In [1]:
```python
## importing all needed modules
from sklearn import datasets
import pandas as pd
import math
import numpy as np

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import pydotplus
```

From here onwanrds Data Preparing takes place where we first add the column names and then convert the continuous data values into discreet data values. Then we drop the unnecessary columns. To summarize it, we perform Data Cleaning.

In [2]:
```python
## loading the dataset here
iris = datasets.load_iris()
```

In [3]:
```python
## Printing the dataset alongwith column names
df = pd.DataFrame(iris.data)
df.columns = ["sl", "sw", 'pl', 'pw']
print(df.head(5))
```

```
    sl   sw   pl   pw
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 [4]:
```python
#Function to find label for a value
#if MIN_Value <=val < (m + Mean_Value) / 2 then it is assigned label a
#if (m + Mean_Value) <=val < Mean_Value then it is assigned label b
#if (Mean_Value) <=vadf, y, unused_featuresl < (Mean_Value + MAX_Value)/2 then it is assigned label c
#if (Mean_Value + MAX_Value)/2 <=val <= MAX_Value  then it is assigned label d

def label(val, *boundaries):
    if (val < boundaries[0]):
        return 'a'
    elif (val < boundaries[1]):
        return 'b'
    elif (val < boundaries[2]):
        return 'c'
    else:
        return 'd'

#Function to convert a continuous data into labelled data
#There are 4 lables  - a, b, c, d
def toLabel(df, old_feature_name):
    second = df[old_feature_name].mean()
    minimum = df[old_feature_name].min()
    first = (minimum + second)/2
    maximum = df[old_feature_name].max()
    third = (maximum + second)/2
    return df[old_feature_name].apply(label, args= (first, second, third))
```

In [5]:
```python
#Convert all columns to labelled data
df['sl_labeled'] = toLabel(df, 'sl')
df['sw_labeled'] = toLabel(df, 'sw')
df['pl_labeled'] = toLabel(df, 'pl')
df['pw_labeled'] = toLabel(df, 'pw')
df.head(10)
```

Out[5]:

	sl	sw	pl	pw	sl_labeled	sw_labeled	pl_labeled	pw_labeled
0	5.1	3.5	1.4	0.2	b	c	a	a
1	4.9	3.0	1.4	0.2	a	b	a	a
2	4.7	3.2	1.3	0.2	a	c	a	a
3	4.6	3.1	1.5	0.2	a	c	a	a
4	5.0	3.6	1.4	0.2	a	c	a	a
5	5.4	3.9	1.7	0.4	b	d	a	a
6	4.6	3.4	1.4	0.3	a	c	a	a
7	5.0	3.4	1.5	0.2	a	c	a	a
8	4.4	2.9	1.4	0.2	a	b	a	a
9	4.9	3.1	1.5	0.1	a	c	a	a

In [6]:
```python
df.drop(['sl', 'sw', 'pl', 'pw'], axis = 1, inplace = True)
```

In [7]:
```python
set(df['sl_labeled'])
```

Out[7]: {'a', 'b', 'c', 'd'}

Here onwards the Decision Tree Implementation takes place. We also store the node values for the tree in a dictionary.

In [8]:
```python
## Here a dictionary is initialized in which we will store the values for decision tree nodes in order to
## print them later. Our dictionary name is "dictionary"
dictionary = {}
dictionary[0] = [50,50,50]
print(type(dictionary))

## Storing the best features in a list bf at each level
bf = []
```

```
<class 'dict'>
```

In [9]:
```python
## This is a function to calculate the entropy of a node. Entropy is also called information(info) of node n
def find_info(l):
    s = l[0]+l[1]+l[2]
    if s==0:
        return 0

    # Here p1, p2, p3 represent the probabilities of the three types of flowers,
    # i.e., Iris-setosa,Iris-versicolor, Iris-virginica
    p1 = l[0]/s
    p2 = l[1]/s
    p3 = l[2]/s

    # Here we check that no probability non positive otherwise
    # it will be inconvinient to take log off that probability
    if p1<=0:
        p1 = 1
    if p2<=0:
        p2 = 1
    if p3<=0:
        p3 = 1

    i = -p1*math.log(p1,2)-p2*math.log(p2,2)-p3*math.log(p3,2)
    i = 0.0 if abs(i)==0.0 else i
    return i
```

In [10]:
```python
def build_tree(df, y, unused_features,it,s):
#     if it>0:
#         print(df)
#         print(y)

    ## BASE CASES
    # 1. unused is empty,i.e., all the features have been splitted upon
    if len(unused_features)==0:

        # the list l stores the values for Iris-setosa,Iris-versicolor, Iris-virginica respectively
        l = [0,0,0]
        for i in y:
            l[i]+=1
        en = find_info(l)

