Decision Tree Classifier

Index

1. [Preparation of the Data]
2. [Visualizing]
3. [Splitting the data: train and test]
4. [Building the Model]
5. [Finding Performance Measures]
6. [Visualization of the Tree]
7. [Calculating Gini Index]
8. [Decision Rule]

Material Reference for Decision Tree Working https://www.saedsayad.com/decision_tree.htm (https://www.saedsayad.com/decision_tree.htm)

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

In [9]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Dataset: Iris Dataset

In [6]:
```python
from sklearn.datasets import load_iris

iris = load_iris() #load_iris ==> class, we are creating an object of the class
```

In [8]:
```python
iris.keys()
```

Out[8]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

In [9]:
```python
iris['data']
```

Out[9]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
```

In [10]:
```
iris['target']
```

Out[10]:
```
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, 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, 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, 2, 2])
```

In [11]:
```
iris['target_names']
```

Out[11]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [12]:
```
iris['feature_names']
```

Out[12]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

Preparation of the data

In [14]:
```
X = iris['data']
X[0:10]
```

Out[14]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1]])
```

In [16]:
```
y = iris['target']
y[0:10]
```

Out[16]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [17]:
```
y[100:120]
```

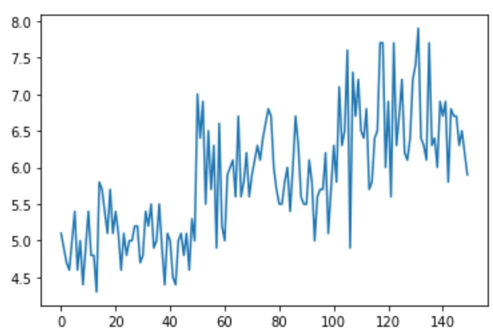
Out[17]:
```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [18]:
```
len(y)
```

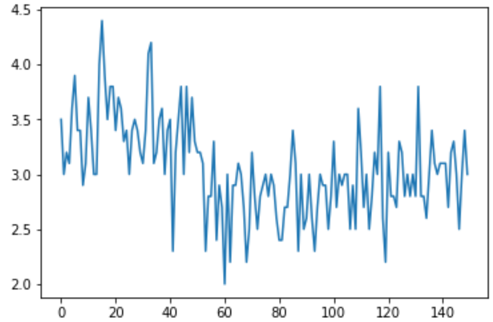
Out[18]:
```
150
```

Visualizing

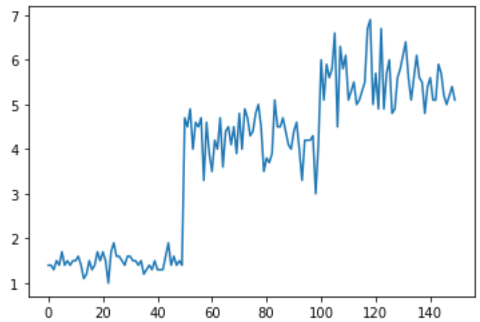
In [19]:
```
plt.plot(X[:,0]); #Sepal Length
```



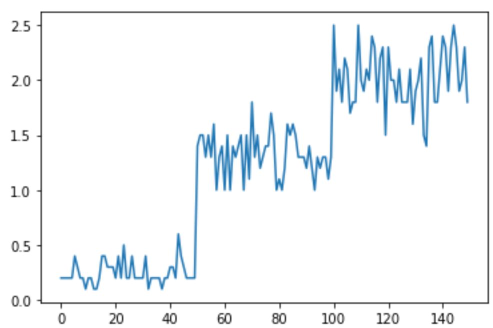
In [20]:
```
plt.plot(X[:,1]); #Sepal Width
```



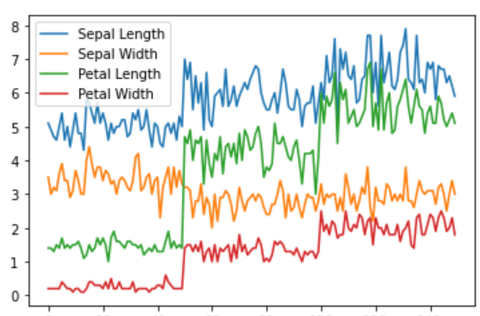
In [21]:
```
plt.plot(X[:,2]); #Petal Length
```



In [22]:
```
plt.plot(X[:,3]); #Petal Width
```



In [24]:
```
plt.plot(X[:,0], label='Sepal Length'); #Sepal Length
plt.plot(X[:,1], label = 'Sepal Width'); #Sepal Width
plt.plot(X[:,2], label = 'Petal Length'); #Petal Length
plt.plot(X[:,3], label = 'Petal Width'); #Petal Width
plt.legend();
```



Splitting the data: train and test

In [26]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[26]:
```
((120, 4), (30, 4), (120,), (30,))
```

Building the model

In [30]:
```
from sklearn.tree import DecisionTreeClassifier

dt_1 = DecisionTreeClassifier()

# Training the model

dt_1 = dt_1.fit(X_train, y_train)
```

In [31]:
```
y_pred = dt_1.predict(X_test)
y_pred
```

Out[31]:
```
array([2, 0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2, 0, 0, 2, 1, 1, 2, 2, 2, 2, 0,
       2, 0, 1, 2, 1, 0, 1, 2])
```

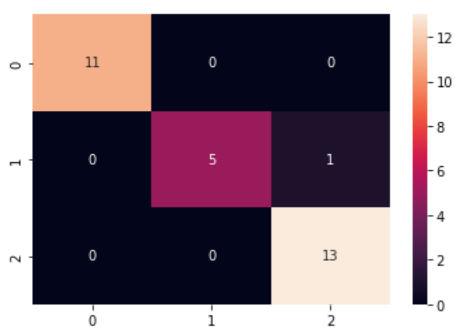
In [32]:
```
y_test
```

Out[32]:
```
array([2, 0, 2, 0, 2, 2, 0, 0, 2, 0, 0, 2, 0, 0, 2, 1, 1, 1, 2, 2, 2, 0,
       2, 0, 1, 2, 1, 0, 1, 2])
```

Finding the performance measures

In [34]:
```
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score# ROC - receiver operating character
```

In [36]:
```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot = True); # annot for displaying values
```



horizontal --> actual

vertical --> predicted

Diagonal ==> correct ones

Non-diagonal ==> mistaken classification

Therefore, only 1 misclassification

In [40]:
```
report = classification_report(y_test, y_pred)

