



**ACHARYA INSTITUTE OF TECHNOLOGY**

Acharya Dr. Sarvepalli Radhakrishnan Road, Bangalore-560 107

**Department OF Information Science & Engineering**

**ARTIFICIAL INTELLIGENCE & MACHINE LEARNING  
LABORATORY MANUAL [18CSL76]  
(VII SEM CSE-CBCS REVISED 2018 SCHEME)**

**Prepared By:**

**Prof. JAGADISH N**

Assistant Professor

Dept of ISE

AcIT, BANGALORE



Acharya Dr. Sarvepalli Radhakrishnan Road, Bangalore-560 107

**Department of Information Science & Engineering**

During the Year: 2023-24

## **Acharya Institute of Technology**

### **Motto:**

“Nurturing Aspiration and supporting Growth”

### **Vision:**

“**Acharya Institute of Technology**, committed to the cause of sustainable value based education in all disciplines, envisions itself as a global fountainhead of innovative human enterprise, with inspirational initiatives for Academic Excellence”.

### **Mission:**

“**Acharya Institute of Technology** strives to provide excellent academic ambiance to the students for achieving global standards of technical education, foster intellectual and personal development, meaningful research and ethical service to sustainable societal needs”.

## **Department of Information Science and Engineering**

### **Vision:**

“To be center of Academic and Research excellence in the field of Information Technology inculcating value based education for the development of quality Human Resource”

### **Mission:**

“Equip students with fundamental concepts, practical knowledge and professional ethics through dedicated faculty for higher studies and professional career in various Scientific, Engineering and Technological streams leading to proficiency in the field of Information Technology”

### **Programme Outcomes(POs):**

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

**Program Educational Objectives (PEOs):**

**PEO1:** Will be competent, creative, and highly valued ethical professionals in industry, academia or entrepreneur.

**PEO2:** Engage in life-long learning and professional development.

**PEO3:** Become effective global collaborators, leading or participating to address technical, business, environmental and societal challenges.

**Program Specific Outcomes (PSOs):**

**PSO1:** Able to apply knowledge of information management and communication systems to provide secured solutions for real time engineering applications.

**PSO2:** Apply best software engineering practices, modern tools and technologies to deliver quality products.

**COURSE OUTCOMES**

CO's		
CO1	Design the program for the given specifications.	
CO2	Simulate the experiments with the given specification.	
CO3	Validate the readings and infer the results graphically/mathematically.	
CO4	Interpret the concepts and results both orally and written.	

**Assessment of course outcomes:**

Course Outcomes-Program Outcomes mapping

COs	Program Outcomes												PSO	
	1	2	3	4	5	6	7	8	9	10	11	12	1	2
CO1	3	2	1	2	3				2	2		2	1	2
CO2	2	2	2	2	3				2	2		2	2	2
CO3	2	2	1	2	3	1			2	2		2	2	2
CO4	3	2	2	3	3	1		2	2	3		2	2	2

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			
<b>Course Learning Objectives:</b> This course (18CSL76) will enable students to:			
<ul style="list-style-type: none"><li>Implement and evaluate AI and ML algorithms in and Python programming language.</li></ul>			
<b>Descriptions (if any):</b>			
<b>Installation procedure of the required software must be demonstrated, carried out in groups and documented in the journal.</b>			
<b>Programs List:</b>			
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 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 Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs		
<b>Laboratory Outcomes:</b> The student should be able to:			
<ul style="list-style-type: none"><li>Implement and demonstrate AI and ML algorithms.</li><li>Evaluate different algorithms.</li></ul>			
<b>Conduct of Practical Examination:</b>			
<ul style="list-style-type: none"><li>Experiment distribution<ul style="list-style-type: none"><li>For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.</li><li>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.</li></ul></li><li>Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.</li><li>Marks Distribution (<i>Courseed to change in accordance with university regulations</i>)<ul style="list-style-type: none"><li>q) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks</li></ul></li></ul>			

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

## 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.

## Anaconda 5.1 For Windows Installer

**A**
**Python 3.6 version \***
[Download](#)
[64-Bit Graphical Installer \(537 MB\)](#) ⓘ  
[32-Bit Graphical Installer \(436 MB\)](#)
**B**
**Python 2.7 version \***
[Download](#)
[64-Bit Graphical Installer \(523 MB\)](#) ⓘ  
[32-Bit Graphical Installer \(420 MB\)](#)
[Behind a firewall?](#)
[How to get Python 3.5 or other Python versions](#)
[How to Install ANACONDA](#)

## 2. Locate your download and double click it.

## Anaconda 5.1 For Windows Installer

**Python 3.6 version \***
[Download](#)
[64-Bit Graphical Installer \(537 MB\)](#) ⓘ  
[32-Bit Graphical Installer \(436 MB\)](#)
**Python 2.7 version \***
[Download](#)
[64-Bit Graphical Installer \(523 MB\)](#) ⓘ  
[32-Bit Graphical Installer \(420 MB\)](#)
[Behind a firewall?](#)
[How to get Python 3.5 or other Python versions](#)
[How to Install ANACONDA](#)

Get Started

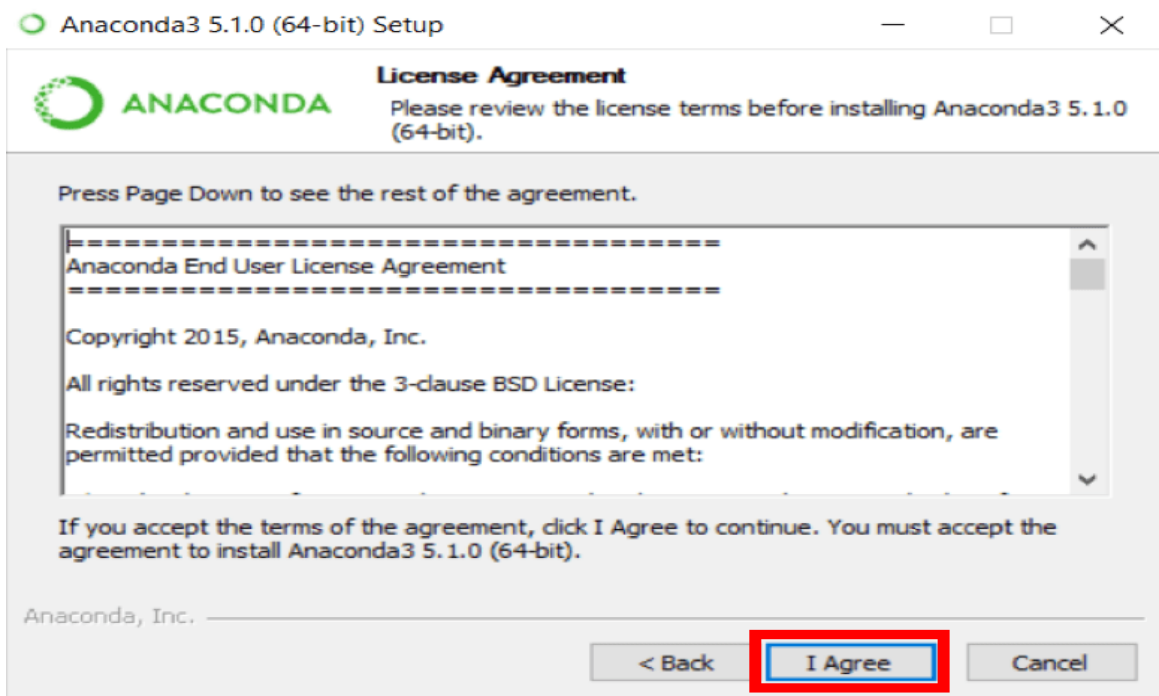

[Show all](#)

When the screen below appears, click on Next.

