Lab Manual

of

Artificial Intelligence Laboratory (CSE608)

Bachelor of Technology (CSE)

By

Ramoliya Kaushal (22000409)

Third Year, Semester 6

Course In-charge: Prof. Jaideepsinh Raulji

NAVRACHANA UNIVERSITY

a UGC recognized University

Department of Computer Science and Engineering

School Engineering and Technology

Navrachana University, Vadodara

Spring Semester

(2025)

INDEX

PRACTICAL: - 1

AIM: Write a program in Python to find factorial of a number using a loop. Also find the same using a recursive function. Implement this creating both the function in a class.

PROGRAM CODE: -

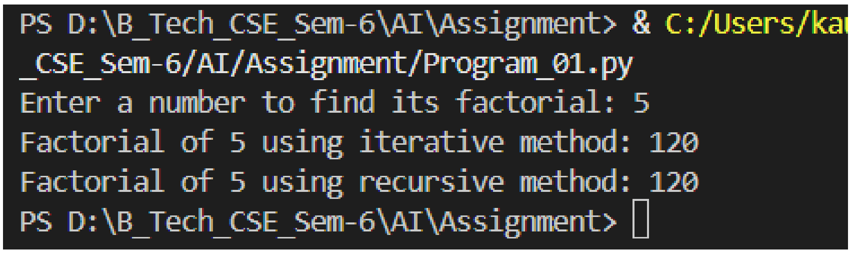
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 1. - Write a program in Python to find factorial of a number using a loop. Also find the same using a recursive function. Implement this creating both the function in a class.
'''
class Factorial:
    # Method to calculate factorial using a loop
    def factorial_iterative(self, n):
        result = 1
        for i in range(1, n + 1):
            result *= i
        return result

    # Method to calculate factorial using recursion
    def factorial_recursive(self, n):
        if n == 0 or n == 1:
            return 1
        else:
            return n * self.factorial_recursive(n - 1)

# Main function to test the class methods
if __name__ == "__main__":
    num = int(input("Enter a number to find its factorial: "))
    fact = Factorial()
```

```
        print(f"Factorial of {num} using iterative method: {fact.factorial_iterative(num)}")

        print(f"Factorial of {num} using recursive method: {fact.factorial_recursive(num)}")
```

INPUT: -

Enter a number to find its factorial: 5

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/ka
_CSE_Sem-6/AI/Assignment/Program_01.py
Enter a number to find its factorial: 5
Factorial of 5 using iterative method: 120
Factorial of 5 using recursive method: 120
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

CONCLUSION: - This program efficiently calculates the factorial of a number using both iterative and recursive methods, showcasing different algorithmic approaches. Encapsulation in a class ensures modularity and reusability of the code.

PRACTICAL: - 2

AIM: Write a python program to implement stack and queue using OOP paradigm.

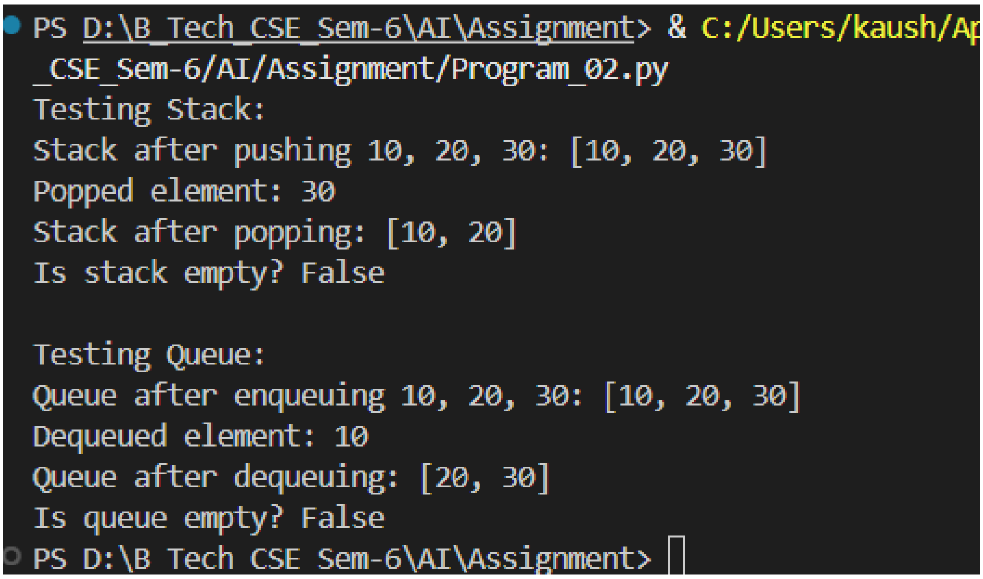
PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 2. - Write a python program to implement stack and queue using OOP paradigm.
'''
class Stack:
    def __init__(self):
        self.stack = []

    def push(self, item):
        self.stack.append(item)

    def pop(self):
        if not self.is_empty():
            return self.stack.pop()
        else:
            return "Stack is empty"

    def is_empty(self):
        return len(self.stack) == 0

    def display(self):
        return self.stack

class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        self.queue.append(item)

    def is_empty(self):
        return len(self.queue) == 0

    def display(self):
        return self.queue

if __name__ == "__main__":
    print("Testing Stack:")
    stack = Stack()
    stack.push(10)
    stack.push(20)
    stack.push(30)
    print("Stack after pushing 10, 20, 30:", stack.display())
    print("Popped element:", stack.pop())
    print("Stack after popping:", stack.display())
    print("Is stack empty?", stack.is_empty())

    print("\nTesting Queue:")
    queue = Queue()
    queue.enqueue(10)
    queue.enqueue(20)
    queue.enqueue(30)
    print("Queue after enqueuing 10, 20, 30:", queue.display())
    if not queue.is_empty():
        removed_element = queue.queue.pop(0)  # Manual removal of the first element
        print("Dequeued element:", removed_element)
    else:
        print("Queue is empty")
    print("Queue after dequeuing:", queue.display())
    print("Is queue empty?", queue.is_empty())
```

INPUT: -

```
print("Testing Stack:")
stack = Stack()
stack.push(10)
stack.push(20)
stack.push(30)
print("Stack after pushing 10, 20, 30:", stack.display())
print("Popped element:", stack.pop())
print("Stack after popping:", stack.display())
print("Is stack empty?", stack.is_empty())

print("\nTesting Queue:")
queue = Queue()
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)
print("Queue after enqueuing 10, 20, 30:", queue.display())
if not queue.is_empty():
    removed_element = queue.queue.pop(0)  # Manual removal of the first element
    print("Dequeued element:", removed_element)
else:
    print("Queue is empty")
print("Queue after dequeuing:", queue.display())
print("Is queue empty?", queue.is_empty())
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/Ap
_CSE_Sem-6/AI/Assignment/Program_02.py
Testing Stack:
Stack after pushing 10, 20, 30: [10, 20, 30]
Popped element: 30
Stack after popping: [10, 20]
Is stack empty? False

Testing Queue:
Queue after enqueuing 10, 20, 30: [10, 20, 30]
Dequeued element: 10
Queue after dequeuing: [20, 30]
Is queue empty? False
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: -This program demonstrates the implementation of stack and queue using the OOP paradigm. It effectively showcases the functionality of both data structures, including operations like push, pop, enqueue, and manual dequeue, while maintaining modularity and clarity in design.

PRACTICAL: - 3

AIM: Write a python program to create a binary tree, add elements, retrieve elements using pre order, post-order and in-order traversal.

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 3. - Write a python program to create a binary tree, add elements,
retrieve elements using pre
order, post-order and in-order traversal.
'''
class Node:
  def __init__(self, value):
    self.value = value
    self.left = None
    self.right = None


class BinaryTree:
  def __init__(self):
    self.root = None


  def add(self, value):
    if self.root is None:
      self.root = Node(value)
    else:
      self._add(self.root, value)


  def _add(self, current, value):
    if value < current.value:
      if current.left is None:
```

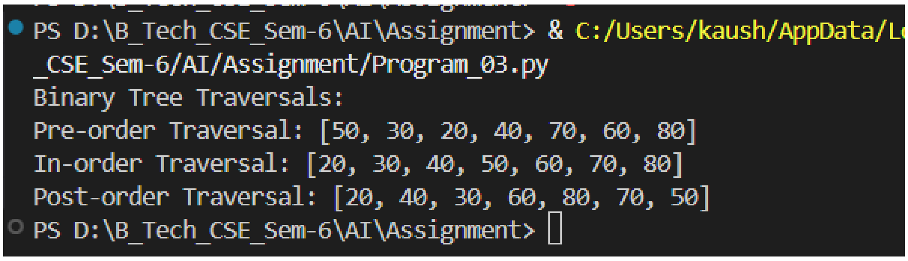
```
                current.left = Node(value)
            else:
                self._add(current.left, value)
        else:
            if current.right is None:
                current.right = Node(value)
            else:
                self._add(current.right, value)

    def pre_order(self, node, result):
        if node:
            result.append(node.value)
            self.pre_order(node.left, result)
            self.pre_order(node.right, result)

    def in_order(self, node, result):
        if node:
            self.in_order(node.left, result)
            result.append(node.value)
            self.in_order(node.right, result)

    def post_order(self, node, result):
        if node:
            self.post_order(node.left, result)
            self.post_order(node.right, result)
            result.append(node.value)

if __name__ == "__main__":
    tree = BinaryTree()
```

```
elements = [50, 30, 70, 20, 40, 60, 80]

for elem in elements:
   tree.add(elem)

print("Binary Tree Traversals:")
pre_order_result = []
tree.pre_order(tree.root, pre_order_result)
print("Pre-order Traversal:", pre_order_result)

in_order_result = []
tree.in_order(tree.root, in_order_result)
print("In-order Traversal:", in_order_result)

post_order_result = []
tree.post_order(tree.root, post_order_result)
print("Post-order Traversal:", post_order_result)
```

INPUT: -

elements = [50, 30, 70, 20, 40, 60, 80]

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/L
_CSE_Sem-6/AI/Assignment/Program_03.py
Binary Tree Traversals:
Pre-order Traversal: [50, 30, 20, 40, 70, 60, 80]
In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
Post-order Traversal: [20, 40, 30, 60, 80, 70, 50]
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program successfully implements a binary tree with methods to add elements and perform pre-order, in-order, and post-order traversals. It demonstrates the fundamental operations of binary trees and provides a clear understanding of tree traversal techniques.

PRACTICAL: - 4

AIM: Write a Program in Python to solve tic-tac-toe problem implementing minimax algorithm.

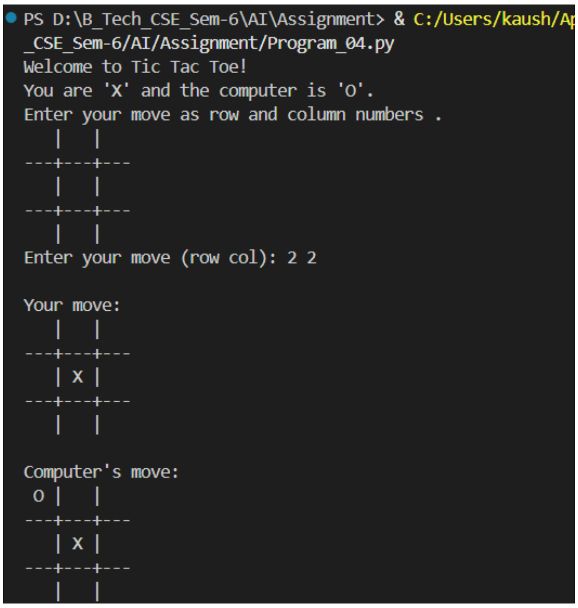
PROGRAM CODE: -

```
'''

@author: 22000409 Kaushal Ramoliya

@description: 4. - Write a Program in Python to solve tic-tac-toe problem
implementing minimax algorithm.

'''

import math


def print_board(board):

    for i in range(3):

        print(" " + " | ".join(board[i*3:(i+1)*3]))

        if i < 2:

            print("---+---+---")


def is_winner(board, player):

    win_conditions = [

        [0, 1, 2], [3, 4, 5], [6, 7, 8],

        [0, 3, 6], [1, 4, 7], [2, 5, 8],

        [0, 4, 8], [2, 4, 6]

    ]

    for condition in win_conditions:

        if board[condition[0]] == board[condition[1]] == board[condition[2]] == player:

            return True

    return False


def is_moves_left(board):
    return ' ' in board


def evaluate(board):
    if is_winner(board, 'O'):
        return 10
    elif is_winner(board, 'X'):
        return -10
    return 0


def minimax(board, depth, is_maximizing):
    score = evaluate(board)

    if score == 10 or score == -10:
        return score
    if not is_moves_left(board):
        return 0

    if is_maximizing:
        best = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'O'
                best = max(best, minimax(board, depth + 1, False))
                board[i] = ' '
        return best
    else:
        best = math.inf
        for i in range(9):
            if board[i] == ' ':
```

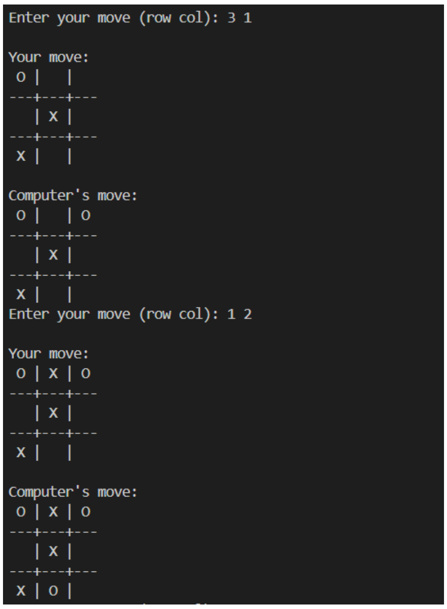
```
            board[i] = 'X'
            best = min(best, minimax(board, depth + 1, True))
            board[i] = ' '
        return best

def find_best_move(board):
    best_val = -math.inf
    best_move = -1

    for i in range(9):
        if board[i] == ' ':
            board[i] = 'O'
            move_val = minimax(board, 0, False)
            board[i] = ' '
            if move_val > best_val:
                best_val = move_val
                best_move = i
    return best_move

def main():
    board = [' ' for _ in range(9)]
    print("Welcome to Tic Tac Toe!")
    print("You are 'X' and the computer is 'O'.")
    print("Enter your move as row and column numbers .")
    print_board(board)

    while True:
        try:
            user_input = input("Enter your move (row col): ")
            row, col = map(int, user_input.split())
            if row < 1 or row > 3 or col < 1 or col > 3:
                print("Invalid input. Row and column numbers must be between 1 and 3.")
                continue
            index = (row - 1) * 3 + (col - 1)
            if board[index] != ' ':
                print("That cell is already occupied. Try another move.")
                continue
        except ValueError:
            print("Invalid input. Please enter two numbers separated by a space.")
            continue

        board[index] = 'X'
        print("\nYour move:")
        print_board(board)

        if is_winner(board, 'X'):
            print("Congratulations! You win!")
            break

        if not is_moves_left(board):
            print("It's a draw!")
            break

        comp_move = find_best_move(board)
        board[comp_move] = 'O'
        print("\nComputer's move:")
        print_board(board)
```

```
        if is_winner(board, 'O'):
            print("Computer wins!")
            break

        if not is_moves_left(board):
            print("It's a draw!")
            break

if __name__ == "__main__":
    main()
```

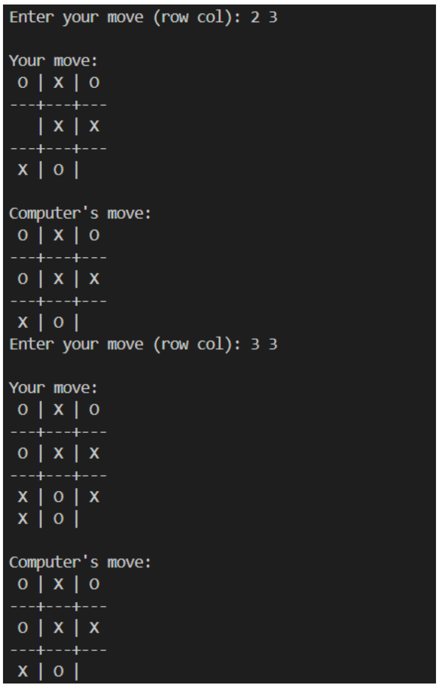
OUTPUT: -



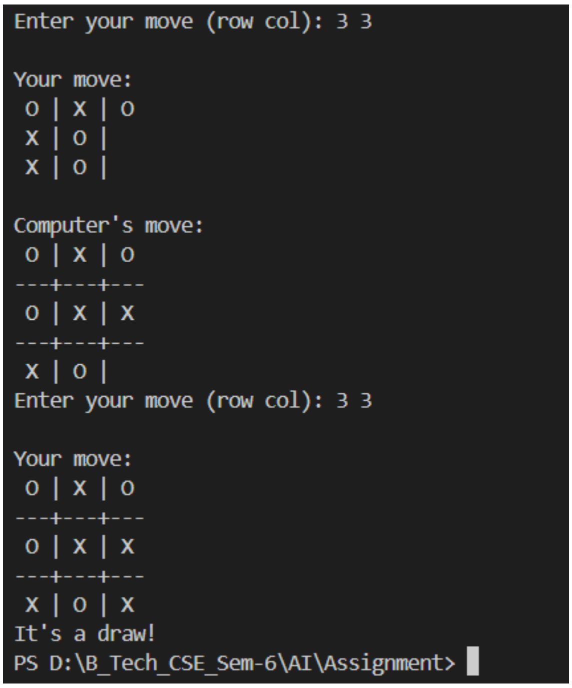
```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/Ap
_CSE_Sem-6/AI/Assignment/Program_04.py
Welcome to Tic Tac Toe!
You are 'X' and the computer is 'O'.
Enter your move as row and column numbers .
   |   |
---+---+---
   |   |
---+---+---
   |   |
Enter your move (row col): 2 2

Your move:
   |   |
---+---+---
   | X |
---+---+---
   |   |

Computer's move:
 O |   |
---+---+---
   | X |
---+---+---
   |   |
```



```
Enter your move (row col): 3 1

Your move:
 O |   |
---+---+---
   | X |
---+---+---
 X |   |

Computer's move:
 O |   | O
---+---+---
   | X |
---+---+---
 X |   |
Enter your move (row col): 1 2

Your move:
 O | X | O
---+---+---
   | X |
---+---+---
 X |   |

Computer's move:
 O | X | O
---+---+---
   | X |
---+---+---
 X | O |
```



```
Enter your move (row col): 2 3

Your move:
 o | x | o
---+---+---
   | x | x
---+---+---
 x | o |

Computer's move:
 o | x | o
---+---+---
 o | x | x
---+---+---
 x | o |
Enter your move (row col): 3 3

Your move:
 o | x | o
---+---+---
 o | x | x
---+---+---
 x | o | x
 x | o |

Computer's move:
 o | x | o
---+---+---
 o | x | x
---+---+---
 x | o |
```



```
Enter your move (row col): 3 3

Your move:
 o | x | o
 x | o |
 x | o |

Computer's move:
 o | x | o
---+---+---
 o | x | x
---+---+---
 x | o |
Enter your move (row col): 3 3

Your move:
 o | x | o
---+---+---
 o | x | x
---+---+---
 x | o | x
It's a draw!
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program effectively implements the Tic-Tac-Toe game using the Minimax algorithm, allowing the computer to make optimal moves. It demonstrates the use of game theory concepts to evaluate all possible outcomes, ensuring the computer either wins or forces a draw when possible. The program also provides an interactive and user-friendly interface for gameplay

PRACTICAL: - 5

AIM: Write a program in Python for Breadth First Search.

PROGRAM CODE: -

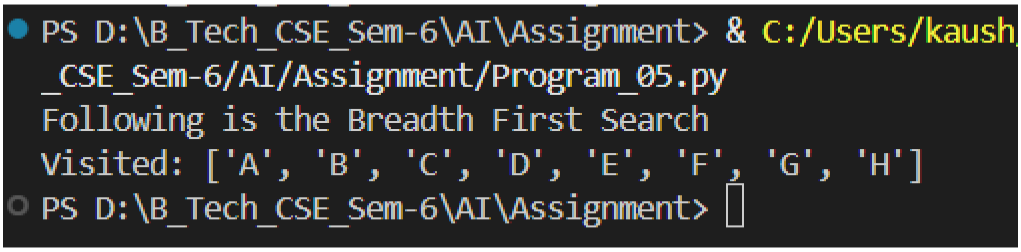
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 5. - Write a program in Python for Breadth First Search.
'''
class Graph:
    def __init__(self, graph):
        self.graph = graph
        self.visited = []
        self.queue = []

    def bfs(self, start_node):
        self.queue.append(start_node)
        self.visited.append(start_node)

        while self.queue:
            node = self.queue.pop(0)
            for child in self.graph[node]:
                if child not in self.visited:
                    self.queue.append(child)
                    self.visited.append(child)

        print("Visited:", self.visited)

# Graph definition
graph_data = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': ['H'],
    'E': ['H'],
    'F': ['H'],
    'G': ['H'],
    'H': []
}

# Creating Graph object
graph = Graph(graph_data)
print("Following is the Breadth First Search")
graph.bfs('A')
```

INPUT: -

```
# Graph definition
graph_data = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': ['H'],
    'E': ['H'],
    'F': ['H'],
    'G': ['H'],
    'H': []
}

# Creating Graph object
graph = Graph(graph_data)
```

```
print("Following is the Breadth First Search")
graph.bfs('A')
```

OUTPUT: -

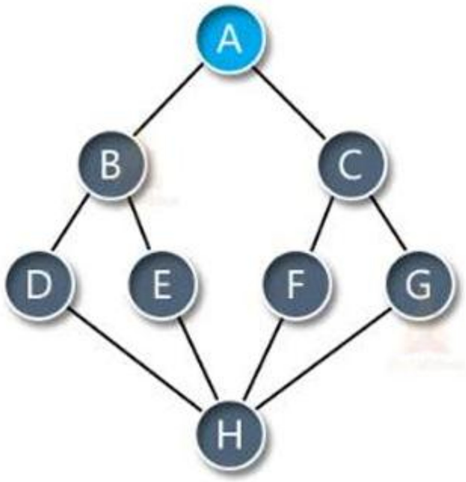


```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush
_CSE_Sem-6/AI/Assignment/Program_05.py
Following is the Breadth First Search
Visited: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program demonstrates the Breadth-First Search (BFS) algorithm for traversing a graph. It effectively explores all nodes level by level starting from a given node, showcasing the BFS traversal technique. The implementation is simple, efficient, and highlights the use of a queue to manage the traversal process.

PRACTICAL: - 6

AIM: Write a python program to implement Breadth First Search and Depth First Search algorithm on following graph. Consider start node as A.



PROGRAM CODE: -

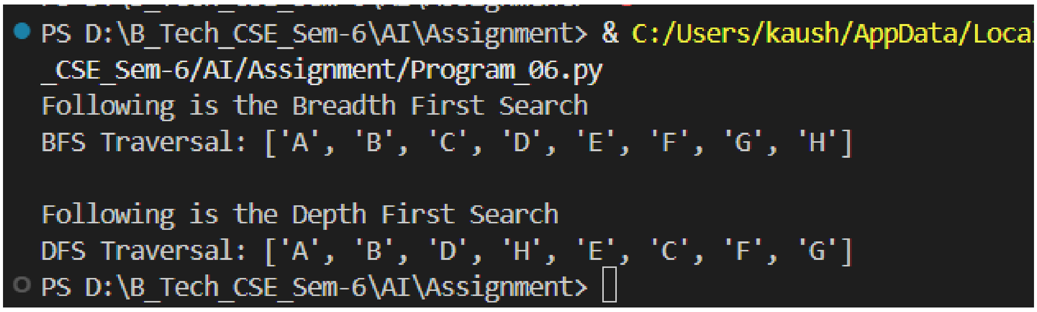
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 6. - Write a python program to implement Breadth First Search and Depth First Search algorithm
on following graph. Consider start node as A.
'''
class Graph:
    def __init__(self, graph):
        self.graph = graph

    def bfs(self, start_node):
        visited = []
        queue = []

        queue.append(start_node)
        visited.append(start_node)

        while queue:
            node = queue.pop(0)
            for child in self.graph[node]:
                if child not in visited:
                    queue.append(child)
                    visited.append(child)

        print("BFS Traversal:", visited)

    def dfs(self, start_node):
        visited = []
        stack = []

        stack.insert(0, start_node)

        while stack:
            node = stack.pop(0)
            if node not in visited:
                visited.append(node)
                for child in reversed(self.graph[node]):
                    if child not in visited:
                        stack.insert(0, child)

        print("DFS Traversal:", visited)

# Graph definition
graph_data = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': ['H'],
    'E': ['H'],
    'F': ['H'],
    'G': ['H'],
    'H': []
}

# Creating Graph object
graph = Graph(graph_data)

# Run both traversals
print("Following is the Breadth First Search")
graph.bfs('A')

print("\nFollowing is the Depth First Search")
graph.dfs('A')
```

INPUT: -

```
# Graph definition
graph_data = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': ['H'],
    'E': ['H'],
    'F': ['H'],
    'G': ['H'],
    'H': []
}
```

```
# Creating Graph object
graph = Graph(graph_data)

# Driver code
print("Following is the Depth First Search")
graph.dfs('A')
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Loca
_CSE_Sem-6/AI/Assignment/Program_06.py
Following is the Breadth First Search
BFS Traversal: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']

Following is the Depth First Search
DFS Traversal: ['A', 'B', 'D', 'H', 'E', 'C', 'F', 'G']
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program efficiently implements both Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms to traverse a graph starting from node 'A

PRACTICAL: - 7

AIM: Write a program in Python to implement Depth First Search.

PROGRAM CODE: -

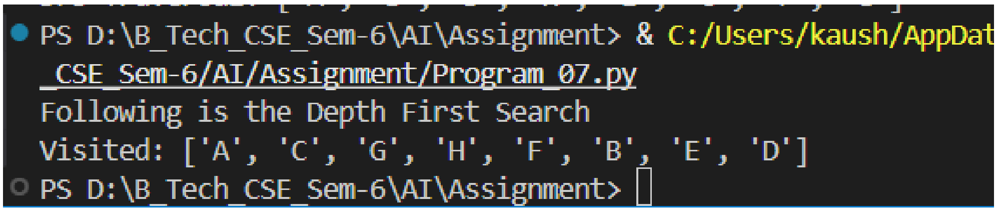
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 7. - Write a program in Python to implement Depth First Search.
'''
class Graph:
    def __init__(self, graph):
        self.graph = graph
        self.visited = []
        self.stack = []

    def dfs(self, start_node):
        self.stack.insert(0, start_node)

        while self.stack:
            node = self.stack.pop(0)
            if node not in self.visited:
                self.visited.append(node)
                for child in self.graph[node]:
                    if child not in self.visited:
                        self.stack.insert(0, child)

        print("Visited:", self.visited)

# Graph definition
graph_data = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': ['H'],
    'E': ['H'],
    'F': ['H'],
    'G': ['H'],
    'H': []
}

# Creating Graph object
graph = Graph(graph_data)

# Driver code
print("Following is the Depth First Search")
graph.dfs('A')
```

