 130.464, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0,
0, 0]'),
 Text(76.83934426229509, 130.464, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0,
0, 2]'),
 Text(76.83934426229509, 188.44799999999998, 'entropy = 0.0\nsamples = 10\nvalu
e = [10, 0, 0, 0]'),
 Text(109.77049180327869, 217.44, 'X[4] <= 0.5\nentropy = 0.65\nsamples = 36\nv
alue = [0, 6, 0, 30]'),
 Text(98.79344262295082, 188.44799999999998, 'entropy = 0.0\nsamples = 19\nvalu
e = [0, 0, 0, 19]'),
 Text(120.74754098360656, 188.44799999999998, 'X[2] <= 0.5\nentropy = 0.937\nsa
mples = 17\nvalue = [0, 6, 0, 11]'),
```

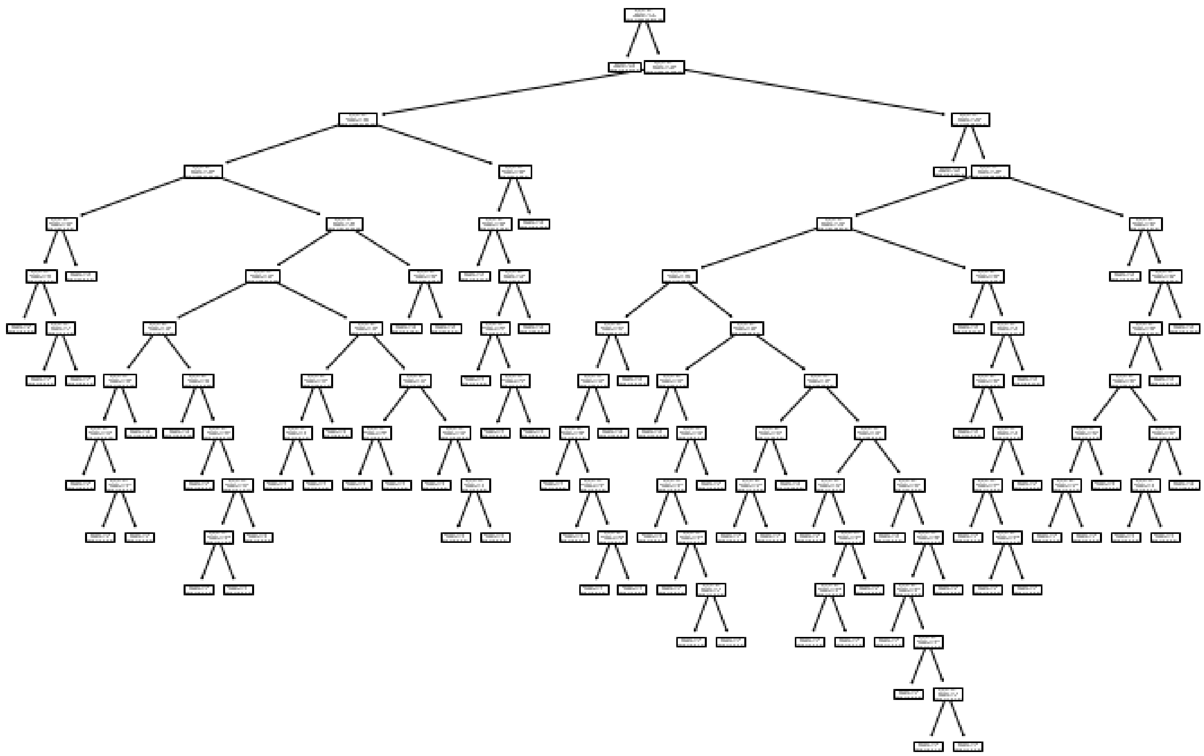
```
 Text(109.77049180327869, 159.45599999999996, 'entropy = 0.0\nsamples = 5\nvalu
e = [0, 5, 0, 0]'),
 Text(131.72459016393444, 159.45599999999996, 'X[2] <= 1.5\nentropy = 0.414\nsa
mples = 12\nvalue = [0, 1, 0, 11]'),
 Text(120.74754098360656, 130.464, 'X[3] <= 1.5\nentropy = 0.918\nsamples = 3\n
value = [0, 1, 0, 2]'),
 Text(109.77049180327869, 101.47199999999998, 'entropy = 0.0\nsamples = 1\nvalu
e = [0, 1, 0, 0]'),
 Text(131.72459016393444, 101.47199999999998, 'entropy = 0.0\nsamples = 2\nvalu
e = [0, 0, 0, 2]'),
 Text(142.7016393442623, 130.464, 'entropy = 0.0\nsamples = 9\nvalue = [0, 0,
0, 9]'),
 Text(203.07540983606557, 246.432, 'X[1] <= 0.5\nentropy = 1.459\nsamples = 33
\nvalue = [13, 15, 5, 0]'),
 Text(175.6327868852459, 217.44, 'X[2] <= 0.5\nentropy = 0.439\nsamples = 11\nv
alue = [10, 0, 1, 0]'),
 Text(164.65573770491804, 188.44799999999998, 'X[3] <= 1.5\nentropy = 1.0\nsamp
les = 2\nvalue = [1, 0, 1, 0]'),
 Text(153.67868852459017, 159.45599999999996, 'entropy = 0.0\nsamples = 1\nvalu
e = [1, 0, 0, 0]'),
 Text(175.6327868852459, 159.45599999999996, 'entropy = 0.0\nsamples = 1\nvalue
= [0, 0, 1, 0]'),
 Text(186.60983606557377, 188.44799999999998, 'entropy = 0.0\nsamples = 9\nvalu
e = [9, 0, 0, 0]'),
 Text(230.51803278688524, 217.44, 'X[2] <= 0.5\nentropy = 1.216\nsamples = 22\n
value = [3, 15, 4, 0]'),
 Text(208.5639344262295, 188.44799999999998, 'X[3] <= 1.5\nentropy = 0.985\nsam
ples = 7\nvalue = [0, 3, 4, 0]'),
 Text(197.58688524590164, 159.45599999999996, 'entropy = 0.0\nsamples = 3\nvalu
e = [0, 3, 0, 0]'),
 Text(219.54098360655738, 159.45599999999996, 'entropy = 0.0\nsamples = 4\nvalu
e = [0, 0, 4, 0]'),
 Text(252.47213114754098, 188.44799999999998, 'X[0] <= 1.5\nentropy = 0.722\nsa
mples = 15\nvalue = [3, 12, 0, 0]'),
 Text(241.4950819672131, 159.45599999999996, 'entropy = 0.0\nsamples = 9\nvalue
= [0, 9, 0, 0]'),
 Text(263.4491803278689, 159.45599999999996, 'X[1] <= 1.5\nentropy = 1.0\nsampl
es = 6\nvalue = [3, 3, 0, 0]'),
 Text(252.47213114754098, 130.464, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3,
0, 0]'),
 Text(274.42622950819674, 130.464, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0,
0, 0]'),
 Text(236.00655737704918, 275.424, 'X[1] <= 0.5\nentropy = 0.949\nsamples = 49
\nvalue = [31, 0, 18, 0]'),
 Text(225.0295081967213, 246.432, 'entropy = 0.0\nsamples = 18\nvalue = [0, 0,
18, 0]'),
 Text(246.98360655737704, 246.432, 'entropy = 0.0\nsamples = 31\nvalue = [31,
0, 0, 0]'),
 Text(285.4032786885246, 333.408, 'X[0] <= 2.5\nentropy = 0.993\nsamples = 69\n
value = [31, 0, 38, 0]'),
 Text(274.42622950819674, 304.416, 'X[0] <= 0.5\nentropy = 0.958\nsamples = 50
\nvalue = [31, 0, 19, 0]'),
 Text(263.4491803278689, 275.424, 'entropy = 0.0\nsamples = 17\nvalue = [0, 0,
17, 0]'),
 Text(285.4032786885246, 275.424, 'X[2] <= 0.5\nentropy = 0.33\nsamples = 33\nv
alue = [31, 0, 2, 0]'),
 Text(274.42622950819674, 246.432, 'X[4] <= 1.5\nentropy = 0.863\nsamples = 7\n
```