        # printing the required information
        print("Level",it)
        print("Count of 0(Iris-setosa)",l[0])
        print("Count of 1(Iris-versicolor)",l[1])
        print("Count of 2(Iris-virginica)",l[2])
        print("Entropy of the current Node is",en)
        print("Reached Leaf Node")
        print()

        # Here we are updating the dictionary for the node values
        if it>=1:
            dictionary[it][s].append(l[0])
            dictionary[it][s].append(l[1])
            dictionary[it][s].append(l[2])

        return

    # 2. y contains only one distinct value,i.e., pure node
    if len(set(y))==1:

        # the list l stores the values for Iris-setosa,Iris-versicolor, Iris-virginica respectively
        l = [0,0,0]
        for i in y:
            l[i]+=1
        en = find_info(l)

        # printing the required information
        print("Level",it)
        print("Count of 0(Iris-setosa)",l[0])
        print("Count of 1(Iris-versicolor)",l[1])
        print("Count of 2(Iris-virginica)",l[2])
        print("Entropy of the current Node is",en)
        print("Reached Leaf Node")
        print()

        # Here we are updating the dictionary for the node values
        if it>=1:
            dictionary[it][s].append(l[0])
            dictionary[it][s].append(l[1])
            dictionary[it][s].append(l[2])

        return

    ## RECURSIVE CASE
    # initializing the required data members
    best_feature = ""
    max_gain =-1
    Iris_setosa = 0
    Iris_versicolor = 0
    Iris_virginica = 0
    en = 0

    # iterating over all the features in the list unused_features
    # This list contains the features left to be splitted upon
    for f in unused_features:
        # print(f)

        # Here a,b,c,d are the four types of values in our dataset
        # They will store three values for Iris-setosa,Iris-versicolor, Iris-virginica respectively
        a = [0,0,0]
        b = [0,0,0]
        c = [0,0,0]
        d = [0,0,0]
        x = 0

        # iterating over the rows of the dataset df
        for i in df.index:
            # print(df[f].iloc[i],y[i])
            if df[f].loc[i]=='a':
                a[y[x]] += 1
            elif df[f].loc[i]=='b':
                b[y[x]] += 1
            elif df[f].loc[i]=='c':
                c[y[x]] += 1
            else:
                d[y[x]] += 1
            x+=1

        # print(a,b,c,d)

        total_a = a[0]+a[1]+a[2]            #total values of a, i.e., Iris-setosa,Iris-versicolor, Iris-virgini
        total_b = b[0]+b[1]+b[2]            #total values of b, i.e., Iris-setosa,Iris-versicolor, Iris-virgini
        total_c = c[0]+c[1]+c[2]            #total values of c, i.e., Iris-setosa,Iris-versicolor, Iris-virgini
        total_d = d[0]+d[1]+d[2]            #total values of d, i.e., Iris-setosa,Iris-versicolor, Iris-virgini

        total = total_a + total_b + total_c + total_d      # total of all values

        # the list l stores the values for Iris-setosa,Iris-versicolor, Iris-virginica respectively
        l = [a[0]+b[0]+c[0]+d[0],a[1]+b[1]+c[1]+d[1],a[2]+b[2]+c[2]+d[2]]
        entropy = find_info(l)

        # These info are for finding the entropy after split in order to calculate info gain later
        info_a = find_info(a)
        info_b = find_info(b)
        info_c = find_info(c)
        info_d = find_info(d)

        # Entropy after split
        info_f = ((abs(total_a)/abs(total))*info_a + (abs(total_b)/abs(total))*info_b +
                  (abs(total_c)/abs(total))*info_c + (abs(total_d)/abs(total))*info_d)

        info_gain = entropy - info_f    # information gain

        # These variables are used to calculate split info, i.e., numerator of gain ratio
        t1 = abs(total_a)/abs(total)
        t2 = abs(total_b)/abs(total)
        t3 = abs(total_c)/abs(total)
        t4 = abs(total_d)/abs(total)

        if t1<=0:
            t1 = 1
        if t2<=0:
            t2 = 1
        if t3<=0:
            t3 = 1
        if t4<=0:
            t4 = 1

        split_info = (-(abs(total_a)/abs(total))*math.log(t1,2) -
                      (abs(total_b)/abs(total))*math.log(t2,2) -
                      (abs(total_c)/abs(total))*math.log(t3,2) -
                      (abs(total_d)/abs(total))*math.log(t4,2))


        gain_ratio = info_gain/split_info       #Calculating gain ratio to decide for the split

        # For finding the best feature to split upon
        if gain_ratio>max_gain:
            max_gain = gain_ratio
            best_feature = f
            Iris_setosa = a[0]+b[0]+c[0]+d[0]
            Iris_versicolor = a[1]+b[1]+c[1]+d[1]
            Iris_virginica = a[2]+b[2]+c[2]+d[2]
            en = entropy