print('The classification report:\n', report)
```

```
The classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.83      0.91         6
           2       0.93      1.00      0.96        13

    accuracy                           0.97        30
   macro avg       0.98      0.94      0.96        30
weighted avg       0.97      0.97      0.97        30
```

In [41]:
```
fpr, tpr = roc_curve(y_test, y_pred)
plt.plot(fpr, tpr)
```

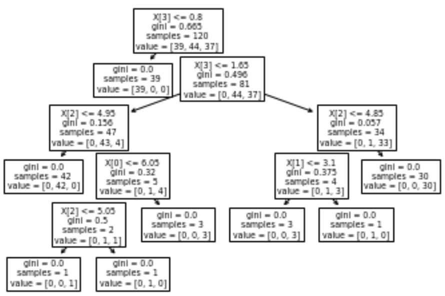
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\URVISH~1\AppData\Local\Temp/ipykernel_17156/75357538.py in <module>
----> 1 fpr, tpr = roc_curve(y_test, y_pred)
      2 plt.plot(fpr, tpr)

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kwargs)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64 
     65             # extra_args > 0

~\anaconda3\lib\site-packages\sklearn\metrics\_ranking.py in roc_curve(y_true, y_score, pos_label, sample_weight, drop_intermed
iate)
    911 
    912     """
--> 913     fps, tps, thresholds = _binary_clf_curve(
    914         y_true, y_score, pos_label=pos_label, sample_weight=sample_weight)
    915 

~\anaconda3\lib\site-packages\sklearn\metrics\_ranking.py in _binary_clf_curve(y_true, y_score, pos_label, sample_weight)
    689     if not (y_type == "binary" or
    690             (y_type == "multiclass" and pos_label is not None)):
--> 691         raise ValueError("{0} format is not supported".format(y_type))
    692 
    693     check_consistent_length(y_true, y_score, sample_weight)

ValueError: multiclass format is not supported
```

NOTE If target variable isn't binary (it is multiclass), roc auc functions don't work

Visualization of the tree

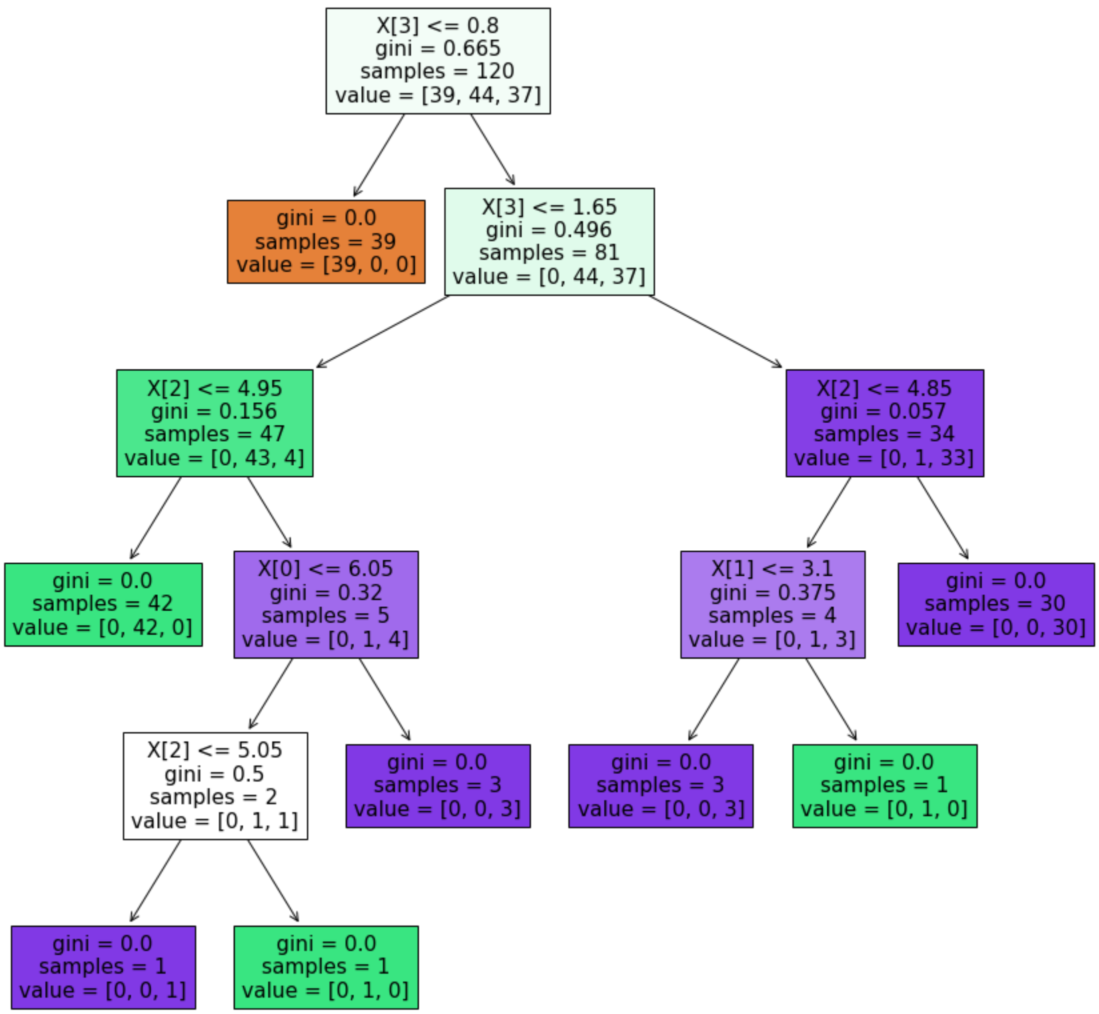
In [42]:
```
from sklearn.tree import plot_tree

plot_tree(dt_1)
```