INPUT: -

```
# Graph definition
graph_data = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': ['H'],
    'E': ['H'],
    'F': ['H'],
    'G': ['H'],
    'H': []
}

# Creating Graph object
```

```
graph = Graph(graph_data)


# Driver code
print("Following is the Depth First Search")
graph.dfs('A')
```

OUTPUT: -

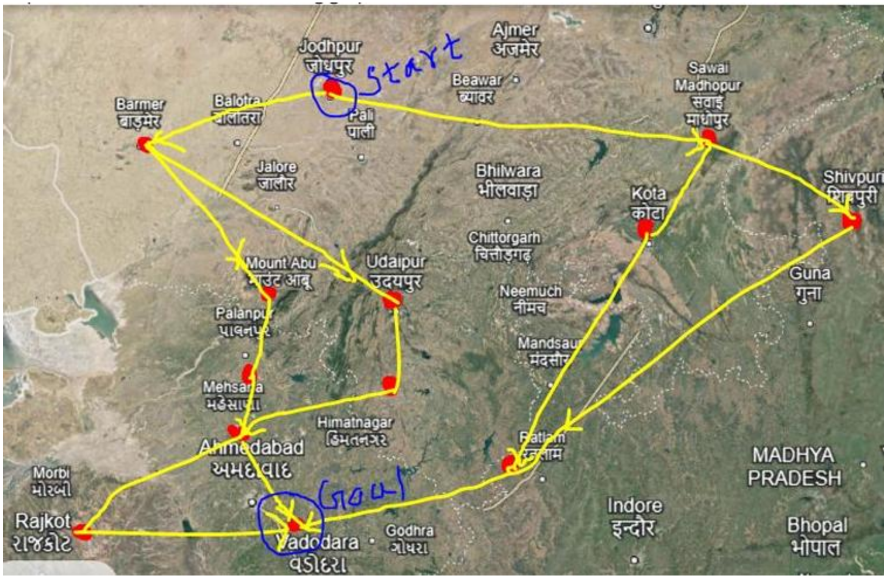


```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppDat
_CSE_Sem-6/AI/Assignment/Program_07.py
Following is the Depth First Search
Visited: ['A', 'C', 'G', 'H', 'F', 'B', 'E', 'D']
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program successfully implements the Depth-First Search (DFS) algorithm to traverse a graph, exploring as far as possible along each branch before backtracking, starting from node 'A'.

PRACTICAL: - 8

AIM: Implement BFS and DFS on following graph.



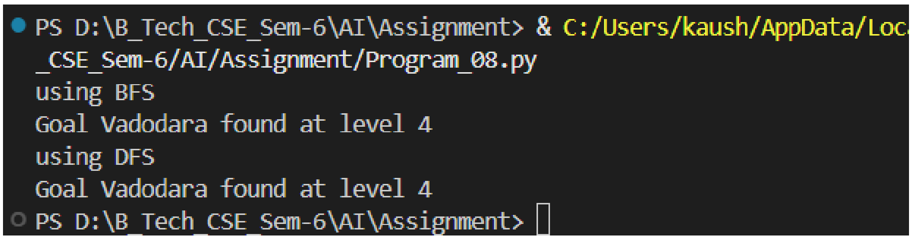
PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 8. - Implement BFS and DFS on following graph.
'''
from collections import deque


class GraphSearch:
    def __init__(self, graph):
        self.graph = graph


    def My_BLS(self, start, goal, limit):
        queue = deque([(start, 0)])  # Use deque for queue functionality
        while queue:
            node, depth = queue.popleft()  # Pop from the left for BFS
            if node == goal:
                return True, goal, depth
            if depth < limit:
                for n in self.graph.get(node, []):
                    queue.append((n, depth + 1))
        return False, goal, -1  # Fixed the return statement

    def My_DLS(self, start, goal, limit):
        stack = [(start, 0)]
        while stack:
            node, depth = stack.pop()
            if node == goal:
                return True, goal, depth
            if depth < limit:
                for n in self.graph.get(node, []):
                    stack.append((n, depth + 1))
        return False, goal, -1  # Fixed the return statement

graph_data = {
    'Jodhpur': ['Barmer', 'Sawai Madhopur'],
    'Barmer': ['Mount Abu', 'Udaipur'],
    'Sawai Madhopur': ['Kota', 'Shivpuri'],
    'Mount Abu': ['Mehsana'],
    'Udaipur': ['Himatnagar'],
    'Kota': ['Ratlam'],
    'Shivpuri': ['Ratlam'],
    'Mehsana': ['Ahmedabad'],
    'Himatnagar': ['Ahmedabad'],
    'Ratlam': ['Vadodara'],
    'Ahmedabad': ['Rajkot', 'Vadodara'],
    'Rajkot': ['Vadodara'],
    'Vadodara': []
}

search = GraphSearch(graph_data)
limit = 8

print("using BFS")
b, g, d = search.My_BLS('Jodhpur', 'Vadodara', limit)
if b:
    print("Goal", g, "found at level", d)
else:
    print("Goal", g, "not found within limit", limit)

print("using DFS")
b, g, d = search.My_DLS('Jodhpur', 'Vadodara', limit)
if b:
    print("Goal", g, "found at level", d)
else:
    print("Goal", g, "not found within limit", limit)
```

INPUT: -

```
graph_data = {
    'Jodhpur': ['Barmer', 'Sawai Madhopur'],
    'Barmer': ['Mount Abu', 'Udaipur'],
    'Sawai Madhopur': ['Kota', 'Shivpuri'],
    'Mount Abu': ['Mehsana'],
    'Udaipur': ['Himatnagar'],
    'Kota': ['Ratlam'],
    'Shivpuri': ['Ratlam'],
    'Mehsana': ['Ahmedabad'],
    'Himatnagar': ['Ahmedabad'],
    'Ratlam': ['Vadodara'],
    'Ahmedabad': ['Rajkot', 'Vadodara'],
    'Rajkot': ['Vadodara'],
    'Vadodara': []
}

search = GraphSearch(graph_data)
limit = 8

print("using BFS")
b, g, d = search.My_BLS('Jodhpur', 'Vadodara', limit)
if b:
    print("Goal", g, "found at level", d)
else:
    print("Goal", g, "not found within limit", limit)

print("using DFS")
b, g, d = search.My_DLS('Jodhpur', 'Vadodara', limit)
if b:
    print("Goal", g, "found at level", d)
else:
    print("Goal", g, "not found within limit", limit)
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Loc
_CSE_Sem-6/AI/Assignment/Program_08.py
using BFS
Goal Vadodara found at level 4
using DFS
Goal Vadodara found at level 4
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program implements both Breadth-Limited Search (BLS) and Depth-Limited Search (DLS) algorithms to traverse a graph and find a goal node within a specified depth limit. It demonstrates the use of BFS and DFS techniques effectively, showcasing their application in constrained search problems.

PRACTICAL: - 9

AIM: Write a program in Python for Best First Search

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 9. - Write a program in Python for Best First Search
'''
from queue import PriorityQueue


class Best:
    def __init__(self, graph):
        self.graph_data = graph
        self.visited = []

    def bestf(self, node, goal):
        self.visited.append(node)
        while True:
            tn = node
            if tn == goal:
                break
            queue = PriorityQueue()
            for neighbour, weight in self.graph_data[tn].items():
                queue.put([weight, neighbour])

            tw, tn = queue.queue[0]
            self.visited.append(tn)
            node = tn
        print("Visited:", self.visited)
```

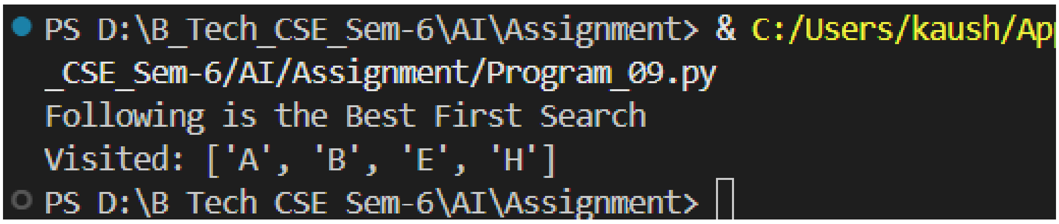
```
# Graph definition
graph = {
'A':{'B':12, 'C':14}, #heuristic value A to H is 13, B to H is 12, C to H is 4
'B':{'D':11, 'E':10},
'C':{'F':6, 'G':7},
'D':{'H':0},
'E':{'H':0},
'F':{'H':0},
'G':{'H':0}
}

# Creating Graph object
graph = Best(graph)

# Driver code
print("Following is the Best First Search")
graph.bestf('A', 'H')
```

INPUT: -

```
# Graph definition
graph = {
'A':{'B':12, 'C':14}, #heuristic value A to H is 13, B to H is 12, C to H is 4
'B':{'D':11, 'E':10},
'C':{'F':6, 'G':7},
'D':{'H':0},
'E':{'H':0},
'F':{'H':0},
'G':{'H':0}
}
```

```
# Creating Graph object
graph = Best(graph)


# Driver code
print("Following is the Best First Search")
graph.bestf('A', 'H')
```

OUTPUT: -

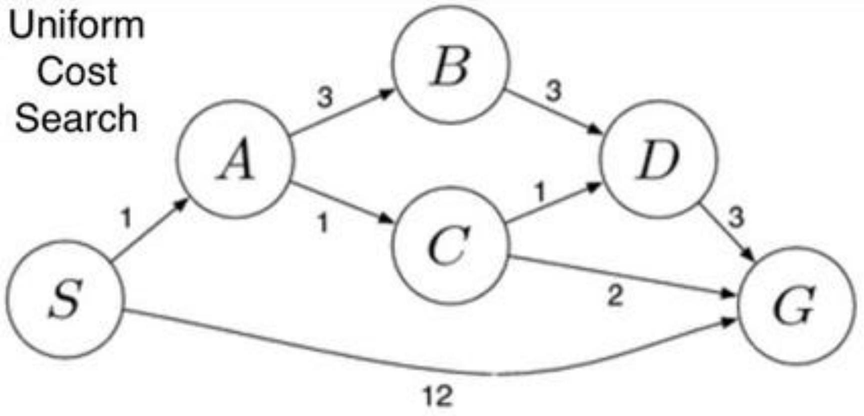


```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/Ap
_CSE_Sem-6/AI/Assignment/Program_09.py
Following is the Best First Search
Visited: ['A', 'B', 'E', 'H']
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program implements the Best First Search algorithm, which uses a priority queue to explore the graph by selecting the node with the lowest heuristic value at each step. It efficiently finds the goal node while minimizing the cost, demonstrating the use of heuristic-based search techniques.

PRACTICAL: - 10

AIM: Write a python program to implement Uniform Cost Search with cumulative cost.



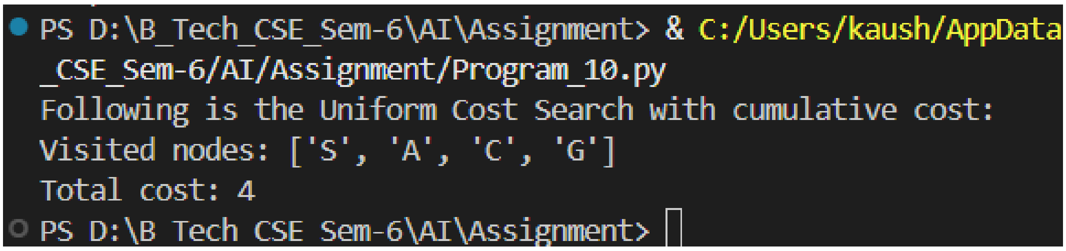
PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 10. - Write a python program to implement Uniform Cost Search with
cumulative cost.
'''
from queue import PriorityQueue


class UCS:
    def __init__(self, graph_data):
        self.graph = graph_data


    def ucsAlgo(self, start, goal):
        queue = PriorityQueue()
        queue.put((0, start, [start]))
        visited = set()


        while queue:
            cost, node, path = queue.get()
            if node in visited:
                continue

            visited.add(node)

            if node == goal:
                print("Visited nodes:", path)
                print("Total cost:", cost)
                return

            for child, weight in self.graph[node].items():
                if child not in visited:
                    queue.put((cost + weight, child, path + [child]))

        print("Goal not reachable")

# Graph representation
graph_data = {
    'S': {'A': 1, 'G': 12},
    'A': {'B': 3, 'C': 1},
    'B': {'D': 3},
    'C': {'D': 1, 'G': 2},
    'D': {'G': 3},
    'G': {}
}

ucs = UCS(graph_data)
print("Following is the Uniform Cost Search with cumulative cost:")
ucs.ucsAlgo('S', 'G')
```

INPUT: -

```
# Graph representation
graph_data = {
   'S': {'A': 1, 'G': 12},
   'A': {'B': 3, 'C': 1},
   'B': {'D': 3},
   'C': {'D': 1, 'G': 2},
   'D': {'G': 3},
   'G': {}
}

ucs = UCS(graph_data)
print("Following is the Uniform Cost Search with cumulative cost:")
ucs.ucsAlgo('S', 'G')
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData
_CSE_Sem-6/AI/Assignment/Program_10.py
Following is the Uniform Cost Search with cumulative cost:
Visited nodes: ['S', 'A', 'C', 'G']
Total cost: 4
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program implements the Uniform Cost Search (UCS) algorithm, which explores the graph by expanding the least cumulative cost node first. It ensures finding the optimal path to the goal node while accurately calculating the total cost, demonstrating an effective cost-based search strategy.

PRACTICAL: - 11

AIM: Write a program in Python for A* Search

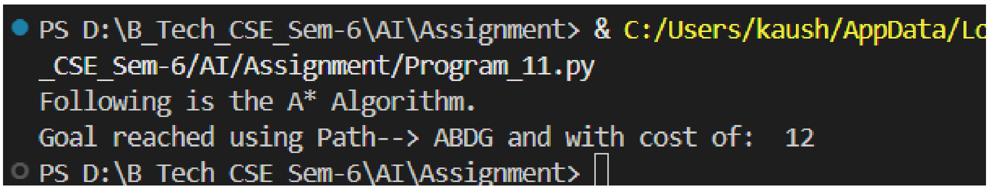
PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 10. - Write a program in Python for A* Search
'''
from queue import PriorityQueue


class GraphAlgorithm:
  def __init__(self, graph):
    self.graph = graph
    self.visited = []


  def astar(self,start,goal):
    pq=[[0,0,0,start]] # Priority queue: (f, h, g, node, path)
    self.visited=[]
    while pq:
      f,h,g,cnode=pq.pop(0)
      self.visited.append([f,h,g,cnode])
      for neigh, wt in self.graph[cnode[-1]].items():
        g1=g+wt[0]
        f1=g1+wt[1]
        path=cnode+neigh
        pq.append([f1,wt[1],g1,path])
        pq=sorted(pq)


    res_visited=[]
    for x in self.visited:
            if x[3].endswith(goal):
                res_visited.append(x)
        return sorted(res_visited)

graph = {
    'A': {'B': [3,8], 'C': [2,9]},
    'B': {'D': [3,7], 'E': [4,6]},
    'C': {'F': [5,4]},
    'D': {'G': [6,0]},
    'E': {'G': [9,0]},
    'F': {'G': [6,0]},
    'G': {}
}
start = 'A'
goal = 'G'

print("Following is the A* Algorithm.")
astar_SearchAlgorithm = GraphAlgorithm(graph)
result=(astar_SearchAlgorithm.astar(start, goal))
print("Goal reached using Path-->", result[0][-1], "and with cost of: ", result[0][-2])
```

INPUT: -

```
graph = {
    'A': {'B': [3,8], 'C': [2,9]},
    'B': {'D': [3,7], 'E': [4,6]},
    'C': {'F': [5,4]},
    'D': {'G': [6,0]},
    'E': {'G': [9,0]},
    'F': {'G': [6,0]},
    'G': {}
}
start = 'A'
goal = 'G'

print("Following is the A* Algorithm.")
astar_SearchAlgorithm = GraphAlgorithm(graph)
result=(astar_SearchAlgorithm.astar(start, goal))
print("Goal reached using Path-->", result[0][-1], "and with cost of: ", result[0][-2])
```

OUTPUT: -



```
● PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Lo
  _CSE_Sem-6/AI/Assignment/Program_11.py
  Following is the A* Algorithm.
  Goal reached using Path--> ABDG and with cost of:  12
○ PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

CONCLUSION: - This program implements the A* Search algorithm, which combines the cost to reach a node (g) and the heuristic estimate to the goal (h) to find the optimal path. It effectively demonstrates heuristic-based pathfinding, ensuring the shortest path to the goal with minimal cost.

PRACTICAL: - 12

AIM: Write a program to implement Depth Limited Search and Iterative Deepening Search on following graph/tree.

```
graph = {
    '6': ['4', '8'],
    '4': ['3', '5'],
    '8': ['9'],
    '3': ['10'],
    '5': ['11'],
    '9': ['12'],
    '10': [],
    '11': [],
    '12': []
}
```

PROGRAM CODE: -

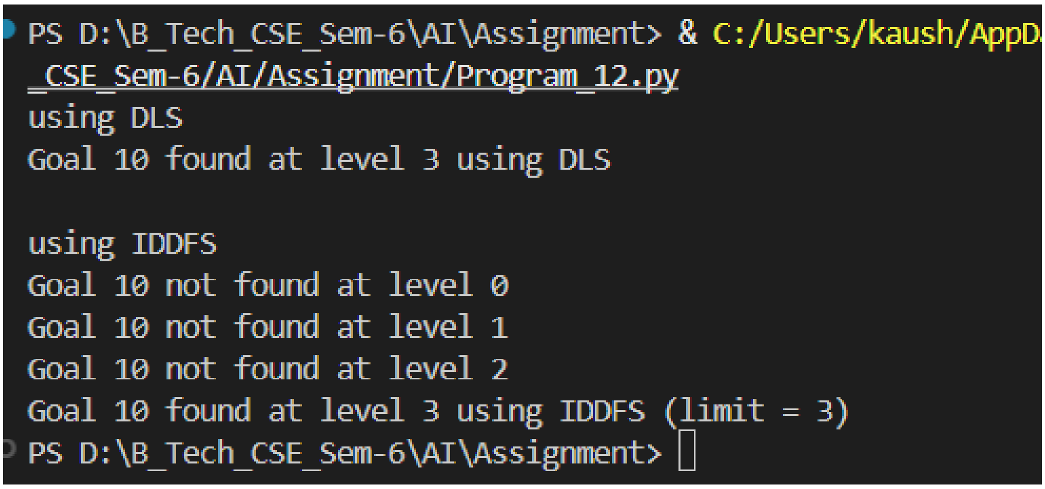
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 12. - Write a program to implement Depth Limited Search and
Iterative Deepening Search on following graph/tree.
'''
class SearchAlgorithms:
    def __init__(self, graph):
        self.graph = graph

    def dls(self, start, goal, limit):
        stack = [(start, 0)]
        while stack:
            node, depth = stack.pop()
            if node == goal:
                return True, goal, depth
            if depth < limit:
                for child in self.graph[node]:
                    stack.insert(0, (child, depth + 1))
        return False, goal, -1
    def iddfs(self, start, goal, max_limit):
        for depth in range(max_limit + 1):
            found, g, d = self.dls(start, goal, depth)
            if found:
                print(f"Goal {g} found at level {d} using IDDFS (limit = {depth})")
                return
            else:
                print(f"Goal {goal} not found at level {depth}")
        print(f"Goal {goal} not found within max depth limit {max_limit}")

# Graph definition
graph_data = {
    '6': ['4', '8'],
    '4': ['3', '5'],
    '8': ['9'],
    '3': ['10'],
    '5': ['11'],
    '9': ['12'],
    '10': [],
    '11': [],
    '12': []
}

search = SearchAlgorithms(graph_data)

print("using DLS")
limit = 3
found, goal, depth = search.dls('6', '10', limit)
```

```
if found:
    print("Goal", goal, "found at level", depth, "using DLS")
else:
    print("Goal", goal, "not found within the depth limit using DLS")


print("\nusing IDDFS")
search.iddfs('6', '10', limit)
```

INPUT: -

```
# Graph definition
graph_data = {
    '6': ['4', '8'],
    '4': ['3', '5'],
    '8': ['9'],
    '3': ['10'],
    '5': ['11'],
    '9': ['12'],
    '10': [],
    '11': [],
    '12': []
}

search = SearchAlgorithms(graph_data)

print("using DLS")
limit = 3
found, goal, depth = search.dls('6', '10', limit)
if found:
    print("Goal", goal, "found at level", depth, "using DLS")
else:
    print("Goal", goal, "not found within the depth limit using DLS")
print("\nusing IDDFS")
search.iddfs('6', '10', limit)
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppD
_CSE_Sem-6/AI/Assignment/Program_12.py
using DLS
Goal 10 found at level 3 using DLS

using IDDFS
Goal 10 not found at level 0
Goal 10 not found at level 1
Goal 10 not found at level 2
Goal 10 found at level 3 using IDDFS (limit = 3)
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program implements Depth-Limited Search (DLS) and Iterative Deepening Depth-First Search (IDDFS) algorithms to traverse a graph/tree. DLS explores nodes up to a specified depth limit, while IDDFS combines the benefits of depth-first and breadth-first search by incrementally increasing the depth limit, ensuring an efficient and complete search strategy.

PRACTICAL: - 13

AIM: Write a program to implement UCS on following graph.

```
graph = {
    'A': {'B': 1, 'C': 2},
    'B': {'D': 3, 'E': 4},
    'C': {'F': 5},
    'D': {'G': 6},
    'E': {'G': 7},
    'F': {'G': 8},
    'G': {}
}
start = 'A'
goal = 'G'
```

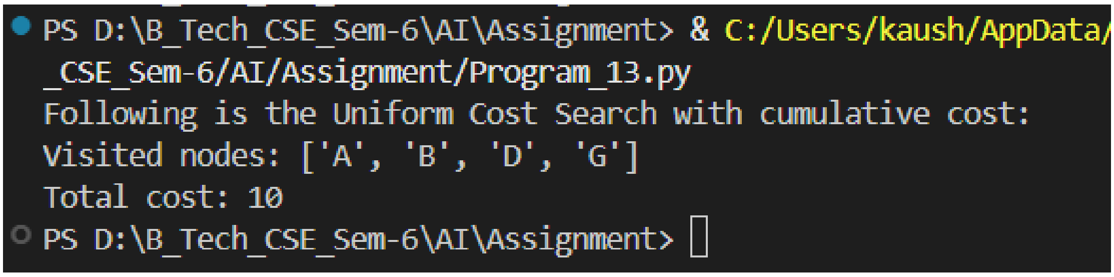
PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 13. - Write a program to implement UCS on following graph.
'''
from queue import PriorityQueue


class UCS:
    def __init__(self, graph_data):
        self.graph = graph_data


    def ucsAlgo(self, start, goal):
        queue = PriorityQueue()
        queue.put((0, start, [start]))
        visited = set()


        while queue:
            cost, node, path = queue.get()


            if node in visited:
                continue

            visited.add(node)

            if node == goal:
                print("Visited nodes:", path)
                print("Total cost:", cost)
                return

            for child, weight in self.graph[node].items():
                if child not in visited:
                    queue.put((cost + weight, child, path + [child]))

        print("Goal not reachable")

graph_data = {
    'A': {'B': 1, 'C': 2},
    'B': {'D': 3, 'E': 4},
    'C': {'F': 5},
    'D': {'G': 6},
    'E': {'G': 7},
    'F': {'G': 8},
    'G': {}
}

ucs = UCS(graph_data)
print("Following is the Uniform Cost Search with cumulative cost:")
ucs.ucsAlgo('A', 'G')
```

INPUT: -

```
graph_data = {
    'A': {'B': 1, 'C': 2},
    'B': {'D': 3, 'E': 4},
    'C': {'F': 5},
    'D': {'G': 6},
    'E': {'G': 7},
    'F': {'G': 8},
    'G': {}
}

ucs = UCS(graph_data)
print("Following is the Uniform Cost Search with cumulative cost:")
ucs.ucsAlgo('A', 'G')
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/
_CSE_Sem-6/AI/Assignment/Program_13.py
Following is the Uniform Cost Search with cumulative cost:
Visited nodes: ['A', 'B', 'D', 'G']
Total cost: 10
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

CONCLUSION: - This program implements the Uniform Cost Search (UCS) algorithm to find the optimal path from a start node to a goal node in a weighted graph. It ensures that the path with the least cumulative cost is selected, demonstrating an efficient and complete cost-based search strategy.

PRACTICAL: - 14

AIM: Write a program to implement Best First Search on following graph.