    # here we know the best feature
    # so we print it out
    print("Level",it)
    print("Count of 0(Iris-setosa)",Iris_setosa)
    print("Count of 1(Iris-versicolor)",Iris_versicolor)
    print("Count of 2(Iris-virginica)",Iris_virginica)
    print("Entropy of the current Node is",en)
    print("Splitting on feature", best_feature, "with gain ratio", max_gain)
    print()

    # Storing the best feature in the list bf
    bf.append(best_feature)

    # updating the dictionary for node values according to their levels
    if it+1 not in dictionary.keys():
        dictionary[it+1] = {}

    # Here we are updating the dictionary for the node values
    if it>=1:
        dictionary[it][s].append(Iris_setosa)
        dictionary[it][s].append(Iris_versicolor)
        dictionary[it][s].append(Iris_virginica)


    # remove best feature from unused features
    unused_features.remove(best_feature)

    # call build tree recursively
    # but also checking if that value is present in the generated dataset or not

    if df.loc[df[best_feature]=='a'].shape[0]!=0:
        # print("a")
        dictionary[it+1]["a"] = []
        build_tree(df.loc[df[best_feature]=='a'],y[df[best_feature]=='a'],unused_features,it+1,"a")

    if df.loc[df[best_feature]=='b'].shape[0]!=0:
        # print("b")
        dictionary[it+1]["b"] = []
        build_tree(df.loc[df[best_feature]=='b'],y[df[best_feature]=='b'],unused_features,it+1,"b")

    if df.loc[df[best_feature]=='c'].shape[0]!=0:
        # print("c")
        dictionary[it+1]["c"] = []
        build_tree(df.loc[df[best_feature]=='c'],y[df[best_feature]=='c'],unused_features,it+1,"c")

    if df.loc[df[best_feature]=='d'].shape[0]!=0:
        # print("d")
        dictionary[it+1]["d"] = []
        build_tree(df.loc[df[best_feature]=='d'],y[df[best_feature]=='d'],unused_features,it+1,"d")
```

In [11]:
```python
y = pd.DataFrame(iris.target)

## Converting y into the desired format
l = []
for i in y.values:
    l.append(i[0])
y = l
y = np.array(y)

## getting the list of all possible features
unused_features = set(df.columns)
# print(unused_features)

## Calling the driver function
# here the last two parameters are the level no and feature value(for printing the tree) respectively
build_tree(df, y, unused_features,0,"a")
```

```
Level 0
Count of 0(Iris-setosa) 50
Count of 1(Iris-versicolor) 50
Count of 2(Iris-virginica) 50
Entropy of the current Node is 1.584962500721156
Splitting on feature pw_labeled with gain ratio 0.699638203622209

Level 1
Count of 0(Iris-setosa) 50
Count of 1(Iris-versicolor) 0
Count of 2(Iris-virginica) 0
Entropy of the current Node is 0.0
Reached Leaf Node

Level 1
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 10
Count of 2(Iris-virginica) 0
Entropy of the current Node is 0.0
Reached Leaf Node

Level 1
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 40
Count of 2(Iris-virginica) 16
Entropy of the current Node is 0.863120568566631
Splitting on feature pl_labeled with gain ratio 0.4334099495621066

Level 2
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 1
Count of 2(Iris-virginica) 0
Entropy of the current Node is 0.0
Reached Leaf Node

Level 2
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 39
Count of 2(Iris-virginica) 8
Entropy of the current Node is 0.6581912658132185
Splitting on feature sl_labeled with gain ratio 0.12674503775809332

Level 3
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 0
Count of 2(Iris-virginica) 1
Entropy of the current Node is 0.0
Reached Leaf Node

Level 3
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 14
Count of 2(Iris-virginica) 0
Entropy of the current Node is 0.0
Reached Leaf Node

Level 3
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 23
Count of 2(Iris-virginica) 7
Entropy of the current Node is 0.783776947484701
Splitting on feature sw_labeled with gain ratio 0.07092036405148876

Level 4
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 3
Count of 2(Iris-virginica) 1
Entropy of the current Node is 0.8112781244591328
Reached Leaf Node

Level 4
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 14
Count of 2(Iris-virginica) 6
Entropy of the current Node is 0.8812908992306927
Reached Leaf Node

Level 4
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 6
Count of 2(Iris-virginica) 0
Entropy of the current Node is 0.0
Reached Leaf Node

Level 3
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 2
Count of 2(Iris-virginica) 0
Entropy of the current Node is 0.0
Reached Leaf Node