Out[42]:
```
[Text(133.92000000000002, 199.32, 'X[3] <= 0.8\ngini = 0.665\nsamples = 120\nvalue = [39, 44, 37]'),
 Text(100.44000000000001, 163.07999999999998, 'gini = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(167.40000000000003, 163.07999999999998, 'X[3] <= 1.65\ngini = 0.496\nsamples = 81\nvalue = [0, 44, 37]'),
 Text(66.96000000000001, 126.83999999999999, 'X[2] <= 4.95\ngini = 0.156\nsamples = 47\nvalue = [0, 43, 4]'),
 Text(33.480000000000004, 90.6, 'gini = 0.0\nsamples = 42\nvalue = [0, 42, 0]'),
 Text(100.44000000000001, 90.6, 'X[0] <= 6.05\ngini = 0.32\nsamples = 5\nvalue = [0, 1, 4]'),
 Text(66.96000000000001, 54.359999999999985, 'X[2] <= 5.05\ngini = 0.5\nsamples = 2\nvalue = [0, 1, 1]'),
 Text(33.480000000000004, 18.119999999999976, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(100.44000000000001, 18.119999999999976, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(133.92000000000002, 54.359999999999985, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(267.84000000000003, 126.83999999999999, 'X[2] <= 4.85\ngini = 0.057\nsamples = 34\nvalue = [0, 1, 33]'),
 Text(234.36, 90.6, 'X[1] <= 3.1\ngini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(200.88000000000002, 54.359999999999985, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(267.84000000000003, 54.359999999999985, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(301.32000000000005, 90.6, 'gini = 0.0\nsamples = 30\nvalue = [0, 0, 30]')]
```



In [46]:
```
plt.figure(figsize = (15, 15)) # resizing
plot_tree(dt_1, filled = True) # filled used for coloring the tree leaves
```

Out[46]:
```
[Text(334.8, 747.4499999999999, 'X[3] <= 0.8\ngini = 0.665\nsamples = 120\nvalue = [39, 44, 37]'),
 Text(251.10000000000002, 611.55, 'gini = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(418.5, 611.55, 'X[3] <= 1.65\ngini = 0.496\nsamples = 81\nvalue = [0, 44, 37]'),
 Text(167.4, 475.65, 'X[2] <= 4.95\ngini = 0.156\nsamples = 47\nvalue = [0, 43, 4]'),
 Text(83.7, 339.74999999999994, 'gini = 0.0\nsamples = 42\nvalue = [0, 42, 0]'),
 Text(251.10000000000002, 339.74999999999994, 'X[0] <= 6.05\ngini = 0.32\nsamples = 5\nvalue = [0, 1, 4]'),
 Text(167.4, 203.8499999999999, 'X[2] <= 5.05\ngini = 0.5\nsamples = 2\nvalue = [0, 1, 1]'),
 Text(83.7, 67.94999999999993, 'gini = 0.0\nsamples = 1\nvalue = [0, 0, 1]'),
 Text(251.10000000000002, 67.94999999999993, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(334.8, 203.8499999999999, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(669.6, 475.65, 'X[2] <= 4.85\ngini = 0.057\nsamples = 34\nvalue = [0, 1, 33]'),
 Text(585.9, 339.74999999999994, 'X[1] <= 3.1\ngini = 0.375\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(502.20000000000005, 203.8499999999999, 'gini = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(669.6, 203.8499999999999, 'gini = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(753.3000000000001, 339.74999999999994, 'gini = 0.0\nsamples = 30\nvalue = [0, 0, 30]')]
```



Orange ==> Class 0

Green ==> Class 1

Purple ==> Class 2

Calculating Gini Index

$gini = 1 - \sum p^2$

In [49]:
```
gini = 1 - (39/120)**2 - (44/120)**2 - (37/120)**2

print('Gini:', gini)
```

```
Gini: 0.6648611111111111
```

Decision Rules

In [53]:
```
from sklearn.tree import export_text

text = export_text(dt_1, feature_names = iris['feature_names'])
print(text)
```

```
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.65
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- sepal length (cm) <= 6.05
|   |   |   |   |--- petal length (cm) <= 5.05
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- petal length (cm) >  5.05
|   |   |   |   |   |--- class: 1
|   |   |   |--- sepal length (cm) >  6.05
|   |   |   |   |--- class: 2
|   |--- petal width (cm) >  1.65
|   |   |--- petal length (cm) <= 4.85
|   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |--- class: 2
|   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```

In [55]:
```
dt_1.criterion
```

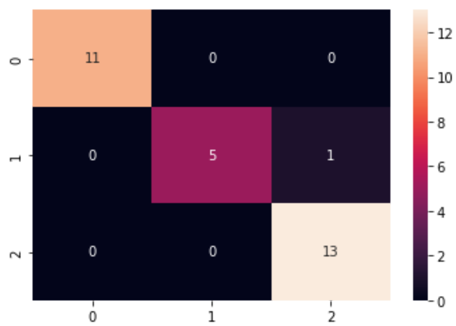
Out[55]:
```
'gini'
```

In [57]:
```
dt_1.tree_.max_depth #level of tree
```

Out[57]:
```
5
```

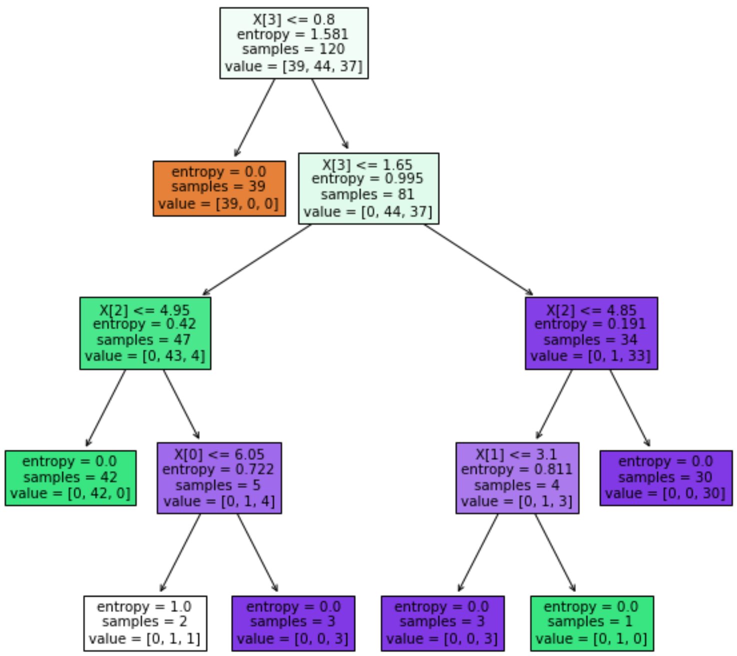
Modifying the model

In [59]:
```
dt_2 = DecisionTreeClassifier(criterion = 'entropy', max_depth = 4)