```
value = [5, 0, 2, 0]'),
 Text(263.4491803278689, 217.44, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0, 0,
0]'),
 Text(285.4032786885246, 217.44, 'X[3] <= 1.5\nentropy = 0.918\nsamples = 3\nva
lue = [1, 0, 2, 0]'),
 Text(274.42622950819674, 188.44799999999998, 'entropy = 0.0\nsamples = 1\nvalu
e = [1, 0, 0, 0]'),
 Text(296.3803278688525, 188.44799999999998, 'entropy = 0.0\nsamples = 2\nvalue
= [0, 0, 2, 0]'),
 Text(296.3803278688525, 246.432, 'entropy = 0.0\nsamples = 26\nvalue = [26, 0,
0, 0]'),
 Text(296.3803278688525, 304.416, 'entropy = 0.0\nsamples = 19\nvalue = [0, 0,
19, 0]'),
 Text(538.9045081967213, 362.4, 'X[5] <= 1.5\nentropy = 1.049\nsamples = 519\nv
alue = [116, 28, 375, 0]'),
 Text(527.9274590163934, 333.408, 'entropy = 0.0\nsamples = 257\nvalue = [0, 0,
257, 0]'),
 Text(549.8815573770491, 333.408, 'X[4] <= 1.5\nentropy = 1.383\nsamples = 262
\nvalue = [116, 28, 118, 0]'),
 Text(463.094262295082, 304.416, 'X[0] <= 2.5\nentropy = 1.432\nsamples = 170\n
value = [91, 28, 51, 0]'),
 Text(377.3360655737705, 275.424, 'X[0] <= 0.5\nentropy = 1.381\nsamples = 125
\nvalue = [74, 28, 23, 0]'),
 Text(340.28852459016395, 246.432, 'X[1] <= 2.5\nentropy = 0.974\nsamples = 42
\nvalue = [25, 0, 17, 0]'),
 Text(329.3114754098361, 217.44, 'X[2] <= 1.5\nentropy = 0.758\nsamples = 32\nv
alue = [25, 0, 7, 0]'),
 Text(318.3344262295082, 188.44799999999998, 'X[4] <= 0.5\nentropy = 0.989\nsam
ples = 16\nvalue = [9, 0, 7, 0]'),
 Text(307.35737704918034, 159.45599999999996, 'entropy = 0.0\nsamples = 7\nvalu
e = [7, 0, 0, 0]'),
 Text(329.3114754098361, 159.45599999999996, 'X[3] <= 1.5\nentropy = 0.764\nsam
ples = 9\nvalue = [2, 0, 7, 0]'),
 Text(318.3344262295082, 130.464, 'entropy = 0.0\nsamples = 6\nvalue = [0, 0,
6, 0]'),
 Text(340.28852459016395, 130.464, 'X[2] <= 0.5\nentropy = 0.918\nsamples = 3\n
value = [2, 0, 1, 0]'),
 Text(329.3114754098361, 101.47199999999998, 'entropy = 0.0\nsamples = 1\nvalue
= [0, 0, 1, 0]'),
 Text(351.2655737704918, 101.47199999999998, 'entropy = 0.0\nsamples = 2\nvalue
= [2, 0, 0, 0]'),
 Text(340.28852459016395, 188.44799999999998, 'entropy = 0.0\nsamples = 16\nval
ue = [16, 0, 0, 0]'),
 Text(351.2655737704918, 217.44, 'entropy = 0.0\nsamples = 10\nvalue = [0, 0, 1
0, 0]'),
 Text(414.38360655737705, 246.432, 'X[1] <= 0.5\nentropy = 1.252\nsamples = 83
\nvalue = [49, 28, 6, 0]'),
 Text(373.21967213114755, 217.44, 'X[4] <= 0.5\nentropy = 0.439\nsamples = 22\n
value = [20, 0, 2, 0]'),
 Text(362.2426229508197, 188.44799999999998, 'entropy = 0.0\nsamples = 12\nvalu
e = [12, 0, 0, 0]'),
 Text(384.1967213114754, 188.44799999999998, 'X[2] <= 1.5\nentropy = 0.722\nsam
ples = 10\nvalue = [8, 0, 2, 0]'),
 Text(373.21967213114755, 159.45599999999996, 'X[0] <= 1.5\nentropy = 0.971\nsa
mples = 5\nvalue = [3, 0, 2, 0]'),
 Text(362.2426229508197, 130.464, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0,
0, 0]'),
```

