SRI SAIRAM COLLEGE OF ENGINEERING
ANEKAL, BENGALURU-562106

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ARTIFICIAL INTELLIGENE & MACHINE LEARNING LABORATORY MANUAL

18CSL76

FOR B.E , VII Semester
AS PER VTU NEW SYLLABUS (CBCS)

Prepared by

Prof. C.Sharon Roji Priya
Mr. V.A.Guruprasath

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

(Effective from the academic year 2018 -2019)

SEMESTER – VII

Course Code	18CSL76	CIE Marks	40
Number of Contact Hours/Week	0:0:2	SEE Marks	60
Total Number of Lab Contact Hours	36	Exam Hours	03

Credits – 2

Course Learning Objectives: This course (18CSL76) will enable students to:

- Implement and evaluate AI and ML algorithms in and Python programming language.

Descriptions (if any):

Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.

Programs List:

1.	Implement A* Search algorithm.
2.	Implement AO* Search algorithm.
3.	For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithmto output a description of the set of all hypotheses consistent with the training examples.
4.	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 toclassify a new sample.
5.	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.
6.	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.
7.	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.
8.	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.
9.	Implement the non-parametric Locally Weighted Regressionalgorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Laboratory Outcomes: The student should be able to:

- Implement and demonstrate AI and ML algorithms.
- Evaluate different algorithms.

Conduct of Practical Examination:

- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
  - q) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  - r) For laboratories having PART A and PART B
    - i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
    - ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Ex. No 1 Implement A* searching technique using Python Programming.

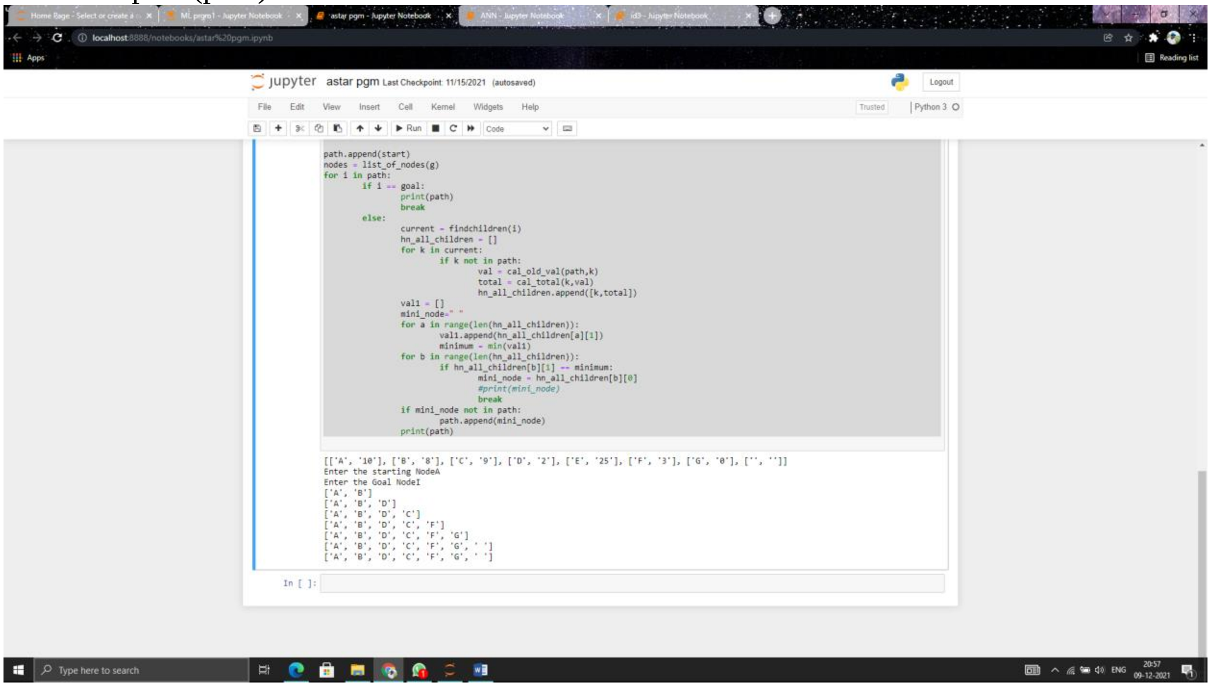
Aim: To implement and test A* searching technique using Python Programming.

Program:

```
import csv

#Function to read the graph

def readgraph(gfile):
        graph = csv.reader(open(gfile, "rt"))
        garray = list(graph)
        return garray

#Function to read the heuristic values

def readheuristic(hfile):
   heu = csv.reader(open(hfile,"rt"))
   H = list(heu)
   return H

def findchildren(Node):
   global g
   child = []
   for i in range(len(g)-1):
      if Node == g[i][0]:
            child.append(g[i][1])
            #child.append(g[i+1][1])
   return child

def calculate_prev_fn(X,P):
   P.append(X)
   prevfn = 0
   if P == []:
      return(0)
   else:
      for i in range (len(P)-1):
         for j in range(len(g)-1):
            if P[i] == g[j][0] and P[i+1] == g[j][1]:
               prevfn = prevfn + int(g[j][2])
   return(prevfn)

def cal_old_val(X,Y):
      L = []
      for c in range(len(X)):
            L.append(X[c])
      L.append(Y)

      oldval = 0
      if L ==[]:
            return (0)
```

```
    else:
        for i in range(len(L)-1):
            for j in range(len(g)-1):
                if L[i] == g[j][0] and L[i+1] == g[j][1]:
                    oldval = oldval + int(g[j][2])

    return(oldval)


def list_of_nodes(g):
  nodes = []
  j=0
  for i in range(len(g)):
    if g[i][0] not in nodes:
      nodes.append(g[i][0])
      j=j+1

def cal_total(k,val):
    for i in range(len(h)):
        if h[i][0] == k:
            total = val + int(h[i][1])
            return(total)

visited = []
path = []
filegraph = "OneDrive\Desktop\graph.csv"
g = readgraph(filegraph)
#print(g)
hfile = "OneDrive\Desktop\heuristic.csv"
h = readheuristic(hfile)
print(h)
start = input("Enter the starting Node")
goal = input("Enter the Goal Node")

path.append(start)
nodes = list_of_nodes(g)
for i in path:
    if i == goal:
        print(path)
        break
    else:
        current = findchildren(i)
        hn_all_children = []
        for k in current:
            if k not in path:
                val = cal_old_val(path,k)
                total = cal_total(k,val)
                hn_all_children.append([k,total])
        val1 = []
        mini_node=" "
        for a in range(len(hn_all_children)):
            val1.append(hn_all_children[a][1])
```

```
                minimum = min(val1)
        for b in range(len(hn_all_children)):
                if hn_all_children[b][1] == minimum:
                        mini_node = hn_all_children[b][0]
                        #print(mini_node)
                        break
        if mini_node not in path:
                path.append(mini_node)
        print(path)
```


