||JAI SRI GURUDEV||

# S J C INSTITUTE OF TECHNOLOGY

BB ROAD, CHICKKABALLAPUR 562101

( AFFLITATED TO VTU,BELAGAVI & APPROVED BY AICTE,NEW DLEHI.)

DEPARTMENT OF

## INFORMATION SCIENCE AND ENGINEERING

# ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY MANUAL (18CSL76)

PREPARED BY:

Mr.Chandra Shekar J.M
Assistant Professor
Department of ISE
SJCIT , CHICKKABALLAPUR

| | Jai Sri Gurudev | |
Sri Adichunchanagiri Shikshana Trust ®

# SJC INSTITUTE OF TECHNOLOGY

Chickballapur – 562 101

Estd: 1986

## Department of Information Science & Engineering

### VISION OF THE INSTITUTE

**Preparing Competent Engineering and Management Professional to Serve the Society**

### MISSION OF THE INSTITUTE

**M1: Providing Students with a Sound Knowledge in Fundamentals of their branch of Study**

**M2: Promoting Excellence in Teaching, Training, Research, and Consultancy**

**M3: Exposing Students to Emerging Frontiers in various domains enabling Continuous Learning**

**M4: Developing Entrepreneurial acumen to venture into Innovative areas**

**M5: Imparting Value-based Professional Education with a sense of Social Responsibility**

|| Jai Sri Gurudev ||

Sri Adichunchanagiri Shikshana Trust ®

# SJC INSTITUTE OF TECHNOLOGY

Chickballapur – 562 101

Estd: 1986

## Department of Information Science & Engineering

### VISION OF THE DEPARTMENT

**Educating Students to Engineer Information Science and Technology for Advancing the Knowledge as to best serve the Real-world.**

### MISSION OF THE DEPARTMENT

**M1: Focusing on Fundamentals and Applied Aspects in both Information Science Theory and Programming practices.**

**M2: Training comprehensively and encouraging R&D culture in trending areas of Information Technology.**

**M3: Collaborating with premier Institutes and Industries to nurture Innovation and learning, in cutting-edge Information Technology.**

**M4: Preparing the students who are much Sought-after, Productive and Well-respected for their work culture having Lifelong Learning practice.**

**M5: Promoting ethical and moral values among the students so as to enable them to emerge as responsible professionals.**

| | Jai Sri Gurudev | |
Sri Adichunchanagiri Shikshana Trust ®

# SJC INSTITUTE OF TECHNOLOGY

Chickballapur – 562 101

Estd: 1986

## Department of Information Science & Engineering

### PROGRAM EDUCATIONAL OBJECTIVES

**PEO1:** Engage in a Successful professional career in Information Science and Technology.

**PEO2:** Pursue higher studies and research to advance the knowledge for Solving Contemporary Problems in the IT industry.

**PEO3:** Adapt to a constantly changing world through Professional Development and Sustained Learning.

**PEO4:** Exhibit professionalism and teamwork with social concern.

**PEO5:** Develop Leadership and Entrepreneurship Skills by incorporating Organizational goals.

### PROGRAM-SPECIFIC OUTCOMES

**PSO1:** Apply the knowledge of data structures, database systems, system programming, networking web development, and AI & ML techniques in engineering the software.

**PSO2:** Exhibit solid foundations and advancements in developing software / hardware systems for solving contemporary problems.

|| Jai Sri Gurudev ||
Sri Adichunchanagiri Shikshana Trust ®

# SJC INSTITUTE OF TECHNOLOGY

Chickballapur – 562 101

Estd: 1986

**Department of Information Science & Engineering**

## ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING LABORATORY

**(Effective from the academic year 2018 -2019)**

| | |
|---|---|
| **SEMESTER – VII** | **Course Code 18CSL76** |
| **CIE Marks** 40 | **SEE Marks** 60 |
| **Number of Contact Hours/Week** 0:0:2 | **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):**

**The 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 algorithm to 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 to classify 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 the 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 the 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.

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

**Laboratory Course Outcomes**: The student should be able to:

Iimplement 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 the procedure to be made zero of the changed part only.
- Marks Distribution *(Coursed to change in accordance with university regulations)*
- For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
- For laboratories having PART A and PART B
  - i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 =40
  - ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

## Installing Anaconda on Windows

This tutorial will demonstrate how you can install Anaconda, a powerful package manager, on Microsoft Windows.

Anaconda is a package manager, an environment manager, and Python distribution that contains a collection of many open source packages. This is advantageous as when you are working on a data science project, you will find that you need many different packages (numpy, scikit-learn, scipy, pandas to name a few), which an installation of Anaconda comes preinstalled with. If you need additional packages after installing Anaconda, you can use Anaconda's package manager, conda, or pip to install those packages. This is highly advantageous as you don't have to manage dependencies between multiple packages yourself. Conda even makes it easy to switch between Python 2 and 3 (you can learn more here). In fact, an installation of Anaconda is also the recommended way to install Jupyter Notebooks which you can learn more about here on the DataCamp community.

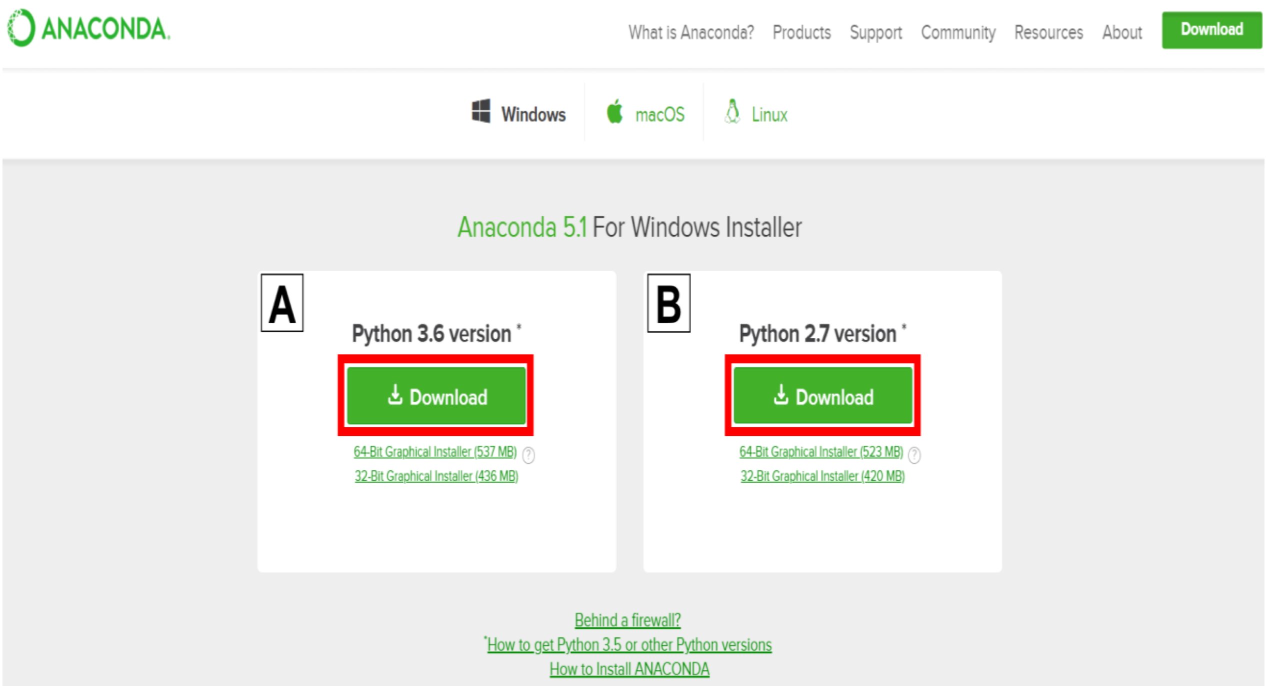
This tutorial will include:

- How to Install Anaconda on Windows
- How to test your installation and fix common installation issues
- What to do after installing Anaconda. With

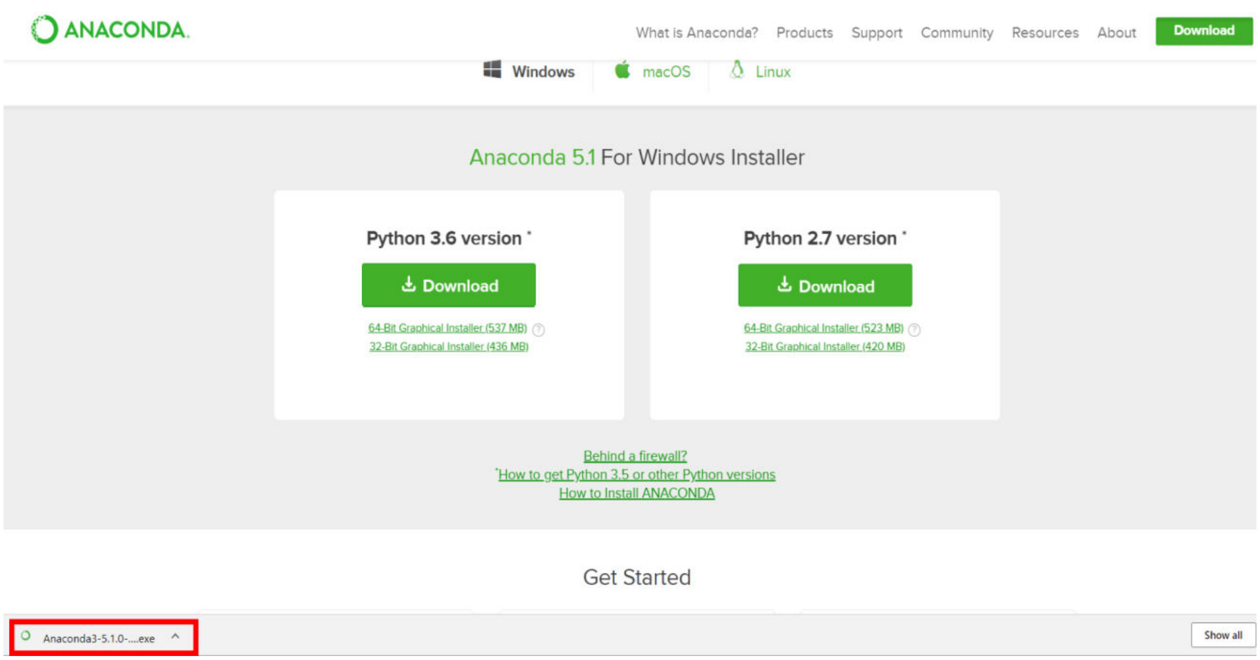
that, let's get started!