```
 Text(384.1967213114754, 130.464, 'X[3] <= 1.5\nentropy = 0.918\nsamples = 3\nv
alue = [1, 0, 2, 0]'),
 Text(373.21967213114755, 101.47199999999998, 'entropy = 0.0\nsamples = 1\nvalu
e = [0, 0, 1, 0]'),
 Text(395.1737704918033, 101.47199999999998, 'X[2] <= 0.5\nentropy = 1.0\nsampl
es = 2\nvalue = [1, 0, 1, 0]'),
 Text(384.1967213114754, 72.47999999999996, 'entropy = 0.0\nsamples = 1\nvalue
= [0, 0, 1, 0]'),
 Text(406.15081967213115, 72.47999999999996, 'entropy = 0.0\nsamples = 1\nvalue
= [1, 0, 0, 0]'),
 Text(395.1737704918033, 159.45599999999996, 'entropy = 0.0\nsamples = 5\nvalue
= [5, 0, 0, 0]'),
 Text(455.54754098360655, 217.44, 'X[1] <= 1.5\nentropy = 1.283\nsamples = 61\n
value = [29, 28, 4, 0]'),
 Text(428.1049180327869, 188.44799999999998, 'X[2] <= 0.5\nentropy = 0.276\nsam
ples = 21\nvalue = [1, 20, 0, 0]'),
 Text(417.127868852459, 159.45599999999996, 'X[4] <= 0.5\nentropy = 0.722\nsamp
les = 5\nvalue = [1, 4, 0, 0]'),
 Text(406.15081967213115, 130.464, 'entropy = 0.0\nsamples = 4\nvalue = [0, 4,
0, 0]'),
 Text(428.1049180327869, 130.464, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0,
0, 0]'),
 Text(439.08196721311475, 159.45599999999996, 'entropy = 0.0\nsamples = 16\nval
ue = [0, 16, 0, 0]'),
 Text(482.9901639344262, 188.44799999999998, 'X[0] <= 1.5\nentropy = 1.157\nsam
ples = 40\nvalue = [28, 8, 4, 0]'),
 Text(461.0360655737705, 159.45599999999996, 'X[1] <= 2.5\nentropy = 1.272\nsam
ples = 16\nvalue = [7, 8, 1, 0]'),
 Text(450.0590163934426, 130.464, 'entropy = 0.0\nsamples = 8\nvalue = [0, 8,
0, 0]'),
 Text(472.01311475409835, 130.464, 'X[2] <= 1.0\nentropy = 0.544\nsamples = 8\n
value = [7, 0, 1, 0]'),
 Text(461.0360655737705, 101.47199999999998, 'X[4] <= 0.5\nentropy = 0.918\nsam
ples = 3\nvalue = [2, 0, 1, 0]'),
 Text(450.0590163934426, 72.47999999999996, 'entropy = 0.0\nsamples = 2\nvalue
= [2, 0, 0, 0]'),
 Text(472.01311475409835, 72.47999999999996, 'entropy = 0.0\nsamples = 1\nvalue
= [0, 0, 1, 0]'),
 Text(482.9901639344262, 101.47199999999998, 'entropy = 0.0\nsamples = 5\nvalue
= [5, 0, 0, 0]'),
 Text(504.94426229508196, 159.45599999999996, 'X[1] <= 2.5\nentropy = 0.544\nsa
mples = 24\nvalue = [21, 0, 3, 0]'),
 Text(493.9672131147541, 130.464, 'entropy = 0.0\nsamples = 14\nvalue = [14, 0,
0, 0]'),
 Text(515.9213114754099, 130.464, 'X[2] <= 1.5\nentropy = 0.881\nsamples = 10\n
value = [7, 0, 3, 0]'),
 Text(504.94426229508196, 101.47199999999998, 'X[4] <= 0.5\nentropy = 0.971\nsa
mples = 5\nvalue = [2, 0, 3, 0]'),
 Text(493.9672131147541, 72.47999999999996, 'entropy = 0.0\nsamples = 1\nvalue
= [1, 0, 0, 0]'),
 Text(515.9213114754099, 72.47999999999996, 'X[2] <= 0.5\nentropy = 0.811\nsamp
les = 4\nvalue = [1, 0, 3, 0]'),
 Text(504.94426229508196, 43.488, 'entropy = 0.0\nsamples = 2\nvalue = [0, 0,
2, 0]'),
 Text(526.8983606557377, 43.488, 'X[3] <= 1.5\nentropy = 1.0\nsamples = 2\nvalu
e = [1, 0, 1, 0]'),
 Text(515.9213114754099, 14.495999999999981, 'entropy = 0.0\nsamples = 1\nvalue
```