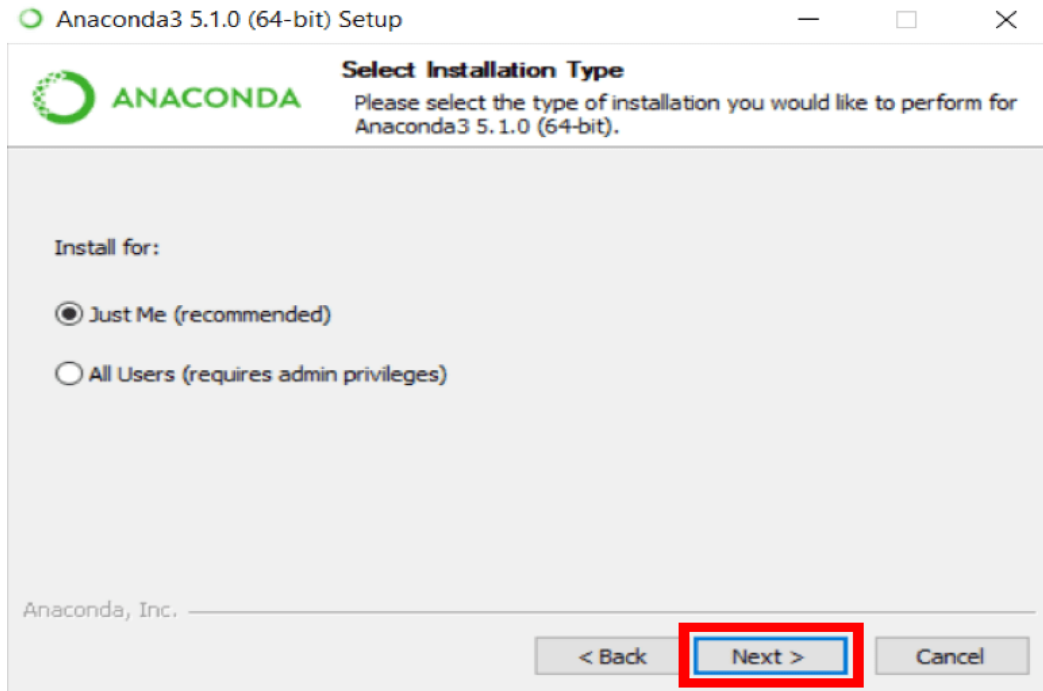




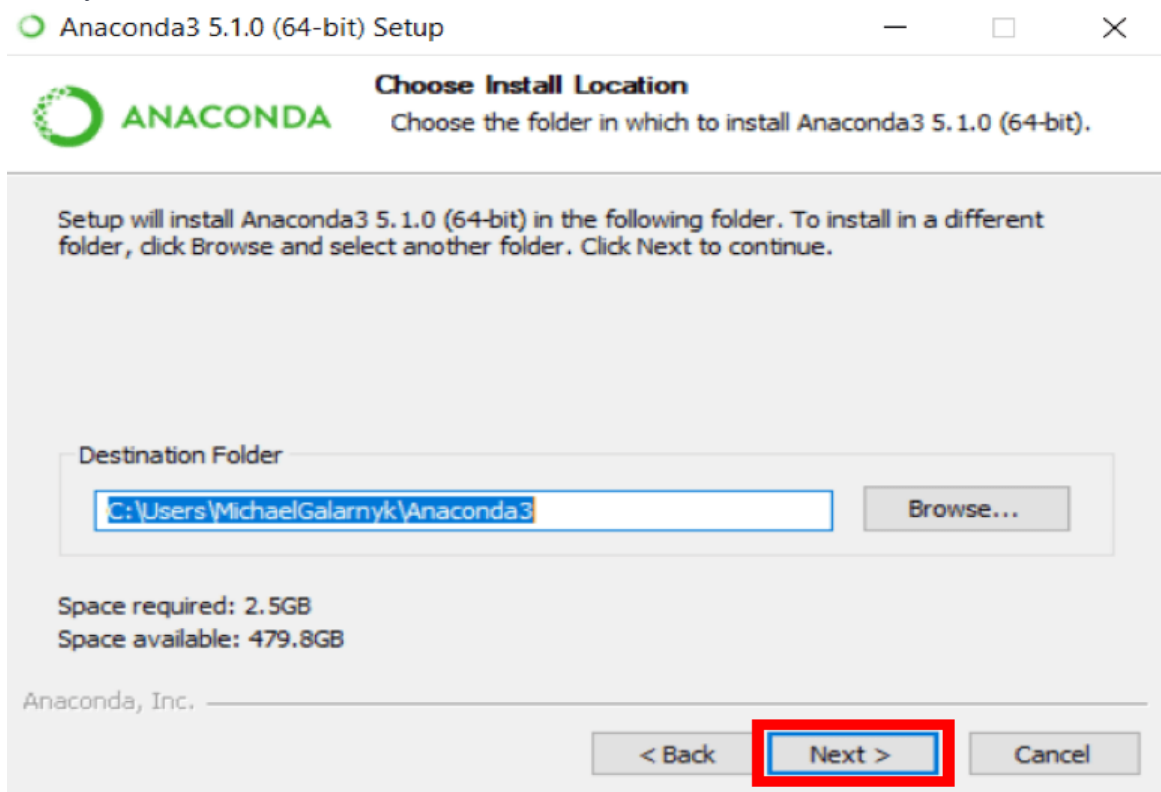
3. Read the license agreement and click on I Agree.



4. Click on Next.



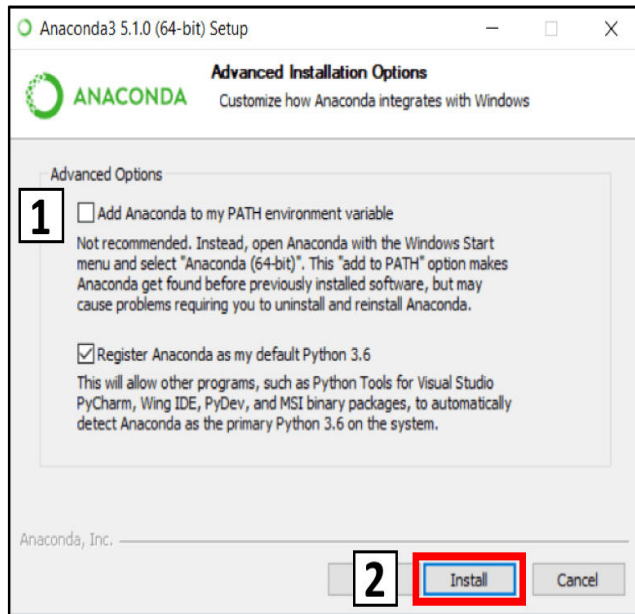
5. Note your installation location and then click Next.