jupyter astar pgm Last Checkpoint: 11/15/2021 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
path.append(start)
nodes = list_of_nodes(g)
for i in path:
        if i == goal:
                print(path)
                break
        else:
                current = findchildren(i)
                hn_all_children = []
                for k in current:
                        if k not in path:
                                val = cal_old_val(path,k)
                                total = cal_total(k,val)
                                hn_all_children.append([k,total])
                val1 = []
                mini_node=" "
                for a in range(len(hn_all_children)):
                        val1.append(hn_all_children[a][1])
                        minimum = min(val1)
                for b in range(len(hn_all_children)):
                        if hn_all_children[b][1] == minimum:
                                mini_node = hn_all_children[b][0]
                                #print(mini_node)
                                break
                if mini_node not in path:
                        path.append(mini_node)
                print(path)
```

```
[['A', '10'], ['B', '8'], ['C', '9'], ['D', '2'], ['E', '25'], ['F', '3'], ['G', '0'], ['', '']]
Enter the starting NodeA
Enter the Goal NodeI
['A', 'B']
['A', 'B', 'D']
['A', 'B', 'D', 'C']
['A', 'B', 'D', 'C', 'F']
['A', 'B', 'D', 'C', 'F', 'G']
['A', 'B', 'D', 'C', 'F', 'G', ' ']
['A', 'B', 'D', 'C', 'F', 'G', ' ']
```

```
In [ ]:
```

Cost value from node to node(g(n)):

A	B	3
A	C	4
B	D	2
C	A	4
C	D	1
C	E	3
C	F	2
D	B	2
D	C	1
D	E	1
E	C	3
E	D	1
E	G	4
F	C	2
F	G	1
G	F	1
G	E	4

Heuristic value from Node to Goal(h(n)):

A	10
B	8
C	9
D	2
E	25
F	3
G	0

Ex. No 2 Implement AO* searching technique using Python Programming.

Aim: To implement and test AO* searching technique using Python Programming.

Program:

```
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 Neighbors 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)
            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



h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
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()

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
```

output:

```
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {}
PROCESSING NODE   : B
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 7, 'I': 7, 'J': 1, 'T': 3
}
```

```
SOLUTION GRAPH    : {}
PROCESSING NODE   : G
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {}
PROCESSING NODE   : B
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 10, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 7, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {}
PROCESSING NODE   : I
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 8, 'H': 7, 'I': 0, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {'I': []}
PROCESSING NODE   : G
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 8, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {'I': [], 'G': ['I']}
PROCESSING NODE   : B
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 12, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3
}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : C
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 1, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G']}
PROCESSING NODE   : J
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G'], 'J': []}
PROCESSING NODE   : C
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 6, 'B': 2, 'C': 1, 'D': 12, 'E': 2, 'F': 1, 'G': 1, 'H': 7, 'I': 0, 'J': 0, 'T': 3}
SOLUTION GRAPH    : {'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J']}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'I': [], 'G': ['I'], 'B': ['G'], 'J': [], 'C': ['J'], 'A': ['B', 'C']}
------------------------------------------------------------
HEURISTIC VALUES  : {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {}
PROCESSING NODE   : E
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 10, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 11, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : A
```

```
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 4, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': []}
PROCESSING NODE   : F
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 6, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': [], 'F': []}
PROCESSING NODE   : D
-----------------------------------------------------------------------------------------
HEURISTIC VALUES  : {'A': 7, 'B': 6, 'C': 12, 'D': 2, 'E': 0, 'F': 0, 'G': 5, 'H': 7}
SOLUTION GRAPH    : {'E': [], 'F': [], 'D': ['E', 'F']}
PROCESSING NODE   : A
-----------------------------------------------------------------------------------------
FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A
------------------------------------------------------------
{'E': [], 'F': [], 'D': ['E', 'F'], 'A': ['D']}
------------------------------------------------------------
```

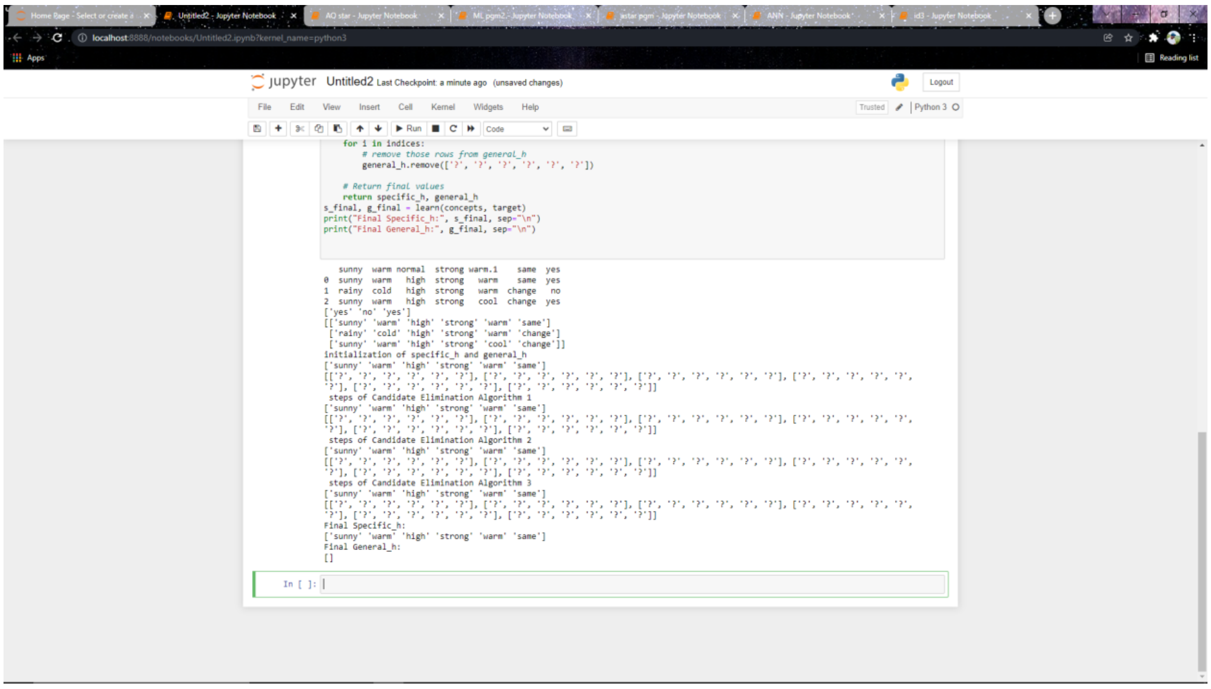
Ex. No 3 For a given set of training data examples stored in a .CSV file, implement and test the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Aim: To implement and test the Candidate-Elimination algorithm which outputs a description of the set of all hypotheses consistent with the training examples.

Program:

```
import numpy as np

import pandas as pd

data = pd.DataFrame(data=pd.read_csv('training_examples.csv'))

print(data)

concepts = np.array(data.iloc[:,0:-1])

target = np.array(data.iloc[:,-1])

print(target)

print(concepts)

def learn(concepts, target):

    '''    learn() function implements the learning method of the Candidate elimination algorithm.
    Arguments:
    concepts - a data frame with all the features
    target - a data frame with corresponding output values
    '''
    # Initialise S0 with the first instance from concepts
    # .copy() makes sure a new list is created instead of just pointing to the same memory location
    specific_h = concepts[0].copy()
    print("initialization of specific_h and general_h")
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print(general_h)
    # The learning iterations
    for i, h in enumerate(concepts):


        # Checking if the hypothesis has a positive target
        if target[i] == "Yes":
            for x in range(len(specific_h)):

                # Change values in S & G only if values change
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'

        # Checking if the hypothesis has a positive target
        if target[i] == "No":
            for x in range(len(specific_h)):

                # For negative hyposthesis change values only  in G
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print(" steps of Candidate Elimination Algorithm",i+1)
        print(specific_h)
        print(general_h)
    # find indices where we have empty rows, meaning those that are unchanged
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        # remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    # Return final values
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

output:



```
    for i in indices:
        # remove those rows from general_h
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    # Return final values
    return specific_h, general_h
s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

```
   sunny  warm normal  strong warm.1    same  yes
0  sunny  warm   high  strong   warm    same  yes
1  rainy  cold   high  strong   warm  change   no
2  sunny  warm   high  strong   cool  change  yes
['yes' 'no' 'yes']
[['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
initialization of specific_h and general_h
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
 steps of Candidate Elimination Algorithm 1
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
 steps of Candidate Elimination Algorithm 2
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
 steps of Candidate Elimination Algorithm 3
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific_h:
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Final General_h:
[]
```

Dataset:

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

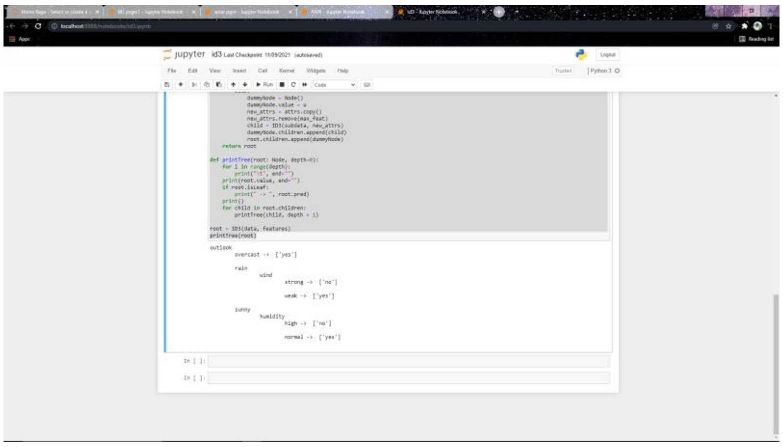
Ex. No 4 Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Aim: To implement and test ID3 algorithm that uses a set of data for building a decision tree.

Program:

```
import pandas as pd
import math
import numpy as np

data = pd.read_csv("OneDrive\Desktop\lab3.csv")
features = [feat for feat in data]
features.remove("answer")

class Node:
    def __init__(self):
        self.children = []
        self.value = ""
        self.isLeaf = False
        self.pred = ""

def entropy(examples):
    pos = 0.0
    neg = 0.0
    for _, row in examples.iterrows():
        if row["answer"] == "yes":
            pos += 1
        else:
            neg += 1
    if pos == 0.0 or neg == 0.0:
        return 0.0
    else:
        p = pos / (pos + neg)
        n = neg / (pos + neg)
        return -(p * math.log(p, 2) + n * math.log(n, 2))

def info_gain(examples, attr):
    uniq = np.unique(examples[attr])
    #print ("\n",uniq)
    gain = entropy(examples)
    #print ("\n",gain)
    for u in uniq:
        subdata = examples[examples[attr] == u]
        #print ("\n",subdata)
        sub_e = entropy(subdata)
        gain -= (float(len(subdata)) / float(len(examples))) * sub_e
        #print ("\n",gain)
    return gain

def ID3(examples, attrs):
    root = Node()

    max_gain = 0
    max_feat = ""
    for feature in attrs:
        #print ("\n",examples)
        gain = info_gain(examples, feature)
        if gain > max_gain:
            max_gain = gain
            max_feat = feature
    root.value = max_feat
    #print ("\nMax feature attr",max_feat)
    uniq = np.unique(examples[max_feat])
    #print ("\n",uniq)
    for u in uniq:
        #print ("\n",u)
        subdata = examples[examples[max_feat] == u]
        #print ("\n",subdata)
        if entropy(subdata) == 0.0:
            newNode = Node()
            newNode.isLeaf = True
            newNode.value = u
            newNode.pred = np.unique(subdata["answer"])
            root.children.append(newNode)
        else:
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = ID3(data, features)
printTree(root)
```

output:



jupyter id3 Last Checkpoint 11/09/2021 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Logout