```
= [0, 0, 1, 0]'),
 Text(537.8754098360656, 14.495999999999981, 'entropy = 0.0\nsamples = 1\nvalue
= [1, 0, 0, 0]'),
 Text(526.8983606557377, 101.47199999999998, 'entropy = 0.0\nsamples = 5\nvalue
= [5, 0, 0, 0]'),
 Text(548.8524590163935, 275.424, 'X[1] <= 0.5\nentropy = 0.956\nsamples = 45\n
value = [17, 0, 28, 0]'),
 Text(537.8754098360656, 246.432, 'entropy = 0.0\nsamples = 11\nvalue = [0, 0,
11, 0]'),
 Text(559.8295081967213, 246.432, 'X[1] <= 2.5\nentropy = 1.0\nsamples = 34\nva
lue = [17, 0, 17, 0]'),
 Text(548.8524590163935, 217.44, 'X[4] <= 0.5\nentropy = 0.828\nsamples = 23\nv
alue = [17, 0, 6, 0]'),
 Text(537.8754098360656, 188.44799999999998, 'entropy = 0.0\nsamples = 11\nvalu
e = [11, 0, 0, 0]'),
 Text(559.8295081967213, 188.44799999999998, 'X[2] <= 1.5\nentropy = 1.0\nsampl
es = 12\nvalue = [6, 0, 6, 0]'),
 Text(548.8524590163935, 159.45599999999996, 'X[3] <= 1.5\nentropy = 0.592\nsam
ples = 7\nvalue = [1, 0, 6, 0]'),
 Text(537.8754098360656, 130.464, 'entropy = 0.0\nsamples = 4\nvalue = [0, 0,
4, 0]'),
 Text(559.8295081967213, 130.464, 'X[2] <= 0.5\nentropy = 0.918\nsamples = 3\nv
alue = [1, 0, 2, 0]'),
 Text(548.8524590163935, 101.47199999999998, 'entropy = 0.0\nsamples = 2\nvalue
= [0, 0, 2, 0]'),
 Text(570.8065573770492, 101.47199999999998, 'entropy = 0.0\nsamples = 1\nvalue
= [1, 0, 0, 0]'),
 Text(570.8065573770492, 159.45599999999996, 'entropy = 0.0\nsamples = 5\nvalue
= [5, 0, 0, 0]'),
 Text(570.8065573770492, 217.44, 'entropy = 0.0\nsamples = 11\nvalue = [0, 0, 1
1, 0]'),
 Text(636.6688524590164, 304.416, 'X[0] <= 0.5\nentropy = 0.844\nsamples = 92\n
value = [25, 0, 67, 0]'),
 Text(625.6918032786886, 275.424, 'entropy = 0.0\nsamples = 24\nvalue = [0, 0,
24, 0]'),
 Text(647.6459016393443, 275.424, 'X[0] <= 2.5\nentropy = 0.949\nsamples = 68\n
value = [25, 0, 43, 0]'),
 Text(636.6688524590164, 246.432, 'X[1] <= 2.5\nentropy = 0.995\nsamples = 46\n
value = [25, 0, 21, 0]'),
 Text(625.6918032786886, 217.44, 'X[1] <= 0.5\nentropy = 0.834\nsamples = 34\nv
alue = [25, 0, 9, 0]'),
 Text(603.7377049180328, 188.44799999999998, 'X[0] <= 1.5\nentropy = 0.971\nsam
ples = 10\nvalue = [4, 0, 6, 0]'),
 Text(592.760655737705, 159.45599999999996, 'X[2] <= 0.5\nentropy = 0.722\nsamp
les = 5\nvalue = [4, 0, 1, 0]'),
 Text(581.7836065573771, 130.464, 'entropy = 0.0\nsamples = 1\nvalue = [0, 0,
1, 0]'),
 Text(603.7377049180328, 130.464, 'entropy = 0.0\nsamples = 4\nvalue = [4, 0,
0, 0]'),
 Text(614.7147540983607, 159.45599999999996, 'entropy = 0.0\nsamples = 5\nvalue
= [0, 0, 5, 0]'),
 Text(647.6459016393443, 188.44799999999998, 'X[2] <= 0.5\nentropy = 0.544\nsam
ples = 24\nvalue = [21, 0, 3, 0]'),
 Text(636.6688524590164, 159.45599999999996, 'X[3] <= 1.5\nentropy = 1.0\nsampl
es = 6\nvalue = [3, 0, 3, 0]'),
 Text(625.6918032786886, 130.464, 'entropy = 0.0\nsamples = 3\nvalue = [3, 0,
0, 0]'),
```