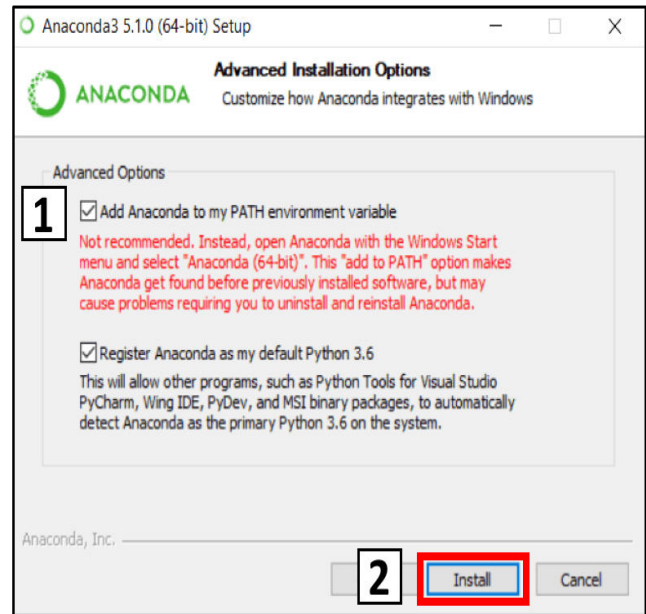
6. This is an important part of the installation process. The recommended approach is to not check the box to add Anaconda to your path. This means you will have to use Anaconda

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.

