In [1]:
```
# import packages
import pandas as pd
```

In [2]:
```
from sklearn.datasets import load_iris
```

In [3]:
```
iris = load_iris()
print(iris)
```

```
{'data': 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],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
```

```
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
```

```
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]]), 'target': 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]), 'target_name
s': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'DESCR': '..
_iris_dataset:\n\nIris plants dataset\n--------------------\n\n**Data Set Cha
racteristics:**\n\n    :Number of Instances: 150 (50 in each of three classe
s)\n    :Number of Attributes: 4 numeric, predictive attributes and the class
\n    :Attribute Information:\n        - sepal length in cm\n        - sepal
width in cm\n        - petal length in cm\n        - petal width in cm\n
- class:\n                - Iris-Setosa\n                - Iris-Versicolour\n
- Iris-Virginica\n                \n    :Summary Statistics:\n\n    =========
===== ==== ==== ======= ===== ====================\n                    Min
Max   Mean    SD   Class Correlation\n    ============== ==== ==== ======= ==
```

```
=== ====================\n    sepal length:   4.3  7.9   5.84   0.83    0.782
6\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n    petal length:
1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal width:    0.1  2.5   1.2
0   0.76    0.9565  (high!)\n    ============== ==== ==== ======= ===== =====
==============\n\n    :Missing Attribute Values: None\n    :Class Distributi
on: 33.3% for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Mich
ael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe fam
ous Iris database, first used by Sir R.A. Fisher. The dataset is taken\nfrom
Fisher\'s paper. Note that it\'s the same as in R, but not as in the UCI\nMac
hine Learning Repository, which has two wrong data points.\n\nThis is perhaps
the best known database to be found in the\npattern recognition literature.
Fisher\'s paper is a classic in the field and\nis referenced frequently to th
is day.  (See Duda & Hart, for example.)  The\ndata set contains 3 classes of
50 instances each, where each class refers to a\ntype of iris plant.  One cla
ss is linearly separable from the other 2; the\nlatter are NOT linearly separ
able from each other.\n\n.. topic:: References\n\n   - Fisher, R.A. "The use
of multiple measurements in taxonomic problems"\n     Annual Eugenics, 7, Par
t II, 179-188 (1936); also in "Contributions to\n     Mathematical Statistic
s" (John Wiley, NY, 1950).\n   - Duda, R.O., & Hart, P.E. (1973) Pattern Clas
sification and Scene Analysis.\n     (Q327.D83) John Wiley & Sons.  ISBN 0-47
1-22361-1.  See page 218.\n   - Dasarathy, B.V. (1980) "Nosing Around the Nei
ghborhood: A New System\n     Structure and Classification Rule for Recogniti
on in Partially Exposed\n     Environments".  IEEE Transactions on Pattern An
alysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n   - Gate
s, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n
on Information Theory, May 1972, 431-433.\n   - See also: 1988 MLC Proceeding
s, 54-64.  Cheeseman et al"s AUTOCLASS II\n     conceptual clustering system
finds 3 classes in the data.\n   - Many, many more ...', 'feature_names': ['s
epal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (c
m)'], 'filename': 'C:\\Users\\Alekhya\\Anaconda3\\lib\\site-packages\\sklearn
\\datasets\\data\\iris.csv'}
```

In [6]:

```
1  input_data = pd.DataFrame(iris.data,columns = ['sepal length (cm)', 'sepal w
2                                                  'petal length (cm)', 'petal w
3  input_data.head()
```

Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [10]:
```
output_data = pd.DataFrame(iris.target,columns = ["target"])
output_data.head()
```

Out[10]:

	target
0	0
1	0
2	0
3	0
4	0

In [11]:
```
input_data.shape
```

Out[11]: (150, 4)

In [12]:
```
output_data.shape
```

Out[12]: (150, 1)

In [13]:
```
input_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
dtypes: float64(4)
memory usage: 4.8 KB
```

In [15]:
```
output_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 1 columns):
target    150 non-null int32
dtypes: int32(1)
memory usage: 680.0 bytes
```

