Program - 01

Implement A* Search Algorithm

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}               #store distance from starting node
    parents = {}         # parents contains an adjacency map of
all nodes
    #distance of starting node from itself is zero
    g[start_node] = 0
    #start_node is root node i.e it has no parent nodes
    #so start_node is set to its own parent node
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        #node with lowest f() is found
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] +
heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                #nodes 'm' not in first and last set are added to
first
                #n is set its parent
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                    #from start through n node
                else:
                    if g[m] > g[n] + weight:
                        #update g(m)
                        g[m] = g[n] + weight
                        #change parent of m to n
                        parents[m] = n
                        #if m in closed set,remove and add to
open
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print('Path does not exist!')
```

```
            return None

        # if the current node is the stop node
        # then we begin reconstructing the path from it to the
start_node
        if n == stop_node:
            path = []
            while parents[n] != n:
                path.append(n)
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Path found: {}'.format(path))
            return path
        # remove n from the open_list, and add it to closed_list
        # because all of his neighbors were inspected
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

#define fuction to return neighbor and its distance
#from the passed node
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

#A star example 1
#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 5,
        'D': 7,
        'E': 3,
        'F': 6,
        'G': 5,
        'H': 3,
        'I': 1,
        'J': 0
    }
```

```
    return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('A', 6), ('C', 3), ('D', 2)],
    'C': [('B', 3), ('D', 1), ('E', 5)],
    'D': [('B', 2), ('C', 1), ('E', 8)],
    'E': [('C', 5), ('D', 8), ('I', 5), ('J', 5)],
    'F': [('A', 3), ('G', 1), ('H', 7)],
    'G': [('F', 1), ('I', 3)],
    'H': [('F', 7), ('I', 2)],
    'I': [('E', 5), ('G', 3), ('H', 2), ('J', 3)],
}

aStarAlgo('A', 'J')

#for simplicity we ll consider heuristic distances given
#and this function returns heuristic distance for all nodes
def heuristic(n):
    H_dist = {
        'A': 11,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0,
    }
    return H_dist[n]

#Describe your graph here
Graph_nodes = {
    'A': [('B', 2), ('E', 3)],
    'B': [('A', 2), ('C', 1), ('G', 9)],
    'C': [('B', 1)],
    'D': [('E', 6), ('G', 1)],
    'E': [('A', 3), ('D', 6)],
    'G': [('B', 9), ('D', 1)]
}

aStarAlgo('A', 'G')
```

Program – 2

Implement AO* Search Algorithm

```
class Graph:
    def __init__(self, graph, heuristicNodeList, startNode):
#instantiate graph object with graph topology, heuristic values,
start node
        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

    def applyAOStar(self): # starts a recursive AO* algorithm
        self.aoStar(self.start, False)

    def getNeighbors(self, v): # gets the Neighbours of a given
node
        return self.graph.get(v,'')

    def getStatus(self,v): # return the status of a given node
        return self.status.get(v,0)

    def setStatus(self,v, val): # set the status of a given node
        self.status[v]=val

    def getHeuristicNodeValue(self, n):
        return self.H.get(n,0) # always return the heuristic
value of a given node

    def setHeuristicNodeValue(self, n, value):
        self.H[n]=value # set the revised heuristic value of a
given node

    def printSolution(self):
        print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE
START NODE:",self.start)
        print("-----------------------------------------------------
----")
        print(self.solutionGraph)
        print("-----------------------------------------------------
----")

    def computeMinimumCostChildNodes(self, v): # Computes the
Minimum Cost of child nodes of a given node v
```

```
        minimumCost=0
        costToChildNodeListDict={}
        costToChildNodeListDict[minimumCost]=[]
        flag=True
        for nodeInfoTupleList in self.getNeighbors(v): # iterate
over all the set of child node/s
            cost=0
            nodeList=[]
            for c, weight in nodeInfoTupleList:
                cost=cost+self.getHeuristicNodeValue(c)+weight
                nodeList.append(c)
            if flag==True: # initialize Minimum Cost with the
cost of first set of child node/s
                minimumCost=cost
                costToChildNodeListDict[minimumCost]=nodeList #
set the Minimum Cost child node/s
                flag=False
            else: # checking the Minimum Cost nodes with the
current Minimum Cost
                if minimumCost>cost:
                    minimumCost=cost
                    costToChildNodeListDict[minimumCost]=nodeList
# set the Minimum Cost child node/s
        return minimumCost, costToChildNodeListDict[minimumCost]
# return Minimum Cost and Minimum Cost child node/s

    def aoStar(self, v, backTracking): # AO* algorithm for a
start node and backTracking status flag
        print("HEURISTIC VALUES :", self.H)
        print("SOLUTION GRAPH :", self.solutionGraph)
        print("PROCESSING NODE :", v)
        print("-----------------------------------------------
---------------------------------------")
        if self.getStatus(v) >= 0: # if status node v >= 0,
compute Minimum Cost nodes of v
            minimumCost, childNodeList =
self.computeMinimumCostChildNodes(v)
            print(minimumCost, childNodeList)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))
            solved=True # check the Minimum Cost nodes of v are
solved
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
```

```
                    solved=solved & False
            if solved==True: # if the Minimum Cost nodes of v are
solved, set the current node status as solved(-1)
                self.setStatus(v,-1)
                self.solutionGraph[v]=childNodeList # update the
solution graph with the solved nodes which may be a part of
solution
            if v!=self.start: # check the current node is the
start node for backtracking the current node value
                self.aoStar(self.parent[v], True) # backtracking
the current node value with backtracking status set to true
            if backTracking==False: # check the current call is
not for backtracking
                for childNode in childNodeList: # for each
Minimum Cost child node
                    self.setStatus(childNode,0) # set the status
of child node to 0(needs exploration)
                    self.aoStar(childNode, False) # Minimum Cost
child node is further explored with backtracking status as false

#for simplicity we ll consider heuristic distances given
print ("Graph - 1")
h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5,
'H': 7, 'I': 7, 'J': 1}
graph1 = {
    'A': [[('B', 1), ('C', 1)], [('D', 1)]],
    'B': [[('G', 1)], [('H', 1)]],
    'C': [[('J', 1)]],
    'D': [[('E', 1), ('F', 1)]],
    'G': [[('I', 1)]]
}

G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()

print ("Graph - 2")
h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5,
'H': 7} # Heuristic values of Nodes
graph2 = { # Graph of Nodes and Edges
    'A': [[('B', 1), ('C', 1)], [('D', 1)]], # Neighbors of Node
'A', B, C & D with repective weights
    'B': [[('G', 1)], [('H', 1)]], # Neighbors are included in a
list of lists
```

```
    'D': [[('E', 1), ('F', 1)]] # Each sublist indicate a "OR" node or "AND" nodes
}

G2 = Graph(graph2, h2, 'A') # Instantiate Graph object with graph, heuristic values and start Node
G2.applyAOStar() # Run the AO* algorithm
G2.printSolution() # Print the solution graph as output of the AO* algorithm search
```

Program - 03

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

