SCHEME: K

Name: _____
Roll No.: ____ Year: 20___ - 20___
Exam Seat No.: _____

LABORATORY MANUAL FOR

AI & ML ALGORITHM (315330)

COMPUTER ENGINEERING GROUP

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION, MUMBAI
(Autonomous) (ISO21001:2018) (ISO/IEC27001:2013)

VISION

To ensure that the Diploma Level Technical Education constantly matches the latest requirements of Technology and industry and includes the all-round personal development of students including social concerns and to become globally competitive, technology led organization.

MISSION

To provide high quality technical and managerial manpower, information and consultancy services to the industry and community to enable the industry and community to face the challenging technological & environmental challenges.

QUALITY POLICY

We, at MSBTE are committed to offer the best in class academic services to the students and institutes to enhance the delight of industry and society. This will be achieved through continual improvement in management practices adopted in the process of curriculum design, development, implementation, evaluation and monitoring system along with adequate faculty development programmes.

CORE VALUES

MSBTE believes in the following:

- Skill development in line with industry requirements
- Industry readiness and improved employability of Diploma holders
- Synergistic relationship with industry
- Collective and Cooperative development of all stake holders
- Technological interventions in societal development
- Access to uniform quality technical education

A Laboratory Manual

For

AI and ML Algorithm

(315330)

SEMESTER-V

"K-SCHEME"

(AI/AN)

Maharashtra State

Board of Technical Education, Mumbai.

(Autonomous) (ISO21001:2018) (ISO/IEC27001:2013)

Maharashtra State Board of Technical Education, Mumbai

(Autonomous) (ISO21001:2018) (ISO/IEC27001:2013)

4th Floor, Government Polytechnic Building, 49, Kherwadi,

Bandra (East), Mumbai – 400051,

(Printed on June 2025)

Maharashtra State Board of Technical Education

Certificate

This is to certify that Mr./Ms.…… Roll No……. of Fifth Semester of Diploma in …… ……..…….…….…… of Institute, ...……...…… ……..… (Instt. Code: ……) has completed the term work satisfactorily in course **AI and ML Algorithm (315330)** for the academic year 20…… to 20…… as prescribed in the curriculum**.**

Place: …… Enrollment No: ……..

Date: ……. Exam. Seat No: …….

Course teacher **Head of the Department** **Principal**

Seal of Institution

PREFACE

The development of the critically important industry-relevant abilities and skills is the main goal of any engineering laboratory or field work in the technical education system. In light of this, MSBTE developed the most recent "K" Scheme curricula for engineering diploma programs, emphasizing outcome-based learning. As a result, a sizable portion of the program is dedicated to practical work. This demonstrates how crucial laboratory work is in helping teachers, instructors, and students understand that every minute of lab time must be used efficiently to create these outcomes rather than wasting it on unnecessary activities. Every practical has thus been created to operate as a "vehicle" to help each student acquire this industry-identified capability in order to ensure the effective implementation of this outcome-based curriculum. The "chalk and duster" practice in the classroom is a challenging way to build practical skills. As a result, the development team of the "K" scheme laboratory manual focused on the intended results when creating the practical, as opposed to the customary approach of performing practical's to "verify the theory".

This lab manual is intended to support all parties involved, particularly the students, instructors, and teachers, in helping the students achieve the pre-established goals. It is required of every student to read through the relevant practical process in its entirety and comprehend the bare minimum of theoretical background related to the practical at least one day in advance of the practical. As a crucial starting point for carrying out the practical, each exercise in this handbook starts with establishing the competency, industry-relevant skills, course outcomes, and practical results. After that, the students will learn about the abilities they will acquire through the process outlined there and the safety measures that must be followed, which will enable them to use in addressing real-world situations in their professional life.

This manual also offers guidance to educators on how to manage resources so that students follow protocols and safety measures methodically and meet learning objectives. This allows teachers and instructors to effectively support student-centered lab activities through each practical exercise.

Today's globalized world has witnessed tremendous technological breakthroughs in surveying equipment and technology. Currently available accurate digital surveying tools are employed because of their speed, precision, and ease of use. The disciplines of civil engineering, mining engineering, environmental engineering, transportation engineering, and marine engineering heavily rely on these tools and applications. Given the importance of remote sensing and Geographic Information Systems (GIS) and their widespread usage in mapping and storing spatial data, it is expected that students will have a basic understanding of these subjects in order to use them in the field. Students who complete this course will have the necessary abilities and competences to perform tasks linked to surveys.

Although best possible care has been taken to check for errors (if any) in this laboratory manual, perfection may elude us as this is the first edition of this manual. Any errors and suggestions for improvement are solicited and highly welcome.

Program outcome (POs)

PO 1. Basic & Discipline specific knowledge: Apply knowledge of basic mathematics, sciences and engineering fundamentals and engineering specialization to solve the engineering problems.

PO 2. Problem Analysis: Identify and analyze well defined engineering problems using codified standard methods.

PO 3. Design /Development Solutions: Design solutions for well-defined technical problems and assist with the design of systems components or processes to meet specified needs.

PO 4. Engineering tools experimentation and testing: Apply modern engineering tools and appropriate technique to conduct standard tests and measurements.

PO 5. Engineering practices for society sustainability and environment: Apply appropriate technology in context of society, sustainability, environment and ethical practices.

PO 6. Project Management: Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities.

PO 7. Lifelong learning: Ability to analyze individual needs and engage in updating in context of technological changes.

List of Relevant Skills

On the successful completion of the course the students will acquire the required industry relevant skills and they will be able to:

1. Implement algorithms using libraries like scikit-learn.
2. Design and build machine learning models.
3. Apply preprocessing techniques on datasets
4. Design and analyze machine learning algorithms and decision-making processes.
5. Identify real-world problems suitable for AI/ML solutions.
6. Draw insights from structured and unstructured datasets.
7. Translate business problems into ML tasks.

Practical Course outcome matrix:

CO1 - Implement relevant search algorithms as applicable to Artificial Intelligence.
CO2 - Apply method for knowledge representation to make informed decisions for various applications.
CO3 - Analyze different forms of data with respect to different phases of Machine Learning.
CO4 - Create data model for Machine Learning Algorithms.
CO5 - Classify the data by performing different Regression Techniques.

Pr. No.	Title of the Practical	Mapped Course Outcome				
		CO1	CO2	CO3	CO4	CO5
1	*Install given Python IDE software and Python "scikit learn" for ML	√	--	--	--	--
2	Write program to Implement Breadth First Search Algorithm (Uninformed) in Python	√	--	--	--	--
3	*Write program to implement Depth First Search Algorithm (Uninformed) in Python	√	--	--	--	--
4	Write program to implement Greedy Best-First (Informed Type) Search Algorithm in python	√	--	--	--	--
5	*Write program to implement A* search (Informed Type) Algorithm in Python	√	--	--	--	--
6	*Write program to implement Bayes' Theorem	--	√	--	--	--
7	Analyze the given Case study: How Turing test is performed between Responder and an Interrogator?	--	--	√	--	--
8	*Explore different dataset finders e.g. Google Dataset Search, Kaggle	--	--	√	--	--
9	*Write program in python to split any dataset into train and tests sets	--	--	√	--	--
10	Analyze E-mail spam and non-spam filtering using Machine Learning through case study	--	--	√	--	--
11	*Create and display a Decision Tree on given dataset	--	--	--	√	--
12	Write program to implement K-means Algorithm	--	--	--	√	--
13	*Write program to calculate cross validation score for any Dataset like IRIS	--	--	--	√	--
14	*Write program to implement Simple Linear Regression using Python	--	--	--	--	√
15	Write program to implement Multiple Linear Regression using Python	--	--	--	--	√
16	*Write program to create confusion matrix to calculate different measures to quantify the quality of the model	--	--	--	--	√

Guidelines to teachers

- Teacher should provide the guideline with demonstration of practical to the students with all features.
- Teacher shall explain prior concepts to the students before starting of each practical.
- Involve students in performance of each practical.
- Teacher should ensure that the respective skills and competencies are developed in the students after the completion of the practical exercise.
- Teachers should give opportunity to students for hands on experience after the demonstration.
- Teacher is expected to share the skills and competencies to be developed in the students.
- Teacher may provide additional knowledge and skills to the students even though not covered in the manual but are expected the students by the industry.
- Finally give practical assignment and assess the performance of students based on task assigned to check whether it is as per the instructions.

Instructions to Students

- Organize the work in the group and make record all programs.
- Students shall develop maintenance skill as expected by industries.
- Students shall attempt to develop related hand-on skills and gain confidence.
- Students shall develop the habits of evolving more ideas, innovations, skills etc. those included in scope of manual
- Students shall refer technical magazines.
- Students should develop habit to submit the practical on date and time.
- Students should be well prepared for viva while submitting write-up of exercise.
- Attach /paste separate papers wherever necessary.

Content Page

List of Practical and Formative Assessment sheet.

Sr. No	Title of the Practical	Date of performance	Date of Submission	Assessment Marks (50)	Dated sign of teacher	Remarks (if any)
1	*Install given Python IDE software and Python "scikit learn" for ML					
2	Write program to Implement Breadth First Search Algorithm (Uninformed) in Python					
3	*Write program to implement Depth First Search Algorithm (Uninformed) in Python					
4	Write program to implement Greedy Best-First (Informed Type) Search Algorithm in Python					
5	*Write program to implement A* search (Informed Type) Algorithm in Python					
6	*Write program to implement Bayes' Theorem					
7	Analyze the given Case study: How Turing test is performed between Responder and an Interrogator?					
8	*Explore different dataset finders e.g. Google Dataset Search, Kaggle					
9	*Write program in python to split any dataset into train and tests sets					

10	Analyze E-mail spam and non-spam filtering using Machine Learning through case study					
11	*Create and display a Decision Tree on given dataset					
12	Write program to implement K-means Algorithm					
13	*Write program to calculate cross validation score for any Dataset like IRIS					
14	*Write program to implement Simple Linear Regression using Python					
15	Write program to implement Multiple Linear Regression using Python					
16	*Write program to create confusion matrix to calculate different measures to quantify the quality of the model					
Total marks :						

These marks are to be transferred in pro-forma published by MSBTE.

- '*' Marked Practical (LLOs) are mandatory.
- Minimum 80% of above list of lab experiment are to be performed.
- Judicial mix of LLOs are to be performed to achieve desired outcomes.

Practical No. 1: Install given Python IDE software and Python "scikit learn" for ML

I. Practical Significance:

Python is a powerful, versatile, and easy-to-learn programming language widely used across various fields such as web development, data science, artificial intelligence, automation, ML and more. Its simple syntax, large standard library, and extensive community support make it ideal for both beginners and experienced developers. Python allows for rapid development and prototyping, integrates well with other technologies, and is freely available as open-source software. Its strong presence in cutting-edge areas like machine learning and data analysis further underscores its significance in modern programming.

Scikit-learn is a powerful and user friendly library that provides a wide range of tools for data preprocessing, model training, evaluation, and tuning all with a consistent and user-friendly interface. It supports numerous essential ML algorithms for tasks such as classification, regression, and clustering, allowing users to build models with just a few lines of code. The library is widely adopted in both academia and industry due to its strong performance, seamless integration with other Python libraries (like NumPy and Pandas), and support for real-world applications. Scikit-learn simplifies complex ML workflows, making it an ideal choice for beginners and professionals alike to experiment, prototype, and deploy machine learning models effectively.

II. Industry/Employer expected outcome(s):

- Install appropriate Python IDLE, pip and scikit learn library for various ML tasks.

III. Course Level Learning Outcome (COs):

- CO1- Implement relevant search algorithms as applicable to Artificial Intelligence.

IV. Laboratory Learning Outcome (LLO):

- LLO 1.1 - Install given IDE for python.

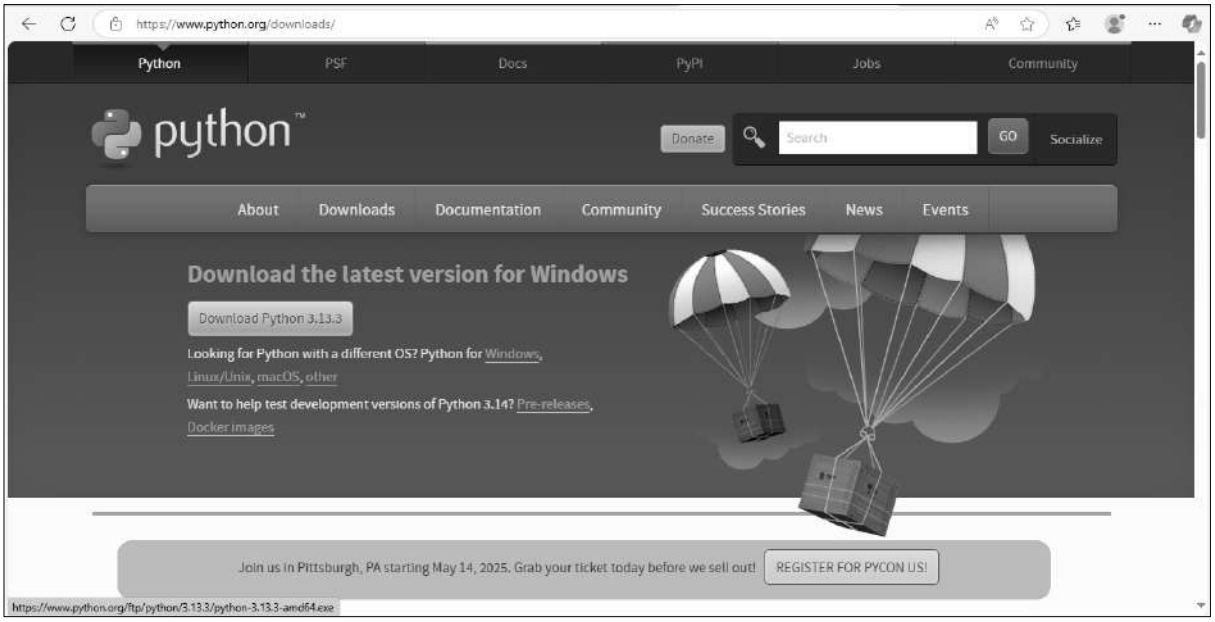
V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

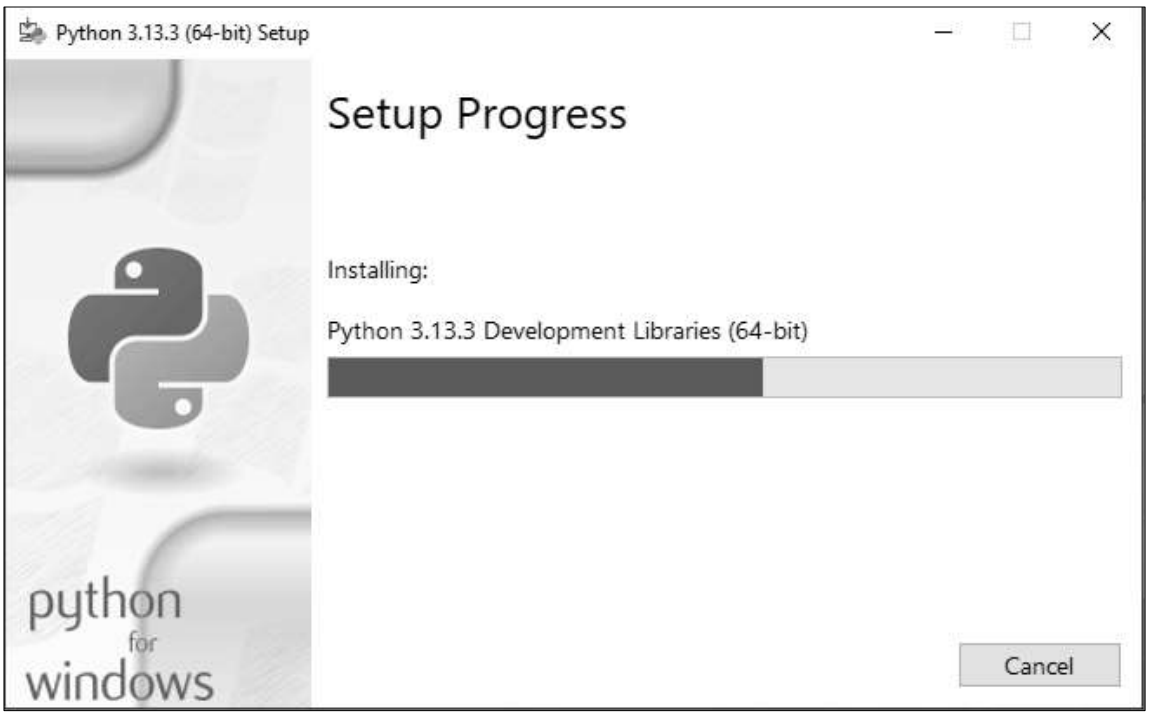
Python is a high-level, interpreted, general-purpose, open-source programming language. It was created by Guido van Rossum and first released in 1991. Python emphasizes code readability and simplicity, using significant indentation instead of braces to define code blocks. Python is the dominant language for AI and ML due to its readable syntax, large ecosystem of libraries such as NumPy, Pandas, TensorFlow, PyTorch, Scikit-learn and Integration with data visualization tools like Matplotlib, Seaborn etc. It allows quick prototyping and robust development of AI/ML models, making it popular in academia, research, and industry.

- **Installing Python:**
  - Visit the official page https://www.python.org/downloads/ for Python on the Windows operating system.
  - Download the installer, such as python-3.13.3-amd64.exe.

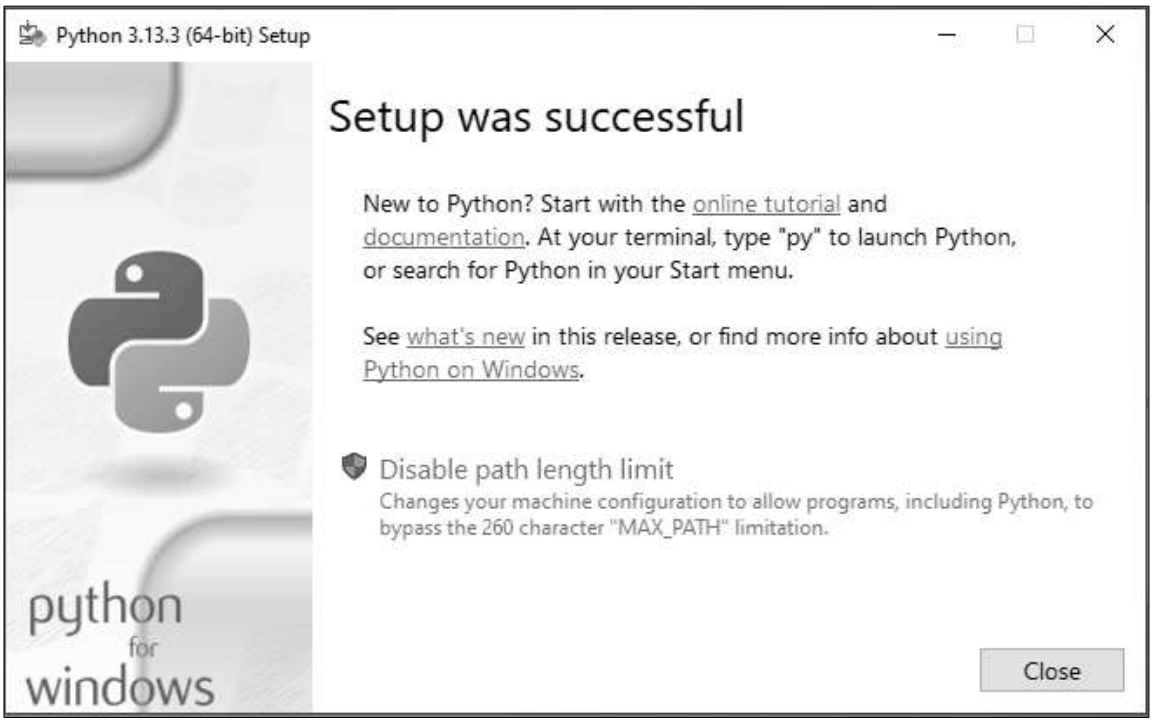


  - Open the above .exe file by double-clicking it to launch the Python installer.

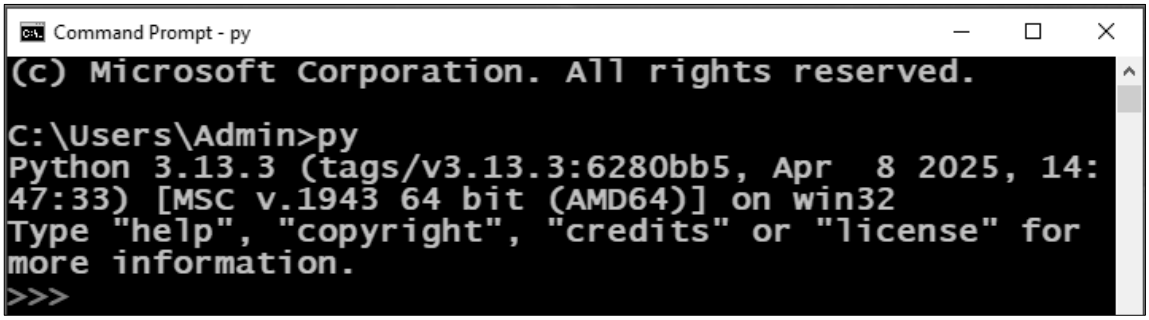
- Follow the prompts to complete the installation process.



- Close the window after successful installation of Python.



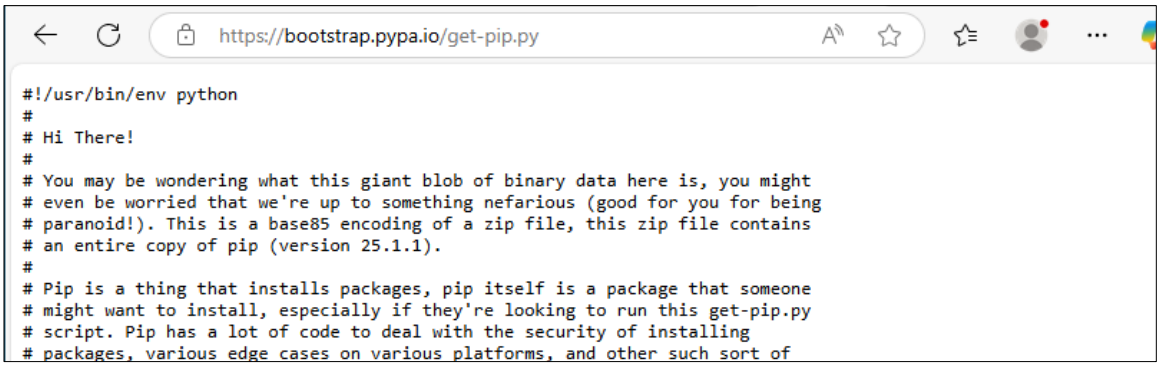
- Verify Python is installed on Windows by using **python --v** or **py** command on command prompt.



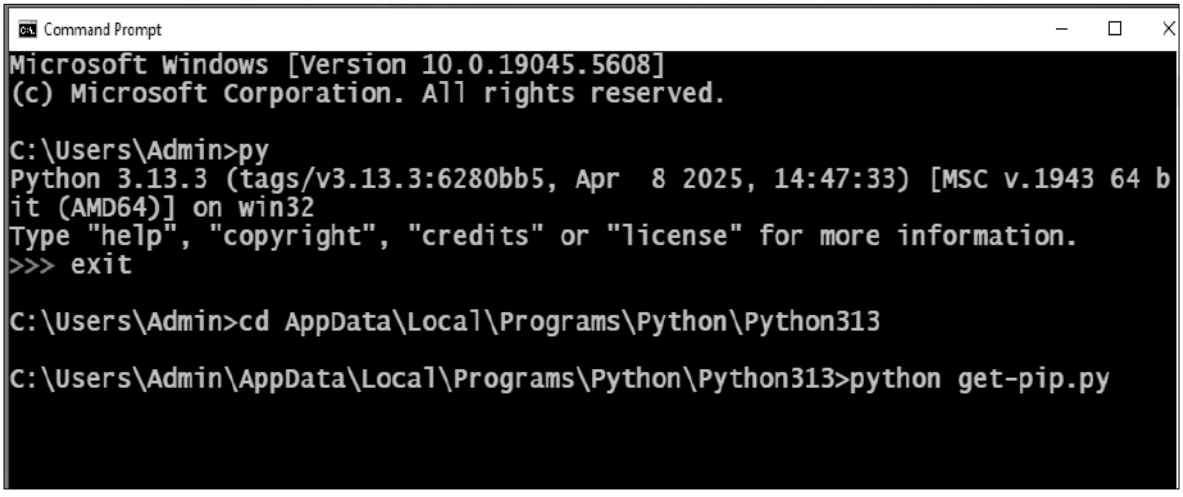
- The version of the python which you have installed will be displayed if the python is successfully installed on your windows.

Note: you can install any other Python Interpreter.