In [16]:
```
input_data.isnull().sum()
```

Out[16]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
dtype: int64
```

In [17]:
```
1 output_data.isnull().sum()
```

Out[17]:
```
target    0
dtype: int64
```

In [18]:
```
1 #spliting the data for testing and trainig
```

In [19]:
```
1 from sklearn.model_selection import train_test_split
```

In [20]:
```
1 x_train,x_test,y_train,y_test = train_test_split(input_data,output_data,
2                                                  test_size=30,random_state=3
```

In [21]:
```
1 #import  required model
2 from sklearn.tree import DecisionTreeClassifier
3
```

In [22]:
```
1 dtc = DecisionTreeClassifier()
```

In [23]:
```
1 dtc.fit(x_train,y_train)
```

Out[23]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [24]:
```
1 help(dtc)
```

```
Help on DecisionTreeClassifier in module sklearn.tree.tree object:

class DecisionTreeClassifier(BaseDecisionTree, sklearn.base.ClassifierMixin)
 |  DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_fe
atures=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.
0, min_impurity_split=None, class_weight=None, presort=False)
 |
 |  A decision tree classifier.
 |
 |  Read more in the :ref:`User Guide <tree>`.
 |
 |  Parameters
 |  ----------
 |  criterion : string, optional (default="gini")
 |      The function to measure the quality of a split. Supported criteria ar
e
 |      "gini" for the Gini impurity and "entropy" for the information gain.
 |
 |  splitter : string, optional (default="best")
 |      The strategy used to choose the split at each node. Supported
 |      strategies are "best" to choose the best split and "random" to choose
 |      the best random split.
 |
 |  max_depth : int or None, optional (default=None)
 |      The maximum depth of the tree. If None, then nodes are expanded until
 |      all leaves are pure or until all leaves contain less than
 |      min_samples_split samples.
 |
 |  min_samples_split : int, float, optional (default=2)
 |      The minimum number of samples required to split an internal node:
 |
 |      - If int, then consider `min_samples_split` as the minimum number.
 |      - If float, then `min_samples_split` is a fraction and
 |        `ceil(min_samples_split * n_samples)` are the minimum
 |        number of samples for each split.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
 |
 |  min_samples_leaf : int, float, optional (default=1)
 |      The minimum number of samples required to be at a leaf node.
 |      A split point at any depth will only be considered if it leaves at
 |      least ``min_samples_leaf`` training samples in each of the left and
 |      right branches.  This may have the effect of smoothing the model,
 |      especially in regression.
 |
 |      - If int, then consider `min_samples_leaf` as the minimum number.
 |      - If float, then `min_samples_leaf` is a fraction and
 |        `ceil(min_samples_leaf * n_samples)` are the minimum
 |        number of samples for each node.
 |
 |      .. versionchanged:: 0.18
 |         Added float values for fractions.
```