```
import numpy as np
import pandas as pd
data=pd.DataFrame(data=pd.read_csv('EnjoySport.csv'))
concepts=np.array(data.iloc[:,0:-1])
target=np.array(data.iloc[:,-1])
def learn(concepts,target):
    specific_h=concepts[0].copy()
    general_h=[["?" for i in range(len(specific_h))] for i in 
range(len(specific_h))]
    for i,h in enumerate(concepts):
        if target[i]=="yes":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    specific_h[x]='?'
                    general_h[x][x]='?'
        if target[i]=="no":
            for x in range(len(specific_h)):
                if h[x]!=specific_h[x]:
                    general_h[x][x]=specific_h[x]
                else:general_h[x][x]='?'
    indices=[i for i,val in enumerate(general_h) if 
val==['?','?','?','?','?','?']]
    print(indices)
    for i in indices:
        general_h.remove(['?','?','?','?','?','?'])
        return specific_h,general_h
s_final,g_final=learn(concepts,target)
print("Final S:",s_final,sep="\n")
print("Final G:",g_final,sep="\n")
data.head()
```

EnjoySport.csv

```
sky,airtemp,humidity,wind,water,forecast,enjoysport
sunny,warm,normal,strong,warm,same,yes
sunny,warm,high,strong,warm,same,yes
rainy,cold,high,strong,warm,change,no
sunny,warm,high,strong,cool,change,yes
```

Program - 04

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

```
import pandas as pd
df_tennis=pd.DataFrame(data=pd.read_csv('PlayTennisTrain.csv'))
print(df_tennis)

def entropy(probs):
    import math
    return sum([-prob*math.log(prob,2)for prob in probs])

def entropy_of_list(a_list):
    from collections import Counter
    cnt=Counter(x for x in a_list)
    print("No and Yes class:",a_list.name,cnt)
    num_instances=len(a_list)*1.0
    probs=[x/num_instances for x in cnt.values()]
    return entropy(probs)

print(df_tennis['playtennis'])
total_entropy=entropy_of_list(df_tennis['playtennis'])
print("entropy of given playtennis dataset:",total_entropy)

def
information_gain(df,split_attribute_name,target_attribute_name,tr
ace=0):
    print("info gain calculation of",split_attribute_name)
    df_split=df.groupby(split_attribute_name)
    for name,group in df_split:
        print(name)
        print(group)
    nobs=len(df.index)*1.0
    df_agg1=df_split.agg({target_attribute_name:lambda
x:entropy_of_list(x)})
    df_agg2=df_split.agg({target_attribute_name:lambda
x:len(x)/nobs})
    df_agg1.columns=['entropy']
    df_agg2.columns=['proportion']
    new_entropy=sum(df_agg1['entropy']*df_agg2['proportion'])
    old_entropy=entropy_of_list(df[target_attribute_name])
    return old_entropy-new_entropy

print("info gain for outlook is :"+
str(information_gain(df_tennis,'outlook','playtennis')),"\n")
print("info gain for humidity is :"+
str(information_gain(df_tennis,'humidity','playtennis')),"\n")
print("info gain for temperature is :"+
str(information_gain(df_tennis,'temperature','playtennis')),"\n")
```

```
def
id3(df,target_attribute_name,attribute_names,default_class=None):
    from collections import Counter
    cnt=Counter(x for x in df[target_attribute_name])
    if len(cnt)==1:
        return next(iter(cnt))
    elif df.empty or (not attribute_names):
        return default_class
    else:
        default_class=max(cnt.keys())
        gainz=[information_gain(df,attr,target_attribute_name)
for attr in attribute_names]
        index_of_max=gainz.index(max(gainz))
        best_attr=attribute_names[index_of_max]
        tree={best_attr:{ }}
        remaining_attribute_names=[i for i in attribute_names if
i!=best_attr]
        for attr_val,data_subset in df.groupby(best_attr):
            subtree=id3(data_subset,target_attribute_name,remaini
ng_attribute_names,default_class)
            tree[best_attr][attr_val]=subtree
        return tree

attribute_names=list(df_tennis.columns)
print("list of attributes:",attribute_names)
attribute_names.remove('playtennis')
print("predicting attributes:",attribute_names)
from pprint import pprint
tree=id3(df_tennis,"playtennis",attribute_names)
pprint("\n\n the result decison tree is:\n")
pprint(tree)

def classify(instance,tree,default=None):
    attribute=next(iter(tree))
    if instance[attribute] in tree[attribute].keys():
        result=tree[attribute][instance[attribute]]
        if isinstance(result,dict):
            return classify(instance,result)
        else:
            return result
    else:
        return default

df_new=pd.read_csv('PlayTennisTest.csv')
df_new['predicted']=df_new.apply(classify,axis=1,args=(tree,'?'))
print(df_new)
```

PlayTennisTrain.csv

```
outlook,temperature,humidity,wind,playtennis
sunny,hot,high,weak,no
```

```
sunny,hot,high,strong,no
overcast,hot,high,weak,yes
rain,mild,high,weak,yes
rain,cool,normal,weak,yes
rain,cool,normal,strong,no
overcast,cool,normal,strong,yes
sunny,mild,high,weak,no
sunny,cool,normal,weak,yes
rain,mild,normal,weak,yes
```

PlayTennisTest.csv

```
outlook,temperature,humidity,wind
sunny,hot,high,weak
sunny,hot,high,strong
overcast,hot,high,weak
rain,mild,high,weak
rain,cool,normal,weak
rain,cool,normal,strong
overcast,cool,normal,strong
sunny,mild,high,weak
sunny,cool,normal,weak
rain,mild,normal,weak
```

Program – 05

Build an Artificial Neural Network by implementing the Back-propagation algorithm and test the same using appropriate data sets.

```
import numpy as np
#X = (Hours sleeping, Hours Studying), y = (test score of the
student)
X = np.array(([2,9],[1,5],[3,6]),dtype=float)
y = np.array(([92],[86],[89]),dtype=float)
#scale units
X = X/np.amax(X,axis=0) #maximum of X array
y = y/100 #maximum test score is 100

class NeuralNetwork(object):
    def __init__(self):
        #parameters
        self.inputsize=2
        self.outputsize=1
        self.hiddensize=3
        self.w1 = np.random.rand(self.inputsize, self.hiddensize)
#(3x2) weight
        self.w2 = np.random.rand(self.hiddensize,
self.outputsize) #(3x1) weight

    def feedforward(self, X):
        #forward propagation through the network
        self.z = np.dot(X,self.w1) # dot product of X(input) &
fil
        self.z2 = self.sigmoid(self.z) #activation function
        self.z3 = np.dot(self.z2,self.w2) #dot product of self.z2
        output = self.sigmoid(self.z3)
        return output

    def sigmoid(self,s,deriv=False):
        if(deriv == True):
            return s * (1-s)
        return 1/(1+np.exp(-s))

    def backward(self,X,y,output):
        #backward propagate through the network
        self.output_error = y - output
        self.output_delta = self.output_error *
self.sigmoid(output, deriv=True)
        self.z2_error = self.output_delta.dot(self.w2.T)
        self.z2_delta = self.z2_error *
self.sigmoid(self.z2,deriv=True)
        self.w1 += X.T.dot(self.z2_delta)
        self.w2 += self.z2.T.dot(self.output_delta)

    def train(self,X,y):
        output = self.feedforward(X)
```

```
        self.backward(X,y,output)