```
 Text(647.6459016393443, 130.464, 'entropy = 0.0\nsamples = 3\nvalue = [0, 0,
3, 0]'),
 Text(658.6229508196722, 159.45599999999996, 'entropy = 0.0\nsamples = 18\nvalu
e = [18, 0, 0, 0]'),
 Text(647.6459016393443, 217.44, 'entropy = 0.0\nsamples = 12\nvalue = [0, 0, 1
2, 0]'),
 Text(658.6229508196722, 246.432, 'entropy = 0.0\nsamples = 22\nvalue = [0, 0,
22, 0]')]
```



IMPORTING THE SECOND DATASET

We are using the wine dataset which is already present in the sklearn.

In [43]:
```
from sklearn import datasets
from sklearn.datasets import load_wine
```

In [44]:
```
wine = datasets.load_wine(as_frame = True)
```

Declare feature vector and target variable

data_wine are the feature variables
target is the target variable

In [45]:
```
data_wine = wine['data']
target = wine['target']
data_wine.head()
```

Out[45]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	

EXPLORATORY DATA ANALYSIS

In [46]:
```
target.head()
```

Out[46]:
```
0    0
1    0
2    0
3    0
4    0
Name: target, dtype: int32
```

In [47]:
```
data_wine.describe()
```

Out[47]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.00000(
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.02927(
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.99885(
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.34000(
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.20500(
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.13500(
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.87500(
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.08000(

In [48]:
```
target.describe()
```

Out[48]:
```
count    178.000000
mean       0.938202
std        0.775035
min        0.000000
25%        0.000000
50%        1.000000
75%        2.000000
max        2.000000
Name: target, dtype: float64
```

In [49]:
```
data_wine.isnull().sum()
```

Out[49]:
```
alcohol                         0
malic_acid                      0
ash                             0
alcalinity_of_ash               0
magnesium                       0
total_phenols                   0
flavanoids                      0
nonflavanoid_phenols            0
proanthocyanins                 0
color_intensity                 0
hue                             0
od280/od315_of_diluted_wines    0
proline                         0
dtype: int64
```

Split data into separate training and test set

In [50]:
```
X_train, X_test, y_train, y_test = train_test_split(data_wine, target, test_
```

In [51]:
```
print('X train shape=', X_train.shape)
print('X test shape=', X_test.shape)
print('y train shape=', y_train.shape)
print('y test shape=', y_test.shape)
```