```
    |
    |  min_weight_fraction_leaf : float, optional (default=0.)
    |      The minimum weighted fraction of the sum total of weights (of all
    |      the input samples) required to be at a leaf node. Samples have
    |      equal weight when sample_weight is not provided.
    |
    |  max_features : int, float, string or None, optional (default=None)
    |      The number of features to consider when looking for the best split:
    |
    |          - If int, then consider `max_features` features at each split.
    |          - If float, then `max_features` is a fraction and
    |            `int(max_features * n_features)` features are considered at eac
h
    |            split.
    |          - If "auto", then `max_features=sqrt(n_features)`.
    |          - If "sqrt", then `max_features=sqrt(n_features)`.
    |          - If "log2", then `max_features=log2(n_features)`.
    |          - If None, then `max_features=n_features`.
    |
    |      Note: the search for a split does not stop until at least one
    |      valid partition of the node samples is found, even if it requires to
    |      effectively inspect more than ``max_features`` features.
    |
    |  random_state : int, RandomState instance or None, optional (default=None)
    |      If int, random_state is the seed used by the random number generator;
    |      If RandomState instance, random_state is the random number generator;
    |      If None, the random number generator is the RandomState instance used
    |      by `np.random`.
    |
    |  max_leaf_nodes : int or None, optional (default=None)
    |      Grow a tree with ``max_leaf_nodes`` in best-first fashion.
    |      Best nodes are defined as relative reduction in impurity.
    |      If None then unlimited number of leaf nodes.
    |
    |  min_impurity_decrease : float, optional (default=0.)
    |      A node will be split if this split induces a decrease of the impurity
    |      greater than or equal to this value.
    |
    |      The weighted impurity decrease equation is the following::
    |
    |          N_t / N * (impurity - N_t_R / N_t * right_impurity
    |                              - N_t_L / N_t * left_impurity)
    |
    |      where ``N`` is the total number of samples, ``N_t`` is the number of
    |      samples at the current node, ``N_t_L`` is the number of samples in th
e
    |      left child, and ``N_t_R`` is the number of samples in the right chil
d.
    |
    |      ``N``, ``N_t``, ``N_t_R`` and ``N_t_L`` all refer to the weighted su
m,
    |      if ``sample_weight`` is passed.
    |
    |      .. versionadded:: 0.19
    |
    |  min_impurity_split : float, (default=1e-7)
    |      Threshold for early stopping in tree growth. A node will split
```

```
|      if its impurity is above the threshold, otherwise it is a leaf.
|
|      .. deprecated:: 0.19
|         ``min_impurity_split`` has been deprecated in favor of
|         ``min_impurity_decrease`` in 0.19. The default value of
|         ``min_impurity_split`` will change from 1e-7 to 0 in 0.23 and it
|         will be removed in 0.25. Use ``min_impurity_decrease`` instead.
|
|  class_weight : dict, list of dicts, "balanced" or None, default=None
|      Weights associated with classes in the form ``{class_label: weight}``
`.
|      If not given, all classes are supposed to have weight one. For
|      multi-output problems, a list of dicts can be provided in the same
|      order as the columns of y.
|
|      Note that for multioutput (including multilabel) weights should be
|      defined for each class of every column in its own dict. For example,
|      for four-class multilabel classification weights should be
|      [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] instead of
|      [{1:1}, {2:5}, {3:1}, {4:1}].
|
|      The "balanced" mode uses the values of y to automatically adjust
|      weights inversely proportional to class frequencies in the input data
|      as ``n_samples / (n_classes * np.bincount(y))``
|
|      For multi-output, the weights of each column of y will be multiplied.
|
|      Note that these weights will be multiplied with sample_weight (passed
|      through the fit method) if sample_weight is specified.
|
|  presort : bool, optional (default=False)
|      Whether to presort the data to speed up the finding of best splits in
|      fitting. For the default settings of a decision tree on large
|      datasets, setting this to true may slow down the training process.
|      When using either a smaller dataset or a restricted depth, this may
|      speed up the training.
|
|  Attributes
|  ----------
|  classes_ : array of shape = [n_classes] or a list of such arrays
|      The classes labels (single output problem),
|      or a list of arrays of class labels (multi-output problem).
|
|  feature_importances_ : array of shape = [n_features]
|      The feature importances. The higher, the more important the
|      feature. The importance of a feature is computed as the (normalized)
|      total reduction of the criterion brought by that feature.  It is also
|      known as the Gini importance [4]_.
|
|  max_features_ : int,
|      The inferred value of max_features.
|
|  n_classes_ : int or list
|      The number of classes (for single output problems),
|      or a list containing the number of classes for each
|      output (for multi-output problems).
|
```