```
graph = {
'A':{'B':12, 'C':14}, #heuristic value A to H is 13, B to H is 12, C to H is 14
'B':{'D':11, 'E':10},
'C':{'F':6, 'G':7},
'D':{'H':0},
'E':{'H':0},
'F':{'H':0},
'G':{'H':0}
}
```

PROGRAM CODE: -

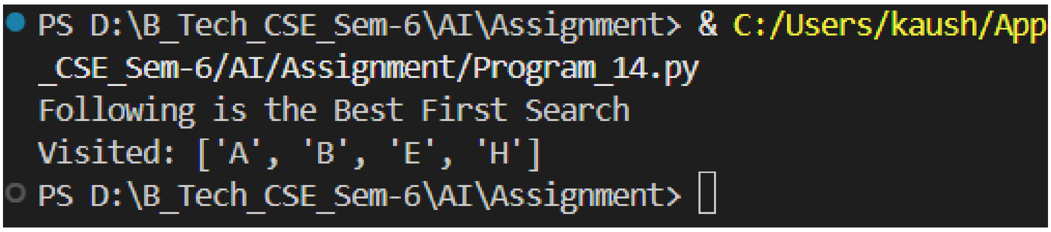
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 14. - Write a program to implement Best First Search on following graph.
'''
from queue import PriorityQueue


class Best:
    def __init__(self, graph):
        self.graph_data = graph
        self.visited = []


    def bestf(self, node, goal):
        self.visited.append(node)
        while True:
            tn = node
            if tn == goal:
                break
            queue = PriorityQueue()
            for neighbour, weight in self.graph_data[tn].items():
                queue.put([weight, neighbour])
```

```
          tw, tn = queue.queue[0]
          self.visited.append(tn)
          node = tn
      print("Visited:", self.visited)

graph = {
'A':{'B':12, 'C':14}, #heuristic value A to H is 13, B to H is 12, C to H is 14
'B':{'D':11, 'E':10},
'C':{'F':6, 'G':7},
'D':{'H':0},
'E':{'H':0},
'F':{'H':0},
'G':{'H':0}
}

graph = Best(graph)

print("Following is the Best First Search")
graph.bestf('A', 'H')
```

INPUT: -

```
graph = {

'A':{'B':12, 'C':14}, #heuristic value A to H is 13, B to H is 12, C to H is 14

'B':{'D':11, 'E':10},

'C':{'F':6, 'G':7},

'D':{'H':0},

'E':{'H':0},
'F':{'H':0},

'G':{'H':0}

}



graph = Best(graph)



print("Following is the Best First Search")

graph.bestf('A', 'H')
```

OUTPUT: -



```
● PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/App
  _CSE_Sem-6/AI/Assignment/Program_14.py
  Following is the Best First Search
  Visited: ['A', 'B', 'E', 'H']
○ PS D:\B_Tech_CSE_Sem-6\AI\Assignment> ▯
```

CONCLUSION: - This program implements the Best First Search algorithm, which uses a heuristic approach to explore the graph by selecting the node with the lowest heuristic value at each step. It efficiently finds the goal node while minimizing the search effort, demonstrating the effectiveness of heuristic-based search techniques.

PRACTICAL: - 15

AIM: Write a program to implement A* search on following graph.

```
graph = {
    'A': {'B': [3,8], 'C': [2,9]},
    'B': {'D': [3,7], 'E': [4,6]},
    'C': {'F': [5,4]},
    'D': {'G': [6,0]},
    'E': {'G': [9,0]},
    'F': {'G': [6,0]},
    'G': {}
}
start = 'A'
goal = 'G'
```

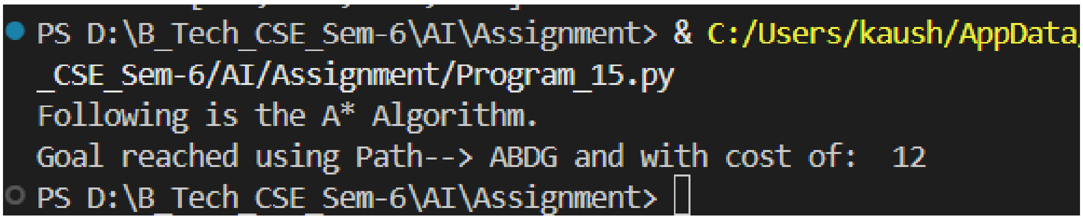
PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 15. - Write a program to implement A* search on following graph.
'''
from queue import PriorityQueue


class GraphAlgorithm:
    def __init__(self, graph):
        self.graph = graph
        self.visited = []


    def astar(self,start,goal):
        pq=[[0,0,0,start]] # Priority queue: (f, h, g, node, path)
        self.visited=[]
        while pq:
            f,h,g,cnode=pq.pop(0)
            self.visited.append([f,h,g,cnode])
            for neigh, wt in self.graph[cnode[-1]].items():
                g1=g+wt[0]
                f1=g1+wt[1]
                path=cnode+neigh
                pq.append([f1,wt[1],g1,path])
                pq=sorted(pq)

        res_visited=[]
        for x in self.visited:
            if x[3].endswith(goal):
                res_visited.append(x)
        return sorted(res_visited)

graph = {
    'A': {'B': [3,8], 'C': [2,9]},
    'B': {'D': [3,7], 'E': [4,6]},
    'C': {'F': [5,4]},
    'D': {'G': [6,0]},
    'E': {'G': [9,0]},
    'F': {'G': [6,0]},
    'G': {}
}
start = 'A'
goal = 'G'

print("Following is the A* Algorithm.")
astar_SearchAlgorithm = GraphAlgorithm(graph)
result=(astar_SearchAlgorithm.astar(start, goal))
print("Goal reached using Path-->", result[0][-1], "and with cost of: ", result[0][-2])
```

INPUT: -

```
graph = {
    'A': {'B': [3,8], 'C': [2,9]},
    'B': {'D': [3,7], 'E': [4,6]},
    'C': {'F': [5,4]},
    'D': {'G': [6,0]},
    'E': {'G': [9,0]},
    'F': {'G': [6,0]},
    'G': {}
}
start = 'A'
goal = 'G'

print("Following is the A* Algorithm.")
astar_SearchAlgorithm = GraphAlgorithm(graph)
result=(astar_SearchAlgorithm.astar(start, goal))
print("Goal reached using Path-->", result[0][-1], "and with cost of: ", result[0][-2])
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData
_CSE_Sem-6/AI/Assignment/Program_15.py
Following is the A* Algorithm.
Goal reached using Path--> ABDG and with cost of:  12
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program implements the A* Search algorithm, which combines the actual cost to reach a node (g) and the heuristic estimate to the goal (h) to find the optimal path. It effectively demonstrates heuristic-based pathfinding, ensuring the shortest and most cost-efficient path to the goal.

PRACTICAL: - 16

AIM: Write a program to implement A* search on the following graph.

Note : Structure of data : “Key Name “ : {“City Name”: Latitude, Longitude, Heuristic value}

```
graph = {
'START': {'Jammu': [32.7266,74.8570,1600]},
'Jammu': {'Amritsar': [31.6339, 74.8722,1400], 'Delhi': [28.7040,77.1024,1300]},
'Amritsar': {'Sri-Gangaganar': [29.9094,73.8800,1340], 'Jodhpur': [26.2389,73.0243, 1230]},
'Delhi': {'Jaipur': [26.9124, 75.7873,1000],'Gwalior':[26.2124, 78.1772,1100]},
'Sri-Gangaganar': {'Udaipur': [24.5854, 73.7125,400]},
'Jodhpur': {'Himmatnagar': [23.5969, 72.9630,300]},
'Jaipur': {'Kota': [25.2138, 75.8648,300]},
'Gwalior': {'Ratlam': [23.3315,75.0367,250]},
'Udaipur': {'Vadodara': [22.3072,73.1812,0]},
'Himmatnagar': {'Vadodara': [22.3072,73.1812,0]},
'Kota': {'Vadodara': [22.3072,73.1812,0]},
'Ratlam': {'Vadodara': [22.3072,73.1812,0]},
}
start = 'Jammu'
goal = 'Vadodara'
```

HINT: -

```
#pip install geopy
import geopy.distance
coords_1 = (22.3071, 73.1812) #Vadodara
coords_2 = (23.0225, 72.5713) #Ahmedabad
print ("distance in kms is ", geopy.distance.geodesic(coords_1, coords_2).km)
```

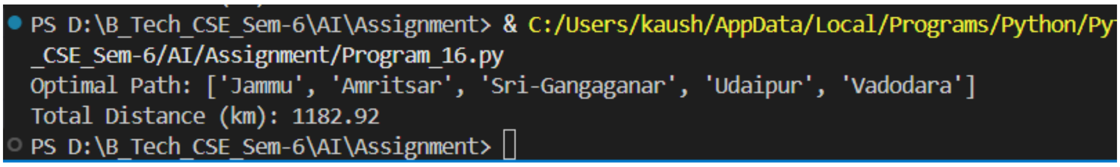
PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 16. - Write a program to implement A* search on the following graph.
'''
from queue import PriorityQueue
from geopy.distance import geodesic


raw_graph = {
    'START': {'Jammu': [32.7266,74.8570,1600]},
    'Jammu': {'Amritsar': [31.6339, 74.8722,1400], 'Delhi': [28.7040,77.1024,1300]},
    'Amritsar': {'Sri-Gangaganar': [29.9094,73.8800,1340], 'Jodhpur': [26.2389,73.0243,1230]},
    'Delhi': {'Jaipur': [26.9124, 75.7873,1000], 'Gwalior': [26.2124, 78.1772,1100]},
    'Sri-Gangaganar': {'Udaipur': [24.5854, 73.7125,400]},
    'Jodhpur': {'Himmatnagar': [23.5969, 72.9630,300]},
    'Jaipur': {'Kota': [25.2138, 75.8648,300]},
    'Gwalior': {'Ratlam': [23.3315,75.0367,250]},
    'Udaipur': {'Vadodara': [22.3072,73.1812,0]},
    'Himmatnagar': {'Vadodara': [22.3072,73.1812,0]},
    'Kota': {'Vadodara': [22.3072,73.1812,0]},
    'Ratlam': {'Vadodara': [22.3072,73.1812,0]},
}


def build_graph(raw_graph):
    graph = {}
    coords = {}

    for node in raw_graph:
        if node not in coords and node != 'START':
            continue
        graph[node] = {}
        for neighbor, (lat, lon, heuristic) in raw_graph[node].items():
            coords[neighbor] = (lat, lon)
            if node == 'START':
                coords['START'] = (0, 0)
                distance = 0
            else:
                distance = geodesic(coords[node], (lat, lon)).km
            graph[node][neighbor] = [distance, heuristic]
    return graph

def a_star_search(graph, start, goal):
    pq = PriorityQueue()  # (f, h, g, node, path)
    pq.put((0, 0, 0, start, [start]))
    visited = {}

    while not pq.empty():
        f, h, g, current, path = pq.get()

        if current in visited and visited[current] <= g:
            continue

        visited[current] = g

        if current == goal:
            return path, round(g, 2)
        for neighbor, (cost, heuristic) in graph.get(current, {}).items():
            new_g = g + cost
            new_f = new_g + heuristic
            pq.put((new_f, heuristic, new_g, neighbor, path + [neighbor]))

    return None, float('inf')

graph = build_graph(raw_graph)
start = 'Jammu'
goal = 'Vadodara'

path, cost = a_star_search(graph, start, goal)

print("Optimal Path:", path)
print("Total Distance (km):", cost)
```

INPUT: -

```
raw_graph = {
    'START': {'Jammu': [32.7266,74.8570,1600]},
    'Jammu': {'Amritsar': [31.6339, 74.8722,1400], 'Delhi': [28.7040,77.1024,1300]},
    'Amritsar': {'Sri-Gangaganar': [29.9094,73.8800,1340], 'Jodhpur':
[26.2389,73.0243,1230]},
    'Delhi': {'Jaipur': [26.9124, 75.7873,1000], 'Gwalior': [26.2124, 78.1772,1100]},
    'Sri-Gangaganar': {'Udaipur': [24.5854, 73.7125,400]},
    'Jodhpur': {'Himmatnagar': [23.5969, 72.9630,300]},
    'Jaipur': {'Kota': [25.2138, 75.8648,300]},
    'Gwalior': {'Ratlam': [23.3315,75.0367,250]},
    'Udaipur': {'Vadodara': [22.3072,73.1812,0]},
    'Himmatnagar': {'Vadodara': [22.3072,73.1812,0]},
    'Kota': {'Vadodara': [22.3072,73.1812,0]},
    'Ratlam': {'Vadodara': [22.3072,73.1812,0]},
}
```

OUTPUT: -

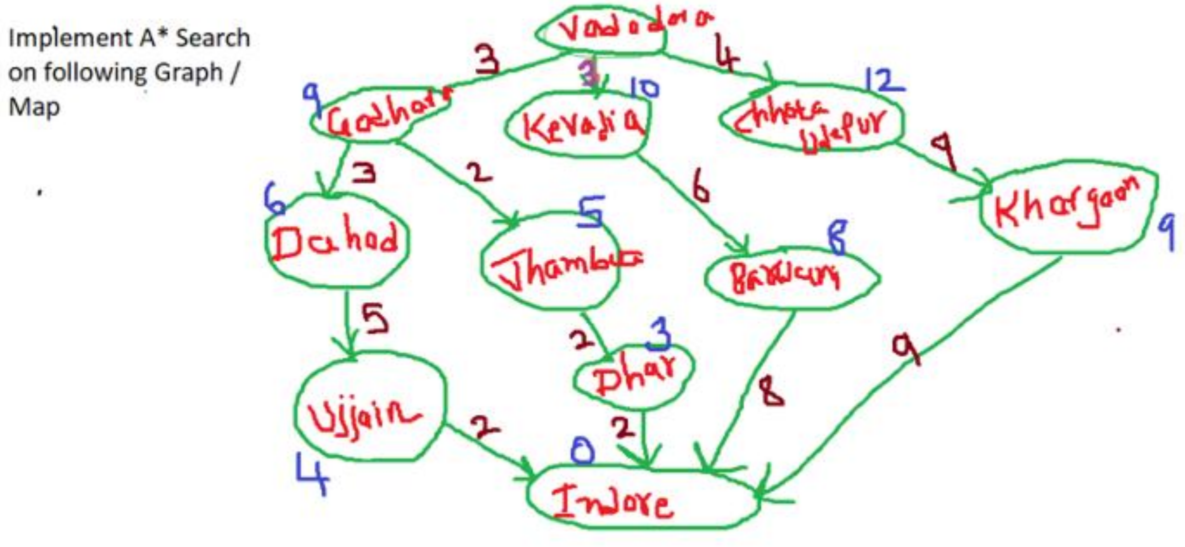


```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Py
_CSE_Sem-6/AI/Assignment/Program_16.py
Optimal Path: ['Jammu', 'Amritsar', 'Sri-Gangaganar', 'Udaipur', 'Vadodara']
Total Distance (km): 1182.92
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

CONCLUSION: - This program implements the A* Search algorithm to find the optimal path between two locations on a graph, considering both actual distances and heuristic estimates. It effectively demonstrates the use of geodesic distances and heuristics to calculate the shortest and most cost-efficient path, ensuring accurate and practical route optimization.

PRACTICAL: - 17

AIM: Implement the following.



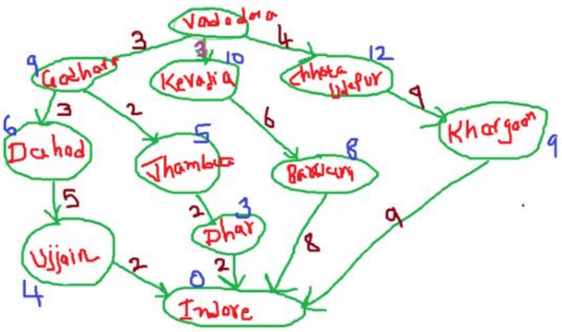
Implement A* Search on following Graph / Map

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 17. - implement A* search on the following graph.
'''
from queue import PriorityQueue


class GraphAlgorithm:
    def __init__(self, graph):
        self.graph = graph
        self.visited = []


    def astar(self, start, goal):
        # pq entries are [f = g+h, h, g, path_list]
        pq = [[0, 0, 0, [start]]]
        self.visited = []

        while pq:
            f, h, g, path = pq.pop(0)
            node = path[-1]
            self.visited.append([f, h, g, path])

            if node == goal:
                break

            for neigh, (cost, h_neigh) in self.graph[node].items():
                g2 = g + cost
                f2 = g2 + h_neigh
                new_path = path + [neigh]
                pq.append([f2, h_neigh, g2, new_path])

            pq.sort(key=lambda x: x[0])

        # filter only those that actually reached the goal
        finals = [v for v in self.visited if v[3][-1] == goal]
        return sorted(finals, key=lambda x: x[0])

graph = {
```

```
'Vadodara': {
    'Godhra':       [3, 9],
    'Kevadia':      [3,10],
    'ChhotaUdepur': [4,12],
},
'Godhra': {
    'Vadodara': [3, 0],
    'Kevadia':  [9,10],
    'Dahod':    [6, 6],
},
'Kevadia': {
    'Vadodara': [3, 0],
    'Godhra':   [9, 9],
    'Thandla':  [2, 5],
},
'ChhotaUdepur': {
    'Vadodara': [4, 0],
    'Khargone': [9, 9],
},
'Khargone': {
    'ChhotaUdepur': [9,12],
    'Barwaha':      [9, 8],
},
'Barwaha': {
    'Khargone': [9, 9],
    'Indore':   [8, 0],
},
'Dahod': {
    'Godhra': [6, 9],
    'Ujjain': [5, 4],
    'Thandla':[3, 5],
},
'Ujjain': {
    'Dahod':   [5, 6],
    'Indore':  [4, 0],
},
'Thandla': {
    'Kevadia': [2,10],
    'Dahod':   [3, 6],
    'Dhar':    [3, 3],
},
'Dhar': {
    'Thandla': [3, 5],
    'Indore':  [2, 0],
},
'Indore': {
        'Ujjain': [4, 4],
        'Barwaha':[8, 8],
        'Dhar':   [2, 3],
    }
}

start = 'Vadodara'
goal  = 'Indore'

astar = GraphAlgorithm(graph)
results = astar.astar(start, goal)

if results:
    best = results[0]
    path_list = best[3]
    cost = best[2]
    print("Best path:", " -> ".join(path_list))
    print("Total cost:", cost)
else:
    print("No path found!")
```

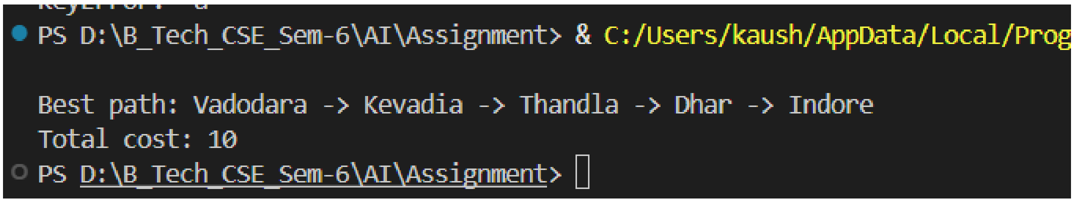
INPUT: -



start = 'Vadodara'

goal = 'Indore'

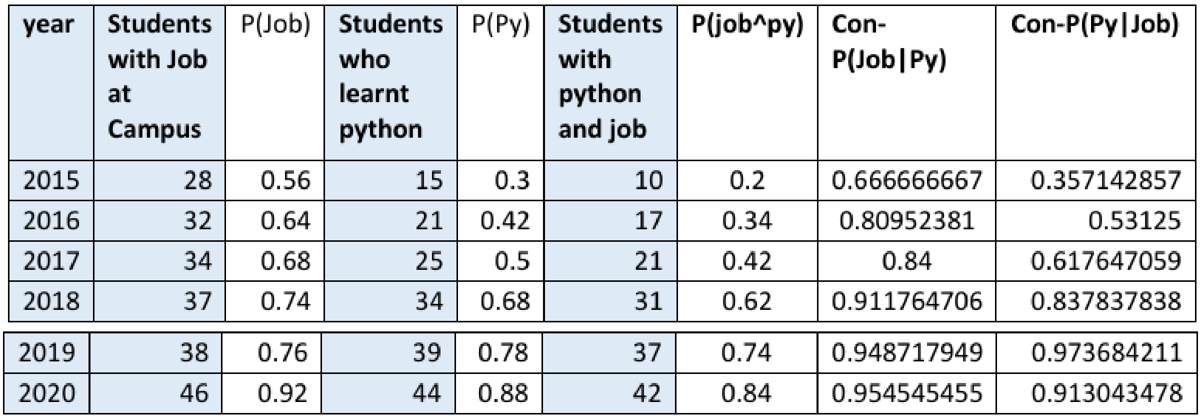
OUTPUT: -


```
KeyError: 'a'
● PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Prog

  Best path: Vadodara -> Kevadia -> Thandla -> Dhar -> Indore
  Total cost: 10
○ PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

CONCLUSION: - This program implements the A* search algorithm to find the optimal path between two nodes in a weighted graph. It combines the actual cost (g) and heuristic cost (h) to prioritize nodes, ensuring an efficient and accurate search for the shortest path. The program successfully demonstrates the application of A* search for pathfinding tasks.

PRACTICAL: - 18

AIM: Write a program in Python for calculating conditional probability for following data in CSV file. The input columns are light blue coloured, remaining are calculative.

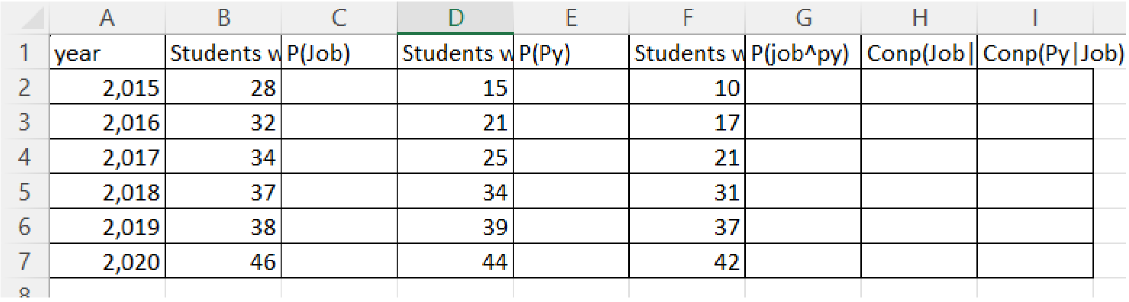


year	Students with Job at Campus	P(Job)	Students who learnt python	P(Py)	Students with python and job	P(job^py)	Con-P(Job|Py)	Con-P(Py|Job)
2015	28	0.56	15	0.3	10	0.2	0.666666667	0.357142857
2016	32	0.64	21	0.42	17	0.34	0.80952381	0.53125
2017	34	0.68	25	0.5	21	0.42	0.84	0.617647059
2018	37	0.74	34	0.68	31	0.62	0.911764706	0.837837838
2019	38	0.76	39	0.78	37	0.74	0.948717949	0.973684211
2020	46	0.92	44	0.88	42	0.84	0.954545455	0.913043478

PROGRAM CODE: -

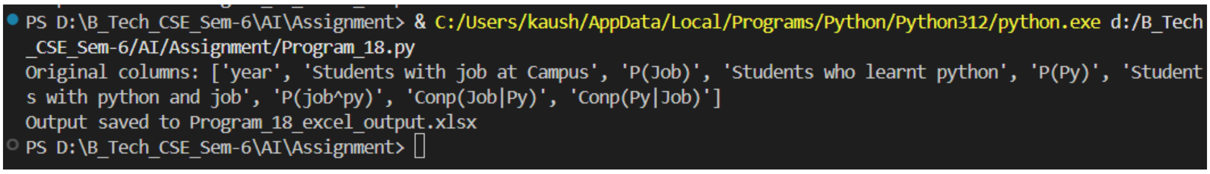
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 18. - Write a program in Python for calculating conditional probability
for following data in CSV
file. The input columns are light blue coloured, remaining are calculative.
'''
import pandas as pd

# Step 1: Load Excel file
df = pd.read_excel("Program_18_excel.xlsx")  # replace with your file name

# Step 2: Debug - Show column names
print("Original columns:", df.columns.tolist())

# Step 3: Rename columns safely
df = df.rename(columns={
    df.columns[1]: "Job",
    df.columns[3]: "Python",
    df.columns[5]: "Both"
})

# Step 4: Total number of students (as per your table structure)
total_students = 50

# Step 5: Perform calculations
df["P(Job)"] = df["Job"] / total_students
df["P(Py)"] = df["Python"] / total_students
df["P(job^py)"] = df["Both"] / total_students
df["Conp(Py|Job)"] = df["P(job^py)"] / df["P(Job)"]

