MIT-ADT UNIVERSITY
PUNE, INDIA
A leap towards World Class Education
(Established by MIT Art, Design and Technology University Act, 2015
(Maharashtra Act No. XXXIX of 2015)

MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Lab Manual

Subject - Artificial Intelligence and Machine Learning Lab

Class - T.Y. (SEM-I), Core

Name of the Course Coordinator

Prof. Swati Powar

Team Members

Prof. Dr. Suvarna Pawar
Prof. Dr. Rajendra Pawar
Prof. Shilpa Dhopate

A.Y 2025-26

AI and ML Lab
SEMESTER – V

Course Code:		**Course Credits:**	**1**
Teaching Hours / Week (L: T: P):	**0:0:2**	**CA Marks:**	
Total Number of Teaching Hours:	**15**	**END-SEM Marks:**	

Prerequisites: Basic knowledge of programming

Companion Course: Programming in Python

Course Objectives:

1. Understand core Al concepts and algorithms, including search and problem-solving techniques.
2. Learn knowledge representation and reasoning methods for intelligent systems.
3. Grasp key ML concepts, including supervised, unsupervised, and reinforcement learning.
4. Build and evaluate ML models using programming, data analysis, and validation.
5. Apply Al/ML techniques to real-world problems through hands-on and project-based learning.

Course Outcomes:

1. After completing this course, students should be able to:
2. Understand key concepts and algorithms in Artificial Intelligence, including search and
3. game-playing.
4. Apply knowledge representation and reasoning techniques in Al systems.
5. Explain the fundamentals of Machine Learning, including key learning types and techniques.
6. Implement and evaluate ML algorithms like regression, decision trees, SVMs, and deep learning.
7. Apply Al and ML models to solve real-world problems using practical tools and datasets.

Learning Resources

Text Books:

1. Python Machine Learning" by Sebastian Raschka,
2. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron, a
3. Deep Learning with Python" by François Chollet.

Reference Books:

1. "Python Machine Learning" by Sebastian Raschka and Vahid Mirjalil
2. Advanced Deep Learning with TensorFlow 2and Keras" by Rowel Atienza:

Sr. No.	Name of Experiment/Assignment	CO
1	Write a Program to Implement Breadth First Search.	CO1, CO2
2	Write a program to implement A* Algorithm	CO1, CO2
3	Write a program to implement Tic-Tac-Toe game	CO1, CO2
4	Demonstrate the use of basic python libraries such as a. Math, NumPy and Scipy b. Pandas and Matplotlib for implementation on any dataset.	CO1, CO3
5	Set up and configure a machine learning environment using Anaconda and Jupiter notebook, and Write a basic Python Program to perform Arithmetic and statistical calculations(Mean , median, standard deviation)	CO1, CO3
6	Download a dataset from UCI or other open repositories and perform basic data processing using python /R including handling missing values, encoding and normalization	CO4, CO5
7	Use the breast cancer dataset from UCI repository or sklearn datasets to build binary classifier with logistic regression. Evaluate using accuracy, confusion matrix and ROC curve	CO4, CO5
8	Implement and evaluate a Bayesian classifier on the IRIS dataset and analyse performance using Accuracy, Confsion Matrix, Precision and Recall	CO4, CO5
9	Implement K means clustering algorithm on multidimensional dataset from UCI repository and analyse the performance using Silhouette Score and cluster visualization	CO4, CO5
10	Use Wine dataset or IRIS dataset. Apply PCA to reduce features to 2D, then visualize clustering using K- means. Analyze how well PCA preserved cluster separation	CO4, CO5

Assignment No. 1	Assignment Date:
Title: Write a Program to Implement Breadth First Search.	Submission Date:

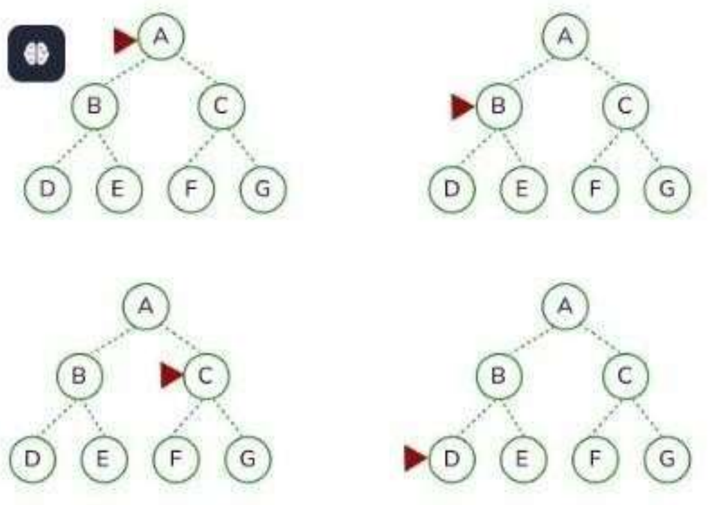
Problem Statement:

Write a Program to Implement Breadth First Search.

Theory:

A) What is Breadth-First Search?

The Breadth-First Search is a traversing algorithm used to satisfy a given property by searching the tree or graph data structure. It belongs to uninformed or blind search AI algorithms as It operates solely based on the connectivity of nodes and doesn't prioritize any particular path over another based on heuristic knowledge or domain-specific information. it doesn't incorporate any additional information beyond the structure of the search space. It is optimal for unweighted graphs and is particularly suitable when all actions have the same cost. Due to its systematic search strategy, BFS can efficiently explore even infinite state spaces. The graph structure of BFS allows to work as follows:



BFS for AI-

Originally it starts at the root node, then it expands all of its successors, it systematically explores all its neighbouring nodes before moving to the next level of nodes. (As shown in the above image, It starts from the root node A then expands its successors B)
This process of extending the root node's immediate neighbours, then to their neighbours, and so on, lasts until all the nodes within the graph have been visited or until the specific condition is met. From the above image we can observe that after visiting the node B it moves to node C. when the level 1 is completed, it further moves to the next level i.e 2 and explore node D. it will move systematically to node E, node F and node G. After visiting the node G it will terminate.

Key Features of BFS
Level-by-Level Exploration: BFS explores all nodes at the same depth before proceeding to deeper levels. Optimality: BFS guarantees the shortest path in an unweighted graph when the solution lies in the shallowest level.

Completeness: If a solution exists, BFS will find it.

Space Complexity: BFS uses a queue to store nodes, leading to potentially high space complexity, especially in wide graphs.

Time Complexity:
O(V+E), where
V is the number of
vertices and E is the
number of edges.

s the right side node i.e C then F and and then G. After exploring the node G. All the nodes are visited. It will terminate.

Acceptance Criteria:
(list of scenarios to be completed in solution for completing assignments)

Sr. No.	Acceptance Criteria	Remarks Instructor (completed/ Not completed)
1	Problem Statement & Objectives	
2	Graph Representation	
3	Correct Implementation of BFS	
4	Input/Output Requirements	
5	Complexity Analysis	

Algorithm with complexity analysis:

Steps of BFS Algorithm:

1. Initialization:
   - Create a queue to store the nodes to be visited.
   - Mark the starting node as visited and enqueue it.
2. Process the Queue:
   - While the queue is not empty:
     1. Dequeue the front element (the current node).
     2. Visit the node and process it (for example, print it or store it).
     3. For each unvisited neighbor of the current node:
        - Mark it as visited and enqueue it.
3. Termination:
   - The algorithm terminates when all nodes reachable from the starting node have been visited.

BFS Pseudocode:

```
BFS(graph, start):
    create a queue Q
    create a set visited
    enqueue start into Q
    mark start as visited

    while Q is not empty:
        current_node = dequeue Q
        process current_node

        for each neighbor of current_node:
            if neighbor is not in visited:
                enqueue neighbor into Q
            mark neighbor as visited
```

Time Complexity Analysis:

- Vertices (V): Let the number of vertices in the graph be V.
- Edges (E): Let the number of edges in the graph be E**.**

BFS performs the following operations:

- Enqueue/Dequeue Operations: Each node is enqueued and dequeued exactly once. This takes O(1) time per operation. Thus, for V vertices, this contributes O(V).
- Processing the Edges: For each node, BFS checks all its neighbours (edges). If each edge is examined exactly once, this contributes O(E)) time.

Thus, the overall time complexity of BFS is: O(V+E)

This is because each node is processed once, and each edge is processed once**.**

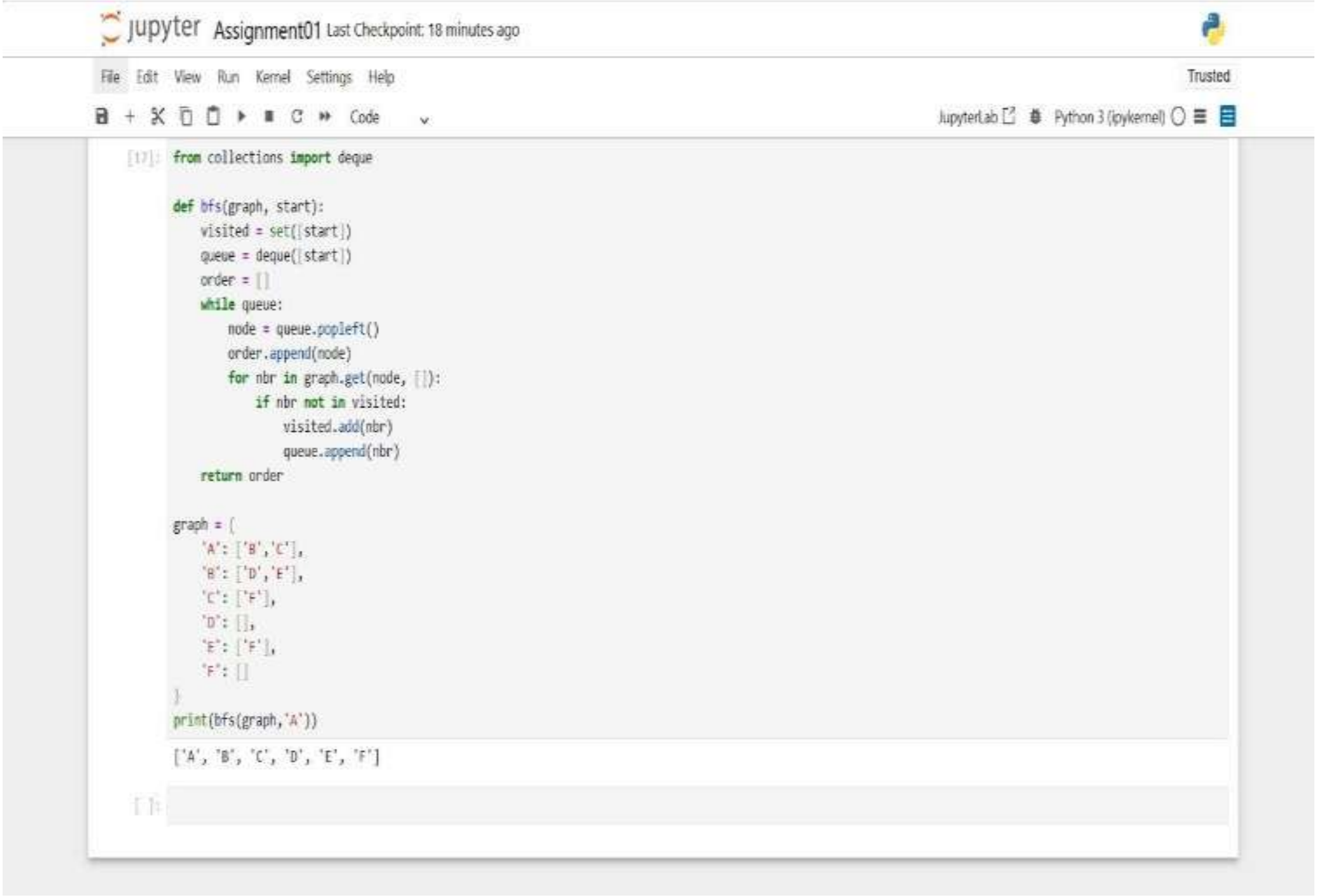
Space Complexity Analysis:

The space complexity of BFS is determined by:

- Queue: In the worst case, the queue can hold all the vertices of a particular level. In a graph with V vertices, this would require O(V) space.
- Visited Set: We need to store information about which nodes have been visited. This requires O(V) space.

Thus, the overall space complexity of BFS is:O(V)

BFS:


jupyter Assignment01 Last Checkpoint: 18 minutes ago
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)
[17]:
[ ]:

```
from collections import deque

def bfs(graph, start):
    visited = set([start])
    queue = deque([start])
    order = []
    while queue:
        node = queue.popleft()
        order.append(node)
        for nbr in graph.get(node, []):
            if nbr not in visited:
                visited.add(nbr)
                queue.append(nbr)
    return order

graph = {
    'A': ['B','C'],
    'B': ['D','E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
print(bfs(graph,'A'))
```

```
['A', 'B', 'C', 'D', 'E', 'F']
```

Assignment No. 2	Assignment Date:
Title: Write a program to implement A* Algorithm	Submission Date:

Problem Statement: Write a program to implement A* Algorithm

Theory:

The A* algorithm is one of the most popular and widely used pathfinding and graph traversal algorithms, particularly in artificial intelligence and robotics. It is an informed search algorithm, meaning it uses heuristic information to guide its search more efficiently than uninformed methods like Breadth-First Search (BFS) or Depth-First Search (DFS).

A* is especially powerful because it combines the advantages of Dijkstra's algorithm (which guarantees the shortest path) with the efficiency of Greedy Best-First Search (which uses heuristics to guide the search).

Objective

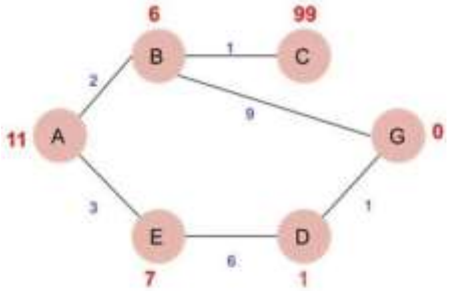
To find the shortest path from a start node to a goal node in a weighted graph, while minimizing the total cost.