```
|  n_features_ : int
|      The number of features when ``fit`` is performed.
|
|  n_outputs_ : int
|      The number of outputs when ``fit`` is performed.
|
|  tree_ : Tree object
|      The underlying Tree object. Please refer to
|      ``help(sklearn.tree._tree.Tree)`` for attributes of Tree object and
|      :ref:`sphx_glr_auto_examples_tree_plot_unveil_tree_structure.py`
|      for basic usage of these attributes.
|
|  Notes
|  -----
|  The default values for the parameters controlling the size of the trees
|  (e.g. ``max_depth``, ``min_samples_leaf``, etc.) lead to fully grown and
|  unpruned trees which can potentially be very large on some data sets. To
|  reduce memory consumption, the complexity and size of the trees should be
|  controlled by setting those parameter values.
|
|  The features are always randomly permuted at each split. Therefore,
|  the best found split may vary, even with the same training data and
|  ``max_features=n_features``, if the improvement of the criterion is
|  identical for several splits enumerated during the search of the best
|  split. To obtain a deterministic behaviour during fitting,
|  ``random_state`` has to be fixed.
|
|  See also
|  --------
|  DecisionTreeRegressor
|
|  References
|  ----------
|
|  .. [1] https://en.wikipedia.org/wiki/Decision_tree_learning (https://en.w
ikipedia.org/wiki/Decision_tree_learning)
|
|  .. [2] L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification
|          and Regression Trees", Wadsworth, Belmont, CA, 1984.
|
|  .. [3] T. Hastie, R. Tibshirani and J. Friedman. "Elements of Statistical
|          Learning", Springer, 2009.
|
|  .. [4] L. Breiman, and A. Cutler, "Random Forests",
|          https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
 (https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm)
|
|  Examples
|  --------
|  >>> from sklearn.datasets import load_iris
|  >>> from sklearn.model_selection import cross_val_score
|  >>> from sklearn.tree import DecisionTreeClassifier
|  >>> clf = DecisionTreeClassifier(random_state=0)
|  >>> iris = load_iris()
|  >>> cross_val_score(clf, iris.data, iris.target, cv=10)
|  ...                             # doctest: +SKIP
|  ...
```

```
 |  array([ 1.     ,  0.93...,  0.86...,  0.93...,  0.93...,
 |          0.93...,  0.93...,  1.     ,  0.93...,  1.      ])
 |
 |  Method resolution order:
 |      DecisionTreeClassifier
 |      BaseDecisionTree
 |      abc.NewBase
 |      sklearn.base.BaseEstimator
 |      sklearn.base.ClassifierMixin
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, criterion='gini', splitter='best', max_depth=None, min_sam
ples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=
None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_
impurity_split=None, class_weight=None, presort=False)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y, sample_weight=None, check_input=True, X_idx_sorted=None)
 |      Build a decision tree classifier from the training set (X, y).
 |
 |      Parameters
 |      ----------
 |      X : array-like or sparse matrix, shape = [n_samples, n_features]
 |          The training input samples. Internally, it will be converted to
 |          ``dtype=np.float32`` and if a sparse matrix is provided
 |          to a sparse ``csc_matrix``.
 |
 |      y : array-like, shape = [n_samples] or [n_samples, n_outputs]
 |          The target values (class labels) as integers or strings.
 |
 |      sample_weight : array-like, shape = [n_samples] or None
 |          Sample weights. If None, then samples are equally weighted. Split
s
 |          that would create child nodes with net zero or negative weight ar
e
 |          ignored while searching for a split in each node. Splits are also
 |          ignored if they would result in any single class carrying a
 |          negative weight in either child node.
 |
 |      check_input : boolean, (default=True)
 |          Allow to bypass several input checking.
 |          Don't use this parameter unless you know what you do.
 |
 |      X_idx_sorted : array-like, shape = [n_samples, n_features], optional
 |          The indexes of the sorted training input samples. If many tree
 |          are grown on the same dataset, this allows the ordering to be
 |          cached between trees. If None, the data will be sorted here.
 |          Don't use this parameter unless you know what to do.
 |
 |      Returns
 |      -------
 |      self : object
 |
 |  predict_log_proba(self, X)
 |      Predict class log-probabilities of the input samples X.
```