# Step 6: Save output to Excel
output_file = "Program_18_excel_output.xlsx"
df.to_excel(output_file, index=False)

print(f"Output saved to {output_file}")
```

INPUT: -

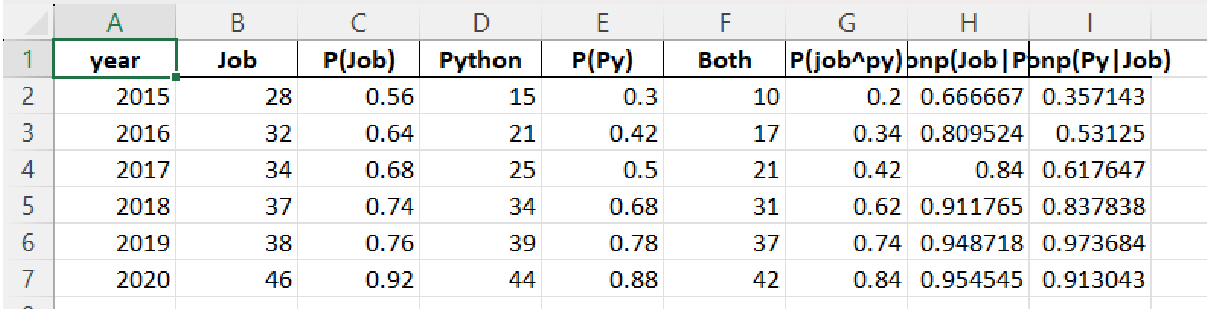


	A	B	C	D	E	F	G	H	I
1	year	Students w	P(Job)	Students w	P(Py)	Students w	P(job^py)	Conp(Job|	Conp(Py|Job)
2	2,015	28		15		10			
3	2,016	32		21		17			
4	2,017	34		25		21			
5	2,018	37		34		31			
6	2,019	38		39		37			
7	2,020	46		44		42			

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/python.exe d:/B_Tech
_CSE_Sem-6/AI/Assignment/Program_18.py
Original columns: ['year', 'Students with job at Campus', 'P(Job)', 'Students who learnt python', 'P(Py)', 'Student
s with python and job', 'P(job^py)', 'Conp(Job|Py)', 'Conp(Py|Job)']
Output saved to Program_18_excel_output.xlsx
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

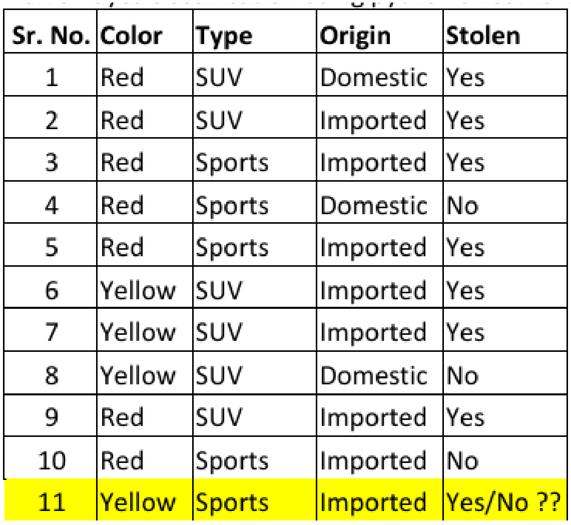


	A	B	C	D	E	F	G	H	I
1	year	Job	P(Job)	Python	P(Py)	Both	P(job^py)	onp(Job|P	onp(Py|Job)
2	2015	28	0.56	15	0.3	10	0.2	0.666667	0.357143
3	2016	32	0.64	21	0.42	17	0.34	0.809524	0.53125
4	2017	34	0.68	25	0.5	21	0.42	0.84	0.617647
5	2018	37	0.74	34	0.68	31	0.62	0.911765	0.837838
6	2019	38	0.76	39	0.78	37	0.74	0.948718	0.973684
7	2020	46	0.92	44	0.88	42	0.84	0.954545	0.913043

CONCLUSION: - This program calculates conditional probabilities from a given dataset in an Excel file. It demonstrates the use of pandas for data manipulation, including renaming columns, performing probability calculations, and saving the results to a new Excel file. The program effectively automates the computation of conditional probabilities, ensuring accuracy and efficiency.

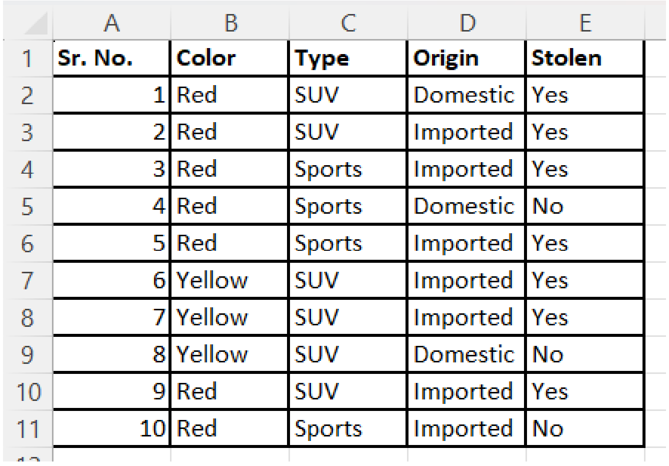
PRACTICAL: - 19

AIM: Naive Bayes classification from scratch using Excel for below given tabular data.



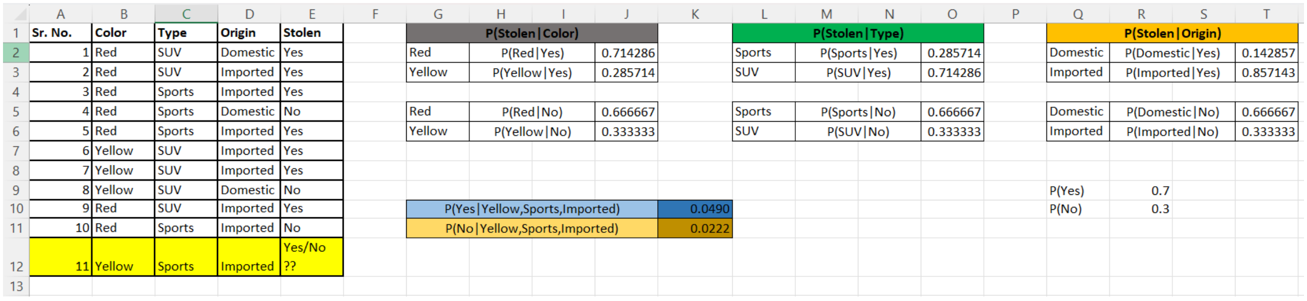
Sr. No.	Color	Type	Origin	Stolen
1	Red	SUV	Domestic	Yes
2	Red	SUV	Imported	Yes
3	Red	Sports	Imported	Yes
4	Red	Sports	Domestic	No
5	Red	Sports	Imported	Yes
6	Yellow	SUV	Imported	Yes
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	Yes
10	Red	Sports	Imported	No
11	Yellow	Sports	Imported	Yes/No ??

INPUT: -



	A	B	C	D	E
1	Sr. No.	Color	Type	Origin	Stolen
2	1	Red	SUV	Domestic	Yes
3	2	Red	SUV	Imported	Yes
4	3	Red	Sports	Imported	Yes
5	4	Red	Sports	Domestic	No
6	5	Red	Sports	Imported	Yes
7	6	Yellow	SUV	Imported	Yes
8	7	Yellow	SUV	Imported	Yes
9	8	Yellow	SUV	Domestic	No
10	9	Red	SUV	Imported	Yes
11	10	Red	Sports	Imported	No

OUTPUT: -

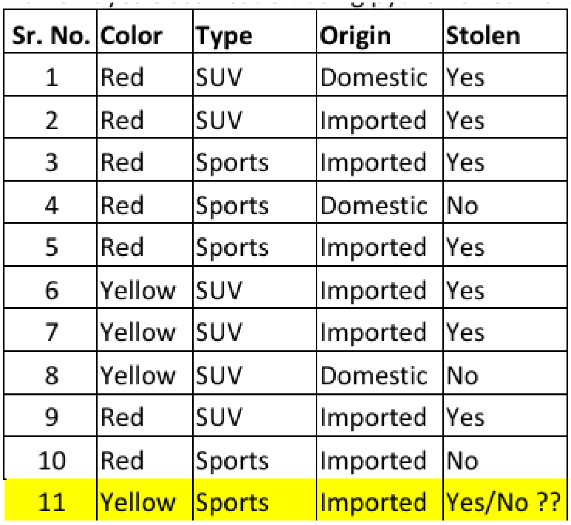


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	Sr. No.	Color	Type	Origin	Stolen		P(Stolen|Color)					P(Stolen|Type)					P(Stolen|Origin)			
2	1	Red	SUV	Domestic	Yes		Red	P(Red|Yes)		0.714286		Sports	P(Sports|Yes)		0.285714		Domestic	P(Domestic|Yes)		0.142857
3	2	Red	SUV	Imported	Yes		Yellow	P(Yellow|Yes)		0.285714		SUV	P(SUV|Yes)		0.714286		Imported	P(Imported|Yes)		0.857143
4	3	Red	Sports	Imported	Yes															
5	4	Red	Sports	Domestic	No		Red	P(Red|No)		0.666667		Sports	P(Sports|No)		0.666667		Domestic	P(Domestic|No)		0.666667
6	5	Red	Sports	Imported	Yes		Yellow	P(Yellow|No)		0.333333		SUV	P(SUV|No)		0.333333		Imported	P(Imported|No)		0.333333
7	6	Yellow	SUV	Imported	Yes															
8	7	Yellow	SUV	Imported	Yes															
9	8	Yellow	SUV	Domestic	No												P(Yes)	0.7		
10	9	Red	SUV	Imported	Yes		P(Yes|Yellow,Sports,Imported)				0.0490						P(No)	0.3		
11	10	Red	Sports	Imported	No		P(No|Yellow,Sports,Imported)				0.0222									
12	11	Yellow	Sports	Imported	Yes/No ??															
13																				

CONCLUSION: - Since P(Yes | Yellow, Sports, Imported) = 0.0490 is greater than P(No | Yellow, Sports, Imported) = 0.0222, the car is more likely to be Stolen.

PRACTICAL: - 20

AIM: Naive Bayes classification using python sklearns lib for below given tabular data.



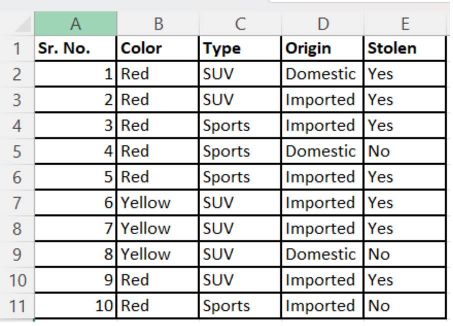
Sr. No.	Color	Type	Origin	Stolen
1	Red	SUV	Domestic	Yes
2	Red	SUV	Imported	Yes
3	Red	Sports	Imported	Yes
4	Red	Sports	Domestic	No
5	Red	Sports	Imported	Yes
6	Yellow	SUV	Imported	Yes
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	Yes
10	Red	Sports	Imported	No
11	Yellow	Sports	Imported	Yes/No ??

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 20. -  Naive Bayes classification using python sklearns lib for below given tabular data.
'''
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.naive_bayes import BernoulliNB


df = pd.read_excel("Program_20_excel.xlsx")


X = df.iloc[:, 1:-1]  # Color, Type, Origin)
y = df.iloc[:, -1]   # (Stolen)


# Encoding categorical variables
le_color = preprocessing.LabelEncoder()
```

```
le_type = preprocessing.LabelEncoder()
le_origin = preprocessing.LabelEncoder()
le_stolen = preprocessing.LabelEncoder()

X['Color'] = le_color.fit_transform(X['Color'])
X['Type'] = le_type.fit_transform(X['Type'])
X['Origin'] = le_origin.fit_transform(X['Origin'])
y = le_stolen.fit_transform(y)

features = np.array(list(zip(X['Color'], X['Type'], X['Origin'])))

# Train the model
model = BernoulliNB()
model.fit(features, y)

# Test the model with a sample input
test_data = np.array([['Yellow', 'Sports', 'Imported']])
test_data[:, 0] = le_color.fit_transform(test_data[:, 0])
test_data[:, 1] = le_type.fit_transform(test_data[:, 1])
test_data[:, 2] = le_origin.fit_transform(test_data[:, 2])
test_data = test_data.astype(int)

# Predict the outcome
predicted = model.predict(test_data)

if predicted[0] == 0:
    print("Car is not stolen")
else:
    print("Car is stolen")
```

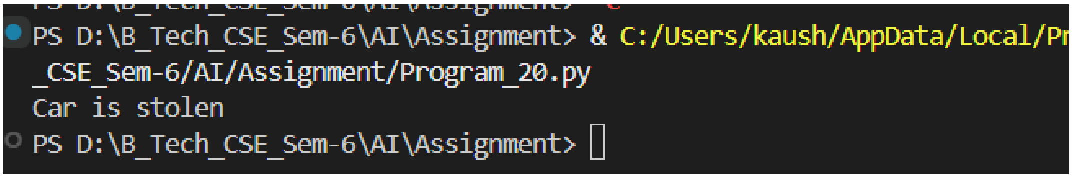
INPUT: -



Sr. No.	Color	Type	Origin	Stolen
1	Red	SUV	Domestic	Yes
2	Red	SUV	Imported	Yes
3	Red	Sports	Imported	Yes
4	Red	Sports	Domestic	No
5	Red	Sports	Imported	Yes
6	Yellow	SUV	Imported	Yes
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	Yes
10	Red	Sports	Imported	No

```
test_data = np.array([['Yellow', 'Sports', 'Imported']])
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Pr
_CSE_Sem-6/AI/Assignment/Program_20.py
Car is stolen
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

CONCLUSION: - This program implements Naive Bayes classification using Python's sklearn library to predict whether a car is stolen based on its attributes (Color, Type, and Origin). It demonstrates the use of label encoding for categorical data, model training with BernoulliNB, and prediction on new data, showcasing the effectiveness of Naive Bayes for classification tasks.

PRACTICAL: - 21

AIM: Create a model to predict next word conditional probability-based prediction model for Gujarati language (Download gujarati text from sources available on internet)

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 21. - Create a model to predict next word conditional probability-based prediction model for
Gujarati language (Download Gujarati text from sources available on the internet)
'''


import nltk
from nltk.util import ngrams
from collections import Counter, defaultdict
import random


with open('Program_21.txt', 'r', encoding='utf-8') as file:
    text = file.read()


tokens = nltk.word_tokenize(text)


bigrams = list(ngrams(tokens, 2))


bigram_counts = Counter(bigrams)
word_counts = Counter(tokens)


conditional_probabilities = defaultdict(dict)
for (w1, w2), count in bigram_counts.items():
    conditional_probabilities[w1][w2] = count / word_counts[w1]
```

```
def predict_next_word(word, conditional_probabilities):
    if word in conditional_probabilities:
        next_words = conditional_probabilities[word]
        predicted_word = max(next_words, key=next_words.get)
        return predicted_word, next_words[predicted_word]
    else:
        return None, None


input_word = input("Enter a Gujarati word: ")
predicted_word, probability = predict_next_word(input_word, conditional_probabilities)


if predicted_word:
    print(f"The predicted next word is: {predicted_word}")
    print(f"Conditional probability of '{predicted_word}' given '{input_word}': {probability}")
else:
    print("No prediction available for the given word.")
```

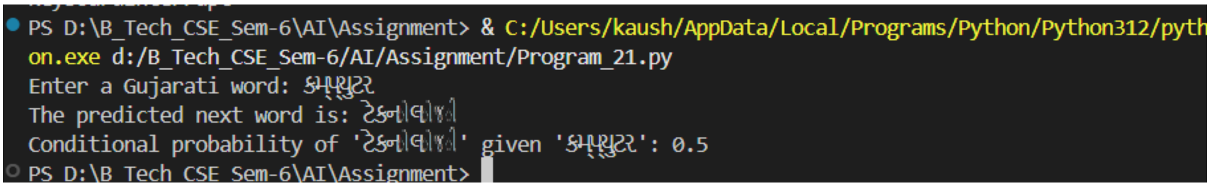
INPUT: -

અજકાલની દ્રુતગતિની દુનિયામાં કમ્પ્યુટર ટેકનોલોજી આપણા જીવનનો અભિન્ન ભાગ બની ગઈ છે. શિક્ષણ, આરોગ્ય, વ્યવસાય, મનોરંજન અને સરકારશાહી ક્ષેત્રોમાં કમ્પ્યુટરની ભૂમિકા ખુબજ મહત્વપૂર્ણ બની ગઈ છે. ઈન્ટરનેટના સાધનથી માહિતી સરળતાથી મેળવી શકાય છે અને લોકો દુનિયાની કોઈપણ ખૂણામાં બેઠા-બેઠા વ્યવહારો કરી શકે છે. કૃત્રિમ બુધ્દ્ધિ (AI), ક્લાઉડ કમ્પ્યુટિંગ, અને સાયબર સુરક્ષા જેવી નવી ટેકનોલોજીઓ કમ્પ્યુટર ક્ષેત્રમાં ક્રાંતિ લાવી રહી છે, જેનાથી માનવ જીવન વધુ સરળ અને વ્યવસ્થિત બની રહ્યું છે.અજકાલની દ્રુતગતિની દુનિયામાં કમ્પ્યુટર ટેકનોલોજી આપણા જીવનનો અભિન્ન ભાગ બની ગઈ છે. શિક્ષણ, આરોગ્ય, વ્યવસાય, મનોરંજન અને સરકારશાહી ક્ષેત્રોમાં કમ્પ્યુટરની ભૂમિકા ખુબજ મહત્વપૂર્ણ બની ગઈ છે. ઈન્ટરનેટના સાધનથી માહિતી સરળતાથી મેળવી શકાય છે અને લોકો દુનિયાની કોઈપણ ખૂણામાં બેઠા-બેઠા વ્યવહારો કરી શકે છે. કૃત્રિમ બુધ્દ્ધિ (AI), ક્લાઉડ કમ્પ્યુટિંગ, અને સાયબર સુરક્ષા જેવી નવી ટેકનોલોજીઓ કમ્પ્યુટર ક્ષેત્રમાં ક્રાંતિ લાવી રહી છે, જેનાથી માનવ જીવન વધુ સરળ અને વ્યવસ્થિત બની રહ્યું છે.અજકાલની દ્રુતગતિની દુનિયામાં કમ્પ્યુટર ટેકનોલોજી આપણા જીવનનો અભિન્ન ભાગ બની ગઈ છે. શિક્ષણ, આરોગ્ય, વ્યવસાય, મનોરંજન અને સરકારશાહી ક્ષેત્રોમાં કમ્પ્યુટરની ભૂમિકા ખુબજ

મહત્વપૂર્ણ બની ગઈ છે. ઈન્ટરનેટના સાધનથી માહિતી સરળતાથી મેળવી શકાય છે અને લોકો દુનિયાની કોઈપણ ખૂણામાં બેઠા-બેઠા વ્યવહારો કરી શકે છે. કૃત્રિમ બુદ્ધિ (AI), ક્લાઉડ કમ્પ્યુટિંગ, અને સાયબર સુરક્ષા જેવી નવી ટેકનોલોજીઓ કમ્પ્યુટર ક્ષેત્રમાં ક્રાંતિ લાવી રહી છે, જેનાથી માનવ જીવન વધુ સરળ અને વ્યવસ્થિત બની રહ્યું છે.

Enter a Gujarati word: કમ્પ્યુટર

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/pyth
on.exe d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_21.py
Enter a Gujarati word: કમ્પ્યુટર
The predicted next word is: ટેક્નોલોજી
Conditional probability of 'ટેક્નોલોજી' given 'કમ્પ્યુટર': 0.5
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program creates a conditional probability-based next-word prediction model for the Gujarati language using bigrams. It effectively calculates the likelihood of the next word based on the input word and provides predictions along with their probabilities, demonstrating the application of natural language processing techniques for Gujarati text.

PRACTICAL: - 22

AIM: Create a model to predict whether a person will have car or not based on dataset attached using Naive Bayes Classifier. (user_data_cars_1.csv)

1. Calculate Entropy and Gini for following dataset in Excel. (playplaynot.csv)
2. Write a python script to implement Decision Tree classifier on same dataset. (playplaynot.csv)
3. Write python script to implement Random Forest classifier on following dataset. (iris.csv)

Attachment playplaynot.csv, iris.csv, ML Observation Table.docx

PROGRAM CODE (22.1): -

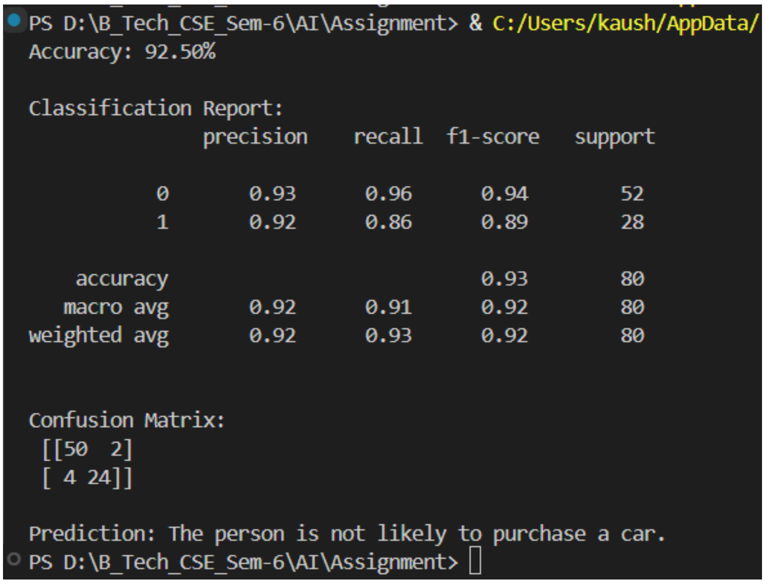
```
'''

@author: 22000409 Kaushal Ramoliya

@description: 22.1 -  Create a model to predict whether a person will have car
or not based on dataset attached using Naive Bayes Classifier.
(user_data_cars_1.csv)

'''

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix


# Load the dataset

df = pd.read_csv("Program_22.1_user_data_cars_1.csv")


# Drop the 'User ID' column as it's not useful for prediction

df = df.drop("User ID", axis=1)


# Encode the 'Gender' column (Male/Female -> 1/0)
```

```
le_gender = LabelEncoder()
df["Gender"] = le_gender.fit_transform(df["Gender"])

# Define features and target
X = df[["Gender", "Age", "EstimatedSalary"]]
y = df["Purchased"]

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Gaussian Naive Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))

# Example prediction
# Example input: Female, Age 30, EstimatedSalary 60000
sample_input = pd.DataFrame({
    "Gender": le_gender.transform(["Female"]),
    "Age": [30],
    "EstimatedSalary": [60000]
})


sample_prediction = model.predict(sample_input)


if sample_prediction[0] == 1:
    print("\nPrediction: The person is likely to purchase a car.")
else:
    print("\nPrediction: The person is not likely to purchase a car.")
```

INPUT: -

```
sample_input = pd.DataFrame({
    "Gender": le_gender.transform(["Female"]),
    "Age": [30],
    "EstimatedSalary": [60000]
})
```

OUTPUT: -



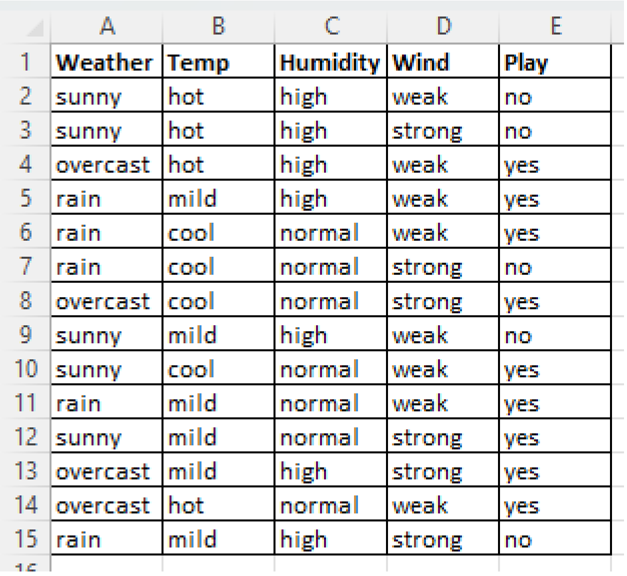
```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/
Accuracy: 92.50%

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.94        52
           1       0.92      0.86      0.89        28

    accuracy                           0.93        80
   macro avg       0.92      0.91      0.92        80
weighted avg       0.92      0.93      0.92        80


Confusion Matrix:
 [[50  2]
 [ 4 24]]

Prediction: The person is not likely to purchase a car.
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - The code uses a Gaussian Naive Bayes Classifier to predict whether a person is likely to purchase a car based on gender, age, and estimated salary, achieving accuracy and evaluation metrics on a test dataset. It also provides an example prediction for a given input.

PROGRAM CODE (22.2 using excel): -

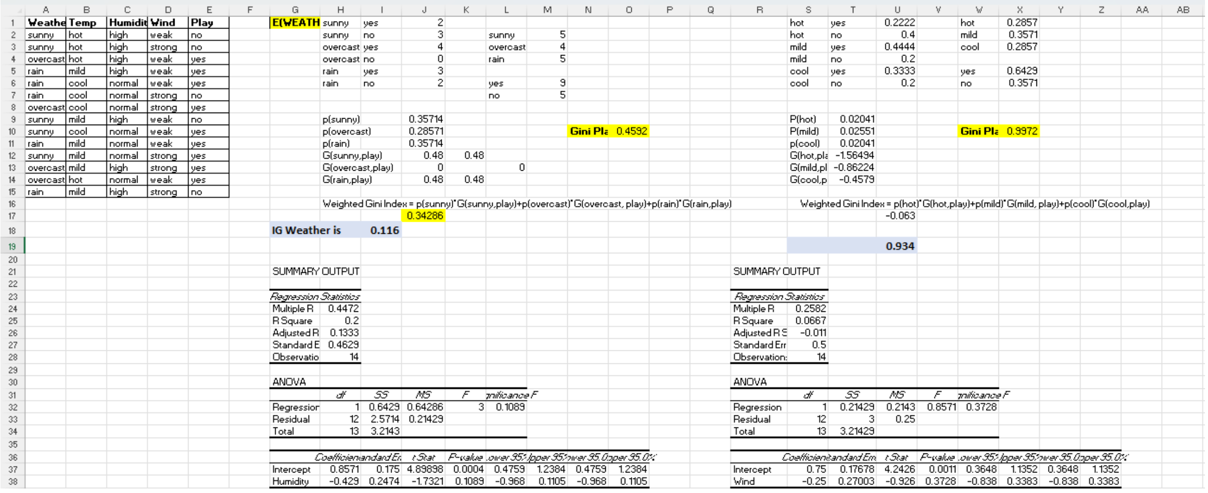
```
'''

@author: 22000409 Kaushal Ramoliya

@description: 22.2 -  Calculate Entropy and Gini for following dataset in Excel. (playplaynot.csv)

