**Data Mining Lab**

**Assignment 06**

**Group** – C

**B.Tech 6th Semester**

**Name** – Neelangshu Roy

**Reg. No.** – 20214060

Objective 1:

To design a tree-based machine learning algorithm and preprocess input dataset into a compatible format for the algorithm.

Outcome:

Students will able to learn various preprocessing techniques, design fit functions to train the model on the input dataset, and then apply the trained model for predicting unseen\unknown data into appropriate class label.

Load vehicle0.dat file in Data Frame using pandas.read\_csv (“vehicle.csv”) and prepare a user-defined decision tree module. Make suitable preprocessing in the data dataset if required.

1. Compute the split point for each attribute in the dataset using the following strategies:

a. Information Gain

b. Gini Indices

c. Gain Ratio

2. Design module for creating the decision tree and its representation in graphical format for the following cases:

a. Binary Tree (each node split into exactly two branches).

b. General Tree (each node may split into more than two branches depending on count nominal labels corresponding attributes).

3. Design module which predicts the class label of unknown and unseen data using tree traversal or any other techniques.

First, set up an example dataset to perform tree construction on.

Python Code to create a dataset and save it into a .csv file :

import math

import pandas as pd

import numpy as np

names=['Attr1','Attr2','Attr3','Attr4','Class']

data=[

    ['Yes','Low','L','Fair','Yes'],

    ['No','High','M','Excellent','Yes'],

    ['No','Medium','S','Poor','Yes'],

    ['Yes','Medium','XL','Fair','No'],

    ['Yes','High','XXL','Fair','Yes'],

    ['No','High','L','Poor','Yes'],

    ['No','Medium','M','Fair','Yes'],

    ['No','Low','M','Excellent','No'],

    ['Yes','Low','XL','Excellent','Yes'],

    ['No','Medium','XL','Fair','Yes'],

    ['Yes','High','S','Fair','Yes'],

    ['No','Low','S','Fair','No'],

    ['Yes','Medium','M','Excellent','Yes'],

    ['Yes','High','L','Excellent','No'],

    ['No','Low','XXL','Poor','Yes']

]

df=pd.DataFrame(data,columns=names)

print(df)

for i in df.columns:

    print(i) # i stores the column names

for i in df.itertuples():

    print(i)

    print('first =',i[0],'last =',i[-1]) # first is the index, then comes the data

# this can be a very easy way to iteratively navigate the data and create a custom

# dictionary with class labels as was visible through the df.columns

# that is, if the data is given in this format !

# if it's in attribute -> then the data under that column

tempfile=df.to\_csv('ass6\_temp\_data.csv')

Python Code for performing on the data :

import math

import pandas as pd

import numpy as np

def calculateEntropy(liss):

    sum=0

    for i in liss:sum+=i

    ans=0

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*math.log2(i/sum)

    return ans

def calculateGini(liss):

    sum=0

    for i in liss:sum+=i

    ans=1

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*\*2

    return ans

df=pd.read\_csv('ass6\_temp\_data.csv')

# print(df)

all\_Attr=[x for x in df.columns[1:]]

# print(all\_Attr)

all\_data=[]

for i in df.itertuples():

    # print(list(i))

    all\_data.append([x for x in i[2:]])

class\_attr\_types={}

for i in all\_data:

    if i[-1] in class\_attr\_types:

        class\_attr\_types[i[-1]]+=1

    else:class\_attr\_types[i[-1]]=1

info\_dataset=calculateEntropy([x for x in class\_attr\_types.values()])

tmplis=[x for x in class\_attr\_types.keys()]

# print('tmplis :',tmplis)

# print('all data:')

# for i in all\_data:

    # print(i)

rest\_attr\_dict={}

for i in range(-1+len(all\_Attr)):

    tmp={}

    for j in range(len(all\_data)):

        if all\_data[j][i] not in tmp:

            tmp[all\_data[j][i]]=[j]

        else:tmp[all\_data[j][i]].append(j)

    tmpp={}

    for key,val in tmp.items():

        cnt=[0]\*len(tmplis)

        for j in val:

            for k in range(len(tmplis)):

                if all\_data[j][-1]==tmplis[k]:

                    cnt[k]+=1

                    break

        tmpp[key]=cnt

    rest\_attr\_dict[all\_Attr[i]]=tmpp

    # print('Attr:',all\_Attr[i],'and tmpp')

    # print(tmpp)

gain\_attr={}

info\_gain\_attr={}

for i in range(-1+len(all\_Attr)):

    gain=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gain += (sum/len(all\_data))\*calculateEntropy(val)

    gain\_attr[all\_Attr[i]]=gain

    info\_gain\_attr[all\_Attr[i]]=info\_dataset-gain

print('Information gain for attributes:')

for k,v in info\_gain\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in info\_gain\_attr.items()]

liss.sort(reverse=True)

print('Proceeding in sorted order by information gain:')

for i in range(len(liss)):

    print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

gini\_dataset=calculateGini([x for x in class\_attr\_types.values()])

gini\_attr={}

del\_gini\_attr={}

for i in range(-1+len(all\_Attr)):

    gini=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gini += (sum/len(all\_data))\*calculateGini(val)

    gini\_attr[all\_Attr[i]]=gini

    del\_gini\_attr[all\_Attr[i]]=gini\_dataset-gini

print('Gini Index for attributes:')

for k,v in del\_gini\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in del\_gini\_attr.items()]

liss.sort(reverse=True)

print('Proceeding in sorted order by Gini indices:')

for i in range(len(liss)):

    print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

gain\_ratio\_attr={}

for i in range(-1+len(all\_Attr)):

    tmpliss=[]