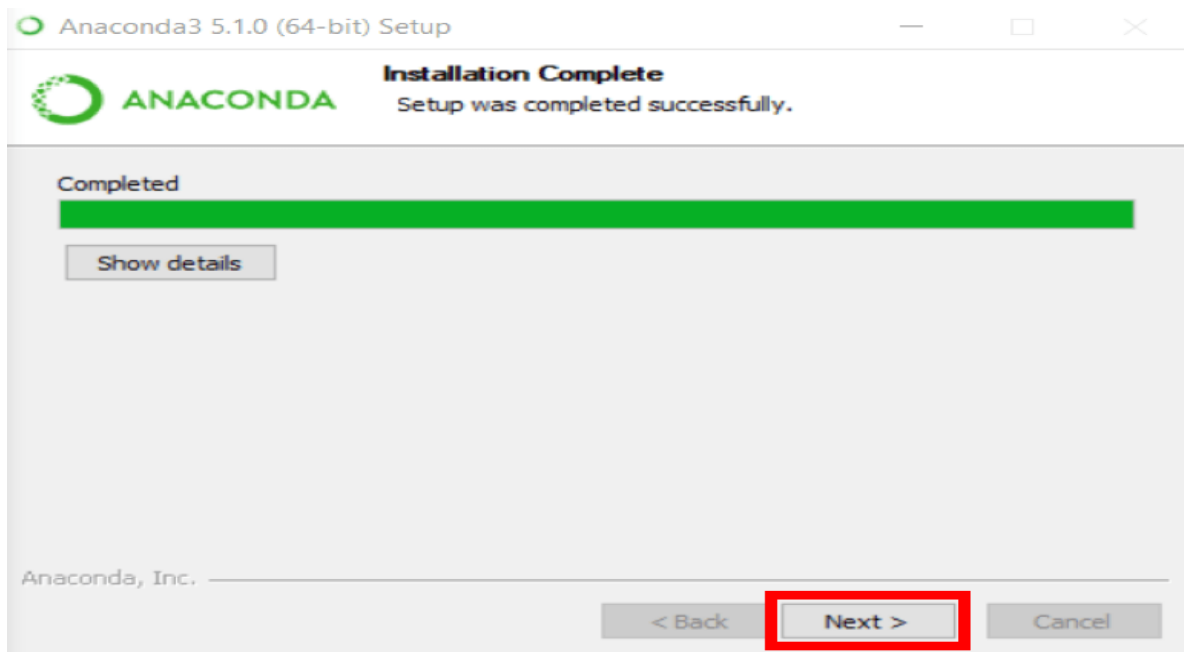
## Recommended Approach



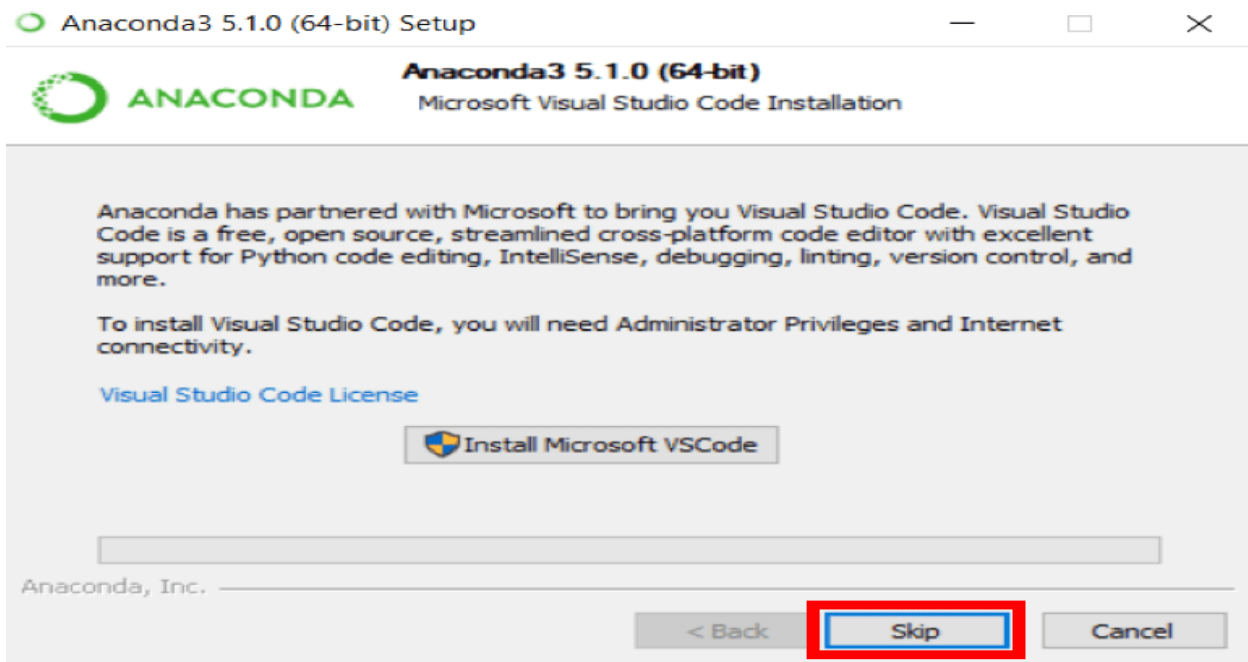
## Alternative Approach



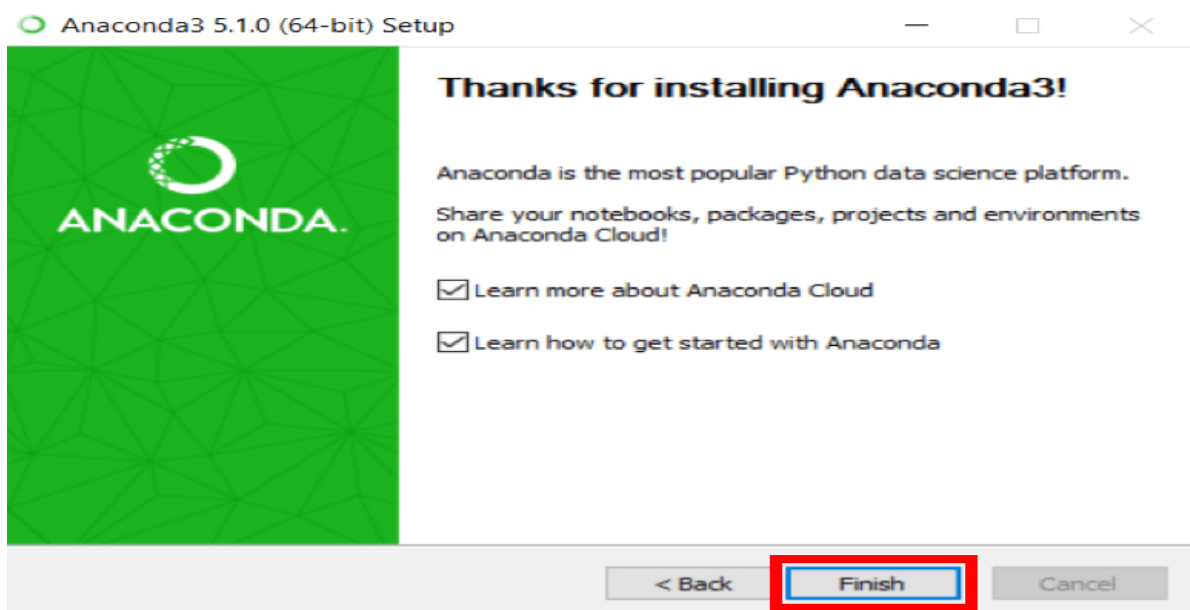
7. Click on Next.



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



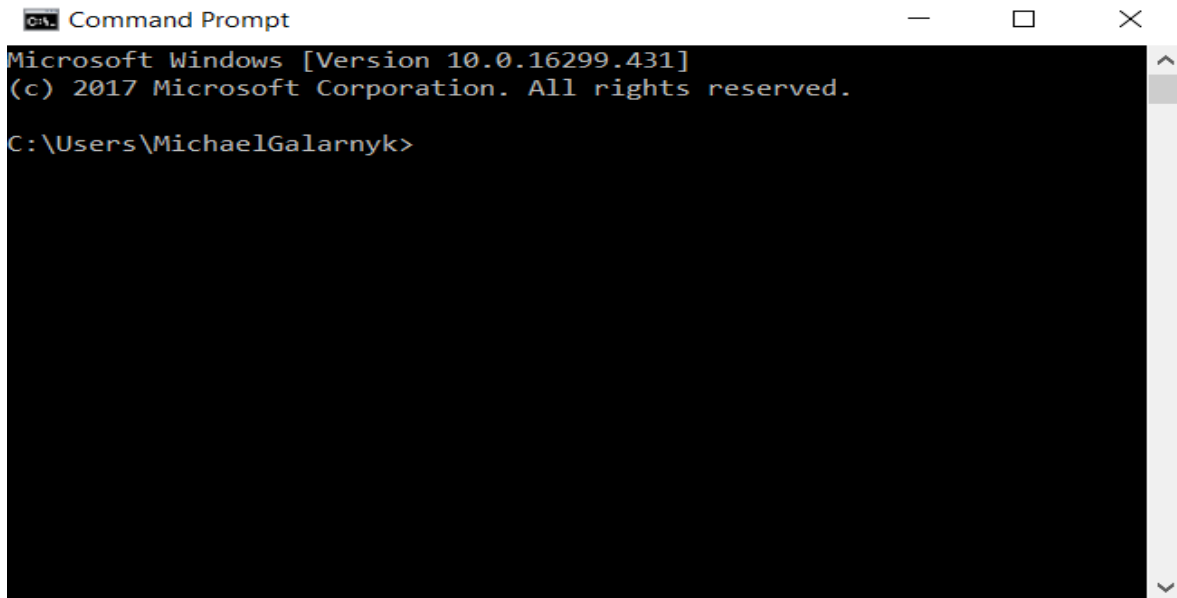
9. Click on Finish.



**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, cmd 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

## Proceed to Step 3

```
Select Command Prompt
Microsoft Windows [Version 10.0.16299.431]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\MichaelGalarnyk>conda --version
'conda' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\MichaelGalarnyk>python --version
'python' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\MichaelGalarnyk>
```

## Anaconda Already Added to Path

```
Select Command Prompt
Microsoft Windows [Version 10.0.16299.431]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\MichaelGalarnyk>conda --version
conda 4.4.10

C:\Users\MichaelGalarnyk>python --version
Python 3.6.4 :: Anaconda, Inc.

C:\Users\MichaelGalarnyk>
```