'''
```

INPUT: -



	A	B	C	D	E
1	Weather	Temp	Humidity	Wind	Play
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no

OUTPUT: -



	A	B	C	D	E
1	Weathe	Temp	Humidit	Wind	Play
2	sunny	hot	high	weak	no
3	sunny	hot	high	strong	no
4	overcast	hot	high	weak	yes
5	rain	mild	high	weak	yes
6	rain	cool	normal	weak	yes
7	rain	cool	normal	strong	no
8	overcast	cool	normal	strong	yes
9	sunny	mild	high	weak	no
10	sunny	cool	normal	weak	yes
11	rain	mild	normal	weak	yes
12	sunny	mild	normal	strong	yes
13	overcast	mild	high	strong	yes
14	overcast	hot	normal	weak	yes
15	rain	mild	high	strong	no

	G	H	I	J	K	L	M	N	O
1	E(WEATH	sunny	yes	2					
2		sunny	no	3		sunny	5		
3		overcast	yes	4		overcast	4		
4		overcast	no	0		rain	5		
5		rain	yes	3					
6		rain	no	2		yes	9		
7						no	5		
9		p(sunny)		0.35714					
10		p(overcast)		0.28571				Gini Pla	0.4592
11		p(rain)		0.35714					
12		G(sunny,play)		0.48	0.48				
13		G(overcast,play)		0		0			
14		G(rain,play)		0.48	0.48				
16		Weighted Gini Index = p(sunny)*G(sunny,play)+p(overcast)*G(overcast, play)+p(rain)*G(rain,play)							
17				0.34286					
18	IG Weather is		0.116						

	S	T	U	V	W	X
1	hot	yes	0.2222		hot	0.2857
2	hot	no	0.4		mild	0.3571
3	mild	yes	0.4444		cool	0.2857
4	mild	no	0.2			
5	cool	yes	0.3333		yes	0.6429
6	cool	no	0.2		no	0.3571
9	P(hot)	0.02041				
10	P(mild)	0.02551			Gini Pla	0.9972
11	p(cool)	0.02041				
12	G(hot,pla	-1.56494				
13	G(mild,pl	-0.86224				
14	G(cool,p	-0.4579				
16	Weighted Gini Index = p(hot)*G(hot,play)+p(mild)*G(mild, play)+p(cool)*G(cool,play)					
17			-0.063			
19			0.934			

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.4472
R Square	0.2
Adjusted R	0.1333
Standard E	0.4629
Observatio	14

ANOVA

	df	SS	MS	F	gnificance F
Regressior	1	0.6429	0.64286	3	0.1089
Residual	12	2.5714	0.21429		
Total	13	3.2143			

	Coefficien	andard Er	t Stat	P-value	ower 95%	Jpper 95%	wer 95.0	pper 95.0%
Intercept	0.8571	0.175	4.89898	0.0004	0.4759	1.2384	0.4759	1.2384
Humidity	-0.429	0.2474	-1.7321	0.1089	-0.968	0.1105	-0.968	0.1105

SUMMARY OUTPUT

Regression Statistics	
Multiple R	0.2582
R Square	0.0667
Adjusted R S	-0.011
Standard Err	0.5
Observation:	14

ANOVA

	df	SS	MS	F	gnificance F
Regression	1	0.21429	0.2143	0.8571	0.3728
Residual	12	3	0.25		
Total	13	3.21429			

	Coefficien	tandard Err	t Stat	P-value	ower 95%	Jpper 95%	wer 95.0	pper 95.0%
Intercept	0.75	0.17678	4.2426	0.0011	0.3648	1.1352	0.3648	1.1352
Wind	-0.25	0.27003	-0.926	0.3728	-0.838	0.3383	-0.838	0.3383

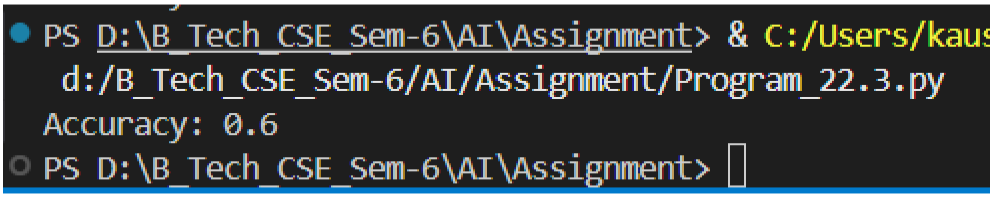
CONCLUSION: - The Excel sheet calculates Entropy and Gini Index for the given dataset to evaluate the impurity of splits for decision-making in classification tasks. It also computes Information Gain (IG) for attributes like Weather, aiding in feature selection.

PROGRAM CODE (22.3): -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 22.3 -  Write a python script to implement Decision Tree classifier on same
dataset.  (playplaynot.csv)
'''
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import LabelEncoder



df = pd.read_csv("Program_22.3_playplaynot.csv")

x = df.iloc[:, 0:4]
y = df.iloc[:, 4]

label_encoder = LabelEncoder()

x.loc[:,'Weather'] = label_encoder.fit_transform(x['Weather'])
x.loc[:,'Temp'] = label_encoder.fit_transform(x['Temp'])
x.loc[:,'Humidity'] = label_encoder.fit_transform(x['Humidity'])
x.loc[:, 'Wind'] = label_encoder.fit_transform(x['Wind'])

y = label_encoder.fit_transform(y)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

clf = DecisionTreeClassifier(criterion="entropy")

clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)

print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaus
 d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_22.3.py
Accuracy: 0.6
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

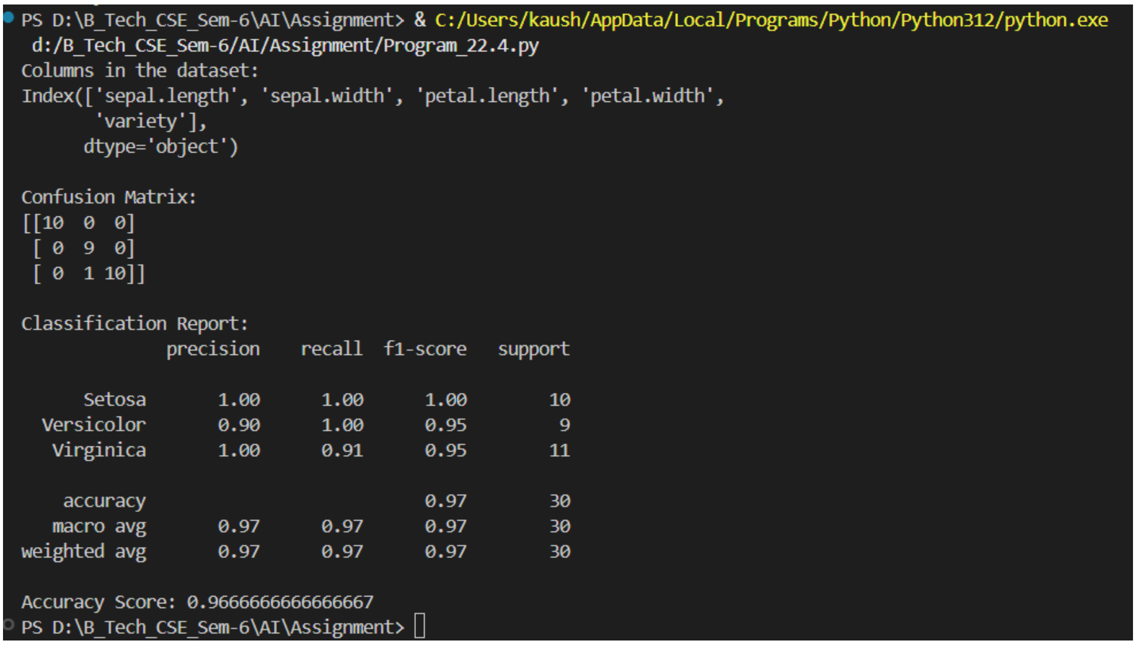
CONCLUSION: - The program implements a Decision Tree Classifier using the "playplaynot.csv" dataset to predict outcomes based on features like Weather, Temperature, Humidity, and Wind. It encodes categorical data, trains the model, and evaluates its accuracy on the test set.

PROGRAM CODE (22.4): -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 22.4 -  Write python script to implement Random Forest classifier on following dataset.
(iris.csv)
'''
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# 1. Load dataset
df = pd.read_csv("Program_22.4_iris.csv")

# 2. Show column names to identify target
print("Columns in the dataset:")
print(df.columns)

# Let's assume the last column is the target (usually correct for iris datasets)
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# 3. Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 4. Initialize and train the Random Forest Classifier
```

```
rfc = RandomForestClassifier(n_estimators=4, random_state=42)
rfc.fit(X_train, y_train)

# 5. Predict
y_pred = rfc.predict(X_test)

# 6. Evaluation
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/python.exe
 d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_22.4.py
Columns in the dataset:
Index(['sepal.length', 'sepal.width', 'petal.length', 'petal.width',
       'variety'],
      dtype='object')

Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  1 10]]

Classification Report:
              precision    recall  f1-score   support

      Setosa       1.00      1.00      1.00        10
  Versicolor       0.90      1.00      0.95         9
   Virginica       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

Accuracy Score: 0.9666666666666667
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - The program implements a Random Forest Classifier on the "iris.csv" dataset to classify iris species based on features. It trains the model, evaluates its performance using metrics like confusion matrix, classification report, and accuracy score, and demonstrates its effectiveness in multi-class classification.

PRACTICAL: - 23

AIM: Write a python script to implement

1. KNN Classifier and
2. KNN Regression

based on match on 3 attached datasets.
Record your observations with different parameters in the ML record sheet attached.
Upload code and ML Observation table.
Data set attached : user_data_cars_1.csv, pima-indiana-diabetes.csv, cars.csv

PROGRAM CODE: -

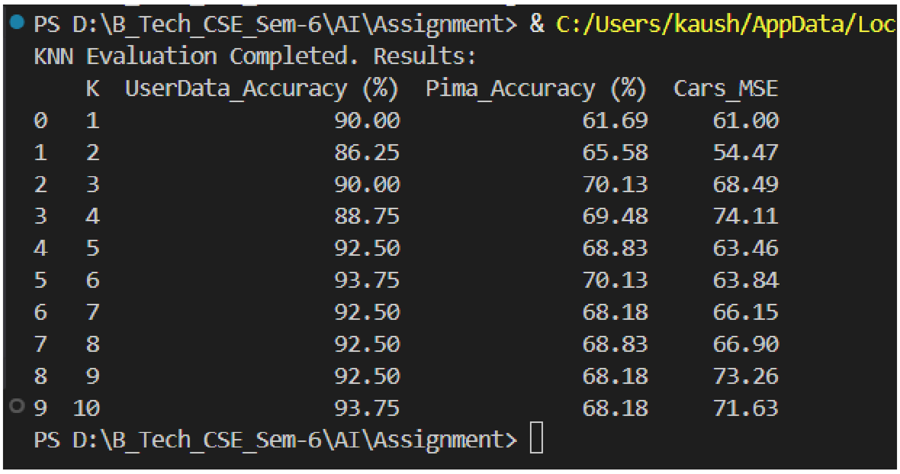
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 23. -  Write a python script to implement
1. KNN Classifier and
2. KNN Regression
'''
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, mean_squared_error


# Load datasets
user_data = pd.read_csv('Program_23_user_data_cars_1.csv')
pima = pd.read_csv('Program_23_pima-indians-diabetes.csv')
cars = pd.read_csv('Program_23cars.csv')


# Clean pima dataset
pima.columns = pima.iloc[0]  # Use the first row as column headers
pima = pima[1:].reset_index(drop=True)
pima = pima.apply(pd.to_numeric, errors='coerce').dropna()
```

```
# Encode 'Gender' in user data
user_data["Gender"] = LabelEncoder().fit_transform(user_data["Gender"])

# Features and targets
X_user = user_data[["Gender", "Age", "EstimatedSalary"]]
y_user = user_data["Purchased"]

X_pima = pima.drop(columns=["class"])
y_pima = pima["class"]

X_cars = cars[["Volume", "Weight"]]
y_cars = cars["CO2"]

# Standardize features
scaler = StandardScaler()
X_user = scaler.fit_transform(X_user)
X_pima = scaler.fit_transform(X_pima)
X_cars = scaler.fit_transform(X_cars)

# Train-Test Split
X_user_train, X_user_test, y_user_train, y_user_test = train_test_split(X_user, y_user, test_size=0.2, random_state=42)
X_pima_train, X_pima_test, y_pima_train, y_pima_test = train_test_split(X_pima, y_pima, test_size=0.2, random_state=42)
X_cars_train, X_cars_test, y_cars_train, y_cars_test = train_test_split(X_cars, y_cars, test_size=0.2, random_state=42)

# Record results
results = {"K": [], "UserData_Accuracy (%)": [], "Pima_Accuracy (%)": [], "Cars_MSE": []}
```

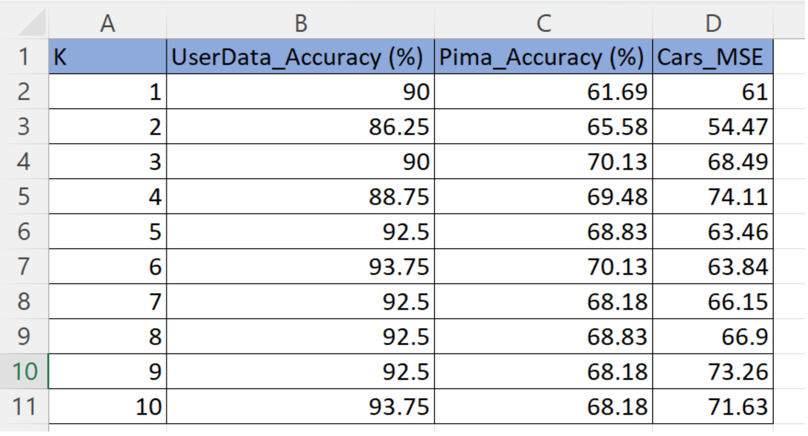
```
# --- KNN Loop ---
for k in range(1, 11):
    # User Data - Classification
    knn_user = KNeighborsClassifier(n_neighbors=k)
    knn_user.fit(X_user_train, y_user_train)
    pred_user = knn_user.predict(X_user_test)
    acc_user = accuracy_score(y_user_test, pred_user) * 100

    # Pima Data - Classification
    knn_pima = KNeighborsClassifier(n_neighbors=k)
    knn_pima.fit(X_pima_train, y_pima_train)
    pred_pima = knn_pima.predict(X_pima_test)
    acc_pima = accuracy_score(y_pima_test, pred_pima) * 100

    # Cars Data - Regression
    knn_cars = KNeighborsRegressor(n_neighbors=k)
    knn_cars.fit(X_cars_train, y_cars_train)
    pred_cars = knn_cars.predict(X_cars_test)
    mse_cars = mean_squared_error(y_cars_test, pred_cars)

    # Save results
    results["K"].append(k)
    results["UserData_Accuracy (%)"].append(round(acc_user, 2))
    results["Pima_Accuracy (%)"].append(round(acc_pima, 2))
    results["Cars_MSE"].append(round(mse_cars, 2))

# Save to CSV
results_df = pd.DataFrame(results)
```

```
results_df.to_csv("Program_23_KNN_Results.csv", index=False)


# Show the final result

print("KNN Evaluation Completed. Results:")

print(results_df)
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Loc
KNN Evaluation Completed. Results:
    K  UserData_Accuracy (%)  Pima_Accuracy (%)  Cars_MSE
0   1                  90.00              61.69     61.00
1   2                  86.25              65.58     54.47
2   3                  90.00              70.13     68.49
3   4                  88.75              69.48     74.11
4   5                  92.50              68.83     63.46
5   6                  93.75              70.13     63.84
6   7                  92.50              68.18     66.15
7   8                  92.50              68.83     66.90
8   9                  92.50              68.18     73.26
9  10                  93.75              68.18     71.63
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```



	A	B	C	D
1	K	UserData_Accuracy (%)	Pima_Accuracy (%)	Cars_MSE
2	1	90	61.69	61
3	2	86.25	65.58	54.47
4	3	90	70.13	68.49
5	4	88.75	69.48	74.11
6	5	92.5	68.83	63.46
7	6	93.75	70.13	63.84
8	7	92.5	68.18	66.15
9	8	92.5	68.83	66.9
10	9	92.5	68.18	73.26
11	10	93.75	68.18	71.63

CONCLUSION: - The program implements K-Nearest Neighbors (KNN) for both classification and regression tasks on three datasets: user data, Pima Indians Diabetes, and car emissions. It evaluates the model's performance for different values of K, recording classification accuracy for user and Pima datasets, and Mean Squared Error (MSE) for car emissions, saving the results to a CSV file.

PRACTICAL: - 24

AIM: Write a python script to implement

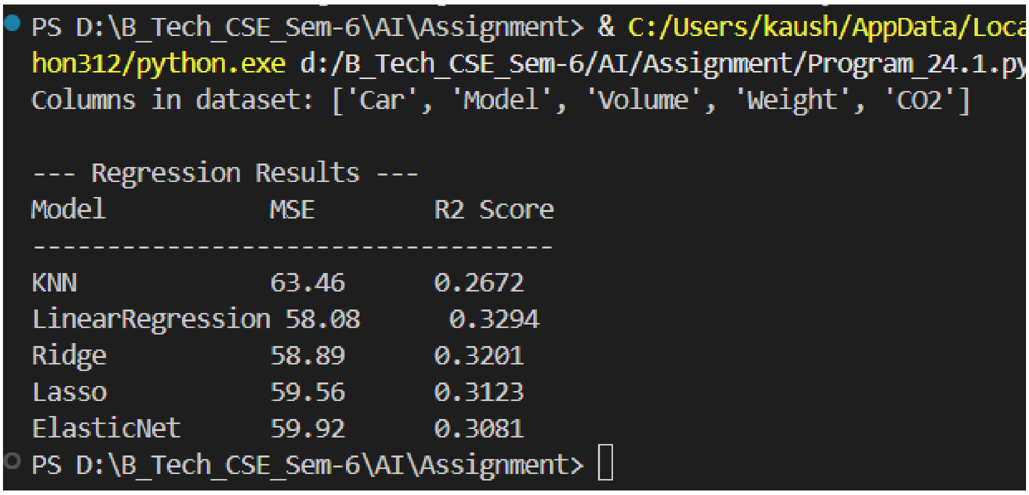
1. Regression using KNN, Linear, Ridge, Lasso and ElasticNet on cars.csv dataset to predict CO2 emission.
2. Classification using LogisticRegression on pima-indiana-diabetes.csv.

PROGRAM CODE (24.1): -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 24.1. -  Regression using KNN, Linear, Ridge, Lasso and ElasticNet on cars.csv dataset to predict
CO2 emission.
'''
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
cars = pd.read_csv('Program_24.1_cars.csv')  # Update path if needed

# Display columns to verify
print("Columns in dataset:", cars.columns.tolist())

# Select features and target
X = cars[['Volume', 'Weight']]
y = cars['CO2']

# Standardize features
```

```
scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Train-Test Split

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)


# Dictionary to store results

results = {}


# Define and evaluate models

models = {

    "KNN": KNeighborsRegressor(n_neighbors=5),

    "LinearRegression": LinearRegression(),

    "Ridge": Ridge(alpha=1.0),

    "Lasso": Lasso(alpha=0.1),

    "ElasticNet": ElasticNet(alpha=0.1, l1_ratio=0.5)

}


# Train, predict and calculate metrics

for name, model in models.items():

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    mse = mean_squared_error(y_test, y_pred)

    r2 = r2_score(y_test, y_pred)

    results[name] = {"MSE": round(mse, 2), "R2_Score": round(r2, 4)}


# Display the results

print("\n--- Regression Results ---")

print("{:<15} {:<10} {:<10}".format("Model", "MSE", "R2 Score"))
print("-" * 35)

for model, metrics in results.items():

    print(f"{model:<15} {metrics['MSE']:<10} {metrics['R2_Score']:<10}")
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Loca
hon312/python.exe d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_24.1.py
Columns in dataset: ['Car', 'Model', 'Volume', 'Weight', 'CO2']

--- Regression Results ---
Model           MSE        R2 Score
-----------------------------------
KNN             63.46      0.2672
LinearRegression 58.08      0.3294
Ridge           58.89      0.3201
Lasso           59.56      0.3123
ElasticNet      59.92      0.3081
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

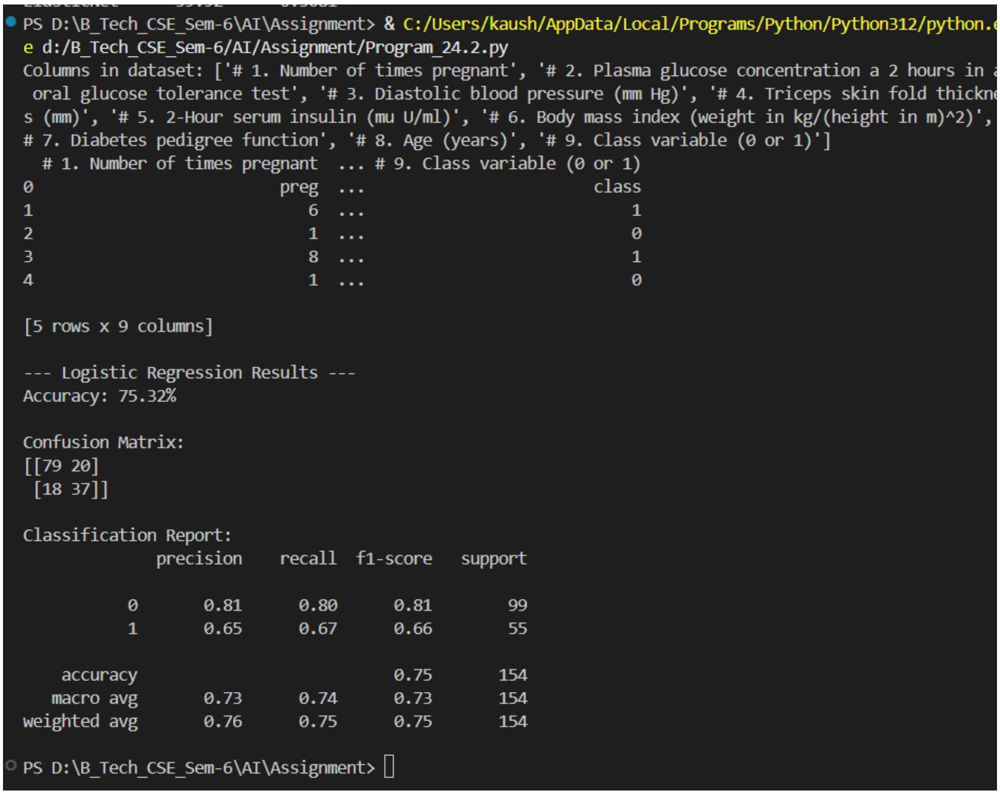
CONCLUSION: - The code evaluates regression models (KNN, Linear, Ridge, Lasso, ElasticNet) on the "cars.csv" dataset to predict CO2 emissions, calculating and comparing their Mean Squared Error (MSE) and R² scores. It identifies the performance of each model for better prediction accuracy.

PROGRAM CODE (24.2): -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 24.2. -  Classification using LogisticRegression on pima-indiana-diabetes.csv.
'''
import pandas as pd
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, classification_report

# Load dataset
pima = pd.read_csv('Program_24.2_pima-indians-diabetes.csv')

# Display columns and first few rows
print("Columns in dataset:", pima.columns.tolist())
print(pima.head())

# Clean dataset (if needed: sometimes first row might be column headers in disguised form)
if not pd.api.types.is_numeric_dtype(pima.iloc[0, 0]):
   pima.columns = pima.iloc[0]
   pima = pima[1:].reset_index(drop=True)

# Convert all to numeric and drop NaNs
pima = pima.apply(pd.to_numeric, errors='coerce').dropna()

# Features and target
X = pima.drop(columns=["class"])
```

```
y = pima["class"]

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

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Logistic Regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluation
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)

# Print results
print("\n--- Logistic Regression Results ---")
print(f"Accuracy: {acc * 100:.2f}%")
print("\nConfusion Matrix:")
print(cm)
print("\nClassification Report:")
print(report)
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/python.e
e d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_24.2.py
Columns in dataset: ['# 1. Number of times pregnant', '# 2. Plasma glucose concentration a 2 hours in a
 oral glucose tolerance test', '# 3. Diastolic blood pressure (mm Hg)', '# 4. Triceps skin fold thickne
s (mm)', '# 5. 2-Hour serum insulin (mu U/ml)', '# 6. Body mass index (weight in kg/(height in m)^2)',
# 7. Diabetes pedigree function', '# 8. Age (years)', '# 9. Class variable (0 or 1)']
  # 1. Number of times pregnant  ... # 9. Class variable (0 or 1)
0                          preg  ...                         class
1                             6  ...                             1
2                             1  ...                             0
3                             8  ...                             1
4                             1  ...                             0

[5 rows x 9 columns]

--- Logistic Regression Results ---
Accuracy: 75.32%

Confusion Matrix:
[[79 20]
 [18 37]]

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.80      0.81        99
           1       0.65      0.67      0.66        55

    accuracy                           0.75       154
   macro avg       0.73      0.74      0.73       154
weighted avg       0.76      0.75      0.75       154

PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```

CONCLUSION: - The code implements Logistic Regression on the "pima-indians-diabetes.csv" dataset to classify diabetes presence, achieving evaluation metrics like accuracy, confusion matrix, and classification report to assess model performance.

PRACTICAL: - 25

AIM: Develop a ML model to predict Quality of Milk (Low, Medium, High) from the given dataset (Milk_Quality.csv).

Perform following operations

1. Read the dataset.
2. Display the shape of dataset
3. Display columns of dataset.
4. Check for null values.
5. Show descriptive statistics of dataset.

Page 5 of 8

6. Display unique values in each column (for pH, Temp, etc)
7. Draw hist plots for each column.
8. Remove outliers if required.
9. Balance the dataset equally for the target output variable by removing or augmenting
records.
10. Using K-Best or any Feature selection technique, use the best X features.
11. Perform scaling or encoding on features.
12. Create multiple models.
13. Select the most appropriate model to host on web creating a web-api and consume.

PROGRAM CODE: -

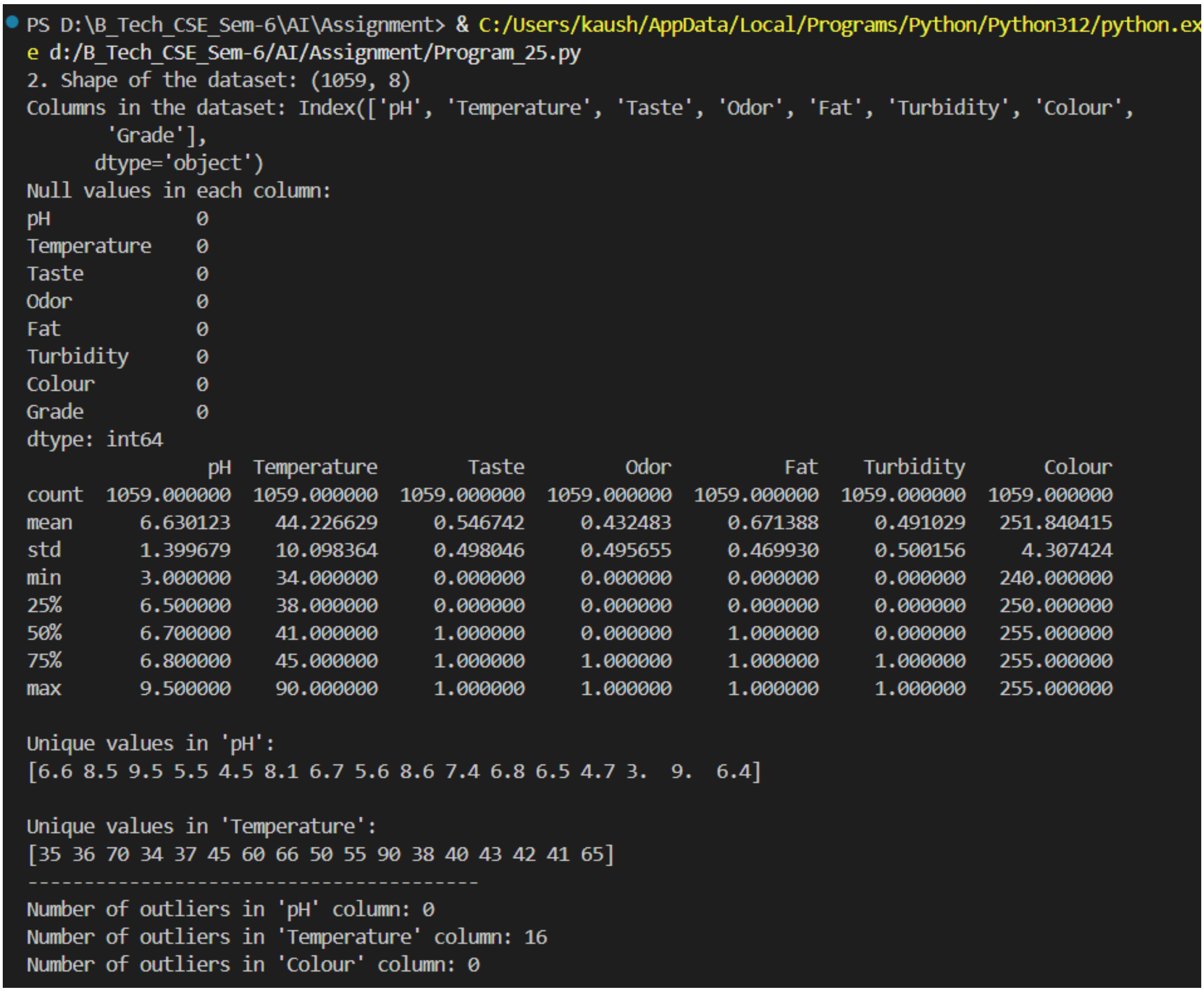
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 25. -  Develop a ML model to predict Quality of Milk (Low, Medium, High) from the given dataset
(Milk_Quality.csv).
'''
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import zscore
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

import seaborn as sns


# 1. Read the dataset

df = pd.read_csv("Program_25_milk_quality.csv")


# Clean column names (remove extra spaces)

df.columns = df.columns.str.strip()


# Fix common typos

df = df.rename(columns={"Temprature": "Temperature"})


# 2. Display the shape of dataset

print("2. Shape of the dataset:", df.shape)


# 3. Display columns of dataset

print("Columns in the dataset:", df.columns)


# 4. Check for null values

print("Null values in each column:")

print(df.isnull().sum())


# 5. Show descriptive statistics of dataset

print(df.describe())


# 6. Display unique values in selected columns

columns = ['pH', 'Temperature']
```

```
for col in columns:
    if col in df.columns:
        print(f"\nUnique values in '{col}':")
        print(df[col].unique())
    else:
        print(f"Column '{col}' not found.")
print("-" * 40)

# 7. Z-score based outlier detection and removal
def remove_outliers_zscore(df, columns, threshold=3):
    df_cleaned = df.copy()
    for col in columns:
        if col in df_cleaned.columns:
            z_scores = zscore(df_cleaned[col])
            df_cleaned = df_cleaned[np.abs(z_scores) <= threshold]
        else:
            print(f"Column '{col}' not found for Z-score outlier detection.")
    return df_cleaned

# Columns to apply Z-score outlier removal
zscore_columns = ['pH', 'Temperature', 'Colour']
df = remove_outliers_zscore(df, zscore_columns)

# 8. Limit data to 256 rows per 'Grade' category
if 'Grade' in df.columns:
    df = df.groupby('Grade').head(256).reset_index(drop=True)
else:
    print("Column 'Grade' not found!")
```