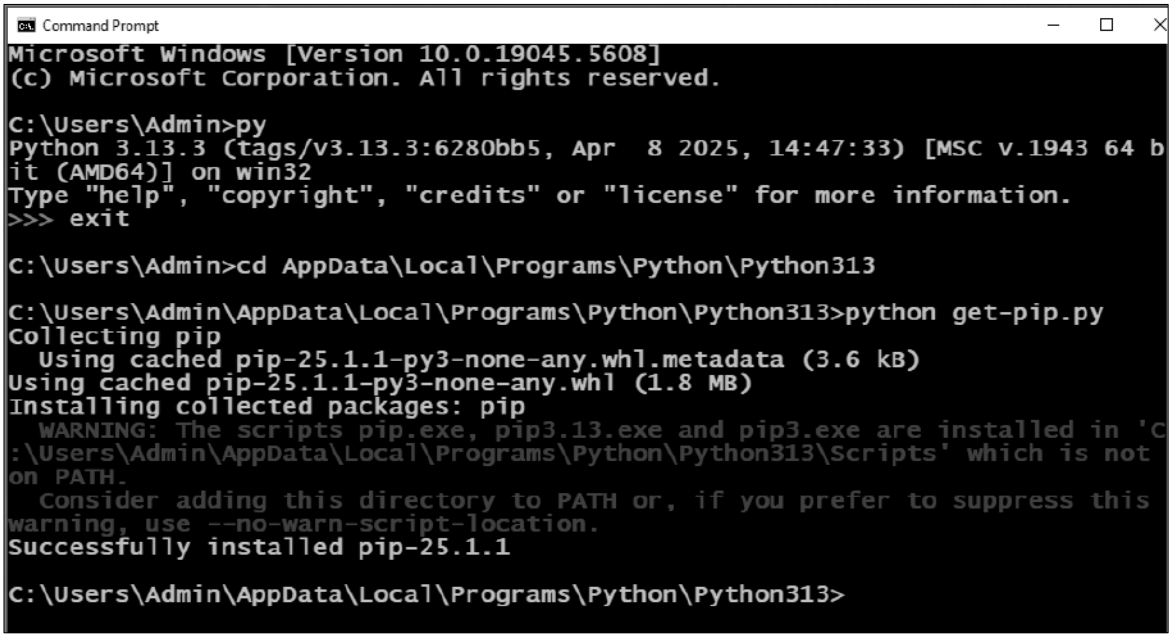
- **Install pip:**
  - Download the **get-pip.py** file from **https://bootstrap.pypa.io/get-pip.py** and store it in the same directory as Python is installed that is wherever **python.exe** is available.



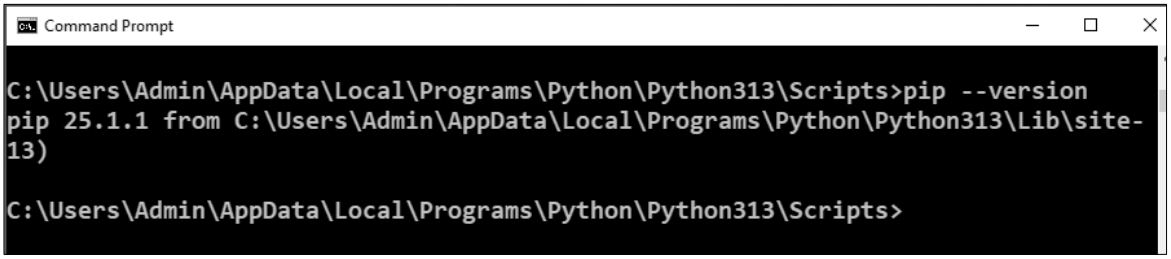
  - Change the current path of the directory in the command prompt to the path of the directory where the above file exists.
  - Execute the **python get-pip.py** command to Install pip



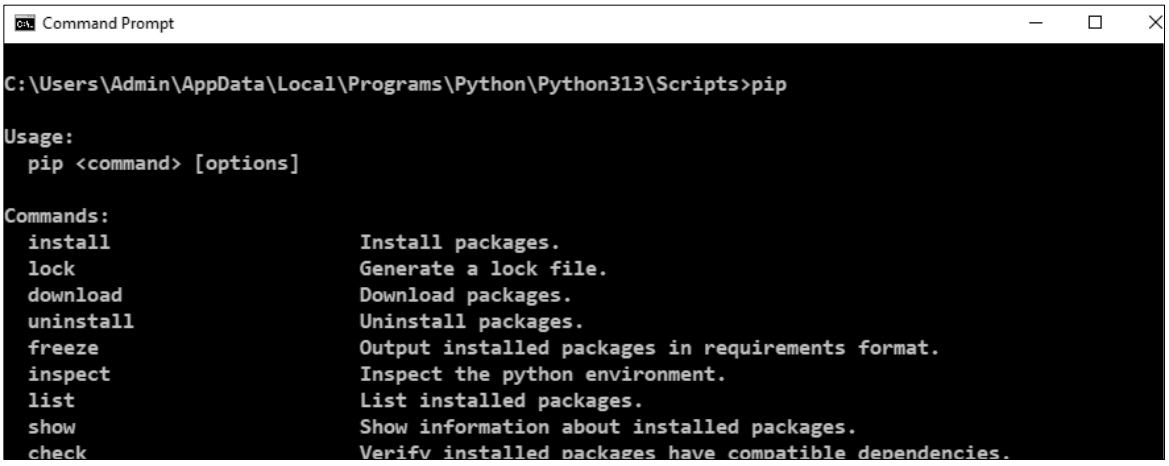
  - Now wait through the installation process of pip. It takes some time.



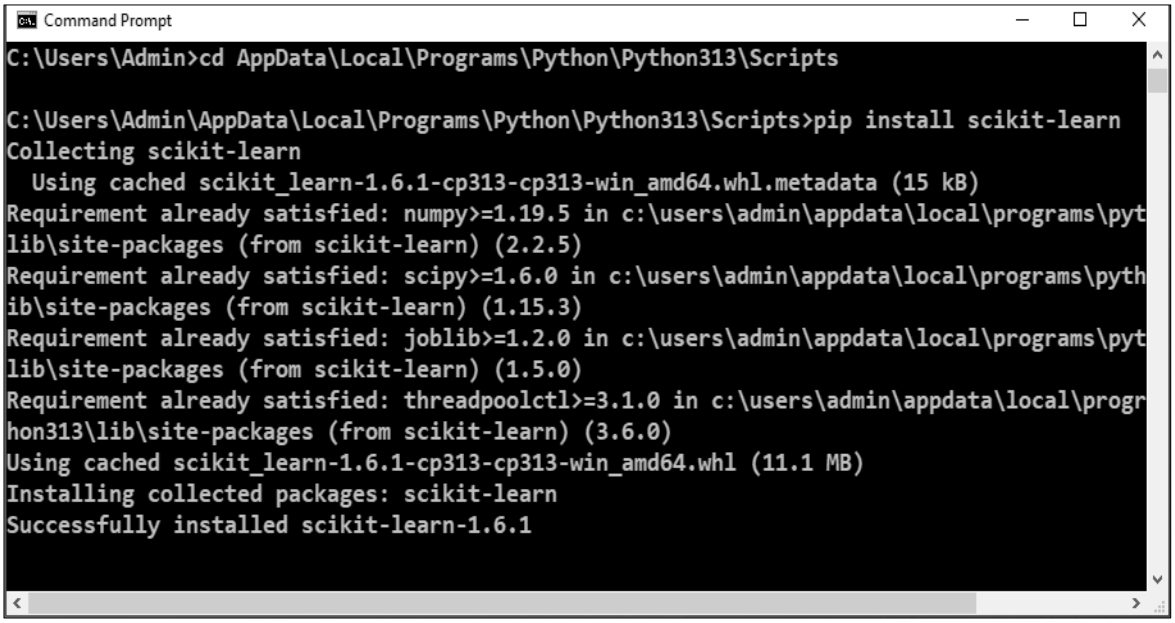
- Change the current path of the directory in the command prompt to the path of the directory where the **pip.exe** file exists.
- After completing above process, check the version of pip by using **pip –version** command.



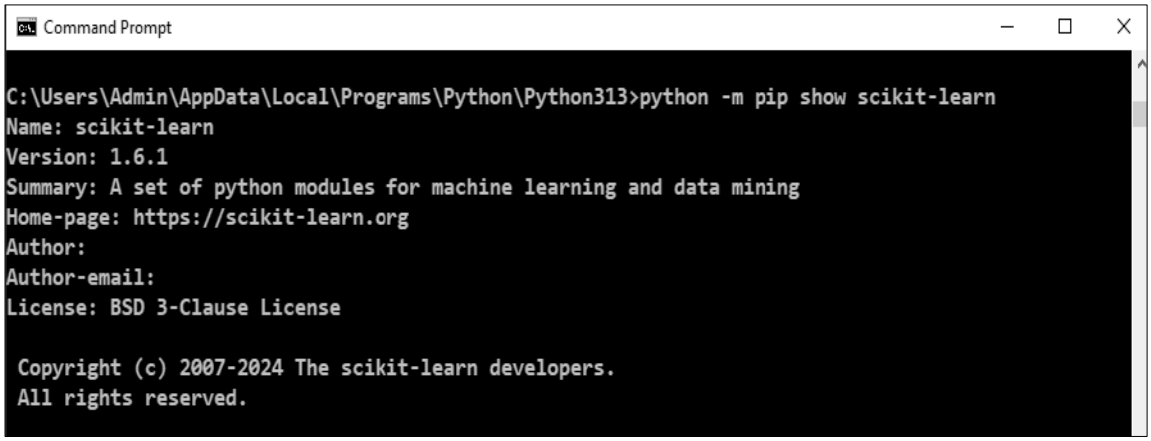
- You can use **pip** command to find syntax and other parameters for installing other packages or configuring them.



- **Installing scikit learn:**
  - Use **pip install scikit-learn** command to install scikit-learn library in python



  - Use **python -m pip show scikit-learn** to verify installation of scikit-learn library which will show following output.



VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

X. Practical related Questions:

(Note: Teacher shall assign any five questions from following or teacher can design more questions if required to achieve CO)

1. Write steps to install python IDLE on Linux.
2. Print the version of Python.
3. List key features of python.
4. What is the use of scikit learn library?
5. What are the Prerequisites for Installing Scikit-learn?
6. What are the applications of scikit learn library?
7. How to install pip? State its use.

Space for Answers

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

XI. References/Suggestions for further reading

- https://www.geeksforgeeks.org/how-to-install-python-on-windows/
- https://www.python.org/download/releases/2.5/msi/
- https://www.maketecheasier.com/set-up-Python-windows10
- https://pytutorial.com/how-to-install-scikit-learn-in-python-step-by-step/
- https://www.geeksforgeeks.org/how-to-install-scikit-learn-in-windows/
- https://www.youtube.com/watch?v=yivyNCtVVDk
- https://youtu.be/zPMr0lEMqpo
- https://www.youtube.com/watch?v=OOytKCeaNBo
- https://www.youtube.com/watch?v=C0zX9IXwgak

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		

Practical No. 2: Write program to Implement Breadth First Search Algorithm (Uninformed) in Python

I. Practical Significance:

Search algorithms are essential tools in computer science and artificial intelligence employed to solve problems that involve locating an item or a solution among a set of potential choices. Informed search algorithms employ supplementary data, such as heuristics or costs, to streamline the search process, resulting in increased efficiency. Uninformed search algorithms, also referred to as blind search algorithms, do not utilize any additional information about the desired outcome.

Breadth-first search (BFS) is an essential uninformed search algorithm that examines all nodes at the current depth level before progressing to nodes at the next depth level. It has the capability to locate the most efficient route in an unweighted graph or tree, making it particularly valuable for solving puzzles that require the least number of moves, planning routes on maps, and navigating social networks. BFS is a complete algorithm that guarantees the best possible solution when all step costs are equal, and its implementation using queues in Python makes it easy to understand and efficient for various applications in artificial intelligence and computer science.

II. Industry/Employer expected outcome(s):

- Search data through hierarchical or networked data.

III. Course Level Learning Outcome (COs):

- CO1 - Implement relevant search algorithms as applicable to Artificial Intelligence.

IV. Laboratory Learning Outcome (LLO):

- LLO 2.1- Implement Breadth First Search Algorithm.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

Breadth First Search (BFS) is an algorithm used to search a graph data structure for a node that meets a set of criteria. It is a vertex-based technique for finding a shortest path in a graph. BFS starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level. It uses a Queue data structure which follows first in first out. The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadthwise fashion. BFS is an uninformed search strategy that explores a graph or tree level by level, starting from the root node (or any arbitrary starting node). It was first formalized as part of general graph traversal techniques and is widely studied in computer science, especially in artificial intelligence and algorithms. The "uninformed" nature of BFS means it has no prior knowledge about the goal's location; instead, it explores all possible paths uniformly.

The algorithm uses a queue which is based on First In First Out (FIFO) strategy to manage the frontier of nodes to be explored. It expands the shallowest (closest to the starting point) unexpanded node first and checks for the goal condition at each expansion. BFS is complete, meaning it is guaranteed to find a solution if one exists. It is also optimal in the case of unweighted graphs, as it always finds the shortest path in terms of the number of edges.

- **The steps of the algorithm work are as follow:**
  1. Start by putting any one of the graph's vertices at the back of the queue.
  2. Now take the front item of the queue and add it to the visited list.
  3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
  4. Keep continuing steps two and three till the queue is empty.

Many times, a graph may contain two different disconnected parts and therefore to make sure that we have visited every vertex, we can also run the BFS algorithm at every node. In undirected graphs, each edge is bidirectional (A—B), so both directions are added. In directed graphs, edges have a direction (A → B), so only one direction is added. The algorithm logic remains largely the same, but the connectivity differs.

Now, we will see how the source code of the program for implementing breadth first search in python. You can create graphs in Python using dictionaries where keys represent nodes and values are lists of adjacent nodes. Consider the following graph in figure 1 which is implemented in the code below:

- **Python Code:**

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)
  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
```

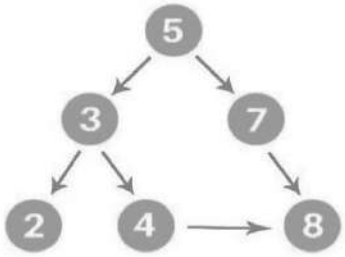


Fig. 1

```
            visited.append(neighbour)
            queue.append(neighbour)

    # Driver Code
    print("Following is the Breadth-First Search")
    bfs(visited, graph, '5')    # function calling

    #Output
    Following is the Breadth-First Search
    5 3 7 2 4 8
```

- **Example**

Let us see how this algorithm works with an example. Here, we will use an undirected graph with 5 vertices shown in fig 2.

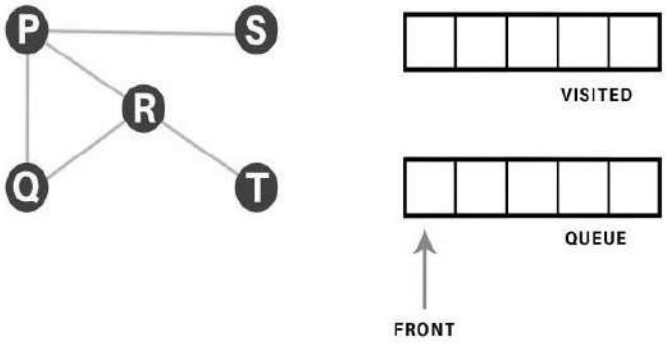


Fig. 2

We begin from the vertex P. Add P to queue shown in fig. 3(a) and next the BFS algorithmic program starts by putting P within the Visited list and puts all its adjacent vertices within the queue shown in fig 3(b).

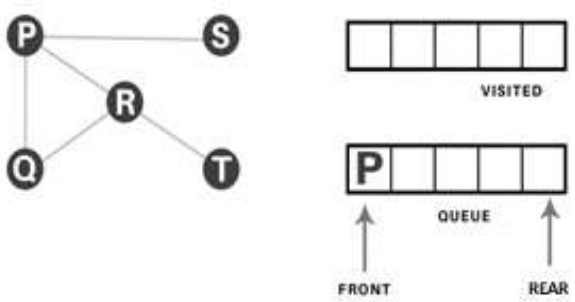


Fig. 3(a)

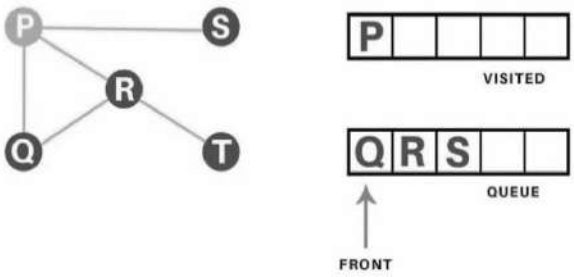


Fig. 3(b)

Next, we have to visit the part at the front of the queue i.e. Q and visit its adjacent nodes shown in fig 4. Since P has already been visited, we have to visit R instead. See the fig 5.

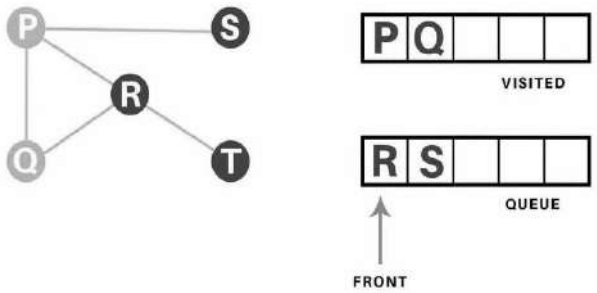


Fig. 4

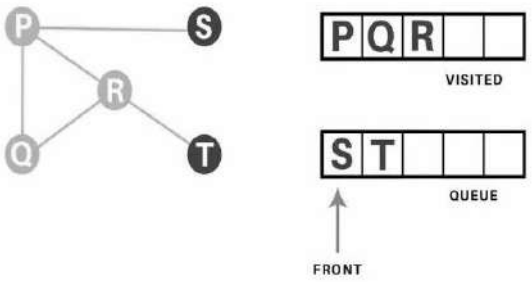


Fig. 5

Vertex R has an unvisited adjacent vertex in T, thus we have to add that to the rear of the queue and visit S, which is at the front of the queue as shown in fig 6.

Now, only T remains within the queue since the only adjacent node of S i.e. P is already visited. We have a tendency to visit it as per fig 6.

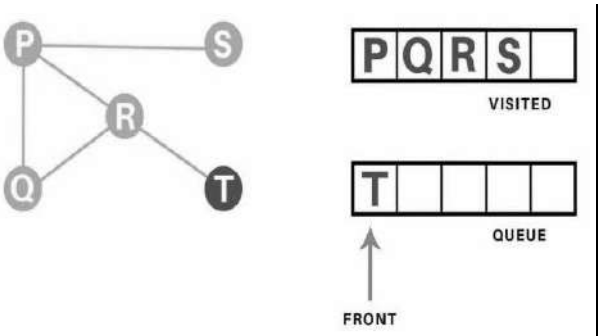


Fig. 6

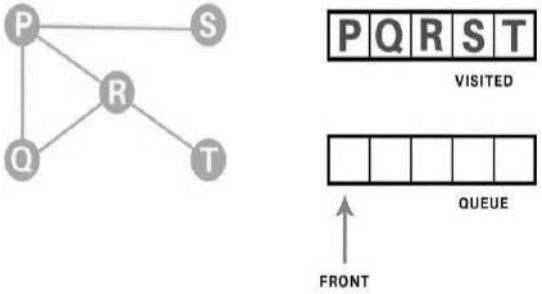


Fig. 7

Since the queue is empty, we've completed the Traversal of the graph and the output is PQRST as shown in fig 7.

- **Applications of BFS Algorithm**

  Breadth-first Search Algorithm has a wide range of applications in the real-world. Some of them are as discussed below:

  1. In GPS navigation, it helps in finding the shortest path available from one point to another.
  2. In pathfinding algorithms
  3. Cycle detection in an undirected graph
  4. In minimum spanning tree
  5. To build index by search index
  6. In Ford-Fulkerson algorithm to find maximum flow in a network.
  7. And most importantly, BFS is a foundational concept in many analytics and data roles.

  The time complexity of the Breadth First Search algorithm is in the form of O(V+E), where V is the representation of the number of nodes and E is the number of edges. Also, the space complexity of the BFS algorithm is O(V). In Python, BFS is commonly implemented using a queue from the collections module (e.g. collections.deque), and often involves managing a visited set to avoid redundant node expansions and infinite loops in cyclic graphs.

  In summary, Breadth-First Search (BFS) is a fundamental algorithm used for traversing or searching graph structures. It starts from a given node and systematically explores all its

neighbors at the present depth level before moving on to the nodes at the next level. The Breadth First Search algorithm is a popular and powerful technique for traversing graphs in a breadth-first manner. It is widely used in various fields such as network routing, social network analysis, and web crawling.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

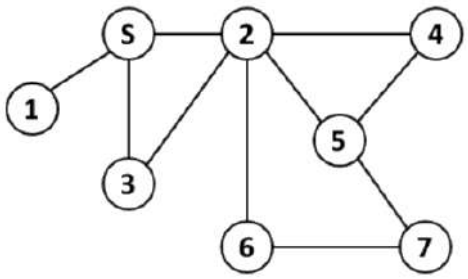
_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any five questions from following or teacher can design more questions if required to achieve CO)*

1. What is Breadth First Search (BFS) Algorithm?
2. How does BFS work?
3. List applications of BFS.
4. Implement the BFS algorithm using collections module and write its output.
5. Write the use of BFS algorithm.
6. Write the advantages and disadvantages of BFS.
7. Use the following graph and write the output for the node 1.
   graph = { 1: [2, 4, 5], 2: [3, 6, 7], 3: [], 4: [], 5: [], 6: [], 7: [] }
8. Write the order of performing BFS traversal on the following graph with S as the starting point.



Space for Answers

XI. References/Suggestions for further reading

- https://coderivers.org/blog/bfs-in-python/
- https://favtutor.com/blogs/breadth-first-search-python
- https://codereview.stackexchange.com/questions/135156/bfs-implementation-in-python-3
- https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
- https://youtu.be/tBc_IvCgXeo
- https://algotree.org/algorithms/tree_graph_traversal/breadth_first_search/
- https://www.colabcodes.com/post/implementing-breadth-first-search-bfs-in-python

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		

Practical No. 3: Write program to Implement Depth First Search Algorithm (Uninformed) in Python

I. Practical Significance:

Depth First Search (DFS) is an uninformed search algorithm significant for its ability to deeply explore graphs or trees using minimal memory, making it ideal for problems like puzzle solving, pathfinding, and graph analysis. In Python, it's easy to implement with recursion or a stack and serves as a foundation for advanced algorithms such as cycle detection and topological sorting. Though not guaranteed to be complete or optimal, DFS is widely used in AI, network analysis, and file system traversal due to its efficiency in exploring large or deep structures.

II. Industry/Employer expected outcome(s):

- Search data through hierarchical or networked data.

III. Course Level Learning Outcome (COs):

- CO1 - Implement relevant search algorithms as applicable to Artificial Intelligence.

IV. Laboratory Learning Outcome (LLO):

- LLO 3.1- Develop Depth First Search Algorithm.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

Depth first search (DFS) is a fundamental algorithm employed for traversing or searching through graph and tree data structures. It plays a crucial role in artificial intelligence (AI) for problem-solving and pathfinding tasks. By thoroughly exploring each branch before retracing steps, DFS mirrors the way humans tackle puzzles or games, often starting from the beginning and working their way through. Search algorithms such as dfs are crucial in artificial intelligence, enabling machines to traverse decision trees, identify cycles in graphs, or tackle intricate problems represented as graphs. DFS's simplicity and efficiency make it a commonly employed technique in artificial intelligence applications, including navigation systems, network analysis, and game-solving strategies.

Depth First Search (DFS) explores each branch of a graph or tree as deeply as possible before backtracking to examine unexplored branches The depth-first approach is highly effective in solving problems that require exploring paths one at a time, making DFS a valuable tool in such scenarios. Depth-first search, or DFS, is a method used to navigate through the graph. Initially, it enables visiting vertices of the graph, but there are numerous algorithms for graphs, which are based on DFS. Consequently, comprehending the fundamentals of depth-first search is crucial for progressing in the field of graph theory. The basic principle of the algorithm is

straightforward: to proceed forward (in-depth) when there is a possibility, otherwise to retrace our steps.

- **The steps of the algorithm work are as follow:**
  1. Add the starting node to the stack and mark it as visited.
  2. Pop the top node from the stack and visit it.
  3. Add all adjacent nodes of the current node to the stack (excluding nodes that have already been visited).
  4. Repeat this process until the stack is empty.

DFS primarily uses a stack, which follows a Last In, First Out (LIFO) order. The stack helps manage the exploration order in DFS. To avoid revisiting the same nodes, we need to track whether a node has been visited. This can be done using a set or list.

Now, we will see how the source code of the program for implementing Depth first search in python. Consider the following fig 1 which is implemented in the code below:

- **Python Code: Recursive Method**

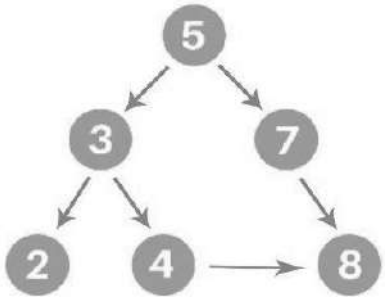


Fig. 1

```
# Using a Python dictionary to act as an adjacency list
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
visited = set() # Set to keep track of visited nodes of graph.
def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')