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        tmpliss.append(sum)

    gain\_ratio\_attr[all\_Attr[i]]=info\_gain\_attr[all\_Attr[i]]/calculateEntropy(tmpliss)

print('Gain ratio for attributes:')

for k,v in gain\_ratio\_attr.items():

    print(k,':','%5f'%v)

liss=[[v,k] for k,v in gain\_ratio\_attr.items()]

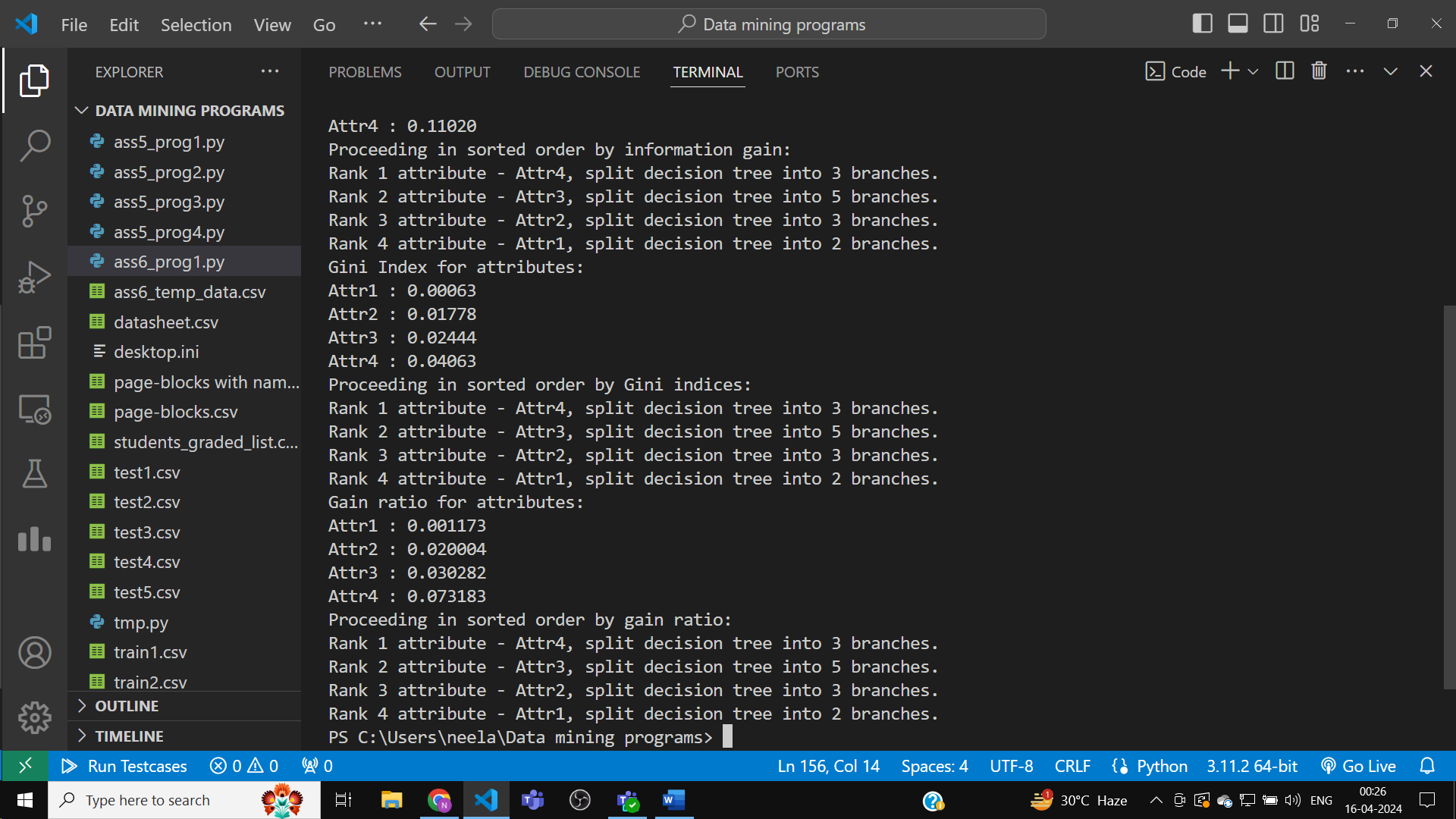
liss.sort(reverse=True)

print('Proceeding in sorted order by gain ratio:')

for i in range(len(liss)):

    print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

Output:



As can be seen, the splitting of tree into several branches depends on the no. of types under the particular attribute on which we are splitting, their split info will decide that.

So, it is not possible to consciously choose the no. of branches to split the decision tree while training it over given data.

Now, let’s train it on the real ‘car data.csv’, downloaded from Kaggle.

The program made above is generic in nature, so it will handle the data’s uniqueness automatically notwithstanding the different string datatypes of the various attributes.