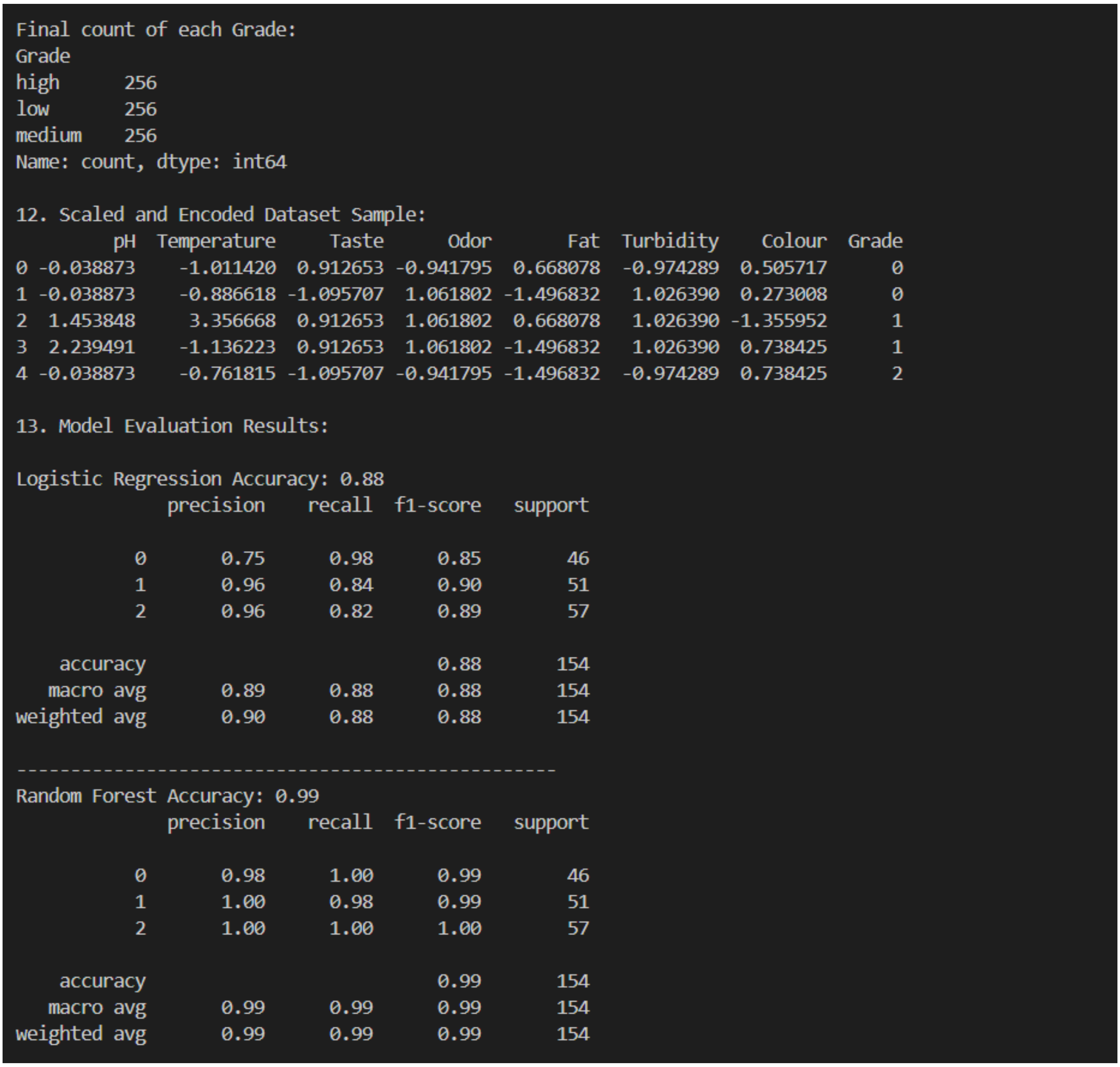
```
columns = ['pH', 'Temperature', 'Colour']
for col in columns:
    if col in df.columns:
        result = np.sum(np.abs(zscore(df[col])) > 3)
        print(f"Number of outliers in '{col}' column: {result}")
    else:
        print(f"Column '{col}' not found!")

# 9. Plot all histograms in a single screen using subplots with "Normal" labels
hist_columns = ['pH', 'Temperature', 'Taste', 'Odor', 'Fat', 'Turbidity', 'Colour']
available_cols = [col for col in hist_columns if col in df.columns]

# Reference "normal" values (you can update these as per domain knowledge)
reference_values = {
    'pH': 7.0,
    'Temperature': 35.0,
    'Taste': 1.0,
    'Odor': 1.0,
    'Fat': 2.0,
    'Turbidity': 1.0,
    'Colour': 255.0
}

plt.figure(figsize=(18, 12))
for i, col in enumerate(available_cols):
    plt.subplot(3, 3, i + 1)
    df[col].plot(kind='hist', bins=30, edgecolor='black')
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.grid(True)

    # Add vertical line and label for "Normal" value
    ref = reference_values.get(col, None)
    if ref is not None:
        plt.axvline(ref, color='red', linestyle='dashed', linewidth=1)
        plt.text(ref, plt.ylim()[1] * 0.9, 'Normal', color='red', fontsize=10, ha='center')

plt.tight_layout()
plt.show()

# 10. Show final count of each grade
print("\nFinal count of each Grade:")
print(df['Grade'].value_counts())

# 11. Show correlation matrix (relationship between all numerical columns)
plt.figure(figsize=(10, 8))
correlation_matrix = df.corr(numeric_only=True)

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5)
plt.title("Correlation Matrix between Features")
plt.show()

# 12. Perform Scaling and Encoding
df_scaled = df.copy()

# Separate numerical and categorical columns
numerical_cols = df_scaled.select_dtypes(include=['float64', 'int64']).columns.tolist()
```

```
categorical_cols = df_scaled.select_dtypes(include=['object']).columns.tolist()

# 12.1 Scale numerical features
scaler = StandardScaler()
df_scaled[numerical_cols] = scaler.fit_transform(df_scaled[numerical_cols])

# 12.2 Encode categorical features
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_scaled[col] = le.fit_transform(df_scaled[col])
    label_encoders[col] = le

print("\n12. Scaled and Encoded Dataset Sample:")
print(df_scaled.head())

# 13. Create and Evaluate Multiple Models
# 13.1 Prepare features and target
if 'Grade' in df_scaled.columns:
    X = df_scaled.drop('Grade', axis=1)
    y = df_scaled['Grade']
else:
    raise ValueError("Target column 'Grade' not found!")

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

# 13.3 Define models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "K-Nearest Neighbors": KNeighborsClassifier()
}

# 13.4 Train and evaluate each model
print("\n13. Model Evaluation Results:\n")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {acc:.2f}")
    print(classification_report(y_test, y_pred, zero_division=0))
    print("-" * 50)

# 13.4 Train and evaluate each model
print("\n13. Model Accuracy Comparison:\n")
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred) * 100  # Convert to percentage
    print(f"{name}: {acc:.2f}%")
```

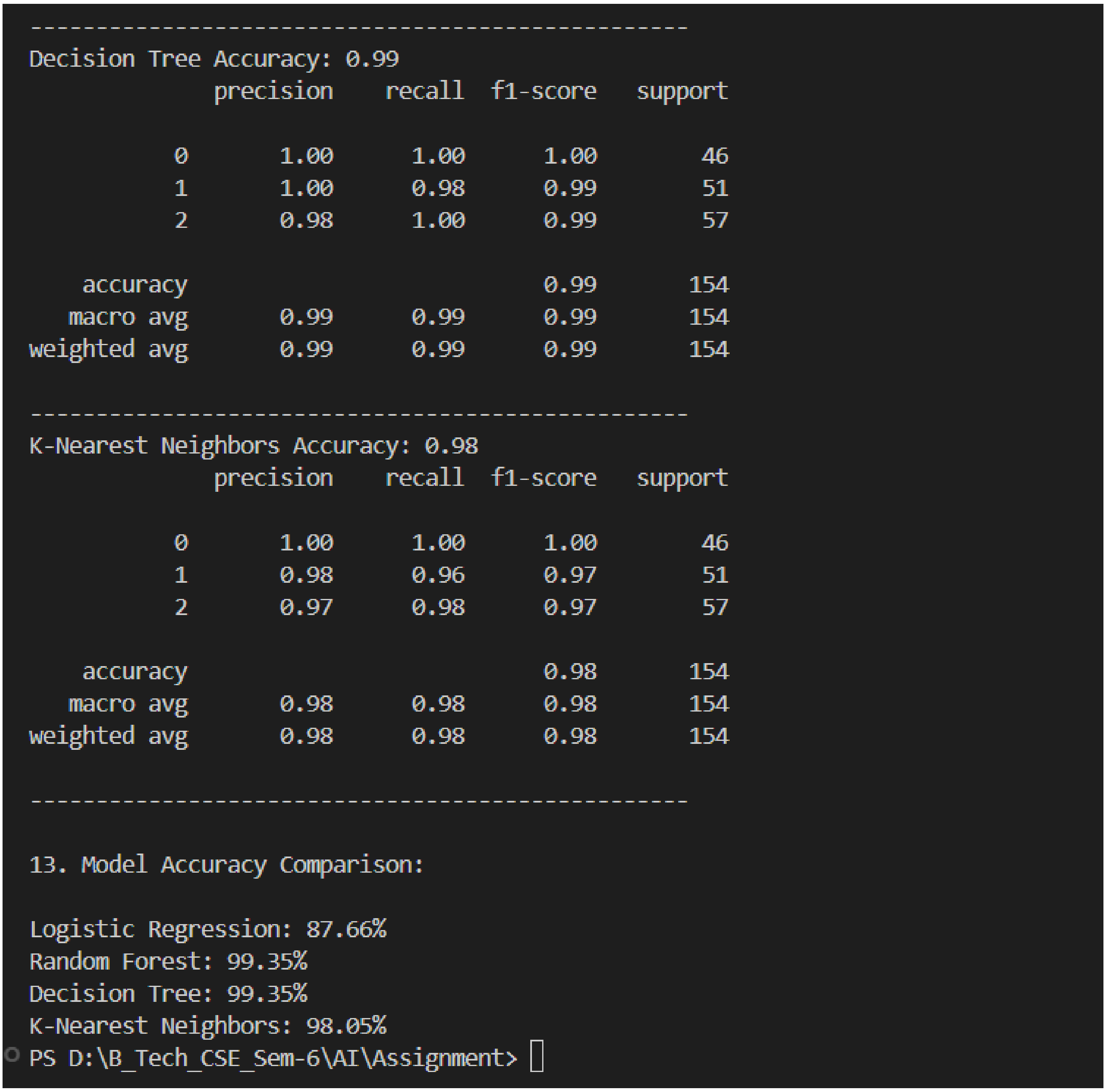
OUTPUT: -



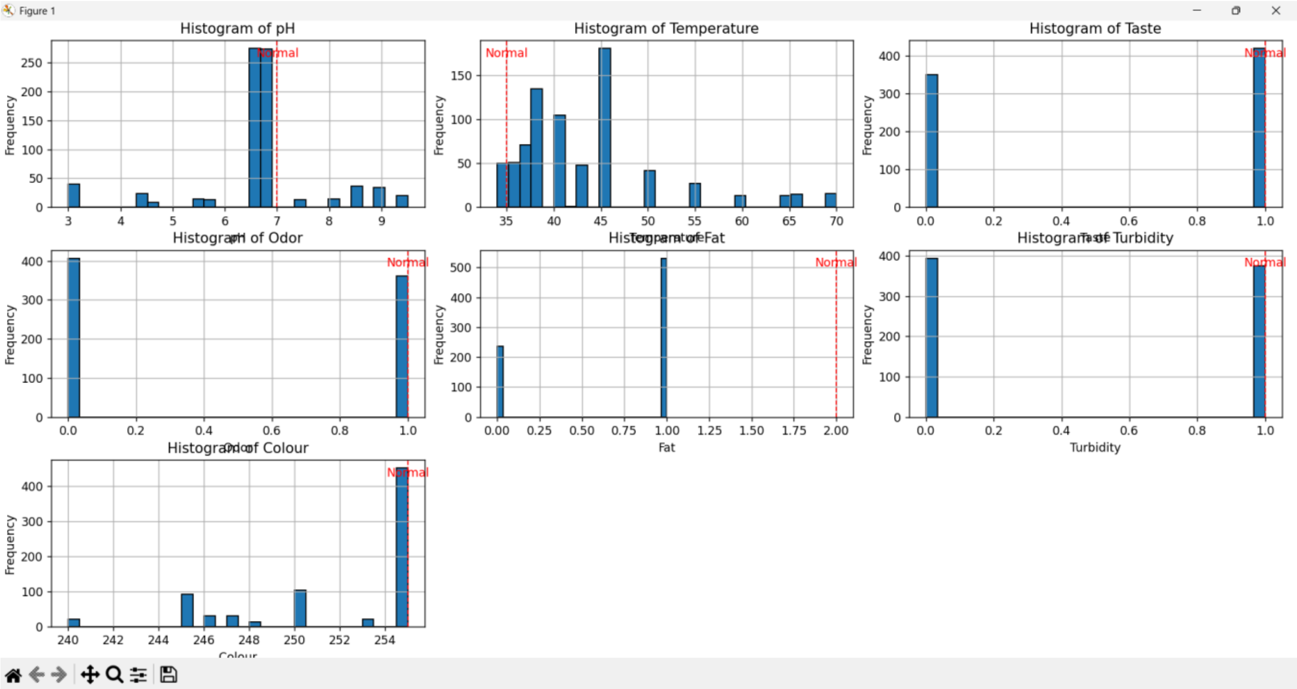
```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/python.ex
e d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_25.py
2. Shape of the dataset: (1059, 8)
Columns in the dataset: Index(['pH', 'Temperature', 'Taste', 'Odor', 'Fat', 'Turbidity', 'Colour',
       'Grade'],
      dtype='object')
Null values in each column:
pH             0
Temperature    0
Taste          0
Odor           0
Fat            0
Turbidity      0
Colour         0
Grade          0
dtype: int64
                pH  Temperature        Taste         Odor          Fat    Turbidity       Colour
count  1059.000000  1059.000000  1059.000000  1059.000000  1059.000000  1059.000000  1059.000000
mean      6.630123    44.226629     0.546742     0.432483     0.671388     0.491029   251.840415
std       1.399679    10.098364     0.498046     0.495655     0.469930     0.500156     4.307424
min       3.000000    34.000000     0.000000     0.000000     0.000000     0.000000   240.000000
25%       6.500000    38.000000     0.000000     0.000000     0.000000     0.000000   250.000000
50%       6.700000    41.000000     1.000000     0.000000     1.000000     0.000000   255.000000
75%       6.800000    45.000000     1.000000     1.000000     1.000000     1.000000   255.000000
max       9.500000    90.000000     1.000000     1.000000     1.000000     1.000000   255.000000

Unique values in 'pH':
[6.6 8.5 9.5 5.5 4.5 8.1 6.7 5.6 8.6 7.4 6.8 6.5 4.7 3.  9.  6.4]

Unique values in 'Temperature':
[35 36 70 34 37 45 60 66 50 55 90 38 40 43 42 41 65]
----------------------------------------
Number of outliers in 'pH' column: 0
Number of outliers in 'Temperature' column: 16
Number of outliers in 'Colour' column: 0
```

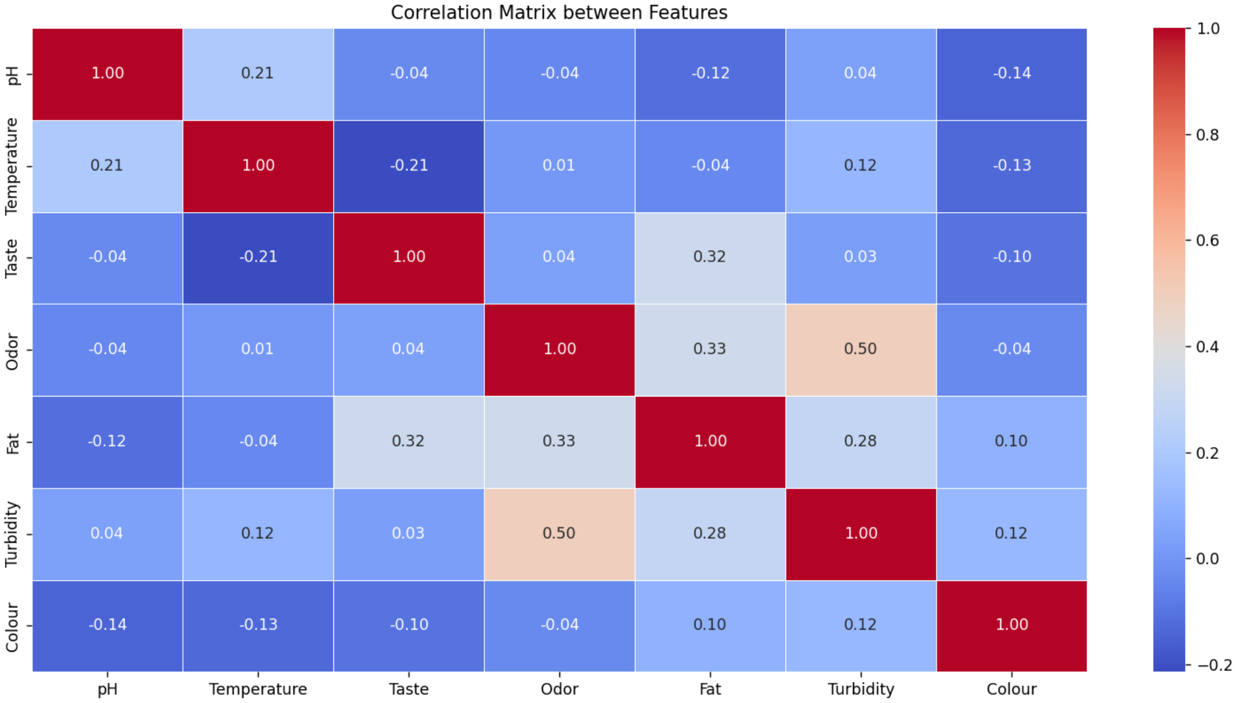


```
Final count of each Grade:
Grade
high      256
low       256
medium    256
Name: count, dtype: int64

12. Scaled and Encoded Dataset Sample:
         pH  Temperature     Taste      Odor       Fat  Turbidity    Colour  Grade
0 -0.038873    -1.011420  0.912653 -0.941795  0.668078  -0.974289  0.505717      0
1 -0.038873    -0.886618 -1.095707  1.061802 -1.496832   1.026390  0.273008      0
2  1.453848     3.356668  0.912653  1.061802  0.668078   1.026390 -1.355952      1
3  2.239491    -1.136223  0.912653  1.061802 -1.496832   1.026390  0.738425      1
4 -0.038873    -0.761815 -1.095707 -0.941795 -1.496832  -0.974289  0.738425      2

13. Model Evaluation Results:

Logistic Regression Accuracy: 0.88
              precision    recall  f1-score   support

           0       0.75      0.98      0.85        46
           1       0.96      0.84      0.90        51
           2       0.96      0.82      0.89        57

    accuracy                           0.88       154
   macro avg       0.89      0.88      0.88       154
weighted avg       0.90      0.88      0.88       154

--------------------------------------------------
Random Forest Accuracy: 0.99
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        46
           1       1.00      0.98      0.99        51
           2       1.00      1.00      1.00        57

    accuracy                           0.99       154
   macro avg       0.99      0.99      0.99       154
weighted avg       0.99      0.99      0.99       154
```



```
--------------------------------------------------
Decision Tree Accuracy: 0.99
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       1.00      0.98      0.99        51
           2       0.98      1.00      0.99        57

    accuracy                           0.99       154
   macro avg       0.99      0.99      0.99       154
weighted avg       0.99      0.99      0.99       154

--------------------------------------------------
K-Nearest Neighbors Accuracy: 0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        46
           1       0.98      0.96      0.97        51
           2       0.97      0.98      0.97        57

    accuracy                           0.98       154
   macro avg       0.98      0.98      0.98       154
weighted avg       0.98      0.98      0.98       154

--------------------------------------------------

13. Model Accuracy Comparison:

Logistic Regression: 87.66%
Random Forest: 99.35%
Decision Tree: 99.35%
K-Nearest Neighbors: 98.05%
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```





CONCLUSION: - The code develops a machine learning pipeline to predict milk quality (Low, Medium, High) using the "Milk_Quality.csv" dataset. It preprocesses data by handling outliers, scaling, and encoding, and evaluates multiple models (Logistic Regression, Random Forest, Decision Tree, K-Nearest Neighbors), comparing their accuracy and classification performance.

PRACTICAL: - 26

AIM: Develop a ML model to predict car price from the given dataset (usedcars.csv). Perform following operations

1. Read the dataset.
2. Display the shape of dataset
3. Display columns of dataset.
4. Check for null values.
5. Show descriptive statistics of dataset.
6. Display unique values in each column.
7. Draw hist plots for each column.
8. Remove outliers if required.
9. Using K-Best or any Feature selection technique, use the best X features.
10. Perform scaling or encoding on features.
11. Create multiple models.
12. Select the most appropriate model to host on web creating a web-api and consume.

PROGRAM CODE: -

```
'''

@author: 22000409 Kaushal Ramoliya

@description: 26. -   Develop a ML model to predict car price from the given dataset (usedcars.csv)

'''

import pandas as pd

from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsRegressor

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.neural_network import MLPRegressor

from sklearn.metrics import mean_squared_error, r2_score


# Load your dataset (example using a CSV file)

df = pd.read_csv('Program_26_usedcars.csv')
```

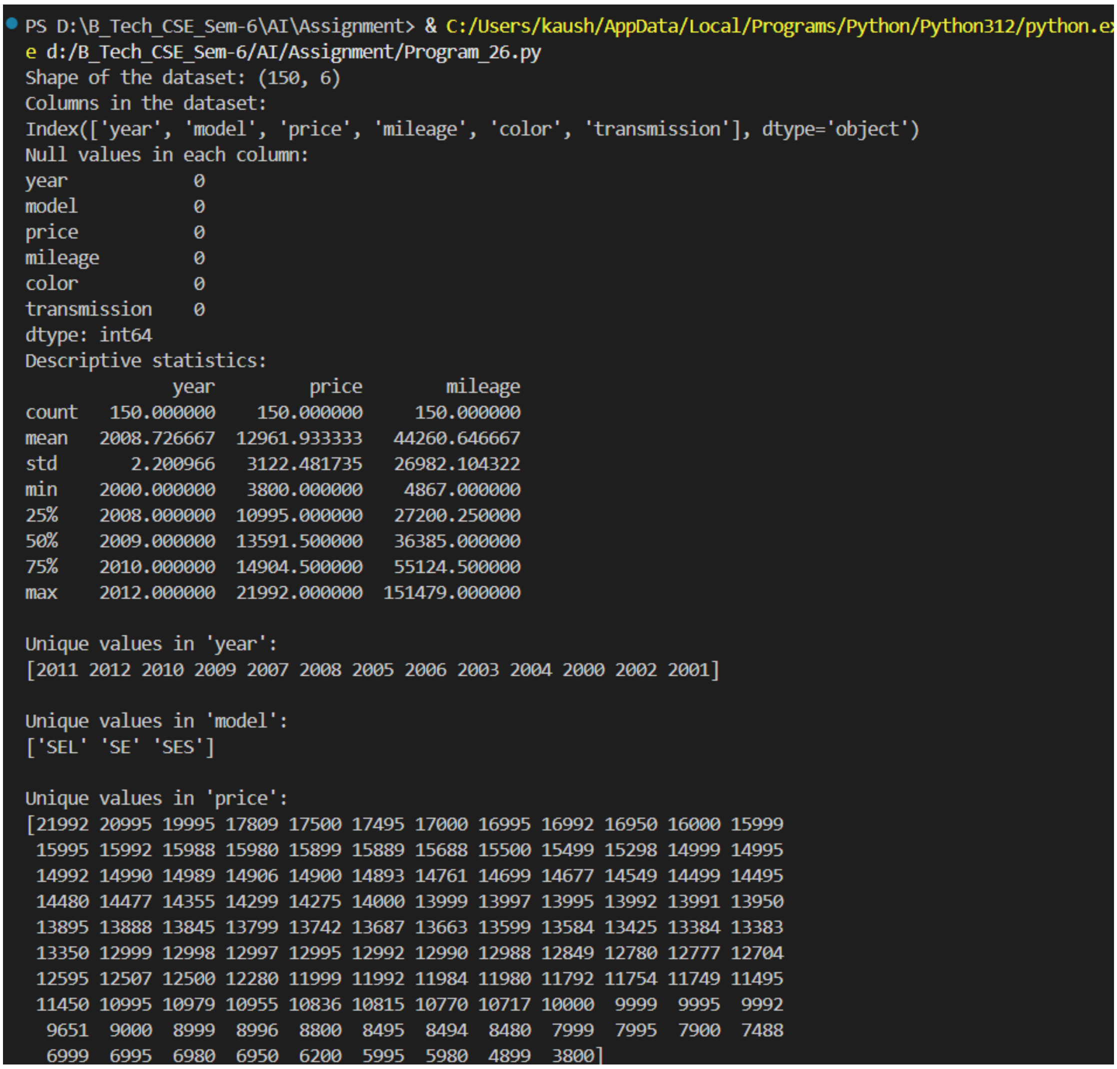
```
# Display the shape
print("Shape of the dataset:", df.shape)  # (rows, columns)