```
            dummyNode = Node()
            dummyNode.value = u
            new_attrs = attrs.copy()
            new_attrs.remove(max_feat)
            child = ID3(subdata, new_attrs)
            dummyNode.children.append(child)
            root.children.append(dummyNode)
    return root

def printTree(root: Node, depth=0):
    for i in range(depth):
        print("\t", end="")
    print(root.value, end="")
    if root.isLeaf:
        print(" -> ", root.pred)
    print()
    for child in root.children:
        printTree(child, depth + 1)

root = ID3(data, features)
printTree(root)
```

```
outlook
	overcast ->  ['yes']

	rain
		wind
			strong ->  ['no']

			weak ->  ['yes']

	sunny
		humidity
			high ->  ['no']

			normal ->  ['yes']
```

```
In [ ]:

In [ ]:
```

Dataset:

outlook	temperature	humidity	wind	answer
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
sunny	mild	normal	strong	yes
overcast	mild	high	strong	yes
overcast	hot	normal	weak	yes
rain	mild	high	strong	no

Ex. No 5 Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Aim: To build an Artificial Neural Network by implementing the Backpropagation algorithm and test it using appropriate data sets.

Program:

```
import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)
y = np.array(([92], [86], [89]), dtype=float)
X = X/np.amax(X,axis=0) #maximum of X array longitudinally
y = y/100

#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5 #Setting training iterations
lr=0.1 #Setting learning rate

inputlayer_neurons = 2 #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer
#weight and bias initialization

wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

#draws a random range of numbers uniformly of dim x*y
for i in range(epoch):
    #Forward Propogation
    hinp1=np.dot(X,wh)
    hinp=hinp1 + bh
    hlayer_act = sigmoid(hinp)
    outinp1=np.dot(hlayer_act,wout)
    outinp= outinp1+bout
    output = sigmoid(outinp)

    #Backpropagation
    EO = y-output
    outgrad = derivatives_sigmoid(output)
    d_output = EO * outgrad
    EH = d_output.dot(wout.T)
    hiddengrad = derivatives_sigmoid(hlayer_act)#how much hidden layer wts contributed to error
    d_hiddenlayer = EH * hiddengrad

    wout += hlayer_act.T.dot(d_output) *lr   # dotproduct of nextlayererror and currentlayerop
    wh += X.T.dot(d_hiddenlayer) *lr

    print ("-----------Epoch-", i+1, "Starts----------")
    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
```

```
        print("Predicted Output: \n" ,output)
        print ("-----------Epoch-", i+1, "Ends----------\n")

    print("Input: \n" + str(X))
    print("Actual Output: \n" + str(y))
    print("Predicted Output: \n" ,output)
```

Input:

```
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.66485937]
 [0.65534372]
 [0.66364741]]
```

Output:

```
----------Epoch- 1 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.63697455]
 [0.62960753]
 [0.63584726]]
-----------Epoch- 1 Ends----------

-----------Epoch- 2 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.64443549]
 [0.63648142]
 [0.64328423]]
-----------Epoch- 2 Ends----------

-----------Epoch- 3 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.65155845]
 [0.64305189]
 [0.65038519]]
-----------Epoch- 3 Ends----------

-----------Epoch- 4 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.65836081]
 [0.64933436]
 [0.65716736]]
-----------Epoch- 4 Ends----------

-----------Epoch- 5 Starts----------
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.66485937]
 [0.65534372]
 [0.66364741]]
-----------Epoch- 5 Ends----------
```

Ex. No 6 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.

Aim: To implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. To compute the accuracy of the classifier, considering few test data sets.

Program:

```
import pandas as pd
msg = pd.read_csv('document.csv', names=['message', 'label'])    #Note: Give proper address of the csv file
print("Total Instances of Dataset: ", msg.shape[0])
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})

X = msg.message
y = msg.labelnum
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
from sklearn.feature_extraction.text import CountVectorizer

count_v = CountVectorizer()
Xtrain_dm = count_v.fit_transform(Xtrain)
Xtest_dm = count_v.transform(Xtest)

df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
print(df[0:5])

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
pred = clf.predict(Xtest_dm)

for doc, p in zip(Xtrain, pred):
    p = 'pos' if p == 1 else 'neg'
    print("%s -> %s" % (doc, p))

from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(ytest, pred))
print('Recall: ', recall_score(ytest, pred))
print('Precision: ', precision_score(ytest, pred))
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

Dataset:

I love this sandwich	pos
This is an amazing place	pos
I feel very good about these beers	pos
This is my best work	pos
What an awesome view	pos
I do not like this restaurant	neg
I am tired of this stuff	neg
I can't deal with this	neg
He is my sworn enemy	neg

My boss is horrible	neg
This is an awesome place	pos
I do not like the taste of this juice	neg
I love to dance	pos
I am sick and tired of this place	neg
What a great holiday	pos
That is a bad locality to stay	neg
We will have good fun tomorrow	pos
I went to my enemy's house today	neg

Output:

```
I am sick and tired of this place -> pos
I do not like the taste of this juice -> neg
I love this sandwich -> neg
I can’t deal with this -> pos
I do not like this restaurant -> neg

Accuracy Metrics:
Accuracy: 0.6
Recall: 0.5
Precision: 1.0
Confusion Matrix:
[[1 0]
[2 2]]
```

Ex. No 7	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.

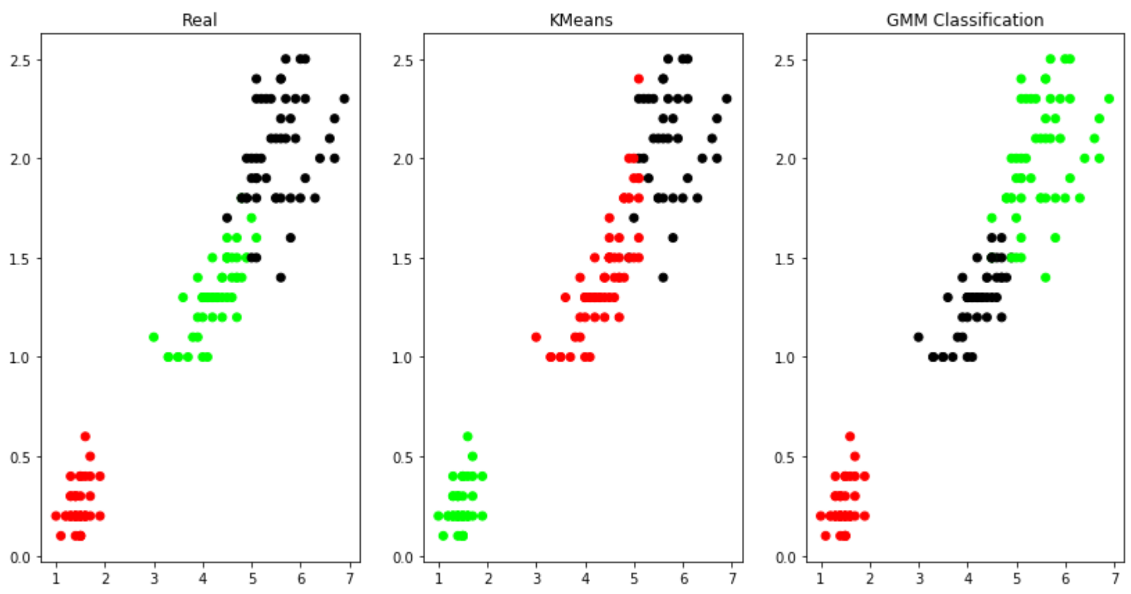
Aim: To apply EM algorithm to cluster a set of data stored in a .CSV file. To clustering using k-Means algorithm with same data set.