## Download and Install Anaconda

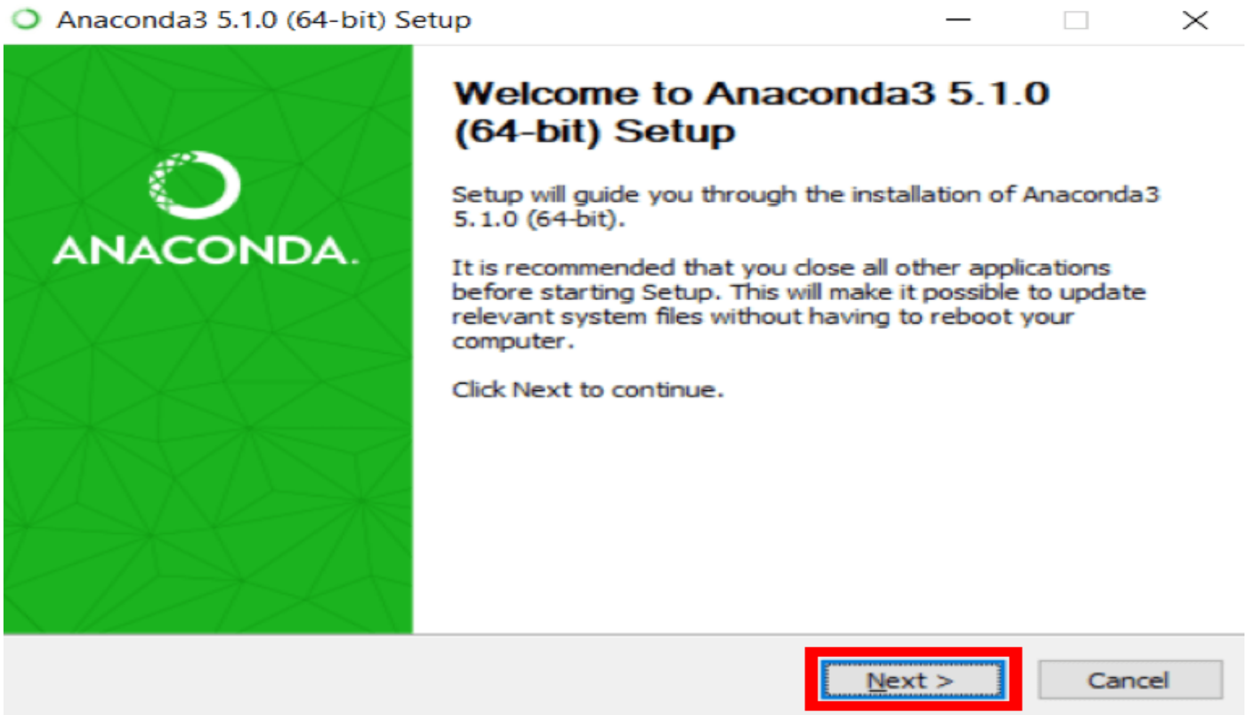
1. Go to the Anaconda Website and choose a Python 3.x graphical installer (A) or a Python 2.x graphical installer (B). If you aren't sure which Python version you want to install, choose Python 3. Do not choose both.



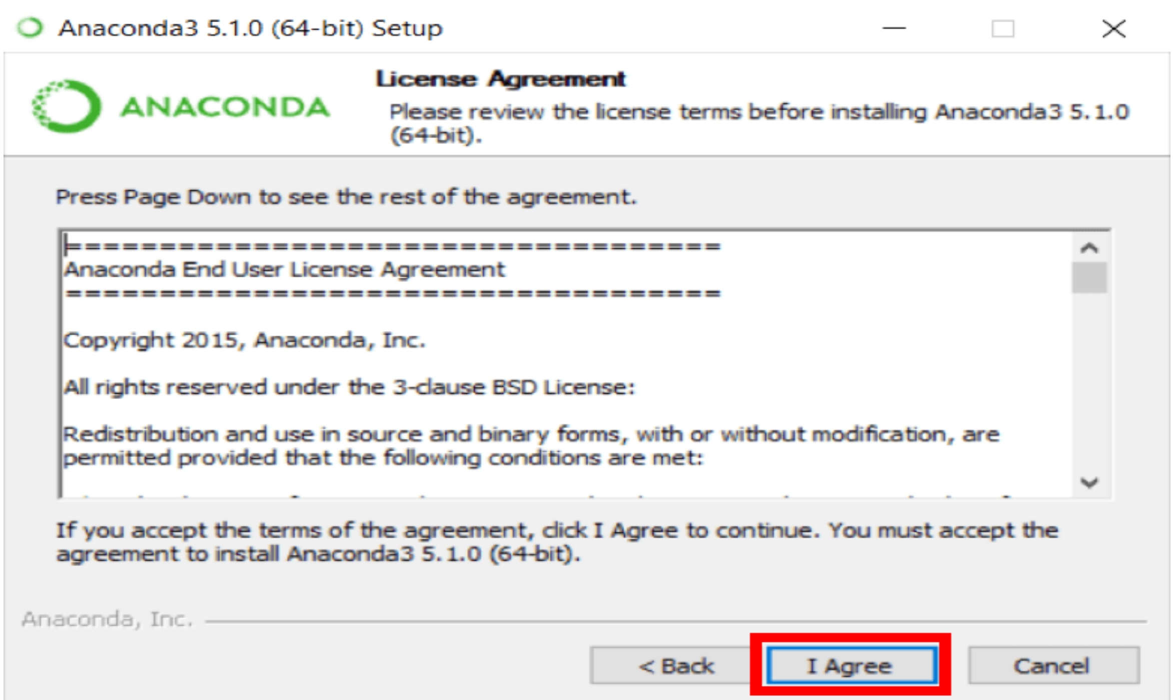

2. Locate your download and double click it.



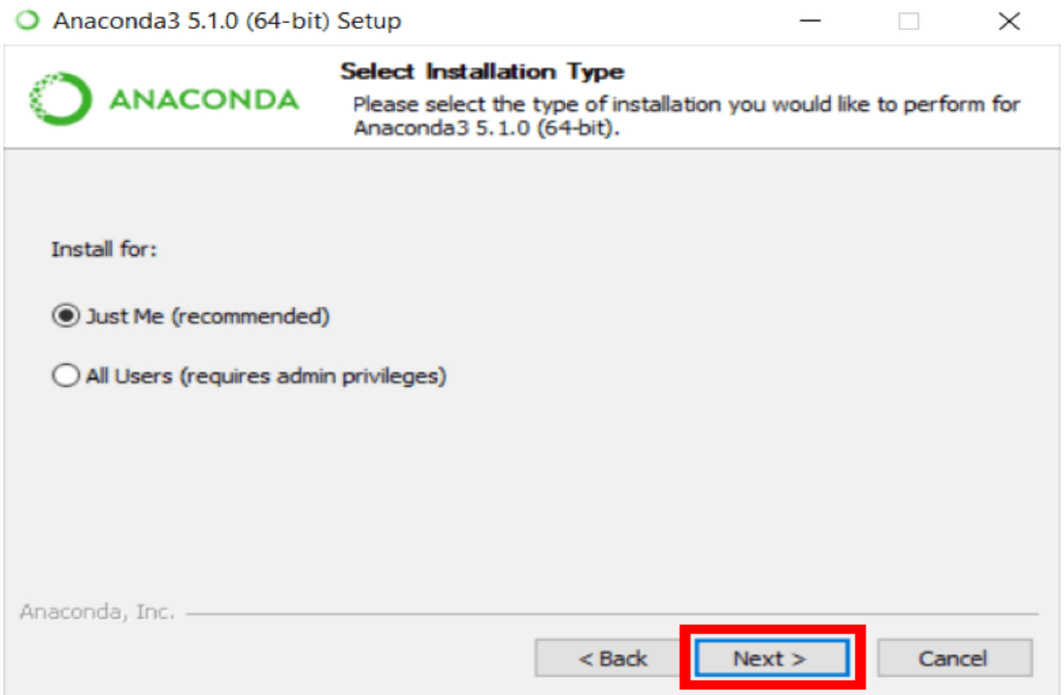

When the screen below appears, click on Next.



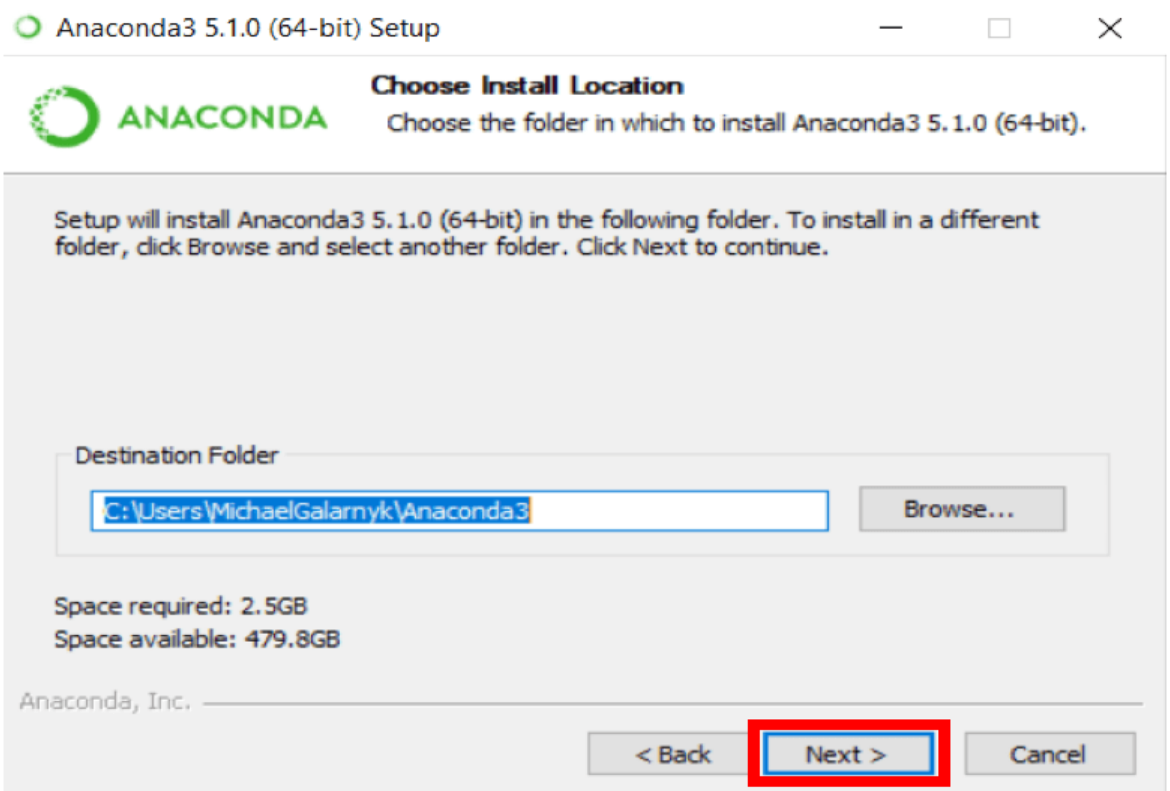

3. Read the license agreement and click on I Agree.



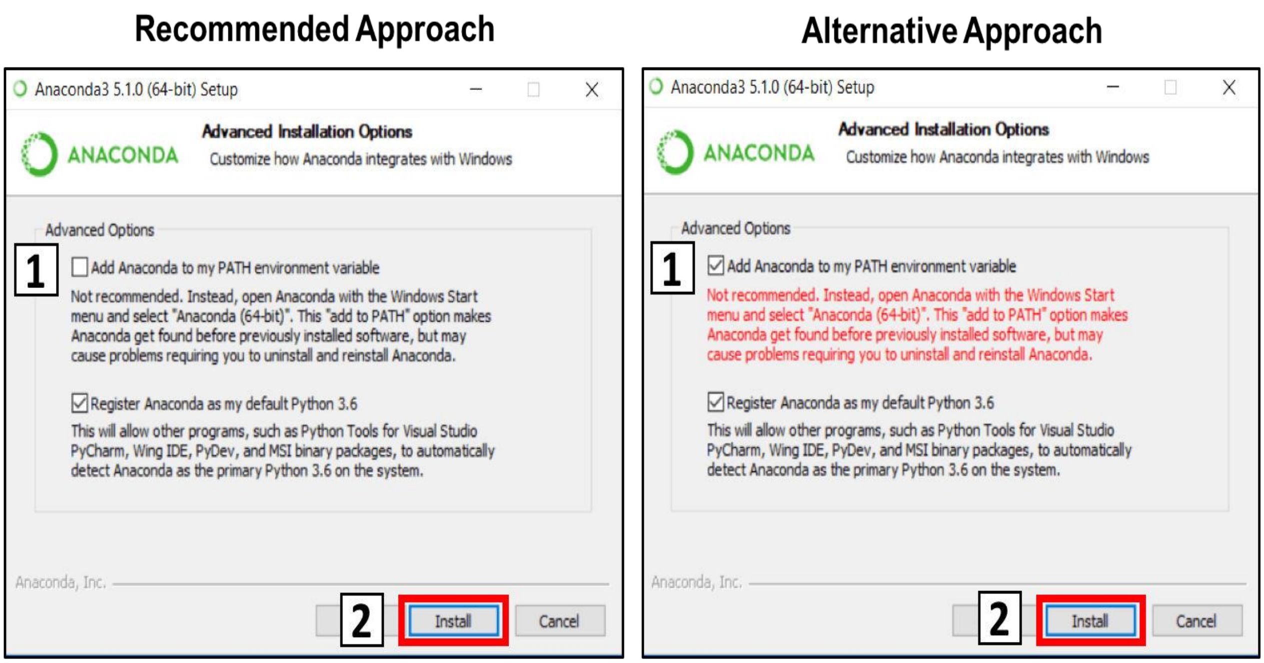
4. Click on Next.



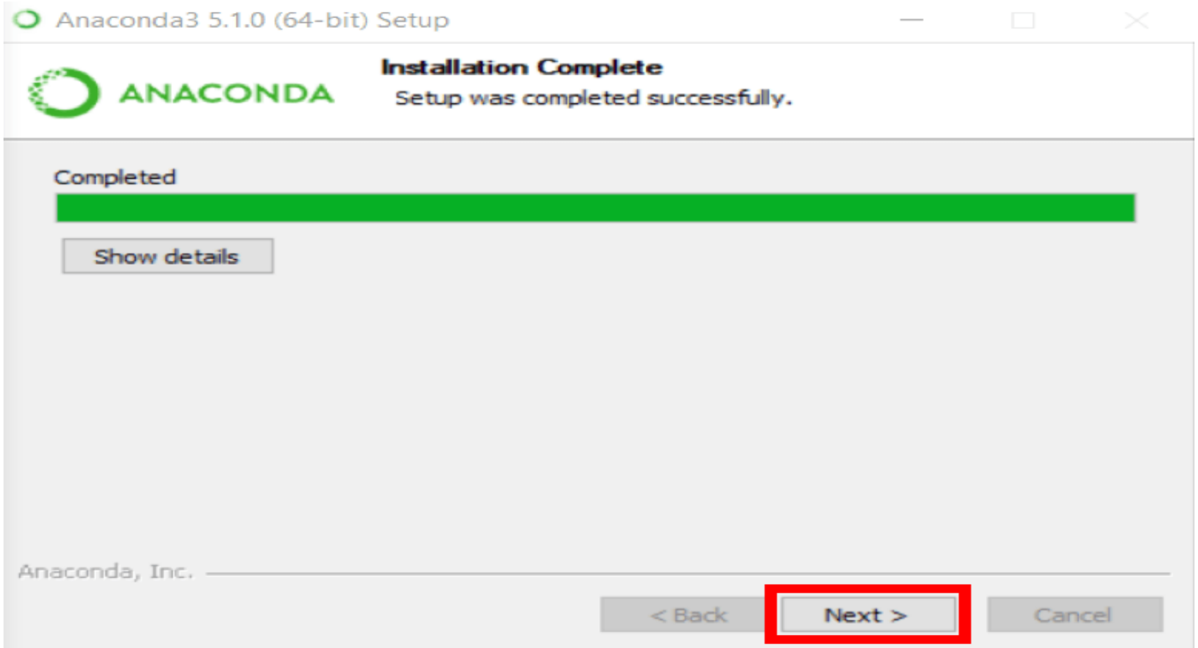

5. Note your installation location and then click Next.



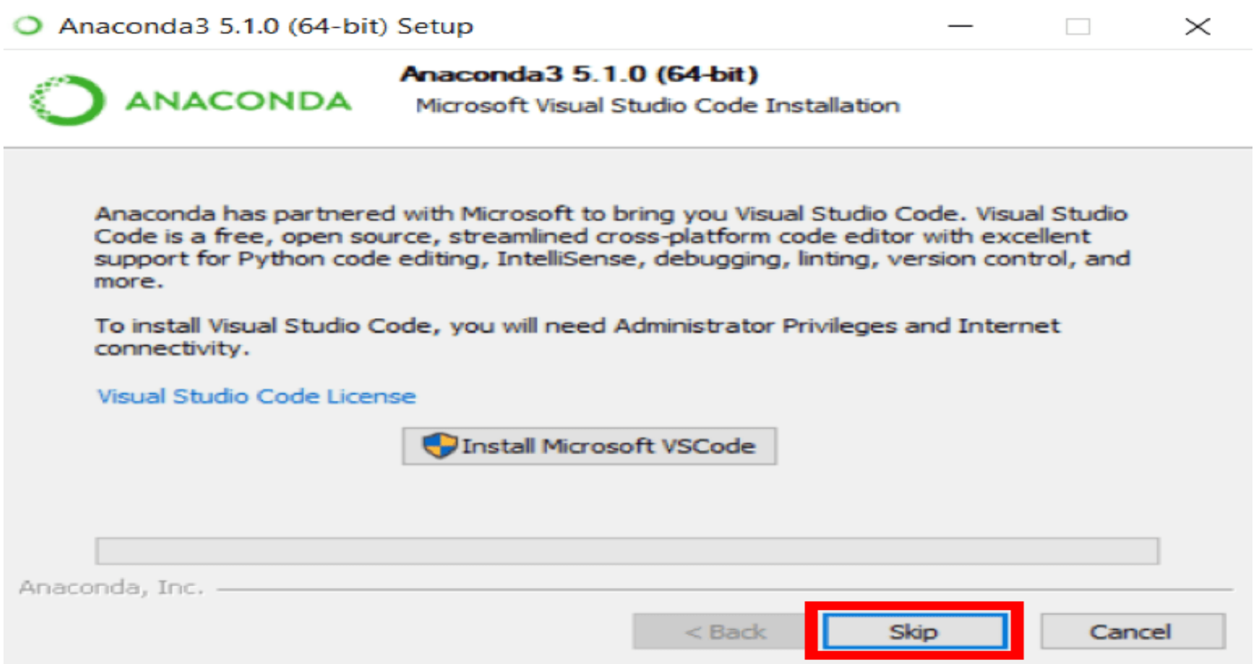
6. This is an important part of the installation process. The recommended approach is to not check the box to add Anaconda to your path. This means you will have to use Anaconda Navigator or the Anaconda Command Prompt (located in the Start Menu under "Anaconda") when you wish to use Anaconda (you can always add Anaconda to your PATH later if you don't check the box). If you want to be able to use Anaconda in your command prompt (or git bash, cmder, powershell etc), please use the alternative approach and check the box.



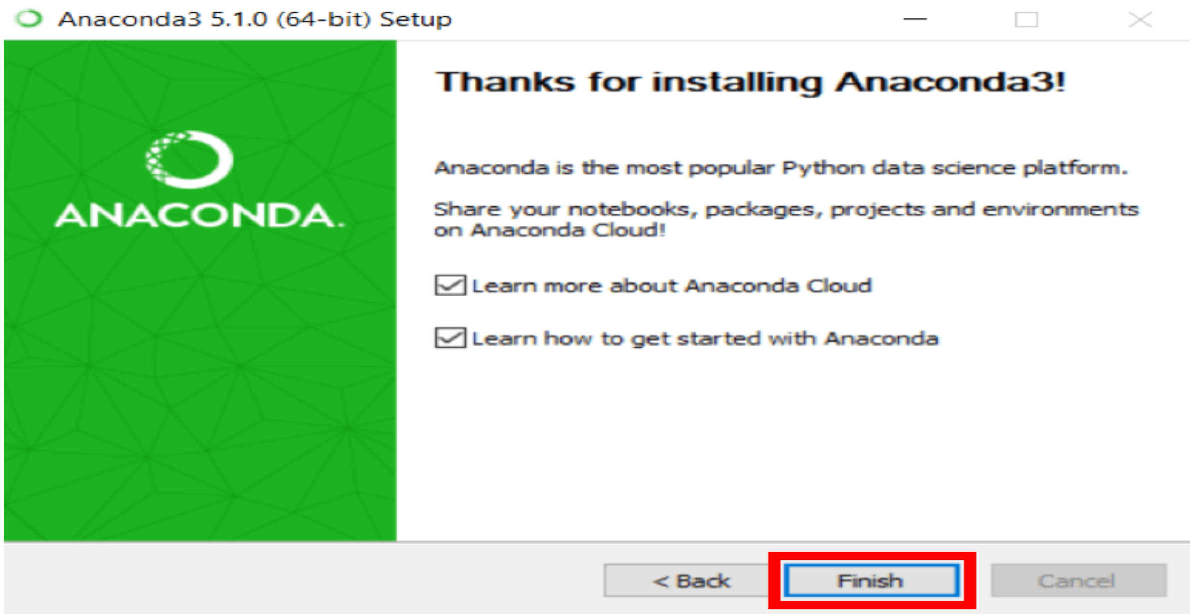

7. Click on Next.




8. you can install Microsoft VSCode if you wish, but it is optional.



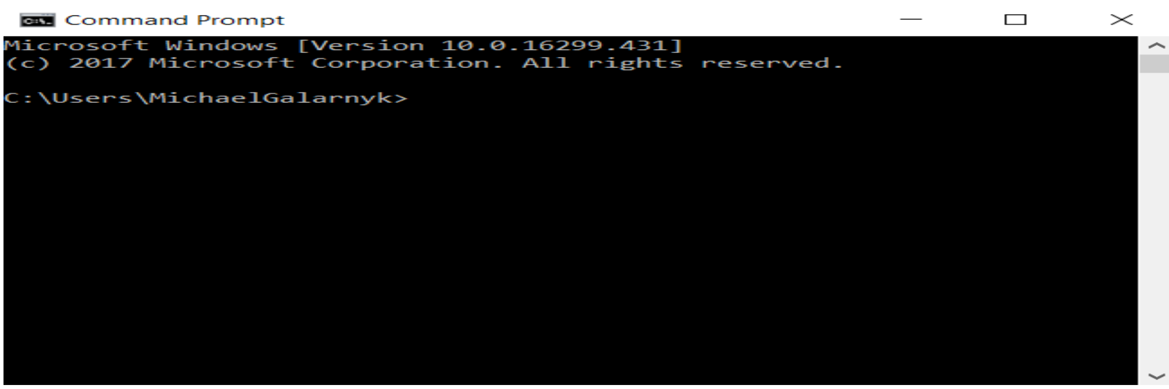

9. Click on Finish.




## Add Anaconda to Path (Optional)

This is an **optional** step. This is for the case where you didn't check the box in step 6 and now want to add Anaconda to your Path. The advantage of this is that you will be able to use Anaconda in your Command Prompt, Git Bash, cmder etc.

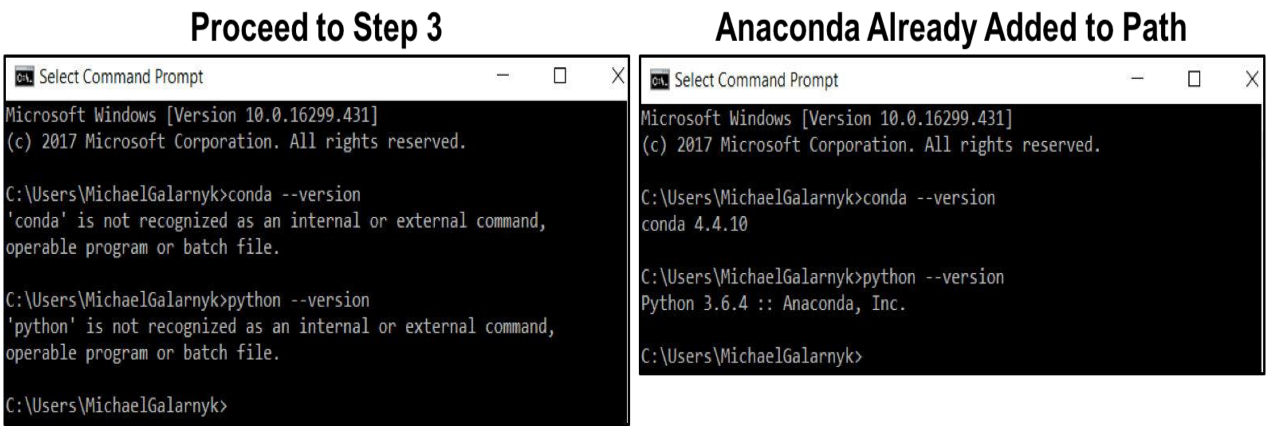
1. Open a Command Prompt.




2. Check if you already have Anaconda added to your path. Enter the commands below into your Command Prompt. This is checking if you already have Anaconda added to your path. If you get a command **not recognized** error like in the left side of the image below, proceed to step

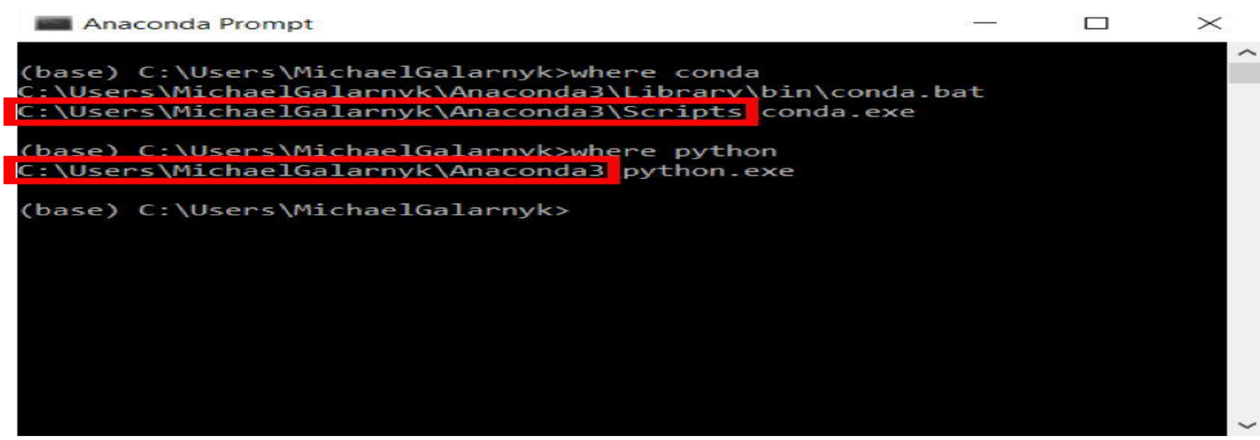
3. If you get an output similar to the right side of the image below, you have already added

Anaconda to your path. conda --
version
python --version



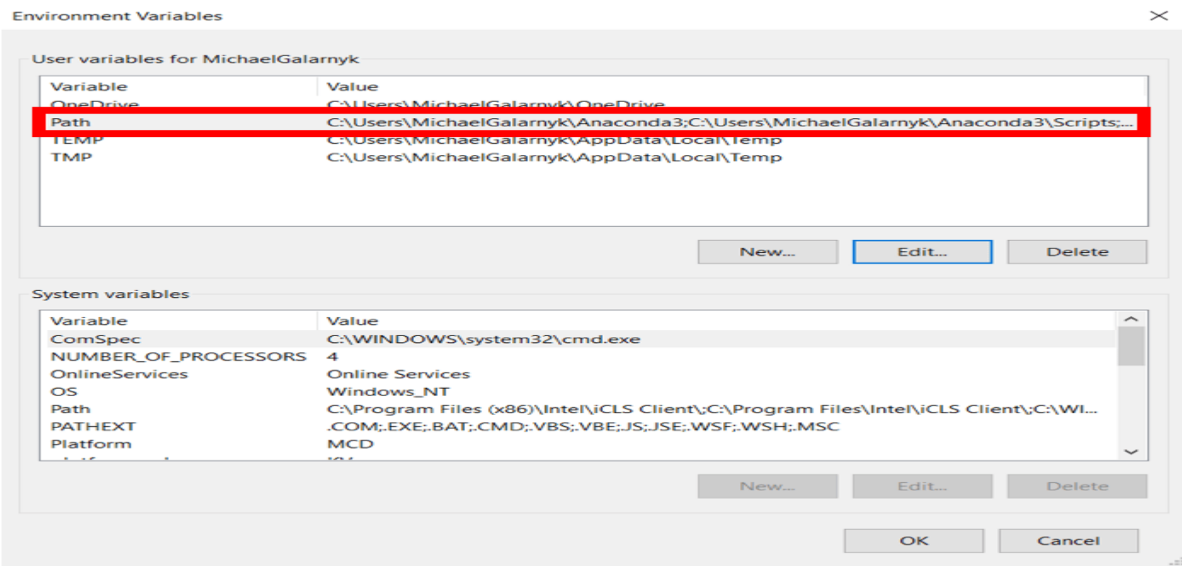

3. If you don't know where your conda and/or python is, open an **Anaconda Prompt** and type in the following commands. This is telling you where conda and python are located on your computer.

Where conda where
python

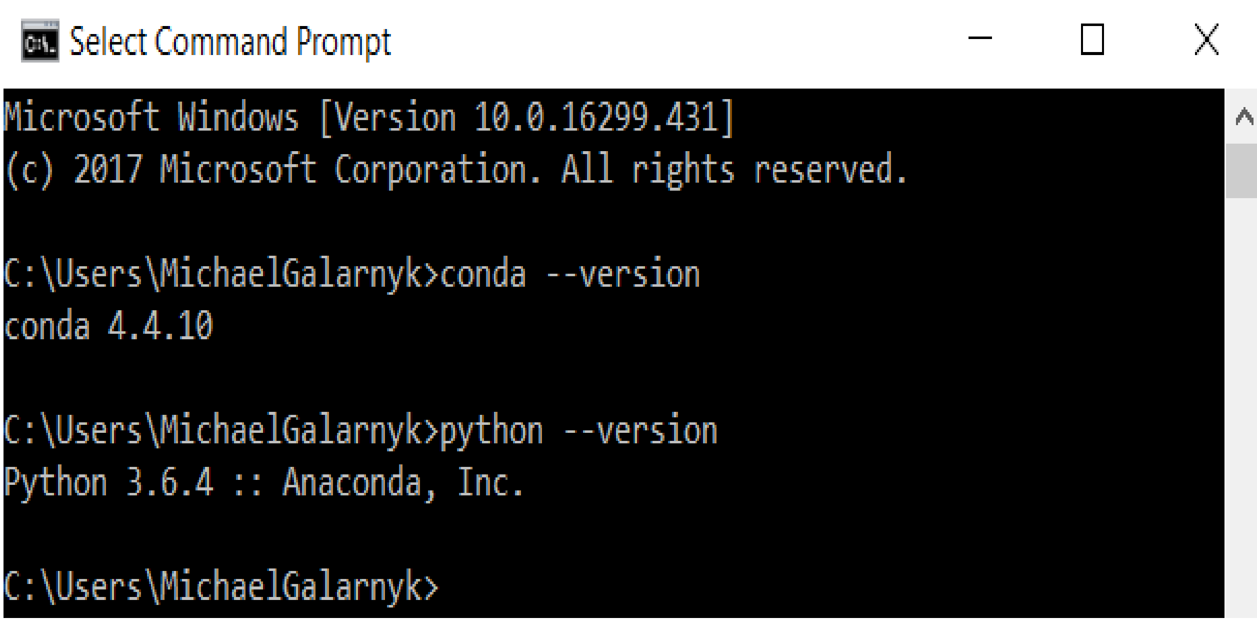



4. Add conda and python to your PATH. You can do this by going to your Environment Variables and adding the output of step 3 (enclosed in the red rectangle) to your path. If you are having issues,

here is a short video on adding conda and python to your PATH.




