```
#!/usr/bin/python3
#import pydotplus
#from sklearn.datasets import load_iris
#from sklearn import tree
#import collections
from math import log
import random
import operator
import math

standard = 4
testData = []

def createDataSet():
#    testData = []
#[Name, age, hotness, working?, num of interests, height, wealthy, decision]
    testData = []
    trainingData = [[1, 23, 2, 1, 2, 148, 0, 1],    #1 = Ava
                    [2, 38, 1, 1, 5, 168, 1, 0],    #2 = Isabella
                    [3, 26, 3, 1, 3, 160, 1, 1],    #3 = Emma
                    [4, 20, 5, 0, 2, 158, 1, 1],    #4 = Olivia
                    [5, 21, 4, 1, 3, 162, 1, 1],    #5 = Taylor
                    [6, 28, 4, 1, 2, 165, 1, 1],    #6 = Emily
                    [7, 35, 2, 0, 1, 145, 0, 0],    #7 = Madison
                    [8, 30, 5, 1, 1, 155, 0, 1],    #8 = Mia
                    [9, 18, 1, 0, 2, 159, 0, 0],    #9 = Ella
                    [10, 19, 0, 0, 0, 168, 1, 0],   #10= Natalie
                    [11, 17, 3, 1, 4, 155, 0, 1],   #11= Lily
                    [12, 28, 1, 0, 1, 180, 0, 0],   #12= Samantha
                    [13, 38, 0, 1, 8, 178, 1, 0],   #13= Hannah
                    [14, 34, 4, 1, 3, 174, 1, 1],   #14= Leah
                    [15, 29, 3, 1, 3, 173, 0, 1],   #15= Jenny
                    [16, 20, 0, 0, 1, 155, 0, 0],   #16= Claire
                    [17, 22, 1, 1, 3, 148, 1, 0],   #17= Alexa
                    [18, 33, 5, 1, 0, 164, 1, 1],   #18= Layla
                    [19, 31, 3, 1, 1, 165, 0, 1],   #19= Bella
                    [20, 26, 4, 1, 3, 172, 0, 1],   #20= Maya
                    [21, 16, 1, 1, 5, 149, 0, 0],   #21= Lucy
                    [22, 20, 3, 0, 1, 167, 1, 1],   #22= Molly
                    [23, 25, 0, 1, 1, 155, 0, 0],   #23= Andrea
                    [24, 19, 3, 0, 1, 150, 0, 0],   #24= Naomi
                    [25, 28, 4, 1, 4, 168, 1, 1],   #25= Ruby
                    ]
    testData.append(trainingData.pop(random.randint(0,len(trainingData)-1)))
    testData.append(trainingData.pop(random.randint(0,len(trainingData)-1)))
    testData.append(trainingData.pop(random.randint(0,len(trainingData)-1)))
    testData.append(trainingData.pop(random.randint(0,len(trainingData)-1)))
    testData.append(trainingData.pop(random.randint(0,len(trainingData)-1)))

    #labels = ["name", "age", "hotness", "working", "numInterest", "height", "wealthy"]
    #return trainingData, testData
```

```
    return testData

def entropy(data):
    entries = len(data)
    #print ("enteries is: ",  entries)
    yesCounts = 0
    #print ("From here")
    for feat in data:   # the number of unique elements and occurance
        currentLabel = feat[-1]
        if currentLabel == 1:
            yesCounts += 1
    noCounts = entries - yesCounts

    entropy = 0.0
    if yesCounts > 0:
        entropy += -1 * (yesCounts / entries) * math.log(float(yesCounts) / entries, 2)
    if noCounts > 0:
        entropy += -1 * (noCounts / entries) * math.log(float(noCounts) / entries, 2)

    if entropy == 0:
        return False
    else:
        return entropy

def majorityCnt(classList):
    classCount = {}
    for v in classList:
        if v not in classCount.keys():
            classCount[v] = 0
            classCount[v] += 1
    sortedClassCount = sorted(classCount.iteritems(), key = operator.itemgetter(1), reverse = True)
    return sortedClassCount[0][0]

def split(data, index, value):
    x = []
    y = []
    for pt in data:
        if pt[index] > value:
            x.append(pt[:])
        else:
            y.append(pt[:])
    return (x,y)

def infoGain(data, index, value, baseEntropy):
    (grp_a, grp_b) = split(data, index, value)

    x = len(grp_a)
    y = len(grp_b)
    z = len(data)
    newInfo = (x / z) * entropy(grp_a) + (y / z) * entropy(grp_b)
    return baseEntropy - newInfo
```

```
    #return entropy(data) - newInfo

def createTree(dataSet):
    classList = [instance[-1] for instance in dataSet] # get the decision mode
    #print ("class list reverse: ", classList.reverse)
    #print (classList[0])
    #print (len(classList))

    # count list
    if(classList.count(classList[0])) == len(classList): # if 1 == 20
        return classList[0]
    if(len(dataSet[0]) == 1):   # if the amount of attribute = 1, stop slipting the tree
        hi = majorityCnt(classList)
        print (hi)
        return majorityCnt(classList)
    print (len(dataSet[0]))

    maxInfoGain = 0
    BestSplit = (0,0)
    baseEntropy = entropy(dataSet)
    print ("Base entropy: ", baseEntropy)

    for index in range(1, 7):   # range of 1 to 6
        minimum = 999999
        maximum = -999999
        for every in dataSet:
            if every[index] < minimum:
                minimum = every[index]
                #print ("minimum: ", minimum)
            if every[index] > maximum:
                maximum = every[index]
                #print ("maximum: ", maximum)
        procedure = ((maximum - minimum) / standard)

        for value in [(minimum + (procedure * t)) for t in range(0, standard)]:
            newEntropy = infoGain(dataSet, index, value, baseEntropy)
            print("New Entropy: ", newEntropy)
            if newEntropy > maxInfoGain:
                maxInfoGain = newEntropy
                BestSplit = (index, value)
    if BestSplit == (0,0): return False
    print ("Best Split: ", BestSplit, maxInfoGain)
    (i, j) = BestSplit
    treeSplit.append([i, j])
    a, b = split(dataSet, i, j)
    for i in a:
        print ("Array a is: ", i)
    print ("")
    print("The length of a array is: " + str(len(a)))
    Amade = createTree(a)
```

```
    for j in b:
        print ("Array b is: ", b)
    print("")
    print("The length of b array is:  " + str(len(a)))
    Bmade = createTree(b)

    if not Amade:
        for every in a:
            print(every[0], every[7])
    if not Bmade:
        for every in b:
            print(every[0], every[7], '\n')
    return True

treeSplit = []
#trainData, testData = createDataSet();
testData = createDataSet();
#createTree(trainData)
createTree(testData)

print("Split line ==========")
for i in treeSplit:
    print(i)
print("Test data line=========")
for i in testData:
    print (i)
```