#Output
Following is the Depth-First Search
5 3 2 4 8 7
```

- **Example**
  Let us see how the DFS algorithm works with an example. Here, we will use an undirected graph with 5 vertices as shown in fig 2.

  We begin from the vertex P, the DFS rule starts by putting it within the Visited list and putting all its adjacent vertices within the stack as per fig 3.

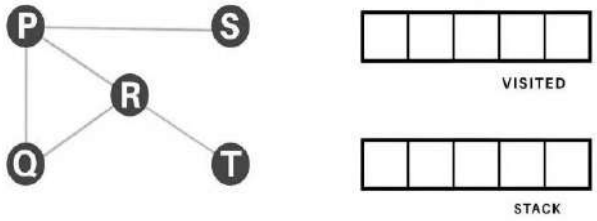


Fig. 2

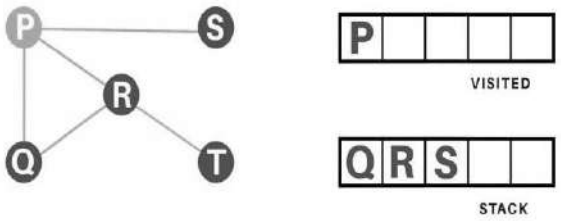


Fig. 3

Next, we tend i.e. Q, and mark it as visited. We proceed with its adjacent nodes of Q. Since Q has been visited already, we tend to visit R instead as shown in fig 4.

Vertex R has the unvisited adjacent vertex in T, therefore we will be adding that to the highest of the stack and visit it as shown in fig 5.

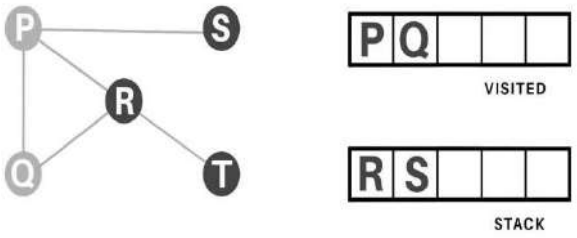


Fig. 4

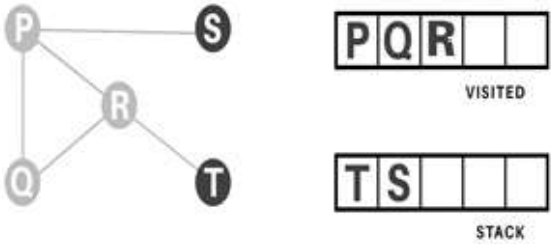


Fig. 5

At last, we will visit the last component S shown in fig 6, it does not have any unvisited adjacent nodes, thus we've completed the Depth First Traversal of the graph and the output is PQRTS which is shown in fig 7.

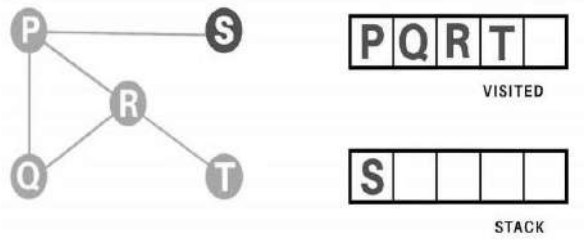


Fig. 6

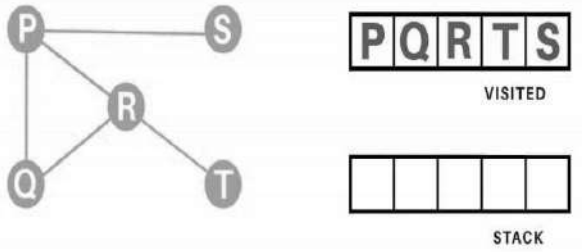


Fig. 7

- **Applications of DFS Algorithm**
    1. **Topological Sorting:** DFS is used to perform topological sorting of a Directed Acyclic Graph (DAG). This is helpful for organizing tasks, compiling programs, and managing dependencies.
    2. **Cycle Detection:** DFS can help detect cycles in a graph. If, during DFS, we encounter a node that has already been visited and is still on the stack, it indicates the presence of a cycle.
    3. **Connected Components:** DFS can be used to find connected components in an undirected graph by performing DFS starting from an unvisited node and marking all reachable nodes.
    4. **Pathfinding and Maze Exploration:** DFS is used in solving maze puzzles, where it explores all possible paths deeply and backtracks when it reaches dead ends.
    5. **Solving Puzzles:** DFS can be used to solve puzzles like Sudoku, N-Queens, and others by exploring all possible configurations and backtracking when necessary.

    The time complexity of DFS is O(V+E), where v represents the number of vertices in the graph and e represents the number of edges. This is because DFS processes each vertex and edge exactly once. The space complexity is O(V), as DFS requires storage for the stack and a visited list, both of which require space proportional to the number of vertices in the graph. DFS is highly efficient in handling both sparse and dense graphs, making it a versatile tool for solving a wide range of graph-related problems.

- **Breadth First Search (BFS) vs. Depth-First Search (DFS)**

    BFS is more effective in finding the shortest path in a graph without any weights, while DFS is more memory-efficient and can be more suitable for tasks like sorting a topological order or solving puzzles.

    In summary, Depth First Search (DFS) is a crucial algorithm in artificial intelligence and computer science, extensively employed for traversing graphs and trees. Its depth-first approach allows for efficient exploration of nodes, making it well-suited for solving problems related to pathfinding, cycle detection, and graph analysis.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

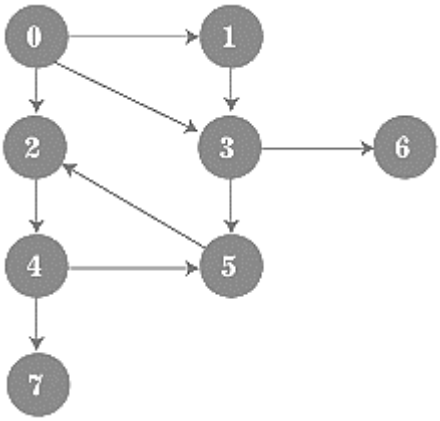
_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any five questions from following or teacher can design more questions if required to achieve CO)*

1. What is Depth First Search (DFS) Algorithm?
2. How does DFS work?
3. List applications of DFS.
4. Implement the DFS algorithm using stack.
5. Compare BFS and DFS.
6. Write the advantages and disadvantages of DFS.
7. Use the following graph and write the output of DFS Traversal from the node ‘A’.
   Graph = { 'B': ['A', 'D', ‘C’], 'A': ['B', 'C'], 'C': ['A', ‘B’, ‘D’, 'E'], 'D': ['B', ‘C’, ‘’E’, ‘F’], 'E': ['C', ‘D’], 'F': ['D'] }
8. Use the following graph and write the output for the node ‘A’.
   Graph = { 'A': ['B', 'C'], 'B': ['A', 'D', 'E'], 'C': ['A', 'F'], 'D': ['B'], 'E': ['B'], 'F': ['C'] }
9. Find the DFS for following graph.



Space for Answers

XI. References/Suggestions for further reading

- https://www.appliedaicourse.com/blog/depth-first-search-in-artificial-intelligence/
- https://favtutor.com/blogs/depth-first-search-python
- https://www.tutorialspoint.com/graph_theory/graph_theory_depth_first_search.htm
- https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
- https://www.educba.com/dfs-algorithm/
- https://youtu.be/f8luGFRtshY

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		

Practical No. 4: Write program to implement Greedy Best-First (Informed Type) Search Algorithm in python

I. Practical Significance:

Greedy Best-First Search is a search method often used in artificial intelligence and pathfinding to locate a path between two points in a graph or tree. It belongs to the category of informed search algorithms because it uses a heuristic function to guess how close a given node is to the goal. At each step, the algorithm picks the node that seems closest to the goal based on this estimate. This "greedy" choice focuses only on immediate progress and ignores the total path cost from the starting point. As a result, it can sometimes make poor decisions, getting stuck in areas that look good at first but don't lead to the best outcome.

II. Industry/Employer expected outcome(s):

- Search data through hierarchical or networked data.

III. Course Level Learning Outcome (COs):

- CO1- Implement relevant search algorithms as applicable to Artificial Intelligence.

IV. Laboratory Learning Outcome (LLO):

- LLO 4.1- Implement Greedy Best-First Search Algorithm.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

Greedy best-first search is an informed search algorithm where the evaluation function is strictly equal to the heuristic function, disregarding the edge weights in a weighted graph because only the heuristic value is considered. In order to search for a goal node it expands the node that is closest to the goal as determined by the heuristic function. This approach assumes that it is likely to lead to a solution quickly. However, the solution from a greedy best-first search may not be optimal since a shorter path may exist.

In this algorithm, search cost is at a minimum since the solution is found without expanding a node that is not on the solution path. This algorithm is minimal, but not complete, since it can lead to a dead end. It's called "Greedy" because at each step it tries to get as close to the goal as it can with minimal cost.

Best-first search offers advantages over other search algorithms but also has limitations. The effectiveness of BFS depends heavily on the quality of the heuristic function used.

- **What is a Heuristic Function?**

  A heuristic function (or simply a heuristic) is an approach used to find solutions more quickly by making educated guesses or approximations. It is especially useful when finding the exact solution is too time-consuming or not possible. Heuristics are commonly evaluated based on two important properties: admissibility and consistency.
  Admissibility: A heuristic is considered admissible if it never overestimates the actual cost of reaching the goal. This means the estimated cost must always be less than or equal to the true minimum cost required to get to the goal.
  Consistency: A heuristic is consistent (or monotonic) if, for every node, its estimated cost to reach the goal is no greater than the cost of reaching a neighboring node plus the estimated cost from that neighbor to the goal. In simple terms, it ensures that the heuristic is logically stable as the search progresses through the graph.

- **The steps of the algorithm work are as follow:**

  The algorithm of the Greedy Best First Search Algorithm is as follows -

  1. Define two empty lists of nodes (let them be openList and closeList).
  2. Insert src node in the openList.
  3. Iterate while the open list is not empty, and do the following -
     a. Find the node with the least h value that is present in openList (let it be node).
     b. Remove the node from the openList, and insert it in the closeList.
     c. Check for all the adjacents nodes of the node, if anyone of them equals the dest. If yes, then terminate the search process as we've reached the destination.
     d. Otherwise, find all the adjacents of the node that are neither present in openList nor closeList and insert them in the openList.
  4. If we have reached here, it means there is no possible path between src and dest.

  To decide which path is the most promising we will use the heuristics function (say $h(n)$) where $h(i)$ denotes the estimated cost of reaching the destination node from the *ith* node. Note that, the value of *h(dest)* will be 0 because no cost is incurred to reach the destination if we are already there.

- **Implementation of GBFS Algorithm Example-1:**
  An example of the best-first search algorithm is below graph shown in figure 1, suppose we have to find the path from A to G.

  1. We are starting from A, so from A there are direct path to node B (with heuristics value of 32), from A to C (with heuristics value of 25) and from A to D (with heuristics value of 35) as shown in Figure 1.
  2. So as per best first search algorithm choose the path with lowest heuristics value, currently C has lowest value among above node. So we will go from A to C.
  3. Now from C we have direct paths as C to F(with heuristics value of 17) and C to E(with heuristics value of 19) , so we will go from C to F.
  4. Now from F we have direct path to go to the goal node G (with heuristics value of 0) , so we will go from F to G.

5. So now the goal node G has been reached and the path we will follow is A→C→F→G.

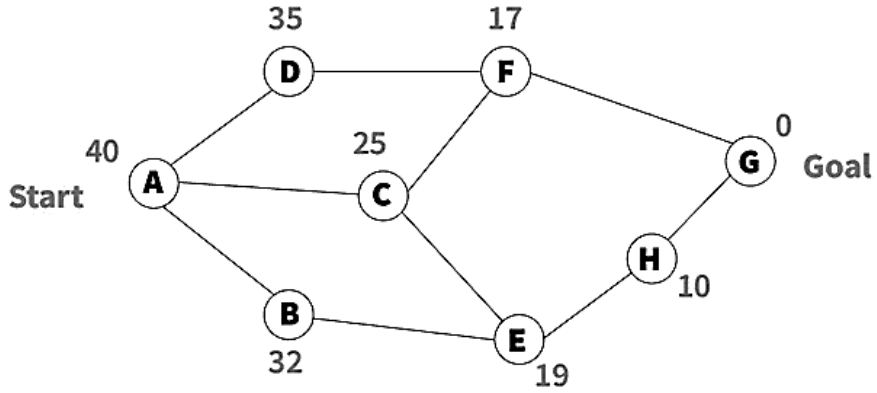


Fig. 1

- **Python Code for above example:**

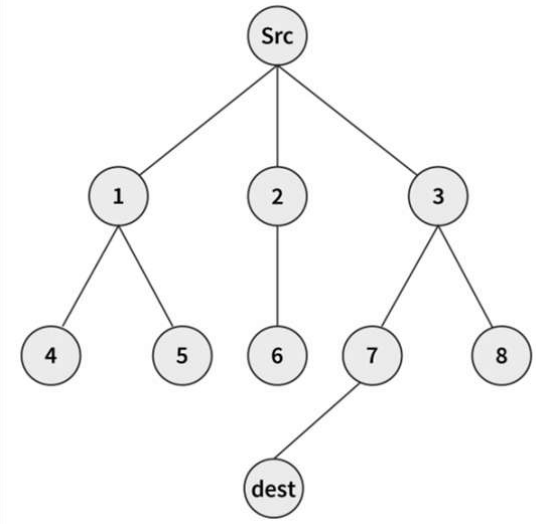
```
import heapq

def greedy_best_first_search(graph, start, goal):
    visited = set()

    # Priority queue ordered by heuristic
    priority_queue = []
    heapq.heappush(priority_queue, (0, start))  # Assuming h(start) = 0 or unknown
    parent = {start: None}

    while priority_queue:
        h, current = heapq.heappop(priority_queue)
        print(f"Visiting: {current} (h={h})")
        if current == goal:
            return reconstruct_path(parent, start, goal)
        visited.add(current)

        for neighbor, heuristic in graph[current]:
            if neighbor not in visited:
                heapq.heappush(priority_queue, (heuristic, neighbor))
                if neighbor not in parent:
                    parent[neighbor] = current

    return None  # No path found
def reconstruct_path(parent, start, goal):
    path = []
    current = goal
    while current:
        path.append(current)
        current = parent[current]
    path.reverse()
    return path
graph = {
    'A': [('B', 32), ('C', 25), ('D', 35)],
    'B': [('E', 32)],
```

```
    'C': [('E', 19), ('F', 17)],
    'D': [('F', 35)],
    'E': [('H', 10)],
    'F': [('G', 0)],
    'H': [('G', 10)],
    'G': []
}
path = greedy_best_first_search(graph, 'A', 'G')
print("Path found:", path)
```

#Output

```
Visiting: A (h=0)
Visiting: C (h=25)
Visiting: F (h=17)
Visiting: G (h=0)
Path found: ['A', 'C', 'F', 'G']
```

- **Example-2:**
  Let's say we have the following graph, in which src and dest nodes are as marked and heuristic values for each of the nodes are written against them in the form of the table shown in Fig.2



Src	20
1	22
2	21
3	10
4	25
5	24
6	30
7	5
8	12
dest	0

Fig. 2

1. Step 1 - Initialize two empty lists i.e. openList and closeList.
2. Step 2- Insert src in the openList, after which we have - openList = [src] closeList = []
3. Step 3 - (Iterating while the open list is not empty) -
   i. Iteration 1 - After this operation we will have
   openList = [1, 2, 3] closeList = [src]
   ii. Iteration 2 - After this operation we will have
   openList = [1, 2, 7, 8] closeList = [src, 3]

iii. Iteration 3 - Upon exploring the adjacents of node 7 we found that dest is one of them. Hence, we return that search result in success, and print the path src→3→7→dest.

- **Applications of GBFS Algorithm**
  1. **Path finding:** Greedy Best-First Search is used to find the shortest path between two points in a graph. It is used in many applications such as video games, robotics, and navigation systems.
  2. **Machine Learning:** Greedy Best-First Search can be used in machine learning algorithms to find the most promising path through a search space.
  3. **Optimization:** Greedy Best-First Search can be used to optimize the parameters of a system in order to achieve the desired result.
  4. **Game AI:** Greedy Best-First Search can be used in game AI to evaluate potential moves and chose the best one.
  5. **Navigation:** Greedy Best-First Search can be use to navigate to find the shortest path between two locations.
  6. **Natural Language Processing:** Greedy Best-First Search can be use in natural language processing tasks such as language translation or speech recognisation to generate the most likely sequence of words.
  7. **Image Processing:** Greedy Best-First Search can be use in image processing to segment image into regions of interest.

In the worst situation, we may require to traverse through all the edges to reach the destination. Hence in the worst case, the time complexity will be O(ElogV). In the worst scenario, we may have all the vertices (nodes) in *openList* which will make the worst case space complexity O(V).

In summary, Greedy Best-First Search is an informed search algorithm that relies on heuristic information to guide the search process. Greedy Best-First Search is suitable when the heuristic provides a good estimate, and the lack of optimality is acceptable. It serves as a valuable tool in scenarios where computational resources are a significant concern, and a quick, heuristic-driven solution is more important than finding the absolute best solution.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

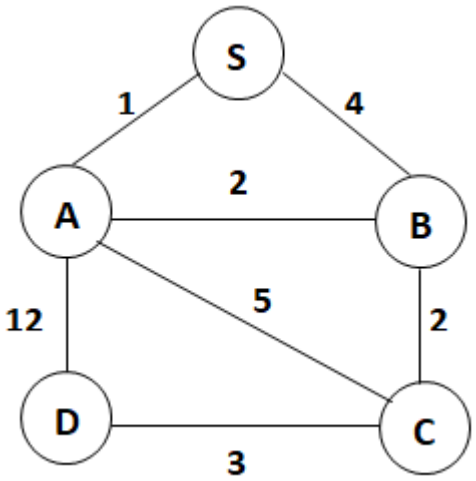
_____

X. Practical related Questions:

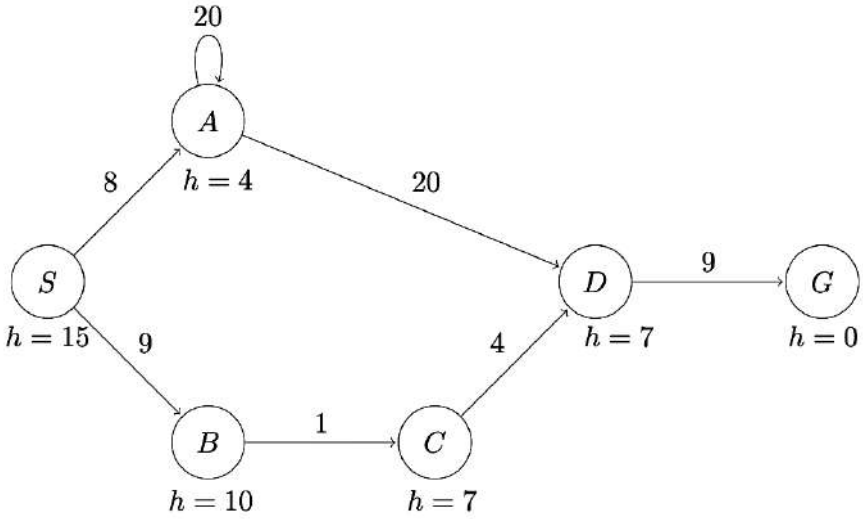
(*Note: Teacher shall assign any five questions from following or teacher can design more questions if required to achieve CO)*

1. Write a function to implement Greedy Best-First Search on a graph using a priority queue. (Use a dictionary to represent the graph and a separate dictionary for heuristic values.)
2. What data structures are essential for implementing GBFS? Why?
3. How does GBFS differ from BFS in terms of data structure usage and node selection?
4. Explain how the heuristic function influences the GBFS algorithm. Provide an example.
5. Write Python code for the implementation of graph with heuristic values shown in figure 2 of example 2.
6. Apply GBFS on below graph and find minimum path from S to D. describe this approach calculation steps and solution.

Heuristic Value H(SLD)	S	A	B	C	D
	7	2	4	1	0



7. Consider the following graph and heuristic values. Find the order in which nodes are expanded with S as start state and G as goal state. Also show the path from S to G and its cost using GBFS. Is the heuristic admissible at nodes B and C? Yes/No and why?



Space for Answers

XI. References/Suggestions for further reading

- https://python.algorithmexamples.com/web/graphs/greedy_best_first.html/
- https://www.tutorialsarena.com/datascience/ai/best-first-search-algorithm-in-artificial-intelligence
- https://www.codecademy.com/resources/docs/ai/search-algorithms/greedy-best-first-search
- https://medium.com/@2200032841cseh/informed-search-greedy-best-first-search-b0ea599aa718
- https://www.scaler.com/topics/best-first-search/
- https://youtu.be/Uzraf9TYhjY

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		

Practical No. 5: Write program to implement A* search (Informed Type) Algorithm in Python

I. Practical Significance:

Using A* Search algorithm in Python is highly valuable for solving real-world pathfinding problems efficiently. This algorithm blends the strengths of Uniform-Cost Search and Greedy Best-First Search by considering both the actual cost to reach a node (g(n)) and the estimated cost to reach the goal (h(n)). When the heuristic used is admissible, A* guarantees finding the best and most accurate path. It is widely applied in areas like GPS systems, game development, robot navigation, and network routing due to its strong balance between speed and accuracy. Implementing A* in Python not only shows a solid grasp of heuristic-driven techniques but also prepares developers to tackle complex, real-time decision-making tasks efficiently.

II. Industry/Employer expected outcome(s):

- Reduce computation time, ensure shortest or most cost-effective paths in real-time systems.

III. Course Level Learning Outcome (COs):

- CO1- Implement relevant search algorithms as applicable to Artificial Intelligence.

IV. Laboratory Learning Outcome (LLO):

- LLO 5.1- Develop A* search Algorithm.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

Graph traversal algorithms play a key role in various fields of computer science, including game design, robotics, and network systems. These algorithms are used to move through and analyze graphs, which are structures made up of nodes (also called vertices) connected by edges. One of the most effective algorithms for pathfinding is the A* (A-Star) algorithm. It is an informed search method, meaning it uses a heuristic function to estimate how close a particular node is to the goal. This helps the algorithm choose better paths and skip exploring areas that are unlikely to lead to an optimal solution.

The A* algorithm is said to be admissible if it always finds the shortest or most efficient path when such a path exists. This happens when the heuristic function (denoted as h) never overestimates the actual cost from the current node to the goal. In other words, if the heuristic always gives a value that is less than or equal to the real cost, the algorithm is guaranteed to return the best possible path. Therefore, with an admissible heuristic, A* ensures optimal results every time.

- **The steps of the algorithm work are as follow:**
  The A* algorithm is best suited for pathfinding problems in graphs and grids, where you need to find the shortest path between two points. It combines features of both uniform-cost search and pure heuristic search to efficiently compute optimal solutions.
  1. Initialize open and closed list of nodes
  2. Set starting node as current node
  3. Loop until solution found:
     a. Consider current node's neighbors
     b. Add neighbors to open list
     c. Set f-cost for each neighbor (g+h)
     d. Sort open list by lowest f-cost
     e. Set lowest f-cost node as current node
     f. Move current node to closed list
  4. Reconstruct path by traversing pointers backwards from goal

  Where:
  - g = cost to move from starting point to current node
  - h = estimated cost to move from current node to goal (heuristic)
  - f = total estimated cost of path through node (f=g+h)

  By leveraging heuristics, A* can find optimal solutions faster than brute force methods.

- **Implementation of A* Algorithm:**

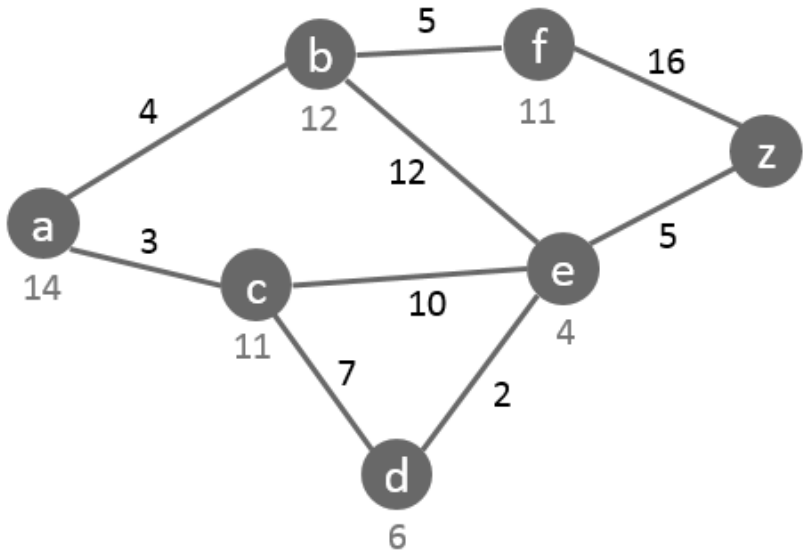


Fig. 1

Here are the step-by-step algorithmic steps of the A* algorithm to find the shortest path from node **a** to node z using the graph provided in figure 1.

- **g(n):** Actual cost from start node to current node n
- **h(n):** Heuristic estimate from node n to the goal
- **f(n) = g(n) + h(n):** Estimated total cost from start to goal through n

  Heuristic values ar: h(a)=14, h(b)=12, h(c)=11, h(d)=6, h(e)=4, h(f)=11, h(z)=0

i. **Start at node a**
g(a) = 0
f(a) = g(a) + h(a) = 0 + 14 = 14
Open list: [a]

ii. **Expand a: neighbors = b (cost 4), c (cost 3)**
g(b) = 0 + 4 = 4 → f(b) = 4 + 12 = 16
g(c) = 0 + 3 = 3 → f(c) = 3 + 11 = 14
Add b, c to open list
Open list: [c (f=14), b (f=16)]

iii. **Expand c (lowest f=14)**
g(d) = 3 + 7 = 10 → f(d) = 10 + 6 = 16
g(e) = 3 + 10 = 13 → f(e) = 13 + 4 = 17
Add d, e
Open list: [b (f=16), d (f=16), e (f=17)]

iv. **Expand b (f=16)**
g(f) = 4 + 5 = 9 → f(f) = 9 + 11 = 20
g(e) = 4 + 12 = 16 → (existing g(e)=13, so ignore this longer path)
Add f
Open list: [d (f=16), e (f=17), f (f=20)]

v. **Expand d (f=16)**
g(e) = 10 + 2 = 12 → f(e) = 12 + 4 = 16 (better than previous g(e)=13, update e)
Update e with path: a → c → d → e
Open list: [e (f=16), f (f=20)]

vi. **Expand e (f=16)**
g(z) = 12 + 5 = 17 → f(z) = 17 + 0 = 17
Add z
Open list: [z (f=17), f (f=20)]

vii. **Expand z (goal node)**
Goal reached with cost 17
Path: a → c → d → e → z

Final Shortest Path: a → c → d → e → z

Total Cost: 3 (a→c) + 7 (c→d) + 2 (d→e) + 5 (e→z) = 17

- **Python Code for above example:**

```
def astaralgo(start_node,stop_node):
   open_set = set(start_node)
   closed_set= set()
   g={}
   parents = {}
   g[start_node] = 0
   parents[start_node]  = start_node
   while len(open_set) > 0:
      n=None
```

```
            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):
                    if m not in open_set  and m not in closed_set:
                            open_set.add(m)
                            parents[m]=n
                            g[m] = g[n] + weight
                    else:
                            if g[m] > g[n] +  weight:
                                g[m] = g[n] + weight
                                parents[m]=n
                                if m in closed_set:
                                    closed_set.remove(m)
                                    open_set.add(m)
            if n==None:
                print("path does not exist!")
                return None
            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
            open_set.remove(n)
            closed_set.add(n)
        print("Path does not exist!")
        return None
    def get_neighbors(v):
        if v in Graph_nodes:
            return Graph_nodes[v]
        else:
            return None

    def heuristic(n):
        H_dist = {
            'a': 14,
            'b': 12,
            'c': 11,
            'd': 6,
            'e': 4,
            'f': 11,
            'z': 0
            }
        return H_dist[n]
```

```
Graph_nodes = {
   'a': [('b', 4), ('c', 3)],
   'b': [('a', 4), ('f', 5), ('e', 12)],
   'c': [('a', 3), ('d', 7), ('e', 10)],
   'd': [('c', 7), ('e', 2)],
   'e': [('c', 10), ('d', 2), ('b', 12), ('f', 16), ('z', 5)],
   'f': [('b', 5), ('e', 16), ('z', 16)],
   'z': [('e', 5), ('f', 16)]
}
astaralgo('a','z')
```

#Output
Path found: [‘a’, ‘c’, ‘d’, ‘e’, ‘z’]

- **Applications of A* Algorithm**

  The A* algorithm excels at finding shortest paths in graphs and grids. This makes it applicable to a wide range of problems:

  1. Pathfinding - Navigation in video games, maps, robotics
  2. Travel routing - Find fastest route in road networks, public transit
  3. Wire routing - Laying out circuit designs
  4. Board games - Chess, checkers, puzzles like the 15-puzzle
  5. Network routing - Optimizing packet routing in networks
  6. Machine learning - Heuristic search in decision trees, neural networks
  7. Bioinformatics - Alignment of protein or gene sequences
  8. Robotics - Motion planning algorithms for robots

  A* combines heuristic guidance with guaranteed optimality, making it a go-to approach for shortest path problems across many domains.

  The algorithm’s worst-case time complexity depends on the heuristic function. In the worst case, i.e. of unbounded search space, the time complexity degenerates to an exponential function $O(b^d)$, where b is the branching factor (the average number of unexplored, adjoining vertices) and d stands for the depth of the shortest path to a solution.

  The space complexity of the A* algorithm is $O(v+e)$ in terms of vertices and edges since it keeps all generated vertices and edges in memory. Expressed in terms of a branching factor and the solution depth, the space complexity of the A* algorithm is $O(b^d)$. High memory requirement renders the A* algorithm less suitable as the size and density of a graph increase, which is considered to be its significant disadvantage.

  In summary, the A* algorithm is optimal, as it will always yield an optimal, shortest possible search path. Furthermore, the A* algorithm will always find a solution if there is one, so it is also complete. Finally, A* is optimally efficient, meaning it will explore as few vertices as possible.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

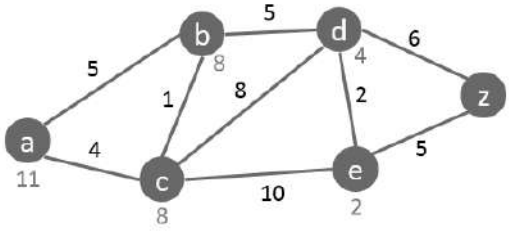
_____

_____

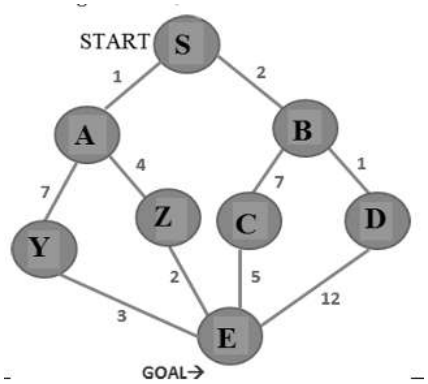
X. Practical related Questions:

(*Note: Teacher shall assign any three questions from following or teacher can design more questions if required to achieve CO)*

1. Find the shortest path for following graph using above given python code.



2. Consider the search problem below with start state S and goal state E. The transaction cost and heuristic values are given. What is the final cost using A* algorithm to reach from the start State to goal state?
   (Heuristic values S=10, A=5, B=6, Y=8, Z=5, C=4, D=15, E=0)



3. Write short note on admissibility of A*.
4. How does A* avoid revisiting nodes unnecessarily? What data structures are typically used for this?
5. Compare GBFS and A*.

Space for Answers

XI. References/Suggestions for further reading

- https://onecompiler.com/python/3xdaztfta.
- https://stackabuse.com/courses/graphs-in-python-theory-and-implementation/lessons/a-star-search-algorithm/
- https://www.datacamp.com/tutorial/a-star-algorithm
- https://www.askpython.com/python/examples/a-star-algorithm
- https://www.colabcodes.com/post/exploring-the-a-search-algorithm-with-python
- https://www.codecademy.com/resources/docs/ai/search-algorithms/a-star-search
- https://llego.dev/posts/implementing-the-a-search-algorithm-python/
- https://academy.finxter.com/python-a-search-algorithm/
- https://www.scaler.com/topics/best-first-search/

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		

Practical No. 6: Write program to implement Bayes' Theorem

I. Practical Significance:

Machine learning is a technique used for analyzing data that involves creating models capable of learning patterns automatically. Rather than being explicitly told what to do, the system improves its performance by repeatedly learning from the data. One widely used machine learning method is the Naive Bayes algorithm, known for its simplicity and efficiency. It can be easily implemented using Python and its available libraries. At the heart of Naive Bayes lies Bayes' Theorem, a key principle in statistics and data science. This theorem provides a way to revise existing predictions or beliefs when new information becomes available. It calculates the probability of an outcome based on prior knowledge and newly observed evidence, helping us make better-informed decisions.

II. Industry/Employer expected outcome(s):

- Improve decision-making in uncertain environments.

III. Course Level Learning Outcome (COs):

- CO2 - Apply method for knowledge representation to make informed decisions for various applications.

IV. Laboratory Learning Outcome (LLO):

- LLO 6.1- Develop a program using Bayes' theorem.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

Naive Bayes is a probabilistic machine learning algorithms based on the Bayes' Theorem. It is a simple yet powerful algorithm because of its understanding, simplicity and ease of implementation. It is popular method for classification applications such as spam filtering and text classification. Implementing Bayes' Theorem in Python is easy and can be done manually with simple math, or by using libraries like scikit-learn for more advanced applications such as Naive Bayes classification.

Let

A1, A2,….., An be a set of mutually exclusive events that together form the sample space S.

Let B be any event from the same sample space, such that P(B) > 0.

Then,

$P(Ak | B) = P(Ak \cap B) / P(A1 \cap B) + P(A2 \cap B) + . . . + P(An \cap B)$

The theorem is expressed mathematically as:

$P(A|B) = (P(B|A)\ P(A)) / P(B)$

Where:

P(A|B): The probability of event A occurring given that B is true.
P(B|A): The probability of event B occurring given that A is true.
P(A): The probability of event A occurring.
P(B): The probability of event B occurring.

Bayesian inference is a statistical method based on Bayes' theorem, which updates the probability of an event as new data becomes available. It is widely used in various fields, such as finance, medicine, and engineering, to make predictions and decisions based on prior knowledge and observed data. In Python, Bayesian inference can be implemented using libraries like NumPy and Matplotlib to generate and visualize posterior distributions.

Bayes' theorem is a fundamental concept in probability theory, forms the foundation of Bayesian inference. This theorem provides a way to calculate the probability of an event occurring given prior knowledge and observed data. Combining our initial beliefs with the likelihood of the evidence, we can arrive at a more accurate posterior probability.

There are many types of Naive Bayes Model, namely

a. Gaussian Naive Bayes
b. Multinomial Naive Bayes
c. Bernoulli Naive Bayes
d. Complement Naive Bayes
e. Out-of-core Naive Bayes

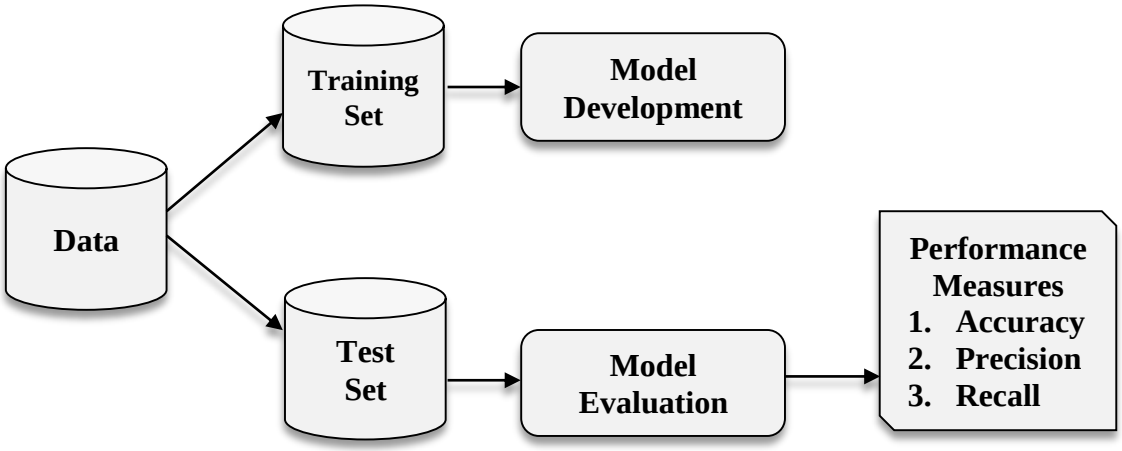


Fig. 1: Naive Bayes Workflow

This diagram in Fig. 1 illustrates the workflow of a Naive Bayes classifier, highlighting both the model development and evaluation process. It demonstrates how raw data is split and used to train and assess a machine learning model.
To train a Naive Bayes model, we need to estimate the prior probabilities P(C) and the likelihood probabilities P(X|C) from the training data. The process involves the following steps:

1. Calculate the prior probabilities P(C) for each class by counting the occurrences of each class in the training data and dividing by the total number of instances.
2. Calculate the likelihood probabilities P(X|C) for each feature and class combination:

3. For continuous features (in the case of Gaussian Naive Bayes), calculate the mean and variance of the feature values for each class.
4. For discrete features (in the case of Multinomial, Bernoulli, or Categorical Naive Bayes), count the occurrences of each feature value for each class and divide by the total number of instances of that class.

To implement the Bayes Theorem algorithm, particularly for classification tasks like Naive Bayes, the following Python libraries are commonly used:

1. Numerical computation: **NumPy**
2. Data handling: **Pandas**
3. Model implementation: **Scikit-learn**
4. Metrics evaluation: **Scikit-learn**
5. Visualization: **Matplotlib, Seaborn**
6. Text preprocessing: **nltk, spaCy, scikit-learn**

- **Implementation of Bayes' Theorem:**

Example 1:

Let's move into more real world example. Suppose we have a list of name. We want to classify these names into Male and Female categories. Our classification process is as show in below.

a) Training

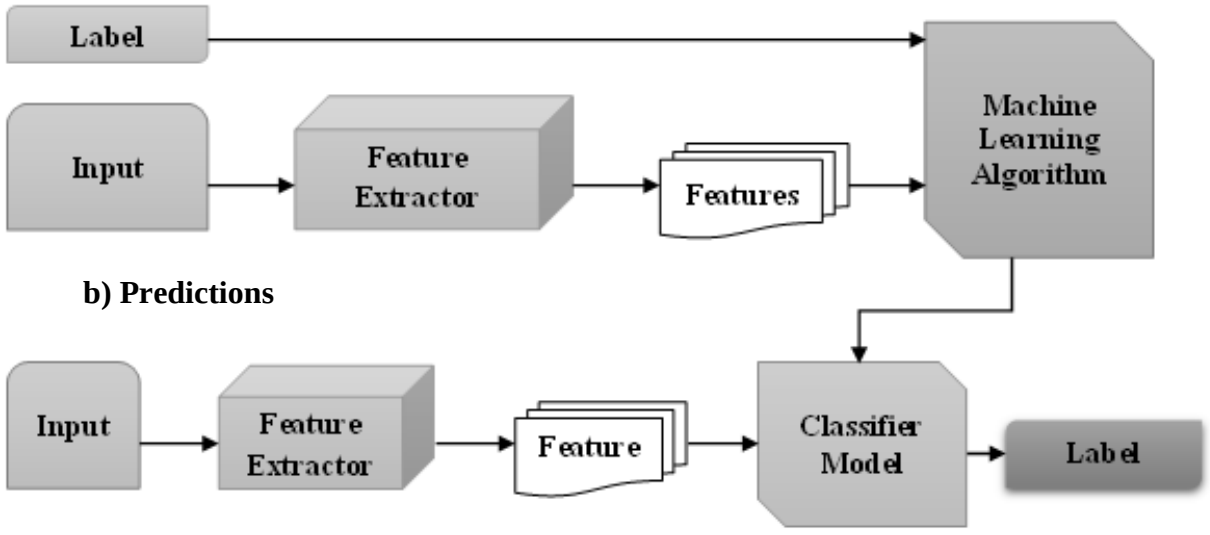


Fig. 2: Training and Prediction

This diagram illustrates the process of machine learning model development and usage, split into two main phases: Training and Prediction.

The training phase involves learning from labeled data to create a model. The prediction phase uses the trained model to classify new, unlabeled data. This diagram represents a typical supervised machine learning workflow.

So first step is feature extraction. We will observe (extract) following evidence from a name last letter, last two letter and last_is_vowel, Next step is machine learning algorithm. we are using Naive Bayes classification.

Lets start implementation in python. As it is more a NLP problem, We could use NLTK module from python. We have two csv files trading file names.txt and predict.txt which contains name to be predicted.

Python Code for Example 1:

```
import numpy as np
import pandas as pd
import nltk
# ------------------- Name Gender Classification -------------------
def get_data(name_file, result_col="gender"):
    df = pd.read_csv(name_file)
    df['last_letter'] = df.iloc[:, 0].apply(lambda x: x[-1])
    df['last_two_letter'] = df.iloc[:, 0].apply(lambda x: x[-2:])
    df['last_is_vowel'] = df.iloc[:, 0].apply(lambda x: int(x[-1].lower() in "aeiouy"))
    features = df[['last_letter', 'last_two_letter', 'last_is_vowel']]
    labels = df[result_col]
    train_dicts = features.to_dict(orient='records')
    return [(feat, label) for feat, label in zip(train_dicts, labels)]

# Train on names.txt
df = pd.read_csv("names.txt")
df['last_letter'] = df.iloc[:, 0].apply(lambda x: x[-1])
df['last_two_letter'] = df.iloc[:, 0].apply(lambda x: x[-2:])
df['last_is_vowel'] = df.iloc[:, 0].apply(lambda x: int(x[-1].lower() in "aeiouy"))
print("\nTraining data with features:\n", df.head())
train_set = get_data("names.txt")
classifier = nltk.NaiveBayesClassifier.train(train_set)

# Predict on predict.txt
print("\nPredictions:")
for name_feat in get_data("predict.txt", result_col="name"):
    predicted = classifier.classify(name_feat[0])
    print(f"{name_feat[1]} == {predicted}")
```

#Output

Training data with features:

	name	gender	last_letter	last_two_letter	last_is_vowel
0	ebin	M	n	in	0
1	leekas	M	s	as	0
2	jinesh	M	h	sh	0
3	neethu	F	u	hu	1
4	mary	F	y	ry	1

Predictions:

sukesh == M

```
jithil == M
sijith == M
maria == F
soumya == F
neethu == F
```

Example 2:

To implement this theorem in python, we have amazing library called scikit-learn in python. In below example we are going to create some random points in three dimensional space. We classified these points onto RED and BLUE. Our task is to classify new points in this three dimensional space into either BLUE or RED. We have to start with importing required modules.

Python Code for Example 2:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB

# ------------------- 3D Points Classification -------------------
# Blue class
x_blue = np.array([1, 2, 1, 5, 1.5, 2.4, 4.9, 4.5])
y_blue = np.array([5, 6.3, 6.1, 4, 3.5, 2, 4.1, 3])
z_blue = np.array([5, 1.3, 1.1, 1, 3.5, 2, 4.1, 3])
# Red class
x_red = np.array([5, 7, 7, 8, 5.5, 6, 6.1, 7.7])
y_red = np.array([5, 7.7, 7, 9, 5, 4, 8.5, 5.5])
z_red = np.array([5, 6.7, 7, 9, 1, 4, 6.5, 5.5])

# Combine data
red_points = np.array(list(zip(x_red, y_red, z_red)))
blue_points = np.array(list(zip(x_blue, y_blazue, z_blue)))
points = np.concatenate([red_points, blue_points])
output = np.concatenate([np.ones(x_red.size), np.zeros(x_blue.size)])

# Train GaussianNB classifier
predictor = np.array([5.3, 4.2, 3.3])
classifier = GaussianNB()
classifier.fit(points, output)
print("Prediction (1 = Red, 0 = Blue):", classifier.predict([predictor])[0])

#Output
Prediction (1 = Red, 0 = Blue): 0.0
```

Example 3:

Let's take a simple problem as Bayes theorem example, Your neighbour is watching Indian football team playing, you hear them cheering and want to estimate the probability that team has scored. If you want to think of it as a ML problem, team scoring is the target and we have one feature here, neighbour cheering for predicting whether team scored or not.

Let's assume probability of goal is 2% = 0.02. And probability of cheering given a goal is scored is 90% = 0.9. The probability of cheering when the goal is not scored assumed to be around 1% = 0.01, So, probability of not goal is (100–2)/100 = 0.98. Our aim is to find the probability of a goal scored while the neighbours cheering, using Bayes theorem

$$P\left(\frac{Goal}{Cheer}\right) = \frac{P\left(\frac{Cheer}{Goal}\right) P(Goal)}{P(Cheer)}$$

$$P\left(\frac{Goal}{Cheer}\right) = \frac{0.9 * 0.02}{(0.9 * 0.02) + (0.98 * 0.01)}$$

$$= 64.7\%$$

- **Applications of Bayes' Theorem**
  - Medical Diagnosis
  - Email or Spam Filtering
  - Recommendation Systems
  - Machine Learning.
  - Sentiment Analysis
  - Document Classification
  - Text & image classification
  - Weather prediction
  - Anomaly detection
  - Image classification

The space complexity of implementing Bayes' theorem can be summarized as O(n * m) where n is the number of features and m is the number of classes. This is because we need to store probabilities for each feature-class combination. And O(n) for storing the prior probabilities of each class.

The space complexity of a Naive Bayes model is also linear concerning the number of features and classes, O(m * k), where m is the number of features and k is the number of classes. This is because the model needs to store the prior probabilities for each class and the likelihood probabilities for each feature-class combination.

In summary, Bayes Theorem is a powerful statistical tool that allows us to update our probabilities based on new evidence. Understanding its principles and implementation using Python can significantly enhance our analytical capabilities. Whether in healthcare, marketing, or machine learning, mastering Bayes Theorem can lead to better decision-making.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any three questions from following or teacher can design more questions if required to achieve CO)*

1. Implement Python program to find the probability of a goal scored while the neighbours cheering given in example 3.
2. What data is stored while using Naive Bayes algorithm.
3. Write a Python function to implement Naive Bayes from scratch for binary classification.
4. How would you use Naive Bayes to detect fraudulent transactions?
5. What are the differences between Gaussian, Multinomial, and Bernoulli Naive Bayes?
6. How do you use GaussianNB from scikit-learn?

Space for Answers

_____

_____

XI. References/Suggestions for further reading

- https://www.statology.org/bayes-theorem-in-python/
- https://www.geeksforgeeks.org/ml-naive-bayes-scratch-implementation-using-python/
- https://www.askpython.com/python/examples/bayesian-inference-in-python
- https://medium.com/@AbhiramiVS/bayes-theorem-with-python-ef2135d850f
- https://www.bridge-global.com/blog/bayes-theorem-implementation-in-python/
- https://www.tutorialspoint.com/machine_learning/machine_learning_bayes_theorem.htm
- https://betanet.net/view-post/understanding-bayes-theorem-through-python

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		

Practical No. 7: Analyze the given Case study: How Turing test is performed between Responder and an Interrogator?

I. Practical Significance:

The Turing Test is commonly used to determine if a machine can mimic human behavior convincingly. During the test, a human communicates via text with both a machine and another person, without knowing which is which. If the human is unable to tell the machine apart from the person based on their replies, the machine is said to have passed the test.

The Turing Test serves as a well-known method for assessing whether a machine can exhibit behavior indistinguishable from that of a human. In this test, a human evaluator interacts through text with both a person and a computer without knowing which is which. If the evaluator cannot reliably identify the machine based on their responses, the machine is considered to have successfully passed the test.

II. Industry/Employer expected outcome(s):

- Study of how turing test is performed between Responder and an Interrogator.

III. Course Level Learning Outcome (COs):

- CO3- Analyze different forms of data with respect to different phases of Machine Learning.

IV. Laboratory Learning Outcome (LLO):

- LLO 7.1-Analyze the process of turing test for given Dataset.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

- **Turing Test**

  The Turing Test offers a seemingly straightforward way to assess whether a machine can exhibit human-like intelligence. In this setup, a human judge converses with both a machine and a human, aiming to identify which is which. If the judge cannot accurately distinguish between them after a series of questions, the machine is considered to have succeeded, indicating it can replicate human thought and behavior.

- **The key criteria include the following for turing test:**
  - **Natural Language Processing (NLP):** The machine must understand and generate human language fluently.
  - **Knowledge Representation:** The machine needs to handle and manipulate knowledge to provide contextually relevant responses.
  - **Reasoning:** The machine should demonstrate some form of logical reasoning, even if flawed, to sustain a conversation.

- **Learning:** Ideally, the machine should learn from the interaction, adapting its responses over time.

- **Roles Involved in Turing Test:**
  - **Interrogator (Judge):**
    A human tasked with determining which of the other two participants is the machine.
    Does not know who is the machine or the human.
    Asks a series of questions via text-based communication to both participants.
  - **Responder A (Human):**
    A real human who tries to answer the Interrogator's questions truthfully.
  - **Responder B (Machine/AI):**
    An artificial intelligence program designed to mimic human conversational behavior.

- **Test Procedure:**
  a) The Interrogator starts asking open-ended or probing questions (e.g., "What do you think about art?" or "Describe your last holiday.").
  b) Both Responders answer independently.
  c) The Interrogator evaluates the naturalness, coherence, and emotional intelligence in the responses.
  d) After a series of interactions, the Interrogator must guess which participant is the machine.
  e) Success Criteria:
     a. If the machine is able to fool the Interrogator a significant portion of the time (e.g., more than 30% in early versions), it is said to have "passed" the Turing Test.
- **Advantages of the Turing Test in Artificial Intelligence**
  - **Evaluating Machine Intelligence:** The Turing Test provides a simple and well-known method for assessing machine intelligence.
  - **Setting a Benchmark:** It establishes a benchmark for AI research and offers a goal for researchers to strive towards.
  - **Inspiring Research:** The Turing Test has inspired numerous studies and experiments aimed at developing machines that can pass the test, driving progress in AI.
  - **Simple to Administer:** The Turing Test is relatively easy to administer, requiring just a computer and a human judge.