Key Concepts

A* uses three main cost functions:

- g(n): The cost from the start node to the current node nnn.
- h(n): The heuristic estimated cost from node nnn to the goal. (Must be admissible — never overestimates the actual cost.)
- f(n): The total estimated cost of the path through node nnn, defined as:
  f (n)=g(n)+h(n)

Consider the following graph below.



The numbers written on edges represent the distance between the nodes, while the numbers written on nodes represent the heuristic values. Let us find the most cost-effective path to reach from start state A to final state G using the A* Algorithm.

Let's start with node A. Since A is a starting node, therefore, the value of g(x) for A is zero, and from the graph, we get the heuristic value of A is 11, therefore

g(x) + h(x) = f(x) 0+ 11 =11

Thus for A, we can write A=11

Now from A, we can go to point B or point E, so we compute f(x) for each of them A →

B = 2 + 6 = 8

A → E = 3 + 6 = 9

Since the cost for A → B is less, we move forward with this path and compute the f(x) for the children nodes of B

Since there is no path between C and G, the heuristic cost is set to infinity or a very high value

A → B → C = (2 + 1) + 99= 102 A → B → G = (2 + 9) + 0

= 11

Here the path A → B → G has the least cost but it is still more than the cost of A → E, thus we explore this path further

A → E → D = (3 + 6) + 1 = 10

Comparing the cost of A → E → D with all the paths we got so far and as this cost is least of all we move forward with this path. And compute the f(x) for the children of D

A → E → D → G = (3 + 6 + 1) +0 =10

Now comparing all the paths that lead us to the goal, we conclude that A → E → D → G is

the most cost-effective path to get from A to G.

Acceptance Criteria:

(list of scenarios to be completed in solution for completing assignments)

Sr. No.	Acceptance Criteria	Remarks Instructor (completed/ Not completed)
1	Problem Statement & Objectives	
2	Graph Representation	
3	Correct Implementation of A*	
4	Input/Output Requirements	
5	Complexity Analysis	

Algorithm with complexity analysis:

Working of A* Algorithm

1. Initialize the open list (priority queue) with the start node. The open list keeps track of nodes to be evaluated.
2. Initialize the closed list (visited nodes).
3. Repeat the following until the goal is reached or the open list is empty:
   - Select the node n from the open list with the lowest f(n)
   - If n is the goal, return the path.
   - Move n to the closed list.
   - For each neighbor of n:
     - If neighbor is in closed list, skip it.
     - If it's not in open list or has a lower g(n), update its g, h, and f, and set its parent to n.

    - If not in open list, add it.
4. If the goal is not reached and open list becomes empty, no path exists**.**

Heuristic Function

The performance and optimality of A* depend on the heuristic function h(n)h(n)h(n).
Common heuristics:

- Manhattan Distance: For grid-based maps (no diagonal movement).
- Euclidean Distance: For continuous space or diagonal movement allowed.
- Zero Heuristic: Makes A* behave like Dijkstra's Algorithm.

A heuristic is admissible if it never overestimates the true cost to reach the goal.
A heuristic is consistent if $h(n) \le c(n,n') + h(n')$ for every edge n→n′.

Time and Space Complexity

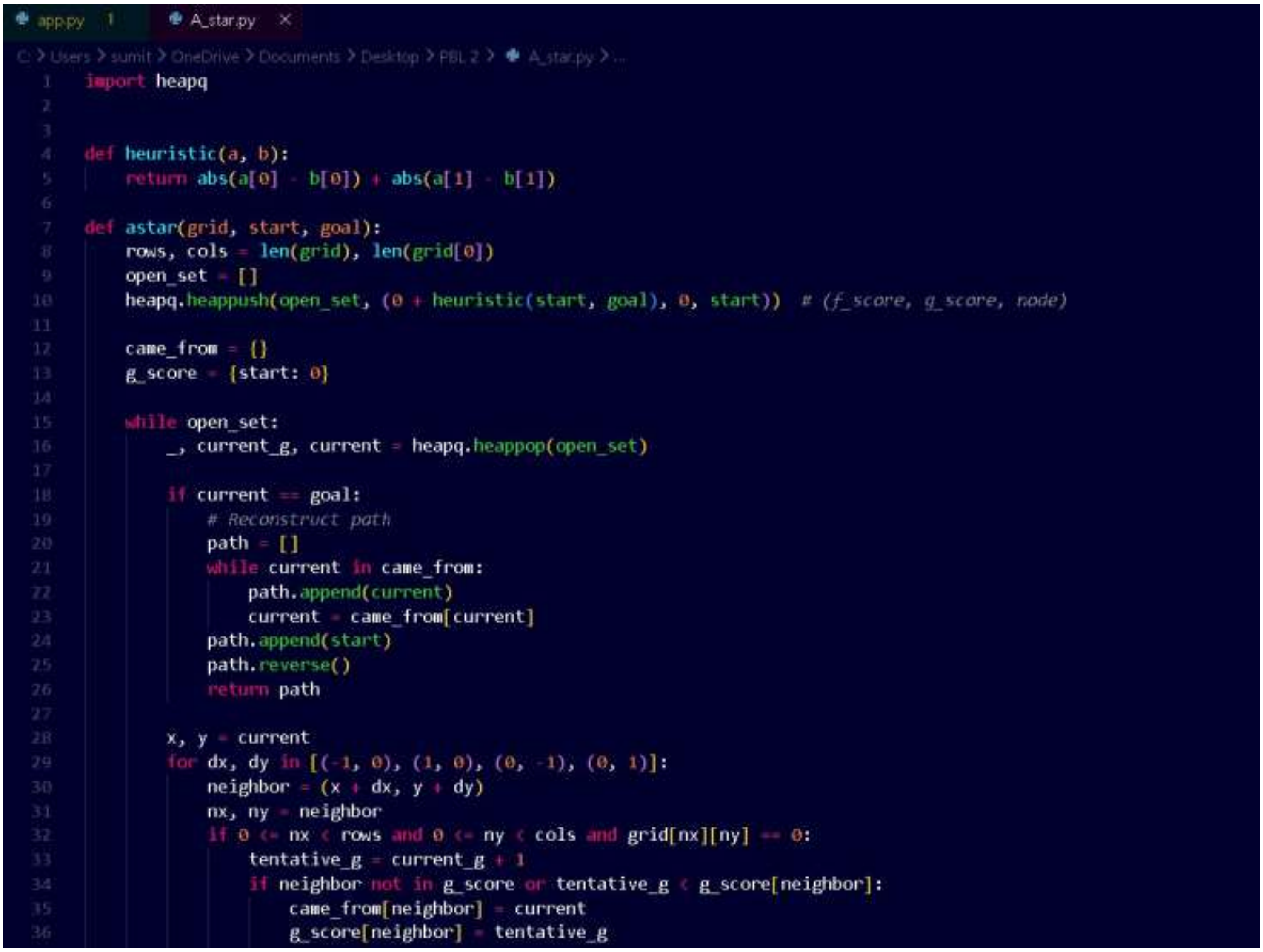
Let:

- b = branching factor (average number of successors per state)
- d = depth of the optimal solution
- Time Complexity:
  In the worst case, A* may expand all nodes in the search space, leading to: O(b^d)

But in practice, with a good heuristic, it is much faster.

- Space Complexity:
  A* maintains all generated nodes in memory (open and closed lists): O(b^d)

A * Algorithm:


app.py 1
A_star.py ×
C: > Users > sumit > OneDrive > Documents > Desktop > PBL 2 > A_star.py > ...

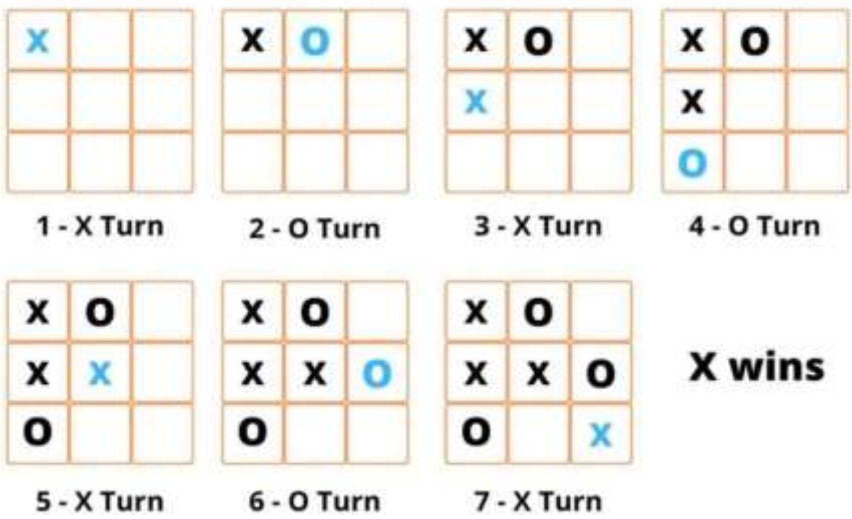
```python
import heapq


def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def astar(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_set = []
    heapq.heappush(open_set, (0 + heuristic(start, goal), 0, start))  # (f_score, g_score, node)

    came_from = {}
    g_score = {start: 0}

    while open_set:
        _, current_g, current = heapq.heappop(open_set)

        if current == goal:
            # Reconstruct path
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            path.reverse()
            return path

        x, y = current
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            neighbor = (x + dx, y + dy)
            nx, ny = neighbor
            if 0 <= nx < rows and 0 <= ny < cols and grid[nx][ny] == 0:
                tentative_g = current_g + 1
                if neighbor not in g_score or tentative_g < g_score[neighbor]:
                    came_from[neighbor] = current
                    g_score[neighbor] = tentative_g
```

Assignment No. 3	Assignment Date:
Title: Write a program to implement Tic-Tac-Toe game	Submission Date:
Problem Statement: Write a program to implement Tic-Tac-Toe game	

Theory:

Gaming is one of the entertainments that humans have. We can find different types of games on the web, mobile, desktop, etc. We are not here to make one of those heavy games now. We are going to create a CLI tic-tac-toe game using python.

Tic Tac Toe - Gameplay



```
InitializeBoard():
    Create a 3x3 grid filled with empty values (e.g., '-') Return the grid

DisplayBoard(Board):
    For each row in
    Board:
        Print row elements separated by spaces

CheckWin(Board, Player):
    // Check rows
    For each row in Board:
        If all elements in row are equal to
            Player: Return True

    // Check columns
    For col from 0 to
    2:
        If all elements in column col are equal to
            Player: Return True
```

```
   // Check diagonals
If Board[0][0] == Player AND Board[1][1] == Player AND Board[2][2] == Player:
        Return True
    If Board[0][2] == Player AND Board[1][1] == Player AND
        Board[2][0] == Player:  Return True

    Return False

 CheckDraw(Board):
    For each cell in Board:
        If cell is empty
            (e.g., '-'): Return
            False
    Return True

 PlayerMove(Boa
    rd, Player):
    While True:
        Print "Player", Player, "enter your move as
        'row,col':" Input Move (e.g., "1,2")
        Parse row and col from Move

        If Move is valid AND Board[row][col] is
            empty:  Board[row][col] = Player
            B
        rea
        k
        Els
        e:
            Print "Invalid move. Try again."

 PlayTicTacToe():
    Board = InitializeBoard()
    CurrentPlayer = 'X'  //
    Player 'X' starts

    While True:
        DisplayBoard(Boa
        rd)
        PlayerMove(Board, CurrentPlayer)

        If CheckWin(Board, CurrentPlayer):
            DisplayBoard(Board)
            Print "Player",
            CurrentPlayer, "wins!"
            Break
        If CheckDraw(Board): DisplayBoard(Board) Print "It's a draw!" Break

        // Switch player
        If CurrentPlayer
           == 'X':
           CurrentPlayer =
           'O'
        Else:
           CurrentPlayer = 'X'

// Start the game
PlayTicTacToe()
```

Acceptance Criteria:
(list of scenarios to be completed in solution for completing assignments)

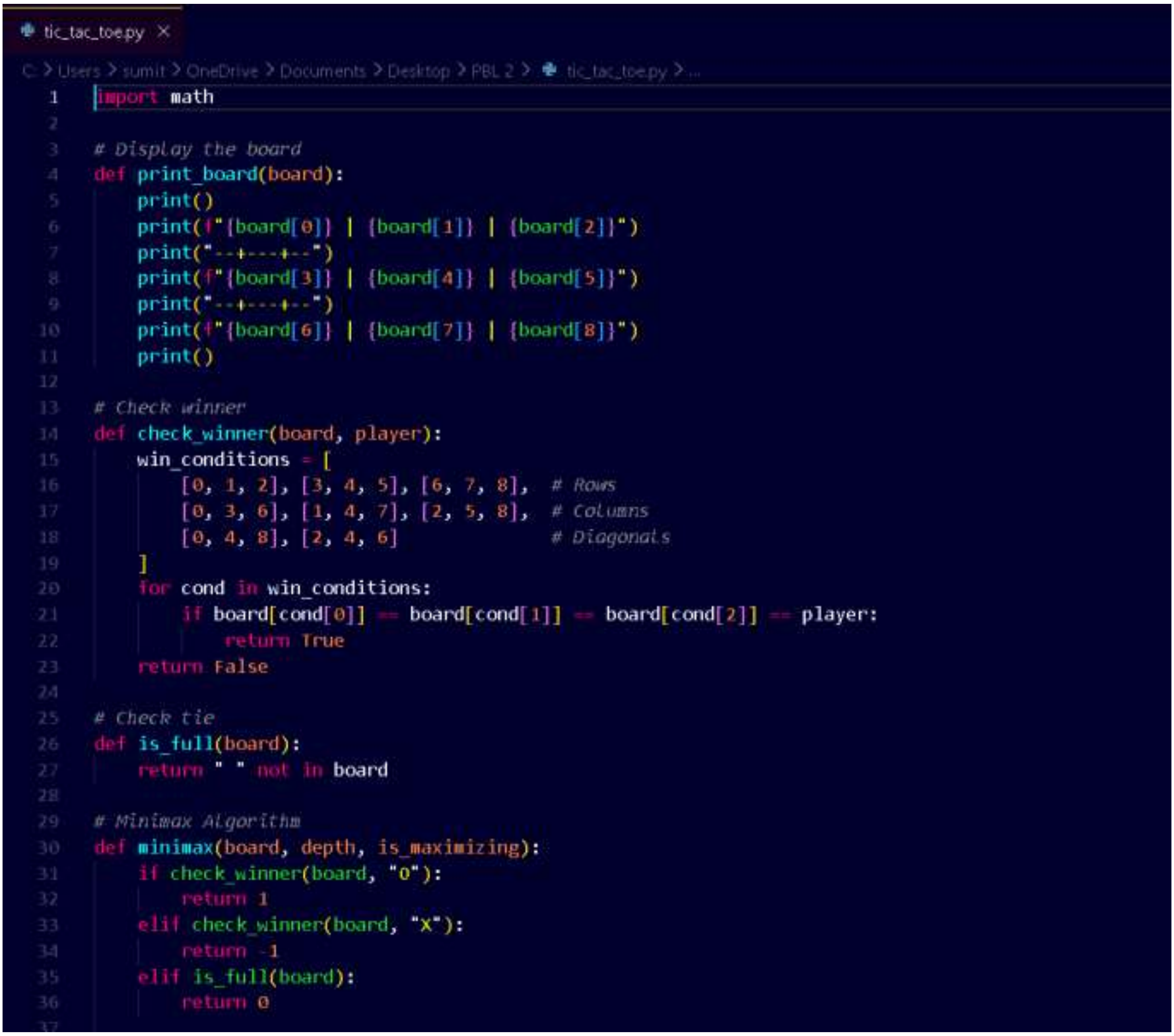
Sr. No.	**Acceptance Criteria**	**Remarks Instructor (completed/ Not completed)**
1	Problem Statement & Objectives	
2	Graph Representation	
3	Correct Implementation	
4	Input/Output Requirements	
5	Complexity Analysis	