Program:

```
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
import sklearn.metrics as metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
names = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width', 'Class']
dataset = pd.read_csv("8-dataset.csv", names=names)  #Note: Give the exact address of the  data set file
X = dataset.iloc[:, :-1]
label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}
y = [label[c] for c in dataset.iloc[:, -1]]
plt.figure(figsize=(14,7))
colormap=np.array(['red','lime','black'])
# REAL PLOT
plt.subplot(1,3,1)
plt.title('Real')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y])

# K-PLOT
model=KMeans(n_clusters=3, random_state=0).fit(X)
plt.subplot(1,3,2)
plt.title('KMeans')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[model.labels_])

print('The accuracy score of K-Mean: ',metrics.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean:\n',metrics.confusion_matrix(y, model.labels_))

# GMM PLOT
gmm=GaussianMixture(n_components=3, random_state=0).fit(X)
y_cluster_gmm=gmm.predict(X)
plt.subplot(1,3,3)
plt.title('GMM Classification')
plt.scatter(X.Petal_Length,X.Petal_Width,c=colormap[y_cluster_gmm])

print('The accuracy score of EM: ',metrics.accuracy_score(y, y_cluster_gmm))
print('The Confusion matrix of EM:\n ',metrics.confusion_matrix(y, y_cluster_gmm)
```

Output:

```
The accuracy score of K-Mean: 0.24
The Confusion matrixof K-Mean:
[[ 0 50 0]
[48 0 2]
[14 0 36]]

The accuracy score of EM: 0.36666666666666664
The Confusion matrix of EM:
[[50 0 0]
[ 0 5 45]
[ 0 50 0]]
```



Dataset:

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa
5.7	3.8	1.7	0.3	Iris-setosa

5.1	3.8	1.5	0.3	Iris-setosa
5.4	3.4	1.7	0.2	Iris-setosa
5.1	3.7	1.5	0.4	Iris-setosa
4.6	3.6	1	0.2	Iris-setosa
5.1	3.3	1.7	0.5	Iris-setosa
4.8	3.4	1.9	0.2	Iris-setosa
5	3	1.6	0.2	Iris-setosa
5	3.4	1.6	0.4	Iris-setosa
5.2	3.5	1.5	0.2	Iris-setosa
5.2	3.4	1.4	0.2	Iris-setosa
4.7	3.2	1.6	0.2	Iris-setosa
4.8	3.1	1.6	0.2	Iris-setosa
5.4	3.4	1.5	0.4	Iris-setosa
5.2	4.1	1.5	0.1	Iris-setosa
5.5	4.2	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5	3.2	1.2	0.2	Iris-setosa
5.5	3.5	1.3	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
4.4	3	1.3	0.2	Iris-setosa
5.1	3.4	1.5	0.2	Iris-setosa
5	3.5	1.3	0.3	Iris-setosa
4.5	2.3	1.3	0.3	Iris-setosa
4.4	3.2	1.3	0.2	Iris-setosa
5	3.5	1.6	0.6	Iris-setosa
5.1	3.8	1.9	0.4	Iris-setosa
4.8	3	1.4	0.3	Iris-setosa
5.1	3.8	1.6	0.2	Iris-setosa
4.6	3.2	1.4	0.2	Iris-setosa
5.3	3.7	1.5	0.2	Iris-setosa
5	3.3	1.4	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
5.7	2.8	4.5	1.3	Iris-versicolor
6.3	3.3	4.7	1.6	Iris-versicolor
4.9	2.4	3.3	1	Iris-versicolor
6.6	2.9	4.6	1.3	Iris-versicolor
5.2	2.7	3.9	1.4	Iris-versicolor
5	2	3.5	1	Iris-versicolor
5.9	3	4.2	1.5	Iris-versicolor
6	2.2	4	1	Iris-versicolor
6.1	2.9	4.7	1.4	Iris-versicolor
5.6	2.9	3.6	1.3	Iris-versicolor
6.7	3.1	4.4	1.4	Iris-versicolor

5.6	3	4.5	1.5	Iris-versicolor
5.8	2.7	4.1	1	Iris-versicolor
6.2	2.2	4.5	1.5	Iris-versicolor
5.6	2.5	3.9	1.1	Iris-versicolor
5.9	3.2	4.8	1.8	Iris-versicolor
6.1	2.8	4	1.3	Iris-versicolor
6.3	2.5	4.9	1.5	Iris-versicolor
6.1	2.8	4.7	1.2	Iris-versicolor
6.4	2.9	4.3	1.3	Iris-versicolor
6.6	3	4.4	1.4	Iris-versicolor
6.8	2.8	4.8	1.4	Iris-versicolor
6.7	3	5	1.7	Iris-versicolor
6	2.9	4.5	1.5	Iris-versicolor
5.7	2.6	3.5	1	Iris-versicolor
5.5	2.4	3.8	1.1	Iris-versicolor
5.5	2.4	3.7	1	Iris-versicolor
5.8	2.7	3.9	1.2	Iris-versicolor
6	2.7	5.1	1.6	Iris-versicolor
5.4	3	4.5	1.5	Iris-versicolor
6	3.4	4.5	1.6	Iris-versicolor
6.7	3.1	4.7	1.5	Iris-versicolor
6.3	2.3	4.4	1.3	Iris-versicolor
5.6	3	4.1	1.3	Iris-versicolor
5.5	2.5	4	1.3	Iris-versicolor
5.5	2.6	4.4	1.2	Iris-versicolor
6.1	3	4.6	1.4	Iris-versicolor
5.8	2.6	4	1.2	Iris-versicolor
5	2.3	3.3	1	Iris-versicolor
5.6	2.7	4.2	1.3	Iris-versicolor
5.7	3	4.2	1.2	Iris-versicolor
5.7	2.9	4.2	1.3	Iris-versicolor
6.2	2.9	4.3	1.3	Iris-versicolor
5.1	2.5	3	1.1	Iris-versicolor
5.7	2.8	4.1	1.3	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica
6.5	3	5.8	2.2	Iris-virginica
7.6	3	6.6	2.1	Iris-virginica
4.9	2.5	4.5	1.7	Iris-virginica
7.3	2.9	6.3	1.8	Iris-virginica
6.7	2.5	5.8	1.8	Iris-virginica
7.2	3.6	6.1	2.5	Iris-virginica
6.5	3.2	5.1	2	Iris-virginica
6.4	2.7	5.3	1.9	Iris-virginica
6.8	3	5.5	2.1	Iris-virginica

5.7	2.5	5	2	Iris-virginica
5.8	2.8	5.1	2.4	Iris-virginica
6.4	3.2	5.3	2.3	Iris-virginica
6.5	3	5.5	1.8	Iris-virginica
7.7	3.8	6.7	2.2	Iris-virginica
7.7	2.6	6.9	2.3	Iris-virginica
6	2.2	5	1.5	Iris-virginica
6.9	3.2	5.7	2.3	Iris-virginica
5.6	2.8	4.9	2	Iris-virginica
7.7	2.8	6.7	2	Iris-virginica
6.3	2.7	4.9	1.8	Iris-virginica
6.7	3.3	5.7	2.1	Iris-virginica
7.2	3.2	6	1.8	Iris-virginica
6.2	2.8	4.8	1.8	Iris-virginica
6.1	3	4.9	1.8	Iris-virginica
6.4	2.8	5.6	2.1	Iris-virginica
7.2	3	5.8	1.6	Iris-virginica
7.4	2.8	6.1	1.9	Iris-virginica
7.9	3.8	6.4	2	Iris-virginica
6.4	2.8	5.6	2.2	Iris-virginica
6.3	2.8	5.1	1.5	Iris-virginica
6.1	2.6	5.6	1.4	Iris-virginica
7.7	3	6.1	2.3	Iris-virginica
6.3	3.4	5.6	2.4	Iris-virginica
6.4	3.1	5.5	1.8	Iris-virginica
6	3	4.8	1.8	Iris-virginica
6.9	3.1	5.4	2.1	Iris-virginica
6.7	3.1	5.6	2.4	Iris-virginica
6.9	3.1	5.1	2.3	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
6.8	3.2	5.9	2.3	Iris-virginica
6.7	3.3	5.7	2.5	Iris-virginica
6.7	3	5.2	2.3	Iris-virginica
6.3	2.5	5	1.9	Iris-virginica
6.5	3	5.2	2	Iris-virginica
6.2	3.4	5.4	2.3	Iris-virginica
5.9	3	5.1	1.8	Iris-virginica

Ex. No 8 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.

Aim: To implement k-Nearest Neighbor algorithm to classify the iris data set.

Program:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe
dataset = pd.read_csv("9-dataset.csv", names=names)  #Note: Give the exact address of the  data set file
X = dataset.iloc[:, :-1]
y = dataset.iloc[:, -1]
print(X.head())
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10)

classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0
print ("\n-------------------------------------------------------------------------")
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print ("-------------------------------------------------------------------------")
for label in ytest:
    print ('%-25s %-25s' % (label, ypred[i]), end="")
    if (label == ypred[i]):
        print (' %-25s' % ('Correct'))
    else:
        print (' %-25s' % ('Wrong'))
    i = i + 1
print ("-------------------------------------------------------------------------")
print("\nConfusion Matrix:\n",metrics.confusion_matrix(ytest, ypred))
print ("-------------------------------------------------------------------------")
print("\nClassification Report:\n",metrics.classification_report(ytest, ypred))
print ("-------------------------------------------------------------------------")
print('Accuracy of the classifer is %0.2f' % metrics.accuracy_score(ytest,ypred))
print ("-------------------------------------------------------------------------")
```

Output:

```
     sepal-length  sepal-width  petal-length  petal-width
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
```

Original Label	**Predicted Label**	**Correct/Wrong**
Iris-versicolor	Iris-versicolor	Correct
Iris-virginica	Iris-versicolor	Wrong
Iris-virginica	Iris-virginica	Correct
Iris-versicolor	Iris-versicolor	Correct
Iris-setosa	Iris-setosa	Correct
Iris-versicolor	Iris-versicolor	Correct
Iris-setosa	Iris-setosa	Correct
Iris-setosa	Iris-setosa	Correct
Iris-virginica	Iris-virginica	Correct
Iris-virginica	Iris-versicolor	Wrong
Iris-virginica	Iris-virginica	Correct
Iris-setosa	Iris-setosa	Correct
Iris-virginica	Iris-virginica	Correct
Iris-virginica	Iris-virginica	Correct
Iris-versicolor	Iris-versicolor	Correct

*Output is manually formatted

Confusion Matrix:

```
 [[4 0 0]
 [0 4 0]
 [0 2 5]]
-----------------------------------------------------------------------

Classification Report:
                 precision    recall    f1-score       support

   Iris-setosa      1.00         1.00     1.00             4
Iris-versicolor     0.67         1.00     0.80             4
 Iris-virginica    1.00          0.71     0.83             7

   avg / total     0.91          0.87     0.87            15

-----------------------------------------------------------------------
Accuracy of the classifier is 0.87
-----------------------------------------------------------------------
```

Dataset:

5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
5.7	4.4	1.5	0.4	Iris-setosa
5.4	3.9	1.3	0.4	Iris-setosa
5.1	3.5	1.4	0.3	Iris-setosa
5.7	3.8	1.7	0.3	Iris-setosa
5.1	3.8	1.5	0.3	Iris-setosa
5.4	3.4	1.7	0.2	Iris-setosa
5.1	3.7	1.5	0.4	Iris-setosa
4.6	3.6	1	0.2	Iris-setosa
5.1	3.3	1.7	0.5	Iris-setosa
4.8	3.4	1.9	0.2	Iris-setosa
5	3	1.6	0.2	Iris-setosa
5	3.4	1.6	0.4	Iris-setosa
5.2	3.5	1.5	0.2	Iris-setosa
5.2	3.4	1.4	0.2	Iris-setosa
4.7	3.2	1.6	0.2	Iris-setosa
4.8	3.1	1.6	0.2	Iris-setosa
5.4	3.4	1.5	0.4	Iris-setosa
5.2	4.1	1.5	0.1	Iris-setosa
5.5	4.2	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5	3.2	1.2	0.2	Iris-setosa
5.5	3.5	1.3	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
4.4	3	1.3	0.2	Iris-setosa
5.1	3.4	1.5	0.2	Iris-setosa
5	3.5	1.3	0.3	Iris-setosa
4.5	2.3	1.3	0.3	Iris-setosa
4.4	3.2	1.3	0.2	Iris-setosa

5	3.5	1.6	0.6	Iris-setosa
5.1	3.8	1.9	0.4	Iris-setosa
4.8	3	1.4	0.3	Iris-setosa
5.1	3.8	1.6	0.2	Iris-setosa
4.6	3.2	1.4	0.2	Iris-setosa
5.3	3.7	1.5	0.2	Iris-setosa
5	3.3	1.4	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
5.7	2.8	4.5	1.3	Iris-versicolor
6.3	3.3	4.7	1.6	Iris-versicolor
4.9	2.4	3.3	1	Iris-versicolor
6.6	2.9	4.6	1.3	Iris-versicolor
5.2	2.7	3.9	1.4	Iris-versicolor
5	2	3.5	1	Iris-versicolor
5.9	3	4.2	1.5	Iris-versicolor
6	2.2	4	1	Iris-versicolor
6.1	2.9	4.7	1.4	Iris-versicolor
5.6	2.9	3.6	1.3	Iris-versicolor
6.7	3.1	4.4	1.4	Iris-versicolor
5.6	3	4.5	1.5	Iris-versicolor
5.8	2.7	4.1	1	Iris-versicolor
6.2	2.2	4.5	1.5	Iris-versicolor
5.6	2.5	3.9	1.1	Iris-versicolor
5.9	3.2	4.8	1.8	Iris-versicolor
6.1	2.8	4	1.3	Iris-versicolor
6.3	2.5	4.9	1.5	Iris-versicolor
6.1	2.8	4.7	1.2	Iris-versicolor
6.4	2.9	4.3	1.3	Iris-versicolor
6.6	3	4.4	1.4	Iris-versicolor
6.8	2.8	4.8	1.4	Iris-versicolor
6.7	3	5	1.7	Iris-versicolor
6	2.9	4.5	1.5	Iris-versicolor
5.7	2.6	3.5	1	Iris-versicolor
5.5	2.4	3.8	1.1	Iris-versicolor
5.5	2.4	3.7	1	Iris-versicolor
5.8	2.7	3.9	1.2	Iris-versicolor
6	2.7	5.1	1.6	Iris-versicolor
5.4	3	4.5	1.5	Iris-versicolor
6	3.4	4.5	1.6	Iris-versicolor
6.7	3.1	4.7	1.5	Iris-versicolor
6.3	2.3	4.4	1.3	Iris-versicolor
5.6	3	4.1	1.3	Iris-versicolor
5.5	2.5	4	1.3	Iris-versicolor

5.5	2.6	4.4	1.2	Iris-versicolor
6.1	3	4.6	1.4	Iris-versicolor
5.8	2.6	4	1.2	Iris-versicolor
5	2.3	3.3	1	Iris-versicolor
5.6	2.7	4.2	1.3	Iris-versicolor
5.7	3	4.2	1.2	Iris-versicolor
5.7	2.9	4.2	1.3	Iris-versicolor
6.2	2.9	4.3	1.3	Iris-versicolor
5.1	2.5	3	1.1	Iris-versicolor
5.7	2.8	4.1	1.3	Iris-versicolor
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica
6.5	3	5.8	2.2	Iris-virginica
7.6	3	6.6	2.1	Iris-virginica
4.9	2.5	4.5	1.7	Iris-virginica
7.3	2.9	6.3	1.8	Iris-virginica
6.7	2.5	5.8	1.8	Iris-virginica
7.2	3.6	6.1	2.5	Iris-virginica
6.5	3.2	5.1	2	Iris-virginica
6.4	2.7	5.3	1.9	Iris-virginica
6.8	3	5.5	2.1	Iris-virginica
5.7	2.5	5	2	Iris-virginica
5.8	2.8	5.1	2.4	Iris-virginica
6.4	3.2	5.3	2.3	Iris-virginica
6.5	3	5.5	1.8	Iris-virginica
7.7	3.8	6.7	2.2	Iris-virginica
7.7	2.6	6.9	2.3	Iris-virginica
6	2.2	5	1.5	Iris-virginica
6.9	3.2	5.7	2.3	Iris-virginica
5.6	2.8	4.9	2	Iris-virginica
7.7	2.8	6.7	2	Iris-virginica
6.3	2.7	4.9	1.8	Iris-virginica
6.7	3.3	5.7	2.1	Iris-virginica
7.2	3.2	6	1.8	Iris-virginica
6.2	2.8	4.8	1.8	Iris-virginica
6.1	3	4.9	1.8	Iris-virginica
6.4	2.8	5.6	2.1	Iris-virginica
7.2	3	5.8	1.6	Iris-virginica
7.4	2.8	6.1	1.9	Iris-virginica
7.9	3.8	6.4	2	Iris-virginica
6.4	2.8	5.6	2.2	Iris-virginica
6.3	2.8	5.1	1.5	Iris-virginica
6.1	2.6	5.6	1.4	Iris-virginica
7.7	3	6.1	2.3	Iris-virginica
6.3	3.4	5.6	2.4	Iris-virginica

6.4	3.1	5.5	1.8	Iris-virginica
6	3	4.8	1.8	Iris-virginica
6.9	3.1	5.4	2.1	Iris-virginica
6.7	3.1	5.6	2.4	Iris-virginica
6.9	3.1	5.1	2.3	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
6.8	3.2	5.9	2.3	Iris-virginica
6.7	3.3	5.7	2.5	Iris-virginica
6.7	3	5.2	2.3	Iris-virginica
6.3	2.5	5	1.9	Iris-virginica
6.5	3	5.2	2	Iris-virginica
6.2	3.4	5.4	2.3	Iris-virginica
5.9	3	5.1	1.8	Iris-virginica

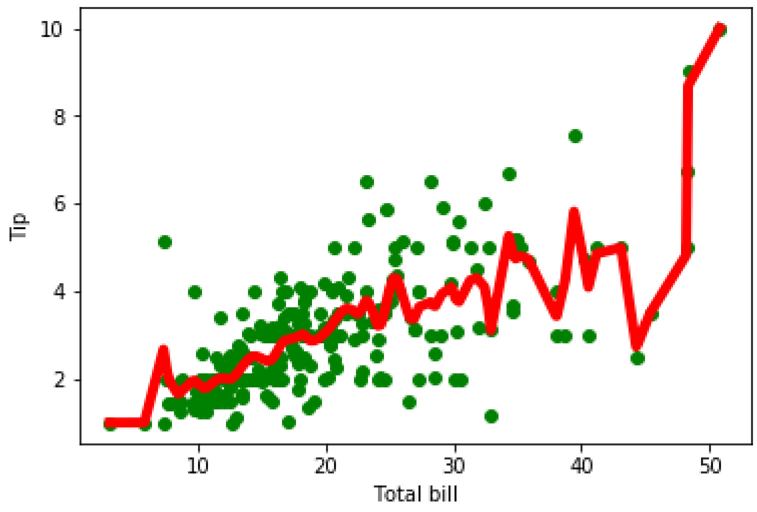
Ex. No 9	Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs

Aim: To implement the non-parametric Locally Weighted Regression algorithm in order to fit data points.

Program:

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

def kernel(point, xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np1.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):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W

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

# load data points
data = pd.read_csv('10-dataset.csv')    #Note: Give the exact address of the  data set file
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)

m= np.shape(mbill)[1]
one = np.mat(np1.ones(m))
X = np.hstack((one.T,mbill.T))

#set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
xsort = X[SortIndex][:,0]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
plt.show();
```