- **Disadvantages of the Turing Test in Artificial Intelligence**
  - **Limited Scope:** The Turing Test focuses primarily on language-based conversations and does not account for other aspects of intelligence, such as perception, problem-solving, and decision-making.
  - **Human Bias:** The results can be influenced by the biases and preferences of the human judge, making it difficult to obtain objective and reliable results.
  - **Not Representative of Real-World AI:** The Turing Test may not accurately represent the kind of intelligence that machines need to demonstrate in real-world applications.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

_____

Practical related Questions:

(*Note: Teacher shall assign any four questions from following or teacher can design more questions if required to achieve CO)*

1. What is the primary objective of the Turing Test?
2. How does the Turing Test simulate human-like conversation?
3. How does the Interrogator distinguish between a human and a machine?
4. Can you provide an example where an AI passed the Turing Test?
5. What are some criticisms of the Turing Test?
6. What ethical considerations arise from AI passing the Turing Test?
7. Why are personal and emotional questions significant in the Turing Test?

Space for Answers

_____

_____

_____

_____

_____

X. References/Suggestions for further reading

- https://www.vationventures.com/glossary/turing-test-definition-explanation-and-use-cases#:~:text=Use%20Cases%20of%20the%20Turing%20Test&text=It%20has%20been%20used%20as,nature%20of%20intelligence%20and%20consciousness.
- https://www.geeksforgeeks.org/artificial-intelligence/turing-test-artificial-intelligence/
- https://www.coursera.org/articles/what-is-the-turing-test
- https://plato.stanford.edu/entries/turing-test/
- https://www.sciencedirect.com/topics/neuroscience/turing-test
- https://youtu.be/3wLqsRLvV-c?si=vRzhoO5XVjQAvcEg
- https://youtu.be/sXx-PpEBR7k?si=3g5SPTb1Ts6Ljdkj
- https://youtu.be/4VROUIAF2Do?si=yOaNvsjSA8Ch_4Ce

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 8: Explore different dataset finders e.g. Google Dataset Search, Kaggle

I. Practical Significance:

A dataset finder plays a significant role in the world of data science, machine learning, research, and analytics by enabling users to easily locate relevant datasets from a wide range of sources. Its primary value lies in saving time and effort that would otherwise be spent searching manually for data. By aggregating publicly available datasets often from government databases, research institutions, and open-source platforms a dataset finder ensures quick access to high-quality, diverse, and reliable data.

In fields such as artificial intelligence, policy making, and business intelligence, having a reliable tool to find datasets directly impacts the quality and efficiency of decision-making processes.

II. Industry/Employer expected outcome(s):

- Search and use different datasets from google search dataset, Kaggle etc.

III. Course Level Learning Outcome (COs):

- CO3- Analyze different forms of data with respect to different phases of Machine Learning.

IV. Laboratory Learning Outcome (LLO):

- LLO 8.1- Analyze different datasets with respect to its use

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

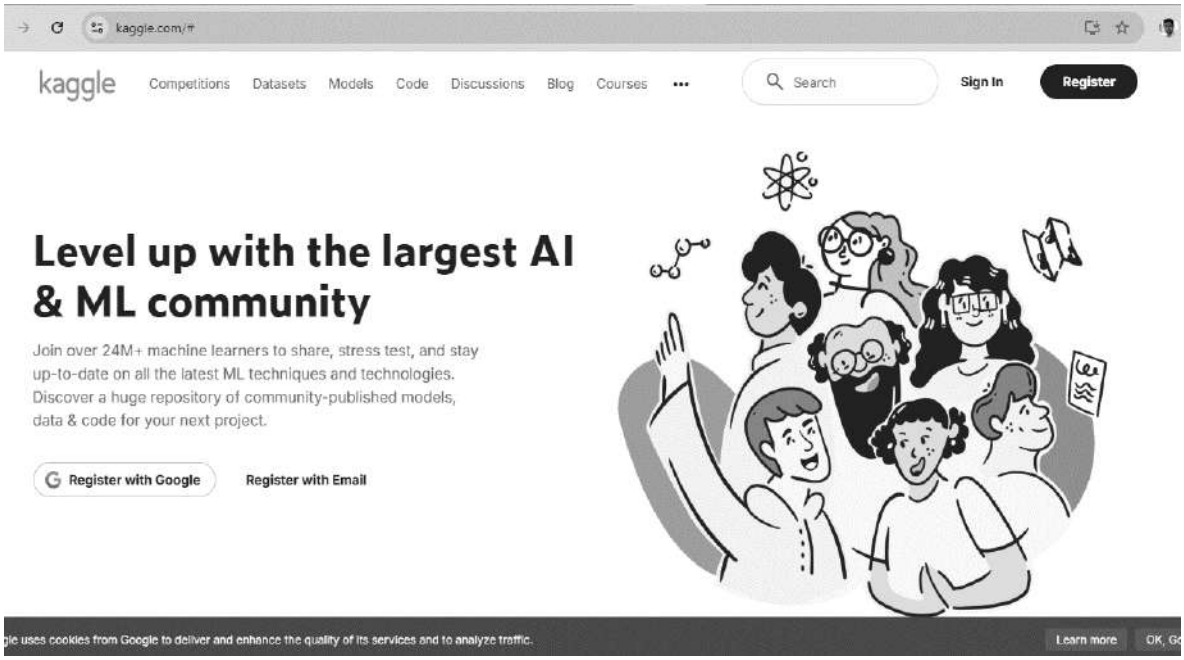
- **Kaggle Datasets:**

  A Kaggle dataset refers to any dataset that is hosted on Kaggle, a popular platform for data science competitions, learning, and collaboration. Kaggle provides a vast repository of public datasets covering a wide range of topics including healthcare, finance, sports, social media, and more. These datasets are contributed by individuals, organizations, and the Kaggle team itself, and they are freely available for download and use in projects.

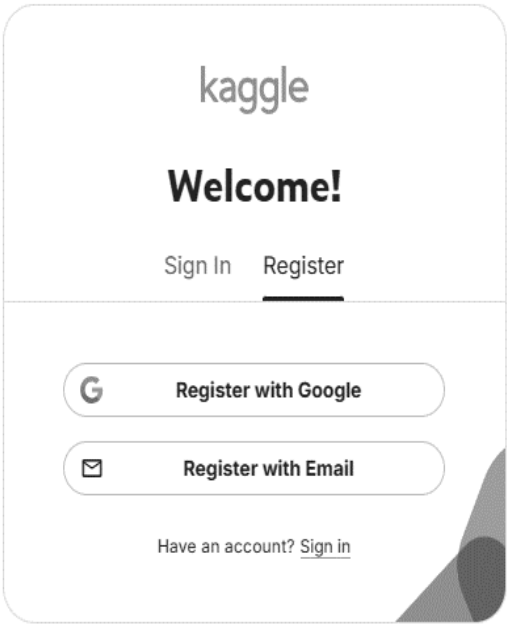
  Kaggle datasets are often used in machine learning model training, data visualization projects, and exploratory data analysis. One of the key advantages of using Kaggle datasets is that many of them come with kernels (code notebooks) and discussions that help users understand the data better and learn from others. As such, Kaggle serves not only as a source of data but also as a community-driven hub for data science learning and experimentation.

- **Steps of download datasets from Kaggle**

1. Visit to website Kaggle.com

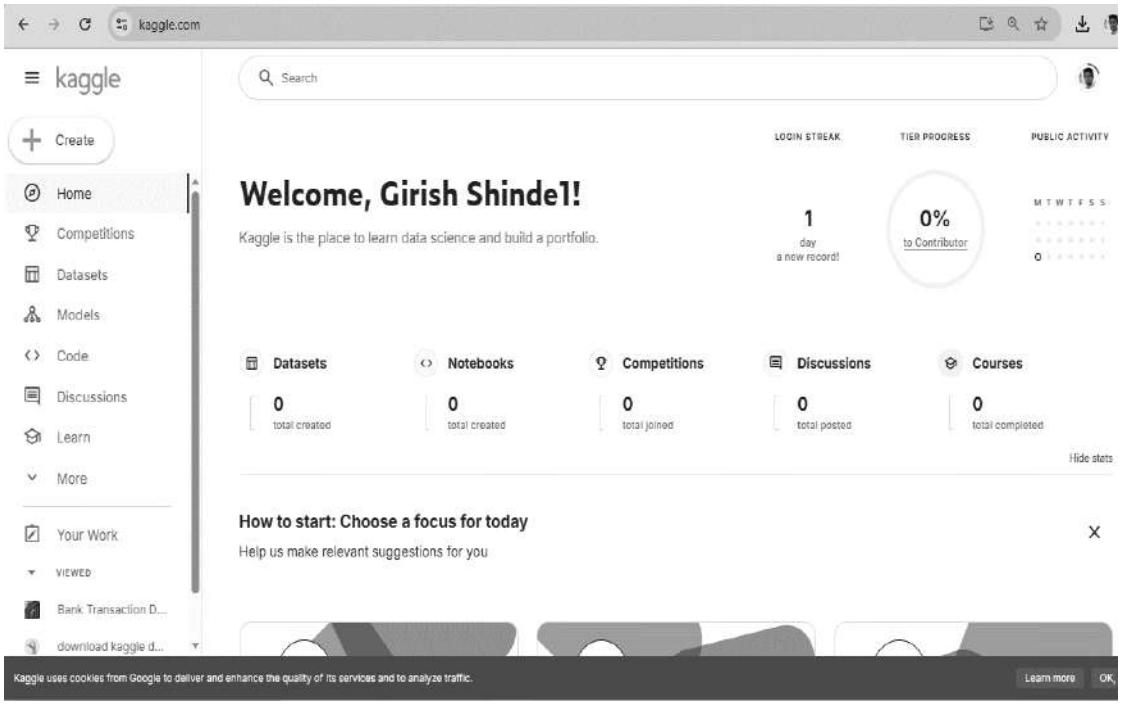


2. Click on register and create the account using email

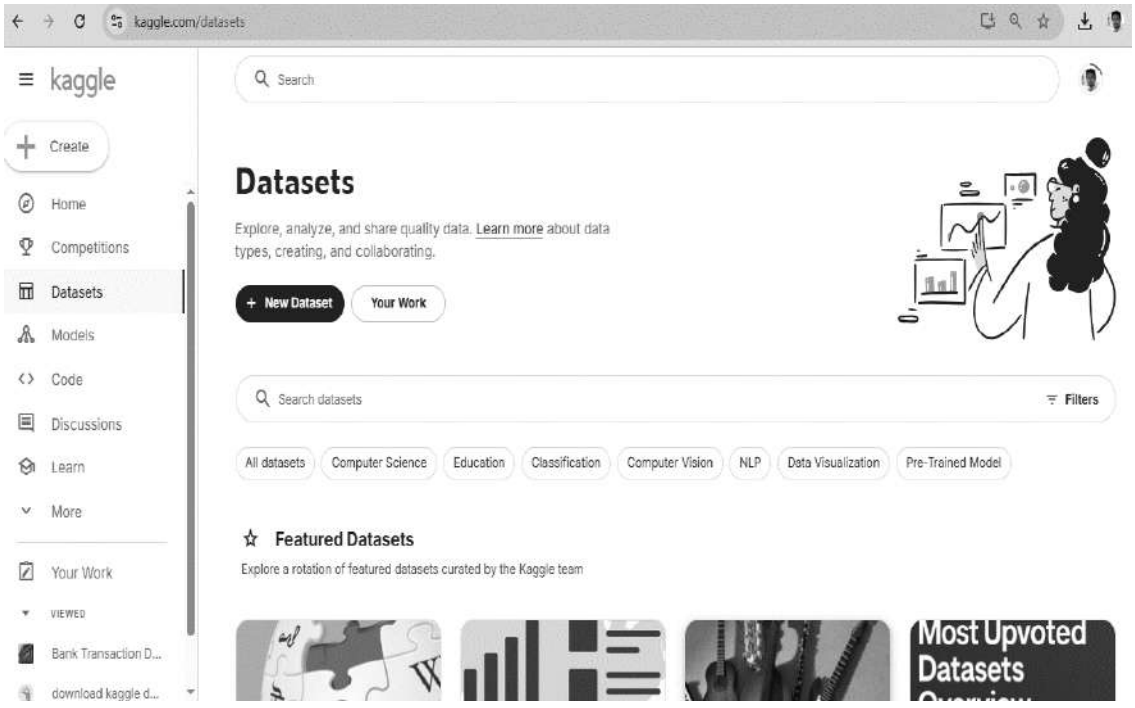


When you link your Google account, Kaggle collects certain information stored in that account that you have configured to make available. By linking your accounts, you authorize Kaggle to access and use your account on the third party service in connection with your use of kaggle.com.

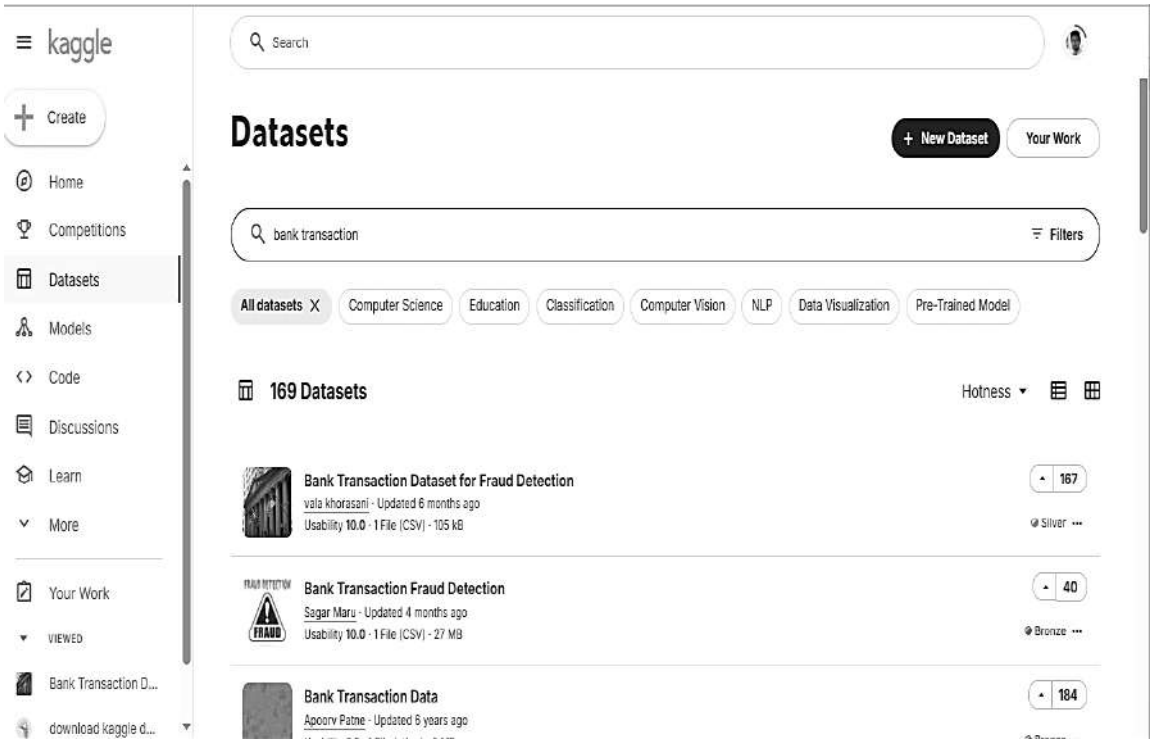
3. Login on the Kaggle.com



4. Select the tab dataset



5. Click on the textbox to search the required datasets (For example Datasets of banking related)



6. You can download it with zipped file or use it using API

```
import kagglehub
path = kagglehub.dataset_download("valakhorasani/bank-transaction-dataset-for-fraud
        detection")
print("Path to dataset files:", path)
```

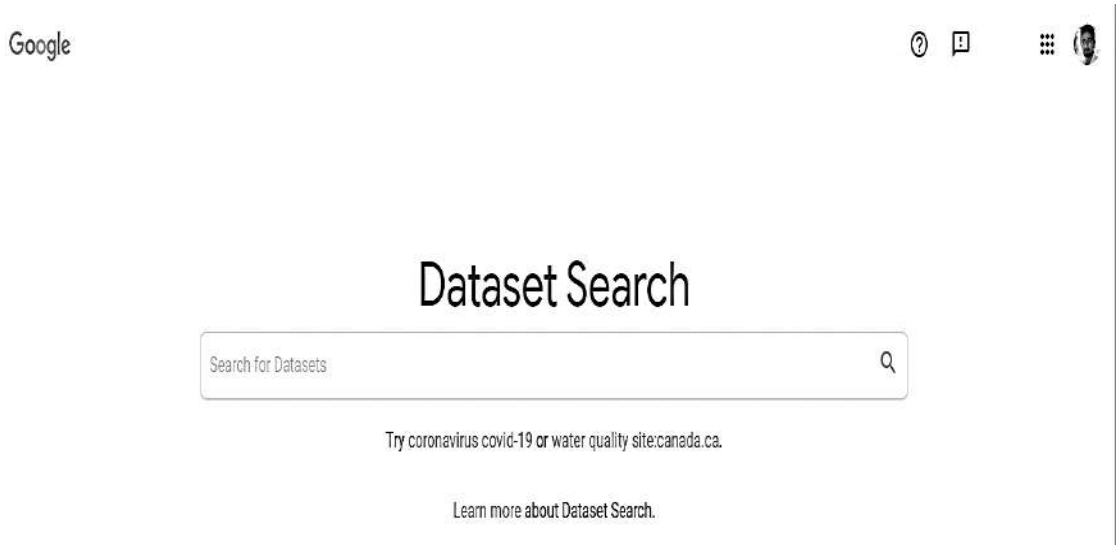
- **Google Datasets Search:**

  Google Dataset Search is a specialized search engine to help researchers, data scientists, journalists, and the general public locate datasets stored across the web. Google Dataset Search is a free tool launched by Google that indexes datasets from various repositories and websites, enabling users to discover data relevant to their needs. It functions similarly to Google Scholar but specifically targets datasets, including open government data, scientific datasets, and datasets hosted by research institutions or private entities.

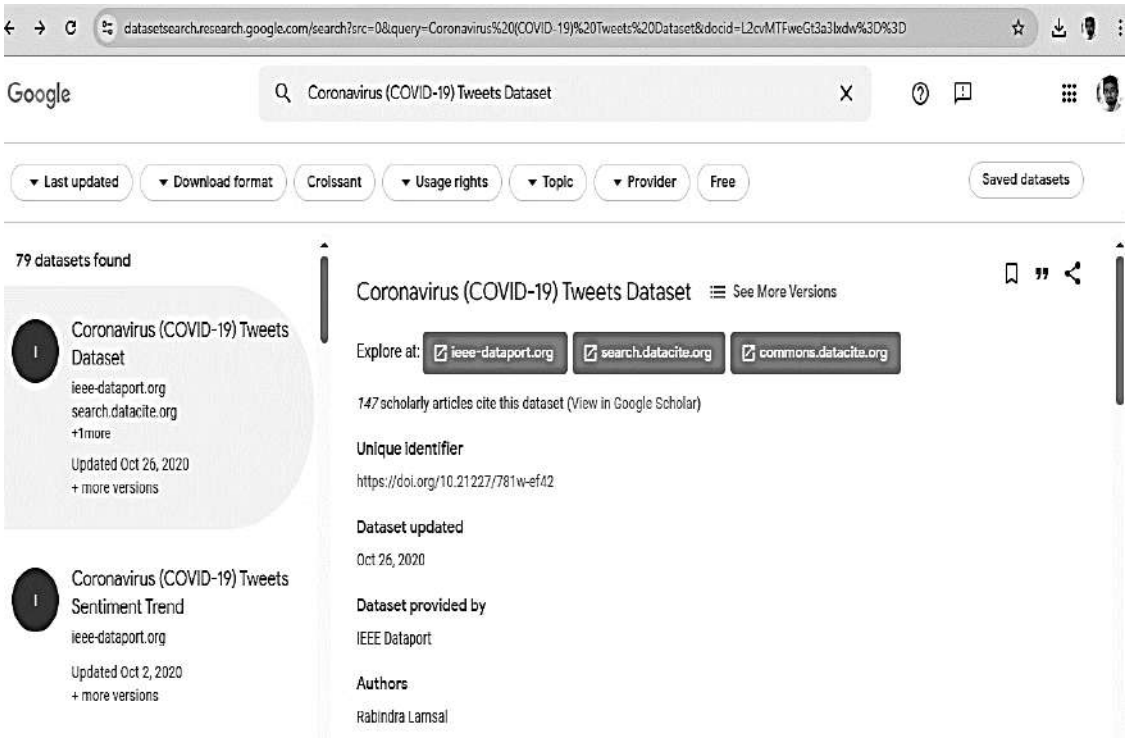
  Google Dataset Search crawls and indexes datasets that are described using schema.org's Dataset markup, which is a standard way to describe data with metadata (e.g., title, description, creator, license, format).

- **Steps of download datasets from Google Dataset Search**

1. Visit to website https://datasetsearch.research.google.com/

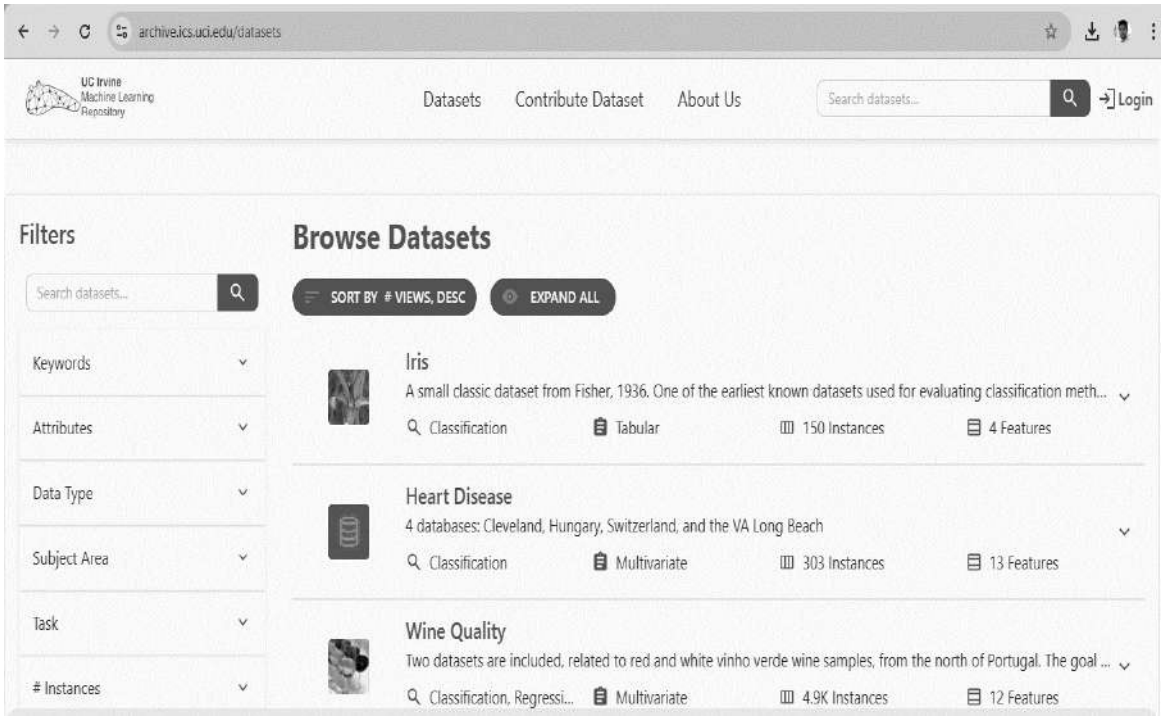


2. Enter Your Search Query (Use keywords relevant to your topic. Example: global temperature data, COVID-19 India, or satellite images CSV)



3. Browse the Results
4. Download the Dataset from the Source Website

- **UCI Machine Learning Repository**
  https://archive.ics.edu/datasets
  Being one of the oldest dataset aggregators on the web, the UCI (Unified Configuration Interface) Machine Learning Repository is a collection of databases, domain theories and data generators used by the machine learning community to analyze machine learning algorithms. The archive was created as an FTP archive in a987 and the website design was later updated in 2007. Having been widely used by students, educators and researchers worldwide the repository earned the status of a primary source of machine learning datasets.



VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

X. Practical related Questions:

(*Note: Teacher shall assign any four questions from following or teacher can design more questions if required to achieve CO)*

1. What is dataset and different types of datasets?
2. What is the purpose of this dataset?
3. List any four General-Purpose Dataset Repositories.
4. Write steps to register on Kaggle Dataset and download the banking related dataset?
5. Difference between database and dataset.
6. How to access google dataset Search?

Space for Answers

XI. References/Suggestions for further reading

- https://www.geeksforgeeks.org/what-is-dataset
- https://www.geeksforgeeks.org/uci-machine-learning-repository
- https://labelyourdata.com/articles/machine-learning/datasets
- https://youtu.be/c5yhVV-eYB0?si=yzdJydu3JSeSdmn9
- https://www.youtube.com/watch?v=_uwucNViakk
- https://youtu.be/-eZaE8xbt0A?si=atSOV3S8pBxcnnic

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 9: Write program in python to split any dataset into train and tests sets

I. Practical Significance:

Splitting a dataset into training and test sets is a fundamental step in any machine learning or data science project, and Python plays a significant role in making this process easy and efficient. The primary purpose of this split is to evaluate how well a model generalizes to unseen data. The training set is used to teach the model, while the test set is reserved for evaluating its performance. This helps prevent over fitting, where a model performs well on training data but fails on new data.