dt_2 = dt_2.fit(X_train, y_train)
y_pred_2 = dt_2.predict(X_test)
cm_2 = confusion_matrix(y_test, y_pred_2)
sns.heatmap(cm_2, annot = True);
```



In [61]:
```
plt.figure(figsize = (10, 10)) # resizing
plot_tree(dt_2, filled = True)
```

Out[61]:
```
[Text(223.2, 489.24, 'X[3] <= 0.8\nentropy = 1.581\nsamples = 120\nvalue = [39, 44, 37]'),
 Text(167.39999999999998, 380.52000000000004, 'entropy = 0.0\nsamples = 39\nvalue = [39, 0, 0]'),
 Text(279.0, 380.52000000000004, 'X[3] <= 1.65\nentropy = 0.995\nsamples = 81\nvalue = [0, 44, 37]'),
 Text(111.6, 271.8, 'X[2] <= 4.95\nentropy = 0.42\nsamples = 47\nvalue = [0, 43, 4]'),
 Text(55.8, 163.08000000000004, 'entropy = 0.0\nsamples = 42\nvalue = [0, 42, 0]'),
 Text(167.39999999999998, 163.08000000000004, 'X[0] <= 6.05\nentropy = 0.722\nsamples = 5\nvalue = [0, 1, 4]'),
 Text(111.6, 54.360000000000014, 'entropy = 1.0\nsamples = 2\nvalue = [0, 1, 1]'),
 Text(223.2, 54.360000000000014, 'entropy = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(446.4, 271.8, 'X[2] <= 4.85\nentropy = 0.191\nsamples = 34\nvalue = [0, 1, 33]'),
 Text(390.59999999999997, 163.08000000000004, 'X[1] <= 3.1\nentropy = 0.811\nsamples = 4\nvalue = [0, 1, 3]'),
 Text(334.79999999999995, 54.360000000000014, 'entropy = 0.0\nsamples = 3\nvalue = [0, 0, 3]'),
 Text(446.4, 54.360000000000014, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1, 0]'),
 Text(502.2, 163.08000000000004, 'entropy = 0.0\nsamples = 30\nvalue = [0, 0, 30]')]
```



--Homework: Perform Decision Tree Classification for German Credit dataset

In [2]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Decision Tree for Breast Cancer

In [11]:
```
from sklearn.datasets import load_breast_cancer

bc = load_breast_cancer() #load_iris ==> class, we are creating an object of the class
```

In [12]:
```
bc.keys()
```

Out[12]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

In [13]:
```
bc['data']
```

Out[13]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [14]:
```
bc['target']
```

Out[14]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 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, 0, 0, 0, 0, 0, 0, 1])
```

In [15]:
```
bc['target_names']
```

Out[15]:
```
array(['malignant', 'benign'], dtype='<U9')
```

In [57]:
```
bc['feature_names']
```

Out[57]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Preparation of the data