Level 2
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 0
Count of 2(Iris-virginica) 8
Entropy of the current Node is 0.0
Reached Leaf Node

Level 1
Count of 0(Iris-setosa) 0
Count of 1(Iris-versicolor) 0
Count of 2(Iris-virginica) 34
Entropy of the current Node is 0.0
Reached Leaf Node
```

In [12]:
```python
## Printing the dictionary to show the tree generated

for i in dictionary:
    print("Level", end=" ")
    print(i,bf[i] if i<len(bf) else "")
    print(dictionary[i])
```

```
Level 0 pw_labeled
[50, 50, 50]
Level 1 pl_labeled
{'a': [50, 0, 0], 'b': [0, 10, 0], 'c': [0, 40, 16], 'd': [0, 0, 34]}
Level 2 sl_labeled
{'b': [0, 1, 0], 'c': [0, 39, 8], 'd': [0, 0, 8]}
Level 3 sw_labeled
{'a': [0, 0, 1], 'b': [0, 14, 0], 'c': [0, 23, 7], 'd': [0, 2, 0]}
Level 4
{'a': [0, 3, 1], 'b': [0, 14, 6], 'c': [0, 6, 0]}
```

Decision Tree For covinience, I will copy the dataset df into df1 and again label the continuous values into discreet ones But here, instead of using class labels as a,b,c,d; I will use class labels as 1,2,3,4

In [13]:
```python
df1 = pd.DataFrame(iris.data)
df1.columns = ["sl", "sw", 'pl', 'pw']
print(df1.head(5))
```

```
    sl   sw   pl   pw
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 [14]:
```python
#Function to find label for a value
#if MIN_Value <=val < (m + Mean_Value) / 2 then it is assigned label 1
#if (m + Mean_Value) <=val < Mean_Value then it is assigned label 2
#if (Mean_Value) <=vadf, y, unused_featuresl < (Mean_Value + MAX_Value)/2 then it is assigned label 3
#if (Mean_Value + MAX_Value)/2 <=val <= MAX_Value  then it is assigned label 4

def label(val, *boundaries):
    if (val < boundaries[0]):
        return 1
    elif (val < boundaries[1]):
        return 2
    elif (val < boundaries[2]):
        return 3
    else:
        return 4

def toLabel(df1, old_feature_name):
    second = df1[old_feature_name].mean()
    minimum = df1[old_feature_name].min()
    first = (minimum + second)/2
    maximum = df1[old_feature_name].max()
    third = (maximum + second)/2
    return df1[old_feature_name].apply(label, args= (first, second, third))
```

In [15]:
```python
#Convert all columns to labelled data
df1['sl_labeled'] = toLabel(df1, 'sl')     #relabeling columns with discrete values and their header name
df1['sw_labeled'] = toLabel(df1, 'sw')
df1['pl_labeled'] = toLabel(df1, 'pl')
df1['pw_labeled'] = toLabel(df1, 'pw')
df1.head()
```

Out[15]:

	sl	sw	pl	pw	sl_labeled	sw_labeled	pl_labeled	pw_labeled
0	5.1	3.5	1.4	0.2	2	3	1	1
1	4.9	3.0	1.4	0.2	1	2	1	1
2	4.7	3.2	1.3	0.2	1	3	1	1
3	4.6	3.1	1.5	0.2	1	3	1	1
4	5.0	3.6	1.4	0.2	1	3	1	1

In [16]:
```python
df1.drop(['sl', 'sw', 'pl', 'pw'], axis = 1, inplace = True)
print(df1.head())
```

```
   sl_labeled  sw_labeled  pl_labeled  pw_labeled
0           2           3           1           1
1           1           2           1           1
2           1           3           1           1
3           1           3           1           1
4           1           3           1           1
```

In [17]:
```python
clf = DecisionTreeClassifier()
clf.fit(df1,y)
```

Out[17]: DecisionTreeClassifier()

In [18]:
```python
#Printing the decision tree formed using pydotplus

dot_data = export_graphviz(clf,out_file=None,filled=True,rounded=True,
                           feature_names=iris.feature_names,special_characters=True,
                           class_names=iris.target_names)
graph = pydotplus.graph_from_dot_data(dot_data)
graph.write_pdf("iris.pdf")
```

Out[18]: True

In [ ]:
```python
n = int(input())
count = 1
current = 1
while(count <= n):
        num = 3 * current + 2
        if num % 4 != 0 :
                print(num, end=" ")
                count += 1
        current += 1
```

In [ ]:
```python
def termsAp(n):
    li = []
    for x in range(1, n + 1000, 1):
        num = 3 * x + 2
        if num % 4 != 0:
            li.append(num)
    for i in range(n):
        print(li[i], end=" ")


n = int(input())
termsAp(n)
```

In [ ]: