**1. Decision Tree**

**Source Code**

import pandas as pd

import numpy as np

df = pd.DataFrame({'District':['Suburban','Suburban','Rural','Urban','Urban','Urban','Rural','Suburban','Suburban','Urban','Suburban','Rural','Rural','Urban'],

    'House Type':['Detached','Detached','Detached','Semi-detached','Semi-detached','Semi-detached','Semi-detached','Terrace','Semi-detached','Terrace','Terrace','Terrace','Detached','Terrace'],

    'Income':['High','High','High','High','Low','Low','Low','High','Low','Low','Low','High','Low','High'],

    'Previous Customer':['No','Yes','No','No','No','Yes','Yes','No','No','No','Yes','Yes','No','Yes'],

    'Outcome':['Not responded','Not responded','Responded','Responded','Responded','Not responded','Responded','Not responded','Responded','Responded','Responded','Responded','Responded','Not responded']},

    columns=['District','House Type','Income','Previous Customer','Outcome'])

#calculate entropy by column function

def entropy(target\_col):

    elements, counts = np.unique(target\_col, return\_counts = True)

    entropy = -np.sum([(counts[i]/np.sum(counts))\*np.log2(counts[i]/np.sum(counts)) for i in range(len(elements))])

    return entropy

#calculate information gain function

def InfoGain(data,split\_attribute\_name,target\_name):

    #calculate total entropy

    total\_entropy = entropy(data[target\_name])

    #calculate weight entropy

    vals,counts= np.unique(data[split\_attribute\_name],return\_counts=True)

    Weighted\_Entropy = np.sum([(counts[i]/np.sum(counts))\* entropy(data.where(data[split\_attribute\_name]==vals[i]).dropna()[target\_name])

                               for i in range(len(vals))])

    #calculate information gain

    Information\_Gain = total\_entropy - Weighted\_Entropy

    return Information\_Gain

#create decision tree by ID3 algorithm

def create\_tree(data,originaldata,features,target\_attribute\_name,parent\_node\_class = None):

    #if pure

    if len(np.unique(data[target\_attribute\_name])) <= 1:

        return np.unique(data[target\_attribute\_name])[0]

    #if no data

    elif len(data)==0:

        return np.unique(originaldata[target\_attribute\_name])\

               [np.argmax(np.unique(originaldata[target\_attribute\_name], return\_counts=True)[1])]

    #if no features

    elif len(features) ==0:

        return parent\_node\_class

    #tree growth

    else:

        parent\_node\_class = np.unique(data[target\_attribute\_name])\

                            [np.argmax(np.unique(data[target\_attribute\_name], return\_counts=True)[1])]

        item\_values = [InfoGain(data,feature,target\_attribute\_name) for feature in features]

        best\_feature\_index = np.argmax(item\_values)

        best\_feature = features[best\_feature\_index]

        tree = {best\_feature:{}}

        features = [i for i in features if i != best\_feature]

        for value in np.unique(data[best\_feature]):

            sub\_data = data.where(data[best\_feature] == value).dropna()

            subtree = create\_tree(sub\_data,data,features,target\_attribute\_name,parent\_node\_class)

            tree[best\_feature][value] = subtree

        return(tree)

print("\n\nDataset")

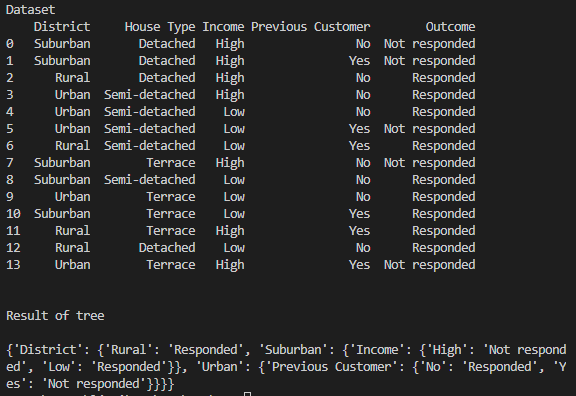
print(df)

tree = create\_tree(df,df,['District','House Type','Income','Previous Customer'],'Outcome')

print("\n\nResult of tree\n")

print(tree)

**Output Screen Capture**



**2. KNN**

**Source Code**

import pandas as pd

import numpy as np

from math import sqrt

df = pd.DataFrame({'HEIGHT(cm)':[158,158,158,160,160,163,163,160,163,165,165,165,168,168,168,170,170,170],

        'WEIGHT(kg)':[58,59,63,59,60,60,61,64,64,61,62,65,62,63,66,63,64,68],

        'T SHIRT SIZE':['M','M','M','M','M','M','M','L','L','L','L','L','L','L','L','L','L','L']},

        columns=['HEIGHT(cm)','WEIGHT(kg)','T SHIRT SIZE'])

df2 = df.copy()

df2 = df2[["HEIGHT(cm)", "WEIGHT(kg)"]]

df2.columns=["HEIGHT(cm)", "WEIGHT(kg)"]

# Set test data and k

str = input("Enter height(cm) and weight(kg) of a sample: ")

sample = str.split()

for i in range(len(sample)):

    sample[i] = int(sample[i])

test\_height = sample[0]

test\_weight = sample[1]

k = int(input("Enter an odd number you want to set 'K': "))

# Append sample(test data) to original dataset

df2.loc[len(df2)] = [sample[0], sample[1]]

df2["T SHIRT SIZE"] = df["T SHIRT SIZE"]

#make list

df2\_list = df2.values.tolist()

# Calculate distance between two points

def distance(row1, row2):     # row=[height, weight]

    distance=0.0

    for i in range(len(row1)-1):

        distance += (float(row1[i]) - float(row2[i]))\*\*2

    return sqrt(distance)

# Sorting neighbors using distance

def search\_neighbors(data, sample, k):

    distances = list()

    for each in data:

        dist = distance(sample, each)

        if(dist==0):

            continue

        distances.append((each,dist))

    distances.sort(key=lambda tup:tup[1])

    neighbors = list()

    for i in range(k):

        neighbors.append(distances[i][0])

    return neighbors

# Search the nearest neighbors

sample = df2\_list[len(df2\_list)-1]

neighbors = search\_neighbors(df2\_list, sample, k)

# Append distance to normalized dataset

distances = list()

for each in df2\_list:

    dist = distance(sample, each)

    distances.append(dist)

df2["DISTANCE"] = distances

# Print dataset and neighbors

print("\nDataset with distance")

print(df2)

print("19 row is a test data.")

print("\nNeighbor that selected")

from pprint import pprint

pprint(neighbors)

# Predict target value with neighbors

def predict(data, sample, k):

    neighbors = search\_neighbors(data, sample, k)

    selected\_size = [row[-1] for row in neighbors]

    # Store value in prediction

    prediction = max(set(selected\_size), key=selected\_size.count)

    return prediction

# Store predicted value in prediction

prediction = predict(df2\_list, sample, k)

# Print the result

print("\n\n")

print("T-shirt size of (", test\_height, "cm,", test\_weight, "kg ) is expected to size", prediction)

**Output Screen Capture**