In [17]:
```
X = bc['data']
X[0:10]
```

Out[17]:
```
array([[1.799e+01, 1.038e+01, 1.228e+02, 1.001e+03, 1.184e-01, 2.776e-01,
        3.001e-01, 1.471e-01, 2.419e-01, 7.871e-02, 1.095e+00, 9.053e-01,
        8.589e+00, 1.534e+02, 6.399e-03, 4.904e-02, 5.373e-02, 1.587e-02,
        3.003e-02, 6.193e-03, 2.538e+01, 1.733e+01, 1.846e+02, 2.019e+03,
        1.622e-01, 6.656e-01, 7.119e-01, 2.654e-01, 4.601e-01, 1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, 1.326e+03, 8.474e-02, 7.864e-02,
        8.690e-02, 7.017e-02, 1.812e-01, 5.667e-02, 5.435e-01, 7.339e-01,
        3.398e+00, 7.408e+01, 5.225e-03, 1.308e-02, 1.860e-02, 1.340e-02,
        1.389e-02, 3.532e-03, 2.499e+01, 2.341e+01, 1.588e+02, 1.956e+03,
        1.238e-01, 1.866e-01, 2.416e-01, 1.860e-01, 2.750e-01, 8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, 1.203e+03, 1.096e-01, 1.599e-01,
        1.974e-01, 1.279e-01, 2.069e-01, 5.999e-02, 7.456e-01, 7.869e-01,
        4.585e+00, 9.403e+01, 6.150e-03, 4.006e-02, 3.832e-02, 2.058e-02,
        2.250e-02, 4.571e-03, 2.357e+01, 2.553e+01, 1.525e+02, 1.709e+03,
        1.444e-01, 4.245e-01, 4.504e-01, 2.430e-01, 3.613e-01, 8.758e-02],
       [1.142e+01, 2.038e+01, 7.758e+01, 3.861e+02, 1.425e-01, 2.839e-01,
        2.414e-01, 1.052e-01, 2.597e-01, 9.744e-02, 4.956e-01, 1.156e+00,
        3.445e+00, 2.723e+01, 9.110e-03, 7.458e-02, 5.661e-02, 1.867e-02,
        5.963e-02, 9.208e-03, 1.491e+01, 2.650e+01, 9.887e+01, 5.677e+02,
        2.098e-01, 8.663e-01, 6.869e-01, 2.575e-01, 6.638e-01, 1.730e-01],
       [2.029e+01, 1.434e+01, 1.351e+02, 1.297e+03, 1.003e-01, 1.328e-01,
        1.980e-01, 1.043e-01, 1.809e-01, 5.883e-02, 7.572e-01, 7.813e-01,
        5.438e+00, 9.444e+01, 1.149e-02, 2.461e-02, 5.688e-02, 1.885e-02,
        1.756e-02, 5.115e-03, 2.254e+01, 1.667e+01, 1.522e+02, 1.575e+03,
        1.374e-01, 2.050e-01, 4.000e-01, 1.625e-01, 2.364e-01, 7.678e-02],
       [1.245e+01, 1.570e+01, 8.257e+01, 4.771e+02, 1.278e-01, 1.700e-01,
        1.578e-01, 8.089e-02, 2.087e-01, 7.613e-02, 3.345e-01, 8.902e-01,
        2.217e+00, 2.719e+01, 7.510e-03, 3.345e-02, 3.672e-02, 1.137e-02,
        2.165e-02, 5.082e-03, 1.547e+01, 2.375e+01, 1.034e+02, 7.416e+02,
        1.791e-01, 5.249e-01, 5.355e-01, 1.741e-01, 3.985e-01, 1.244e-01],
       [1.825e+01, 1.998e+01, 1.196e+02, 1.040e+03, 9.463e-02, 1.090e-01,
        1.127e-01, 7.400e-02, 1.794e-01, 5.742e-02, 4.467e-01, 7.732e-01,
        3.180e+00, 5.391e+01, 4.314e-03, 1.382e-02, 2.254e-02, 1.039e-02,
        1.369e-02, 2.179e-03, 2.288e+01, 2.766e+01, 1.532e+02, 1.606e+03,
        1.442e-01, 2.576e-01, 3.784e-01, 1.932e-01, 3.063e-01, 8.368e-02],
       [1.371e+01, 2.083e+01, 9.020e+01, 5.779e+02, 1.189e-01, 1.645e-01,
        9.366e-02, 5.985e-02, 2.196e-01, 7.451e-02, 5.835e-01, 1.377e+00,
        3.856e+00, 5.096e+01, 8.805e-03, 3.029e-02, 2.488e-02, 1.448e-02,
        1.486e-02, 5.412e-03, 1.706e+01, 2.814e+01, 1.106e+02, 8.970e+02,
        1.654e-01, 3.682e-01, 2.678e-01, 1.556e-01, 3.196e-01, 1.151e-01],
       [1.300e+01, 2.182e+01, 8.750e+01, 5.198e+02, 1.273e-01, 1.932e-01,
        1.859e-01, 9.353e-02, 2.350e-01, 7.389e-02, 3.063e-01, 1.002e+00,
        2.406e+00, 2.432e+01, 5.731e-03, 3.502e-02, 3.553e-02, 1.226e-02,
        2.143e-02, 3.749e-03, 1.549e+01, 3.073e+01, 1.062e+02, 7.393e+02,
        1.703e-01, 5.401e-01, 5.390e-01, 2.060e-01, 4.378e-01, 1.072e-01],
       [1.246e+01, 2.404e+01, 8.397e+01, 4.759e+02, 1.186e-01, 2.396e-01,
        2.273e-01, 8.543e-02, 2.030e-01, 8.243e-02, 2.976e-01, 1.599e+00,
        2.039e+00, 2.394e+01, 7.149e-03, 7.217e-02, 7.743e-02, 1.432e-02,
        1.789e-02, 1.008e-02, 1.509e+01, 4.068e+01, 9.765e+01, 7.114e+02,
        1.853e-01, 1.058e+00, 1.105e+00, 2.210e-01, 4.366e-01, 2.075e-01]])
```

In [18]:
```
y = bc['target']
```

In [20]:
```
y[100:120]
```

Out[20]:
```
array([0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0])
```

In [21]:
```
len(X),len(y)
```

Out[21]:
```
(569, 569)
```

Splitting the data

In [30]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[30]:
```
((455, 30), (114, 30), (455,), (114,))
```

Buiding the model

In [58]:
```
from sklearn.tree import DecisionTreeClassifier

dt_1 = DecisionTreeClassifier()

# Training the model

dt_1 = dt_1.fit(X_train, y_train)
```

In [59]:
```
y_pred = dt_1.predict(X_test)
y_pred
```

Out[59]:
```
array([0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1])
```

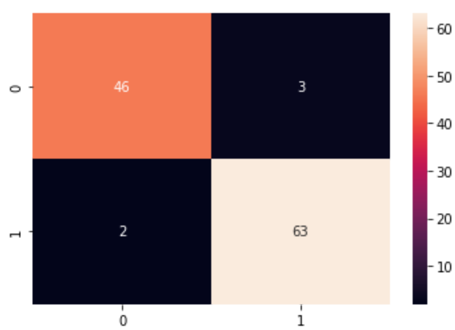
In [34]:
```
y_test
```

Out[34]:
```
array([0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 0, 1])
```

Finding performance measures

In [35]:
```
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score# ROC - receiver operating character
```

In [36]:
```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot = True);
```

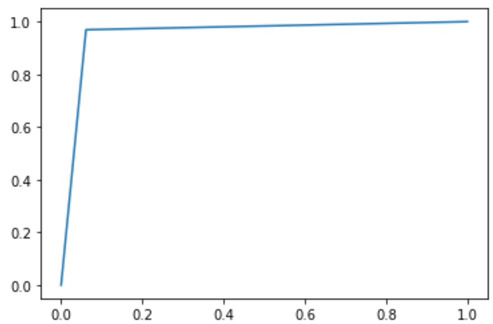


In [37]:
```
report = classification_report(y_test, y_pred)

print('The classification report:\n', report)
```

```
The classification report:
               precision    recall  f1-score   support

           0       0.96      0.94      0.95        49
           1       0.95      0.97      0.96        65

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.96       114
weighted avg       0.96      0.96      0.96       114
```

In [39]:
```
fpr, tpr,_ = roc_curve(y_test, y_pred)
plt.plot(fpr, tpr)
```

Out[39]:
```
[<matplotlib.lines.Line2D at 0x2063516aac0>]
```



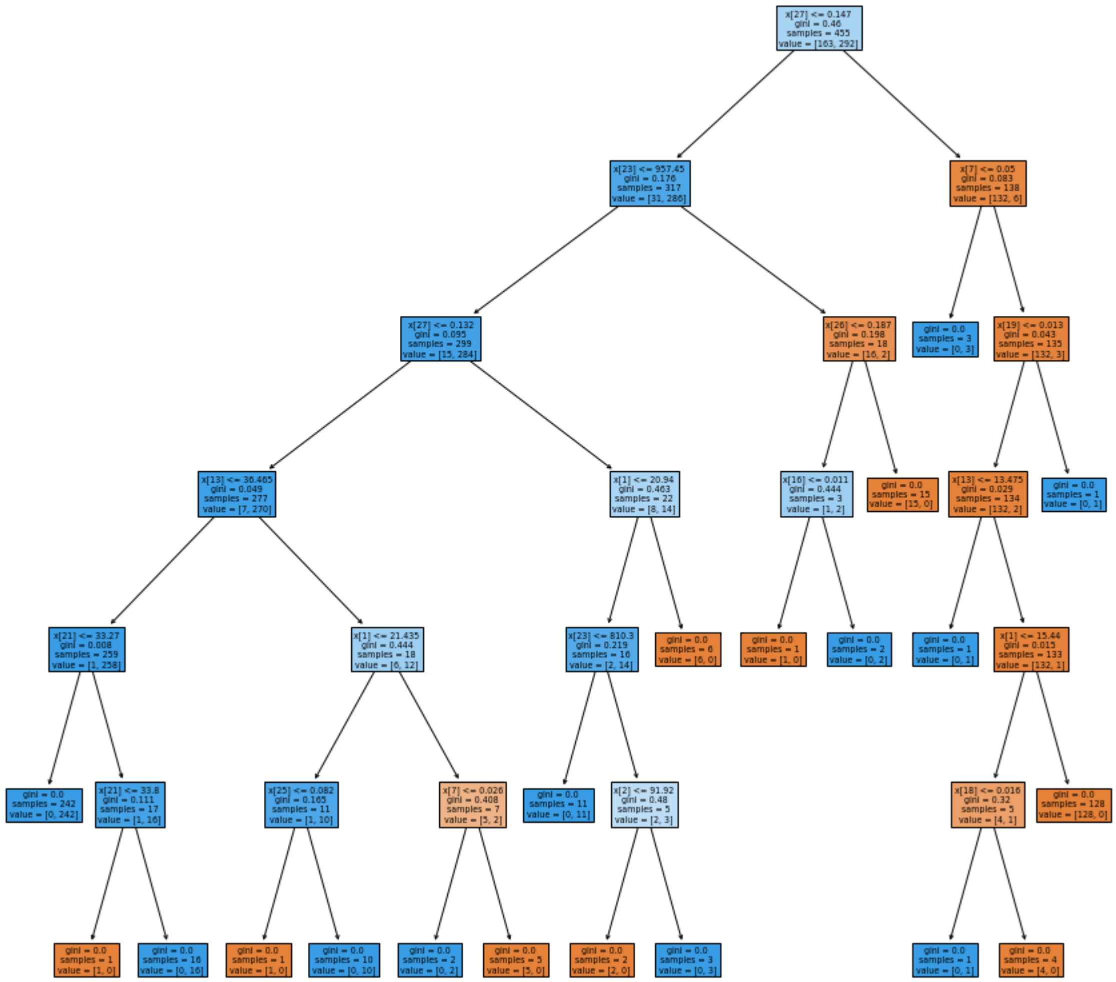
Visualization of the tree

In [40]:
```
from sklearn.tree import plot_tree

plot_tree(dt_1)
```

Out[40]:
```
[Text(0.7331730769230769, 0.9285714285714286, 'x[27] <= 0.147\ngini = 0.46\nsamples = 455\nvalue = [163, 292]'),
 Text(0.5817307692307693, 0.7857142857142857, 'x[23] <= 957.45\ngini = 0.176\nsamples = 317\nvalue = [31, 286]'),
 Text(0.3942307692307692, 0.6428571428571429, 'x[27] <= 0.132\ngini = 0.095\nsamples = 299\nvalue = [15, 284]'),
 Text(0.21153846153846154, 0.5, 'x[13] <= 36.465\ngini = 0.049\nsamples = 277\nvalue = [7, 270]'),
 Text(0.07692307692307693, 0.35714285714285715, 'x[21] <= 33.27\ngini = 0.008\nsamples = 259\nvalue = [1, 258]'),
 Text(0.038461538461538464, 0.21428571428571427, 'gini = 0.0\nsamples = 242\nvalue = [0, 242]'),
 Text(0.11538461538461539, 0.21428571428571427, 'x[21] <= 33.8\ngini = 0.111\nsamples = 17\nvalue = [1, 16]'),
 Text(0.07692307692307693, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.15384615384615385, 0.07142857142857142, 'gini = 0.0\nsamples = 16\nvalue = [0, 16]'),
 Text(0.34615384615384615, 0.35714285714285715, 'x[1] <= 21.435\ngini = 0.444\nsamples = 18\nvalue = [6, 12]'),
 Text(0.2692307692307692, 0.21428571428571427, 'x[25] <= 0.082\ngini = 0.165\nsamples = 11\nvalue = [1, 10]'),
 Text(0.23076923076923078, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3076923076923077, 0.07142857142857142, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]'),
 Text(0.4230769230769231, 0.21428571428571427, 'x[7] <= 0.026\ngini = 0.408\nsamples = 7\nvalue = [5, 2]'),
 Text(0.38461538461538464, 0.07142857142857142, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.46153846153846156, 0.07142857142857142, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.5769230769230769, 0.5, 'x[1] <= 20.94\ngini = 0.463\nsamples = 22\nvalue = [8, 14]'),
 Text(0.5384615384615384, 0.35714285714285715, 'x[23] <= 810.3\ngini = 0.219\nsamples = 16\nvalue = [2, 14]'),
 Text(0.5, 0.21428571428571427, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
```

In [41]:
```python
plt.figure(figsize = (15, 15)) # resizing
plot_tree(dt_1, filled = True)
```