```
|
|      Parameters
|      ----------
|      X : array-like or sparse matrix of shape = [n_samples, n_features]
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      Returns
|      -------
|      p : array of shape = [n_samples, n_classes], or a list of n_outputs
|          such arrays if n_outputs > 1.
|          The class log-probabilities of the input samples. The order of th
e
|          classes corresponds to that in the attribute `classes_`.
|
|  predict_proba(self, X, check_input=True)
|      Predict class probabilities of the input samples X.
|
|      The predicted class probability is the fraction of samples of the sam
e
|      class in a leaf.
|
|      check_input : boolean, (default=True)
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      Parameters
|      ----------
|      X : array-like or sparse matrix of shape = [n_samples, n_features]
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : bool
|          Run check_array on X.
|
|      Returns
|      -------
|      p : array of shape = [n_samples, n_classes], or a list of n_outputs
|          such arrays if n_outputs > 1.
|          The class probabilities of the input samples. The order of the
|          classes corresponds to that in the attribute `classes_`.
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __abstractmethods__ = frozenset()
|
|  ----------------------------------------------------------------------
|  Methods inherited from BaseDecisionTree:
|
|  apply(self, X, check_input=True)
|      Returns the index of the leaf that each sample is predicted as.
|
|      .. versionadded:: 0.17
|
```

```
|      Parameters
|      ----------
|      X : array_like or sparse matrix, shape = [n_samples, n_features]
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : boolean, (default=True)
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      Returns
|      -------
|      X_leaves : array_like, shape = [n_samples,]
|          For each datapoint x in X, return the index of the leaf x
|          ends up in. Leaves are numbered within
|          ``[0; self.tree_.node_count)``, possibly with gaps in the
|          numbering.
|
|  decision_path(self, X, check_input=True)
|      Return the decision path in the tree
|
|      .. versionadded:: 0.18
|
|      Parameters
|      ----------
|      X : array_like or sparse matrix, shape = [n_samples, n_features]
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : boolean, (default=True)
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      Returns
|      -------
|      indicator : sparse csr array, shape = [n_samples, n_nodes]
|          Return a node indicator matrix where non zero elements
|          indicates that the samples goes through the nodes.
|
|  predict(self, X, check_input=True)
|      Predict class or regression value for X.
|
|      For a classification model, the predicted class for each sample in X
is
|      returned. For a regression model, the predicted value based on X is
|      returned.
|
|      Parameters
|      ----------
|      X : array-like or sparse matrix of shape = [n_samples, n_features]
|          The input samples. Internally, it will be converted to
|          ``dtype=np.float32`` and if a sparse matrix is provided
|          to a sparse ``csr_matrix``.
|
|      check_input : boolean, (default=True)
```

```
|          Allow to bypass several input checking.
|          Don't use this parameter unless you know what you do.
|
|      Returns
|      -------
|      y : array of shape = [n_samples] or [n_samples, n_outputs]
|          The predicted classes, or the predict values.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from BaseDecisionTree:
|
|  feature_importances_
|      Return the feature importances.
|
|      The importance of a feature is computed as the (normalized) total
|      reduction of the criterion brought by that feature.
|      It is also known as the Gini importance.
|
|      Returns
|      -------
|      feature_importances_ : array, shape = [n_features]
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.BaseEstimator:
|
|  __getstate__(self)
|
|  __repr__(self)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : boolean, optional
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : mapping of string to any
|          Parameter names mapped to their values.
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as pipelines). The latter have parameters of the form
|      ``<component>__<parameter>`` so that it's possible to update each
|      component of a nested object.
|
|      Returns
|      -------
|      self
```