NN = NeuralNetwork()
for i in range(50000):
    if(i % 100 == 0):
        print("Loss: "+ str(np.mean(np.square(y - 
NN.feedforward(X)))))
    NN.train(X,y)

print("Input: "+str(X))
print("Actual output: " + str(y))
print("Predicted output: " + str(NN.feedforward(X)))
print("Loss:"+ str(np.mean(np.square(y - NN.feedforward(X)))))
```

Program - 06

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

```
import pandas as pd
import numpy as np

data = pd.DataFrame(data=pd.read_csv('Human.csv'))
person = pd.DataFrame(data=pd.read_csv('Person.csv'))

n_male = data['Gender'][data['Gender'] == 'male'].count()
n_female = data['Gender'][data['Gender'] == 'female'].count()
total_ppl = data['Gender'].count()

P_male = n_male/total_ppl
P_female = n_female/total_ppl

data_means = data.groupby('Gender').mean()
data_variance = data.groupby('Gender').var()

male_height_mean = data_means['Height'][data_variance.index ==
'male'].values[0]
male_weight_mean = data_means['Weight'][data_variance.index ==
'male'].values[0]
male_footsize_mean = data_means['Foot_Size'][data_variance.index
== 'male'].values[0]

male_height_variance =
data_variance['Height'][data_variance.index == 'male'].values[0]
male_weight_variance =
data_variance['Weight'][data_variance.index == 'male'].values[0]
male_footsize_variance =
data_variance['Foot_Size'][data_variance.index ==
'male'].values[0]

female_height_mean = data_means['Height'][data_variance.index ==
'female'].values[0]
female_weight_mean = data_means['Weight'][data_variance.index ==
'female'].values[0]
female_footsize_mean =
data_means['Foot_Size'][data_variance.index ==
'female'].values[0]

female_height_variance =
data_variance['Height'][data_variance.index ==
'female'].values[0]
female_weight_variance =
data_variance['Weight'][data_variance.index ==
'female'].values[0]
```

```
female_footsize_variance =
data_variance['Foot_Size'][data_variance.index ==
'female'].values[0]

def p_x_given_y(x, mean_y, variance_y):
    # Input the arguments into a probability density function
    p = 1/(np.sqrt(2*np.pi*variance_y)) * np.exp((-(x-
mean_y)**2)/(2*variance_y))
    return p

PMale = P_male * p_x_given_y(person['Height'][0],
male_height_mean, male_height_variance) *
p_x_given_y(person['Weight'][0], male_weight_mean,
male_weight_variance) * p_x_given_y(person['Foot_Size'][0],
male_footsize_mean, male_footsize_variance)

PFemale = P_female * p_x_given_y(person['Height'][0],
female_height_mean, female_height_variance) *
p_x_given_y(person['Weight'][0], female_weight_mean,
female_weight_variance) * p_x_given_y(person['Foot_Size'][0],
female_footsize_mean, female_footsize_variance)

if(PMale > PFemale):
    print("The given data belongs to Male with Probability of
",PMale)
else:
    print("The given data belongs to Female with Probability of
",PFemale)
```

Human.csv

```
Gender,Height,Weight,Foot_Size
male,6,180,12
male,5.92,190,11
male,5.58,170,12
male,5.92,165,10
female,5,100,6
female,5.5,150,8
female,5.42,130,7
female,5.75,150,9
```

Person.csv

```
Height,Weight,Foot_Size
6,130,8
```

Program - 07

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.

```python
# KMeans Algorithm:
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score

iris = datasets.load_iris()
X = pd.DataFrame(iris.data)
X.columns =
['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y = pd.DataFrame(iris.target)
y.columns = ['Targets']
kmeans = KMeans(n_clusters = 3)
clusters = kmeans.fit_predict(X)

from scipy.stats import mode
labels = np.zeros_like(clusters)
for i in range(3):
    cat = (clusters == i)
    labels[cat] = mode(iris.target[cat])[0]
acc = accuracy_score(iris.target,labels)
print('Accuracy = ',acc)
plt.figure(figsize = (10,10))
colormap = np.array(['red','lime','blue'])
plt.subplot(2,2,1)
plt.scatter(X.Petal_Length,X.Petal_Width,c =
colormap[y.Targets],s = 40)
plt.title('Real Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.subplot(2,2,2)
plt.scatter(X.Petal_Length,X.Petal_Width,c = colormap[labels],s =
40)
plt.title('KMeans Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
```

```
# EM Algorithm :
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
scaled_X = scaler.transform(X)
xs = pd.DataFrame(scaled_X,columns = X.columns)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components = 3)
gmm_y = gmm.fit_predict(xs)
labels = np.zeros_like(clusters)
for i in range(3):
    cat = (gmm_y == i)
    labels[cat] = mode(iris.target[cat])[0]
acc = accuracy_score(iris.target,labels)
print("Accuracy using GMM = ",acc)
plt.subplot(2,2,3)
plt.scatter(X.Petal_Length,X.Petal_Width,c = colormap[gmm_y],s =
40)
plt.subplots_adjust(hspace = 0.4,wspace = 0.4)
plt.title('GMM Clusters')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