Python, with libraries like scikit-learn, provides simple and reliable functions such as train_test_split() that allow users to randomly divide data while maintaining balanced distributions and reproducibility. By using Python to split data correctly, data scientists and machine learning practitioners can assess the true effectiveness of their models, compare different algorithms, and make better decisions when deploying models to real-world applications.

II. Industry/Employer expected outcome(s):

- To split any dataset into train and tests sets.

III. Course Level Learning Outcome (COs):

- CO3- Analyze different forms of data with respect to different phases of Machine Learning.

IV. Laboratory Learning Outcome (LLO):

- LLO 9.1- Develop program based on training and testing datasets.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

- **Dataset Splitting**

  Scikit-learn is one of the most widely used machine learning libraries in Python. It provides a range of tools for building models, pre-processing data, and evaluating performance. For splitting datasets, it provides a handy function called train_test_split() within the model_selection module, making it simple to divide your data into training and testing sets.

- **Training Sets:**

  In machine learning, a training set is the data used to teach a model how to make predictions or decisions. It's like providing examples to the model so it can learn patterns and relationships within the data. The training set is essential for supervised learning, where the model learns by comparing its predictions to known outcomes.

- **Test Sets:**

  In machine learning, a test set is a portion of the data that is not used during model training but is kept separate to evaluate the model's performance on unseen data after it has been

trained. It helps assess how well the model generalizes to new, unknown data and indicates if the model has learned the underlying patterns or simply memorized the training data, a phenomenon known as over fitting.

- **General Algorithm:**

1**. Load and Prepare Data:**
   - Load your dataset into a suitable format, such as a Pandas DataFrame. If necessary, preprocess the data (e.g., handle missing values, encode categorical features).

2. **Separate Features and Target:**
   - Identify the independent variables (features) and the dependent variable (target) you want to predict.

3. **Split the Data:**
   - Use the train_test_split function from scikit-learn.
   - Specify the dataset, the proportion of data to be used for testing (e.g., 20% for testing), and a random state for reproducibility.
   - The function will return four arrays: X_train, X_test, y_train, and y_test representing the training features, testing features, training target, and testing target, respectively.

4. **Training and Testing:**
   - X_train and y_train are used to train the machine learning model.
   - X_test and y_test are used to evaluate the model's performance on unseen data.

- **Program:**

```
import pandas as pd
from sklearn.model_selection import train_test_split
def split_dataset(data, test_size=0.2, random_state=None):
    """
    Splits a dataset into training and testing sets.
    Args: data (pd.DataFrame): The dataset to split.
        test_size (float, optional): The proportion of the dataset to include in the test split.
                                     Should be between 0.0 and 1.0. Defaults to 0.2.
        random_state (int, optional): Controls the shuffling applied to the data before splitting.
Pass an int for reproducible output across multiple function calls. Defaults to None.
    Returns:  tuple: A tuple containing the training and testing sets (X_train, X_test, y_train,
y_test).
              If the input data does not have a target variable, it returns (X_train, X_test).
    """
    if isinstance(data, pd.DataFrame):
        X = data.iloc[:, :-1]
        y = data.iloc[:, -1] if len(data.columns) > 1 else None
        if y is not None:
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
            random_state=random_state)
           return X_train, X_test, y_train, y_test
        else:
```

```
        X_train, X_test = train_test_split(X, test_size=test_size, random_state=random_state)
        return X_train, X_test
    else:
        raise TypeError("Input data must be a pandas DataFrame.")
if __name__ == '__main__':
    # Example usage:
    data = pd.DataFrame({   'feature1': [1, 2, 3, 4, 5], 'feature2': [6, 7, 8, 9, 10], 'target': [11, 12, 13, 14, 15]    })

    # Split the dataset into 80% training and 20% testing sets
    X_train, X_test, y_train, y_test = split_dataset(data, test_size=0.2, random_state=42)
    print("X_train:")
    print(X_train)
    print("X_test:")
    print(X_test)
    print("y_train:")
    print(y_train)
    print("y_test:")
    print(y_test)
```

#Output

```
X_train:
   feature1  feature2
4         5        10
2         3         8
0         1         6
3         4         9

X_test:
   feature1  feature2
1         2         7

y_train:
4    15
2    13
0    11
3    14

Name: target, dtype: int64

y_test:
1    12

Name: target, dtype: int64
```

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any three questions from following or teacher can design more questions if required to achieve CO)*

1. What is training set?
2. What is test set?
3. Split the dataset of student records into the training set and test set.
4. Split the dataset of inventory management into the training set and test set.
5. How many parameters are required for the train_test_split() function? Elaborate it.

Space for Answers

_____

_____

_____

_____

XI. References/Suggestions for further reading

- https://www.techtarget.com/searchenterpriseai/definition/data-splitting
- https://www.geeksforgeeks.org/how-to-split-a-dataset-into-train-and-test-sets-using-python/
- https://builtin.com/data-science/train-test-split
- https://global.trocco.io/blogs/why-do-you-split-data-into-training-and-validation-sets#:~:text=By%20splitting%20the%20data%20into,model%20generalizes%20to%20new%20data.
- https://youtu.be/fwY9Qv96DJY?si=lLh4lwXyBgRxdIzy
- https://youtu.be/uIfdKDixXFY?si=oP0UPV3qlTeGPEOc
- https://youtu.be/rCevxk3jeKs?si=07w0B14tKaappYBp

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 10: Analyze E-mail spam and non-spam filtering using Machine Learning through case study

I. Practical Significance:

Machine learning can effectively classify email spam and non-spam. This can be achieved by training ML models on labeled datasets of emails, which are then used to predict the spam or non-spam status of new, unseen emails. Common ML algorithms include Naive Bayes, Support Vector Machines, and KNN, often used in conjunction with Natural Language Processing (NLP) techniques like tokenization, stop-word removal, and stemming.

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. Machine learning methods of recent are being used to successfully detect and filter spam emails.

II. Industry/Employer expected outcome(s):

- Study of Email spam and non-spam filtering using Machine Learning.

III. Course Level Learning Outcome (COs):

- CO3- Analyze different forms of data with respect to different phases of Machine Learning.

IV. Laboratory Learning Outcome (LLO):

- LLO 10.1- Analyze the mail filtering process.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

- **Steps to detect email spam using machine learning**

1. Data Collection

Collect a labeled dataset of emails or messages. Each sample should have a label indicating whether it's spam or ham (not spam). Popular datasets include the UCI SMS Spam Collection and SpamAssassin.

2. Data Preprocessing

a. Clean and prepare the raw text data:
b. Convert all text to lowercase
c. Remove punctuation, numbers, and special characters
d. Remove stopwords (e.g., “the”, “and”, “is”)
e. Apply stemming or lemmatization to reduce words to their base form

3. Feature Extraction

a. Convert text into numerical format so it can be fed into a machine learning model:
b. Bag of Words (BoW) – counts the frequency of words
c. TF-IDF (Term Frequency-Inverse Document Frequency) – gives importance to less common but more meaningful words

d. Optionally, use word embeddings (Word2Vec, GloVe) or BERT embeddings for advanced models

4. Splitting the Dataset

a. Divide the dataset into:
b. Training set – to train the model (usually 70-80% of the data)
c. Test set – to evaluate the model's performance (20-30%)

5. Model Selection and Training

a. Choose a machine learning algorithm. Common choices include:
b. Naive Bayes – fast and effective for text classification
c. Logistic Regression
d. Support Vector Machine (SVM)
e. Random Forest
f. Deep Learning models (e.g., LSTM, BERT for more complex tasks)
g. Train the model on the training data using the selected algorithm.

6. Model Evaluation

a. Use the test data to evaluate the model's performance with metrics like:
b. Accuracy
c. Precision – how many predicted spams were actually spam
d. Recall – how many actual spams were correctly predicted
e. F1 Score – balance between precision and recall
f. Confusion Matrix – shows true/false positives and negatives

7. Model Tuning (Optional)

a. Improve performance by:
b. Hyper parameter tuning
c. Cross-validation
d. Trying different feature extraction techniques

8. Deployment

Integrate the trained model into an email system to classify incoming emails in real time or batches. Use user feedback to continuously retrain and improve the model.

9. Monitoring and Maintenance

Monitor the model's performance over time to handle concept drift (changes in spam patterns) and update the model as needed.

These steps form a complete pipeline for detecting spam emails using machine learning from raw text to a deployed, intelligent filter.

- **There are several email spam filtering methods**

1. **Content Based Filtering Technique:** Content based filtering is usually used to create automatic filtering rules and to classify emails using machine learning approaches, such as Naïve Bayesian classification, Support Vector Machine, K Nearest Neighbor, Neural Networks. This method normally analyses words, the occurrence, and distributions of words and phrases in the content of emails and used then use generated rules to filter the incoming email spams.
2. **Case Base Spam Filtering Method:** Case base or sample base filtering is one of the popular spam filtering methods. Firstly, all emails both non-spam and spam emails are

extracted from each user's email using collection model. Subsequently, pre-processing steps are carried out to transform the email using client interface, feature extraction, and selection, grouping of email data, and evaluating the process. The data is then classified into two vector sets. Lastly, the machine learning algorithm is used to train datasets and test them to decide whether the incoming mails are spam or non-spam.

3. **Heuristic or Rule Based Spam Filtering Technique:** This approach uses already created rules or heuristics to assess a huge number of patterns which are usually regular expressions against a chosen message. Several similar patterns increase the score of a message. In contrast, it deducts from the score if any of the patterns did not correspond. Any message's score that surpasses a specific threshold is filtered as spam; else it is counted as valid. While some ranking rules do not change over time, other rules require constant updating to be able to cope effectively with the menace of spammers who continuously introduce new spam messages that can easily escape without been noticed from email filters. A good example of a rule based spam filter is SpamAssassin.
4. **Previous Likeness Based Spam Filtering Technique:** This approach uses memory-based, or instance-based, machine learning methods to classify incoming emails based to their resemblance to stored examples (e.g. training emails). The attributes of the email are used to create a multi-dimensional space vector, which is used to plot new instances as points. The new instances are afterward allocated to the most popular class of its K-closest training instances. This approach uses the k-nearest neighbor (kNN) for filtering spam emails.
5. **Adaptive Spam Filtering Technique:** The method detects and filters spam by grouping them into different classes. It divides an email corpus into various groups, each group has an emblematic text. A comparison is made between each incoming email and each group, and a percentage of similarity is produced to decide the probable group the email.

- **Sample Code of mail spam or non-spam verification**

```
# Import necessary libraries
import pandas as pd  # For handling data in a structured way (like a spreadsheet)
from sklearn.model_selection import train_test_split  # For splitting data into training and testing sets
from sklearn.feature_extraction.text import CountVectorizer  # For converting text data into numerical features
from sklearn.naive_bayes import MultinomialNB  # The machine learning algorithm we'll use (good for text)
from sklearn.metrics import accuracy_score, classification_report # For checking how well our model performs
# --- 1. Load the Dataset ---
# For this case study, we'll use a publicly available dataset. This dataset usually has two columns: #one for the email text and one for the label (spam/ham). 'ham' means not spam.
# We'll create a small, illustrative dataset directly in the code.
# In a real scenario, you would load this from a CSV file, e.g., using pd.read_csv('spam.csv', encoding='latin-1')
```

```
data = {
    'email_text': [
        "Hey, are we meeting tomorrow?",
        "WINNER!! You have won a $1000 cash prize. Click here.",
        "Can you please send me the report?",
        "Urgent: Your account has been compromised. Update your details now!",
        "Lunch meeting scheduled for 1 PM.",
        "Claim your free vacation package today!",
        "Thank you for your subscription.",
        "Special offer just for you, limited time only.",
        "Don't forget the grocery list.",
    ],
    'label': [
        'ham',  # Not spam
        'spam', # Spam
        'ham',  # Not spam
        'spam', # Spam
        'ham',  # Not spam
        'spam', # Spam
        'ham',  # Not spam
        'spam', # Spam
        'ham',  # Not spam
        'spam'  # Spam
    ]
}
df = pd.DataFrame(data)
# --- 2. Basic Data Preprocessing ---
# Machine learning models understand numbers, not text.
# We also need to convert our labels ('spam', 'ham') into numerical form (e.g., 1 for spam, 0 for ham).
df['label_num'] = df['label'].map({'ham': 0, 'spam': 1})
# Separate the features (email text) and the target (spam/ham label)
X = df['email_text']  # The email content
y = df['label_num']   # The numerical label (0 or 1)

# --- 3. Split Data into Training and Testing Sets ---
# We train the model on one part of the data and test its performance on another unseen part.
# test_size=0.25 means 25% of the data will be used for testing, and 75% for training.
# random_state ensures that the split is the same every time we run the code, for reproducibility.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(f"--- Data Split ---")
print(f"Training samples: {len(X_train)}")
```

```
print(f"Testing samples: {len(X_test)}\n")

# --- 4. Feature Extraction (Text to Numbers) ---
# We need to convert the text of the emails into a numerical format that the model can work with.
# CountVectorizer does this by counting the occurrences of each word in each email.
vectorizer = CountVectorizer(stop_words='english') # 'stop_words' removes common words like 'the', 'is', 'and'
# Fit the vectorizer on the training data and then transform both training and testing data.
# Fit: learns the vocabulary from the training data.
# Transform: converts the text into a matrix of word counts based on the learned vocabulary.
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test) # Use the same vocabulary learned from training data
# print("--- Feature Vectors (Snippet) ---")
# print(X_train_counts.toarray()) # .toarray() converts the sparse matrix to a dense array for viewing
# print("\n")

# --- 5. Train the Machine Learning Model ---
# We'll use the Multinomial Naive Bayes classifier.
# It's a simple and effective algorithm for text classification tasks like spam filtering.
model = MultinomialNB()
# Train the model using the training data (email word counts and their corresponding labels).
model.fit(X_train_counts, y_train)
print("--- Model Training Complete ---\n")

# --- 6. Make Predictions on the Test Set ---
# Now, let's use our trained model to predict whether the emails in our test set are spam or ham.
predictions = model.predict(X_test_counts)

    # --- 7. Evaluate the Model ---
# Let's see how well our model performed on the unseen test data.
accuracy = accuracy_score(y_test, predictions)
report = classification_report(y_test, predictions, target_names=['Ham (Not Spam)', 'Spam'], zero_division=0)
print("--- Model Evaluation ---")
print(f"Accuracy: {accuracy:.2f}") # Format accuracy to 2 decimal places
print("Classification Report:")
print(report)

# --- Case Study: Analyzing Some Predictions ---
print("\n--- Case Study: Predictions on Test Data ---")
```

```
for i in range(len(X_test)):
    email_text = X_test.iloc[i]
    actual_label = "Spam" if y_test.iloc[i] == 1 else "Ham (Not Spam)"
    predicted_label = "Spam" if predictions[i] == 1 else "Ham (Not Spam)"
    print(f"Email: \"{email_text[:50]}...\"") # Print first 50 characters of the email
    print(f"  Actual: {actual_label}")
    print(f"  Predicted: {predicted_label}")
    print("-" * 20)
```

#OUTPUT

```
--- Case Study: Predictions on Test Data ---
Email: "Don't forget the grocery list...."
  Actual: Ham (Not Spam)
  Predicted: Ham (Not Spam)
--------------------
Email: "WINNER!! You have won a $1000 cash prize. Click he..."
  Actual: Spam
  Predicted: Ham (Not Spam)
--------------------
```

```
# --- How to use the model for new emails (Example) ---
print("\n--- Predicting New Emails ---")
new_emails = [
    "Hello, just checking in.",
    "Congratulations! You've won a million dollars, click this link now!",
    "Meeting reminder for tomorrow morning.",
    "Free money, no questions asked, guaranteed income!"
]
# Convert the new emails into numerical features using the SAME vectorizer
new_emails_counts = vectorizer.transform(new_emails)
# Make predictions
new_predictions = model.predict(new_emails_counts)
for email, prediction in zip(new_emails, new_predictions):
    label = "Spam" if prediction == 1 else "Ham (Not Spam)"
    print(f"Email: \"{email}\" -> Predicted: {label}")
```

#OUTPUT

```
--- Predicting New Emails ---
Email: "Hello, just checking in." -> Predicted: Spam
Email: "Congratulations! You've won a million dollars, click this link now!" -> Predicted: Ham (Not Spam)
Email: "Meeting reminder for tomorrow morning." -> Predicted: Ham (Not Spam)
Email: "Free money, no questions asked, guaranteed income!" -> Predicted: Ham (Not Spam)
```

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any four questions from following or teacher can design more questions if required to achieve CO)*

1. Which are the email spam filtering methods?
2. How to divide dataset into test set and train set?
3. How does a Naive Bayes classifier work particularly well for spam detection?
4. How do you preprocess raw emails for ML (e.g., tokenization, stopword removal)?
5. What are some popular datasets for email spam filtering?
6. What are effective techniques to clean and parse HTML or attachments in emails?

Space for Answers

_____

_____

_____

XI. References/Suggestions for further reading

- https://www.techtarget.com/searchsecurity/definition/spam
- https://www.darktrace.com/cyber-ai-glossary/email-spam
- https://www.cisco.com/site/us/en/learn/topics/security/what-is-spam.html#:~:text=Often%2C%20spam%20email%20is%20sent,gain%20access%20to%20your%20computer.
- https://www.kaggle.com/datasets/venky73/spam-mails-dataset
- https://www.quora.com/What-is-the-best-way-to-parse-through-emails
- https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 11: Create and display a Decision Tree on given dataset

I. Practical Significance:

Classification involves two main stages: a training phase and a prediction phase. During training, a model is built using labeled data to learn patterns and relationships. In the prediction phase, this model is applied to new or unseen data to determine the correct output or category. One of the simplest and most commonly used algorithms for classification is the Decision Tree. This method is widely favored for its ability to visually represent decision-making steps, making the results easy to interpret. Decision trees can also be applied to both classification and regression tasks, offering versatility in solving various types of problems.

II. Industry/Employer expected outcome(s):

- Train decision tree on the dataset to solve a specific business problem.
- Aligning the output with the business goal.

III. Course Level Learning Outcome (COs):

- CO4 - Create data model for Machine Learning Algorithms.

IV. Laboratory Learning Outcome (LLO):

- LLO 11.1-Implement Supervised Learning.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

A Decision Tree is a type of supervised learning algorithm commonly used in machine learning tasks. It works by organizing decisions in a tree-like structure, where each internal node represents a test on a specific feature, and each branch shows the outcome of that test. The leaf nodes represent the final output or target value. The process begins at the root node, where a data sample is evaluated. At every node, a condition based on one of the sample's features determines the path to the next node. This continues until the sample reaches a leaf, where a prediction is made. The algorithm learns by identifying the most effective conditions (or rules) at each step to divide the data, using criteria such as information gain or Gini index to decide how to split the data efficiently.

The decision trees can be divided, with respect to the target values, into:

- **Classification** trees used to classify samples, assign to a limited set of values - classes. In scikit-learn it is DecisionTreeClassifier.
- **Regression** trees used to assign samples into numerical values within the range. In scikit-learn it is DecisionTreeRegressor.

Decision trees are considered a fundamental tool in machine learning. They provide logical insights into complex datasets. It has a hierarchical tree structure with a root node, branches, internal nodes, and leaf nodes. Following are the Decision trees terminologies:

a. **Root node:** The root node is the beginning point of a decision tree where the whole dataset starts to divide based on different features present in the dataset.
b. **Decision nodes:** Nodes with children nodes represent a decision to be taken. The root node (if having children) is also a decision node.
c. **Leaf nodes:** Nodes that indicate the final categorization or result when additional splitting is not possible. Terminal nodes are another name for leaf nodes.
d. **Branches or subtrees:** A branch or subtree is a component of the decision tree that is part of the larger structure. Within the tree, it symbolizes a certain decision-making and result-oriented path.
e. **Pruning:** It is the practice of eliminating or chopping down particular decision tree nodes to simplify the model and avoid overfitting.
f. **Parent and child nodes:** In a decision tree, a node that can be divided is called a parent node, and nodes that emerge from it are called its child nodes. The parent node represents a decision or circumstance, and the child nodes represent possible outcomes or additional decisions based on that situation.

Decision trees are a popular tool in decision analysis. Decision trees are a popular tool in decision analysis. They help support decisions by providing a clear visual representation of all possible options and their outcomes. Below are 5 ways to visualize Decision Tree in Python:

1. print text representation of the tree with sklearn.tree.export_text method
2. plot with sklearn.tree.plot_tree method (matplotlib needed)
3. plot with sklearn.tree.export_graphviz method (graphviz needed)
4. plot with dtreeviz package (dtreeviz and graphviz needed)
5. plot with our own supertree package

Here are the step-by-step instructions to create a Decision Tree for classification or regression tasks using Python and scikit-learn.

1. Identify the goal as classification or regression.
2. Use a built-in dataset from Open Data Platforms for Free Public Data Sets or import from Python libraries like BeautifulSoup, Scrapy and sklearn or load your own (e.g. .CSV).
3. Preprocess the Data which includes – checking missing values, encoding categorical variables, dropping irrelevant columns.
4. Split or divide the data into features X and target y, then into training and testing sets.
5. Create and Train the Decision Tree
6. Make Predictions
7. Evaluate the Model
8. Interpret the Tree
9. If you want, save and Load the Model and Tune Hyper-parameters

Example 1: To load and prepare Titanic passenger data for analysis or modeling, the following code selects relevant features and ensures there are no missing values before further processing like building a machine learning model or exploratory data analysis.

Python Code:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
data = pd.read_csv('https://github.com/datagy/data/raw/main/titanic.csv', usecols=['Survived',
'Pclass', 'Gender', 'Age', 'SibSp', 'Fare', 'Embarked'])
data = data.dropna()
print(data.head())
```

#Output

	Survived	Pclass	Gender	Age	SibSp	Fare	Embarked
0	0	3	male	22.0	1	7.2500	S
1	1	1	female	38.0	1	71.2833	C
2	1	3	female	26.0	0	7.9250	S
3	1	1	female	35.0	1	53.1000	S
4	0	3	male	35.0	0	8.0500	S

Example 2: Following is the Python program to load, view, and understand the structure of the Wine dataset from scikit-learn.

Python Code:

```
import pandas as pd
from sklearn.datasets import load_wine
data = load_wine() # Loading dataset
# Converting data to a Data Frame to view properly
wine = pd.DataFrame(data['data'], columns = data['feature_names']) wine['target'] =
pd.Series(data['target'], name = 'target_values') # Configuring pandas to show all features
pd.set_option("display.max_rows", None, "display.max_columns", None)
print(wine.head())
print("Total number of observations: {}".format(len(wine)))
```

#Check the output for above program

Example 3: Python example using a Decision Tree to help decide an engineering branch after 12th based on some basic preferences like interest in subjects, math skills, and other criteria.

Python Code:

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
# Sample data: features and corresponding chosen branches
# Features: ['Math_Score', 'Physics_Interest', 'Chemistry_Interest', 'Creativity',
'Programming_Skills']
# Values are scaled 0 (low) to 10 (high)
```

```
data = {
    'Math_Score': [9, 8, 7, 6, 5, 4, 7, 8, 3, 6],
    'Physics_Interest': [8, 7, 9, 5, 6, 4, 8, 6, 3, 5],
    'Chemistry_Interest': [5, 4, 6, 9, 7, 8, 4, 3, 6, 7],
    'Creativity': [3, 2, 4, 8, 7, 9, 3, 2, 7, 6],
    'Programming_Skills': [7, 9, 6, 3, 2, 1, 9, 8, 1, 2],
    'Branch': [
        'Computer Science', 'Computer Science', 'Electrical', 'Chemical', 'Chemical', 'Biotech',
        'Computer Science', 'Electrical', 'Biotech', 'Chemical' ]
}
# Create DataFrame
df = pd.DataFrame(data)
# Features and target
X = df.drop('Branch', axis=1)
y = df['Branch']
# Train Decision Tree
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X, y)
# Print decision rules
rules = export_text(clf, feature_names=list(X.columns))
print("Decision Rules:\n", rules)
# Example: Predict branch for a new student
new_student = [[8, 7, 5, 3, 8]]  # Scores for the features in the order of columns
predicted_branch = clf.predict(new_student)
print("\nPredicted Branch for new student:", predicted_branch[0])
```

#Output

Decision Rules:

```
|--- Creativity <= 5.00
|   |--- Programming_Skills <= 8.50
|   |   |--- Math_Score <= 8.50
|   |   |   |--- class: Electrical
|   |   |--- Math_Score >  8.50
|   |   |   |--- class: Computer Science
|   |--- Programming_Skills >  8.50
|   |   |--- class: Computer Science
|--- Creativity >  5.00
|   |--- Physics_Interest <= 4.50
|   |   |--- class: Biotech
|   |--- Physics_Interest >  4.50
|   |   |--- class: Chemical
```

Predicted Branch for new student: Electrical

Example 4:

Let’s apply this algorithm to the wine dataset, which contains attributes of wine samples categorized into three classes [class_0, class_1, class_2]. We’ll use Python’s scikit-learn library

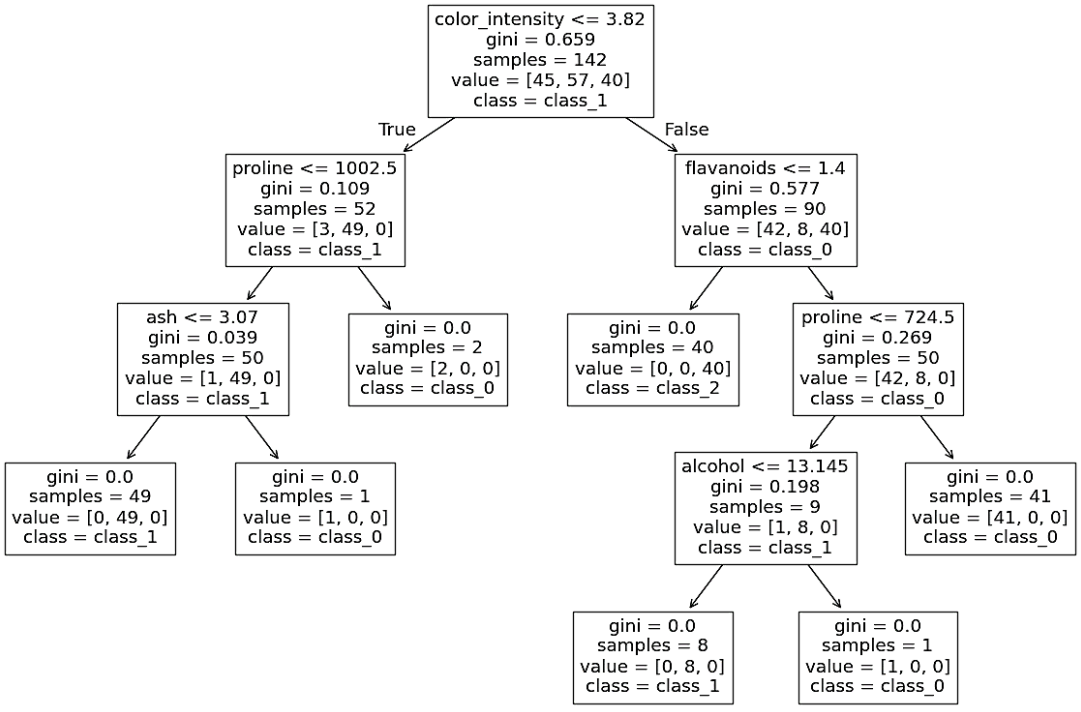
for implementing the decision tree classifier. The decision rule for classifying wines into particular classes using decision trees is detevrmined based on the attribute values of the wine characteristics.