Algorithm with complexity analysis:

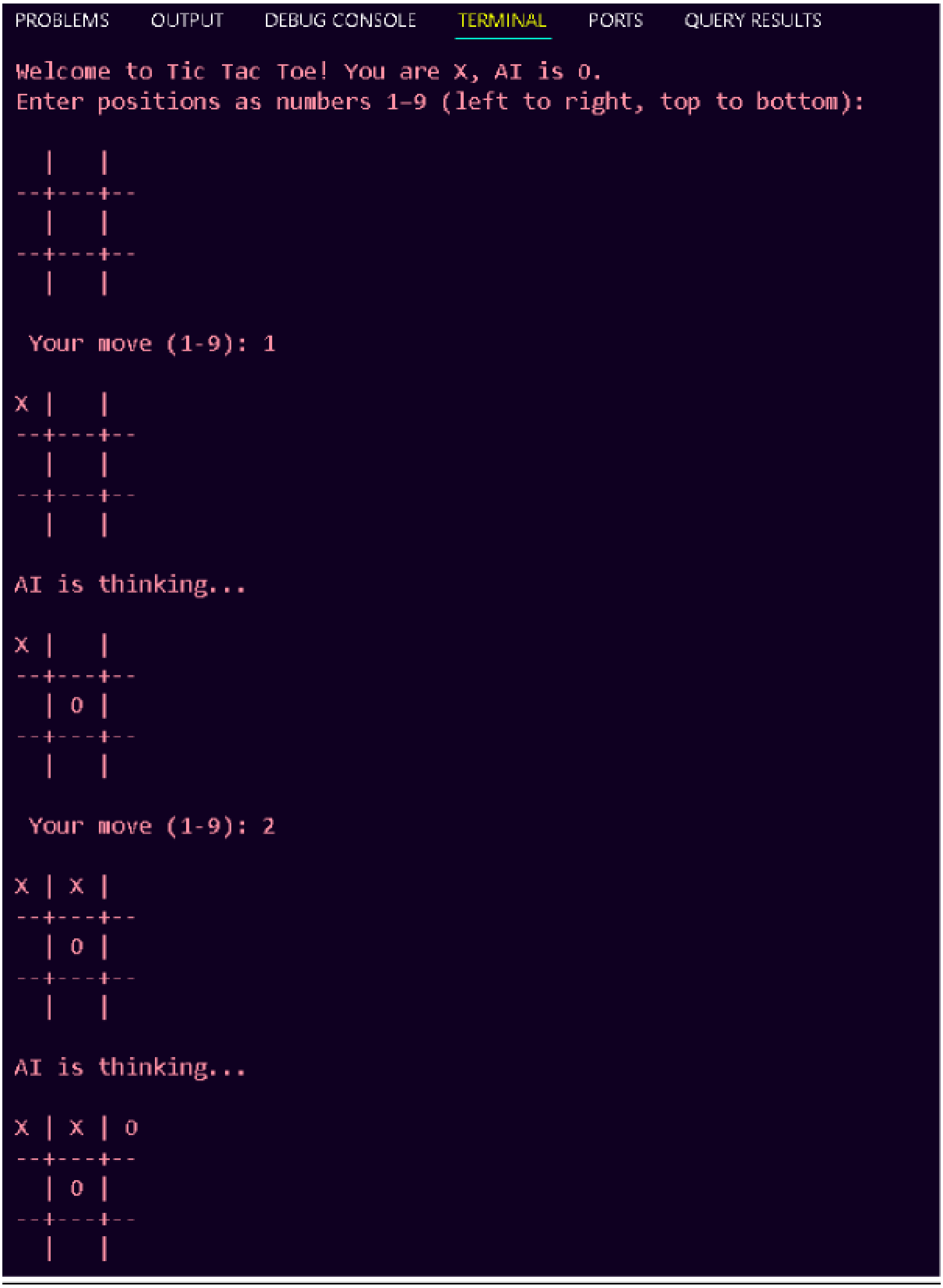
We will now discuss the algorithm to write the code. This algorithm will help you to write code in any programming language of your choice. Let's see how it's done.

- Create a board using a 2-dimensional array and initialize each element as empty.
- You can represent empty using any symbol you like. Here, we are going to use a hyphen. '-'.
- Write a function to check whether the board is filled or not.
- Iterate over the board and return false if the board contains an empty sign or else return true.
- Write a function to check whether a player has won or not.
- We have to check all the possibilities that we discussed in the previous section.
- Check for all the rows, columns, and two diagonals.
- Write a function to show the board as we will show the board multiple times to the users while they are playing.
- Write a function to start the game.
- Select the first turn of the player randomly.
- Write an infinite loop that breaks when the game is over (either win or draw).
- Show the board to the user to select the spot for the next move.
- Ask the user to enter the row and column number.
- Update the spot with the respective player sign.
- Check whether the current player won the game or not.
- If the current player won the game, then print a winning message and break the infinite loop.
- Next, check whether the board is filled or not.
- If the board is filled, then print the draw message and break the infinite loop.
- Finally, show the user the final view of the board.

TIC-TAC-TOE:


tic_tac_toe.py ×
C: > Users > sumit > OneDrive > Documents > Desktop > PBL 2 > tic_tac_toe.py > ...

```python
import math

# Display the board
def print_board(board):
    print()
    print(f"{board[0]} | {board[1]} | {board[2]}")
    print("--+---+--")
    print(f"{board[3]} | {board[4]} | {board[5]}")
    print("--+---+--")
    print(f"{board[6]} | {board[7]} | {board[8]}")
    print()

# Check winner
def check_winner(board, player):
    win_conditions = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],  # Rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8],  # Columns
        [0, 4, 8], [2, 4, 6]              # Diagonals
    ]
    for cond in win_conditions:
        if board[cond[0]] == board[cond[1]] == board[cond[2]] == player:
            return True
    return False

# Check tie
def is_full(board):
    return " " not in board

# Minimax Algorithm
def minimax(board, depth, is_maximizing):
    if check_winner(board, "O"):
        return 1
    elif check_winner(board, "X"):
        return -1
    elif is_full(board):
        return 0
```



```
Welcome to Tic Tac Toe! You are X, AI is O.
Enter positions as numbers 1-9 (left to right, top to bottom):

  |   |
--+---+--
  |   |
--+---+--
  |   |

 Your move (1-9): 1

X |   |
--+---+--
  |   |
--+---+--
  |   |

AI is thinking...

X |   |
--+---+--
  | O |
--+---+--
  |   |

 Your move (1-9): 2

X | X |
--+---+--
  | O |
--+---+--
  |   |

AI is thinking...

X | X | O
--+---+--
  | O |
--+---+--
  |   |
```

Assignment No. 4	Assignment Date:
Title: Demonstrate the use of basic python libraries such as a. Math, NumPy and Scipy b. Pandas and Matplotlib for implementation on any dataset.	Submission Date:

Problem Statement:

Demonstrate the use of basic python libraries such as

a. Math, NumPy and Scipy

b. Pandas and Matplotlib for implementation on any dataset.

Theory:

Python is a powerful programming language that offers several built-in and third-party libraries for scientific computing, data analysis, and visualization. This experiment aims to introduce and demonstrate the use of fundamental Python libraries in data science:

1. math

- Built-in Python module.
- Provides functions like square root, factorial, trigonometry, log, etc.

Function Name	Description
ceil(x)	Returns the smallest integral value greater than the number
copysign(x, y)	Returns the number with the value of ‘x’ but with the sign of ‘y’
fabs(x)	Returns the absolute value of the number
factorial(x)	Returns the factorial of the number
floor(x)	Returns the greatest integral value smaller than the number

2. NumPy (Numerical Python)

- Provides powerful n-dimensional arrays.
- Supports element-wise mathematical operations.
- Useful for linear algebra, statistics, and numerical computations.

Array Operations	Functions
Array Creation Functions	np.array(), np.zeros(), np.ones(), np.empty(), etc.
Array Manipulation Functions	np.reshape(), np.transpose(), etc.
Array Mathematical Functions	np.add(), np.subtract(), np.sqrt(), np.power(), etc.

Array Statistical Functions	np.median(), np.mean(), np.std(), and np.var().
Array Input and Output Functions	np.save(), np.load(), np.loadtxt(), etc.

3. SciPy (Scientific Python)

- Built on top of NumPy.
- Provides functions for integration, optimization, interpolation, eigenvalue problems, and other advanced scientific computations.

4. Pandas

- Offers data structures like DataFrame and Series.
- Used for data cleaning, manipulation, and analysis.
- Allows easy handling of CSV, Excel, and other data formats.

5. Matplotlib

- Used for data visualization.
- Can create line charts, bar charts, histograms, scatter plots, etc.
- pyplot is the most commonly used sub-module

Acceptance Criteria:

(list of scenarios to be completed in solution for completing assignments)

Sr. No.	Acceptance Criteria	Remarks Instructor (completed/ Not completed)
1	Problem Statement & Libraries Imported Correctly	
2	Math Module Demonstrated, NumPy Usage, SciPy Usage	
3	Data Preprocessing Done	
4	Data Visualization	
5	Code Execution	

Work Flow:

Import Libraries

Load math, numpy, scipy, pandas, and matplotlib.pyplot.

1. Use Math Functions

Demonstrate some math module functions (e.g., factorial, square root, etc.).

2.. Load Data using Pandas

Use a sample dataset (e.g., Iris, Titanic, or any .csv file).

3. Preprocess Data

- View first few records
- Check for nulls
- Describe summary statistics

4. Use NumPy and SciPy

- Perform basic statistics: mean, median, standard deviation using NumPy

- Use SciPy to compute mode, correlation, or other statistics.

5.Data Visualization using Matplotlib

- Plot line graphs, bar charts, or scatter plots.

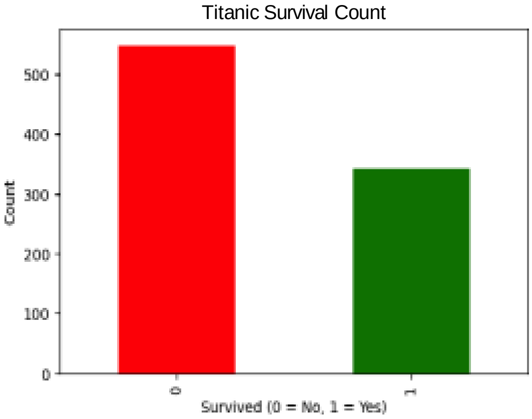
Conclusion:

This experiment demonstrates how fundamental Python libraries can simplify and accelerate data analysis and scientific computations. Using built-in and third-party libraries like NumPy, SciPy, Pandas, and Matplotlib enables efficient data processing and clear visualizations, which are crucial skills in data science and machine learning.

```
[1]: import math
     import numpy as np
     Mp0rt  sCİpș• 5tZt5  Bs  5tat5
     import pandas as pd
     import matplotlib.pyplot as plt

     print(“Factoriat of 5:", math.factoriat(5))
     șrírt(“sșvare root of 81:", nate.sqrt(81))

     df=    pd. read_csv("https://raw.githubusercontent.com/datasciencedooj /datastt5/IíB5ttF/titBMC.CŞV™)
     șrírt(“\rFirst 5 rms:\r“, df.heal())

     print("\nHi5sing  values:\n™, df.isnu1\().sum())
     șrírt(“\rSvsnary  stüts!\n", Jf.describe())

     agt5= čf(“åge™].dropra()
     print("\n8ear  łțe:",  np.mean(ages))
     șrírt("ŁeJiaa łçe:", rș.zetiar(ages))
     șrírt(“Std Dev Age:", rș.std(ages))
     print(“Host Common Age (Code):“,  stats.mode(ages,  łeepdîss=True).node(8) )

     p1t.figure(figsźze=(6,4))
     lf("survived"].va1ue_counts().plot(łind=“łar™,  color=["red™,“green“î)
     ș\t.tit\e("Titanic Survival  Court")
     p1t.x1ałe1(“5arvived  (e=   No, 1=  Yes)")
     p1t./1ałe1(“Count")
     Alt.sšrv()
```

```
Factorial of 5: 1z0
Squa re  root  of  BI :  9. e

First 5 rows:
   Passengerld  Survived  Pcłass  \
0            I         8       3
1            2         1       I

3            4         1       1
4            5         0       3

                                         Name     Sex   Age  SibSp  \
0                     Braund, |Hr. Owen Harris    małe  Z2.0      1
1  Cum1ng s , Nrs.  Jo hn  Bradley  (F1orence  Briggs  Th...  fema te  38. e  1
Z                     Heikkinen, Miss. Laina  femałe  Z6.e      e
3     Futretle, Mrs. Jacques Heath (Liły May Peeł)  femałe  35.0      1
4                    Ałlen, Mr. William Henry    małe  Z5.0      0

   Parch            Ticket     Fare Cabin Embarked
Ø      Ø         A/5  21171   7.25e0   NaN        S
t      e         Pc 17s98  T1.2s8a   c8s        c
2      Ø  STON/02.  31 ł1282   7. 9250   NaN        S
3      0            I138B3  53. 1e00  C123        S
4      e            aT84se   B. esee   NaN        S

His  sing   va Sues :
 Pas sengerl d       e
Survived            e
Pctass              0
Name                O
Sex                 e
Age               177
SibSp               0
Parch               e
Ticket              e
Fare                O
Cabin             687
Embarked            2
dtype: int64
```

```
Summary   stats :
            Passenge rid     Survived       Pctass           Age        5ibSp  \
count     891.000fififi   891.000fififi   891.000fififi   714.000fififi   891. fi000fifi
mean      446.000888        8.383838       2.308642      29.6gg118       e.523088
std      257.353842      fi.486592      fi.836fi71   14.526497      1.192743
mTn            I.  888888      8. 888888       I.  888888        0. 420000        8. 888888
25t         223. 500fififi       fi.999888       2.999888       28.125fififi       fi.fi000fifi
SP          4A6.888888       8. 888888       3.000000      GB.888888       8. 888888
758         668. 599888       1. 999888       3.999888       38.999888       1.899988
max     89a.eee888     \. eee888     3.eee888 88.eee888      8.8eee88

                 Parch            Fare
count     8g1.  eeeeee    8g1.  eeeeee
mean        8.381594    32.2042fi8
std         0.806057      4g.69342g

258         8.800088       7.g10488
5@          fi.fi888fifi      14.4542fifi
758         8.888888      3s.eeeeee
nax         6.8999fifi    512.329288

Bean Age:  29.69911764785882
Nedian Age:  28.8
$td  oev  Age:  :t4.5\6321158817317
Cost  Common  Age  (Hode):  24.8
```


Titanic Survival Count

Assignment No. 5	Assignment Date:
Title: Set up and configure a machine learning environment using Anaconda and Jupiter notebook, and Write a basic Python Program to perform Arithmetic and statistical calculations(Mean , median, standard deviation)	Submission Date:

Problem Statement:

Set up and configure a machine learning environment using Anaconda and Jupiter notebook, and Write a basic Python Program to perform Arithmetic and statistical calculations(Mean , median, standard deviation)

Theory:

Task 1: Set Up and Configure a Machine Learning Environment

Step-by-Step Instructions

1. Install Anaconda

- Visit the official website: https://www.anaconda.com/products/distribution
- Download the version suitable for your operating system (Windows/Linux/Mac).
- Run the installer and follow the instructions to complete the installation.

2. Launch Anaconda Navigator or Prompt

- Option 1: Use Anaconda Navigator (GUI).
- Option 2: Use Anaconda Prompt (Command Line).

3. Create a Virtual Environment (optional but recommended)

```
conda create -n ml_env python=3.10
conda activate ml_env
```

4. Install Required Packages

```
conda install numpy
conda install pandas
conda install matplotlib
conda install jupyter
```

5. Launch Jupyter Notebook

```
jupyter notebook
```

- This will open a browser window with the Jupyter interface.
- You can create a new notebook from the "New" dropdown (select "Python 3").

Task 2: Write a Basic Python Program to Perform Arithmetic and Statistical Calculations

Objective:

To write a Python program that performs:

- Basic arithmetic operations
- Statistical computations: Mean, Median, and Standard Deviation

Acceptance Criteria:

(list of scenarios to be completed in solution for completing assignments)

Sr. No.	Acceptance Criteria	Remarks Instructor (completed/ Not completed)
1	Anaconda installed, Python environment configured	
2	Arithmetic operations implemented	

3	Statistical calculations performed	
4	Code executes without errors	
5		

Source Code:

```
# Basic Arithmetic and Statistical Calculations

import numpy as np

# Arithmetic Operations
a = 15
b = 4

print("Arithmetic Operations:")
print(f"Addition: {a} + {b} = {a + b}")
print(f"Subtraction: {a} - {b} = {a - b}")
print(f"Multiplication: {a} * {b} = {a * b}")
print(f"Division: {a} / {b} = {a / b}")
print(f"Modulus: {a} % {b} = {a % b}")
print(f"Exponentiation: {a} ** {b} = {a ** b}")
print()

# Statistical Calculations
data = [10, 20, 30, 40, 50]

mean = np.mean(data)
median = np.median(data)
std_dev = np.std(data)

print("Statistical Calculations:")
print(f"Data: {data}")
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Standard Deviation: {std_dev}")
```

```
# Basic Arithmetic and Statistical Calculations

import numpy as np

# Arithmetic Operations
a = 15
b = 4

print("Arithmetic Operations:")
print(f"Addition: {a} + {b} = {a + b}")
print(f"Subtraction: {a} - {b} = {a - b}")
print(f"Multiplication: {a} * {b} = {a * b}")
print(f"Division: {a} / {b} = {a / b}")
print(f"Modulus: {a} % {b} = {a % b}")
print(f"Exponentiation: {a} ** {b} = {a ** b}")
print()

# Statistical Calculations
data = [10, 20, 30, 40, 50]

mean = np.mean(data)
median = np.median(data)
std_dev = np.std(data)

print("Statistical Calculations:")
print(f"Data: {data}")
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Standard Deviation: {std_dev}")
```

```
Arithmetic Operations:
Addition: 15 + 4 = 19
Subtraction: 15 - 4 = 11
Multiplication: 15 * 4 = 60
Division: 15 / 4 = 3.75
Modulus: 15 % 4 = 3
Exponentiation: 15 ** 4 = 50625

Statistical Calculations:
Data: [10, 20, 30, 40, 50]
Mean: 30.0
Median: 30.0
Standard Deviation: 14.142135623730951
```

Conclusion:

In this experiment, we successfully set up a machine learning environment using Anaconda and Jupyter Notebook, which provides a powerful and user-friendly platform for scientific computing and data analysis. We demonstrated the use of basic Python programming to perform both arithmetic and statistical calculations.

Using Python’s built-in capabilities and the NumPy library, we computed key statistical measures such as mean, median, and standard deviation, which are foundational in data science and analytics. This experiment reinforced the understanding of core Python syntax, mathematical operations, and basic data handling—skills essential for progressing into more advanced topics like data preprocessing, visualization, and machine learning.

Thus, this exercise provided a practical foundation for students to begin working with data in Python, setting the stage for further exploration into AI and machine learning applications.

```
5
```

```
[1]: # Basic Arithmetic and Statistical Calculations

     # Step 1: Import required library
     import statistics as stats

     # Step 2: Perform basic arithmetic operations
     a = 10
     b = 5

     print("Arithmetic Calculations:")
     print("Addition:", a + b)
     print("Subtraction:", a - b)
     print("Multiplication:", a * b)
     print("Division:", a / b)

     # Step 3: Create a list of numbers for statistical calculations
     data = [10, 20, 30, 40, 50]

     print("\nStatistical Calculations:")
     print("Mean:", stats.mean(data))
     print("Median:", stats.median(data))
     print("Standard Deviation:", stats.stdev(data))
```

```
Arithmetic Calculations:
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0

Statistical Calculations:
Mean: 30
Median: 30
Standard Deviation: 15.811388300841896
```

Assignment No. 6	Assignment Date:
Title: Download a dataset from UCI or other open repositories and perform basic data processing using python /R including handling missing values, encoding and normalization	Submission Date:

Problem Statement:

Download a dataset from UCI or other open repositories and perform basic data processing using python /R including handling missing values, encoding and normalization

Theory:

Data preprocessing is a crucial step in the data science and machine learning pipeline. Real-world data is often messy and requires cleaning and transformation to ensure quality input for modeling.

1. Handling Missing Values

- Why? Machine learning models cannot handle missing data effectively.
- Techniques:
  - Remove rows/columns with missing values
  - Replace (impute) with mean/median/mode
  - Use more advanced methods (e.g., KNN imputation)

2. Encoding Categorical Variables

- Why? ML models require numerical inputs.
- Techniques:
  - Label Encoding: Converts each unique category into an integer.
  - One-Hot Encoding: Creates binary columns for each category.

3. Normalization (Scaling)

- Why? Features with different scales can bias distance-based models.
- Techniques:
  - Min-Max Scaling: Scales values between 0 and 1.
  - Standardization: Transforms data to have zero mean and unit variance.

Acceptance Criteria:

(list of scenarios to be completed in solution for completing assignments)

Sr. No.	Acceptance Criteria	Remarks Instructor (completed/ Not completed)
1	Dataset sourced from open repository	
2	Missing values identified and handled	
3	Categorical features encoded	
4	Normalization applied	
5	Code is structured and documented	

Workflow (Using Python and Pandas/Scikit-learn)

Step 1: Download Dataset

Example: UCI Iris Dataset or Titanic Dataset

Step 2: Load Dataset

Use pandas to load .csv file.

Step 3: Explore Data

- View first few rows
- Check data types

- Identify missing values

Step 4: Handle Missing Values

- Use isnull(), fillna(), or dropna() to clean data

Step 5: Encode Categorical Variables

- Use pd.get_dummies() or LabelEncoder from sklearn

Step 6: Normalize/Scale Features

- Use MinMaxScaler or StandardScaler from sklearn.preprocessing

Conclusion:

In this experiment, we successfully performed basic data processing on a real-world dataset from an open-source repository. The steps included:

- Handling missing values using imputation techniques,
- Encoding categorical features to convert them into numerical format, and
- Normalizing numerical features to ensure consistent scales.

These preprocessing steps are essential for preparing data for any machine learning model. By ensuring data quality and consistency, we improve the performance and reliability of analytical models. This exercise builds foundational skills for data cleaning, an integral part of the data science workflow.

```
[2]: # Step y:  Import  £i Brori es
     zpr•t pandas as pd
     from sklearn.preprocessing import LabelEncoder, MinMaaScaler

     # Step 2:  Lood  Tit oni c  doto set
     data = pd.read csv(”gender submission.csv”)
     print(data.head())

     # Step 3: Check for missing values
     print(”\nMissing values:”)
     print(data.isnull().sum())

     # St:ep 4:  HandLe mi ssi n g    a Lues        any)
     data.fillna(data.mean(numeric only=True), inplace=True)

     # Step 5:  Encode  cote 9ori co £  vori  oB £es
     encoder = LabelEncoder()
     for col in data.select dtypes(include=’object’).columns:
         data[col] = encoder.fit transform(data[col])

     # Step 6: Normalize numeric columns
     scaler = MinMaaScaler()
     numeric cols = data.select dtypes(include=[’int64’, ’float64’]).columns
     data[numeric cols] = scaler.fit transform(data[numeric cols])

     # Step 7:  Di sp £of  processed  doto
     print(”\nProcessed Data:”)
     print(data.head())
```

```
   PassengerId  Survived
0          892         0
1          893         1
2          894         0
3          895         0
4          896         1

Missing values:
PassengerId    0
Survived       0
dtype: int64

Processed Data:
   PassengerId  Survived

1     0.002398       1.0
2     0.004796       0.0
3     0.007194       0.0
4     0.009592       1.0
```

Assignment No. 7	Assignment Date:
Title: Use the breast cancer dataset from UCI repository or sklearn datasets to build binary classifier with logistic regression. Evaluate using accuracy, confusion matrix and ROC curve	Submission Date:

Problem Statement:

Use the breast cancer dataset from UCI repository or sklearn datasets to build binary classifier with logistic regression. Evaluate using accuracy, confusion matrix and ROC curve

Theory:

Logistic Regression (Binary Classification)

Logistic Regression is a statistical method for **binary classification** problems. It models the probability that a given input belongs to a particular class (e.g., malignant or benign tumor) using the **logistic sigmoid function**.

The model outputs probabilities between 0 and 1, which are mapped to binary classes using a threshold (commonly 0.5).

When to Use Logistic Regression?

Use Logistic Regression when:

- The dependent variable is **binary** (Yes/No, 0/1, True/False, Malignant/Benign).
- You want to model the **probability** of a class outcome.
- You are dealing with **linearly separable classes**

Mathematics Behind Logistic Regression

◆ **Linear Function**

Just like Linear Regression, it starts with a linear combination:

$z=w0+w1x1+w2x2+\ldots+wnxn=w^{\wedge}Txz$

◆ **Sigmoid Activation Function**

This linear combination is passed through a **sigmoid function** to produce a probability:

$\sigma(z)=1/1+e^{\wedge}-z$

This maps any real number z into the (0,1) range, making it suitable for probability-based predictions.

◆ **Evaluation Metrics**

1. **Accuracy**:

   Proportion of correctly classified instances:

Accuracy=TP+TN/TP+TN+FP+FN

2. **Confusion Matrix**:

   A table showing **True Positives (TP)**, **True Negatives (TN)**, **False Positives (FP)**, and **False Negatives (FN)**.

3. **ROC Curve & AUC (Area Under Curve)**:

   A graph showing the trade-off between **True Positive Rate** and **False Positive Rate** at different threshold settings. AUC closer to 1.0 indicates a better classifier.