5. Open a **new Command Prompt**. Try typing conda --version and python --version into the **Command Prompt** to check to see if everything went well.



PROGRAM.NO.1

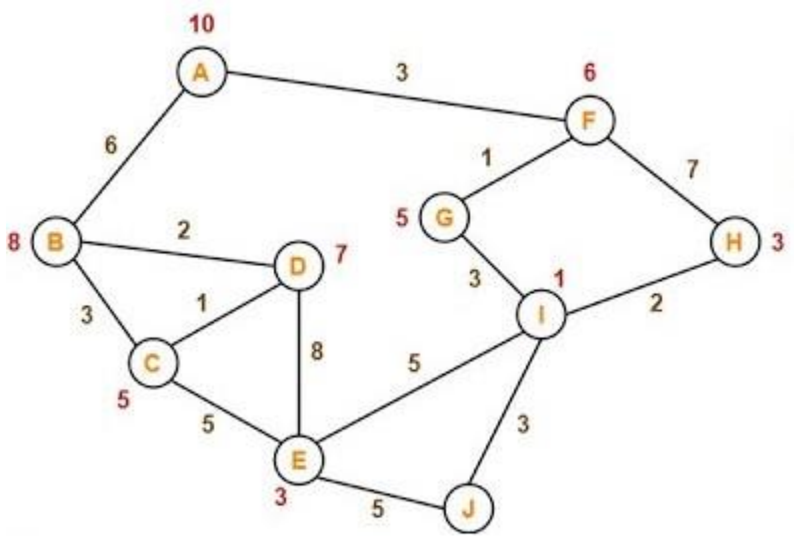
***Implement A* Search algorithm.***

```
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
            #for each node m,compare its distance from start i.e g(m) to the
            #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
```



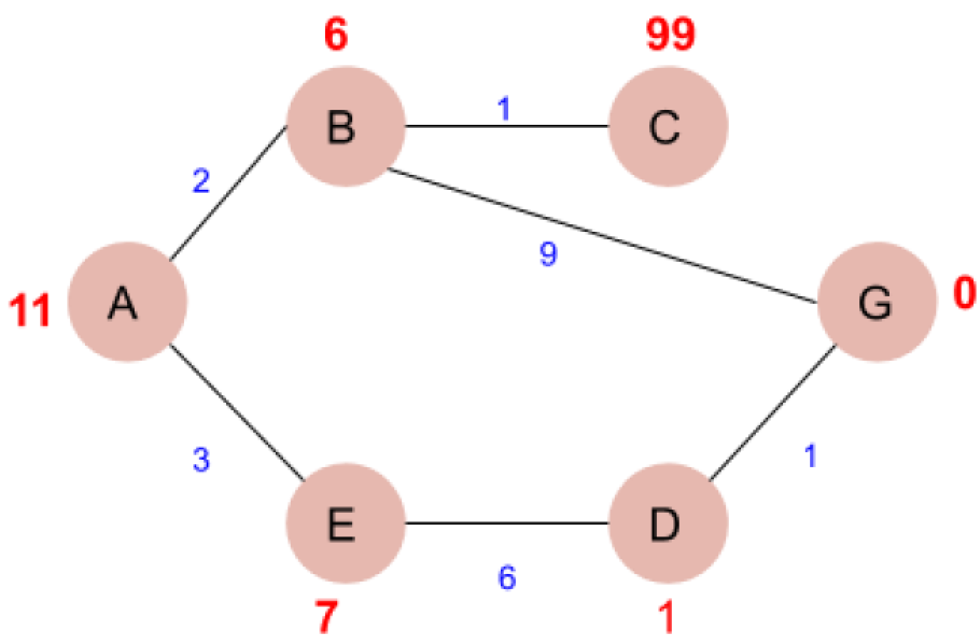

```
#for simplicity we ll consider heuristic distances given

#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')
```

Output:

Path found: ['A', 'F', 'G', 'I', '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')
```

Output:

Path found: ['A', 'E', 'D', 'G']

PROGRAM.NO.2

**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 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)
```

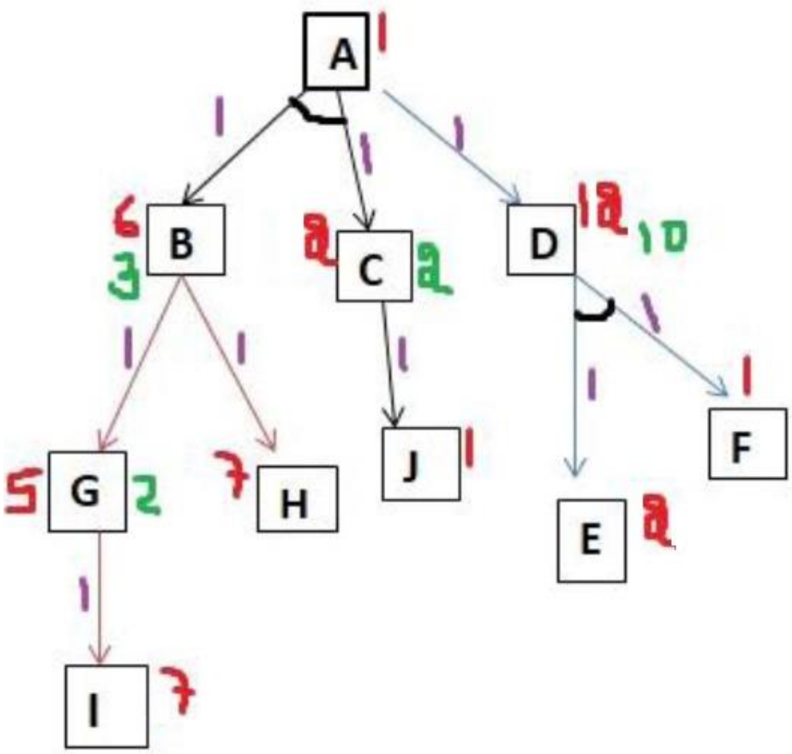
```
    def setStatus(self,v, val): # set the status of a given node

        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)

        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
```

Graph – 1 as Input to AO Star Search Algorithm




```
#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()
```

Output of AO Star Search Algorithm

**Graph – 1**

HEURISTIC VALUES : {‘A’: 1, ‘B’: 6, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 5, ‘H’: 7, ‘I’: 7, ‘J’: 1}
SOLUTION GRAPH : {}

PROCESSING NODE : A

10 [‘B’, ‘C’]
HEURISTIC VALUES : {‘A’: 10, ‘B’: 6, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 5, ‘H’: 7, ‘I’: 7, ‘J’: 1}
SOLUTION GRAPH : {}

PROCESSING NODE : B

6 [‘G’]
HEURISTIC VALUES : {‘A’: 10, ‘B’: 6, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 5, ‘H’: 7, ‘I’: 7, ‘J’: 1}
SOLUTION GRAPH : {}

PROCESSING NODE : A

10 [‘B’, ‘C’]
HEURISTIC VALUES : {‘A’: 10, ‘B’: 6, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 5, ‘H’: 7, ‘I’: 7, ‘J’: 1}
SOLUTION GRAPH : {}

PROCESSING NODE : G

8 [‘I’]
HEURISTIC VALUES : {‘A’: 10, ‘B’: 6, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 8, ‘H’: 7, ‘I’: 7, ‘J’: 1}
SOLUTION GRAPH : {}

PROCESSING NODE : B

8 [‘H’]
HEURISTIC VALUES : {‘A’: 10, ‘B’: 8, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 8, ‘H’: 7, ‘I’: 7, ‘J’: 1}
SOLUTION GRAPH : {}

PROCESSING NODE : A

12 [‘B’, ‘C’]
HEURISTIC VALUES : {‘A’: 12, ‘B’: 8, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 8, ‘H’: 7, ‘I’: 7, ‘J’: 1}
SOLUTION GRAPH : {}

PROCESSING NODE : I

0 []
HEURISTIC VALUES : {‘A’: 12, ‘B’: 8, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 8, ‘H’: 7, ‘I’: 0, ‘J’: 1}
SOLUTION GRAPH : {‘I’: []}

PROCESSING NODE : G

1 [‘I’]
HEURISTIC VALUES : {‘A’: 12, ‘B’: 8, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 1, ‘H’: 7, ‘I’: 0, ‘J’: 1}
SOLUTION GRAPH : {‘I’: [], ‘G’: [‘I’]}

PROCESSING NODE : B

2 [‘G’]
HEURISTIC VALUES : {‘A’: 12, ‘B’: 2, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 1, ‘H’: 7, ‘I’: 0, ‘J’: 1}
SOLUTION GRAPH : {‘I’: [], ‘G’: [‘I’], ‘B’: [‘G’]}

PROCESSING NODE : A

6 [‘B’, ‘C’]
HEURISTIC VALUES : {‘A’: 6, ‘B’: 2, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 1, ‘H’: 7, ‘I’: 0, ‘J’: 1}
SOLUTION GRAPH : {‘I’: [], ‘G’: [‘I’], ‘B’: [‘G’]}

PROCESSING NODE : C

2 [‘J’]
HEURISTIC VALUES : {‘A’: 6, ‘B’: 2, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 1, ‘H’: 7, ‘I’: 0, ‘J’: 1}
SOLUTION GRAPH : {‘I’: [], ‘G’: [‘I’], ‘B’: [‘G’]}

PROCESSING NODE : A

6 [‘B’, ‘C’]
HEURISTIC VALUES : {‘A’: 6, ‘B’: 2, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 1, ‘H’: 7, ‘I’: 0, ‘J’: 1}
SOLUTION GRAPH : {‘I’: [], ‘G’: [‘I’], ‘B’: [‘G’]}

PROCESSING NODE : J

0 []
HEURISTIC VALUES : {‘A’: 6, ‘B’: 2, ‘C’: 2, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 1, ‘H’: 7, ‘I’: 0, ‘J’: 0}
SOLUTION GRAPH : {‘I’: [], ‘G’: [‘I’], ‘B’: [‘G’], ‘J’: []}

PROCESSING NODE : C

1 [‘J’]
HEURISTIC VALUES : {‘A’: 6, ‘B’: 2, ‘C’: 1, ‘D’: 12, ‘E’: 2, ‘F’: 1, ‘G’: 1, ‘H’: 7, ‘I’: 0, ‘J’: 0}
SOLUTION GRAPH : {‘I’: [], ‘G’: [‘I’], ‘B’: [‘G’], ‘J’: [], ‘C’: [‘J’]}

PROCESSING NODE : A

5 [‘B’, ‘C’]

FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START NODE: A

{‘I’: [], ‘G’: [‘I’], ‘B’: [‘G’], ‘J’: [], ‘C’: [‘J’], ‘A’: [‘B’, ‘C’]}

**PROGRAM-3**

**3. 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.**

## Candidate Elimination Algorithm Machine Learning

For each training example d, do:

If d is positive example

Remove from G any hypothesis h inconsistent with d

For each hypothesis s in S not consistent with d:

Remove s from S

Add to S all minimal generalizations of s consistent with d and having a generalization in G

Remove from S any hypothesis with a more specific h in S

If d is negative example

Remove from S any hypothesis h inconsistent with d

For each hypothesis g in G not consistent with d:

Remove g from G

Add to G all minimal specializations of g consistent with d and having a specialization in S

Remove from G any hypothesis having a more general hypothesis in G

**Program:**

```
import numpy as np
```

```
import pandas as pd



data = pd.read_csv(path+'/enjoysport.csv')

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

print("\nInstances are:\n",concepts)

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

print("\nTarget Values are: ",target)



def learn(concepts, target):

    specific_h = concepts[0].copy()

    print("\nInitialization of specific_h and genearal_h")

    print("\nSpecific Boundary: ", specific_h)

    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]

    print("\nGeneric Boundary: ",general_h)



    for i, h in enumerate(concepts):

        print("\nInstance", i+1 , "is ", h)

        if target[i] == "yes":

            print("Instance is Positive ")

            for x in range(len(specific_h)):
```

```
                if h[x]!= specific_h[x]:

                    specific_h[x] ='?'

                    general_h[x][x] ='?'



        if target[i] == "no":

            print("Instance is Negative ")

            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] = '?'



        print("Specific Bundary after ", i+1, "Instance is ", specific_h)

        print("Generic Boundary after ", i+1, "Instance is ", general_h)

        print("\n")



    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]

    for i in indices:

        general_h.remove(['?', '?', '?', '?', '?', '?'])

    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")
```

Dataset:

EnjoySport Dataset is saved as .csv (comma separated values) file in the current working directory ot

| Outlook | 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 |

Output:

Instances are:
[[‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’]
[‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’]
[‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’]]

Target Values are: [‘yes’ ‘yes’ ‘no’ ‘yes’]

Initialization of specific_h and genearal_h

Specific Boundary: [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary: [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

otherwise use the complete path of the dataset set in the program:

Instance 1 is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’] Instance is Positive

Specific Bundary after 1 Instance is [‘sunny’ ‘warm’ ‘normal’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary after 1 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 2 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘warm’ ‘same’] Instance is Positive

Specific Bundary after 2 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary after 2 Instance is [[‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Instance 3 is [‘rainy’ ‘cold’ ‘high’ ‘strong’ ‘warm’ ‘change’] Instance is Negative

Specific Bundary after 3 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘warm’ ‘same’]

Generic Boundary after 3 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘same’]]

Instance 4 is [‘sunny’ ‘warm’ ‘high’ ‘strong’ ‘cool’ ‘change’] Instance is Positive

Specific Bundary after 4 Instance is [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]

Generic Boundary after 4 Instance is [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’]]

Final Specific_h: [‘sunny’ ‘warm’ ‘?’ ‘strong’ ‘?’ ‘?’]

Final General_h: [[‘sunny’, ‘?’, ‘?’, ‘?’, ‘?’, ‘?’], [‘?’, ‘warm’, ‘?’, ‘?’, ‘?’, ‘?’]]

**PROGRAM-4**

**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 applying this knowledge to classify a new sample.**

<u>Decision Tree ID3 Algorithm Machine Learning</u>

ID3(Examples, Target_attribute, Attributes)

Examples are the training examples.

Target_attribute is the attribute whose value is to be predicted by the tree.

Attributes is a list of other attributes that may be tested by the learned decision tree.

Returns a decision tree that correctly classifies the given Examples.

Create a Root node for the tree

If all Examples are positive, Return the single-node tree Root, with label = +

If all Examples are negative, Return the single-node tree Root, with label = -

If Attributes is empty, Return the single-node tree Root,

with label = most common value of Target_attribute in Examples

Otherwise Begin

A ← the attribute from Attributes that best* classifies Examples

The decision attribute for Root ← A

For each possible value, vi, of A,

Add a new tree branch below Root, corresponding to the test A = vi

Let Examples vi, be the subset of Examples that have value vi for A

If Examples vi , is empty

Then below this new branch add a leaf node with

label = most common value of Target_attribute in Examples

Else

below this new branch add the subtree

ID3(Examples vi, Targe_tattribute, Attributes – {A}))

End

Return Root

The best attribute is the one with the highest information gain

**ENTROPY:**

Entropy measures the impurity of a collection of examples.

$$Entropy\ (S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

Where, $p_+$ is the proportion of positive examples in S

$p_-$ is the proportion of negative examples in S.

**INFORMATION GAIN:**

***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.

The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Dataset:

PlayTennis Dataset is saved as .csv (comma separated values) file in the current working directory otherwise use the complete path of the dataset set in the program:

| **Day** | ***Outlook*** | ***Temperature*** | ***Humidity*** | ***Wind*** | ***PlayTennis*** |
|---|---|---|---|---|---|
| **D1** | Sunny | Hot | High | Weak | No |
| **D2** | Sunny | Hot | High | Strong | No |
| **D3** | Overcast | Hot | High | Weak | Yes |

| | | | | | |
|---|---|---|---|---|---|
| **D4** | Rain | Mild | High | Weak | Yes |
| **D5** | Rain | Cool | Normal | Weak | Yes |
| **D6** | Rain | Cool | Normal | Strong | No |
| **D7** | Overcast | Cool | Normal | Strong | Yes |
| **D8** | Sunny | Mild | High | Weak | No |
| **D9** | Sunny | Cool | Normal | Weak | Yes |
| **D10** | Rain | Mild | Normal | Weak | Yes |
| **D11** | Sunny | Mild | Normal | Strong | Yes |
| **D12** | Overcast | Mild | High | Strong | Yes |
| **D13** | Overcast | Hot | Normal | Weak | Yes |
| **D14** | Rain | Mild | High | Strong | No |

**Python Program**

```
import pandas as pd

import math

import numpy as np



data = pd.read_csv("3-dataset.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:

```
Outlook
 rain
      Wind
              strong
                     no
              weak
                     yes
 overcast
      yes

 sunny
      Humidity
              normal
                     yes
              high
                     no
```

**PROGRAM-5**

**5. Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.**

```
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)
```

Training Examples:

| **Example** | **Sleep** | **Study** | **Expected % in Exams** |
|---|---|---|---|
| **1** | 2 | 9 | 92 |
| **2** | 1 | 5 | 86 |
| **3** | 3 | 6 | 89 |

Normalize the input

| **Example** | **Sleep** | **Study** | **Expected % in Exams** |
|---|---|---|---|
| **1** | 2/3 = 0.66666667 | 9/9 = 1 | 0.92 |
| **2** | 1/3 = 0.33333333 | 5/9 = 0.55555556 | 0.86 |
| **3** | 3/3 = 1 | 6/9 = 0.66666667 | 0.89 |

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:
[[0.81951208]
[0.8007242 ]
[0.82485744]]
————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:
[[0.82033938]
[0.80153634]
[0.82568134]]
————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:
[[0.82115226]
[0.80233463]
[0.82649072]]
————Epoch- 3 Ends———-

**PROGRAM-6:**

**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.**

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Where,
**P(h|D)** is the probability of hypothesis h given the data D. This is called the **posterior probability**.
**P(D|h)** is the probability of data d given that the hypothesis h was true.
**P(h)** is the probability of hypothesis h being true. This is called the **prior probability of h. P(D)** is the probability of the data. This is called the **prior probability of D**
After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.
Bayes theorem to calculate the posterior probability of each candidate hypothesis is ***hMAP*** is a MAP hypothesis provided.

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Ignoring P(D) since it is a constant)
Gaussian Naive Bayes
A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution
Representation for Gaussian Naive Bayes
We calculate the probabilities for input values for each class using a frequency. With real- valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.
This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.
Gaussian Naive Bayes Model from Data
The probability density function for the normal distribution is defined by two parameters (mean and standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$ Mean

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2\right]^{0.5}$$ Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$ Normal distribution

***<u>Examples:</u>***

The data set used in this program is the ***Pima Indians Diabetes problem***.
This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
The attributes are Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome
Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)
Sample Examples:

| Examples | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabeticPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 148 | 72 | 35 | | 33.6 | 0.627 | 50 | 1 |
| 2 | 1 | 85 | 66 | 29 | | 26.6 | 0.351 | 31 | |
| 3 | 8 | 183 | 64 | | | 23.3 | 0.672 | 32 | 1 |
| 4 | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | |
| 5 | | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |
| 6 | 5 | 116 | 74 | | | 25.6 | 0.201 | 30 | |
| 7 | 3 | 78 | 50 | 32 | 88 | 31 | 0.248 | 26 | 1 |
| 8 | 10 | 115 | | | | 35.3 | 0.134 | 29 | |
| 9 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 | 1 |
| 10 | 8 | 125 | 96 | | | | 0.232 | 54 | 1 |

**Python Program**

```
import csv
import random
import math

def loadcsv(filename):
	lines = csv.reader(open(filename, "r"));
	dataset = list(lines)
```

```
        for i in range(len(dataset)):
    #converting strings into numbers for processing
                dataset[i] = [float(x) for x in dataset[i]]

        return dataset

def splitdataset(dataset, splitratio):
  #67% training size
        trainsize = int(len(dataset) * splitratio);
        trainset = []
        copy = list(dataset);
        while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for training data
                index = random.randrange(len(copy));
                trainset.append(copy.pop(index))
        return [trainset, copy]

def separatebyclass(dataset):
        separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
        for i in range(len(dataset)):
                vector = dataset[i]
                if (vector[-1] not in separated):
                        separated[vector[-1]] = []
                separated[vector[-1]].append(vector)
        return separated

def mean(numbers):
        return sum(numbers)/float(len(numbers))

def stdev(numbers):
        avg = mean(numbers)
        variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
        return math.sqrt(variance)

def summarize(dataset): #creates a dictionary of classes
        summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)];
        del summaries[-1] #excluding labels +ve or -ve
        return summaries

def summarizebyclass(dataset):
        separated = separatebyclass(dataset);
  #print(separated)
        summaries = {}
        for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
                summaries[classvalue] = summarize(instances) #summarize is used to cal to mean and
std
        return summaries

def calculateprobability(x, mean, stdev):
        exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
```

```
        return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

def calculateclassprobabilities(summaries, inputvector):
        probabilities = {} # probabilities contains the all prob of all class of test data
        for classvalue, classsummaries in summaries.items():#class and attribute information as mean
and sd
                probabilities[classvalue] = 1
                for i in range(len(classsummaries)):
                        mean, stdev = classsummaries[i] #take mean and sd of every attribute for
class 0 and 1 seperaely
                        x = inputvector[i] #testvector's first attribute
                        probabilities[classvalue] *= calculateprobability(x, mean, stdev);#use normal
dist
        return probabilities

def predict(summaries, inputvector): #training and test data is passed
        probabilities = calculateclassprobabilities(summaries, inputvector)
        bestLabel, bestProb = None, -1
        for classvalue, probability in probabilities.items():#assigns that class which has he highest prob
                if bestLabel is None or probability > bestProb:
                        bestProb = probability
                        bestLabel = classvalue
        return bestLabel

def getpredictions(summaries, testset):
        predictions = []
        for i in range(len(testset)):
                result = predict(summaries, testset[i])
                predictions.append(result)
        return predictions

def getaccuracy(testset, predictions):
        correct = 0
        for i in range(len(testset)):
                if testset[i][-1] == predictions[i]:
                        correct += 1
        return (correct/float(len(testset))) * 100.0

def main():
        filename = 'naivedata.csv'
        splitratio = 0.67
        dataset = loadcsv(filename);

        trainingset, testset = splitdataset(dataset, splitratio)
        print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingset),
len(testset)))
        # prepare model
        summaries = summarizebyclass(trainingset);
        #print(summaries)
    # test model
        predictions = getpredictions(summaries, testset) #find the predictions of test data with the
training data
        accuracy = getaccuracy(testset, predictions)
        print('Accuracy of the classifier is : {0}%'.format(accuracy))
```

main()
Output
Split 768 rows into train=514 and test=254
Rows Accuracy of the classifier is : 71.65354330708661%

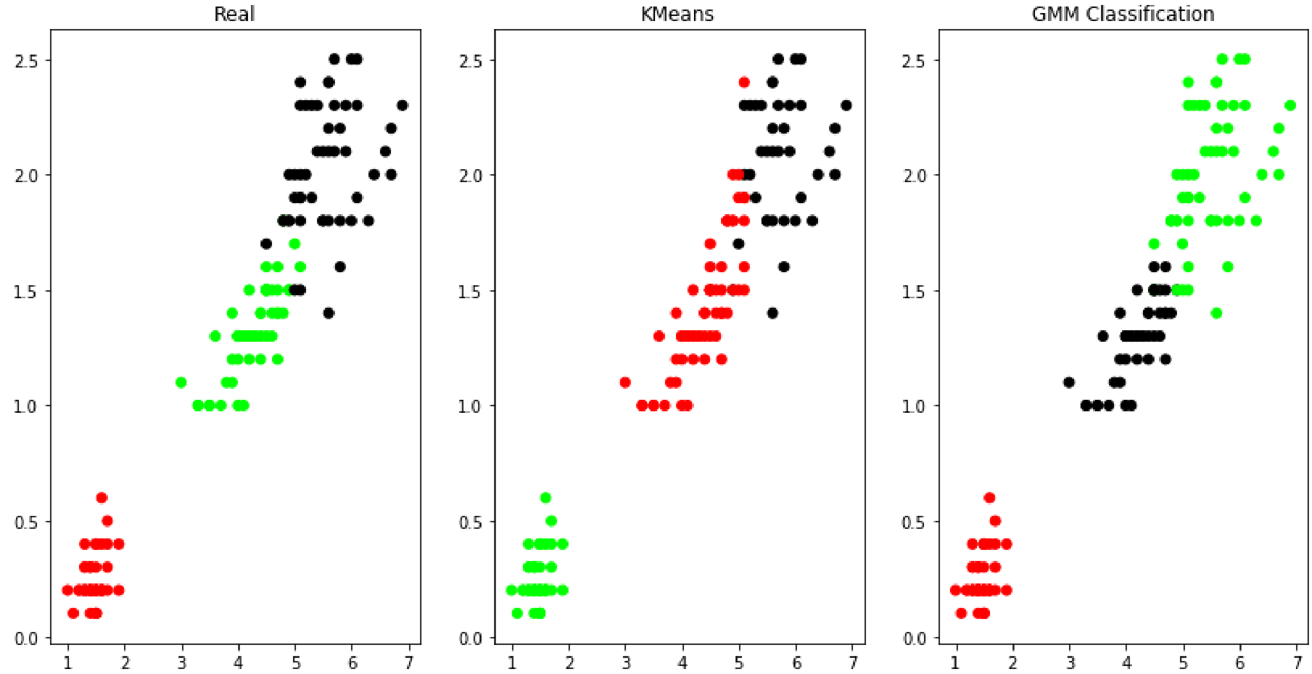
**PROGRAM-7:**

**7._Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using the 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 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)

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: 0.3666666666666664
The Confusion matrix of EM:
[[50 0 0]
[ 0 5 45]
[ 0 50 0]]



**PROGRAM-8:**

**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.**

## K-Nearest Neighbor Algorithm

Training algorithm:

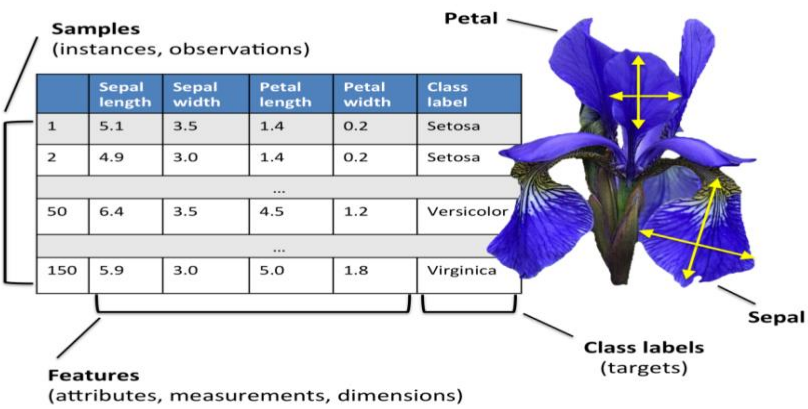
- For each training example (x, f (x)), add the example to the list training examples Classification algorithm:
    - Given a query instance $x_q$ to be classified,
        - Let $x_1 \ldots x_k$ denote the k instances from training examples that are nearest to $x_q$
        - Return

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

- Where, $f(x_i)$ function to calculate the mean value of the k nearest training examples.

Data Set:

Iris Plants Dataset: Dataset contains 150 instances (50 in each of three classes) Number of Attributes: 4 numeric, predictive attributes and the Class.

Sample Data

| | sepal-length | sepal-width | petal-length | petal-width | Class |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

Python 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)
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

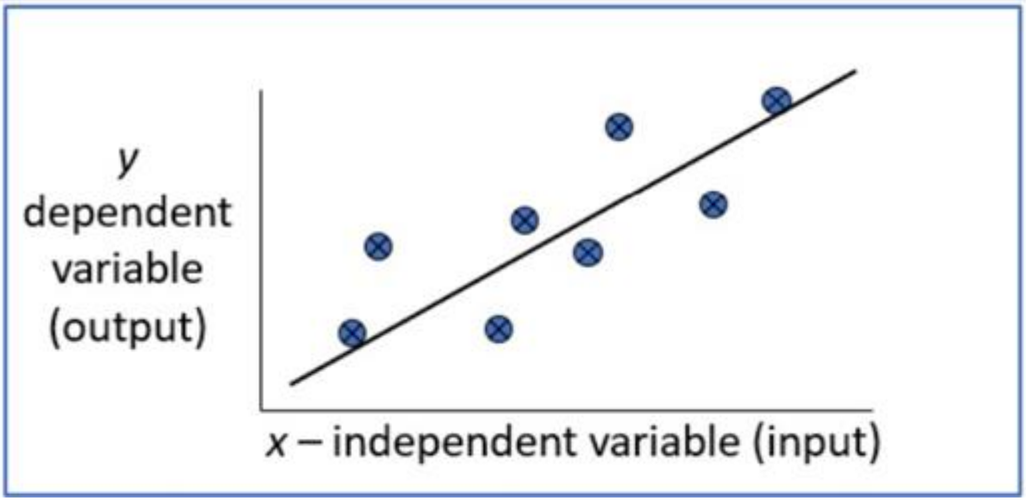
```
   sepal-length  sepal-width  petal-length  petal-width
0           5.1          3.5           1.4          0.2
1           4.9          3.0           1.4          0.2
2           4.7          3.2           1.3          0.2
3           4.6          3.1           1.5          0.2
4           5.0          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
-----------------------------------------------------------------------

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 classifer is 0.87
-----------------------------------------------------------------------
```

**PROGRAM-9:**

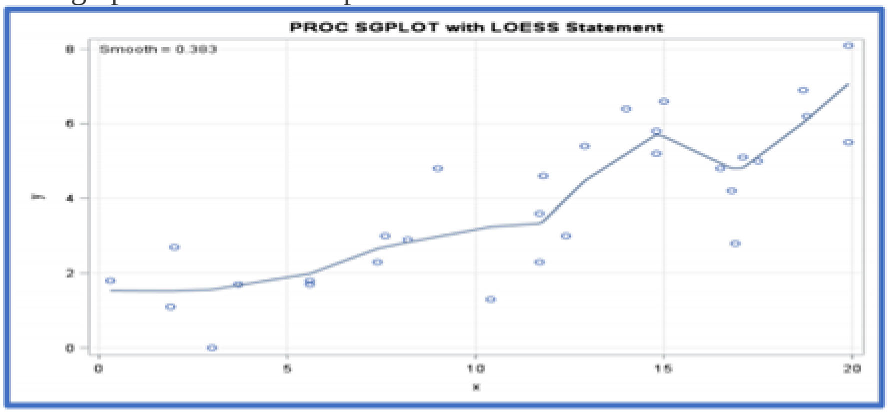
**9. Implement the non-parametric Locally Weighted Regression algorithm in Python in order to fit data points. Select the appropriate data set for your experiment and draw graphs.**

Locally Weighted Regression Algorithm
Regression:

- Regression is a technique from statistics that are used to predict values of the desired target quantity when the target quantity is continuous.
  - In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
    - y is called the dependent variable.
    - x is called the independent variable.




Loess/Lowess Regression:
Loess regression is a nonparametric technique that uses local weighted regression to fit a smooth curve through points in a scatter plot.




Lowess Algorithm:
Locally weighted regression is a very powerful nonparametric model used in statistical learning.
Given a dataset X, y, we attempt to find a model parameter β(x) that minimizes residual sum of weighted squared errors.
The weights are given by a kernel function (k or w) which can be chosen arbitrarily
Algorithm
1. Read the Given data Sample to X and the curve (linear or non linear) to Y
2. Set the value for Smoothening parameter or Free parameter say τ
3. Set the bias /Point of interest set x0 which is a subset of X
4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using:

$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

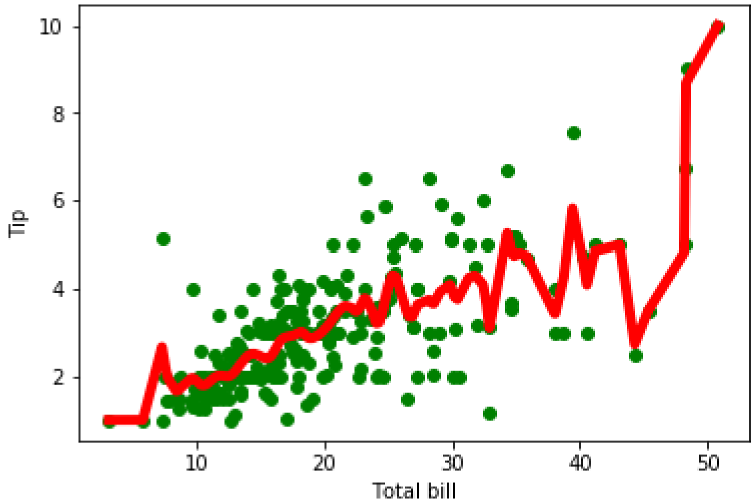
6. Prediction = x0*β

Python 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')
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



## Content Beyond Syllabus:

## Additional Programs

In hill climbing the basic idea is to always head towards a state which is better than the current one.
So, if you are in town A and you can get to town B and town C (and your target is town D) then you should make a move IF town B or C appear nearer to town D than town A does.

Simplest Hill-CLimbing Search Algorithm

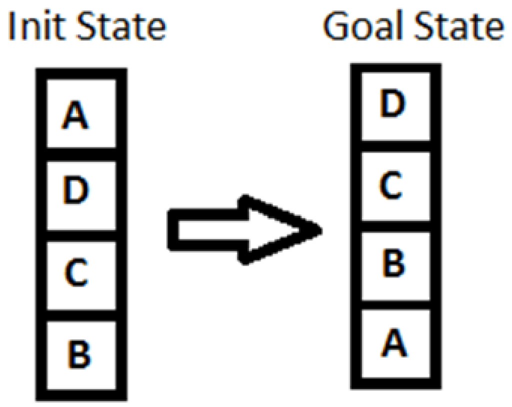
1. Evaluate the initial state.
   If it is also goal state then return it, otherwise continue with the initial state as the current state.

2. Loop until the solution is found or until there are no new operators to be applied in the current state

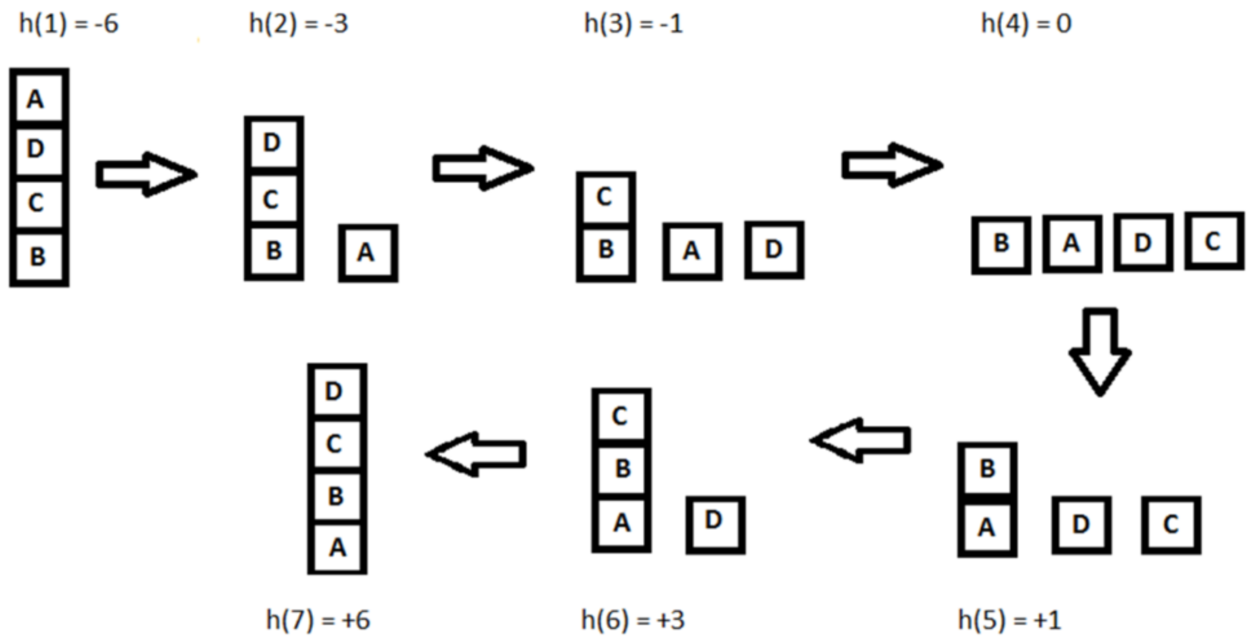
   a) Select an operator that has not yet been applied to the current state and apply it to produce new state

   b) Evaluate the new state
      i) If it is a goal state then return it and quit
      ii) If it is not a goal state but it is better than the current state, then make it as current state
      iii) If it is not better than the current state, then continue in loop.

To understand the concept easily, we will take up a very simple example,




The key point while solving any hill-climbing problem is to choose an appropriate heuristic function.
Let’s define such function h:
h(x) = +1 for all the blocks in the support structure if the block is correctly positioned otherwise -1 for all the blocks in the support structure.




Steepest-Ascent Hill Climbing Search Algorithm in Artificial Intelligence
A variation on simple hill climbing.
Instead of moving to the first state that is better, move to the best possible state that is one move away.
The order of operators does not matter.
Not just climbing to a better state, climbing up the steepest slope.
Considers all the moves from the current state.
Selects the best one as the next state.
Basic hill-climbing first applies one operator n gets a new state. If it is better that becomes the current state whereas the steepest climbing tests all possible solutions n chooses the best.
1. Evaluate the initial state.
If it is also a goal state then return it and quit. Otherwise continue with the initial state as the current state.

2. Loop until a solution is found or until a complete iteration produces no change to current state:

a) Let SUCC be a state such that any possible successor of the current state will be better than SUCC.

b) For each operator that applies to the current state do:
i) Apply the operator and generate a new state.
ii) Evaluate the new state. If it is a goal state, then return it and quit.
If not compare it to SUCC. If it is better, then set SUCC to this state.
If it is not better, leave SUCC alone.
c) IF the SUCC is better than current state, then set current state to SUCC.

Hill Climbing Termination
Local Optimum: all neighboring states are worse or the same.
Plateau – all neighbouring states are the same as the current state.
Ridge – local optimum that is caused by the inability to apply 2 operators at once.

Disadvantages of Hill Climbing
Hill climbing is a local method: Decides what to do next by looking only at the “immediate” consequences of its choices.

Will terminate when at a local optimum.
The order of application of operators can make a big difference.
Global information might be encoded in heuristic functions.

## Lab Evaluation:

| SL NO | USN NO | NAME | Details | E 1/ P 1 | E 2/ P 2 | E 3/ P 3 | E 4/ P 4 | E 5/ P 5 | E 6/ P 6 | E 7/ P 7 | E 8/ P 8 | E9 /P 9 | Avg (25 M) | Test (15 M) | Final Marks | Signature |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | | | a | | | | | | | | | | | | | |
| | | | b | | | | | | | | | | | | | |
| | | | c | | | | | | | | | | | | | |
| | | | d | | | | | | | | | | | | | |
| | | | TOT | | | | | | | | | | | | | |
| **2** | | | a | | | | | | | | | | | | | |
| | | | b | | | | | | | | | | | | | |
| | | | c | | | | | | | | | | | | | |
| | | | d | | | | | | | | | | | | | |
| | | | TOT | | | | | | | | | | | | | |
| **3** | | | a | | | | | | | | | | | | | |
| | | | b | | | | | | | | | | | | | |
| | | | c | | | | | | | | | | | | | |
| | | | d | | | | | | | | | | | | | |
| | | | TOT | | | | | | | | | | | | | |

**Rubrics for Lab**

**1. FOR 40 MARKS (2018 NEW SCHEME)**

| Sl. No. | DESCRIPTION | MARKS |
|---|---|---|
| 1. | **CONTINUOUS EVALUATION** | **25** |
| | a. Observation write up & punctuality | 5.0 |
| | b. Conduction of experiment and output | 8.0 |
| | c. Viva voce | 4.0 |
| | d. Record write up | 8.0 |
| 2. | **INTERNAL TEST** | 15.0 |