```
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from sklearn.base.BaseEstimator:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from sklearn.base.ClassifierMixin:
|
|  score(self, X, y, sample_weight=None)
|      Returns the mean accuracy on the given test data and labels.
|
|      In multi-label classification, this is the subset accuracy
|      which is a harsh metric since you require for each sample that
|      each label set be correctly predicted.
|
|      Parameters
|      ----------
|      X : array-like, shape = (n_samples, n_features)
|          Test samples.
|
|      y : array-like, shape = (n_samples) or (n_samples, n_outputs)
|          True labels for X.
|
|      sample_weight : array-like, shape = [n_samples], optional
|          Sample weights.
|
|      Returns
|      -------
|      score : float
|          Mean accuracy of self.predict(X) wrt. y.
```

In [25]:
```
1 #predict the values
```

In [35]:
```
1 pred = dtc.predict(x_test)
2 pred
```

Out[35]:
```
array([0, 0, 0, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 2, 0, 1, 2, 2, 0, 2, 2,
       2, 1, 0, 2, 1, 1, 1, 1])
```

In [27]:
```
1 from sklearn.metrics import accuracy_score,confusion_matrix,classification_r
```

In [37]:
```
1 accuracy_score(y_test,pred)
```

Out[37]:
```
0.9666666666666667
```

In [38]:
```
1 confusion_matrix(y_test,pred)
```

Out[38]:
```
array([[10,  0,  0],
       [ 0, 10,  0],
       [ 0,  1,  9]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.91      1.00      0.95        10
           2       1.00      0.90      0.95        10

   micro avg       0.97      0.97      0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

Graphviz

pydotplus

- pip install graphviz
- conda install graphviz
- pip install pydotplus
- conda install pydotplus

In [41]:
```
1 conda install graphviz
```

```
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: C:\Users\Alekhya\Anaconda3

  added / updated specs:
    - graphviz


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2021.5.25  |       haa95532_1         147 KB
    conda-4.10.1               |   py37haa95532_1         3.1 MB
    ------------------------------------------------------------
                                           Total:         3.2 MB

The following packages will be UPDATED:

  ca-certificates                    2021.1.19-haa95532_1 --> 2021.5.25-haa
95532_1
  conda                              4.9.2-py37haa95532_0 --> 4.10.1-py37ha
a95532_1



Downloading and Extracting Packages

conda-4.10.1         | 3.1 MB    |            |   0%
conda-4.10.1         | 3.1 MB    |            |   1%
conda-4.10.1         | 3.1 MB    | 2          |   3%
conda-4.10.1         | 3.1 MB    | 3          |   4%
conda-4.10.1         | 3.1 MB    | 5          |   6%
conda-4.10.1         | 3.1 MB    | 9          |  10%
conda-4.10.1         | 3.1 MB    | #2         |  13%
conda-4.10.1         | 3.1 MB    | #6         |  16%
conda-4.10.1         | 3.1 MB    | #9         |  19%
conda-4.10.1         | 3.1 MB    | ##4        |  24%
conda-4.10.1         | 3.1 MB    | ##9        |  29%
conda-4.10.1         | 3.1 MB    | ###3       |  33%
conda-4.10.1         | 3.1 MB    | ###7       |  37%
conda-4.10.1         | 3.1 MB    | ####7      |  47%
conda-4.10.1         | 3.1 MB    | #####2     |  52%
conda-4.10.1         | 3.1 MB    | #####6     |  57%
conda-4.10.1         | 3.1 MB    | ######     |  60%
conda-4.10.1         | 3.1 MB    | #######    |  70%
conda-4.10.1         | 3.1 MB    | #######4   |  75%
conda-4.10.1         | 3.1 MB    | #######8   |  79%
conda-4.10.1         | 3.1 MB    | ########2  |  83%
conda-4.10.1         | 3.1 MB    | #########6 |  96%
conda-4.10.1         | 3.1 MB    | ########## | 100%
```

```
ca-certificates-2021 | 147 KB    |            |   0%
ca-certificates-2021 | 147 KB    | #          |  11%
ca-certificates-2021 | 147 KB    | ########## | 100%
ca-certificates-2021 | 147 KB    | ########## | 100%
Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done

Note: you may need to restart the kernel to use updated packages.
```

In [42]:
```
1 conda install pydotplus
```

```
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

In [44]:
```
1 from sklearn.tree import export_graphviz
2 from sklearn.externals.six import StringIO
```

In [46]:
```
1 from IPython.display import Image
2 import pydotplus
```

In [47]:
```
1 help(export_graphviz)
```

```
Help on function export_graphviz in module sklearn.tree.export:

export_graphviz(decision_tree, out_file=None, max_depth=None, feature_names=Non
e, class_names=None, label='all', filled=False, leaves_parallel=False, impurity
=True, node_ids=False, proportion=False, rotate=False, rounded=False, special_c
haracters=False, precision=3)
    Export a decision tree in DOT format.