Out[41]:
```
[Text(0.7331730769230769, 0.9285714285714286, 'x[27] <= 0.147\ngini = 0.46\nsamples = 455\nvalue = [163, 292]'),
 Text(0.5817307692307693, 0.7857142857142857, 'x[23] <= 957.45\ngini = 0.176\nsamples = 317\nvalue = [31, 286]'),
 Text(0.3942307692307692, 0.6428571428571429, 'x[27] <= 0.132\ngini = 0.095\nsamples = 299\nvalue = [15, 284]'),
 Text(0.21153846153846154, 0.5, 'x[13] <= 36.465\ngini = 0.049\nsamples = 277\nvalue = [7, 270]'),
 Text(0.07692307692307693, 0.35714285714285715, 'x[21] <= 33.27\ngini = 0.008\nsamples = 259\nvalue = [1, 258]'),
 Text(0.038461538461538464, 0.21428571428571427, 'gini = 0.0\nsamples = 242\nvalue = [0, 242]'),
 Text(0.11538461538461539, 0.21428571428571427, 'x[21] <= 33.8\ngini = 0.111\nsamples = 17\nvalue = [1, 16]'),
 Text(0.07692307692307693, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.15384615384615385, 0.07142857142857142, 'gini = 0.0\nsamples = 16\nvalue = [0, 16]'),
 Text(0.34615384615384615, 0.35714285714285715, 'x[1] <= 21.435\ngini = 0.444\nsamples = 18\nvalue = [6, 12]'),
 Text(0.2692307692307692, 0.21428571428571427, 'x[25] <= 0.082\ngini = 0.165\nsamples = 11\nvalue = [1, 10]'),
 Text(0.23076923076923078, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.3076923076923077, 0.07142857142857142, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]'),
 Text(0.4230769230769231, 0.21428571428571427, 'x[7] <= 0.026\ngini = 0.408\nsamples = 7\nvalue = [5, 2]'),
 Text(0.38461538461538464, 0.07142857142857142, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.46153846153846156, 0.07142857142857142, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(0.5769230769230769, 0.5, 'x[1] <= 20.94\ngini = 0.463\nsamples = 22\nvalue = [8, 14]'),
 Text(0.5384615384615384, 0.35714285714285715, 'x[23] <= 810.3\ngini = 0.219\nsamples = 16\nvalue = [2, 14]'),
 Text(0.5, 0.21428571428571427, 'gini = 0.0\nsamples = 11\nvalue = [0, 11]'),
 Text(0.5769230769230769, 0.21428571428571427, 'x[2] <= 91.92\ngini = 0.48\nsamples = 5\nvalue = [2, 3]'),
 Text(0.5384615384615384, 0.07142857142857142, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.6153846153846154, 0.07142857142857142, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.6153846153846154, 0.35714285714285715, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.7692307692307693, 0.6428571428571429, 'x[26] <= 0.187\ngini = 0.198\nsamples = 18\nvalue = [16, 2]'),
 Text(0.7307692307692307, 0.5, 'x[16] <= 0.011\ngini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.6923076923076923, 0.35714285714285715, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7692307692307693, 0.35714285714285715, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.8076923076923077, 0.5, 'gini = 0.0\nsamples = 15\nvalue = [15, 0]'),
 Text(0.8846153846153846, 0.7857142857142857, 'x[7] <= 0.05\ngini = 0.083\nsamples = 138\nvalue = [132, 6]'),
 Text(0.8461538461538461, 0.6428571428571429, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9230769230769231, 0.6428571428571429, 'x[19] <= 0.013\ngini = 0.043\nsamples = 135\nvalue = [132, 3]'),
 Text(0.8846153846153846, 0.5, 'x[13] <= 13.475\ngini = 0.029\nsamples = 134\nvalue = [132, 2]'),
 Text(0.8461538461538461, 0.35714285714285715, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.9230769230769231, 0.35714285714285715, 'x[1] <= 15.44\ngini = 0.015\nsamples = 133\nvalue = [132, 1]'),
 Text(0.8846153846153846, 0.21428571428571427, 'x[18] <= 0.016\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
 Text(0.8461538461538461, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.9230769230769231, 0.07142857142857142, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.9615384615384616, 0.21428571428571427, 'gini = 0.0\nsamples = 128\nvalue = [128, 0]'),
 Text(0.9615384615384616, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]')]
```



Calculating Gini Index

In [43]:
```python
gini = 1 - (163/455)**2 - (292/455)**2

print('Gini:', gini)
```

```
Gini: 0.45980920178722373
```

the Gini index varies between values 0 and 1,

where 0 expresses the purity of classification, i.e. All the elements belong to a specified class or only one class exists there.

And 1 indicates the random distribution of elements across various classes.

The value of 0.5 of the Gini Index shows an equal distribution of elements over some classes.

Decision Rules

In [63]:
```
from sklearn.tree import export_text

text = export_text(dt_1, feature_names = bc['feature_names'])
print(text)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
C:\Users\URVISH~1\AppData\Local\Temp/ipykernel_21656/1325174591.py in <module>
      1 from sklearn.tree import export_text
      2
----> 3 text = export_text(dt_1, feature_names = bc['feature_names'])
      4 print(text)

~\anaconda3\lib\site-packages\sklearn\tree\_export.py in export_text(decision_tree, feature_names, max_depth, spacing, decimal
s, show_weights)
   1014         value_fmt = "{}{} value: {}\n"
   1015
-> 1016     if feature_names:
   1017         feature_names_ = [
   1018             feature_names[i] if i != _tree.TREE_UNDEFINED else None

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()
```

In [47]:
```
dt_1.criterion
```

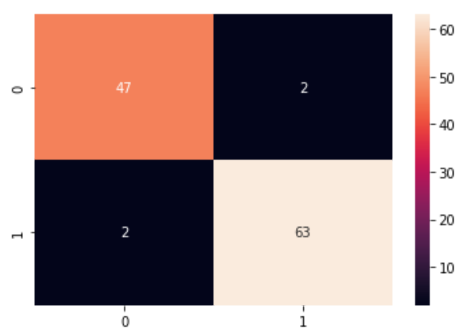
Out[47]: 'gini'