Output:



Dataset:

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	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	Female	No	Sun	Dinner	4
15.42	1.57	Male	No	Sun	Dinner	2
18.43	3	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	Male	No	Sat	Dinner	2
12.69	2	Male	No	Sat	Dinner	2
21.7	4.3	Male	No	Sat	Dinner	2
19.65	3	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	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	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	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	Male	No	Sun	Dinner	2
22.23	5	Male	No	Sun	Dinner	2
32.4	6	Male	No	Sun	Dinner	4
28.55	2.05	Male	No	Sun	Dinner	3
18.04	3	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	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	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	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	Female	No	Sat	Dinner	3
26.86	3.14	Female	Yes	Sat	Dinner	2

25.28	5	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	Male	No	Thur	Lunch	4
22.76	3	Male	No	Thur	Lunch	2
17.29	2.71	Male	No	Thur	Lunch	2
19.44	3	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	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	Male	No	Thur	Lunch	2
18.28	4	Male	No	Thur	Lunch	2
24.71	5.85	Male	No	Thur	Lunch	2
21.16	3	Male	No	Thur	Lunch	2
28.97	3	Male	Yes	Fri	Dinner	2
22.49	3.5	Male	No	Fri	Dinner	2
5.75	1	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	Male	Yes	Fri	Dinner	2
12.03	1.5	Male	Yes	Fri	Dinner	2
21.01	3	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	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	Female	Yes	Sat	Dinner	2
14	3	Male	No	Sat	Dinner	2
7.25	1	Female	No	Sat	Dinner	1
38.07	4	Male	No	Sun	Dinner	3
23.95	2.55	Male	No	Sun	Dinner	2
25.71	4	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	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	Female	No	Thur	Lunch	2
11.38	2	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	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	Female	No	Thur	Lunch	2
14.15	2	Female	No	Thur	Lunch	2
16	2	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	Male	No	Thur	Lunch	5
27.05	5	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	Male	No	Thur	Lunch	2
14.07	2.5	Male	No	Sun	Dinner	2
13.13	2	Male	No	Sun	Dinner	2
17.26	2.74	Male	No	Sun	Dinner	3
24.55	2	Male	No	Sun	Dinner	4
19.77	2	Male	No	Sun	Dinner	4
29.85	5.14	Female	No	Sun	Dinner	5
48.17	5	Male	No	Sun	Dinner	6
25	3.75	Female	No	Sun	Dinner	4
13.39	2.61	Female	No	Sun	Dinner	2
16.49	2	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	Female	No	Sun	Dinner	3
13.81	2	Male	No	Sun	Dinner	2
17.51	3	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	Female	Yes	Sat	Dinner	2
50.81	10	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	Male	Yes	Sun	Dinner	2
32.9	3.11	Male	Yes	Sun	Dinner	2
17.89	2	Male	Yes	Sun	Dinner	2
14.48	2	Male	Yes	Sun	Dinner	2
9.6	4	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	Male	Yes	Sun	Dinner	2
20.69	5	Male	No	Sun	Dinner	5
20.9	3.5	Female	Yes	Sun	Dinner	3
30.46	2	Male	Yes	Sun	Dinner	5
18.15	3.5	Female	Yes	Sun	Dinner	3
23.1	4	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	Male	Yes	Thur	Lunch	2
7.56	1.44	Male	No	Thur	Lunch	2
10.34	2	Male	Yes	Thur	Lunch	2
43.11	5	Female	Yes	Thur	Lunch	4
13	2	Female	Yes	Thur	Lunch	2
13.51	2	Male	Yes	Thur	Lunch	2
18.71	4	Male	Yes	Thur	Lunch	3
12.74	2.01	Female	Yes	Thur	Lunch	2
13	2	Female	Yes	Thur	Lunch	2
16.4	2.5	Female	Yes	Thur	Lunch	2
20.53	4	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	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	Male	Yes	Sat	Dinner	3
25.89	5.16	Male	Yes	Sat	Dinner	4
48.33	9	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	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	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	Female	Yes	Fri	Lunch	2
20.45	3	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	Male	Yes	Sat	Dinner	4
15.69	3	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	Male	Yes	Sat	Dinner	2
10.07	1.25	Male	No	Sat	Dinner	2
12.6	1	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	Female	Yes	Sat	Dinner	2
22.67	2	Male	Yes	Sat	Dinner	2
17.82	1.75	Male	No	Sat	Dinner	2
18.78	3	Female	No	Thur	Dinner	2