For example, a decision rule could be that wines with certain levels of acidity, alcohol percentage, and color intensity belong to class_0, while wines with different attribute values belong to class_1 or class_2. These decision rules are learned during the training process of the decision tree algorithm based on the patterns and relationships found in the dataset.

Python Code:

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
wine = load_wine()
X = wine.data
y = wine.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Splitting the dataset into training and testing sets
clf = DecisionTreeClassifier() # Initialize the decision tree classifier
clf.fit(X_train, y_train) # Fitting the classifier on the training data
# Plot the decision tree
plt.figure(figsize=(15, 10))
plot_tree(clf, filled=True, feature_names=wine.feature_names, class_names=wine.target _
names)
plt.savefig("plot.png", bbox_inches='tight')
plt.show()
```

#Output



- **Applications of Decision Tree**
  - Medical Diagnosis – Predict diseases based on symptoms and test results
  - Customer Segmentation – Group customers for targeted marketing
  - Credit Risk Assessment – Evaluate loan applicant risk levels
  - Spam Detection – Classify emails as spam or not spam
  - Sales Forecasting – Predict future sales trends
  - Price Prediction – Estimate product or property prices
  - Energy Load Forecasting – Predict electricity demand
  - Churn Prediction – Identify customers likely to leave a service
  - Recommendation Systems – Suggest products or content
  - Fraud Detection – Spot potentially fraudulent transactions
  - Feature Selection – Identify important variables in datasets
  - Business Decision Support – Analyze strategic scenarios
  - Legal Risk Assessment – Predict recidivism or case outcomes
  - Genomic and Bioinformatics Analysis – Classify gene/protein functions
  - Health Monitoring Systems – Detect anomalies in health data

The time complexity of training a decision tree is typically O(m·n·log n), where n is the number of samples, m is the number of features, and the log n factor accounts for sorting during split selection; if the tree depth is limited to d, the complexity becomes O(m·n·d). The space complexity during training is O(n·m) to store the dataset, and the final tree structure requires O(n) space in the worst case. For prediction, the time complexity for a single sample is O(d), where d is the depth of the tree.

To summarize, using a decision tree offers a straightforward and easy-to-understand method for handling both classification and regression tasks. The algorithm works by dividing the dataset into smaller subsets based on feature values, making the decision-making process more transparent and interpretable. Techniques like depth-first or breadth-first search can be used to explore the tree's structure, helping to analyze how decisions are made, improve efficiency, and assess various outcomes along different branches.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any five questions from following or teacher can design more questions if required to achieve CO)*

1. What parameters can you tune in a decision tree?
2. What is the maximum depth of the tree, and why did you choose it?
3. Train a DecisionTreeClassifier (or DecisionTreeRegressor) from sklearn tree. What parameters did you use?
4. Load any dataset into a Panda's DataFrame. Print basic info using df.info() and df.describe().
5. Create a Decision Tree Classifier for Iris Dataset.
6. Consider the following table of observations:

No.	Outlook	Temperature	Humidity	Windy	Play Golf?
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	Y
4	rain	mild	high	false	Y
5	rain	cool	normal	false	Y
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	Y
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	Y
10	rain	mild	normal	false	Y
11	sunny	mild	normal	true	Y
12	overcast	mild	high	true	Y
13	overcast	hot	normal	false	Y
14	rain	mild	high	true	N

From the classified examples in the above table, construct two decision trees for the classification "Play Golf." For the first tree, use Temperature as the root node. (This is a really bad choice.) Continue, using your best judgment for selecting other attributes.

Remember that different attributes can be used in different branches on a given level of the tree. For the second tree, follow the Decision-Tree-Learning algorithm. As your Choose-Attribute function, choose the attribute with the highest information gain. Draw the decision trees.

Space for Answers

XI. References/Suggestions for further reading

- https://mljar.com/blog/visualize-decision-tree/
- https://www.datacamp.com/tutorial/decision-tree-classification-python
- https://datamapu.com/posts/classical_ml/decision_tree_classification_example/
- https://learningdaily.dev/scikit-learn-decision-tree-a-step-by-step-guide-92db8298e488
- https://machinelearningmodels.org/building-a-decision-tree-classifier-in-scikit-learn/#google_vignette
- https://colab.research.google.com/github/gumption/Python_for_Data_Science/blob/master/4_Python_Simple_Decision_Tree.ipynb

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		

Practical No. 12: Write program to implement K-means Algorithm

I. Practical Significance:

Every machine Learning engineer wants to achieve accurate predictions with their algorithms. Such learning algorithms are generally divided into two types: supervised and unsupervised. K-means clustering is one of the unsupervised algorithms where the available input data does not have a labeled response.

K-means clustering is a popular unsupervised learning algorithm used to group data points into clusters based on their similarity. It works iteratively by assigning data points to the nearest centroid (cluster center) and then recalculating the centroids based on the new cluster assignments. The algorithm continues this process until the cluster assignments no longer change significantly or a maximum number of iterations is reached.

II. Industry/Employer expected outcome(s):

- To implement K-means Algorithm.

III. Course Level Learning Outcome (COs):

- CO4- Create data model for Machine Learning Algorithms.

IV. Laboratory Learning Outcome (LLO):

- LLO 12.1- Develop program on Unsupervised Learning.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

- **Clustering**

  Clustering is like sorting a bunch of similar items into different groups based on their characteristics. In data mining and machine learning, it's a powerful technique used to group similar data points together, making it easier to find patterns or understand large datasets. Essentially, clustering helps identify natural groupings in your data.

- **Types of Clustering**

  Clustering is a type of unsupervised learning wherein data points are grouped into different sets based on their degree of similarity.

  The various types of clustering are:

  - **Hierarchical clustering**

    Hierarchical clustering uses a tree-like structure, like so:

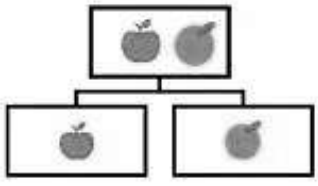


Fig.1: Hierarchical clustering

In agglomerative clustering, there is a bottom-up approach. We begin with each element as a separate cluster and merge them into successively more massive clusters, as shown below:

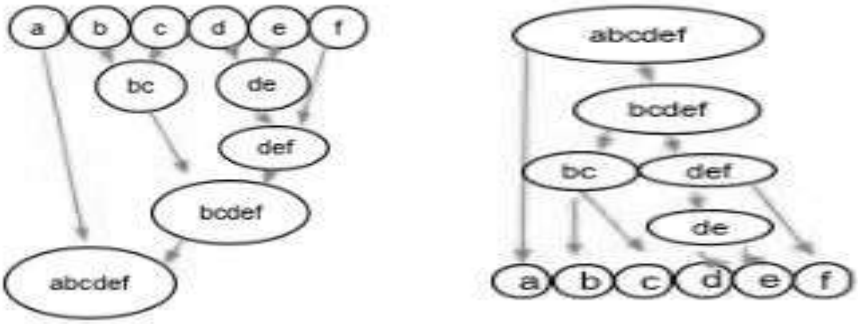


Fig.2: Divisive clustering

Divisive clustering is a top-down approach. We begin with the whole set and proceed to divide it into successively smaller clusters, as you can see in figure.

- **Partitioning Clustering**

  Partitioning clustering is split into two subtypes - K-Means clustering and Fuzzy C-Means. In k-means clustering, the objects are divided into several clusters mentioned by the number 'K.' So if we say K = 2, the objects are divided into two clusters, c1 and c2, as shown:

  Here, the features or characteristics are compared, and all objects having similar characteristics are clustered together.

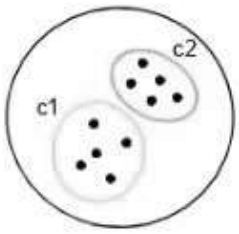


Fig.3: Partitioning Clustering

- **K-Means Clustering**

  K-means clustering is a way of grouping data based on how similar or close the data points are to each other. Imagine you have a bunch of points, and you want to group them into clusters. The algorithm works by first randomly picking some central points (called centroids) and then assigning every data point to the nearest centroid.

  Once that's done, it recalculates the centroids based on the new groupings and repeats the process until the clusters make sense. It's a pretty fast and efficient method, but it works best when the clusters are distinct and not too mixed up. One challenge, though, is figuring out the right number of clusters (K) beforehand. Plus, if there's a lot of noise or overlap in the data, K Means might not perform as well.

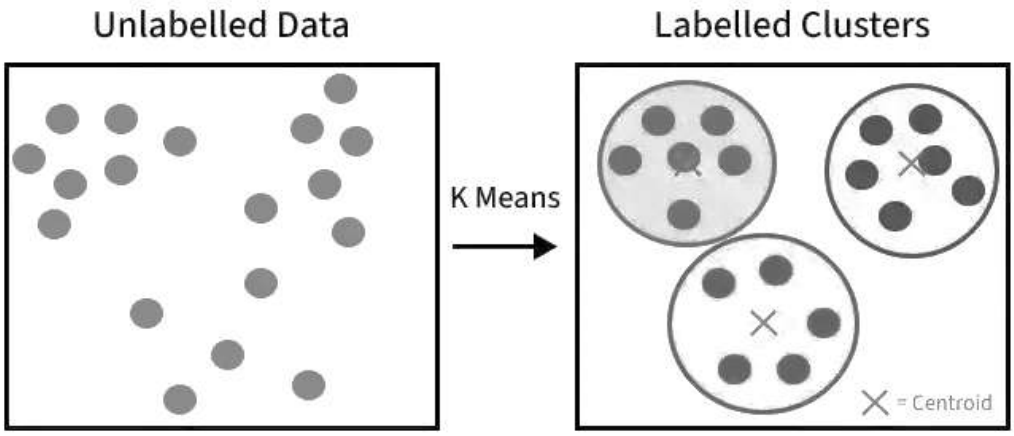


Fig 4: K-Means Clustering

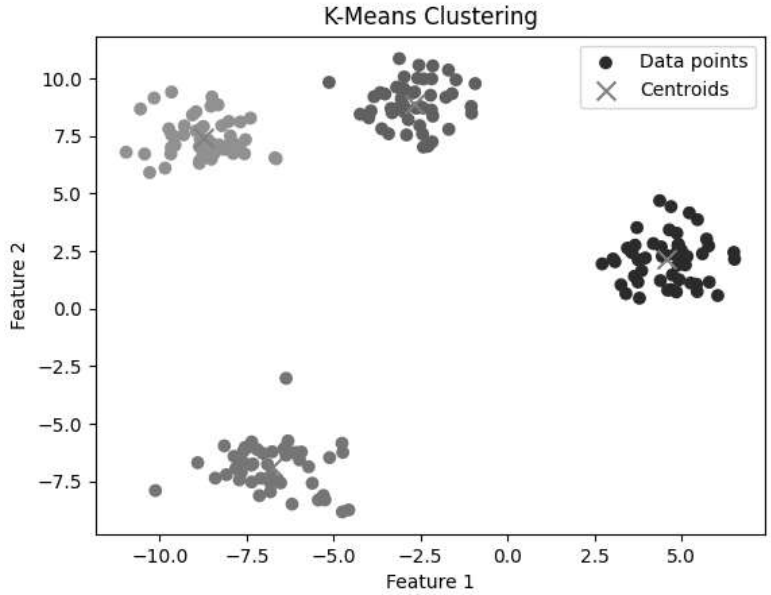
- **K-Means Clustering Algorithm**
  Let's say we have x1, x2, x3…… x(n) as our inputs, and we want to split this into K clusters.
  The steps to form clusters are:
  Step 1: Choose K random points as cluster centers called centroids.
  Step 2: Assign each x(i) to the closest cluster by implementing euclidean distance (i.e., calculating its distance to each centroid)
  Step 3: Identify new centroids by taking the average of the assigned points.
  Step 4: Keep repeating step 2 and step 3 until convergence is achieved

- **Program Code:**

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
# Generate sample data (replace with your actual data)
data, _ = make_blobs(n_samples=200, centers=4, random_state=42)
# Choose the number of clusters (k) - use elbow method or silhouette analysis for optimal k
k = 4
# Initialize KMeans
kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
# Fit the KMeans model to the data
kmeans.fit(data)
# Get cluster labels
labels = kmeans.labels_
# Get cluster centers
centers = kmeans.cluster_centers_
# Plot the results
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis', label='Data points')
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=100, label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
```

```
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

#Output



VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any four questions from following or teacher can design more questions if required to achieve CO)*

1. What does K stand for in the algorithm?
2. What is an example of k-means in real life?
3. What is the difference between KNN and K-Means?
4. What is the K-Means clustering principle?
5. What are centroids in K-Means?
6. What is the time complexity of the K-Means algorithm?
7. Can K-Means be used for non-convex shaped clusters? Why or why not?
8. What are some limitations of the K-Means algorithm?
9. In what situations might K-Means clustering fail?

Space for Answers

XI. References/Suggestions for further reading

- https://www.simplilearn.com/tutorials/machine-learning-tutorial/k-means-clustering-algorithm
- https://www.geeksforgeeks.org/machine-learning/k-means-clustering-introduction/
- https://domino.ai/blog/getting-started-with-k-means-clustering-in-python
- https://www.w3schools.com/python/python_ml_k-means.asp
- https://youtu.be/4b5d3muPQmA?si=EO6hB8eW4fs86n5P
- https://youtu.be/R2e3Ls9H_fc?si=1vOQA-UF1rivyTzf
- https://youtu.be/5FpsGnkbEpM?si=2swOE5O9n7fdI1rw

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 13: Write program to calculate cross validation score for any Dataset like IRIS

I. Practical Significance:

Cross-validation is a technique used to check how well a machine learning model performs on unseen data. It splits the data into several parts, trains the model on some parts and tests it on the remaining part repeating this process multiple times. Finally the results from each validation step are averaged to produce a more accurate estimate of the model's performance.

The main purpose of cross validation is to prevent overfitting. If you want to make sure your machine learning model is not just memorizing the training data but is capable of adapting to real-world data cross-validation is a commonly used technique.

II. Industry/Employer expected outcome(s):

- To calculate cross validation score for any Dataset like IRIS

III. Course Level Learning Outcome (COs):

- CO4- Create data model for Machine Learning Algorithms.

IV. Laboratory Learning Outcome (LLO):

- LLO 13.1- Implement cross validation in python

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

- **Cross validation**

  Cross validation is an important topic in the Machine Learning to ensure that our model is robust enough. Cross-validation is a predictive assessment technique used in machine learning to estimate the capabilities of a machine learning model. If you work in machine learning, you can use cross-validation as a statistical model to compare and select machine learning models for a specific application. Traditional training strategy is using 3 parts of the dataset for training, testing a validation.

  Training set - is used to train the model and optimize the hyper parameters of the model

  Testing set - is used to evaluate that the models generalize enough to correctly work on data it was not trained on. However through the person doing the optimization some knowledge about the test set eventually leaks into the model.

  Validation set - for that reason we use validation set which is used as the final check that model is able to generalize with previously unknown data.

- **Types of Cross-Validation**

  There are several types of cross validation techniques which are as follows:

  1. Holdout Validation

  In Holdout Validation we perform training on the 50% of the given dataset and rest 50% is used for the testing purpose. It's a simple and quick way to evaluate a model. The major

drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model that can lead to higher bias.

2. LOOCV (Leave One Out Cross Validation)

In this method we perform training on the whole dataset but leaves only one data-point of the available dataset and then iterates for each data-point. In LOOCV the model is trained on n−1 samples and tested on the one omitted sample repeating this process for each data point in the dataset. It has some advantages as well as disadvantages also.

- An advantage of using this method is that we make use of all data points and hence it is low bias.
- The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation.
- Another drawback is it takes a lot of execution time as it iterates over the number of data points we have.

3. Stratified Cross-Validation

It is a technique used in machine learning to ensure that each fold of the cross-validation process maintains the same class distribution as the entire dataset. This is particularly important when dealing with imbalanced datasets where certain classes may be under represented.

In this method:

- The dataset is divided into k folds while maintaining the proportion of classes in each fold.
- During each iteration, one-fold is used for testing and the remaining folds are used for training.
- The process is repeated k times with each fold serving as the test set exactly once.

Stratified Cross-Validation is essential when dealing with classification problems where maintaining the balance of class distribution is crucial for the model to generalize well to unseen data.

4. K-Fold Cross Validation

In K-Fold Cross Validation we split the dataset into k number of subsets known as folds then we perform training on the all the subsets but leave one (k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Note: It is always suggested that the value of k should be 10 as the lower value of k takes towards validation and higher value of k leads to LOOCV method.

- **Sample Program**

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np
```

```
# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize a model (e.g., Logistic Regression)
model = LogisticRegression(solver='liblinear', multi_class='ovr')

# Calculate cross-validation scores (e.g., 5-fold cross-validation)
cv_scores = cross_val_score(model, X_train, y_train, cv=5)

# Print the cross-validation scores
print("Cross-validation scores:", cv_scores)

# Print the mean cross-validation score
print("Mean cross-validation score:", np.mean(cv_scores))

# Fit the model on the entire training set
model.fit(X_train, y_train)

# Evaluate the model on the test set
accuracy = model.score(X_test, y_test)
print("Test set accuracy:", accuracy)
```

#Output

```
Cross-validation scores: [0.91666667 0.95833333 0.91666667 0.95833333 1.]
Mean cross-validation score: 0.95
Test set accuracy: 1.0
```

- **Program Overview**
  - **Import Libraries:** Import necessary modules from scikit-learn for dataset loading, model selection, and cross-validation. Also, import numpy for numerical operations.
  - **Load Dataset:** Load the Iris dataset using load_iris(). The features are assigned to X, and the target labels to y.
  - **Initialize Model:** A Logistic Regression model is initialized.
  - **Define Cross-Validation Strategy:** Stratified K-Fold cross-validation is used to ensure that each fold has a representative distribution of classes. The dataset is divided into 5 folds, shuffled and a random state is set for reproducibility.
  - **Calculate Cross-Validation Scores:** cross_val_score is used to calculate the score for each fold. This function takes the model, features, target, and cross-validation strategy as input.
  - **Print Results:** The individual scores for each fold and the mean score are printed.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any three questions from following or teacher can design more questions if required to achieve CO)*

1. What is cross validation and why is it important in model evaluation?
2. Describe the types of cross validation?
3. Describe Leave one out cross validation with suitable example?
4. Which are the advantage and disadvantage of cross validation?
5. What is negative and positive cross validation?

Space for Answers

_____

_____

_____

_____

_____

XI. References/Suggestions for further reading

- https://github.com/vaasha/Machine-leaning-in-examples/blob/master/sklearn/cross-validation/Cross%20Validation.ipynb
- https://scikit-learn.org/stable/modules/cross_validation.html
- https://www.geeksforgeeks.org/cross-validation-machine-learning/
- https://www.kaggle.com/code/jsanjanaa/k-fold-cross-validation-in-iris-dataset
- https://pianalytix.com/k-fold-cross-validation/

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 14: Write program to implement Simple Linear Regression using Python

I. Practical Significance:

Simple linear regression is a statistical method used to model the relationship between a dependent variable and a single independent variable. It assumes a linear relationship, meaning the change in the dependent variable is proportional to the change in the independent variable.

This method enables the identification and measurement of the connection between two variables, facilitating predictions and insights into how alterations in one influence the other. As a fundamental statistical technique, it is widely applied across disciplines for purposes such as forecasting, examining interrelationships, and guiding data-driven decisions.

II. Industry/Employer expected outcome(s):

- To implement simple linear regression using python.

III. Course Level Learning Outcome (COs):

- CO5- Classify the data by performing different Regression Techniques.

IV. Laboratory Learning Outcome (LLO):

- LLO 14.1- Develop program for Simple Linear Regression.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

- **Simple Linear Regression**

  Simple Linear Regression is a statistical method that models the relationship between a dependent variable and a single independent variable, typically represented by a straight-line equation. For instance, a business might use this approach to assess whether increased advertising spending correlates with higher sales figures.

  The below graph explains the relationship between advertising expenditure and sales using simple linear regression:

  The relationship between the dependent and independent variables is represented by the simple linear equation:

  $$y=mx+b$$

  Here:

  - y is the predicted value (dependent variable).
  - m is the slope of the line
  - x is the independent variable.
  - b is the y-intercept (the value of y when x is 0).

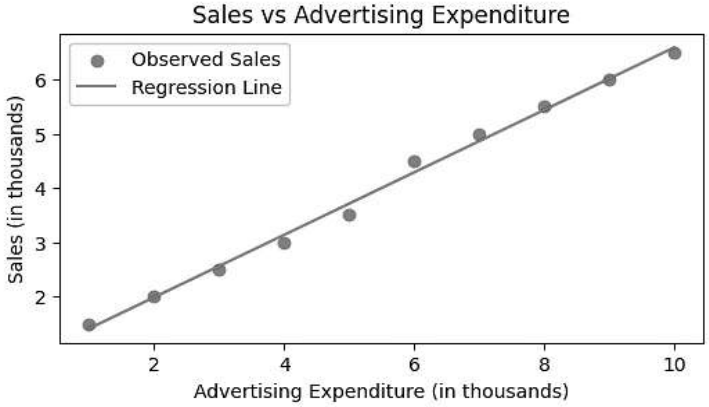


Fig.1: Simple Linear Regression

In this equation m signifies the slope of the line indicating how much y changes for a one-unit increase in x, a positive m suggests a direct relationship while a negative m indicates an inverse relationship.

To better understand this relationship, we can express it in a more statistical context using the linear regression formula:

$$Y = \beta_0 + \beta_1 x$$

In simple linear regression the parameters β_0 and β_1 play crucial roles in defining the relationship between the independent variable X and the dependent variable Y in the regression equation.

Intercept β_0:

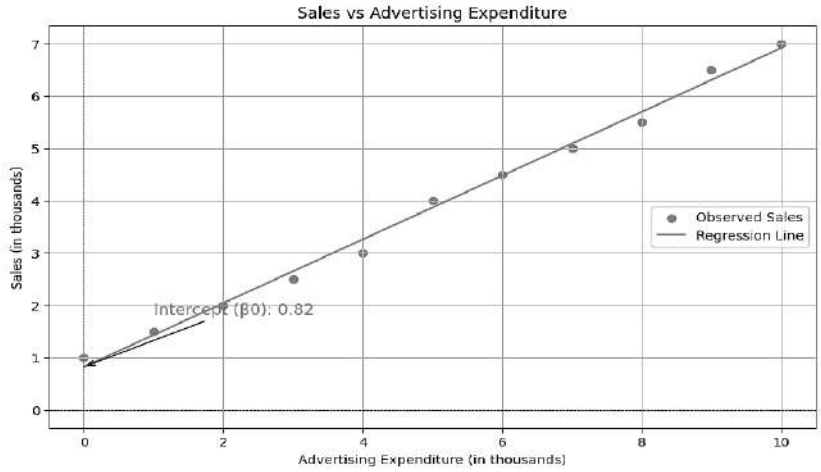


Fig.2: Linear Regression Plot

- The intercept β_0 represents the predicted value of the dependent variable Y when the independent variable X is zero. In other words, it is the point where the regression line crosses the y-axis.
- It provides a baseline value for Y and helps us understand the expected outcome when there is no influence from X. For example, if you were predicting sales based on advertising expenditure it would indicate the estimated sales when no money is spent on advertising.