In [48]:
```
dt_1.tree_.max_depth #Level of tree
```

Out[48]: 6

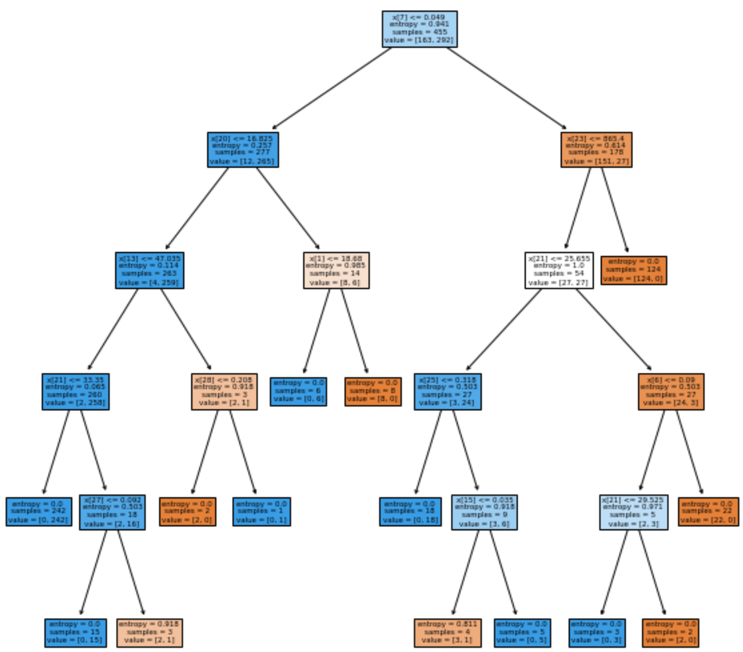
Modifying the model

In [53]:
```
dt_2 = DecisionTreeClassifier(criterion = 'entropy', max_depth = 5)

dt_2 = dt_2.fit(X_train, y_train)
y_pred_2 = dt_2.predict(X_test)
cm_2 = confusion_matrix(y_test, y_pred_2)
sns.heatmap(cm_2, annot = True);
```



In [54]:
```
plt.figure(figsize = (10, 10)) # resizing
plot_tree(dt_2, filled = True)
```

Out[54]:
```
[Text(0.5625, 0.9166666666666666, 'x[7] <= 0.049\nentropy = 0.941\nsamples = 455\nvalue = [163, 292]'),
 Text(0.325, 0.75, 'x[20] <= 16.825\nentropy = 0.257\nsamples = 277\nvalue = [12, 265]'),
 Text(0.2, 0.5833333333333334, 'x[13] <= 47.035\nentropy = 0.114\nsamples = 263\nvalue = [4, 259]'),
 Text(0.1, 0.4166666666666667, 'x[21] <= 33.35\nentropy = 0.065\nsamples = 260\nvalue = [2, 258]'),
 Text(0.05, 0.25, 'entropy = 0.0\nsamples = 242\nvalue = [0, 242]'),
 Text(0.15, 0.25, 'x[27] <= 0.092\nentropy = 0.503\nsamples = 18\nvalue = [2, 16]'),
 Text(0.1, 0.08333333333333333, 'entropy = 0.0\nsamples = 15\nvalue = [0, 15]'),
 Text(0.2, 0.08333333333333333, 'entropy = 0.918\nsamples = 3\nvalue = [2, 1]'),
 Text(0.3, 0.4166666666666667, 'x[28] <= 0.208\nentropy = 0.918\nsamples = 3\nvalue = [2, 1]'),
 Text(0.25, 0.25, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.35, 0.25, 'entropy = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.45, 0.5833333333333334, 'x[1] <= 18.68\nentropy = 0.985\nsamples = 14\nvalue = [8, 6]'),
 Text(0.4, 0.4166666666666667, 'entropy = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.5, 0.4166666666666667, 'entropy = 0.0\nsamples = 8\nvalue = [8, 0]'),
 Text(0.8, 0.75, 'x[23] <= 865.4\nentropy = 0.614\nsamples = 178\nvalue = [151, 27]'),
 Text(0.75, 0.5833333333333334, 'x[21] <= 25.655\nentropy = 1.0\nsamples = 54\nvalue = [27, 27]'),
 Text(0.6, 0.4166666666666667, 'x[25] <= 0.318\nentropy = 0.503\nsamples = 27\nvalue = [3, 24]'),
 Text(0.55, 0.25, 'entropy = 0.0\nsamples = 18\nvalue = [0, 18]'),
 Text(0.65, 0.25, 'x[15] <= 0.035\nentropy = 0.918\nsamples = 9\nvalue = [3, 6]'),
 Text(0.6, 0.08333333333333333, 'entropy = 0.811\nsamples = 4\nvalue = [3, 1]'),
 Text(0.7, 0.08333333333333333, 'entropy = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.9, 0.4166666666666667, 'x[6] <= 0.09\nentropy = 0.503\nsamples = 27\nvalue = [24, 3]'),
 Text(0.85, 0.25, 'x[21] <= 29.525\nentropy = 0.971\nsamples = 5\nvalue = [2, 3]'),
 Text(0.8, 0.08333333333333333, 'entropy = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.9, 0.08333333333333333, 'entropy = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.95, 0.25, 'entropy = 0.0\nsamples = 22\nvalue = [22, 0]'),
 Text(0.85, 0.5833333333333334, 'entropy = 0.0\nsamples = 124\nvalue = [124, 0]')]
```



In [55]:
```
cm_2
```

Out[55]:
```
array([[47,  2],
       [ 2, 63]], dtype=int64)
```

In [56]:
```
report = classification_report(y_test, y_pred_2)

print('The classification report:\n', report)
```

```
The classification report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96        49
           1       0.97      0.97      0.97        65

    accuracy                           0.96       114
   macro avg       0.96      0.96      0.96       114
weighted avg       0.96      0.96      0.96       114
```

Loading the dataset: from sklearn.datasets import load_breast_cancer

Splitting the data: from sklearn.model_selection import train_test_split

Building the model: from sklearn.tree import DecisionTreeClassifier

Finding perfromance measures: from sklearn.metrics import confusion_matrix, classification_report, roc_curve, roc_auc_score

Visualization of the tree: from sklearn.tree import plot_tree

Decision Rules Display: from sklearn.tree import export_text

```
In [ ]:
```