Iris.csv

```
Id,SepalLengthCm,SepalWidthCm,Pe
talLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
8,5.0,3.4,1.5,0.2,Iris-setosa
9,4.4,2.9,1.4,0.2,Iris-setosa
10,4.9,3.1,1.5,0.1,Iris-setosa
11,5.4,3.7,1.5,0.2,Iris-setosa
12,4.8,3.4,1.6,0.2,Iris-setosa
13,4.8,3.0,1.4,0.1,Iris-setosa
14,4.3,3.0,1.1,0.1,Iris-setosa
15,5.8,4.0,1.2,0.2,Iris-setosa
16,5.7,4.4,1.5,0.4,Iris-setosa
17,5.4,3.9,1.3,0.4,Iris-setosa
18,5.1,3.5,1.4,0.3,Iris-setosa
19,5.7,3.8,1.7,0.3,Iris-setosa
20,5.1,3.8,1.5,0.3,Iris-setosa
21,5.4,3.4,1.7,0.2,Iris-setosa
22,5.1,3.7,1.5,0.4,Iris-setosa
23,4.6,3.6,1.0,0.2,Iris-setosa
24,5.1,3.3,1.7,0.5,Iris-setosa
25,4.8,3.4,1.9,0.2,Iris-setosa
26,5.0,3.0,1.6,0.2,Iris-setosa
27,5.0,3.4,1.6,0.4,Iris-setosa
28,5.2,3.5,1.5,0.2,Iris-setosa
29,5.2,3.4,1.4,0.2,Iris-setosa
30,4.7,3.2,1.6,0.2,Iris-setosa
31,4.8,3.1,1.6,0.2,Iris-setosa
32,5.4,3.4,1.5,0.4,Iris-setosa
33,5.2,4.1,1.5,0.1,Iris-setosa
34,5.5,4.2,1.4,0.2,Iris-setosa
35,4.9,3.1,1.5,0.1,Iris-setosa
36,5.0,3.2,1.2,0.2,Iris-setosa
```

```
37,5.5,3.5,1.3,0.2,Iris-setosa
38,4.9,3.1,1.5,0.1,Iris-setosa
39,4.4,3.0,1.3,0.2,Iris-setosa
40,5.1,3.4,1.5,0.2,Iris-setosa
41,5.0,3.5,1.3,0.3,Iris-setosa
42,4.5,2.3,1.3,0.3,Iris-setosa
43,4.4,3.2,1.3,0.2,Iris-setosa
44,5.0,3.5,1.6,0.6,Iris-setosa
45,5.1,3.8,1.9,0.4,Iris-setosa
46,4.8,3.0,1.4,0.3,Iris-setosa
47,5.1,3.8,1.6,0.2,Iris-setosa
48,4.6,3.2,1.4,0.2,Iris-setosa
49,5.3,3.7,1.5,0.2,Iris-setosa
50,5.0,3.3,1.4,0.2,Iris-setosa
51,7.0,3.2,4.7,1.4,Iris-
versicolor
52,6.4,3.2,4.5,1.5,Iris-
versicolor
53,6.9,3.1,4.9,1.5,Iris-
versicolor
54,5.5,2.3,4.0,1.3,Iris-
versicolor
55,6.5,2.8,4.6,1.5,Iris-
versicolor
56,5.7,2.8,4.5,1.3,Iris-
versicolor
57,6.3,3.3,4.7,1.6,Iris-
versicolor
58,4.9,2.4,3.3,1.0,Iris-
versicolor
59,6.6,2.9,4.6,1.3,Iris-
versicolor
60,5.2,2.7,3.9,1.4,Iris-
versicolor
61,5.0,2.0,3.5,1.0,Iris-
versicolor
62,5.9,3.0,4.2,1.5,Iris-
versicolor
63,6.0,2.2,4.0,1.0,Iris-
versicolor
64,6.1,2.9,4.7,1.4,Iris-
versicolor
65,5.6,2.9,3.6,1.3,Iris-
versicolor
66,6.7,3.1,4.4,1.4,Iris-
versicolor
67,5.6,3.0,4.5,1.5,Iris-
versicolor
68,5.8,2.7,4.1,1.0,Iris-
versicolor
69,6.2,2.2,4.5,1.5,Iris-
versicolor
70,5.6,2.5,3.9,1.1,Iris-
versicolor
71,5.9,3.2,4.8,1.8,Iris-
versicolor
72,6.1,2.8,4.0,1.3,Iris-
versicolor
73,6.3,2.5,4.9,1.5,Iris-
versicolor
74,6.1,2.8,4.7,1.2,Iris-
versicolor
75,6.4,2.9,4.3,1.3,Iris-
versicolor
76,6.6,3.0,4.4,1.4,Iris-
versicolor
77,6.8,2.8,4.8,1.4,Iris-
versicolor
78,6.7,3.0,5.0,1.7,Iris-
versicolor
79,6.0,2.9,4.5,1.5,Iris-
versicolor
80,5.7,2.6,3.5,1.0,Iris-
versicolor
81,5.5,2.4,3.8,1.1,Iris-
versicolor
82,5.5,2.4,3.7,1.0,Iris-
versicolor
83,5.8,2.7,3.9,1.2,Iris-
versicolor
84,6.0,2.7,5.1,1.6,Iris-
versicolor
85,5.4,3.0,4.5,1.5,Iris-
versicolor
86,6.0,3.4,4.5,1.6,Iris-
versicolor
87,6.7,3.1,4.7,1.5,Iris-
versicolor
```

```
88,6.3,2.3,4.4,1.3,Iris-
versicolor
89,5.6,3.0,4.1,1.3,Iris-
versicolor
90,5.5,2.5,4.0,1.3,Iris-
versicolor
91,5.5,2.6,4.4,1.2,Iris-
versicolor
92,6.1,3.0,4.6,1.4,Iris-
versicolor
93,5.8,2.6,4.0,1.2,Iris-
versicolor
94,5.0,2.3,3.3,1.0,Iris-
versicolor
95,5.6,2.7,4.2,1.3,Iris-
versicolor
96,5.7,3.0,4.2,1.2,Iris-
versicolor
97,5.7,2.9,4.2,1.3,Iris-
versicolor
98,6.2,2.9,4.3,1.3,Iris-
versicolor
99,5.1,2.5,3.0,1.1,Iris-
versicolor
100,5.7,2.8,4.1,1.3,Iris-
versicolor
101,6.3,3.3,6.0,2.5,Iris-
virginica
102,5.8,2.7,5.1,1.9,Iris-
virginica
103,7.1,3.0,5.9,2.1,Iris-
virginica
104,6.3,2.9,5.6,1.8,Iris-
virginica
105,6.5,3.0,5.8,2.2,Iris-
virginica
106,7.6,3.0,6.6,2.1,Iris-
virginica
107,4.9,2.5,4.5,1.7,Iris-
virginica
108,7.3,2.9,6.3,1.8,Iris-
virginica
109,6.7,2.5,5.8,1.8,Iris-
virginica
110,7.2,3.6,6.1,2.5,Iris-
virginica
111,6.5,3.2,5.1,2.0,Iris-
virginica
112,6.4,2.7,5.3,1.9,Iris-
virginica
113,6.8,3.0,5.5,2.1,Iris-
virginica
114,5.7,2.5,5.0,2.0,Iris-
virginica
115,5.8,2.8,5.1,2.4,Iris-
virginica
116,6.4,3.2,5.3,2.3,Iris-
virginica
117,6.5,3.0,5.5,1.8,Iris-
virginica
118,7.7,3.8,6.7,2.2,Iris-
virginica
119,7.7,2.6,6.9,2.3,Iris-
virginica
120,6.0,2.2,5.0,1.5,Iris-
virginica
121,6.9,3.2,5.7,2.3,Iris-
virginica
122,5.6,2.8,4.9,2.0,Iris-
virginica
123,7.7,2.8,6.7,2.0,Iris-
virginica
124,6.3,2.7,4.9,1.8,Iris-
virginica
125,6.7,3.3,5.7,2.1,Iris-
virginica
126,7.2,3.2,6.0,1.8,Iris-
virginica
127,6.2,2.8,4.8,1.8,Iris-
virginica
128,6.1,3.0,4.9,1.8,Iris-
virginica
129,6.4,2.8,5.6,2.1,Iris-
virginica
130,7.2,3.0,5.8,1.6,Iris-
virginica
131,7.4,2.8,6.1,1.9,Iris-
virginica
```

```
132,7.9,3.8,6.4,2.0,Iris-
virginica
133,6.4,2.8,5.6,2.2,Iris-
virginica
134,6.3,2.8,5.1,1.5,Iris-
virginica
135,6.1,2.6,5.6,1.4,Iris-
virginica
136,7.7,3.0,6.1,2.3,Iris-
virginica
137,6.3,3.4,5.6,2.4,Iris-
virginica
138,6.4,3.1,5.5,1.8,Iris-
virginica
139,6.0,3.0,4.8,1.8,Iris-
virginica
140,6.9,3.1,5.4,2.1,Iris-
virginica
141,6.7,3.1,5.6,2.4,Iris-
virginica
142,6.9,3.1,5.1,2.3,Iris-
virginica
143,5.8,2.7,5.1,1.9,Iris-
virginica
144,6.8,3.2,5.9,2.3,Iris-
virginica
145,6.7,3.3,5.7,2.5,Iris-
virginica
146,6.7,3.0,5.2,2.3,Iris-
virginica
147,6.3,2.5,5.0,1.9,Iris-
virginica
148,6.5,3.0,5.2,2.0,Iris-
virginica
149,6.2,3.4,5.4,2.3,Iris-
virginica
150,5.9,3.0,5.1,1.8,Iris-
virginica
```