Slope β_1:

- A positive β_1 value suggests that as X increases, Y also increases, indicating a direct relationship.
- Conversely, a negative β_1 value indicates that an increase in X leads to a decrease in Y, indicating an inverse relationship.

For instance, if $\beta_1=2$, this would mean that for every additional unit spent on advertising, sales are expected to increase by 2 units.

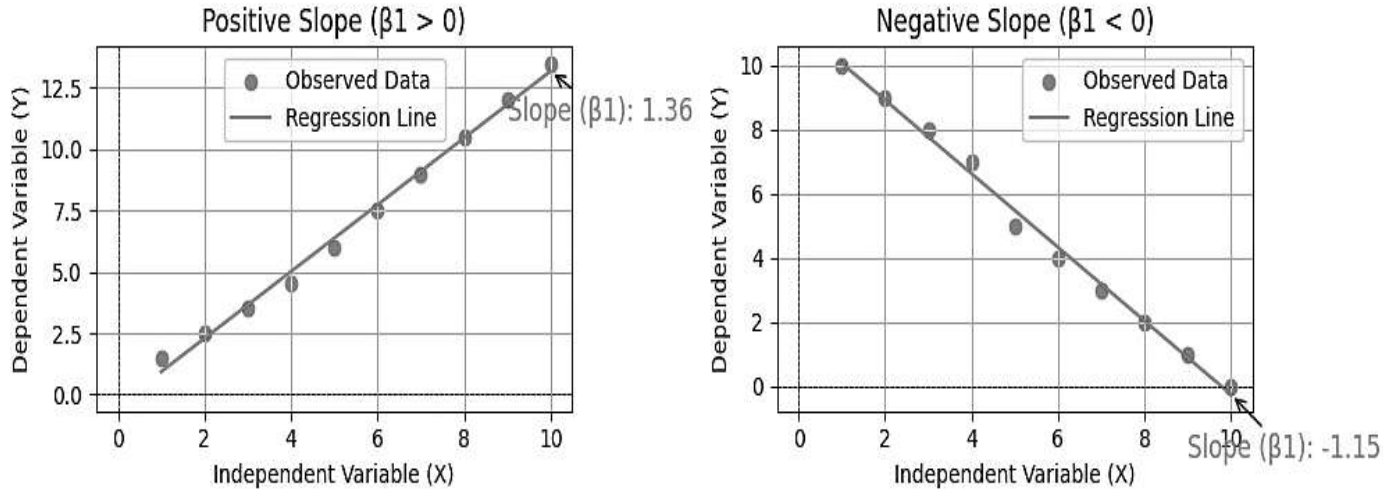


Fig.2: Type of slope

Linear regression can be used to predict sales based on advertising data by modeling the relationship between advertising spending (on TV, radio, newspaper, etc.) and sales figures. This involves using historical data to train a model that can then forecast future sales based on new advertising spending inputs.

Program Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
# 1. Data Loading and Preparation
df = pd.read_csv('advertising.csv') # Replace with your file path
X = df[['TV', 'Radio', 'Newspaper']]
y = df['Sales']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
# 2. Model Training
model = LinearRegression()
model.fit(X_train, y_train)
# 3. Model Evaluation
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
```

```
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
print(f"R-squared: {r2}")
# 4. Making Predictions
new_data = pd.DataFrame({'TV': [100, 200], 'Radio': [50, 100], 'Newspaper': [20, 40]})
predictions = model.predict(new_data)
print("Predicted Sales:", predictions)
```

#Output

```
Mean Squared Error: 14.43692607720132
Root Mean Squared Error: 3.7995955149464686
R-squared: -1.8517384843854452
Predicted Sales: [19.7314462  41.20888503]
```

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3 or i5 with RAM 4 GB or more and HDD space of 10 GB	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

X. Practical related Questions:

(*Note: Teacher shall assign any three questions from following or teacher can design more questions if required to achieve CO)*

1. What is the equation of a simple linear regression model?
2. What do the coefficients (β_0 and β_1) represent?
3. What are polynomial regression and when is it used?
4. What is the difference between underfitting and overfitting in regression?
5. What is the role of the error term in linear regression?
6. How do you evaluate the performance of a linear regression model?

Space for Answers

XI. References/Suggestions for further reading

- https://www.w3schools.com/python/python_ml_linear_regression.asp
- https://www.geeksforgeeks.org/machine-learning/simple-linear-regression-in-python/
- https://medium.com/@shuv.sdr/simple-linear-regression-in-python-a0069b325bf8
- https://realpython.com/linear-regression-in-python/
- https://youtu.be/7ArmBVF2dCs?si=NwxQeGnN_Rubg-z8
- https://youtu.be/qxo8p8PtFeA?si=omDiE_ZU_gz2OZbD
- https://youtu.be/cA09_IdteV0?si=TnYIkesyiDhv2kp8
- https://youtu.be/O2Cw82YR5Bo?si=bveDd5_p49R-N9M9

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 15: Write program to implement Multiple Linear Regression using Python

I. Practical Significance:

Linear regression is a statistical method employed to predict outcomes by modeling the relationship between a dependent variable and a single independent variable through a linear equation. When multiple independent variables are involved, the approach extends to Multiple Linear Regression. This technique enables the assessment of how various factors collectively impact the dependent variable.

Multiple linear regression is a statistical technique that examines how a dependent variable is influenced by two or more independent variables. By fitting a linear equation to the observed data, it allows for the analysis of how variations in the input variables impact the outcome.

II. Industry/Employer expected outcome(s):

- To implement multiple linear regression using python.

III. Course Level Learning Outcome (COs):

- CO5- Classify the data by performing different Regression Techniques.

IV. Laboratory Learning Outcome (LLO):

- LLO 15.1- Implement Multiple Linear Regression in Python.

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.

VI. Relevant Theoretical Background:

- **Multiple Linear Regression**

  Steps to perform multiple linear regression are similar to that of simple linear Regression but difference comes in the evaluation process. We can use it to find out which factor has the highest influence on the predicted output and how different variables are related to each other. Equation for multiple linear regression is:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

  Where,

  - y is the dependent variable
  - $X_1, X_2, \cdots X_n$ are the independent variables
  - $\beta 0$ is the intercept
  - $\beta 1, \beta 2, \cdots \beta n$ are the slopes

  The primary objective of the algorithm is to determine the optimal linear equation that accurately predicts outcomes using the independent variables. A regression model is trained on

a dataset containing known input (X) and output (y) pairs, enabling it to forecast y values for new, unseen X inputs.

In multiple regression model we may encounter categorical data such as gender (male/female), location (urban/rural), etc. Since regression models require numerical inputs then categorical data must be transformed into a usable form. This is where Dummy Variables used. These are binary variables (0 or 1) that represent the presence or absence of each category. For example:

- Male: 1 if male, 0 otherwise
- Female: 1 if female, 0 otherwise

Gender	Male	Female
Male	1	0
Male	1	0
Female	0	1
Female	0	1
Male	1	0
Female	0	1
Male	1	0

Table 1: categorical data

In the case of multiple categories, we create a dummy variable for each category excluding one to avoid multicollinearity. This process is called one-hot encoding which converts categorical variables into a numerical format suitable for regression models.

Multicollinearity in Multiple Linear Regression

Multicollinearity arises when two or more independent variables are highly correlated with each other. This can make it difficult to find the individual contribution of each variable to the dependent variable.

To detect multicollinearity we can use:

1. **Correlation Matrix:** A correlation matrix helps to find relationships between independent variables. High correlations (close to 1 or -1) suggest multicollinearity.
2. **VIF (Variance Inflation Factor):** VIF quantifies how much the variance of a regression coefficient increases if predictors are correlated. A high VIF typically above 10 indicates multicollinearity.

Assumptions of Multiple Regression Model

Similar to simple linear regression we have some assumptions in multiple linear regression which are as follows:

1. **Linearity**: Relationship between dependent and independent variables should be linear.
2. **Homoscedasticity**: Variance of errors should remain constant across all levels of independent variables.
3. **Multivariate Normality**: Residuals should follow a normal distribution.
4. **No Multicollinearity**: Independent variables should not be highly correlated.

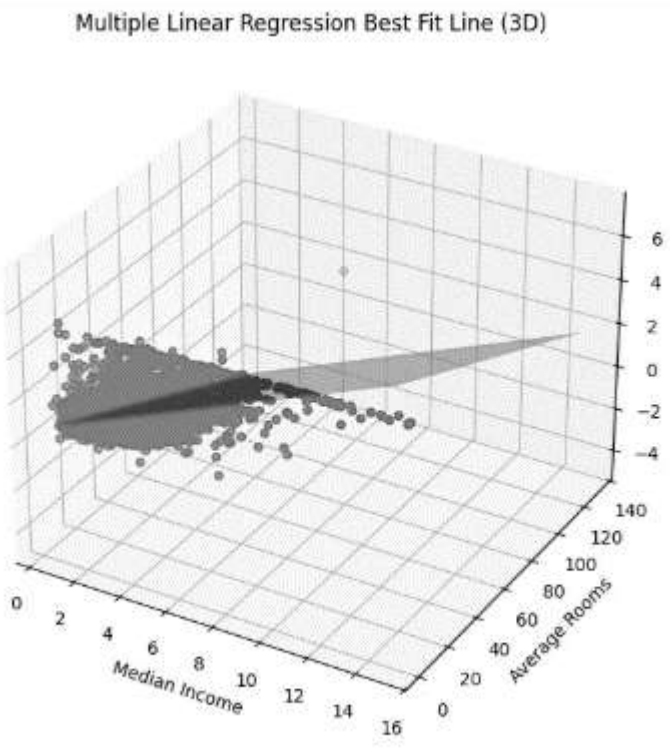
Generic Algorithm Steps:

Step1: Importing Libraries
Step2: Loading Dataset
Step3: Selecting Features for Visualization
Step4: Train-Test Split
Step5: Initializing and Training Model
Step6: Making Predictions
Step7: Visualizing Best Fit Line in 3D

Program Code:

```
import numpy as np    #importing packages
import pandas as pd #importing packages
import matplotlib.pyplot as plt   #importing packages
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.datasets import fetch_california_housing

california_housing = fetch_california_housing()
X = pd.DataFrame(california_housing.data, columns=california_housing.feature_names)
y = pd.Series(california_housing.target)
X = X[['MedInc', 'AveRooms']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_test['MedInc'], X_test['AveRooms'], y_test, color='blue', label='Actual Data')
x1_range = np.linspace(X_test['MedInc'].min(), X_test['MedInc'].max(), 100)
x2_range = np.linspace(X_test['AveRooms'].min(), X_test['AveRooms'].max(), 100)
x1, x2 = np.meshgrid(x1_range, x2_range)
z = model.predict(np.c_[x1.ravel(), x2.ravel()]).reshape(x1.shape)

ax.plot_surface(x1, x2, z, color='red', alpha=0.5, rstride=100, cstride=100)
ax.set_xlabel('Median Income')        #Label for graph x axis
ax.set_ylabel('Average Rooms')        #Label for graph y axis
ax.set_zlabel('House Price')          #Label for graph z axis
ax.set_title('Multiple Linear Regression Best Fit Line (3D)')
plt.show()       #For plotting graph
```

Output:



Visualizing Multiple Linear Regression

Multiple Linear Regression effectively captures how several factors together influence a target variable which helps in providing a practical approach for predictive modeling in real-world scenarios.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

X. Practical related Questions:

(*Note: Teacher shall assign any three questions from following or teacher can design more questions if required to achieve CO)*

1. What is multicollinearity, and how can it be detected?
2. How do you interpret the coefficients in a multiple regression model?
3. Difference between simple and multiple regression.
4. How many variables can be used in multiple regression?
5. What is p-value in multiple regression?
6. Why do we use multiple regression?

Space for Answers

XI. References/Suggestions for further reading

- https://www.w3schools.com/python/python_ml_multiple_regression.asp
- https://www.geeksforgeeks.org/machine-learning/ml-multiple-linear-regression-using-python/
- https://www.scribbr.com/statistics/multiple-linear-regression/#:~:text=What%20is%20multiple%20linear%20regression,variables%20using%20a%20straight%20line.`
- https://youtu.be/i3IadpjctWg?si=11b66g7iFdArvsYo
- https://youtu.be/MgsKmqEcQiE?si=ZT5ZRlnqU5QgN_ig
- https://youtu.be/fYStutigCkE?si=noIgoOKZ1lxzvKTD

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
1.	Expected output	10%	
2.	Timely completion of practical	10%	
3.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated sign of Course Teacher**		

Practical No. 16: Write program to create confusion matrix to calculate different measures to quantify the quality of the model

I. Practical Significance:

A confusion matrix is a key tool for evaluating the performance of machine learning models, especially in classification tasks, where the goal is to assign data points to specific categories. It provides a detailed summary of prediction results by showing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) in a 2×2 table for binary classification problems. By analyzing these four values, we can calculate several important metrics such as accuracy, precision, recall, and F1-score, offering a deeper understanding of how well the model performs. The term "confusion matrix" reflects the idea that it reveals how often the model is confusing one class for another. Since classification models are not always perfect, this matrix helps highlight both correct and incorrect predictions by comparing actual labels with predicted labels.

II. Industry/Employer expected outcome(s):

- Improve decision making.

III. Course Level Learning Outcome (COs):

- CO5 - Classify the data by performing different Regression Techniques.

IV. Laboratory Learning Outcome (LLO):

- LLO 16.1-Implement program for confusion matrix..

V. Relevant Affective Domain related Outcome(s):

- Follow safety practices.
- Demonstrate working as leader or a team leader.
- Follow ethical practices.
- Translate problems into ML tasks.

VI. Relevant Theoretical Background:

A confusion matrix is a straightforward tool used to assess the performance of a classification model. In machine learning, classification involves assigning data points to predefined categories based on their features. The confusion matrix compares the model's predicted outputs with the actual outcomes, helping identify which predictions were correct and where the model went wrong. This breakdown gives insight into the types of errors the model is making, which is useful for improving its accuracy. The matrix organizes the results into four key groups:

- **True Positive (TP):** The model correctly predicted a positive outcome i.e. the actual outcome was positive.
- **True Negative (TN):** The model correctly predicted a negative outcome i.e the actual outcome was negative.
- **False Positive (FP):** The model incorrectly predicted a positive outcome i.e the actual outcome was negative. It is also known as a Type I error.

- **False Negative (FN):** The model incorrectly predicted a negative outcome i.e the actual outcome was positive. It is also known as a Type II error.

The confusion matrix is also useful for computing important evaluation metrics such as accuracy**,** precision**,** recall**,** and the F1 score**,** which are especially valuable when dealing with imbalanced datasets**.** These metrics provide a clearer picture of how well a machine learning model is performing. Accuracy reflects the overall percentage of correct predictions. Precision indicates how many of the predicted positive results are actually correct. Recall shows how many of the actual positive cases the model was able to identify. The F1 score combines both precision and recall into a single metric by calculating their harmonic mean, giving a balanced measure of model performance. They are as follows:

Measure	Formula	What It Tells You
Accuracy	$\frac{TP + TN}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$	Overall how often the classifier is correct
Precision	$\frac{TP}{\text{TP} + \text{FP}}$	How many predicted positives are actually correct
Recall (Sensitivity, TPR)	$\frac{TP}{\text{TP} + \text{FN}}$	How well the model identifies actual positives
Specificity (TNR)	$\frac{TN}{\text{TN} + \text{FP}}$	How well the model identifies actual negatives
F1 Score	$\frac{2 * Precision * Recall}{Precision + Recall}$	Harmonic mean of precision and recall; balances both
False Positive Rate (FPR)	$\frac{FP}{\text{TN} + \text{FP}}$	Proportion of negatives incorrectly classified as positives
False Negative Rate (FNR)	$\frac{FN}{\text{FN} + \text{TP}}$	Proportion of positives incorrectly classified as negatives
Negative Predictive Value (NPV)	$\frac{TN}{\text{TN} + \text{FN}}$	Probability that a predicted negative is actually negative

Table 1: Summary of Confusion Matrix Evaluation Metrics

Among the most traditional and early uses of classification is email segregation. Emails were often required to be classified as spam or not spam (i.e., being worthy of being in the inbox). It is done by developing a classifier model that goes through the content to identify if the email is spam or not.

To develop such a classification model, a large amount of well-labeled email is required so that in the training phase, enough data is available. As inherently this is a text-based problem, the email content is converted into features where each feature is a word.

The observation for the model is whether a particular word was present / how many times the word was present in a document.

- **Types of Classification in Machine Learning**
  1. **Binary Classification:** Predicts one of two possible classes (e.g., Yes/No, Spam/Not Spam, Fraud/Not Fraud).

Example Algorithms: Logistic Regression, Support Vector Machines (SVM), Decision Trees.

2. **Multi-Class Classification:** Predicts more than two classes, where each instance belongs to exactly one class.
   Example Algorithms: Random Forest, Naive Bayes, k-Nearest Neighbors (k-NN), Neural Networks.
3. **Multi-Label Classification:** A single instance can belong to multiple classes simultaneously. E. g. A movie can be tagged as both "Action" and "Comedy".
   Example Algorithms: Binary Relevance, Classifier Chains, Deep Learning models.
4. **Imbalanced Classification:** Deals with datasets where one class is significantly underrepresented compared to others. E. g. Fraud detection (fraud cases are rare).
   Techniques: SMOTE (Synthetic Minority Oversampling), Cost-Sensitive Learning.
5. **Hierarchical Classification:** Classes are structured in a hierarchy (parent-child relationships). E. g. Organizing products into categories and subcategories.
   Approach: Uses tree-based classifiers or recursive methods.

- **Creating a Confusion Matrix:**

```
import matplotlib.pyplot as plt
import numpy
from sklearn import metrics
actual = numpy.random.binomial(1,.9,size = 1000)
predicted = numpy.random.binomial(1,.9,size = 1000)
confusion_matrix = metrics.confusion_matrix(actual, predicted)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix  = confusion_matrix,
display_labels = [0, 1])
cm_display.plot()
plt.show()
Accuracy = metrics.accuracy_score(actual,  predicted)
print("Accuracy is: ",Accuracy)
```

#Output shown as accuracy and fig. 1

```
Accuracy is:  0.815
```

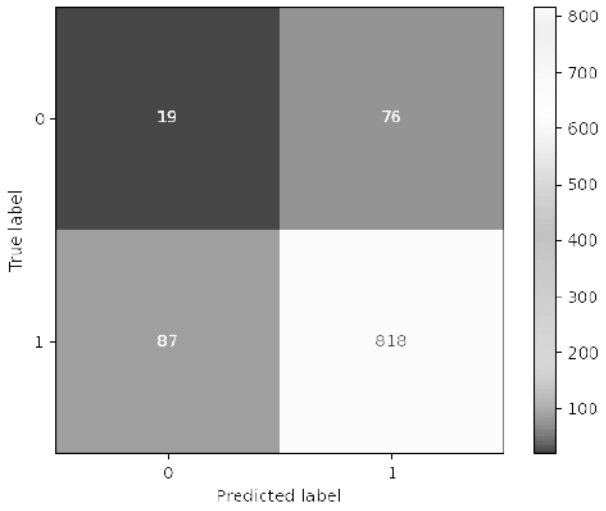


Fig. 1

Implementing confusion matrices in Python is straightforward with libraries such as Sklearn, TensorFlow, and Keras. We'll explore into a practical example using Sklearn to assess a movie recommendation system's performance.

Steps to create a Confusion Matrix:

1. First, import the essential modules
2. Next, prepare arrays for true and predicted labels
3. Then, generate the confusion matrix
4. Calculate key metrics

Using above steps, following example creates a Confusion Matrix for user input in the form of an array values using numpy.

- **PythonCode:**

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
import numpy as np
y_true = np.array([1, 1, 0, 1, 0, 0, 1, 0, 1, 0])
y_pred = np.array([1, 1, 0, 0, 1, 1, 1, 0, 1, 0])
cm = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", cm)
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

#Output

```
Confusion Matrix:
 [[3 2]
 [1 4]]
Accuracy: 0.70
Precision: 0.67
Recall: 0.80
F1 Score: 0.73
```

- **Applications of Confusion Matrix:**
  1. **Model Evaluation:** The primary application of the confusion matrix is to evaluate the performance of a classification model. It provides insights into the model's accuracy, precision, recall, and F1-score.
  2. **Medical Diagnosis:** The confusion matrix finds extensive use in medical fields for diagnosing diseases based on tests or images. It aids in quantifying the accuracy of diagnostic tests and identifying the balance between false positives and false negatives.

3. **Fraud Detection:** Banks and financial institutions use confusion matrices to detect fraudulent transactions by showcasing how AI algorithms help identify patterns of fraudulent activities.
4. **Natural Language Processing (NLP):** NLP models use confusion matrices to evaluate sentiment analysis, text classification, and named entity recognition.
5. **Customer Churn Prediction:** Confusion matrices play a pivotal role in predicting customer churn and show how AI-driven models use historical data to anticipate and mitigate customer attrition.
6. **Image and Object Recognition:** Confusion matrices assist in training models to identify objects in images, enabling technologies like self-driving cars and facial recognition systems.
7. **A/B Testing:** A/B testing is crucial for optimizing user experiences. Confusion matrices help analyses the results of A/B tests, enabling data-driven decisions in user engagement strategies.

The time complexity of creating a confusion matrix is O(n), where n is the number of predictions, since each prediction involves a constant-time comparison between the actual and predicted class. For binary classification, the space complexity is O(1) as only four values (TP, TN, FP, FN) are stored. In multi-class classification with k classes, the space complexity becomes $O(k^2)$ because a k X k matrix is required to capture all possible actual vs. predicted class combinations. Despite this, confusion matrices are generally efficient and scalable, especially since most real-world problems involve a relatively small number of classes.

In conclusion, the confusion matrix is a valuable tool for evaluating the performance of ML models. Understanding its components and the metrics derived from it, such as accuracy, precision, recall, and F1 score, allows us to determine whether an ML model is suitable for a specific purpose. By analyzing and improving the results of the confusion matrix, we can optimize ML algorithms and ensure accurate predictions in real-world applications.

VII. Required Resources:

Sr. No.	Name of Resource	Specification	Quantity	Remark (if any)
1.	Computer System	Computer with Processor i3/ i5 or above with 4 GB or more RAM and HDD space of 10 GB.	01 for each student	
2.	Operating System	Windows 8 or later/Linux		
3.	Python Interpreter/ IDLE	Any open source IDE such as IDLE, Jupyter Notebook, Spyder, PyCharm, Thonny, vscode, Google Colab etc.		
4.	Printer	Laser/Inkjet Printer A4 size	01 for Batch	

VIII. Precautions to be followed:

1. Handle computer system with care.
2. Strictly follow the instructions for writing, compiling, and executing the program.
3. Start and Shutdown system with proper procedure.
4. Don't forget to save file before execution.

IX. Conclusion:

_____

_____

_____

_____

X. Practical related Questions:

(*Note: Teacher shall assign any five questions from following or teacher can design more questions if required to achieve CO)*

1. How is a confusion matrix used for binary classification?
2. What are the key components of a confusion matrix?
3. How can confusion matrices be visualized?
4. Why are confusion matrices important for model evaluation?
5. Develop a binary classification model to predict whether an email is spam or not spam assuming suitable data.
6. A hospital uses a machine learning model to predict whether a patient has COVID-19. Out of 200 patients, the model correctly predicted 70 true positives, 100 true negatives, but also produced 20 false positives and 10 false negatives. Based on this information:
   a. What is the model's accuracy, precision, recall, specificity, and F1 score?
   b. If the hospital prioritizes identifying all actual COVID-19 cases (minimizing false negatives), is this model suitable?
   c. What are the possible consequences of the model's false positives and false negatives in a real-world healthcare setting?
7. For the confusion matrix shown below, determine the accuracy, precision, recall and F1-score.

n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	**50**	**10**
Actual: YES	**5**	**100**

Space for Answers

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

XI. References/Suggestions for further reading

- https://www.w3schools.com/python/python_ml_confusion_matrix.asp
- https://keylabs.ai/blog/how-to-use-a-confusion-matrix-for-model-evaluation/
- https://medium.com/@saurabhdhandeblog/confusion-matrix-explained-calculating-accuracy-tpr-fpr-tnr-precision-and-prevalence-87557fe8714d
- https://www.codespeedy.com/confusion-matrix-and-performance-measures-in-ml/
- https://www.toolify.ai/ai-news/understanding-confusion-matrix-a-key-tool-for-ml-model-evaluation-2416937
- https://www.analytixlabs.co.in/blog/classification-in-machine-learning/
- https://www.geeksforgeeks.org/confusion-matrix-machine-learning/
- https://encord.com/glossary/confusion-matrix/

XII. Assessment Scheme: 50 Marks

Sr. No.	Performance Indicators	Weightage	Marks Obtained
A.	**Process Related: 30 Marks**	**60%**	
1.	Logic formation	40%	
2.	Debugging ability	10%	
3.	Correctness of program code	10%	
B.	**Product Related: 20 Marks**	**40%**	
3.	Expected output	10%	
4.	Timely completion of practical	10%	
5.	Answer to sample questions	20%	
C.	**Total marks obtained out of 50 Marks**		
D.	**Dated signature of Course Teacher**		