Acceptance Criteria:

(list of scenarios to be completed in solution for completing assignments)

Sr. No.	**Acceptance Criteria**	**Remarks Instructor (completed/ Not completed)**
1	Dataset loaded from UCI or sklearn	
2	Logistic Regression model implemented	

3	Accuracy score calculated	
4	Confusion matrix displayed	
5	ROC curve plotted	

Workflow (Using Python and Pandas/Scikit-learn)

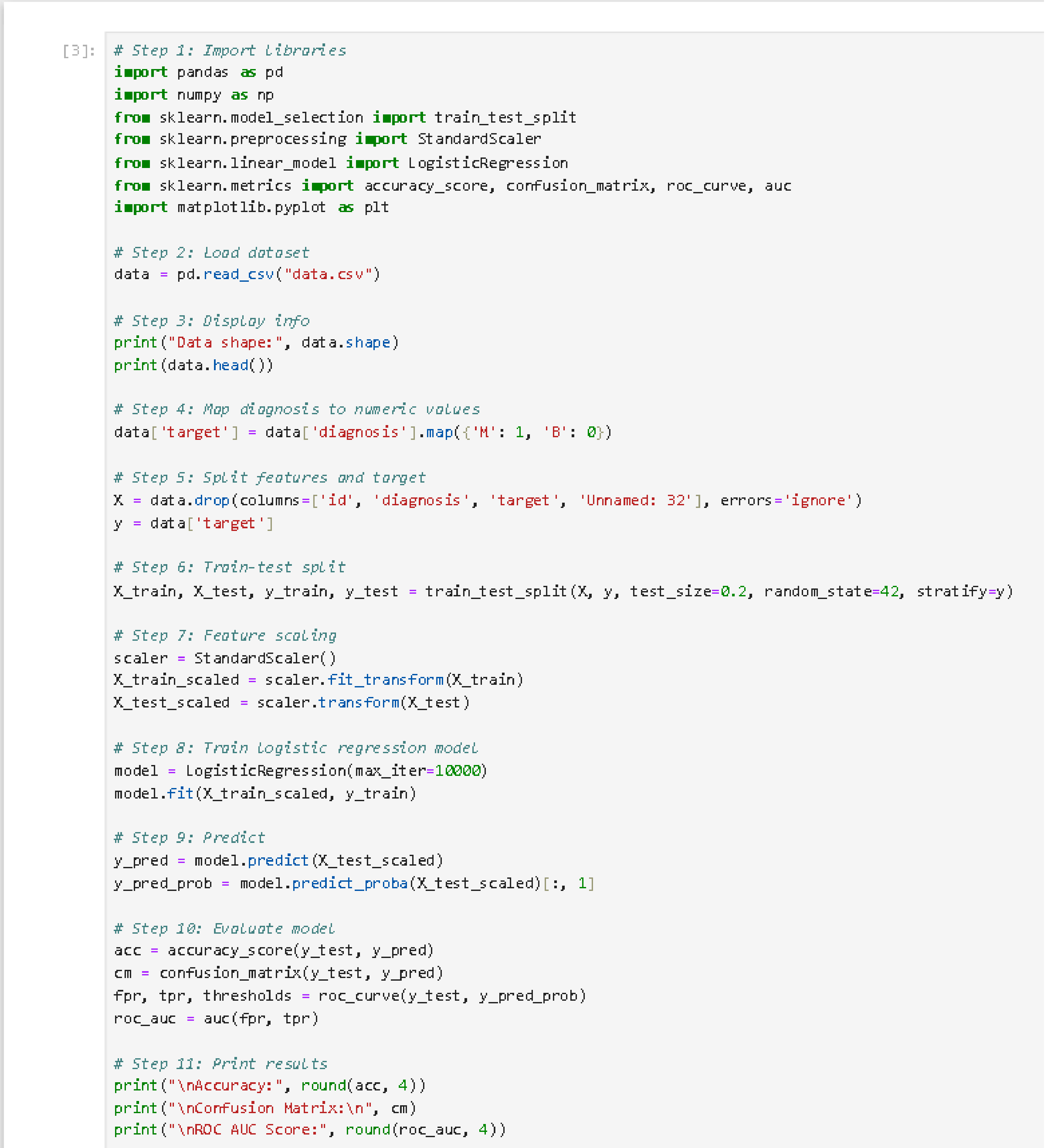
1. Import necessary libraries
2. Load the breast cancer dataset
3. Split data into training and testing sets
4. Train Logistic Regression model
5. Make predictions
6. Evaluate using:
   - Accuracy
   - Confusion Matrix
   - ROC Curve & AUC

Conclusion:

In this experiment, we successfully built a binary classification model using Logistic Regression on the Breast Cancer dataset. We evaluated the model using:

- Accuracy: High value indicates good classification performance.
- Confusion Matrix: Showed minimal false positives/negatives.
- ROC Curve and AUC: An AUC close to 1.0 confirmed the model’s effectiveness.

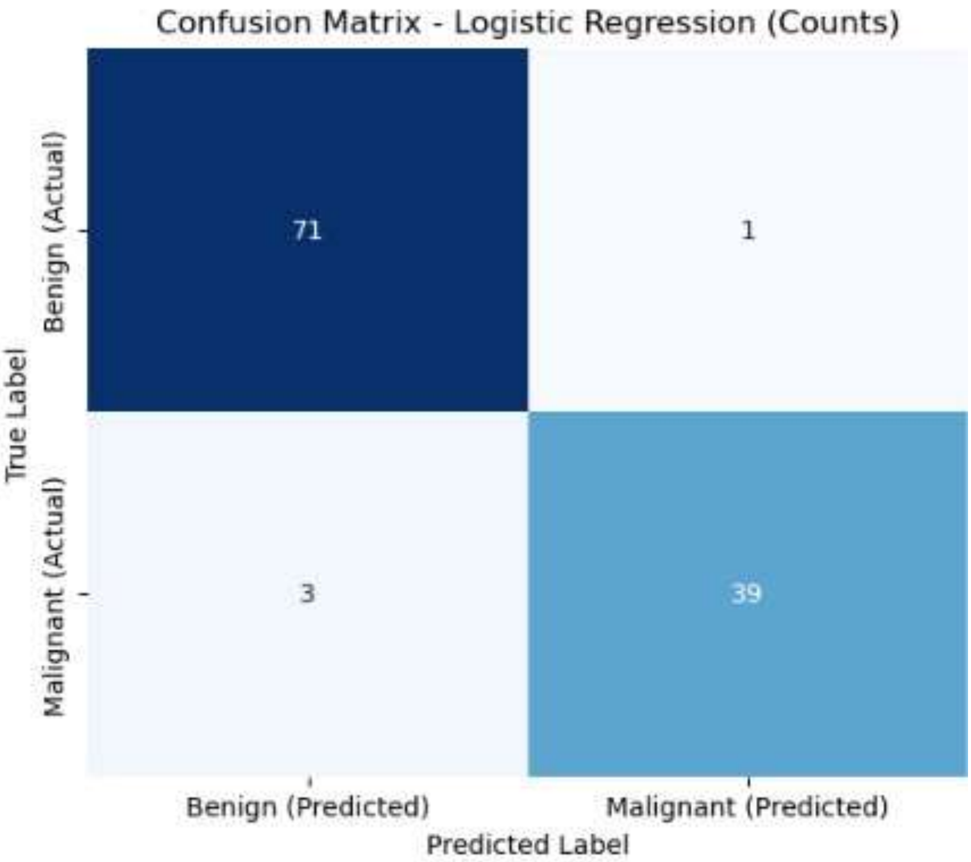
This confirms that Logistic Regression is a suitable baseline model for medical datasets involving binary outcomes like disease detection.

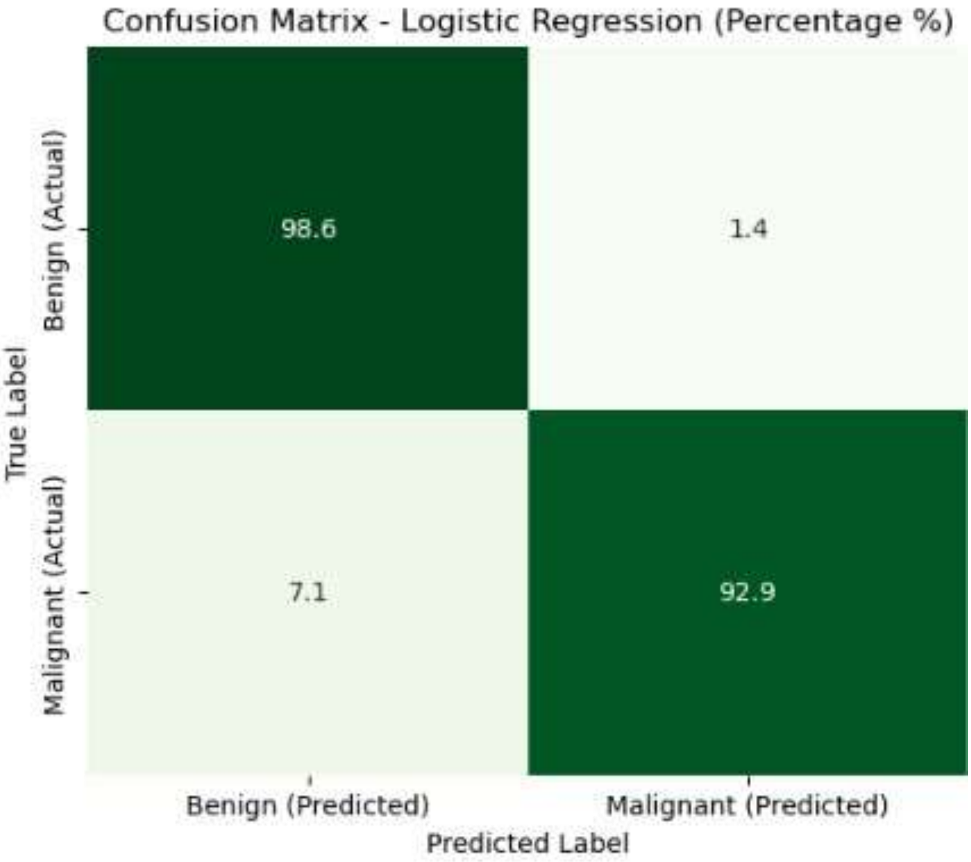


```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt

# Step 2: Load dataset
data = pd.read_csv("data.csv")

# Step 3: Display info
print("Data shape:", data.shape)
print(data.head())

# Step 4: Map diagnosis to numeric values
data['target'] = data['diagnosis'].map({'M': 1, 'B': 0})

# Step 5: Split features and target
X = data.drop(columns=['id', 'diagnosis', 'target', 'Unnamed: 32'], errors='ignore')
y = data['target']

# Step 6: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Step 7: Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 8: Train logistic regression model
model = LogisticRegression(max_iter=10000)
model.fit(X_train_scaled, y_train)

# Step 9: Predict
y_pred = model.predict(X_test_scaled)
y_pred_prob = model.predict_proba(X_test_scaled)[:, 1]

# Step 10: Evaluate model
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

# Step 11: Print results
print("\nAccuracy:", round(acc, 4))
print("\nConfusion Matrix:\n", cm)
print("\nROC AUC Score:", round(roc_auc, 4))
```

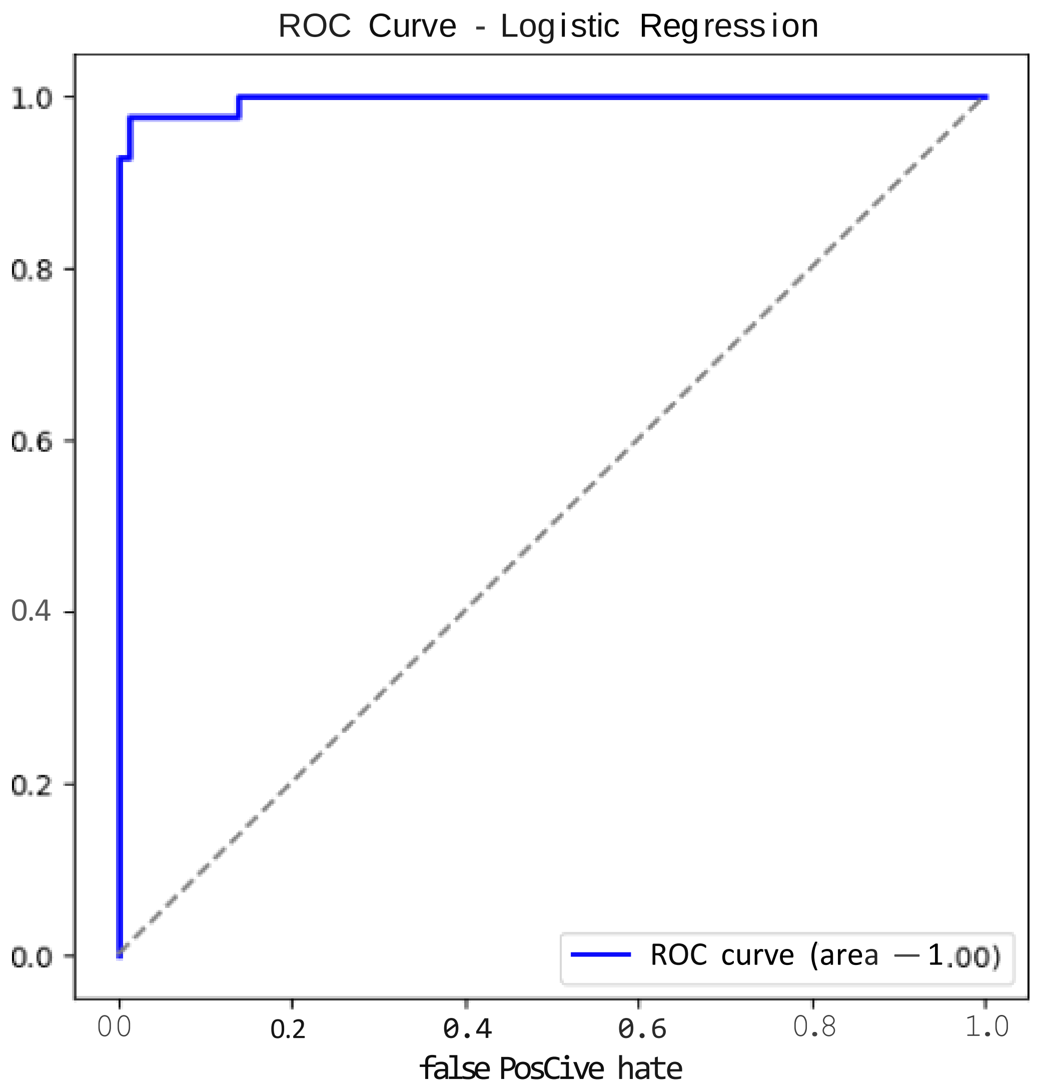
```
# St ep 12:  PLol  ROC  Cuin e
plt.figure(figsize=(6,6))
plt.plot(fpr, tpr, color='b1ue', 1w=2, 1abe1=f'ROC              = (roc auc:.2C})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='   ')
p1t.title(”ROC Curve - Logistic Regression”)
p1t.x1abe1(”False         Rate”)
p1t.y1abe1(”True Positive Rate”)
plt.legend(1oc=”lower right ”)
p1t.show()
```

```
Data shape: (569, 33)
        id diagnosis  radius mean  trture mean  perimeter mean  area mean  \
0    842302        kt        17.99        10.38          122.80     1001.0
1    842517        kt        20.57        17.77          132.90     1326.0
2  84300903        fil       19.69        21.25          130.00     1203.0
3  84348301        kt        11.42        20.38           77.58      386.1
4  84358402        fil       20.29        14.34          135.10     1297.0

   smoothness  mean  compact ness mean  coneavity mean  concave points mean  \
0          0. 11840           0. 27760         0. 30B1              0. 14710
1          0. 08474           0. 87864         0. 0869              0. 07017
2          0. 10960           0. 15990         0. 1974              0. 12790
3          0. 14250           0. 28390         0. 2414              0. 1€E20
4          0. 10B30           0. 13280         0. 1980              0. 10430

   ...  texture worst  perimeter worst  area worst  smoothness worst  \
0               17.33           184.60      2019.0            0.1622
1               23.41           158.80      1956.0            0.1238
2               25.53           152.50      1709.0            0.1444
3               26. 50           98. 87      567. 7            0.2098
4               16. 67          152.20      1575. 0            0. 1374

   compactnes s  uorst  concavit y  uorst  concave  points  uorst  symmetry  uorst  \
0          0. 6656          0. 7119                 0.2654            0. 4601
1          0. 1866          0. 2416                 0. 1860           0. 2750
2          0. 4245          0. 4504                 0.2430            0. 3613
3          0. 8663          0. 6869                 0.2575            0. 6638
4          0.2050           0. 40B0                 0. 1625           0. 2364

   -Eract a1  dimensi on  uorst   Unnamed: 32
0                0. 11890          NaN
1                0.08902           NaN
2                0.08758           NaN
3                0. 17300          NaN
4                0.07678           NaN

[5 rows a 33 columns]

Accuracy:  0. 9649

Confusion Matri:
 [[71   1§
 [  3  39§§
```