```
X train shape= (106, 13)
X test shape= (72, 13)
y train shape= (106,)
y test shape= (72,)
```

Decision Tree Classifier with criterion gini index

In [52]:
```
clf_gini = DecisionTreeClassifier(criterion='gini')
clf_gini.fit(X_train, y_train)
```

Out[52]:
```
DecisionTreeClassifier()
```

Predict the Test set results and Check accuracy score

In [53]:
```
y_pred_gini = clf_gini.predict(X_test)
```

In [54]:
```
print('Model accuracy score with criterion gini index: {0:0.4f}'. format(acc
```

```
Model accuracy score with criterion gini index: 0.9306
```

Compare the train-set and test-set accuracy

In [55]:
```
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
```

Out[55]:
```
array([2, 2, 1, 1, 0, 2, 0, 1, 2, 0, 1, 0, 2, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 2, 1, 1, 1, 1, 0, 0, 0, 2, 0, 1, 2, 2, 0, 1, 0, 1, 1, 0, 2, 1,
       1, 2, 2, 1, 1, 1, 2, 2, 1, 0, 1, 0, 2, 1, 1, 0, 0, 1, 0, 0, 0, 2,
       0, 2, 2, 0, 1, 1, 2, 0, 1, 1, 0, 0, 0, 1, 1, 0, 2, 2, 1, 1, 1, 0,
       2, 2, 2, 2, 2, 1, 0, 0, 2, 1, 1, 2, 1, 2, 2, 1, 2, 0])
```

In [56]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train
```

```
Training-set accuracy score: 1.0000
```