Program - 08

Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

```
import pandas as pd
import numpy as np
import sklearn as sl
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
iris=datasets.load_iris()
iris.data.shape,iris.target.shape
#((150, 4), (150,))

X_train,X_test,y_train,y_test=train_test_split(iris.data,iris.tar
get,test_size=0.2,random_state=0)
X_train.shape,y_train.shape
#((120, 4), (120,))

X_test.shape,y_train.shape
#((30, 4), (30,))

clf=KNeighborsClassifier()
clf.fit(X_train,y_train)
KNeighborsClassifier(algorithm='auto',leaf_size=30,metric='minkow
ski',metric_params=None, n_jobs=None, n_neighbors=5, p=2,
weights='uniform')
clf.score(X_test,y_test)
accuracy=clf.score(X_test,y_test)
print(accuracy)

example_measures=np.array([[4.7,3.2,2,0.2],[5.1,2.4,4.3,1.3]])
example=example_measures.reshape(2,-1)
prediction=clf.predict(example)
print(prediction)
```

Iris.csv

```
Id,SepalLengthCm,SepalWidthCm,Pe
talLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
8,5.0,3.4,1.5,0.2,Iris-setosa
9,4.4,2.9,1.4,0.2,Iris-setosa
10,4.9,3.1,1.5,0.1,Iris-setosa
11,5.4,3.7,1.5,0.2,Iris-setosa
12,4.8,3.4,1.6,0.2,Iris-setosa
13,4.8,3.0,1.4,0.1,Iris-setosa
14,4.3,3.0,1.1,0.1,Iris-setosa
15,5.8,4.0,1.2,0.2,Iris-setosa
16,5.7,4.4,1.5,0.4,Iris-setosa
17,5.4,3.9,1.3,0.4,Iris-setosa
18,5.1,3.5,1.4,0.3,Iris-setosa
19,5.7,3.8,1.7,0.3,Iris-setosa
20,5.1,3.8,1.5,0.3,Iris-setosa
```

```
21,5.4,3.4,1.7,0.2,Iris-setosa
22,5.1,3.7,1.5,0.4,Iris-setosa
23,4.6,3.6,1.0,0.2,Iris-setosa
24,5.1,3.3,1.7,0.5,Iris-setosa
25,4.8,3.4,1.9,0.2,Iris-setosa
26,5.0,3.0,1.6,0.2,Iris-setosa
27,5.0,3.4,1.6,0.4,Iris-setosa
28,5.2,3.5,1.5,0.2,Iris-setosa
29,5.2,3.4,1.4,0.2,Iris-setosa
30,4.7,3.2,1.6,0.2,Iris-setosa
31,4.8,3.1,1.6,0.2,Iris-setosa
32,5.4,3.4,1.5,0.4,Iris-setosa
33,5.2,4.1,1.5,0.1,Iris-setosa
34,5.5,4.2,1.4,0.2,Iris-setosa
35,4.9,3.1,1.5,0.1,Iris-setosa
36,5.0,3.2,1.2,0.2,Iris-setosa
37,5.5,3.5,1.3,0.2,Iris-setosa
38,4.9,3.1,1.5,0.1,Iris-setosa
39,4.4,3.0,1.3,0.2,Iris-setosa
40,5.1,3.4,1.5,0.2,Iris-setosa
41,5.0,3.5,1.3,0.3,Iris-setosa
42,4.5,2.3,1.3,0.3,Iris-setosa
43,4.4,3.2,1.3,0.2,Iris-setosa
44,5.0,3.5,1.6,0.6,Iris-setosa
45,5.1,3.8,1.9,0.4,Iris-setosa
46,4.8,3.0,1.4,0.3,Iris-setosa
47,5.1,3.8,1.6,0.2,Iris-setosa
48,4.6,3.2,1.4,0.2,Iris-setosa
49,5.3,3.7,1.5,0.2,Iris-setosa
50,5.0,3.3,1.4,0.2,Iris-setosa
51,7.0,3.2,4.7,1.4,Iris-
versicolor
52,6.4,3.2,4.5,1.5,Iris-
versicolor
53,6.9,3.1,4.9,1.5,Iris-
versicolor
54,5.5,2.3,4.0,1.3,Iris-
versicolor
55,6.5,2.8,4.6,1.5,Iris-
versicolor
56,5.7,2.8,4.5,1.3,Iris-
versicolor
57,6.3,3.3,4.7,1.6,Iris-
versicolor
58,4.9,2.4,3.3,1.0,Iris-
versicolor
59,6.6,2.9,4.6,1.3,Iris-
versicolor
60,5.2,2.7,3.9,1.4,Iris-
versicolor
61,5.0,2.0,3.5,1.0,Iris-
versicolor
62,5.9,3.0,4.2,1.5,Iris-
versicolor
63,6.0,2.2,4.0,1.0,Iris-
versicolor
64,6.1,2.9,4.7,1.4,Iris-
versicolor
65,5.6,2.9,3.6,1.3,Iris-
versicolor
66,6.7,3.1,4.4,1.4,Iris-
versicolor
67,5.6,3.0,4.5,1.5,Iris-
versicolor
68,5.8,2.7,4.1,1.0,Iris-
versicolor
69,6.2,2.2,4.5,1.5,Iris-
versicolor
70,5.6,2.5,3.9,1.1,Iris-
versicolor
71,5.9,3.2,4.8,1.8,Iris-
versicolor
72,6.1,2.8,4.0,1.3,Iris-
versicolor
73,6.3,2.5,4.9,1.5,Iris-
versicolor
74,6.1,2.8,4.7,1.2,Iris-
versicolor
75,6.4,2.9,4.3,1.3,Iris-
versicolor
76,6.6,3.0,4.4,1.4,Iris-
versicolor
77,6.8,2.8,4.8,1.4,Iris-
versicolor
78,6.7,3.0,5.0,1.7,Iris-
versicolor
79,6.0,2.9,4.5,1.5,Iris-
versicolor
```

```
80,5.7,2.6,3.5,1.0,Iris-
versicolor
81,5.5,2.4,3.8,1.1,Iris-
versicolor
82,5.5,2.4,3.7,1.0,Iris-
versicolor
83,5.8,2.7,3.9,1.2,Iris-
versicolor
84,6.0,2.7,5.1,1.6,Iris-
versicolor
85,5.4,3.0,4.5,1.5,Iris-
versicolor
86,6.0,3.4,4.5,1.6,Iris-
versicolor
87,6.7,3.1,4.7,1.5,Iris-
versicolor
88,6.3,2.3,4.4,1.3,Iris-
versicolor
89,5.6,3.0,4.1,1.3,Iris-
versicolor
90,5.5,2.5,4.0,1.3,Iris-
versicolor
91,5.5,2.6,4.4,1.2,Iris-
versicolor
92,6.1,3.0,4.6,1.4,Iris-
versicolor
93,5.8,2.6,4.0,1.2,Iris-
versicolor
94,5.0,2.3,3.3,1.0,Iris-
versicolor
95,5.6,2.7,4.2,1.3,Iris-
versicolor
96,5.7,3.0,4.2,1.2,Iris-
versicolor
97,5.7,2.9,4.2,1.3,Iris-
versicolor
98,6.2,2.9,4.3,1.3,Iris-
versicolor
99,5.1,2.5,3.0,1.1,Iris-
versicolor
100,5.7,2.8,4.1,1.3,Iris-
versicolor
101,6.3,3.3,6.0,2.5,Iris-
virginica
102,5.8,2.7,5.1,1.9,Iris-
virginica
103,7.1,3.0,5.9,2.1,Iris-
virginica
104,6.3,2.9,5.6,1.8,Iris-
virginica
105,6.5,3.0,5.8,2.2,Iris-
virginica
106,7.6,3.0,6.6,2.1,Iris-
virginica
107,4.9,2.5,4.5,1.7,Iris-
virginica
108,7.3,2.9,6.3,1.8,Iris-
virginica
109,6.7,2.5,5.8,1.8,Iris-
virginica
110,7.2,3.6,6.1,2.5,Iris-
virginica
111,6.5,3.2,5.1,2.0,Iris-
virginica
112,6.4,2.7,5.3,1.9,Iris-
virginica
113,6.8,3.0,5.5,2.1,Iris-
virginica
114,5.7,2.5,5.0,2.0,Iris-
virginica
115,5.8,2.8,5.1,2.4,Iris-
virginica
116,6.4,3.2,5.3,2.3,Iris-
virginica
117,6.5,3.0,5.5,1.8,Iris-
virginica
118,7.7,3.8,6.7,2.2,Iris-
virginica
119,7.7,2.6,6.9,2.3,Iris-
virginica
120,6.0,2.2,5.0,1.5,Iris-
virginica
121,6.9,3.2,5.7,2.3,Iris-
virginica
122,5.6,2.8,4.9,2.0,Iris-
virginica
123,7.7,2.8,6.7,2.0,Iris-
virginica
```

```
124,6.3,2.7,4.9,1.8,Iris-
virginica
125,6.7,3.3,5.7,2.1,Iris-
virginica
126,7.2,3.2,6.0,1.8,Iris-
virginica
127,6.2,2.8,4.8,1.8,Iris-
virginica
128,6.1,3.0,4.9,1.8,Iris-
virginica
129,6.4,2.8,5.6,2.1,Iris-
virginica
130,7.2,3.0,5.8,1.6,Iris-
virginica
131,7.4,2.8,6.1,1.9,Iris-
virginica
132,7.9,3.8,6.4,2.0,Iris-
virginica
133,6.4,2.8,5.6,2.2,Iris-
virginica
134,6.3,2.8,5.1,1.5,Iris-
virginica
135,6.1,2.6,5.6,1.4,Iris-
virginica
136,7.7,3.0,6.1,2.3,Iris-
virginica
137,6.3,3.4,5.6,2.4,Iris-
virginica
138,6.4,3.1,5.5,1.8,Iris-
virginica
139,6.0,3.0,4.8,1.8,Iris-
virginica
140,6.9,3.1,5.4,2.1,Iris-
virginica
141,6.7,3.1,5.6,2.4,Iris-
virginica
142,6.9,3.1,5.1,2.3,Iris-
virginica
143,5.8,2.7,5.1,1.9,Iris-
virginica
144,6.8,3.2,5.9,2.3,Iris-
virginica
145,6.7,3.3,5.7,2.5,Iris-
virginica
146,6.7,3.0,5.2,2.3,Iris-
virginica
147,6.3,2.5,5.0,1.9,Iris-
virginica
148,6.5,3.0,5.2,2.0,Iris-
virginica
149,6.2,3.4,5.4,2.3,Iris-
virginica
150,5.9,3.0,5.1,1.8,Iris-
virginica
```