Python Code :

import math

import pandas as pd

import numpy as np

def calculateEntropy(liss):

    sum=0

    for i in liss:sum+=i

    ans=0

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*math.log2(i/sum)

    return ans

def calculateGini(liss):

    sum=0

    for i in liss:sum+=i

    ans=1

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*\*2

    return ans

df=pd.read\_csv('car data.csv')

# print(df)

all\_Attr=[x for x in df.columns[1:]]

# print(all\_Attr)

all\_data=[]

for i in df.itertuples():

    # print(list(i))

    all\_data.append([x for x in i[2:]])

class\_attr\_types={}

for i in all\_data:

    if i[-1] in class\_attr\_types:

        class\_attr\_types[i[-1]]+=1

    else:class\_attr\_types[i[-1]]=1

info\_dataset=calculateEntropy([x for x in class\_attr\_types.values()])

tmplis=[x for x in class\_attr\_types.keys()]

# print('tmplis :',tmplis)

# print('all data:')

# for i in all\_data:

    # print(i)

rest\_attr\_dict={}

for i in range(-1+len(all\_Attr)):

    tmp={}

    for j in range(len(all\_data)):

        if all\_data[j][i] not in tmp:

            tmp[all\_data[j][i]]=[j]

        else:tmp[all\_data[j][i]].append(j)

    tmpp={}

    for key,val in tmp.items():

        cnt=[0]\*len(tmplis)

        for j in val:

            for k in range(len(tmplis)):

                if all\_data[j][-1]==tmplis[k]:

                    cnt[k]+=1

                    break

        tmpp[key]=cnt

    rest\_attr\_dict[all\_Attr[i]]=tmpp

    # print('Attr:',all\_Attr[i],'and tmpp')

    # print(tmpp)

gain\_attr={}

info\_gain\_attr={}

for i in range(-1+len(all\_Attr)):

    gain=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gain += (sum/len(all\_data))\*calculateEntropy(val)

    gain\_attr[all\_Attr[i]]=gain

    info\_gain\_attr[all\_Attr[i]]=info\_dataset-gain

print('Information gain for attributes:')

for k,v in info\_gain\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in info\_gain\_attr.items()]

liss.sort(reverse=True)

print('Proceeding in sorted order by information gain:')

for i in range(len(liss)):

    print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

gini\_dataset=calculateGini([x for x in class\_attr\_types.values()])

gini\_attr={}

del\_gini\_attr={}

for i in range(-1+len(all\_Attr)):

    gini=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gini += (sum/len(all\_data))\*calculateGini(val)

    gini\_attr[all\_Attr[i]]=gini

    del\_gini\_attr[all\_Attr[i]]=gini\_dataset-gini

print('Gini Index for attributes:')

for k,v in del\_gini\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in del\_gini\_attr.items()]

liss.sort(reverse=True)

print('Proceeding in sorted order by Gini indices:')

for i in range(len(liss)):

    print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

gain\_ratio\_attr={}

for i in range(-1+len(all\_Attr)):

    tmpliss=[]

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        tmpliss.append(sum)

    gain\_ratio\_attr[all\_Attr[i]]=info\_gain\_attr[all\_Attr[i]]/calculateEntropy(tmpliss)

print('Gain ratio for attributes:')

for k,v in gain\_ratio\_attr.items():

    print(k,':','%5f'%v)

liss=[[v,k] for k,v in gain\_ratio\_attr.items()]

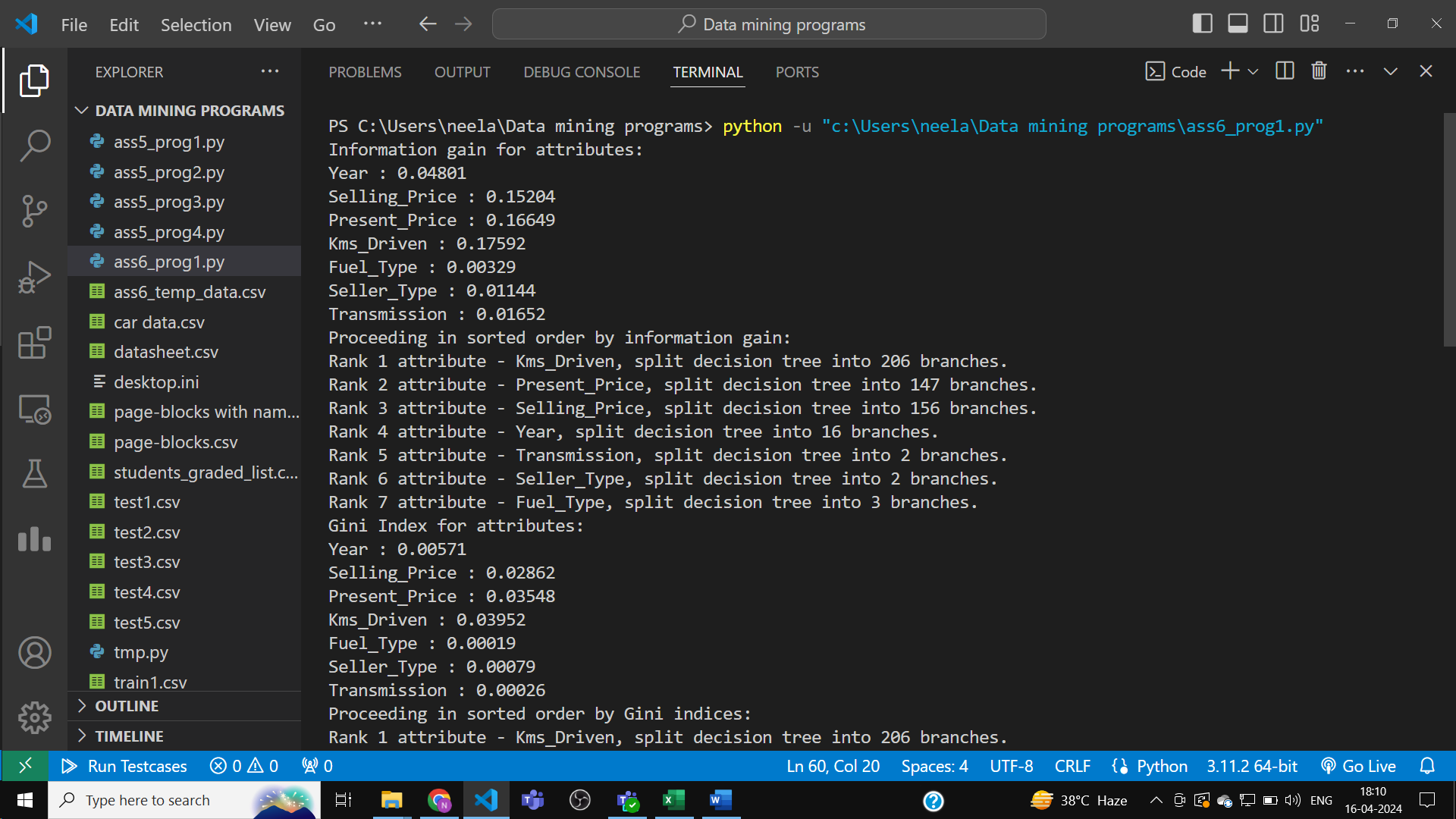
liss.sort(reverse=True)

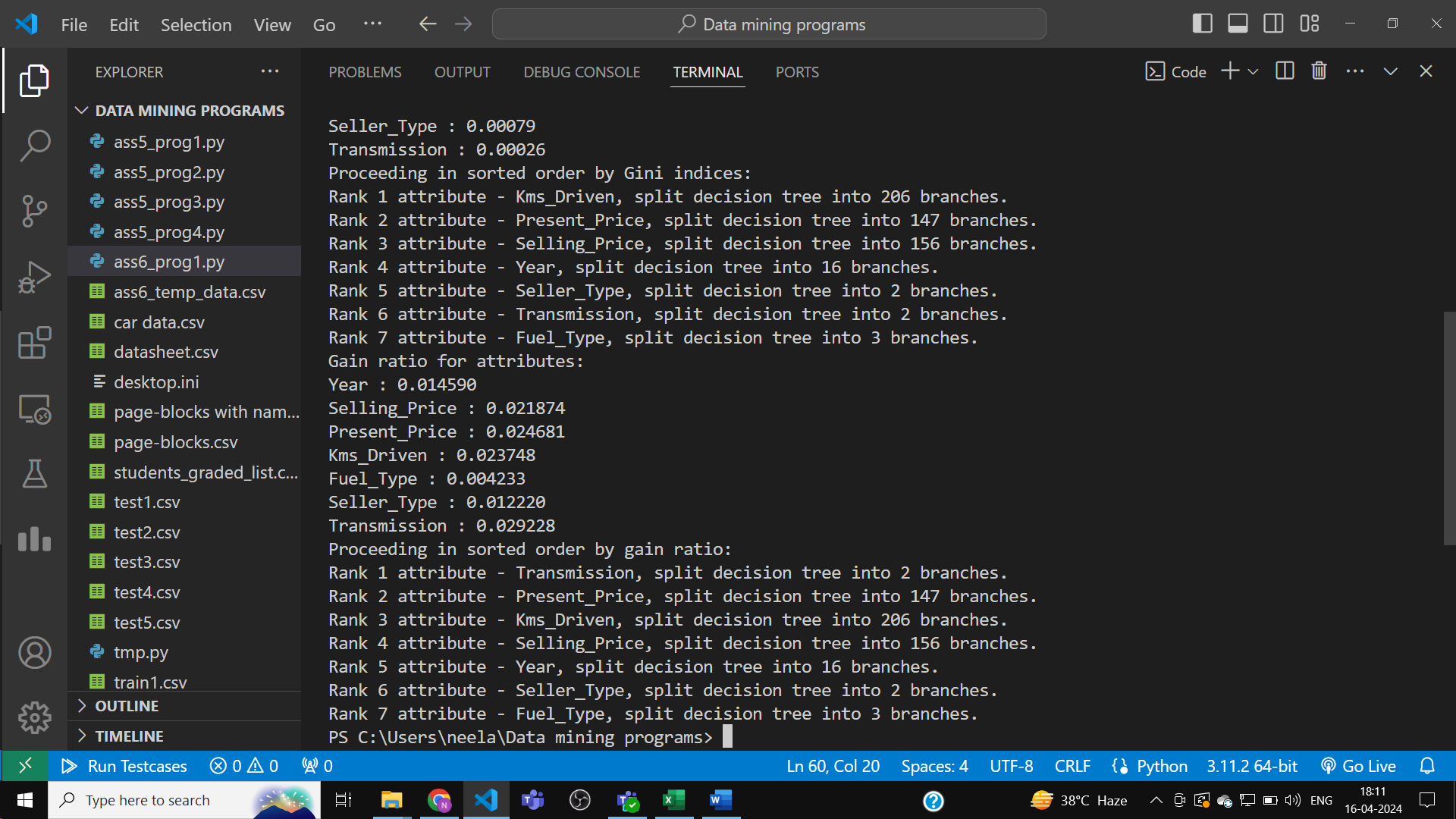
print('Proceeding in sorted order by gain ratio:')

for i in range(len(liss)):

    print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

Output:





Now to implement decision tree for various attribute-splitting quantities :

First, Python code :

import math

import pandas as pd

import numpy as np

def calculateEntropy(liss):

    sum=0

    for i in liss:sum+=i

    ans=0

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*math.log2(i/sum)

    return ans

def calculateGini(liss):

    sum=0

    for i in liss:sum+=i

    ans=1

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*\*2

    return ans

class tree:

    def \_\_init\_\_(self):

        self.whereto={}

        self.decision,self.thisIdx=None,None

def buildtree(tuple: list,idx: int,node: tree,liss: list):

    node.thisIdx=liss[idx]

    if idx==-1+len(liss):

        if tuple[node.thisIdx] not in node.whereto:

            node.whereto[tuple[node.thisIdx]]=1

        else:node.whereto[tuple[node.thisIdx]]+=1

        node.thisIdx=-1

        return

    if tuple[node.thisIdx] not in node.whereto:

        node.whereto[tuple[node.thisIdx]]=tree()

        buildtree(tuple,idx+1,node.whereto[tuple[node.thisIdx]],liss)

    else:buildtree(tuple,idx+1,node.whereto[tuple[node.thisIdx]],liss)

def traverseTree(node: tree):

    if node.thisIdx<0:

        print(node.whereto)

        return

    print(node.whereto)

    for v in node.whereto.values():

        traverseTree(v)

def navigate(tuple: list,node: tree,priority: dict):

    if node.thisIdx<0:

        if node.decision!=None:

            return node.decision

        tmplis=[]

        for k,v in node.whereto.items():

            tmplis.append([v,k])

        tmplis.sort(reverse=True)

        tmpp=[]

        for i in range(len(tmplis)):

            if tmplis[i][0]==tmplis[0][0]:

                tmpp.append(i)

            else:break

        tmpp2=[]

        for item in tmpp:

            tmpp2.append([priority[tmplis[item][1]],tmplis[item][1]])

        tmpp2.sort(reverse=True)

        # print('tmpp2:',tmpp2)

        node.decision=tmpp2[0][1]

        return node.decision

    if tuple[node.thisIdx] not in node.whereto:

        return 'Indeterminate'

    return navigate(tuple,node.whereto[tuple[node.thisIdx]],priority)

'''

names=['Attr1','Attr2','Attr3','Attr4','Class']

data=[

    ['Yes','Low','L','Fair','Yes'],

    ['No','High','M','Excellent','Yes'],

    ['No','Medium','S','Poor','Yes'],

    ['Yes','Medium','XL','Fair','No'],

    ['Yes','High','XXL','Fair','Yes'],

    ['No','High','L','Poor','Yes'],

    ['No','Medium','M','Fair','Yes'],

    ['No','Low','M','Excellent','No'],

    ['Yes','Low','XL','Excellent','Yes'],

    ['No','Medium','XL','Fair','Yes'],

    ['Yes','High','S','Fair','Yes'],

    ['No','Low','S','Fair','No'],

    ['Yes','Medium','M','Excellent','Yes'],

    ['Yes','High','L','Excellent','No'],

    ['No','Low','XXL','Poor','Yes']

]

df=pd.DataFrame(data,columns=names)

print(df)

for i in df.columns:

    print(i) # i stores the column names

for i in df.itertuples():

    print(i)

    print('first =',i[0],'last =',i[-1]) # first is the index, then comes the data

# this can be a very easy way to iteratively navigate the data and create a custom

# dictionary with class labels as was visible through the df.columns

# that is, if the data is given in this format !

# if it's in attribute -> then the data under that column

tempfile=df.to\_csv('ass6\_temp\_data.csv')

'''

df=pd.read\_csv('ass6\_temp\_data.csv')

# df=pd.read\_csv('car data.csv')

print(df)

all\_Attr=[x for x in df.columns[1:]]

# print(all\_Attr)

all\_data=[]

for i in df.itertuples():

    # print(list(i))

    all\_data.append([x for x in i[2:]])

class\_attr\_types={}

for i in all\_data:

    if i[-1] in class\_attr\_types:

        class\_attr\_types[i[-1]]+=1

    else:class\_attr\_types[i[-1]]=1

# print('Class attribute is',all\_Attr[-1])

# print('class attr\_types: ')

# print(class\_attr\_types)

info\_dataset=calculateEntropy([x for x in class\_attr\_types.values()])

tmplis=[x for x in class\_attr\_types.keys()]

# print('tmplis :',tmplis)

# print('all data:')

# for i in all\_data:

    # print(i)

rest\_attr\_dict={}

for i in range(-1+len(all\_Attr)):

    tmp={}

    for j in range(len(all\_data)):

        if all\_data[j][i] not in tmp:

            tmp[all\_data[j][i]]=[j]

        else:tmp[all\_data[j][i]].append(j)

    tmpp={}

    for key,val in tmp.items():

        cnt=[0]\*len(tmplis)

        for j in val:

            for k in range(len(tmplis)):

                if all\_data[j][-1]==tmplis[k]:

                    cnt[k]+=1

                    break

        tmpp[key]=cnt

    rest\_attr\_dict[all\_Attr[i]]=tmpp

    # print('Attr:',all\_Attr[i],'and tmpp')

    # print(tmpp)

class\_attr\_types\_priority={}

class\_type\_lis=[]

for k,v in class\_attr\_types.items():

    class\_type\_lis.append([v,k])

class\_type\_lis.sort(reverse=True)

for i in range(len(class\_type\_lis)):

    class\_attr\_types\_priority[class\_type\_lis[i][1]]=len(class\_type\_lis)-i

rest\_attr\_indices={}

idx=0

for k in rest\_attr\_dict.keys():

    rest\_attr\_indices[k]=idx

    idx+=1

gain\_attr={}

info\_gain\_attr={}

for i in range(-1+len(all\_Attr)):

    gain=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gain += (sum/len(all\_data))\*calculateEntropy(val)

    gain\_attr[all\_Attr[i]]=gain

    info\_gain\_attr[all\_Attr[i]]=info\_dataset-gain

print('Information gain for attributes:')

for k,v in info\_gain\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in info\_gain\_attr.items()]

liss.sort(reverse=True)

# print('Proceeding in sorted order by information gain:')

# for i in range(len(liss)):

#     print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

# print(liss)

tmpp=[rest\_attr\_indices[x[1]] for x in liss]

infogain\_tree=tree()

# print('tmpp:',tmpp)

temp\_data\_test=['Yes','Low','S','Fair','?']

for tuple in all\_data:

    buildtree(tuple,0,infogain\_tree,tmpp)

# traverseTree(infogain\_tree)

print('Random test data:',temp\_data\_test)

print('Answer under class',all\_Attr[-1],'for random test data using information gain decision tree:',navigate(temp\_data\_test,infogain\_tree,class\_attr\_types\_priority))

gini\_dataset=calculateGini([x for x in class\_attr\_types.values()])

gini\_attr={}

del\_gini\_attr={}

for i in range(-1+len(all\_Attr)):

    gini=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gini += (sum/len(all\_data))\*calculateGini(val)

    gini\_attr[all\_Attr[i]]=gini

    del\_gini\_attr[all\_Attr[i]]=gini\_dataset-gini

print('Gini Index for attributes:')

for k,v in del\_gini\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in del\_gini\_attr.items()]

liss.sort(reverse=True)

# print('Proceeding in sorted order by Gini indices:')

# for i in range(len(liss)):

#     print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

tmpp=[rest\_attr\_indices[x[1]] for x in liss]

gini\_tree=tree()

# print('tmpp:',tmpp)

temp\_data\_test=['No','High','XXL','Fair','?']

for tuple in all\_data:

    buildtree(tuple,0,gini\_tree,tmpp)

# traverseTree(infogain\_tree)

print('Random test data:',temp\_data\_test)

print('Answer under class',all\_Attr[-1],'for random test data using Gini Index decision tree:',navigate(temp\_data\_test,gini\_tree,class\_attr\_types\_priority))

gain\_ratio\_attr={}

for i in range(-1+len(all\_Attr)):

    tmpliss=[]

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        tmpliss.append(sum)

    gain\_ratio\_attr[all\_Attr[i]]=info\_gain\_attr[all\_Attr[i]]/calculateEntropy(tmpliss)

print('Gain ratio for attributes:')

for k,v in gain\_ratio\_attr.items():

    print(k,':','%5f'%v)

liss=[[v,k] for k,v in gain\_ratio\_attr.items()]

liss.sort(reverse=True)

# print('Proceeding in sorted order by gain ratio:')

# for i in range(len(liss)):

#     print('Rank '+str(i+1)+' attribute - '+liss[i][1]+', split decision tree into '+str(len(rest\_attr\_dict[liss[i][1]]))+' branches.')

tmpp=[rest\_attr\_indices[x[1]] for x in liss]

gainratio\_tree=tree()

for tuple in all\_data:

    buildtree(tuple,0,gainratio\_tree,tmpp)