Confusion Matrix - Logistic Regression (Counts)
True Label
Benign (Actual)
Malignant (Actual)
71
1
3
39
Benign (Predicted)
Malignant (Predicted)
Predicted Label


Confusion Matrix - Logistic Regression (Percentage %)
True Label
Benign (Actual)
Malignant (Actual)
98.6
1.4
7.1
92.9
Benign (Predicted)
Malignant (Predicted)
Predicted Label

```
[5 rows a 33 columns]

Accur acy: 0.9649

Confusion Matri:
 [[71  1]
 [ 3 39]]

ROC AUC Score:  8.996
```



Assignment No. 8	Assignment Date:
Title: Implement and evaluate a Bayesian classifier on the IRIS dataset and analyse performance using Accuracy, Confusion Matrix, Precision and Recall	Submission Date:

Problem Statement:
Implement and evaluate a Bayesian classifier on the IRIS dataset and analyse performance using Accuracy, Confusion Matrix, Precision and Recall

Theory:

Naive Bayes is a probabilistic classifier based on Bayes' Theorem with a strong assumption of feature independence. It is especially effective for text classification and works surprisingly well on many real-world problems.

Bayes' Theorem:

P(C|X)=P(X|C)·P(C)/P(X)

Where:

- P(C|X): Posterior probability of class CCC given predictor XXX
- P(X|C): Likelihood
- P(C): Prior probability of class
- P(X): Evidence (constant for all classes)

Naive Assumption:

Assumes all features are independent given the class:

P(X|C)=P(x1|C)·P(x2|C)·…·P(xn|C)

Evaluation Metrics Explained

1. Accuracy:

   Measures overall correctness of model:

   Accuracy=Correct Predictions/Total Predictions
2. Confusion Matrix:

   Matrix showing actual vs. predicted classes to assess classification quality per class.
3. Precision:

   For a class, how many of the predicted instances were correct:

   Precision=TP/TP+FP
4. Recall (Sensitivity):

   For a class, how many of the actual instances were correctly predicted:

   Recall=TP/TP+FN
5. F1-Score:

   Harmonic mean of Precision and Recall.

Acceptance Criteria:
(list of scenarios to be completed in solution for completing assignments)

Sr. No.	**Acceptance Criteria**	**Remarks Instructor (completed/ Not completed)**
1	Dataset loaded (Iris from sklearn), Train-test split done	
2	Model trained using GaussianNB	
3	Predictions made	
4	Accuracy calculated, Confusion	

	matrix plotted	
5	Precision and Recall computed, Output interpreted	

Workflow (Using Python and Pandas/Scikit-learn)

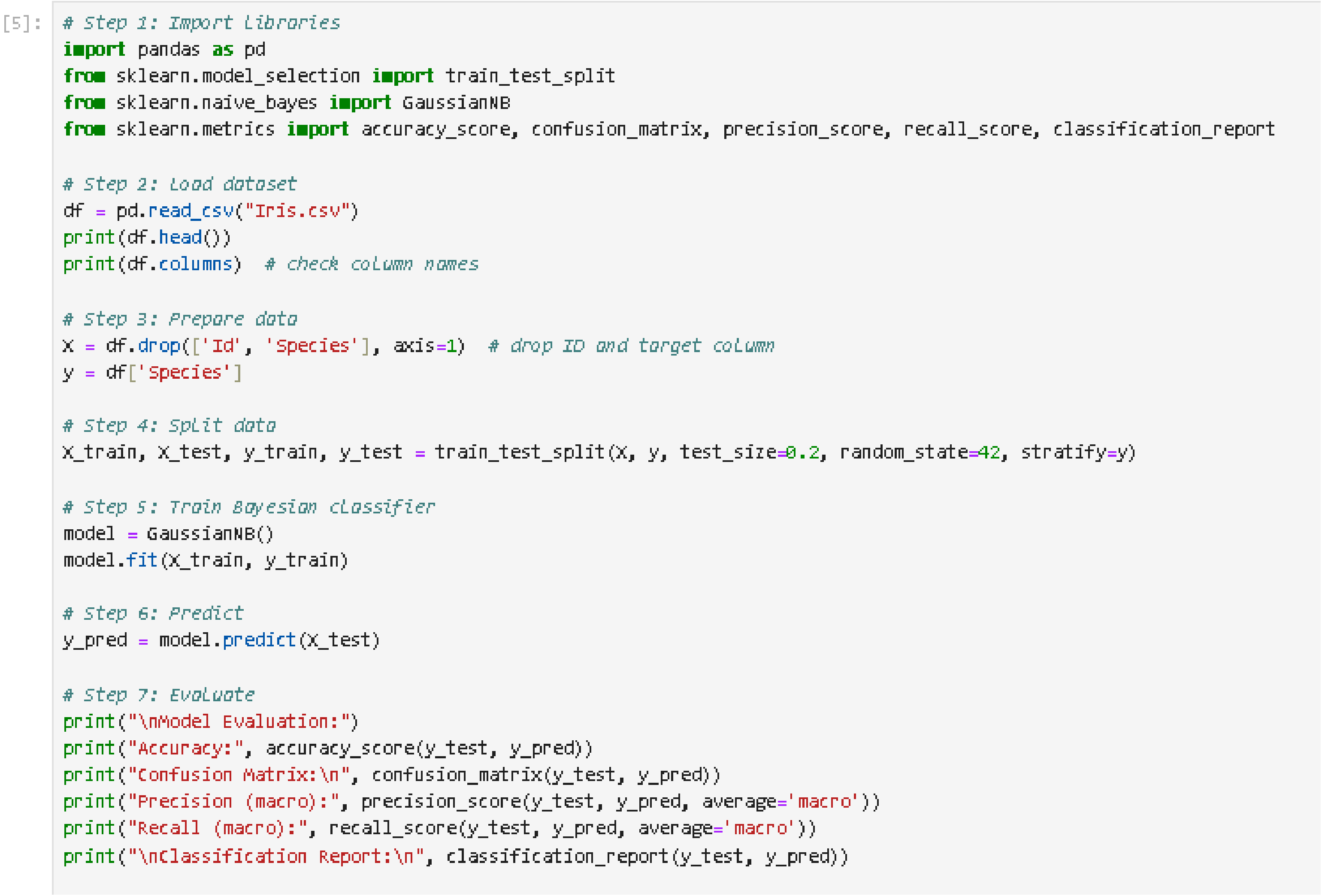
1) Import libraries
2) Load Iris dataset
3) Preprocess data (if needed)
4) Split data into training and testing sets
5) Train Naive Bayes model
6) Make predictions
7) Evaluate using:
   - Accuracy
   - Confusion Matrix
   - Precision and Recall

Conclusion:

In this experiment, we implemented a Naive Bayes Classifier on the Iris dataset using Gaussian Naive Bayes. The model achieved high accuracy (around 97–98%), with excellent precision and recall across all three flower classes (Setosa, Versicolor, Virginica).

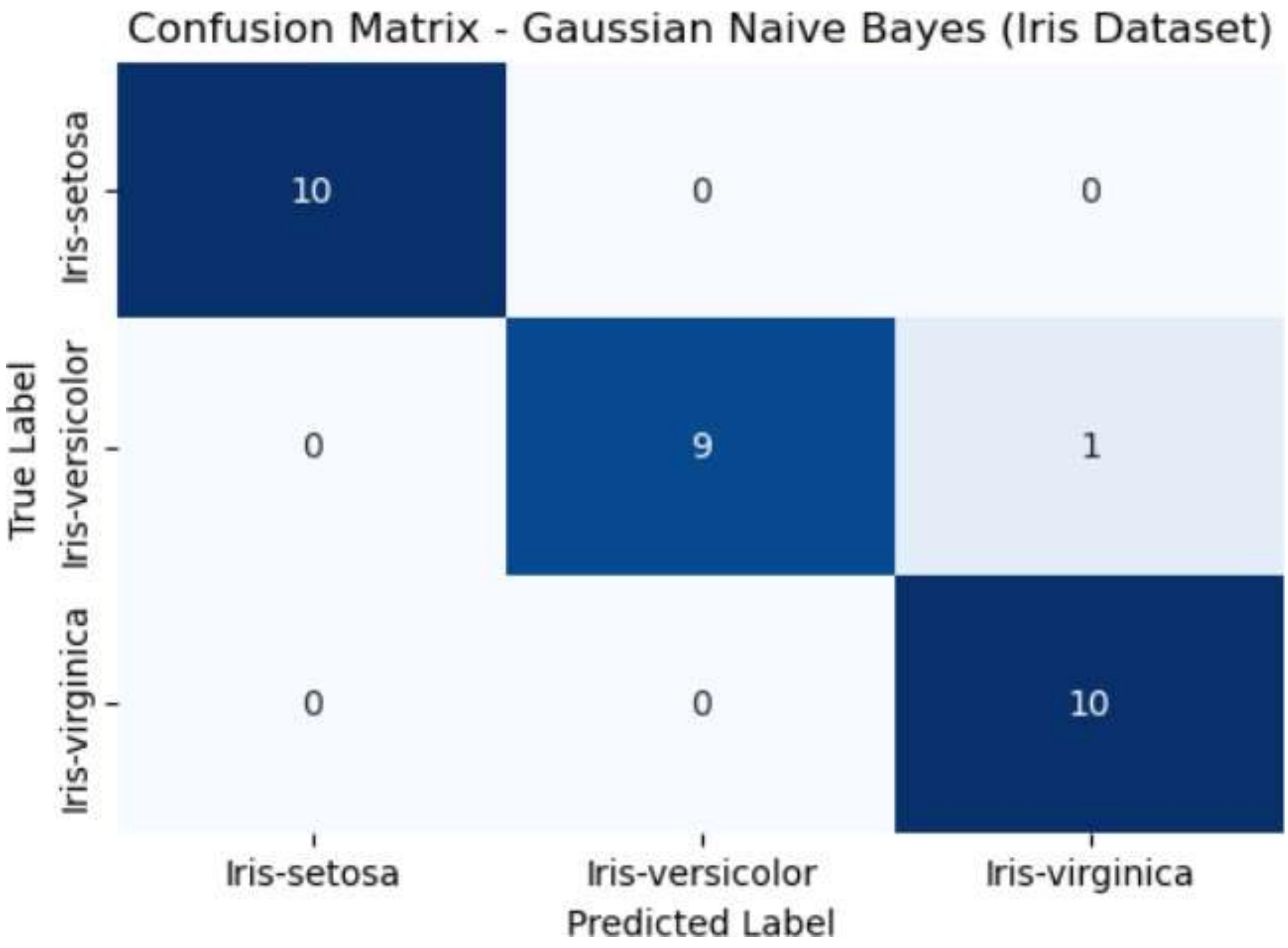
This shows that Naive Bayes, despite its simple assumptions of feature independence, performs surprisingly well on structured datasets with clearly distinguishable classes. It is efficient, interpretable, and a great baseline for many classification problems.