Print the scores on training and test set

In [57]:
```
print('Training set score: {:.4f}'.format(clf_gini.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(clf_gini.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.9306
```

EVALUATION METRICS

In [58]:
```
print('Mean Absolute Error : ', metrics.mean_absolute_error(y_test, y_pred_g
print('Mean Squared Error : ', metrics.mean_squared_error(y_test, y_pred_gin
print('Root Mean Squared Error : ', np.sqrt(metrics.mean_squared_error(y_tes
```

```
Mean Absolute Error :  0.08333333333333333
Mean Squared Error :  0.1111111111111111
Root Mean Squared Error :  0.3333333333333333
```

Classification Report

In [59]:
```
print(classification_report(y_test, y_pred_gini))
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.95        28
           1       0.93      0.93      0.93        27
           2       0.89      0.94      0.91        17

    accuracy                           0.93        72
   macro avg       0.93      0.93      0.93        72
weighted avg       0.93      0.93      0.93        72
```

Decision Tree Classifier with criterion entropy

In [60]:
```
clf_en = DecisionTreeClassifier(criterion='entropy',max_depth=4)
```

In [61]:
```
clf_en.fit(X_train, y_train)
```

Out[61]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

Predict the Test set results and Check accuracy score

In [62]:
```
y_pred_en = clf_en.predict(X_test)
print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accura
```

```
Model accuracy score with criterion entropy: 0.9306
```

Compare the train-set and test-set accuracy

In [64]:
```
y_pred_train_en = clf_en.predict(X_train)
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train
```

```
Training-set accuracy score: 1.0000
```

Print the scores on training and test set

In [65]:
```
print('Training set score: {:.4f}'.format(clf_en.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf_en.score(X_test, y_test)))
```

```
Training set score: 1.0000
Test set score: 0.9306
```

EVALUATION METRICS

In [68]:
```
print('Mean Absolute Error : ', metrics.mean_absolute_error(y_test, y_pred_e
print('Mean Squared Error : ', metrics.mean_squared_error(y_test, y_pred_en)
print('Root Mean Squared Error : ', np.sqrt(metrics.mean_squared_error(y_tes
```

```
Mean Absolute Error :  0.06944444444444445
Mean Squared Error :  0.06944444444444445
Root Mean Squared Error :  0.26352313834736496
```

Confusion matrix

In [69]:
```
cm = confusion_matrix(y_test, y_pred_en)
print('Confusion matrix\n\n', cm)
```

```
Confusion matrix

 [[27  1  0]
 [ 3 24  0]
 [ 0  1 16]]
```

Classification Report

In [70]:
```
print(classification_report(y_test, y_pred_en))
```

```
              precision    recall  f1-score   support

           0       0.90      0.96      0.93        28
           1       0.92      0.89      0.91        27
           2       1.00      0.94      0.97        17

    accuracy                           0.93        72
   macro avg       0.94      0.93      0.94        72
weighted avg       0.93      0.93      0.93        72
```