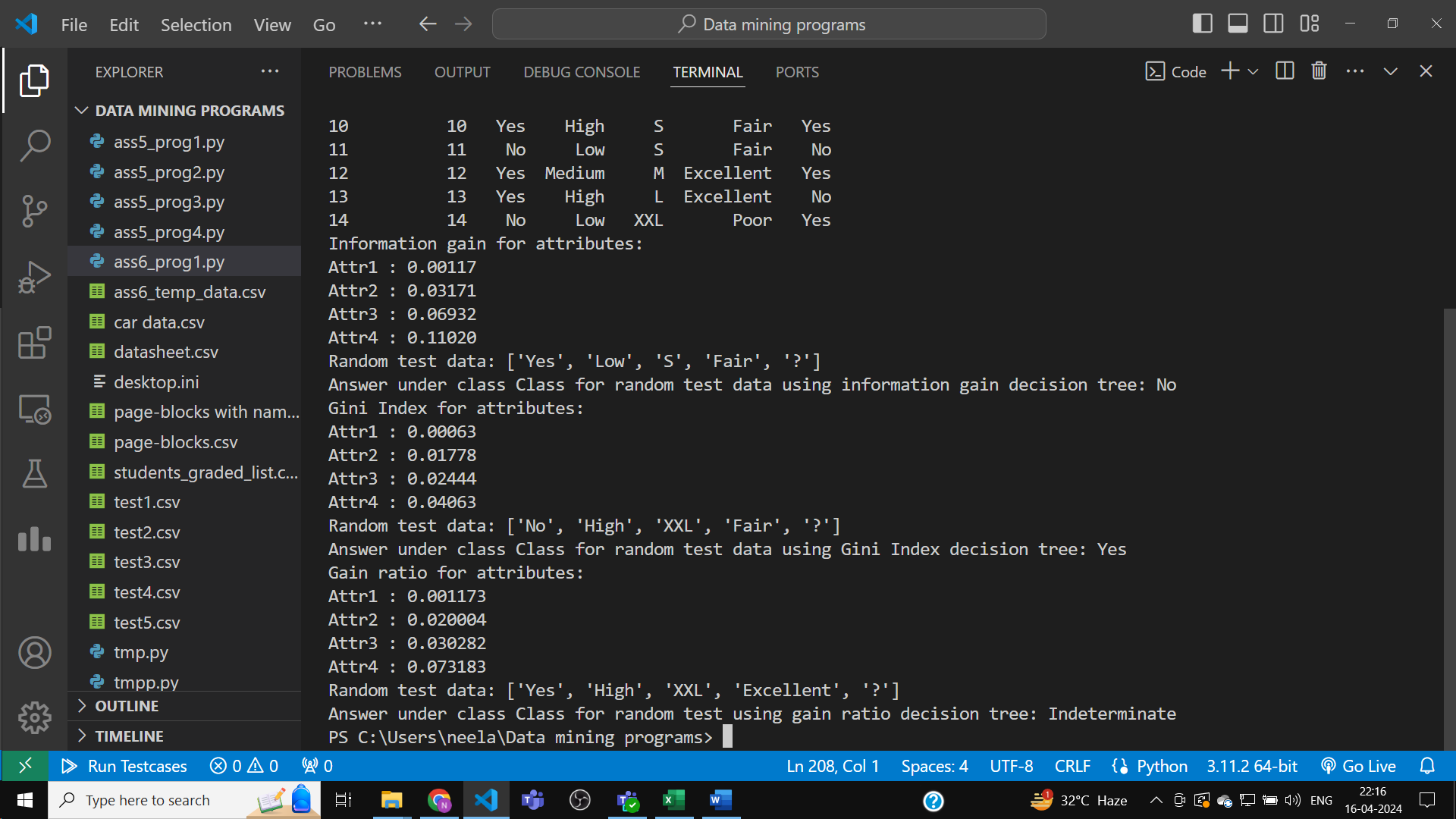
temp\_data\_test=['Yes','High','XXL','Excellent','?']

print('Random test data:',temp\_data\_test)

print('Answer under class',all\_Attr[-1],'for random test using gain ratio decision tree:',navigate(temp\_data\_test,gainratio\_tree,class\_attr\_types\_priority))

This was done for the temporary small dataset made.

Output:



Now, deploying same model for the car data csv file.

Python Code :

import math

import pandas as pd

import numpy as np

def calculateEntropy(liss):

    sum=0

    for i in liss:sum+=i

    ans=0

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*math.log2(i/sum)

    return ans

def calculateGini(liss):

    sum=0

    for i in liss:sum+=i

    ans=1

    for i in liss:

        if i!=0:

            ans-=(i/sum)\*\*2

    return ans

class tree:

    def \_\_init\_\_(self):

        self.whereto={}

        self.decision,self.thisIdx=None,None

def buildtree(tuple: list,idx: int,node: tree,liss: list):

    node.thisIdx=liss[idx]

    if idx==-1+len(liss):

        if tuple[node.thisIdx] not in node.whereto:

            node.whereto[tuple[node.thisIdx]]=1

        else:node.whereto[tuple[node.thisIdx]]+=1

        node.thisIdx=-1

        return

    if tuple[node.thisIdx] not in node.whereto:

        node.whereto[tuple[node.thisIdx]]=tree()

        buildtree(tuple,idx+1,node.whereto[tuple[node.thisIdx]],liss)

    else:buildtree(tuple,idx+1,node.whereto[tuple[node.thisIdx]],liss)

def traverseTree(node: tree):

    if node.thisIdx<0:

        print(node.whereto)

        return

    print(node.whereto)

    for v in node.whereto.values():

        traverseTree(v)

def navigate(tuple: list,node: tree,priority: dict):

    if node.thisIdx<0:

        if node.decision!=None:

            return node.decision

        tmplis=[]

        for k,v in node.whereto.items():

            tmplis.append([v,k])

        tmplis.sort(reverse=True)

        tmpp=[]

        for i in range(len(tmplis)):

            if tmplis[i][0]==tmplis[0][0]:

                tmpp.append(i)

            else:break

        tmpp2=[]

        for item in tmpp:

            tmpp2.append([priority[tmplis[item][1]],tmplis[item][1]])

        tmpp2.sort(reverse=True)

        # print('tmpp2:',tmpp2)

        node.decision=tmpp2[0][1]

        return node.decision

    if tuple[node.thisIdx] not in node.whereto:

        return 'Indeterminate'

    return navigate(tuple,node.whereto[tuple[node.thisIdx]],priority)