Program - 09

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def kernal(point,xmat,k):
    m,n=np.shape(xmat)
    weights=np.mat(np.eye((m)))
    for j in range(m):
        diff=point-x[j]
        weights[j,j]=np.exp(diff*diff.T/(-2.0*k**2))
        return weights

def localweight(point,xmat,ymat,k):
    wt=kernal(point,xmat,k)
    w=(x.T*(wt*x)).I*(x.T*wt*ymat.T)
    return w

def localweightregression(xmat,ymat,k):
    m,n=np.shape(xmat)
    ypred=np.zeros(m)
    #print(m)
    #print(n)
    #print(ypred)
    for i in range(m):
        ypred[i]=xmat[i]*localweight(xmat[i],xmat,ymat,k)
        print(ypred[i])
    return ypred

data=pd.read_csv('Tips.csv')
cola=np.array(data.total_bill)
colb=np.array(data.tip)
#print(cola)
#print(colb)
mcola=np.mat(cola)
#print(mcola)
mcolb=np.mat(colb)
#print(mcolb)
m=np.shape(mcolb)[1]
#print(m)
one=np.ones((1,m),dtype=int)
#print(one)
x=np.hstack((one.T,mcola.T))
print(x.shape)
#print(x)
ypred=localweightregression(x,mcolb,0.5)
#print(ypred)
xsort=x.copy()
```

```
xsort.sort(axis=0)
#print(xsort)
plt.scatter(cola,colb,color='blue')
plt.plot(xsort[:,1],ypred[x[:,1].argsort(0)],color='yellow',linewidth=5)
plt.xlabel('Total Bill')
plt.ylabel('tip')
plt.show()
```

Tips.csv

```
total_bill,tip,sex,smoker,day,time,size
16.99,1.01,Female,No,Sun,Dinner,2
10.34,1.66,Male,No,Sun,Dinner,3
21.01,3.5,Male,No,Sun,Dinner,3
23.68,3.31,Male,No,Sun,Dinner,2
24.59,3.61,Female,No,Sun,Dinner,4
25.29,4.71,Male,No,Sun,Dinner,4
8.77,2.0,Male,No,Sun,Dinner,2
26.88,3.12,Male,No,Sun,Dinner,4
15.04,1.96,Male,No,Sun,Dinner,2
14.78,3.23,Male,No,Sun,Dinner,2
10.27,1.71,Male,No,Sun,Dinner,2
35.26,5.0,Female,No,Sun,Dinner,4
15.42,1.57,Male,No,Sun,Dinner,2
18.43,3.0,Male,No,Sun,Dinner,4
14.83,3.02,Female,No,Sun,Dinner,2
21.58,3.92,Male,No,Sun,Dinner,2
10.33,1.67,Female,No,Sun,Dinner,3
16.29,3.71,Male,No,Sun,Dinner,3
16.97,3.5,Female,No,Sun,Dinner,3
20.65,3.35,Male,No,Sat,Dinner,3
17.92,4.08,Male,No,Sat,Dinner,2
20.29,2.75,Female,No,Sat,Dinner,2
15.77,2.23,Female,No,Sat,Dinner,2
39.42,7.58,Male,No,Sat,Dinner,4
19.82,3.18,Male,No,Sat,Dinner,2
17.81,2.34,Male,No,Sat,Dinner,4
13.37,2.0,Male,No,Sat,Dinner,2
12.69,2.0,Male,No,Sat,Dinner,2
21.7,4.3,Male,No,Sat,Dinner,2
19.65,3.0,Female,No,Sat,Dinner,2
9.55,1.45,Male,No,Sat,Dinner,2
18.35,2.5,Male,No,Sat,Dinner,4
15.06,3.0,Female,No,Sat,Dinner,2
20.69,2.45,Female,No,Sat,Dinner,4
17.78,3.27,Male,No,Sat,Dinner,2
24.06,3.6,Male,No,Sat,Dinner,3
16.31,2.0,Male,No,Sat,Dinner,3
16.93,3.07,Female,No,Sat,Dinner,3
18.69,2.31,Male,No,Sat,Dinner,3
31.27,5.0,Male,No,Sat,Dinner,3
16.04,2.24,Male,No,Sat,Dinner,3
17.46,2.54,Male,No,Sun,Dinner,2
13.94,3.06,Male,No,Sun,Dinner,2
9.68,1.32,Male,No,Sun,Dinner,2
30.4,5.6,Male,No,Sun,Dinner,4
18.29,3.0,Male,No,Sun,Dinner,2
22.23,5.0,Male,No,Sun,Dinner,2
32.4,6.0,Male,No,Sun,Dinner,4
28.55,2.05,Male,No,Sun,Dinner,3
18.04,3.0,Male,No,Sun,Dinner,2
12.54,2.5,Male,No,Sun,Dinner,2
10.29,2.6,Female,No,Sun,Dinner,2
34.81,5.2,Female,No,Sun,Dinner,4
9.94,1.56,Male,No,Sun,Dinner,2
25.56,4.34,Male,No,Sun,Dinner,4
19.49,3.51,Male,No,Sun,Dinner,2
38.01,3.0,Male,Yes,Sat,Dinner,4
26.41,1.5,Female,No,Sat,Dinner,2
11.24,1.76,Male,Yes,Sat,Dinner,2
48.27,6.73,Male,No,Sat,Dinner,4
20.29,3.21,Male,Yes,Sat,Dinner,2
13.81,2.0,Male,Yes,Sat,Dinner,2
```

```
11.02,1.98,Male,Yes,Sat,Dinner,2
18.29,3.76,Male,Yes,Sat,Dinner,4
17.59,2.64,Male,No,Sat,Dinner,3
20.08,3.15,Male,No,Sat,Dinner,3
16.45,2.47,Female,No,Sat,Dinner,2
3.07,1.0,Female,Yes,Sat,Dinner,1
20.23,2.01,Male,No,Sat,Dinner,2
15.01,2.09,Male,Yes,Sat,Dinner,2
12.02,1.97,Male,No,Sat,Dinner,2
17.07,3.0,Female,No,Sat,Dinner,3
26.86,3.14,Female,Yes,Sat,Dinner,2
25.28,5.0,Female,Yes,Sat,Dinner,2
14.73,2.2,Female,No,Sat,Dinner,2
10.51,1.25,Male,No,Sat,Dinner,2
17.92,3.08,Male,Yes,Sat,Dinner,2
27.2,4.0,Male,No,Thur,Lunch,4
22.76,3.0,Male,No,Thur,Lunch,2
17.29,2.71,Male,No,Thur,Lunch,2
19.44,3.0,Male,Yes,Thur,Lunch,2
16.66,3.4,Male,No,Thur,Lunch,2
10.07,1.83,Female,No,Thur,Lunch,1
32.68,5.0,Male,Yes,Thur,Lunch,2
15.98,2.03,Male,No,Thur,Lunch,2
34.83,5.17,Female,No,Thur,Lunch,4
13.03,2.0,Male,No,Thur,Lunch,2
18.28,4.0,Male,No,Thur,Lunch,2
24.71,5.85,Male,No,Thur,Lunch,2
21.16,3.0,Male,No,Thur,Lunch,2
28.97,3.0,Male,Yes,Fri,Dinner,2
22.49,3.5,Male,No,Fri,Dinner,2
5.75,1.0,Female,Yes,Fri,Dinner,2
16.32,4.3,Female,Yes,Fri,Dinner,2
22.75,3.25,Female,No,Fri,Dinner,2
40.17,4.73,Male,Yes,Fri,Dinner,4
27.28,4.0,Male,Yes,Fri,Dinner,2
12.03,1.5,Male,Yes,Fri,Dinner,2
21.01,3.0,Male,Yes,Fri,Dinner,2
12.46,1.5,Male,No,Fri,Dinner,2
11.35,2.5,Female,Yes,Fri,Dinner,2
15.38,3.0,Female,Yes,Fri,Dinner,2
44.3,2.5,Female,Yes,Sat,Dinner,3
22.42,3.48,Female,Yes,Sat,Dinner,2
20.92,4.08,Female,No,Sat,Dinner,2
15.36,1.64,Male,Yes,Sat,Dinner,2
20.49,4.06,Male,Yes,Sat,Dinner,2
25.21,4.29,Male,Yes,Sat,Dinner,2
18.24,3.76,Male,No,Sat,Dinner,2
14.31,4.0,Female,Yes,Sat,Dinner,2
14.0,3.0,Male,No,Sat,Dinner,2
7.25,1.0,Female,No,Sat,Dinner,1
38.07,4.0,Male,No,Sun,Dinner,3
23.95,2.55,Male,No,Sun,Dinner,2
25.71,4.0,Female,No,Sun,Dinner,3
17.31,3.5,Female,No,Sun,Dinner,2
29.93,5.07,Male,No,Sun,Dinner,4
10.65,1.5,Female,No,Thur,Lunch,2
12.43,1.8,Female,No,Thur,Lunch,2
24.08,2.92,Female,No,Thur,Lunch,4
11.69,2.31,Male,No,Thur,Lunch,2
13.42,1.68,Female,No,Thur,Lunch,2
14.26,2.5,Male,No,Thur,Lunch,2
15.95,2.0,Male,No,Thur,Lunch,2
12.48,2.52,Female,No,Thur,Lunch,2
29.8,4.2,Female,No,Thur,Lunch,6
8.52,1.48,Male,No,Thur,Lunch,2
14.52,2.0,Female,No,Thur,Lunch,2
11.38,2.0,Female,No,Thur,Lunch,2
22.82,2.18,Male,No,Thur,Lunch,3
19.08,1.5,Male,No,Thur,Lunch,2
20.27,2.83,Female,No,Thur,Lunch,2
11.17,1.5,Female,No,Thur,Lunch,2
12.26,2.0,Female,No,Thur,Lunch,2
18.26,3.25,Female,No,Thur,Lunch,2
```

```
8.51,1.25,Female,No,Thur,Lunch,2
10.33,2.0,Female,No,Thur,Lunch,2
14.15,2.0,Female,No,Thur,Lunch,2
16.0,2.0,Male,Yes,Thur,Lunch,2
13.16,2.75,Female,No,Thur,Lunch,
2
17.47,3.5,Female,No,Thur,Lunch,2
34.3,6.7,Male,No,Thur,Lunch,6
41.19,5.0,Male,No,Thur,Lunch,5
27.05,5.0,Female,No,Thur,Lunch,6
16.43,2.3,Female,No,Thur,Lunch,2
8.35,1.5,Female,No,Thur,Lunch,2
18.64,1.36,Female,No,Thur,Lunch,
3
11.87,1.63,Female,No,Thur,Lunch,
2
9.78,1.73,Male,No,Thur,Lunch,2
7.51,2.0,Male,No,Thur,Lunch,2
14.07,2.5,Male,No,Sun,Dinner,2
13.13,2.0,Male,No,Sun,Dinner,2
17.26,2.74,Male,No,Sun,Dinner,3
24.55,2.0,Male,No,Sun,Dinner,4
19.77,2.0,Male,No,Sun,Dinner,4
29.85,5.14,Female,No,Sun,Dinner,
5
48.17,5.0,Male,No,Sun,Dinner,6
25.0,3.75,Female,No,Sun,Dinner,4
13.39,2.61,Female,No,Sun,Dinner,
2
16.49,2.0,Male,No,Sun,Dinner,4
21.5,3.5,Male,No,Sun,Dinner,4
12.66,2.5,Male,No,Sun,Dinner,2
16.21,2.0,Female,No,Sun,Dinner,3
13.81,2.0,Male,No,Sun,Dinner,2
17.51,3.0,Female,Yes,Sun,Dinner,
2
24.52,3.48,Male,No,Sun,Dinner,3
20.76,2.24,Male,No,Sun,Dinner,2
31.71,4.5,Male,No,Sun,Dinner,4
10.59,1.61,Female,Yes,Sat,Dinner
,2
10.63,2.0,Female,Yes,Sat,Dinner,
2
50.81,10.0,Male,Yes,Sat,Dinner,3
15.81,3.16,Male,Yes,Sat,Dinner,2
7.25,5.15,Male,Yes,Sun,Dinner,2
31.85,3.18,Male,Yes,Sun,Dinner,2
16.82,4.0,Male,Yes,Sun,Dinner,2
32.9,3.11,Male,Yes,Sun,Dinner,2
17.89,2.0,Male,Yes,Sun,Dinner,2
14.48,2.0,Male,Yes,Sun,Dinner,2
9.6,4.0,Female,Yes,Sun,Dinner,2
34.63,3.55,Male,Yes,Sun,Dinner,2
34.65,3.68,Male,Yes,Sun,Dinner,4
23.33,5.65,Male,Yes,Sun,Dinner,2
45.35,3.5,Male,Yes,Sun,Dinner,3
23.17,6.5,Male,Yes,Sun,Dinner,4
40.55,3.0,Male,Yes,Sun,Dinner,2
20.69,5.0,Male,No,Sun,Dinner,5
20.9,3.5,Female,Yes,Sun,Dinner,3
30.46,2.0,Male,Yes,Sun,Dinner,5
18.15,3.5,Female,Yes,Sun,Dinner,
3
23.1,4.0,Male,Yes,Sun,Dinner,3
15.69,1.5,Male,Yes,Sun,Dinner,2
19.81,4.19,Female,Yes,Thur,Lunch
,2
28.44,2.56,Male,Yes,Thur,Lunch,2
15.48,2.02,Male,Yes,Thur,Lunch,2
16.58,4.0,Male,Yes,Thur,Lunch,2
7.56,1.44,Male,No,Thur,Lunch,2
10.34,2.0,Male,Yes,Thur,Lunch,2
43.11,5.0,Female,Yes,Thur,Lunch,
4
13.0,2.0,Female,Yes,Thur,Lunch,2
13.51,2.0,Male,Yes,Thur,Lunch,2
18.71,4.0,Male,Yes,Thur,Lunch,3
12.74,2.01,Female,Yes,Thur,Lunch
,2
13.0,2.0,Female,Yes,Thur,Lunch,2
16.4,2.5,Female,Yes,Thur,Lunch,2
20.53,4.0,Male,Yes,Thur,Lunch,4
16.47,3.23,Female,Yes,Thur,Lunch
,3
26.59,3.41,Male,Yes,Sat,Dinner,3
38.73,3.0,Male,Yes,Sat,Dinner,4
24.27,2.03,Male,Yes,Sat,Dinner,2
12.76,2.23,Female,Yes,Sat,Dinner
,2
30.06,2.0,Male,Yes,Sat,Dinner,3
```

```
25.89,5.16,Male,Yes,Sat,Dinner,4
48.33,9.0,Male,No,Sat,Dinner,4
13.27,2.5,Female,Yes,Sat,Dinner,2
28.17,6.5,Female,Yes,Sat,Dinner,3
12.9,1.1,Female,Yes,Sat,Dinner,2
28.15,3.0,Male,Yes,Sat,Dinner,5
11.59,1.5,Male,Yes,Sat,Dinner,2
7.74,1.44,Male,Yes,Sat,Dinner,2
30.14,3.09,Female,Yes,Sat,Dinner,4
12.16,2.2,Male,Yes,Fri,Lunch,2
13.42,3.48,Female,Yes,Fri,Lunch,2
8.58,1.92,Male,Yes,Fri,Lunch,1
15.98,3.0,Female,No,Fri,Lunch,3
13.42,1.58,Male,Yes,Fri,Lunch,2
16.27,2.5,Female,Yes,Fri,Lunch,2
10.09,2.0,Female,Yes,Fri,Lunch,2
20.45,3.0,Male,No,Sat,Dinner,4
13.28,2.72,Male,No,Sat,Dinner,2
22.12,2.88,Female,Yes,Sat,Dinner,2
24.01,2.0,Male,Yes,Sat,Dinner,4
15.69,3.0,Male,Yes,Sat,Dinner,3
11.61,3.39,Male,No,Sat,Dinner,2
10.77,1.47,Male,No,Sat,Dinner,2
15.53,3.0,Male,Yes,Sat,Dinner,2
10.07,1.25,Male,No,Sat,Dinner,2
12.6,1.0,Male,Yes,Sat,Dinner,2
32.83,1.17,Male,Yes,Sat,Dinner,2
35.83,4.67,Female,No,Sat,Dinner,3
29.03,5.92,Male,No,Sat,Dinner,3
27.18,2.0,Female,Yes,Sat,Dinner,2
22.67,2.0,Male,Yes,Sat,Dinner,2
17.82,1.75,Male,No,Sat,Dinner,2
18.78,3.0,Female,No,Thur,Dinner,2
```