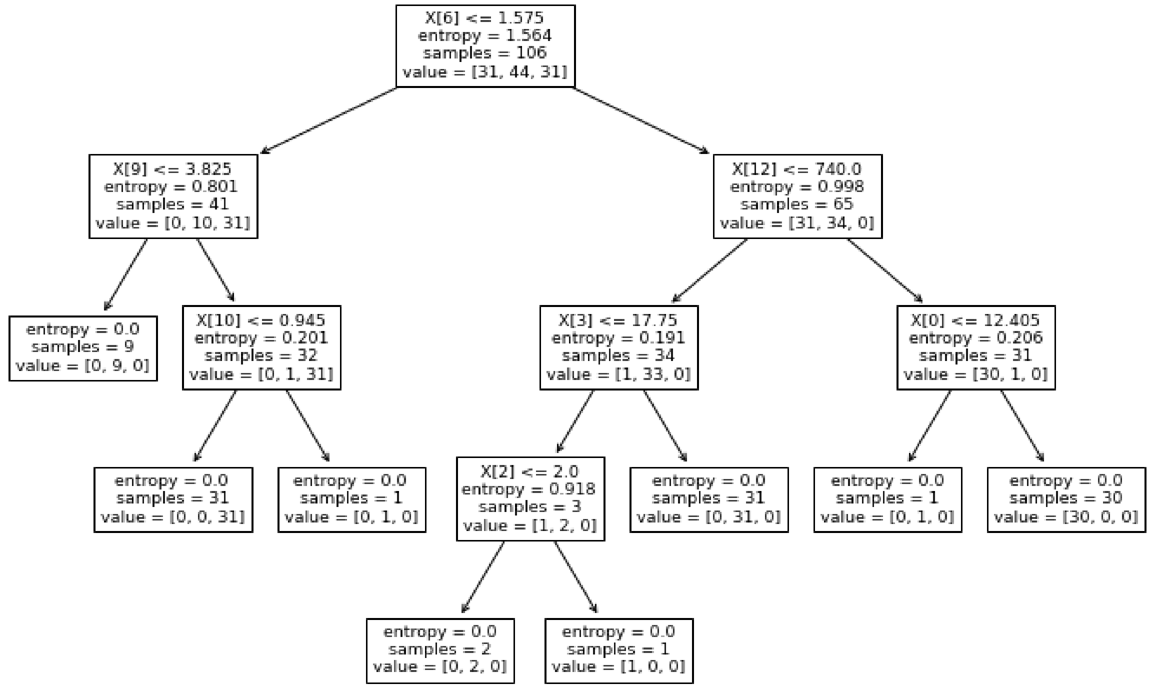
Visualize decision-trees¶

In [71]:
```python
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_en.fit(X_train, y_train))
```

Out[71]:
```
[Text(283.2923076923077, 391.392, 'X[6] <= 1.575\nentropy = 1.564\nsamples = 10
6\nvalue = [31, 44, 31]'),
 Text(103.01538461538462, 304.416, 'X[9] <= 3.825\nentropy = 0.801\nsamples = 4
1\nvalue = [0, 10, 31]'),
 Text(51.50769230769231, 217.44, 'entropy = 0.0\nsamples = 9\nvalue = [0, 9,
0]'),
 Text(154.52307692307693, 217.44, 'X[10] <= 0.945\nentropy = 0.201\nsamples = 3
2\nvalue = [0, 1, 31]'),
 Text(103.01538461538462, 130.464, 'entropy = 0.0\nsamples = 31\nvalue = [0, 0,
31]'),
 Text(206.03076923076924, 130.464, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(463.5692307692308, 304.416, 'X[12] <= 740.0\nentropy = 0.998\nsamples = 6
5\nvalue = [31, 34, 0]'),
 Text(360.55384615384617, 217.44, 'X[3] <= 17.75\nentropy = 0.191\nsamples = 34
\nvalue = [1, 33, 0]'),
 Text(309.04615384615386, 130.464, 'X[2] <= 2.0\nentropy = 0.918\nsamples = 3\n
value = [1, 2, 0]'),
 Text(257.53846153846155, 43.488, 'entropy = 0.0\nsamples = 2\nvalue = [0, 2,
0]'),
 Text(360.55384615384617, 43.488, 'entropy = 0.0\nsamples = 1\nvalue = [1, 0,
0]'),
 Text(412.0615384615385, 130.464, 'entropy = 0.0\nsamples = 31\nvalue = [0, 31,
0]'),
 Text(566.5846153846154, 217.44, 'X[0] <= 12.405\nentropy = 0.206\nsamples = 31
\nvalue = [30, 1, 0]'),
 Text(515.0769230769231, 130.464, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1,
0]'),
 Text(618.0923076923077, 130.464, 'entropy = 0.0\nsamples = 30\nvalue = [30, 0,
0]')]
```



CONCLUSION:

Imported two dataset and perfomed classification using decision tree with various criterion. Evaluated each model on various evaluation metrics.

REFERENCES:

https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial/notebook (https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial/notebook) https://www.kaggle.com/code/satishgunjal/tutorial-decision-tree/notebook (https://www.kaggle.com/code/satishgunjal/tutorial-decision-tree/notebook) https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)