    This function generates a GraphViz representation of the decision tree,
    which is then written into `out_file`. Once exported, graphical renderings
    can be generated using, for example::

        $ dot -Tps tree.dot -o tree.ps      (PostScript format)
        $ dot -Tpng tree.dot -o tree.png    (PNG format)

    The sample counts that are shown are weighted with any sample_weights that
    might be present.

    Read more in the :ref:`User Guide <tree>`.

    Parameters
    ----------
    decision_tree : decision tree regressor or classifier
        The decision tree to be exported to GraphViz.

    out_file : file object or string, optional (default=None)
        Handle or name of the output file. If ``None``, the result is
        returned as a string.

        .. versionchanged:: 0.20
            Default of out_file changed from "tree.dot" to None.

    max_depth : int, optional (default=None)
        The maximum depth of the representation. If None, the tree is fully
        generated.

    feature_names : list of strings, optional (default=None)
        Names of each of the features.

    class_names : list of strings, bool or None, optional (default=None)
        Names of each of the target classes in ascending numerical order.
        Only relevant for classification and not supported for multi-output.
        If ``True``, shows a symbolic representation of the class name.

    label : {'all', 'root', 'none'}, optional (default='all')
        Whether to show informative labels for impurity, etc.
        Options include 'all' to show at every node, 'root' to show only at
        the top root node, or 'none' to not show at any node.

    filled : bool, optional (default=False)
        When set to ``True``, paint nodes to indicate majority class for
        classification, extremity of values for regression, or purity of node
        for multi-output.
```

```
leaves_parallel : bool, optional (default=False)
    When set to ``True``, draw all leaf nodes at the bottom of the tree.

impurity : bool, optional (default=True)
    When set to ``True``, show the impurity at each node.

node_ids : bool, optional (default=False)
    When set to ``True``, show the ID number on each node.

proportion : bool, optional (default=False)
    When set to ``True``, change the display of 'values' and/or 'samples'
    to be proportions and percentages respectively.

rotate : bool, optional (default=False)
    When set to ``True``, orient tree left to right rather than top-down.

rounded : bool, optional (default=False)
    When set to ``True``, draw node boxes with rounded corners and use
    Helvetica fonts instead of Times-Roman.

special_characters : bool, optional (default=False)
    When set to ``False``, ignore special characters for PostScript
    compatibility.

precision : int, optional (default=3)
    Number of digits of precision for floating point in the values of
    impurity, threshold and value attributes of each node.

Returns
-------
dot_data : string
    String representation of the input tree in GraphViz dot format.
    Only returned if ``out_file`` is None.

    .. versionadded:: 0.18

Examples
--------
>>> from sklearn.datasets import load_iris
>>> from sklearn import tree

>>> clf = tree.DecisionTreeClassifier()
>>> iris = load_iris()