# df=pd.read\_csv('ass6\_temp\_data.csv')

df=pd.read\_csv('car data.csv')

# print(df)

all\_Attr=[x for x in df.columns[1:]]

print(all\_Attr)

all\_data=[]

for i in df.itertuples():

    all\_data.append([x for x in i[1:-1+len(i)]])

print(all\_data[2])

df=pd.DataFrame(all\_data,columns=all\_Attr)

df=df.reset\_index(drop=False)

print(df)

class\_attr\_types={}

for i in all\_data:

    if i[-1] in class\_attr\_types:

        class\_attr\_types[i[-1]]+=1

    else:class\_attr\_types[i[-1]]=1

info\_dataset=calculateEntropy([x for x in class\_attr\_types.values()])

tmplis=[x for x in class\_attr\_types.keys()]

rest\_attr\_dict={}

for i in range(-1+len(all\_Attr)):

    tmp={}

    for j in range(len(all\_data)):

        if all\_data[j][i] not in tmp:

            tmp[all\_data[j][i]]=[j]

        else:tmp[all\_data[j][i]].append(j)

    tmpp={}

    for key,val in tmp.items():

        cnt=[0]\*len(tmplis)

        for j in val:

            for k in range(len(tmplis)):

                if all\_data[j][-1]==tmplis[k]:

                    cnt[k]+=1

                    break

        tmpp[key]=cnt

    rest\_attr\_dict[all\_Attr[i]]=tmpp

class\_attr\_types\_priority={}

class\_type\_lis=[]

for k,v in class\_attr\_types.items():

    class\_type\_lis.append([v,k])

class\_type\_lis.sort(reverse=True)

for i in range(len(class\_type\_lis)):

    class\_attr\_types\_priority[class\_type\_lis[i][1]]=len(class\_type\_lis)-i

rest\_attr\_indices={}

idx=0

for k in rest\_attr\_dict.keys():

    rest\_attr\_indices[k]=idx

    idx+=1

gain\_attr={}

info\_gain\_attr={}

for i in range(-1+len(all\_Attr)):

    gain=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gain += (sum/len(all\_data))\*calculateEntropy(val)

    gain\_attr[all\_Attr[i]]=gain

    info\_gain\_attr[all\_Attr[i]]=info\_dataset-gain

print('Information gain for attributes:')

for k,v in info\_gain\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in info\_gain\_attr.items()]

liss.sort(reverse=True)

tmpp=[rest\_attr\_indices[x[1]] for x in liss]

infogain\_tree=tree()

temp\_data\_test=['ritz',2014,3.35,5.59,27000,'Petrol','Dealer','Manual','?']

for tuple in all\_data:

    buildtree(tuple,0,infogain\_tree,tmpp)

print('Random test data:',temp\_data\_test)

print('Answer under class',all\_Attr[-1],'for random test data using information gain decision tree:',navigate(temp\_data\_test,infogain\_tree,class\_attr\_types\_priority))

gini\_dataset=calculateGini([x for x in class\_attr\_types.values()])

gini\_attr={}

del\_gini\_attr={}

for i in range(-1+len(all\_Attr)):

    gini=0

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        gini += (sum/len(all\_data))\*calculateGini(val)

    gini\_attr[all\_Attr[i]]=gini

    del\_gini\_attr[all\_Attr[i]]=gini\_dataset-gini

print('Gini Index for attributes:')

for k,v in del\_gini\_attr.items():

    print(k,':','%.5f'%v)

liss=[[v,k] for k,v in del\_gini\_attr.items()]

liss.sort(reverse=True)

tmpp=[rest\_attr\_indices[x[1]] for x in liss]

gini\_tree=tree()

temp\_data\_test=['sx4',2017,3.35,5.59,69000,'Petrol','Individual','Automatic','?']

for tuple in all\_data:

    buildtree(tuple,0,gini\_tree,tmpp)

print('Random test data:',temp\_data\_test)

print('Answer under class',all\_Attr[-1],'for random test data using Gini Index decision tree:',navigate(temp\_data\_test,gini\_tree,class\_attr\_types\_priority))

gain\_ratio\_attr={}

for i in range(-1+len(all\_Attr)):

    tmpliss=[]

    for key,val in rest\_attr\_dict[all\_Attr[i]].items():

        sum=0

        for item in val:sum+=item

        tmpliss.append(sum)

    gain\_ratio\_attr[all\_Attr[i]]=info\_gain\_attr[all\_Attr[i]]/calculateEntropy(tmpliss)

print('Gain ratio for attributes:')

for k,v in gain\_ratio\_attr.items():

    print(k,':','%5f'%v)

liss=[[v,k] for k,v in gain\_ratio\_attr.items()]

liss.sort(reverse=True)

tmpp=[rest\_attr\_indices[x[1]] for x in liss]

gainratio\_tree=tree()

for tuple in all\_data:

    buildtree(tuple,0,gainratio\_tree,tmpp)

temp\_data\_test=['ciaz',2015,5.35,5.59,43000,'Diesel','Dealer','Manual','?']

print('Random test data:',temp\_data\_test)

print('Answer under class',all\_Attr[-1],'for random test using gain ratio decision tree:',navigate(temp\_data\_test,gainratio\_tree,class\_attr\_types\_priority))