8



```
[5]: # Step 1: Import Libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, classification_report

# Step 2: Load dataset
df = pd.read_csv("Iris.csv")
print(df.head())
print(df.columns)  # check column names

# Step 3: Prepare data
X = df.drop(['Id', 'Species'], axis=1)  # drop ID and target column
y = df['Species']

# Step 4: Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Step 5: Train Bayesian classifier
model = GaussianNB()
model.fit(X_train, y_train)

# Step 6: Predict
y_pred = model.predict(X_test)

# Step 7: Evaluate
print("\nModel Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Precision (macro):", precision_score(y_test, y_pred, average='macro'))
print("Recall (macro):", recall_score(y_test, y_pred, average='macro'))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
     Id   SepalLemgthe     SepalNmiöth€    PetadLemgth€    PMatemiöthe        Species
0    1            5.1            3.5            1. 4            0.2   In 1s-s   0sa
1    2            4. 9           3.0            1. 4            0.2   In 1s-s   0sa
2    3            4. 7           3.2            1. 3            0.2   In 1s-s   0sa
3    4            4. 6           3.1            1. 5            0.2   In 1s-s   0sa
4    5            5.0            3.6            1. 4            0.2   In 1s-s   0sa
Inda( ['Id' , 'SepalLengthere', 'SepalNiöthere', 'PetadLengthere', 'PMa1Nidthire',
       'Species'],
      dtype='object')

model Evatuation:
Accu£acy: 0.9666666666666667
Confusion matrix:

 [ 0  9  1]

Precision (macro): e.9696969696969697
Recat   (macro): e.9666666666666667

classifi cation Report:
                     precision    recall  fi- score   support

    I£1S-SAOSa          1.BB      1.BB      1.BB        10
Iris-versicolor         1.BB      B.9Z      B.95        10
 Iris-virginica         B.91      1.BB      B.95        10

       accuracy                             B.97        30
      macro avg         B.97      B.97      B.97        3B
   weigttedavg          B.97      B.97      B.97        30
```



	Iris-setosa	Iris-versicolor	Iris-virginica
Iris-setosa	10	0	0
Iris-versicolor	0	9	1
Iris-virginica	0	0	10

Assignment No. 9	Assignment Date:
Title: Implement K means clustering algorithm on multidimensional dataset from UCI repository and analyze the performance using Silhouette Score and cluster visualization	Submission Date:

Problem Statement:

Implement K means clustering algorithm on multidimensional dataset from UCI repository and analyze the performance using Silhouette Score and cluster visualization.

Theory:

K-Means Clustering

K-Means is an **unsupervised learning** algorithm used for **clustering** data into **K distinct groups** based on feature similarity.

➢ **How it works**

1. Choose the number of clusters K
2. Initialize K centroids randomly
3. Assign each data point to the nearest centroid
4. Recompute centroids as the mean of assigned points
5. Repeat steps 3–4 until convergence (centroids stop changing)

➢ **Silhouette Score**

Used to measure the **quality of clustering**:

- Ranges from **-1 to +1**
- Closer to **1** indicates well-separated clusters

Silhouette Score=b−a/max(a,b)

Where:

- a: Mean intra-cluster distance
- b: Mean nearest-cluster distance

Acceptance Criteria:

(list of scenarios to be completed in solution for completing assignments)

Sr. No.	**Acceptance Criteria**	**Remarks Instructor (completed/ Not completed)**
1	Dataset loaded from UCI/sklearn, Data normalized	
2	K-Means clustering applied	
3	Silhouette Score computed	
4	PCA used for dimensionality reduction	
5	Cluster plot generated, Output interpreted	

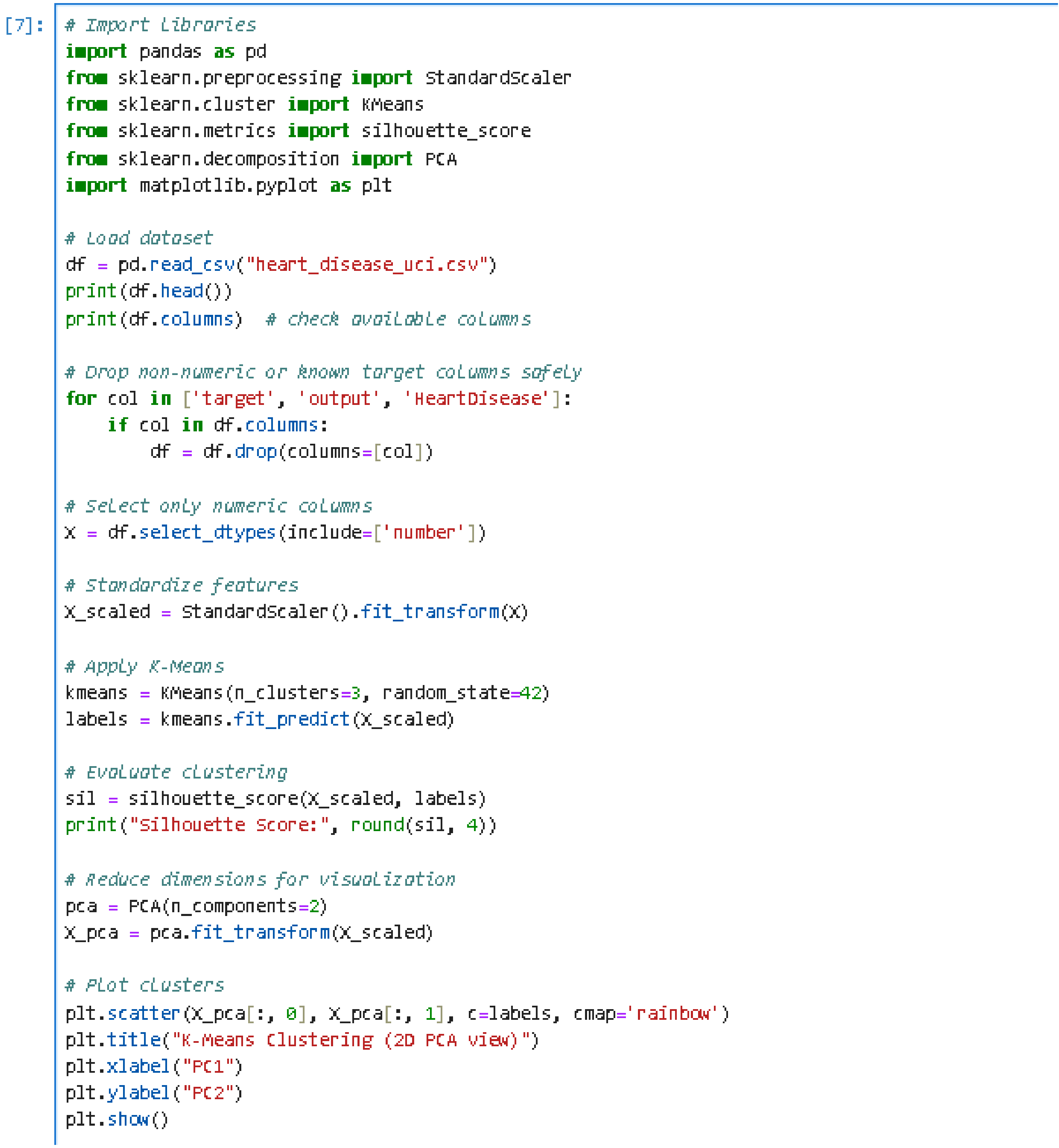
Workflow (Using Python and Pandas/Scikit-learn)

1) Load a multidimensional dataset from UCI (e.g., **Wine**, **Iris**, or **Breast Cancer**)
2) Preprocess the data (normalization or scaling)
3) Choose an appropriate value for K
4) Apply K-Means clustering
5) Visualize the clusters (use PCA if data has >2 dimensions)
6) Evaluate the clustering using Silhouette Score

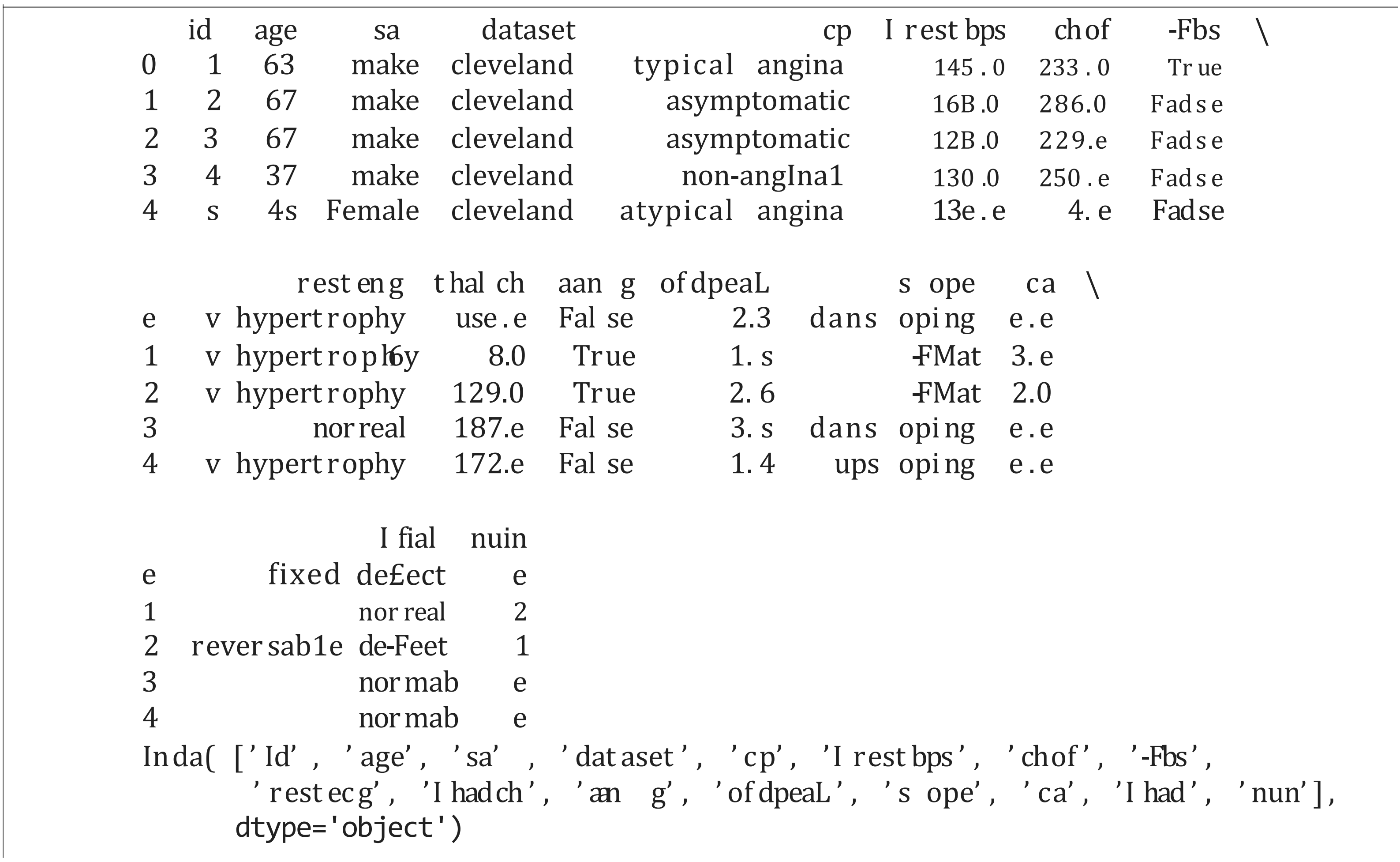
Conclusion:

In this experiment, we successfully applied K-Means clustering to the Wine dataset from the UCI repository. The data was preprocessed using standardization, and the clusters were visualized using PCA. The Silhouette Score helped us evaluate the cluster separation and cohesion. Though K-Means does not use label information, it was able to form meaningful groupings in the dataset, showing its utility for unsupervised pattern discovery.

This demonstrates the practical value of K-Means in clustering multidimensional data and the usefulness of Silhouette Score for model evaluation.

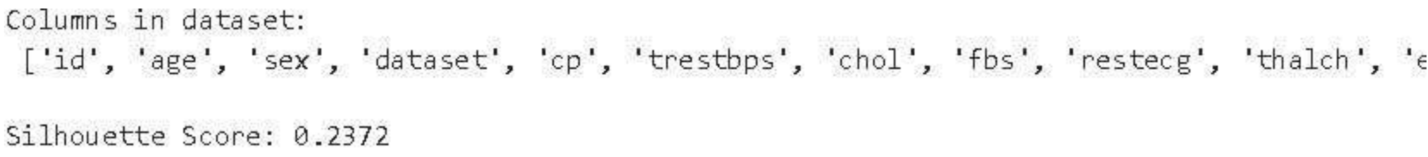


```
[7]: # Import libraries
     import pandas as pd
     from sklearn.preprocessing import StandardScaler
     from sklearn.cluster import KMeans
     from sklearn.metrics import silhouette_score
     from sklearn.decomposition import PCA
     import matplotlib.pyplot as plt

     # Load dataset
     df = pd.read_csv("heart_disease_uci.csv")
     print(df.head())
     print(df.columns)  # check available columns

     # Drop non-numeric or known target columns safely
     for col in ['target', 'output', 'HeartDisease']:
         if col in df.columns:
             df = df.drop(columns=[col])

     # Select only numeric columns
     X = df.select_dtypes(include=['number'])

     # Standardize features
     X_scaled = StandardScaler().fit_transform(X)

     # Apply K-Means
     kmeans = KMeans(n_clusters=3, random_state=42)
     labels = kmeans.fit_predict(X_scaled)

     # Evaluate clustering
     sil = silhouette_score(X_scaled, labels)
     print("Silhouette Score:", round(sil, 4))

     # Reduce dimensions for visualization
     pca = PCA(n_components=2)
     X_pca = pca.fit_transform(X_scaled)

     # Plot clusters
     plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='rainbow')
     plt.title("K-Means Clustering (2D PCA view)")
     plt.xlabel("PC1")
     plt.ylabel("PC2")
     plt.show()
```