>>> clf = clf.fit(iris.data, iris.target)
>>> tree.export_graphviz(clf,
...     out_file='tree.dot')                # doctest: +SKIP
```

In [48]:

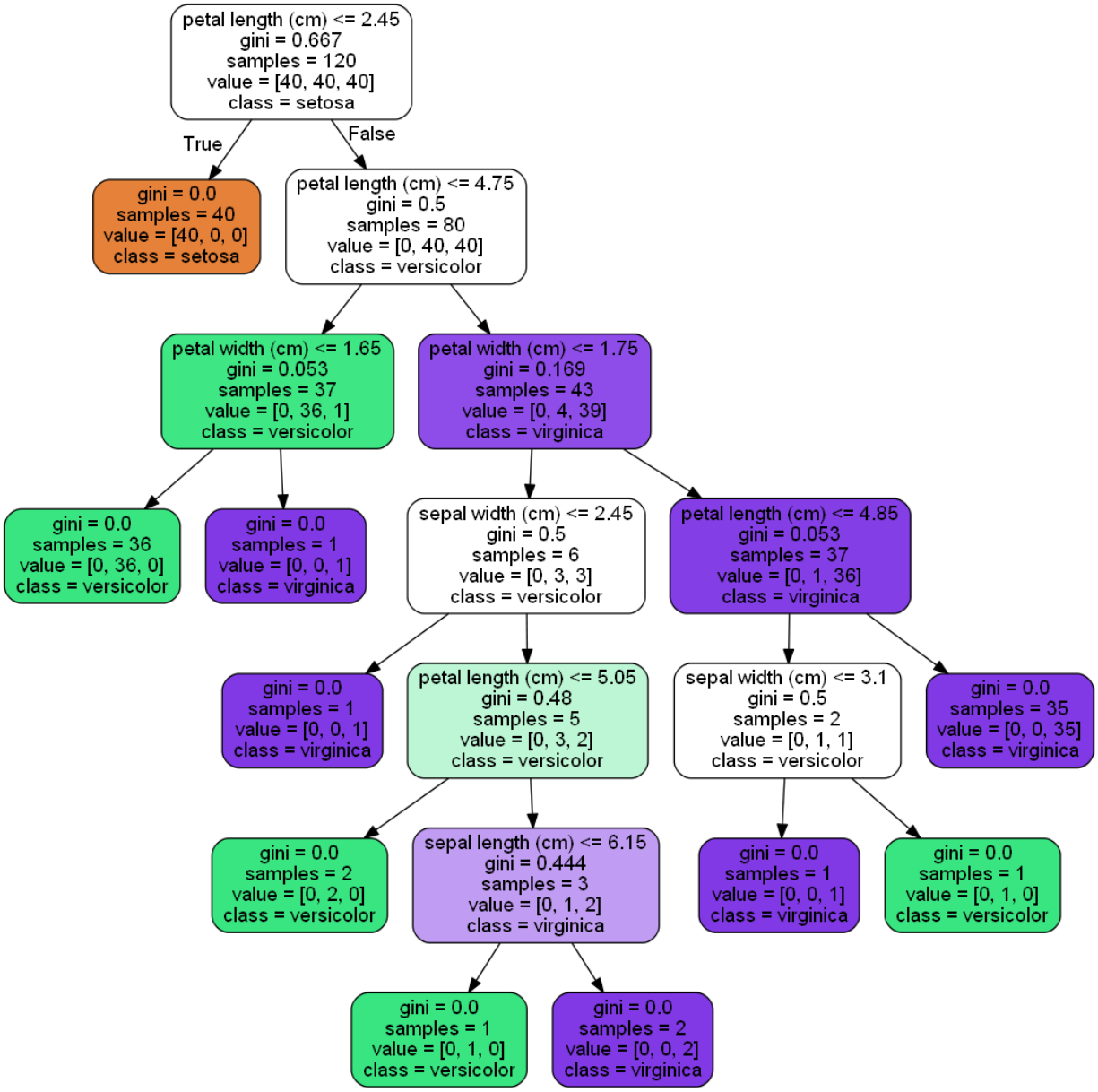
```
dot_data = StringIO()
export_graphviz(dtc,out_file = "irisimage.dot",feature_names=iris.feature_na
                class_names=iris.target_names,rounded = True,filled = True)
```

In [49]:
```
dot_data1 = StringIO()
export_graphviz(dtc,out_file = dot_data1,feature_names=iris.feature_names,
                class_names=iris.target_names,rounded = True,filled = True)
```

In [50]:
```
pyplo = pydotplus.graph_from_dot_data(dot_data1.getvalue())
```

In [51]:
```
Image(pyplo.create_png())
```

Out[51]:



- apply decision tree classifier for below dataset

link (https://www.kaggle.com/abbasit/kyphosis-dataset)

In [ ]:
```

```