# Display the column names
print("Columns in the dataset:")
print(df.columns)

# Check for null values in each column
print("Null values in each column:")
print(df.isnull().sum())

# Show descriptive statistics
print("Descriptive statistics:")
print(df.describe())

# Display unique values for each column
for column in df.columns:
    print(f"\nUnique values in '{column}':")
    print(df[column].unique())

# Perform one-hot encoding on categorical features
encoder = OneHotEncoder(sparse_output=False, drop='first')  # Use sparse_output
instead of sparse
categorical_cols = df.select_dtypes(include=['object']).columns
encoded_features = encoder.fit_transform(df[categorical_cols])

# Combine numerical features and encoded categorical features
numerical_cols = df.select_dtypes(include=['number']).columns
encoded_df = pd.DataFrame(encoded_features,
columns=encoder.get_feature_names_out(categorical_cols))
```

```
df_clean = pd.concat([df[numerical_cols].reset_index(drop=True),
encoded_df.reset_index(drop=True)], axis=1)


# Replace 'target_column' with the actual target column name in your dataset
X = df_clean.drop(columns=['price'])
y = df_clean['price']


# Split 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)


# Dictionary to store model performance
model_performance = {}


# 1. K-Nearest Neighbors Regressor (KNNR)
knn = KNeighborsRegressor(n_neighbors=10)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
knn_mse = mean_squared_error(y_test, knn_pred)
knn_r2 = r2_score(y_test, knn_pred)
model_performance['KNNR'] = (knn_mse, knn_r2)


# 2. Linear Regression (LiR)
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
lr_mse = mean_squared_error(y_test, lr_pred)
lr_r2 = r2_score(y_test, lr_pred)
model_performance['Linear Regression'] = (lr_mse, lr_r2)
```

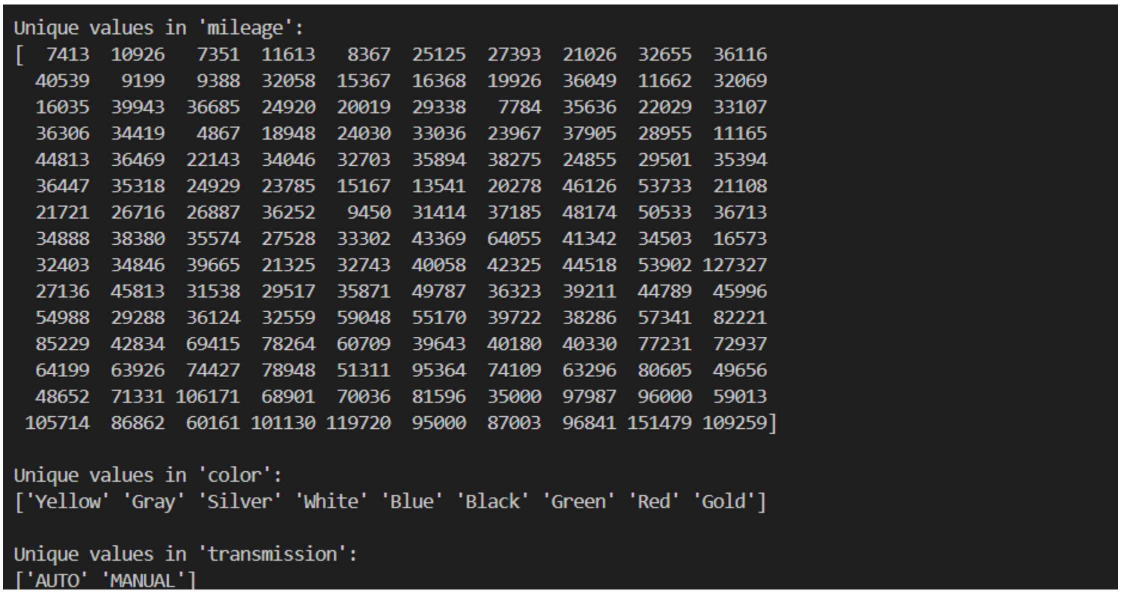
```
# 3. Random Forest Regressor (RFR)
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(X_train, y_train)
rfr_pred = rfr.predict(X_test)
rfr_mse = mean_squared_error(y_test, rfr_pred)
rfr_r2 = r2_score(y_test, rfr_pred)
model_performance['Random Forest Regressor'] = (rfr_mse, rfr_r2)

# 4. Multi-Layer Perceptron (MLP) with different architectures
mlp_1 = MLPRegressor(hidden_layer_sizes=(5, 8, 1), max_iter=500, random_state=42)
mlp_1.fit(X_train, y_train)
mlp_1_pred = mlp_1.predict(X_test)
mlp_1_mse = mean_squared_error(y_test, mlp_1_pred)
mlp_1_r2 = r2_score(y_test, mlp_1_pred)
model_performance['MLP (5/8/1)'] = (mlp_1_mse, mlp_1_r2)

mlp_2 = MLPRegressor(hidden_layer_sizes=(5, 8, 4, 1), max_iter=500, random_state=42)
mlp_2.fit(X_train, y_train)
mlp_2_pred = mlp_2.predict(X_test)
mlp_2_mse = mean_squared_error(y_test, mlp_2_pred)
mlp_2_r2 = r2_score(y_test, mlp_2_pred)
model_performance['MLP (5/8/4/1)'] = (mlp_2_mse, mlp_2_r2)

mlp_3 = MLPRegressor(hidden_layer_sizes=(5, 8, 12, 8, 1), max_iter=500, random_state=42)
mlp_3.fit(X_train, y_train)
mlp_3_pred = mlp_3.predict(X_test)
mlp_3_mse = mean_squared_error(y_test, mlp_3_pred)
```

```
mlp_3_r2 = r2_score(y_test, mlp_3_pred)
model_performance['MLP (5/8/12/8/1)'] = (mlp_3_mse, mlp_3_r2)


# Print model comparison
print("\nModel Performance Comparison:")
for model, (mse, r2) in model_performance.items():
    print(f"{model}: Mean Squared Error = {mse:.4f}, R² = {r2:.4f}")



# Find the best model based on MSE and R²
best_model = min(model_performance.items(), key=lambda x: (x[1][0], -x[1][1]))
print(f"\nBest Model: {best_model[0]}")
print(f"Mean Squared Error = {best_model[1][0]:.4f}, R² = {best_model[1][1]:.4f}")
```

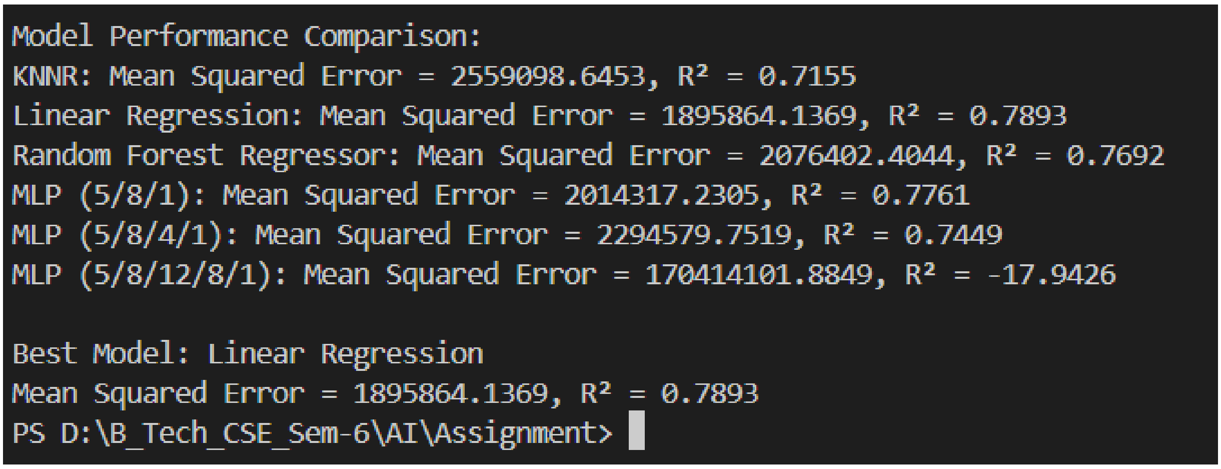
OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/python.ex
e d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_26.py
Shape of the dataset: (150, 6)
Columns in the dataset:
Index(['year', 'model', 'price', 'mileage', 'color', 'transmission'], dtype='object')
Null values in each column:
year            0
model           0
price           0
mileage         0
color           0
transmission    0
dtype: int64
Descriptive statistics:
              year         price        mileage
count   150.000000    150.000000     150.000000
mean   2008.726667  12961.933333   44260.646667
std       2.200966   3122.481735   26982.104322
min    2000.000000   3800.000000    4867.000000
25%    2008.000000  10995.000000   27200.250000
50%    2009.000000  13591.500000   36385.000000
75%    2010.000000  14904.500000   55124.500000
max    2012.000000  21992.000000  151479.000000

Unique values in 'year':
[2011 2012 2010 2009 2007 2008 2005 2006 2003 2004 2000 2002 2001]

Unique values in 'model':
['SEL' 'SE' 'SES']

Unique values in 'price':
[21992 20995 19995 17809 17500 17495 17000 16995 16992 16950 16000 15999
 15995 15992 15988 15980 15899 15889 15688 15500 15499 15298 14999 14995
 14992 14990 14989 14906 14900 14893 14761 14699 14677 14549 14499 14495
 14480 14477 14355 14299 14275 14000 13999 13997 13995 13992 13991 13950
 13895 13888 13845 13799 13742 13687 13663 13599 13584 13425 13384 13383
 13350 12999 12998 12997 12995 12992 12990 12988 12849 12780 12777 12704
 12595 12507 12500 12280 11999 11992 11984 11980 11792 11754 11749 11495
 11450 10995 10979 10955 10836 10815 10770 10717 10000  9999  9995  9992
  9651  9000  8999  8996  8800  8495  8494  8480  7999  7995  7900  7488
  6999  6995  6980  6950  6200  5995  5980  4899  3800]
```



```
Unique values in 'mileage':
[  7413  10926   7351  11613   8367  25125  27393  21026  32655  36116
  40539   9199   9388  32058  15367  16368  19926  36049  11662  32069
  16035  39943  36685  24920  20019  29338   7784  35636  22029  33107
  36306  34419   4867  18948  24030  33036  23967  37905  28955  11165
  44813  36469  22143  34046  32703  35894  38275  24855  29501  35394
  36447  35318  24929  23785  15167  13541  20278  46126  53733  21108
  21721  26716  26887  36252   9450  31414  37185  48174  50533  36713
  34888  38380  35574  27528  33302  43369  64055  41342  34503  16573
  32403  34846  39665  21325  32743  40058  42325  44518  53902 127327
  27136  45813  31538  29517  35871  49787  36323  39211  44789  45996
  54988  29288  36124  32559  59048  55170  39722  38286  57341  82221
  85229  42834  69415  78264  60709  39643  40180  40330  77231  72937
  64199  63926  74427  78948  51311  95364  74109  63296  80605  49656
  48652  71331 106171  68901  70036  81596  35000  97987  96000  59013
 105714  86862  60161 101130 119720  95000  87003  96841 151479 109259]

Unique values in 'color':
['Yellow' 'Gray' 'Silver' 'White' 'Blue' 'Black' 'Green' 'Red' 'Gold']

Unique values in 'transmission':
['AUTO' 'MANUAL']
```



```
Model Performance Comparison:
KNNR: Mean Squared Error = 2559098.6453, R² = 0.7155
Linear Regression: Mean Squared Error = 1895864.1369, R² = 0.7893
Random Forest Regressor: Mean Squared Error = 2076402.4044, R² = 0.7692
MLP (5/8/1): Mean Squared Error = 2014317.2305, R² = 0.7761
MLP (5/8/4/1): Mean Squared Error = 2294579.7519, R² = 0.7449
MLP (5/8/12/8/1): Mean Squared Error = 170414101.8849, R² = -17.9426

Best Model: Linear Regression
Mean Squared Error = 1895864.1369, R² = 0.7893
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - The code develops multiple regression models (KNN, Linear Regression, Random Forest, and Multi-Layer Perceptron with various architectures) to predict car prices using the "usedcars.csv" dataset. It evaluates each model's performance based on Mean Squared Error (MSE) and R^2 scores, identifying the best model for accurate price prediction.

PRACTICAL: - 27

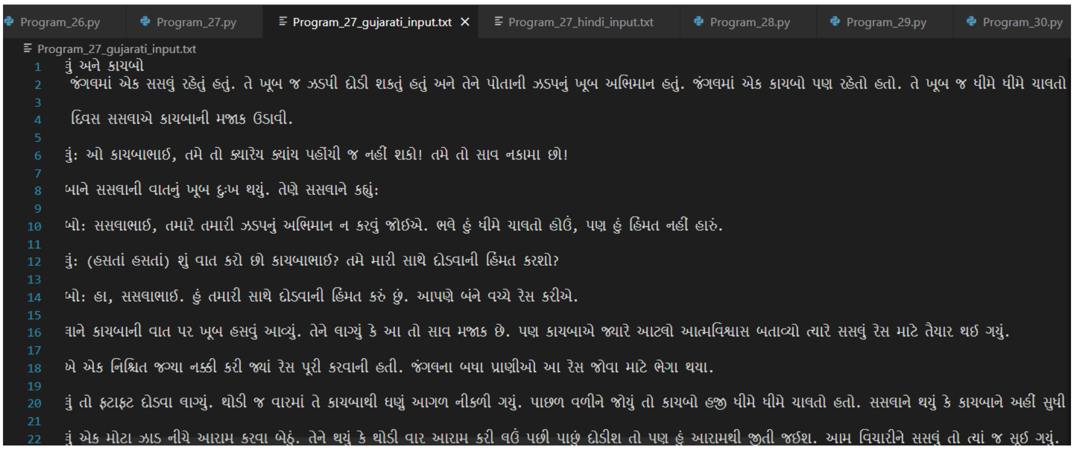
AIM: Write a python script to transliterate between hindi and Gujarati and vice-versa.
Please find unicode chart.
https://www.ssec.wisc.edu/~tomw/java/unicode.html

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 27. - Write a python script to transliterate between hindi and Gujarati and vice-versa.
Please find unicode chart
'''
def transliterate(content, mode="gujarati_to_hindi"):
    result = ""

    for ch in content:
        code_point = ord(ch)
        if mode == "gujarati_to_hindi":
            # Gujarati Unicode Range
            if 2688 <= code_point <= 2815:
                result += chr(code_point - 384)
            else:
                result += ch
        elif mode == "hindi_to_gujarati":
            # Hindi Unicode Range
            if 2304 <= code_point <= 2431:
                result += chr(code_point + 384)
            else:
                result += ch
    return result
```

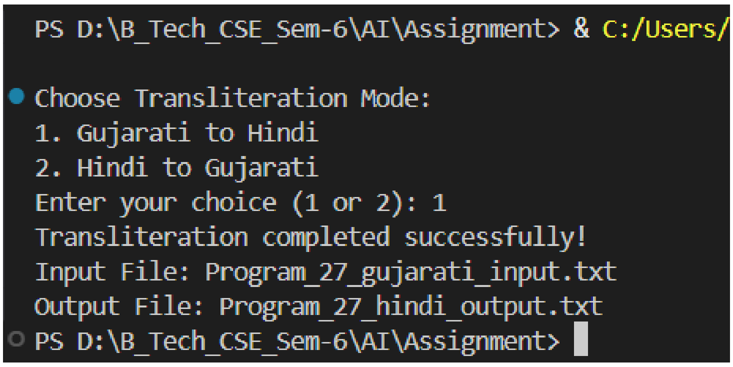
```
# Main Program
if __name__ == "__main__":
    print("Choose Transliteration Mode:")
    print("1. Gujarati to Hindi")
    print("2. Hindi to Gujarati")

    choice = input("Enter your choice (1 or 2): ")

    if choice == "1":
        input_filename = "Program_27_gujarati_input.txt"
        output_filename = "Program_27_hindi_output.txt"
        mode = "gujarati_to_hindi"
    elif choice == "2":
        input_filename = "Program_27_hindi_input.txt"
        output_filename = "Program_27_gujarati_output.txt"
        mode = "hindi_to_gujarati"
    else:
        print("Invalid choice. Please select 1 or 2.")
        exit()

    try:
        with open(input_filename, "r", encoding="utf-8") as fp:
            content = fp.read()

        transliterated_content = transliterate(content, mode)

        with open(output_filename, "w", encoding="utf-8") as fw:
            fw.write(transliterated_content)
        print(f"Transliteration completed successfully!")
        print(f"Input File: {input_filename}")
        print(f"Output File: {output_filename}")

    except FileNotFoundError:
        print(f"Error: The file {input_filename} was not found.")
```

INPUT (Gujrati to Hindi) : -



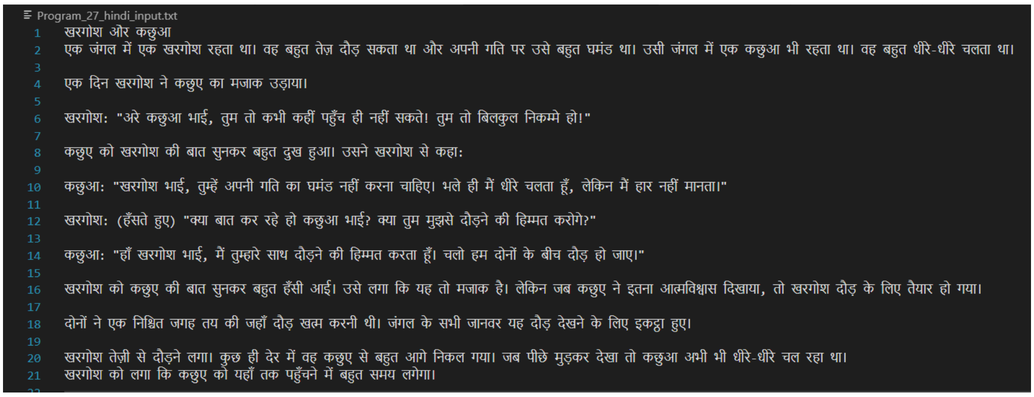
```
Program_27_gujarati_input.txt
ું અને કાચબો
 જંગલમાં એક સસલું રહેતું હતું. તે ખૂબ જ ઝડપી દોડી શકતું હતું અને તેને પોતાની ઝડપનું ખૂબ અભિમાન હતું. જંગલમાં એક કાચબો પણ રહેતો હતો. તે ખૂબ જ ધીમે ધીમે ચાલતો

 દિવસ સસલાએ કાચબાની મજાક ઉડાવી.

ું: ઓ કાચબાભાઈ, તમે તો ક્યારેય ક્યાંય પહોંચી જ નહીં શકો! તમે તો સાવ નકામા છો!

બાને સસલાની વાતનું ખૂબ દુઃખ થયું. તેણે સસલાને કહ્યું:

બો: સસલાભાઈ, તમારે તમારી ઝડપનું અભિમાન ન કરવું જોઈએ. ભલે હું ધીમે ચાલતો હોઉં, પણ હું હિંમત નહીં હારું.

ું: (હસતાં હસતાં) શું વાત કરો છો કાચબાભાઈ? તમે મારી સાથે દોડવાની હિંમત કરશો?

બો: હા, સસલાભાઈ. હું તમારી સાથે દોડવાની હિંમત કરું છું. આપણે બંને વચ્ચે રેસ કરીએ.

ાને કાચબાની વાત પર ખૂબ હસવું આવ્યું. તેને લાગ્યું કે આ તો સાવ મજાક છે. પણ કાચબાએ જ્યારે આટલો આત્મવિશ્વાસ બતાવ્યો ત્યારે સસલું રેસ માટે તૈયાર થઈ ગયું.

ખે એક નિશ્ચિત જગ્યા નક્કી કરી જ્યાં રેસ પૂરી કરવાની હતી. જંગલના બધા પ્રાણીઓ આ રેસ જોવા માટે ભેગા થયા.

ું તો ફટાફટ દોડવા લાગ્યું. થોડી જ વારમાં તે કાચબાથી ઘણું આગળ નીકળી ગયું. પાછળ વળીને જોયું તો કાચબો હજી ધીમે ધીમે ચાલતો હતો. સસલાને થયું કે કાચબાને અહીં સુધી

ું એક મોટા ઝાડ નીચે આરામ કરવા બેઠું. તેને થયું કે થોડી વાર આરામ કરી લઉં પછી પાછું દોડીશ તો પણ હું આરામથી જીતી જઈશ. આમ વિચારીને સસલું તો ત્યાં જ સૂઈ ગયું.
```

OUTPUT: -



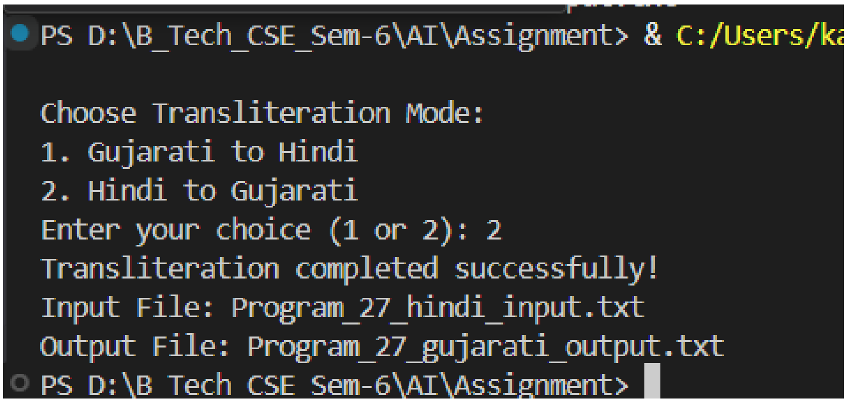
```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/

Choose Transliteration Mode:
1. Gujarati to Hindi
2. Hindi to Gujarati
Enter your choice (1 or 2): 1
Transliteration completed successfully!
Input File: Program_27_gujarati_input.txt
Output File: Program_27_hindi_output.txt
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

INPUT (Hindi to Gujrati): -



```
Program_27_hindi_input.txt
खरगोश और कछुआ
एक जंगल में एक खरगोश रहता था। वह बहुत तेज़ दौड़ सकता था और अपनी गति पर उसे बहुत घमंड था। उसी जंगल में एक कछुआ भी रहता था। वह बहुत धीरे-धीरे चलता था।

एक दिन खरगोश ने कछुए का मजाक उड़ाया।

खरगोश: "अरे कछुआ भाई, तुम तो कभी कहीं पहुँच ही नहीं सकते! तुम तो बिलकुल निकम्मे हो!"

कछुए को खरगोश की बात सुनकर बहुत दुख हुआ। उसने खरगोश से कहा:

कछुआ: "खरगोश भाई, तुम्हें अपनी गति का घमंड नहीं करना चाहिए। भले ही मैं धीरे चलता हूँ, लेकिन मैं हार नहीं मानता।"

खरगोश: (हँसते हुए) "क्या बात कर रहे हो कछुआ भाई? क्या तुम मुझसे दौड़ने की हिम्मत करोगे?"

कछुआ: "हाँ खरगोश भाई, मैं तुम्हारे साथ दौड़ने की हिम्मत करता हूँ। चलो हम दोनों के बीच दौड़ हो जाए।"

खरगोश को कछुए की बात सुनकर बहुत हँसी आई। उसे लगा कि यह तो मजाक है। लेकिन जब कछुए ने इतना आत्मविश्वास दिखाया, तो खरगोश दौड़ के लिए तैयार हो गया।

दोनों ने एक निश्चित जगह तय की जहाँ दौड़ खत्म करनी थी। जंगल के सभी जानवर यह दौड़ देखने के लिए इकट्ठा हुए।

खरगोश तेज़ी से दौड़ने लगा। कुछ ही देर में वह कछुए से बहुत आगे निकल गया। जब पीछे मुड़कर देखा तो कछुआ अभी भी धीरे-धीरे चल रहा था।
खरगोश को लगा कि कछुए को यहाँ तक पहुँचने में बहुत समय लगेगा।
```

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/ka

Choose Transliteration Mode:
1. Gujarati to Hindi
2. Hindi to Gujarati
Enter your choice (1 or 2): 2
Transliteration completed successfully!
Input File: Program_27_hindi_input.txt
Output File: Program_27_gujarati_output.txt
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program provides a script to transliterate text between Gujarati and Hindi languages using Unicode character mapping. It efficiently handles the conversion by adjusting Unicode code points and supports file-based input and output, making it a practical tool for transliteration tasks.

PRACTICAL: - 28

AIM: Write a Python script for language transliteration between Gujarati and English Script.

Input : આપણે બધા કત્રિમ બદ્ધિ ત્રિષય શીખી રહ્યા છે.

output : Aapde badha krutrim buddhi vishay sikhi rahya chee.

PROGRAM CODE: -

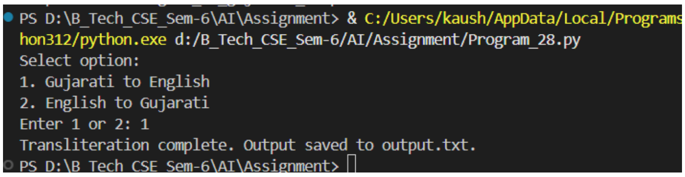
```
'''

@author: 22000409 Kaushal Ramoliya

@description: 28. -  Write a Python script for language transliteration between
Gujarati and English Script.