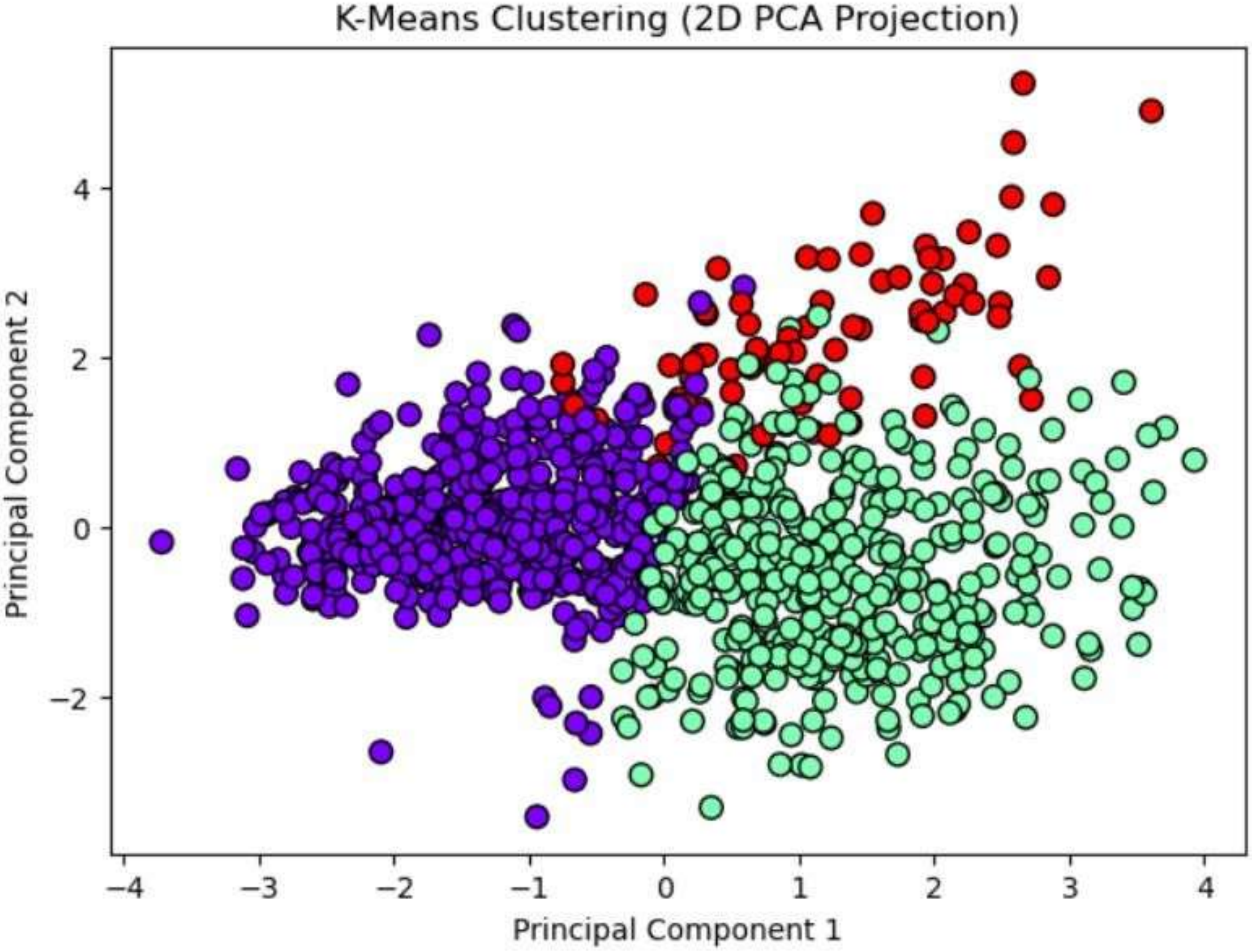


```
   id  age      sa    dataset                cp  I restbps  chof   -Fbs  \
0   1   63    make  cleveland   typical angina       145.0  233.0   True
1   2   67    make  cleveland     asymptomatic       16B.0  286.0  Fadse
2   3   67    make  cleveland     asymptomatic       12B.0  229.e  Fadse
3   4   37    make  cleveland      non-angIna1       130.0  250.e  Fadse
4   s   4s  Female  cleveland  atypical angina       13e.e    4.e  Fadse

         resteng  thalch  aan g  ofdpeaL      s ope   ca  \
e  v hypertrophy   use.e  False      2.3  dans oping  e.e
1  v hypertroph6y    8.0   True      1.s      -FMat  3.e
2  v hypertrophy   129.0   True      2.6      -FMat  2.0
3        norreal   187.e  False      3.s  dans oping  e.e
4  v hypertrophy   172.e  False      1.4   ups oping  e.e

               I fial  nuin
e       fixed de£ect     e
1            norreal     2
2  reversab1e de-Feet    1
3             normab     e
4             normab     e
Inda( ['Id' , 'age', 'sa' , 'dataset', 'cp', 'I restbps', 'chof', '-Fbs',
       'restecg', 'I hadch', 'aan  g', 'ofdpeaL', 's ope', 'ca', 'I had', 'nun'],
      dtype='object')
```

9



```
Columns in dataset:
 ['id', 'age', 'sex', 'dataset', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalch', 'e

Silhouette Score: 0.2372
```



Assignment No. 10	Assignment Date:
Title: Use Wine dataset or IRIS dataset. Apply PCA to reduce features to 2D, then visualize clustering using K- means. Analyze how well PCA preserved cluster separation	Submission Date:

Problem Statement:

Use Wine dataset or IRIS dataset. Apply PCA to reduce features to 2D, then visualize clustering using K- means. Analyze how well PCA preserved cluster separation

Theory:

Principal Component Analysis (PCA)

PCA is a **dimensionality reduction technique** that transforms high-dimensional data into fewer dimensions while retaining as much **variance (information)** as possible.

- It finds new orthogonal axes (principal components).
- The first principal component captures the most variance, the second captures the next most, and so on.
- PCA is **unsupervised** and does **not use class labels**.

◆ **K-Means Clustering**

K-Means groups data into **K clusters** based on feature similarity.

- It minimizes the **intra-cluster distance**.
- The clustering can be visualized better in 2D after PCA.

Acceptance Criteria:

(list of scenarios to be completed in solution for completing assignments)

Sr. No.	Acceptance Criteria	Remarks Instructor (completed/ Not completed)
1	Dataset used (Wine or Iris), PCA applied and explained variance shown	
2	K-Means applied to 2D PCA data	
3	Cluster visualization created	
4	Silhouette Score computed	
5	Result analyzed	

Workflow (Using Python and Pandas/Scikit-learn)

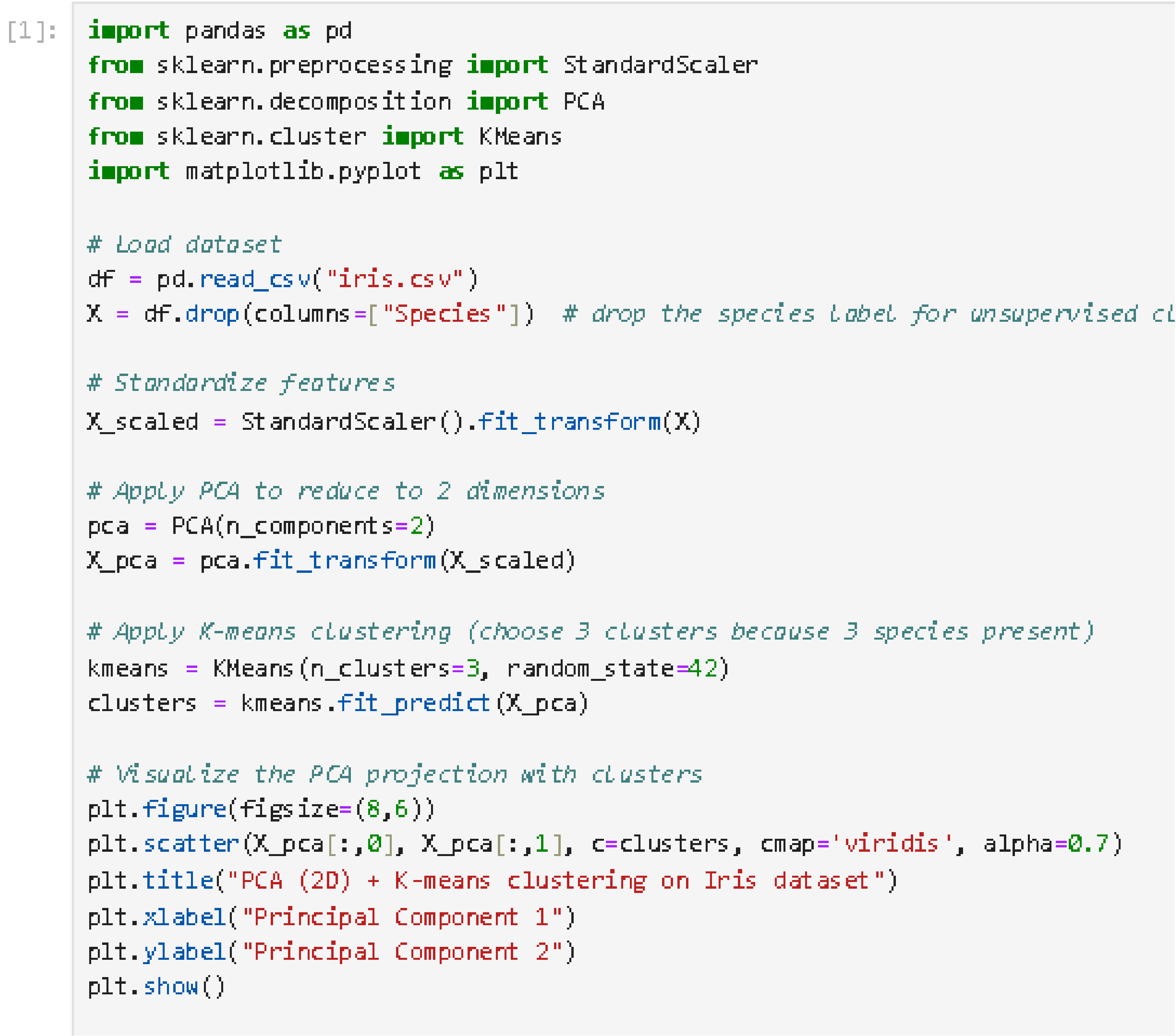
1. Load the **Wine** or **Iris** dataset
2. Standardize the features
3. Apply **PCA** to reduce dimensions to 2
4. Use **K-Means** to cluster the data in 2D
5. Visualize the clusters
6. Compare clusters to actual labels to analyze separation
7. Compute **Silhouette Score** to measure clustering quality

Conclusion:

In this experiment, we applied PCA to reduce the Wine dataset from 13 features to 2 principal components. Although some variance is lost (only ~55% retained), the PCA-transformed data still preserved meaningful structure, allowing K-Means clustering to produce visibly distinct clusters.

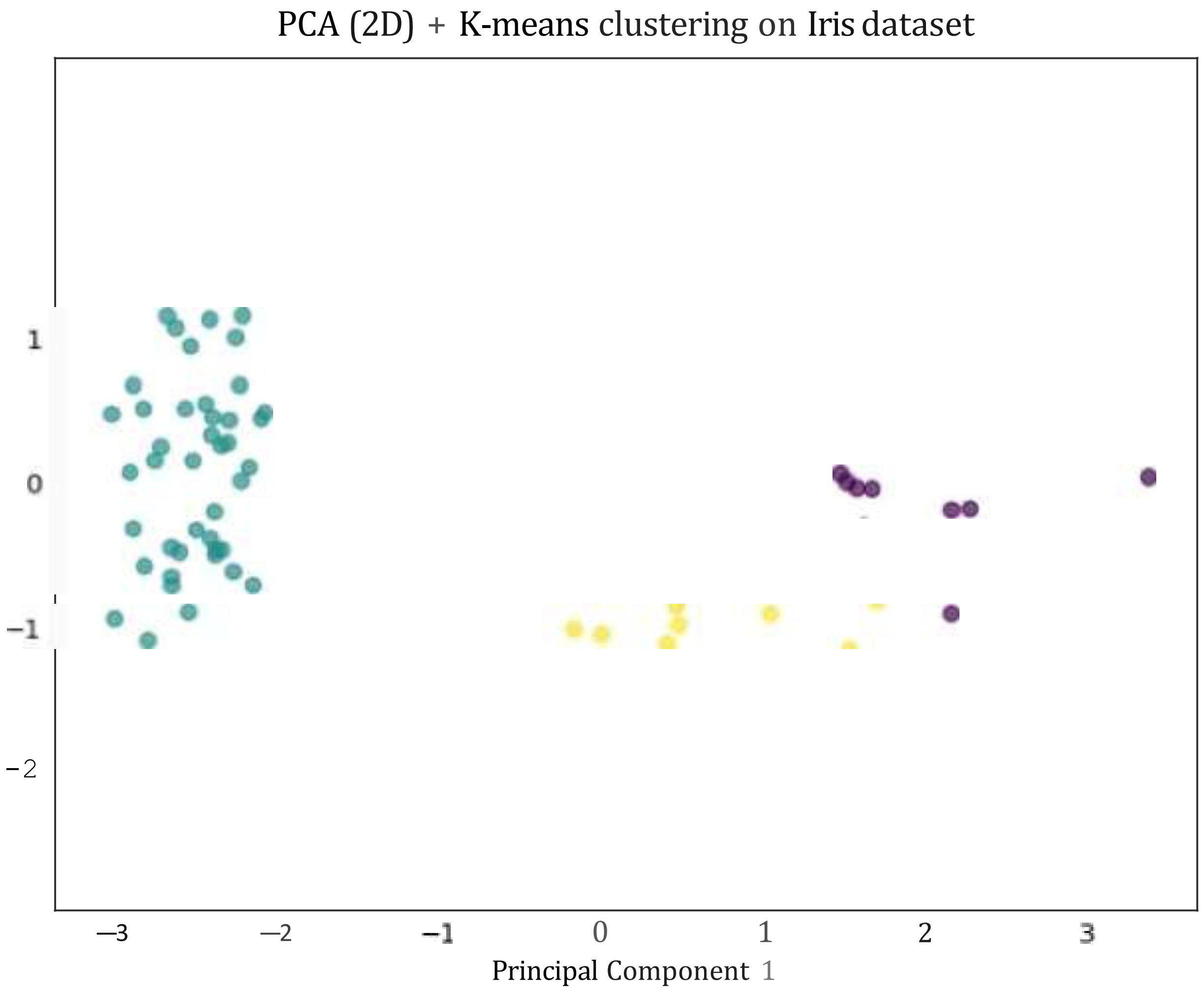
The Silhouette Score further confirmed good clustering performance in the reduced 2D space. This shows that PCA is effective not only for visualization but also for dimensionality reduction before clustering, especially when data is high-dimensional.

10



[1]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("iris.csv")
X = df.drop(columns=["Species"])  # drop the species label for unsupervised cl

# Standardize features
X_scaled = StandardScaler().fit_transform(X)

# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Apply K-means clustering (choose 3 clusters because 3 species present)
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_pca)

# Visualize the PCA projection with clusters
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=clusters, cmap='viridis', alpha=0.7)
plt.title("PCA (2D) + K-means clustering on Iris dataset")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```



PCA (2D) + K-means clustering on Iris dataset