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

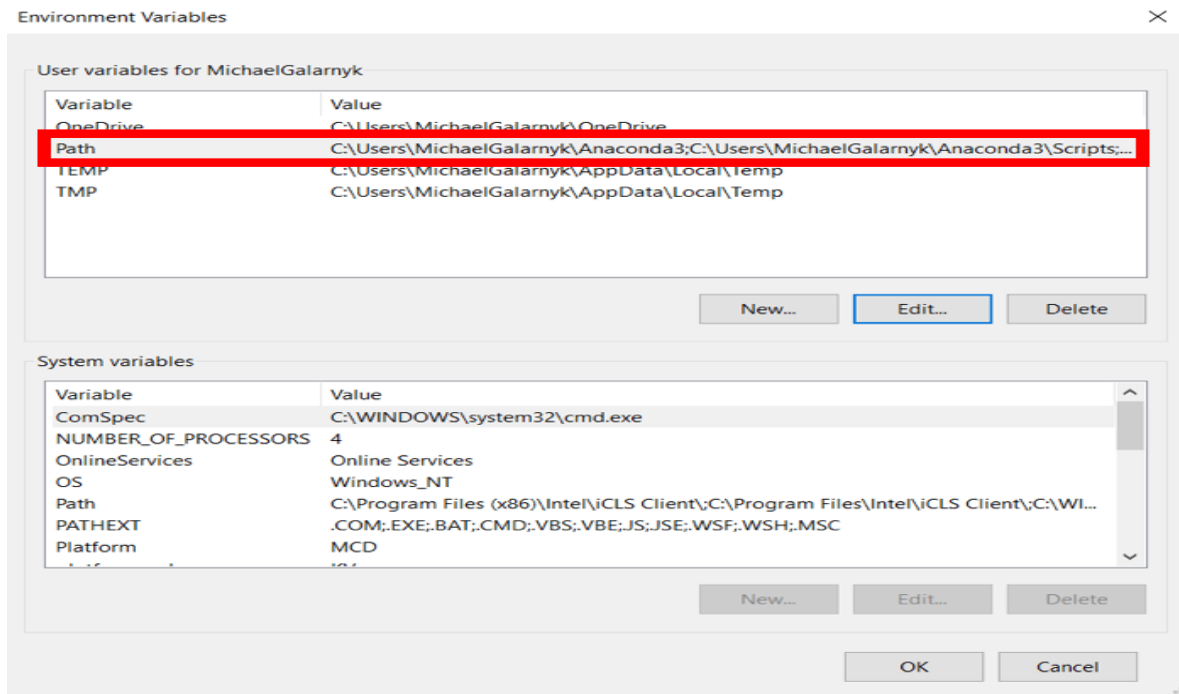
where python

```
Anaconda Prompt
(base) C:\Users\MichaelGalarnyk>where conda
C:\Users\MichaelGalarnyk\Anaconda3\Library\bin\conda.bat
C:\Users\MichaelGalarnyk\Anaconda3\Scripts conda.exe

(base) C:\Users\MichaelGalarnyk>where python
C:\Users\MichaelGalarnyk\Anaconda3 python.exe

(base) C:\Users\MichaelGalarnyk>
```

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.



Select Command Prompt



```
Microsoft Windows [Version 10.0.16299.431]  
(c) 2017 Microsoft Corporation. All rights reserved.
```

```
C:\Users\MichaelGalarnyk>conda --version  
conda 4.4.10
```

```
C:\Users\MichaelGalarnyk>python --version  
Python 3.6.4 :: Anaconda, Inc.
```

```
C:\Users\MichaelGalarnyk>
```



PROGRAM.NO.1

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

```
def A_star(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
```

```
            if m not in open_set and m not in closed_set:
```

```
                open_set.add(m)
```

```
                #n is set its parent
```

```
                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: # if better cost found, then update the existing cost 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 reconstructin 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('Optimal 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 = {
        'S': 8,
        'A': 8,
        'B': 4,
        'C': 3,
        'D': 1000,
        'E': 1000,
        'G': 0,
    }

    return H_dist[n]

#Describe your graph here
Graph_nodes = {'S': [['A', 1], ['B', 5], ['C', 8]],
               'A': [['D', 3], ['E', 7], ['G', 9]],
               'B': [['G', 4]],
               'C': [['G', 5]],
               'D': None,
               'E': None}

A_star('S', 'G')
```

**OUTPUT:- (Note:-better to Run in Jupyter Note book of Anaconda for correct output)**

Optimal Path :  
['S', 'B', 'G']

PROGRAM.Mo.2(AO\* Algorithm)

**Implement AO\* Search algorithm.**

```
# Function to implement recursive AO* Algorithm
```

```
def recAOSTar(n):
```

```
    global finalPath
```

```
    print("Expanding Node : ", n)
```

```
    and_nodes = []
```

```
    or_nodes = []
```

```
#Segregation of AND and OR nodes
```

```
if (n in allNodes):
```

```
    if 'AND' in allNodes[n]:
```

```
        and_nodes = allNodes[n]['AND']
```

```
    if 'OR' in allNodes[n]:
```

```
        or_nodes = allNodes[n]['OR']
```

```
# If leaf node then return
```

```
if len(and_nodes) == 0 and len(or_nodes) == 0:
```

```
    return
```

```
solvable = False
```

```
marked = { }
```

```
while not solvable:
```

```
    # If all the child nodes are visited and expanded, take the least cost of all the child nodes
```

```
    if len(marked) == len(and_nodes) + len(or_nodes):
```

```
        min_cost_least, min_cost_group_least = least_cost_group(and_nodes, or_nodes, { })
```

```
        solvable = True
```

```
        change_heuristic(n, min_cost_least)
```

```
        optimal_child_group[n] = min_cost_group_least
```

```
        continue
```

```
# Least cost of the unmarked child nodes
```

```
min_cost, min_cost_group = least_cost_group(and_nodes, or_nodes, marked)
```

```
is_expanded = False
```

```
# If the child nodes have sub trees then recursively visit them to recalculate the heuristic of the child node
```

```
if len(min_cost_group) > 1:
```

```
    if (min_cost_group[0] in allNodes):
```

```
        is_expanded = True
```

```
        recAOSTar(min_cost_group[0])
    if (min_cost_group[1] in allNodes):
        is_expanded = True
        recAOSTar(min_cost_group[1])
    else:
        if (min_cost_group in allNodes):
            is_expanded = True
            recAOSTar(min_cost_group)

    # If the child node had any subtree and expanded, verify if the new heuristic value is still the least
    among all nodes
    if is_expanded:
        min_cost_verify, min_cost_group_verify = least_cost_group(and_nodes, or_nodes, { })

        if min_cost_group == min_cost_group_verify:
            solvable = True
            change_heuristic(n, min_cost_verify)
            optimal_child_group[n] = min_cost_group

    # If the child node does not have any subtrees then no change in heuristic, so update the min cost of
    the current node
    else:
        solvable = True
        change_heuristic(n, min_cost)
        optimal_child_group[n] = min_cost_group

    #Mark the child node which was expanded
    marked[min_cost_group] = 1
    return heuristic(n)

# Function to calculate the min cost among all the child nodes
def least_cost_group(and_nodes, or_nodes, marked):
    node_wise_cost = { }

    for node_pair in and_nodes:
        if not node_pair[0] + node_pair[1] in marked:
            cost = 0
            cost = cost + heuristic(node_pair[0]) + heuristic(node_pair[1]) + 2
            node_wise_cost[node_pair[0] + node_pair[1]] = cost
```

```
for node in or_nodes:
    if not node in marked:
        cost = 0
        cost = cost + heuristic(node) + 1
        node_wise_cost[node] = cost

min_cost = 999999
min_cost_group = None

# Calculates the min heuristic
for costKey in node_wise_cost:
    if node_wise_cost[costKey] < min_cost:
        min_cost = node_wise_cost[costKey]
        min_cost_group = costKey

return [min_cost, min_cost_group]

# Returns heuristic of a node
def heuristic(n):
    return H_dist[n]

# Updates the heuristic of a node
def change_heuristic(n, cost):
    H_dist[n] = cost
    return

# Function to print the optimal cost nodes
def print_path(node):
    print(optimal_child_group[node], end="")
    node = optimal_child_group[node]

    if len(node) > 1:
        if node[0] in optimal_child_group:
            print("->", end="")
            print_path(node[0])
        if node[1] in optimal_child_group:
            print("->", end="")
            print_path(node[1])
    else:
```

```
if node in optimal_child_group:
    print("->", end="")
    print_path(node)

#Describe the heuristic here
H_dist = { 'A': -1,'B': 4, 'C': 2, 'D': 3, 'E': 6,'F': 8, 'G': 2,'H': 0, 'I': 0, 'J': 0}

#Describe your graph here
allNodes = {
    'A': {'AND': [('C', 'D')], 'OR': ['B']},
    'B': {'OR': ['E', 'F']},
    'C': {'OR': ['G'], 'AND': [('H', 'I')]},
    'D': {'OR': ['J']}
}

optimal_child_group = {}
optimal_cost = recAOSTar('A')

print('Nodes which gives optimal cost are')
print_path('A')
print("\nOptimal Cost is :: ', optimal_cost)
OUTPUT:-

Expanding Node : A
Expanding Node : B
Expanding Node : C
Expanding Node : D
Nodes which gives optimal cost are
CD->HI->J
Optimal Cost is :: 5
```

Program.No.3(Candidate Elimination Algorithm)

***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.***

```
dataarr=[]
with open('data/enjoysport.csv') as f:
    for line in f:
        dataarr.append(line.strip().split(','))

rows = len(dataarr)
cols = len(dataarr[0])
shypo = ['0']*(cols-1)
ghypo = ['?']*(cols-1)
print("Initial Specific Hypothesis is ", shypo)
print("Initial General Hypothesis is ", ghypo)

for x in range(1, rows):
    lst = dataarr[x]

    if lst[cols-1] == "1":
        for i in range(0, cols-1):
            if shypo[i] == lst[i]:
                continue
            shypo[i] = '?' if shypo[i] != '0' else lst[i]
        for g in ghypo:
            if g[i] != '?' and shypo[i] == '?':
                ghypo.remove(g)

    elif lst[cols-1] == "0":
        ghypo.clear()
        for i in range(0, cols-1):
            if lst[i] != shypo[i] and shypo[i] != '?':
                temp_list = ['?']*i + [shypo[i]] + ['?']*(cols-2-i)
                if temp_list not in ghypo:
                    ghypo.append(temp_list)

    print("S Hypothesis after row ", x, " = ", shypo)
    print("G Hypothesis after row ", x, " = ", ghypo)
print("Final SHypothesis ", shypo)
print("Final GHypothesis ", ghypo)
```

**OUTPUT:- (Note:-Use Enjoysport csv file)**



Initial Specific Hypothesis is = ['0', '0', '0', '0', '0', '0']  
Initial General Hypothesis is = [['?', '?', '?', '?', '?', '?']]  
S Hypothesis after row 1 = ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']  
G Hypothesis after row 1 = [['?', '?', '?', '?', '?', '?']]  
S Hypothesis after row 2 = ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']  
G Hypothesis after row 2 = [['?', '?', '?', '?', '?', '?']]  
S Hypothesis after row 3 = ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']  
G Hypothesis after row 3 = [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]  
S Hypothesis after row 4 = ['Sunny', 'Warm', '?', 'Strong', '?', '?']  
G Hypothesis after row 4 = [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]  
Final SHypothesis ['Sunny', 'Warm', '?', 'Strong', '?', '?']  
Final GHypothesis [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]

PROGRAM.NO.4(Decision trees)

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

```
import math

def dataset_split(data, arc, val):
    newData = []
    # iterate through every record in the data-set and split the data-set
    for rec in data:
        if rec[arc] == val:
            reducedSet = list(rec[:arc])
            reducedSet.extend(rec[arc+1:])
            newData.append(reducedSet)
    return newData

def calc_entropy(data):
    # Calculate the length of the data-set
    entries = len(data)
    labels = {}
    # Read the class labels from the data-set file into the dict object "labels"
    for rec in data:
        label = rec[-1]
        if label not in labels.keys():
            labels[label] = 0
        labels[label] += 1
    entropy = 0.0
    # For every class label (x) calculate the probability p(x)
    for key in labels:
        prob = float(labels[key])/entries
        # Entropy formula calculation
        entropy -= prob * math.log(prob, 2)
    # Return the entropy of the data-set
    return entropy

def attribute_selection(data):
    features = len(data[0]) - 1
    baseEntropy = calc_entropy(data)
    max_InfoGain = 0.0
    bestAttr = -1

    for i in range(features):
        # store the values of the features in a variable
        AttrList = [rec[i] for rec in data]

        # get the unique values from the feature values
        uniqueVals = set(AttrList)
        newEntropy = 0.0
        attrEntropy = 0.0
        for value in uniqueVals:
            # function call to split the data-set
```

```
newData = dataset_split(data, i, value)

# probability calculation
prob = len(newData)/float(len(data))

# entropy calculation for the attributes
newEntropy = prob * calc_entropy(newData)
attrEntropy += newEntropy

# calculation of Information Gain
infoGain = baseEntropy - attrEntropy
# identify the attribute with max info-gain
if infoGain > max_InfoGain:
    max_InfoGain = infoGain
    bestAttr = i

# return the attribute identified
return bestAttr

def decision_tree(data, labels):
    # list variable to store the class-labels (terminal nodes of decision tree)
    classList = [rec[-1] for rec in data]

    if classList.count(classList[0]) == len(classList):
        return classList[0]

    maxGainNode = attribute_selection(data)
    treeLabel = labels[maxGainNode]

    theTree = {treeLabel: {}}
    del(labels[maxGainNode])

    # get the unique values of the attribute identified
    nodeValues = [rec[maxGainNode] for rec in data]
    uniqueVals = set(nodeValues)
    for value in uniqueVals:
        subLabels = labels[:]

        # update the non-terminal node values of the decision tree
        theTree[treeLabel][value] = decision_tree(dataset_split(data, maxGainNode, value), subLabels)
    return theTree

def print_tree(tree, level):
    if tree == 'yes' or tree == 'no':
        print('*level, 'd =', tree)
```

```
        return
    for key,value in tree.items():
        print(' ' * level, key)
        print_tree(value, level * 2)

with open('data/tennis.csv', 'r') as csvfile:
    fdata = [line.strip() for line in csvfile]
    metadata = fdata[0].split(',')
    train_data = [x.split(',') for x in fdata[1:]]

tree = decision_tree(train_data, metadata)
print_tree(tree, 1)
print(tree)
```

**OUTPUT :- (Note:-Use tennis.csv as dataset)**

Outlook

overcast

    d = yes

rain

    Wind

        weak

            d = yes

        strong

            d = no

sunny

    Humidity

        high

            d = no

        normal

            d = yes

```
{'Outlook': {'overcast': 'yes', 'rain': {'Wind': {'weak': 'yes', 'strong': 'no'}}, 'sunny': {'Humidity': {'high': 'no', 'normal': 'yes'}}}}
```

**PROGRAM.No.5 (Backproagation Algorithm)**

***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)

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

def der_sigmoid(x):
    return x * (1 - x)

epoch = 5000
lr = 0.01
neurons_i = 2
neurons_h = 3
neurons_o = 1

weight_h = np.random.uniform(size=(neurons_i, neurons_h))
bias_h = np.random.uniform(size=(1, neurons_h))
weight_o = np.random.uniform(size=(neurons_h, neurons_o))
bias_o = np.random.uniform(size=(1, neurons_o))

for i in range(epoch):
    inp_h = np.dot(X, weight_h) + bias_h
    out_h = sigmoid(inp_h)

    inp_o = np.dot(out_h, weight_o) + bias_o
    out_o = sigmoid(inp_o)

    err_o = y - out_o
    grad_o = der_sigmoid(out_o)
    delta_o = err_o * grad_o

    err_h = delta_o.dot(weight_o.T)
    grad_h = der_sigmoid(out_h)
    delta_h = err_h * grad_h

    weight_o += out_h.T.dot(delta_o) * lr
    weight_h += X.T.dot(delta_h) * lr

print('Input: ', X)
print('Actual: ', y)
print('Predicted: ', out_o)
```

**OUTPUT:-**

```
Input: [[ 0.66666667  1.      ]
 [ 0.33333333  0.55555556]
 [ 1.      0.66666667]]
Actual: [[ 0.92]
 [ 0.86]
 [ 0.89]]
```

Predicted: [[ 0.89371021]  
[ 0.87852765]  
[ 0.89052431]]

***PROGRAM.NO.6 (Naive bayes Classifier)***

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

```
import pandas as pd
import numpy as np
mush = pd.read_csv("data/mushrooms.csv")
```

```
mush = mush.replace('?', np.nan)
mush.dropna(axis=1, inplace=True)
target = 'class'
features = mush.columns[mush.columns != target]
target_classes = mush[target].unique()
test = mush.sample(frac=.3)
mush = mush.drop(test.index)
cond_probs = {}
target_class_prob = {}
for t in target_classes:
    mush_t = mush[mush[target] == t][features]
    target_class_prob[t] = float(len(mush_t) / len(mush))
    class_prob = {}

    for col in mush_t.columns:
        col_prob = {}
        for val, cnt in mush_t[col].value_counts().iteritems():
            pr = cnt/len(mush_t)
            col_prob[val] = pr
        class_prob[col] = col_prob
    cond_probs[t] = class_prob
def calc_probs(x):
    probs = {}
    for t in target_classes:
        p = target_class_prob[t]
        for col, val in x.iteritems():
            try:
                p *= cond_probs[t][col][val]
            except:
                p = 0
        probs[t] = p
    return probs
def classify(x):
    probs = calc_probs(x)
    max = 0
    max_class = ""
    for cl, pr in probs.items():
        if pr > max:
            max = pr
            max_class = cl
    return max_class
b = []
for i in mush.index:
    b.append(classify(mush.loc[i, features]) == mush.loc[i, target])
print(sum(b), "correct of", len(mush))
print("Accuracy:", sum(b)/len(mush))
# Test data
```

```
b = []
for i in test.index:
    b.append(classify(test.loc[i, features]) == test.loc[i, target])
print(sum(b), "correct of", len(test))
print("Accuracy:", sum(b)/len(test))
```

**OUTPUT :- (NOTE:-Use Mushroom.csv as dataset)**

5671 correct of 5687  
Accuracy: 0.997186565851943

2435 correct of 2437  
Accuracy: 0.9991793188346327

5674 correct of 5687  
Accuracy: 0.9977140847547037

2433 correct of 2437  
Accuracy: 0.9983586376692655

5671 correct of 5687  
Accuracy: 0.997186565851943

2434 correct of 2437  
Accuracy: 0.9987689782519491

***PROGRAM.No.7(K-Means and EM algorithm)***

***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.***

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



```
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans

data = pd.read_csv('data/ex.csv')

f1 = data['V1'].values
f2 = data['V2'].values
X = np.array(list(zip(f1, f2)))
print("x: ", X)

print('Graph for whole dataset')
plt.scatter(f1, f2, c='black') # size can be set by adding s=size as param
plt.show()

kmeans = KMeans(2)
labels = kmeans.fit(X).predict(X)
print("labels for kmeans:", labels)

print('Graph using Kmeans Algorithm')
plt.scatter(f1, f2, c=labels)

centroids = kmeans.cluster_centers_
print("centroids:", centroids)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', c='red')

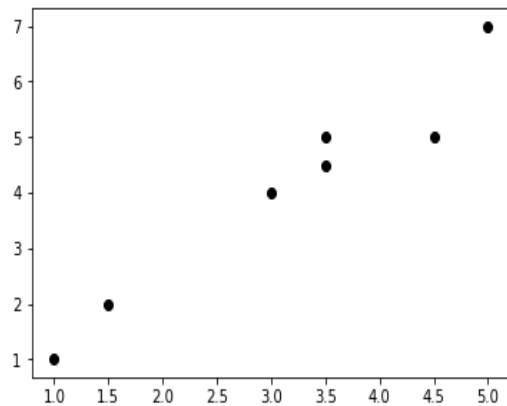
plt.show()

gmm = GaussianMixture(2)
labels = gmm.fit(X).predict(X)
print("Labels for GMM: ", labels)

print('Graph using EM Algorithm')
plt.scatter(f1, f2, c=labels)
plt.show()
```

OUTPUT:- (NOTE:-USE ex.csv )

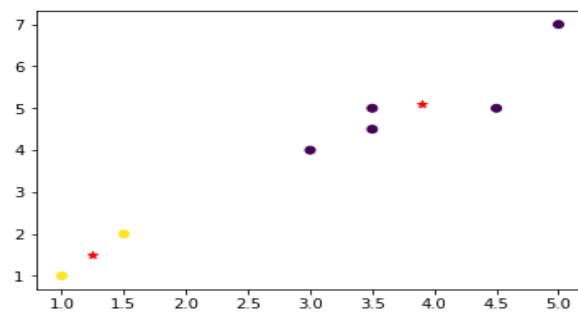
```
x: [[ 1.  1.] [ 1.5  2.] [ 3.  4.] [ 5.  7.] [ 3.5  5.] [ 4.5  5.] [ 3.5  4.5]]
Graph for whole dataset
```



labels for kmeans: [1 1 0 0 0 0 0]

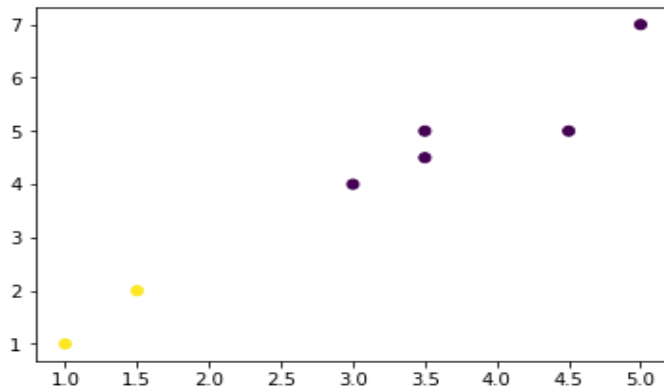
Graph using Kmeans Algorithm

centroids:  $\begin{bmatrix} 3.9 & 5.1 \\ 1.25 & 1.5 \end{bmatrix}$

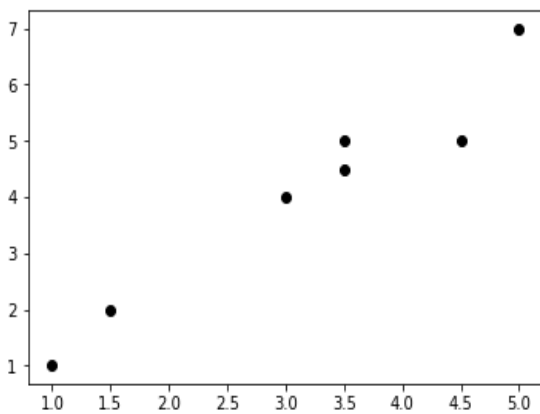


Labels for GMM: [1 1 0 0 0 0 0]

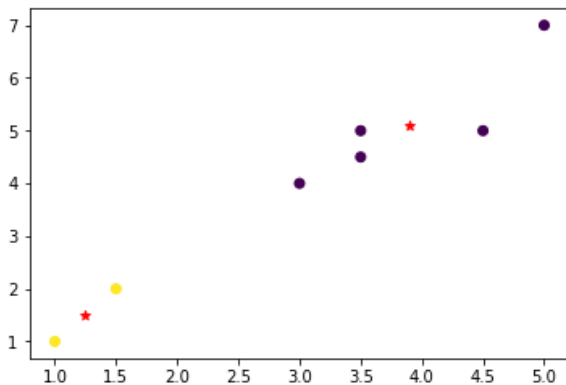
Graph using EM Algorithm



x: [[ 1. 1.] [ 1.5 2.] [ 3. 4.] [ 5. 7.] [ 3.5 5.] [ 4.5 5.] [ 3.5 4.5]]  
Graph for whole dataset

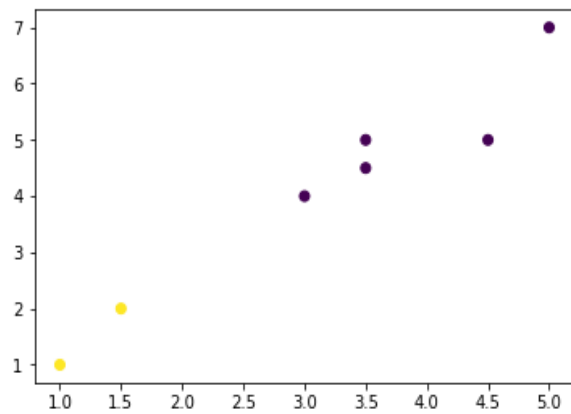


labels for kmeans: [1 1 0 0 0 0 0]  
Graph using Kmeans Algorithm  
centroids: [[ 3.9 5.1 ] [ 1.25 1.5 ]]



Labels for GMM: [1 1 0 0 0 0 0]

Graph using EM Algorithm

**PROGRAM.NO.8(KNN)**

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

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.model_selection import train_test_split
```

```
iris_dataset = load_iris()
targets = iris_dataset.target_names
```

```
print("Class : number")
for i in range(len(targets)):
    print(targets[i], ': ', i)

X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"])
kn = KNeighborsClassifier(1)
kn.fit(X_train, y_train)

for i in range(len(X_test)):
    x_new = np.array([X_test[i]])
    prediction = kn.predict(x_new)
    print("Actual:[{0}] [{1}], Predicted:{2} {3}".format(y_test[i], targets[y_test[i]], prediction,
    targets[prediction]))

print("\nAccuracy: ", kn.score(X_test, y_test))
```

**OUTPUT:-**

```
Class : number
setosa : 0
versicolor : 1
virginica : 2
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[0] [setosa], Predicted:[0] ['setosa']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[0] [setosa], Predicted:[0] ['setosa']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[0] [setosa], Predicted:[0] ['setosa']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[0] [setosa], Predicted:[0] ['setosa']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[2] [virginica], Predicted:[2] ['virginica']
Actual:[0] [setosa], Predicted:[0] ['setosa']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
Actual:[1] [versicolor], Predicted:[1] ['versicolor']
```

```
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[1] [versicolor],Predicted:[2] ['virginica']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[0] [setosa],Predicted:[0] ['setosa']
Actual:[1] [versicolor],Predicted:[1] ['versicolor']
Actual:[2] [virginica],Predicted:[2] ['virginica']
Actual:[0] [setosa],Predicted:[0] ['setosa']
```

Accuracy: 0.973684210526

### ***PROGRAM.NO.9(Locally Weighted Regression Algorithm)***

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

```
from math import ceil
import numpy as np
from scipy import linalg
```

```
def lowess(x, y, f=2. / 3., iter=3):
    n = len(x)
    r = int(ceil(f * n))
    h = [np.sort(np.abs(x - x[i]))[r] for i in range(n)]
    w = np.clip(np.abs((x[:, None] - x[None, :]) / h), 0.0, 1.0)
    w = (1 - w ** 3) ** 3
```

```
yest = np.zeros(n)
delta = np.ones(n)
for iteration in range(iter):
    for i in range(n):
        weights = delta * w[:, i]
        b = np.array([np.sum(weights * y), np.sum(weights * y * x)])
        A = np.array([[np.sum(weights), np.sum(weights * x)],
                       [np.sum(weights * x), np.sum(weights * x * x)]])
        beta = linalg.solve(A, b)
        yest[i] = beta[0] + beta[1] * x[i]

    residuals = y - yest
    s = np.median(np.abs(residuals))
    delta = np.clip(residuals / (6.0 * s), -1, 1)
    delta = (1 - delta ** 2) ** 2
```

```
return yest
```

```
if __name__ == '__main__':
    import math
    n = 100
    x = np.linspace(0, 2 * math.pi, n)
    y = np.sin(x) + 0.3 * np.random.randn(n)

    # Straight Line Fitting
    # x=np.linspace(0,2.5,n) # For Linear
    # y= 1 + 0.25*np.random.randn(n) # For Linear

    f = 0.25
    yest = lowess(x, y, f, 3)
    import pylab as pl
    pl.clf()
    pl.plot(x, y, label='y noisy')
    pl.plot(x, yest, label='y predicted')
    pl.legend()
    pl.show()
```

### **OUTPUT:-**