Input : આપણે બધા કત્રિમ બદ્ધિ ત્રિષય શીખી રહ્યા છે.

output : Aapde badha krutrim buddhi vishay sikhi rahya chee.

'''

# Simple transliteration maps

gujarati_to_english_map = {

   'અ': 'a', 'આ': 'aa', 'ઇ': 'i', 'ઈ': 'ee', 'ઉ': 'u', 'ઊ': 'oo',

   'ઋ': 'ru', 'એ': 'e', 'ઐ': 'ai', 'ઓ': 'o', 'ઔ': 'au',

   'ક': 'k', 'ખ': 'kh', 'ગ': 'g', 'ઘ': 'gh', 'ઙ': 'ng',

   'ચ': 'ch', 'છ': 'chh', 'જ': 'j', 'ઝ': 'jh', 'ઞ': 'ny',

   'ટ': 't', 'ઠ': 'th', 'ડ': 'd', 'ઢ': 'dh', 'ણ': 'n',

   'ત': 't', 'થ': 'th', 'દ': 'd', 'ધ': 'dh', 'ન': 'n',

   'પ': 'p', 'ફ': 'ph', 'બ': 'b', 'ભ': 'bh', 'મ': 'm',

   'ય': 'y', 'ર': 'r', 'લ': 'l', 'વ': 'v',

   'શ': 'sh', 'ષ': 'sh', 'સ': 's', 'હ': 'h',

   'ળ': 'l', 'ક્ષ': 'ksh', 'જ્ઞ': 'gy',

   'ા': 'aa', 'િ': 'i', 'ી': 'ee', 'ુ': 'u', 'ૂ': 'oo',

   'ે': 'e', 'ૈ': 'ai', 'ો': 'o', 'ૌ': 'au',

   'ૃ': 'ru',

   '્': '',    # halant

   'ં': 'n', 'ઃ': 'h', 'ઁ': 'n'
}

# Reverse mapping for English to Gujarati
english_to_gujarati_map = {v: k for k, v in gujarati_to_english_map.items()}

# Special cases where mapping conflicts (like 'sh' for both શ and ષ)
# So you can manually fix if needed.

# Matras list
matras = ['ા', 'િ', 'ી', 'ુ', 'ૂ', 'ે', 'ૈ', 'ો', 'ૌ', 'ૃ']

def transliterate_gujarati_to_english(text):
    result = ''
    skip_next = False
    for idx, char in enumerate(text):
        if skip_next:
            skip_next = False
            continue

        if char == '્' and idx > 0:
            continue

        if idx + 1 < len(text) and text[idx + 1] in matras:
            base = gujarati_to_english_map.get(char, char)
            matra = gujarati_to_english_map.get(text[idx + 1], '')
            result += base + matra
            skip_next = True
        else:
            result += gujarati_to_english_map.get(char, char)
    return result


def transliterate_english_to_gujarati(text):
    result = ''
    idx = 0
    while idx < len(text):
        match = ''
        match_char = ''

        # Try to match the longest possible sequence (3-letter, 2-letter, 1-letter)
        for l in [3, 2, 1]:
            if idx + l <= len(text):
                part = text[idx:idx+l]
                if part in english_to_gujarati_map:
                    match = part
                    match_char = english_to_gujarati_map[part]
                    break

        if match:
            result += match_char
            idx += len(match)
        else:
            result += text[idx]
            idx += 1

    return result

if __name__ == "__main__":
```

```
print("Select option:")
print("1. Gujarati to English")
print("2. English to Gujarati")
choice = input("Enter 1 or 2: ")

with open('Program_28_input.txt', 'r', encoding='utf-8') as f:
    input_text = f.read()

if choice == '1':
    output_text = transliterate_gujarati_to_english(input_text)
elif choice == '2':
    output_text = transliterate_english_to_gujarati(input_text)
else:
    print("Invalid choice.")
    exit()

with open('Program_28_output.txt', 'w', encoding='utf-8') as f:
    f.write(output_text)

print("Transliteration complete. Output saved to output.txt.")
```

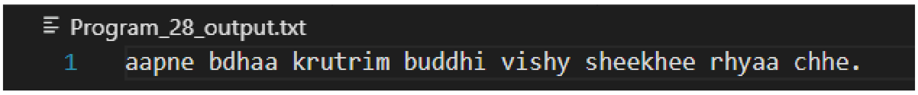
INPUT (Gujrati to English) : -

આપણે બધા કૃત્રિમ બુદ્ધિ વિષય શીખી રહ્યા છે.

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs
hon312/python.exe d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_28.py
Select option:
1. Gujarati to English
2. English to Gujarati
Enter 1 or 2: 1
Transliteration complete. Output saved to output.txt.
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```



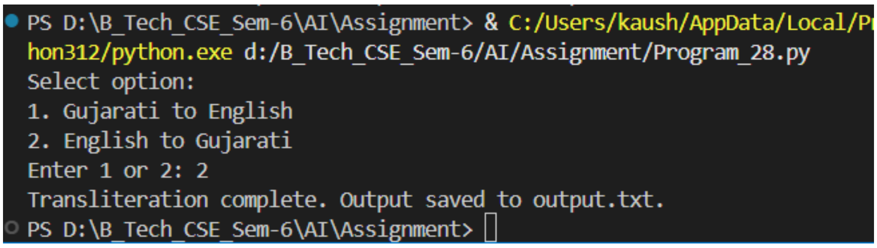
Program_28_output.txt

```
aapne bdhaa krutrim buddhi vishy sheekhee rhyaa chhe.
```

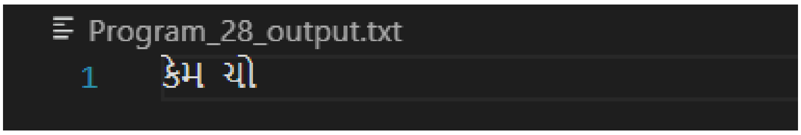
INPUT (Hindi to English): -

kem cho

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Pr
hon312/python.exe d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_28.py
Select option:
1. Gujarati to English
2. English to Gujarati
Enter 1 or 2: 2
Transliteration complete. Output saved to output.txt.
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```



Program_28_output.txt

```
કેમ ચો
```

CONCLUSION: - This program provides a script for transliteration between Gujarati and English scripts using predefined character mappings. It efficiently handles the conversion of text from one script to another, supporting matras and special cases, and saves the transliterated output to a file. This demonstrates a practical application of transliteration for language processing tasks.

PRACTICAL: - 29

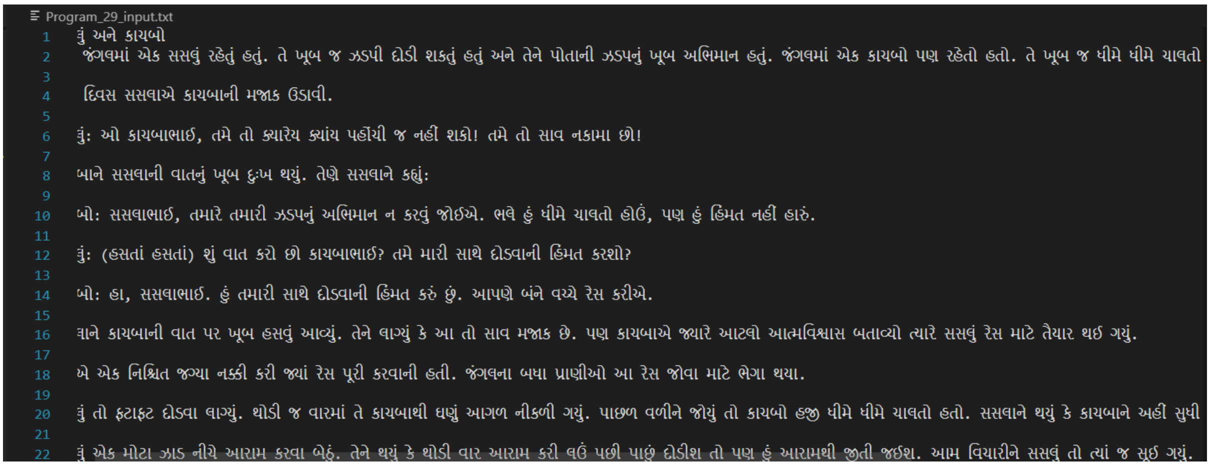
AIM: Write an Object-Oriented Program which reads texts from a file. It must display file statistics a below.

a. No. of sentences.
b. No. of words.
c. No. of total characters (Does not include whitespace)
d. No. of whitespaces
e. Total no. of digits, uppercase and lowercase letters.

PROGRAM CODE: -

```
class TextFileAnalyzer:
    def __init__(self, input_file, output_file):
        self.input_file = input_file
        self.output_file = output_file
        self.text = ""
        self.stats = {
            "sentences": 0,
            "words": 0,
            "characters": 0,
            "whitespaces": 0,
            "digits": 0,
            "uppercase_letters": 0,
            "lowercase_letters": 0
        }

    def read_file(self):
        try:
            with open(self.input_file, 'r', encoding='utf-8') as file:
                self.text = file.read()
        except FileNotFoundError:
            print(f"Error: File '{self.input_file}' not found.")
    def analyze(self):
        self.stats["sentences"] = self.text.count('.') + self.text.count('!') +
self.text.count('?')
        self.stats["words"] = len(self.text.split())
        self.stats["whitespaces"] = self.text.count(' ')
        self.stats["characters"] = len([c for c in self.text if not c.isspace()])
        self.stats["digits"] = sum(c.isdigit() for c in self.text)
        self.stats["uppercase_letters"] = sum(c.isupper() for c in self.text)
        self.stats["lowercase_letters"] = sum(c.islower() for c in self.text)

    def write_output(self):
        with open(self.output_file, 'w', encoding='utf-8') as file:
            file.write(f"Number of sentences: {self.stats['sentences']}\n")
            file.write(f"Number of words: {self.stats['words']}\n")
            file.write(f"Number of total characters (excluding whitespace):
{self.stats['characters']}\n")
            file.write(f"Number of whitespaces: {self.stats['whitespaces']}\n")
            file.write(f"Total number of digits: {self.stats['digits']}\n")
            file.write(f"Total number of uppercase letters:
{self.stats['uppercase_letters']}\n")
            file.write(f"Total number of lowercase letters:
{self.stats['lowercase_letters']}\n")

    def process(self):
        self.read_file()
        self.analyze()
        self.write_output()

if __name__ == "__main__":
    analyzer = TextFileAnalyzer("Program_29_input.txt", "Program_29_output.txt")
    analyzer.process()
```

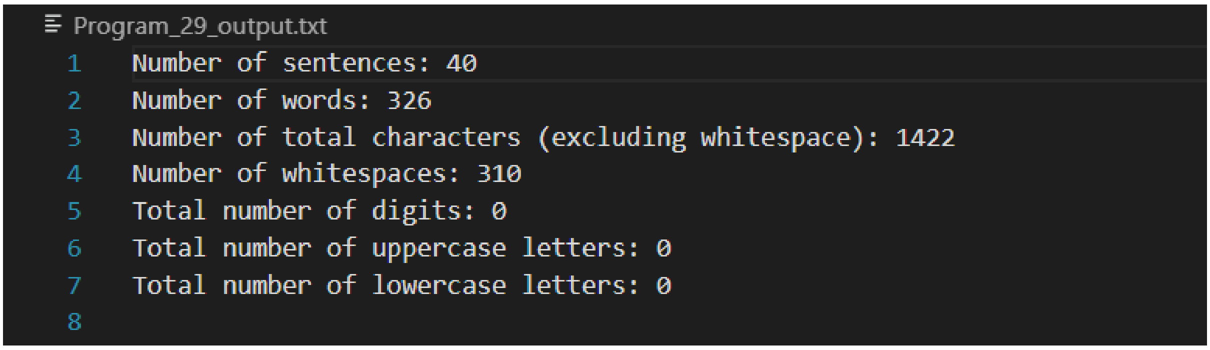
INPUT: -



Program_29_input.txt

```
ું અને કાચબો
 જંગલમાં એક સસલું રહેતું હતું. તે ખૂબ જ ઝડપી દોડી શકતું હતું અને તેને પોતાની ઝડપનું ખૂબ અભિમાન હતું. જંગલમાં એક કાચબો પણ રહેતો હતો. તે ખૂબ જ ધીમે ધીમે ચાલતો

 દિવસ સસલાએ કાચબાની મજાક ઉડાવી.

ું: ઓ કાચબાભાઈ, તમે તો ક્યારેય ક્યાંય પહોંચી જ નહીં શકો! તમે તો સાવ નકામા છો!

બાને સસલાની વાતનું ખૂબ દુઃખ થયું. તેણે સસલાને કહ્યું:

બો: સસલાભાઇ, તમારે તમારી ઝડપનું અભિમાન ન કરવું જોઈએ. ભલે હું ધીમે ચાલતો હોઉં, પણ હું હિંમત નહીં હારું.

ું: (હસતાં હસતાં) શું વાત કરો છો કાચબાભાઈ? તમે મારી સાથે દોડવાની હિંમત કરશો?

બો: હા, સસલાભાઈ. હું તમારી સાથે દોડવાની હિંમત કરું છું. આપણે બંને વચ્ચે રેસ કરીએ.

લાને કાચબાની વાત પર ખૂબ હસવું આવ્યું. તેને લાગ્યું કે આ તો સાવ મજાક છે. પણ કાચબાએ જ્યારે આટલો આત્મવિશ્વાસ બતાવ્યો ત્યારે સસલું રેસ માટે તૈયાર થઈ ગયું.

બે એક નિશ્ચિત જગ્યા નક્કી કરી જ્યાં રેસ પૂરી કરવાની હતી. જંગલના બધા પ્રાણીઓ આ રેસ જોવા માટે ભેગા થયા.

ું તો ફટાફટ દોડવા લાગ્યું. થોડી જ વારમાં તે કાચબાથી ઘણું આગળ નીકળી ગયું. પાછળ વળીને જોયું તો કાચબો હજી ધીમે ધીમે ચાલતો હતો. સસલાને થયું કે કાચબાને અહીં સુધી

ું એક મોટા ઝાડ નીચે આરામ કરવા બેઠું. તેને થયું કે થોડી વાર આરામ કરી લઉં પછી પાછું દોડીશ તો પણ હું આરામથી જીતી જઈશ. આમ વિચારીને સસલું તો ત્યાં જ સૂઈ ગયું.
```

OUTPUT: -



Program_29_output.txt

```
Number of sentences: 40
Number of words: 326
Number of total characters (excluding whitespace): 1422
Number of whitespaces: 310
Total number of digits: 0
Total number of uppercase letters: 0
Total number of lowercase letters: 0

```

CONCLUSION: -This program implements a class to analyze a text file and generate statistics such as the number of sentences, words, characters, whitespaces, digits, uppercase letters, and lowercase letters. It demonstrates efficient file handling, text processing, and output generation, making it a useful tool for text analysis tasks.

PRACTICAL: - 30

AIM: Write an Object Oriented Program which creates vocabulary of words and also counts each word in a document.
Eg. Content
The birds are flying. The boy is walking. The Ganges are great river system. The Narmada river flows from rift valley.
output :
[(The,3), (birds,1), (are,1), (birds,1), (are,2), (flying,1), (boy,1), (river,2)]

PROGRAM CODE: -

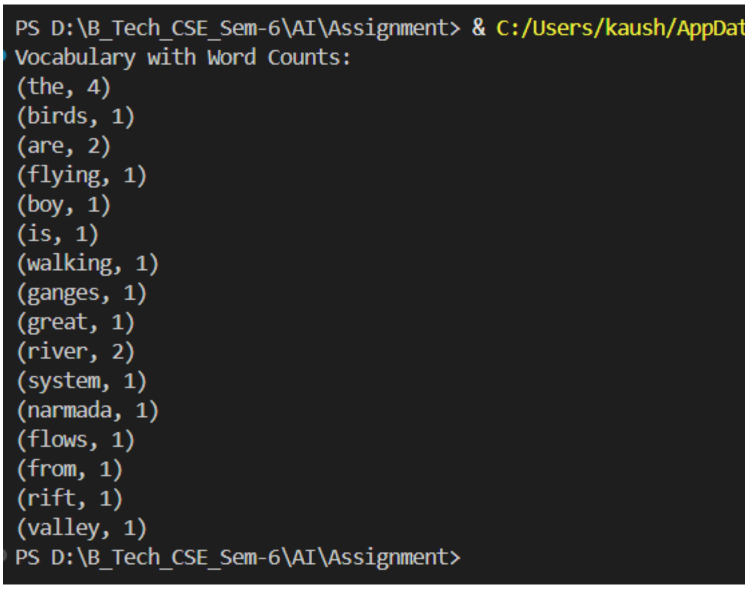
```
'''

@author: 22000409 Kaushal Ramoliya

@description: 30. -  Write an Object Oriented Program which creates vocabulary of

words and also counts each word in a document.

Eg. Content

The birds are flying. The boy is walking. The Ganges are great river system. The

Narmada

river flows from rift valley.

output :

[(The,3), (birds,1), (are,1), (birds,1), (are,2), (flying,1), (boy,1), (river,2)]

'''

import re

from collections import Counter



class Vocabulary:

    def __init__(self, file_path):

        self.file_path = file_path

        self.word_counts = Counter()
```

```
def process_document(self):
    # Read the file content
    with open(self.file_path, 'r', encoding='utf-8') as file:
        content = file.read()

    # Remove punctuation and convert to lowercase
    content = re.sub(r'[^\w\s]', '', content).lower()

    # Tokenize the text into words
    words = content.split()

    # Count the frequency of each word
    self.word_counts = Counter(words)

def get_vocabulary(self):
    # Return the vocabulary as a list of tuples (word, count)
    return list(self.word_counts.items())

def display_vocabulary(self):
    # Display the vocabulary
    print("Vocabulary with Word Counts:")
    for word, count in self.word_counts.items():
        print(f"({word}, {count})")


# Example usage

file_path = "Program_30_input.txt"  # Replace with the path to your input file

vocab = Vocabulary(file_path)

vocab.process_document()

vocab.display_vocabulary()
```

INPUT: -

The birds are flying. The boy is walking. The Ganges are great river system. The Narmada river flows from rift valley.

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppDat
Vocabulary with Word Counts:
(the, 4)
(birds, 1)
(are, 2)
(flying, 1)
(boy, 1)
(is, 1)
(walking, 1)
(ganges, 1)
(great, 1)
(river, 2)
(system, 1)
(narmada, 1)
(flows, 1)
(from, 1)
(rift, 1)
(valley, 1)
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program implements a Vocabulary class to create a vocabulary of words from a document and count their occurrences. It demonstrates efficient text processing by removing punctuation, converting text to lowercase, tokenizing words, and calculating word frequencies, providing a clear and structured approach to vocabulary generation and analysis.

PRACTICAL: - 31

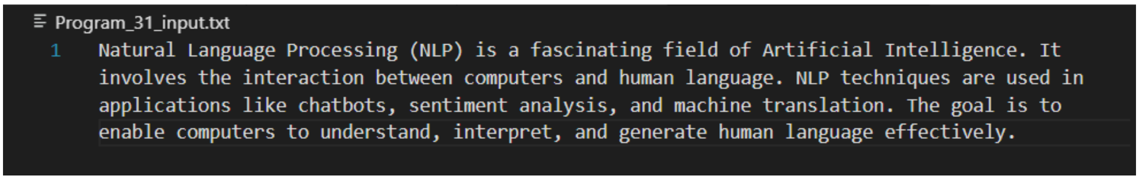
AIM: Develop an NLP application which tokenizes text, removes punctuation marks, converts to lower case, removes spelling errors, removes stopwords, convert to root word using either stemmer or lemmatizer and displays counts/frequency of the main text words.

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 31. - Develop an NLP application which tokenizes text, removes punctuation marks, converts to lower case, removes spelling errors, removes stopwords, convert to root word using either stemmer or lemmatizer and displays counts/frequency of the main text words.
'''
import string
from collections import Counter
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from textblob import TextBlob
import nltk

# Step 1: Read input text
with open('Program_31_input.txt', 'r') as file:
    text = file.read()

# Step 2: Convert to lowercase
text = text.lower()

# Step 3: Remove punctuation
text = text.translate(str.maketrans('', '', string.punctuation))
```

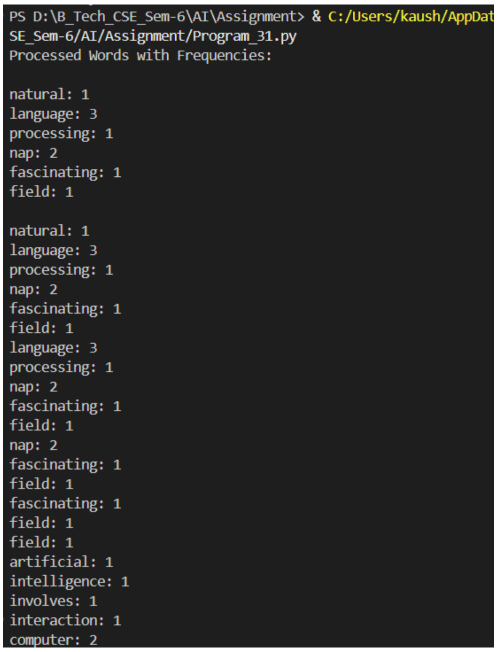
```
# Step 4: Correct spelling using TextBlob
corrected_text = str(TextBlob(text).correct())

# Write corrected text to output.txt
with open('Program_31_output.txt', 'w') as file:
    file.write(corrected_text)

# Step 5: Tokenize corrected text
tokens = word_tokenize(corrected_text)

# Step 6: Remove stopwords
stop_words = set(stopwords.words('english'))
filtered_tokens = [word for word in tokens if word not in stop_words]

# Step 7: Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]

# Step 8: Count word frequencies
word_counts = Counter(lemmatized_tokens)

# Step 9: Display word frequencies
print("Processed Words with Frequencies:\n")
for word, freq in word_counts.items():
    print(f"{word}: {freq}")
```

INPUT: -

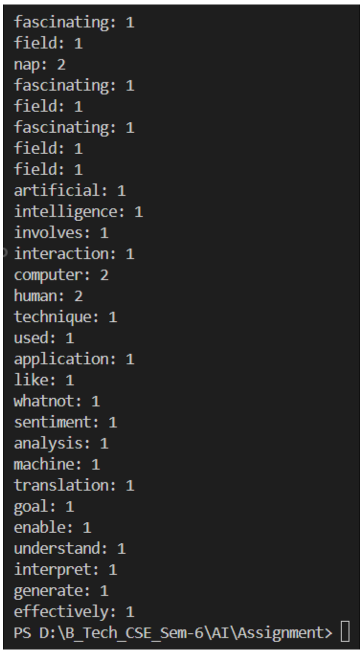


```
Program_31_input.txt
Natural Language Processing (NLP) is a fascinating field of Artificial Intelligence. It
involves the interaction between computers and human language. NLP techniques are used in
applications like chatbots, sentiment analysis, and machine translation. The goal is to
enable computers to understand, interpret, and generate human language effectively.
```

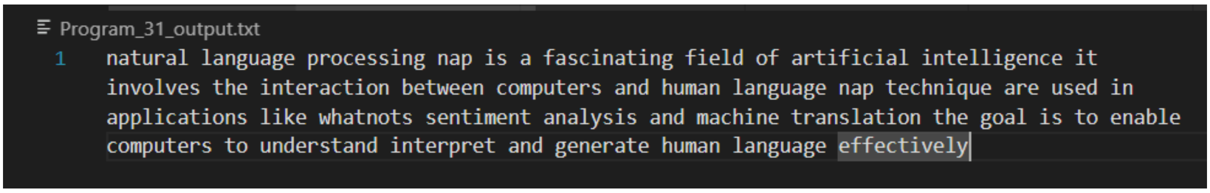
OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppDat
SE_Sem-6/AI/Assignment/Program_31.py
Processed Words with Frequencies:

natural: 1
language: 3
processing: 1
nap: 2
fascinating: 1
field: 1

natural: 1
language: 3
processing: 1
nap: 2
fascinating: 1
field: 1
language: 3
processing: 1
nap: 2
fascinating: 1
field: 1
nap: 2
fascinating: 1
field: 1
fascinating: 1
field: 1
field: 1
artificial: 1
intelligence: 1
involves: 1
interaction: 1
computer: 2
```



```
fascinating: 1
field: 1
nap: 2
fascinating: 1
field: 1
fascinating: 1
field: 1
field: 1
artificial: 1
intelligence: 1
involves: 1
interaction: 1
computer: 2
human: 2
technique: 1
used: 1
application: 1
like: 1
whatnot: 1
sentiment: 1
analysis: 1
machine: 1
translation: 1
goal: 1
enable: 1
understand: 1
interpret: 1
generate: 1
effectively: 1
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> 
```



```
Program_31_output.txt
natural language processing nap is a fascinating field of artificial intelligence it
involves the interaction between computers and human language nap technique are used in
applications like whatnots sentiment analysis and machine translation the goal is to enable
computers to understand interpret and generate human language effectively
```

CONCLUSION: - This program implements an NLP application that processes text by tokenizing, removing punctuation, converting to lowercase, correcting spelling errors, removing stopwords, and lemmatizing words. It then calculates and displays the frequency of the processed words. This demonstrates a comprehensive pipeline for text preprocessing and analysis, which is essential for various NLP tasks.

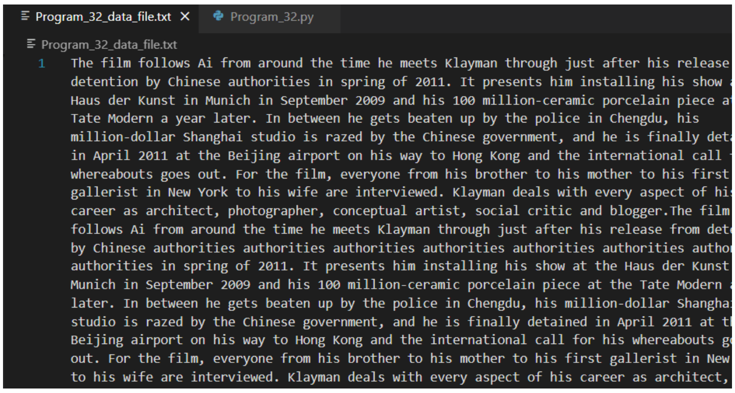
PRACTICAL: - 32

AIM: Write a program for next word prediction using N-Gram conditional probability.

PROGRAM CODE: -

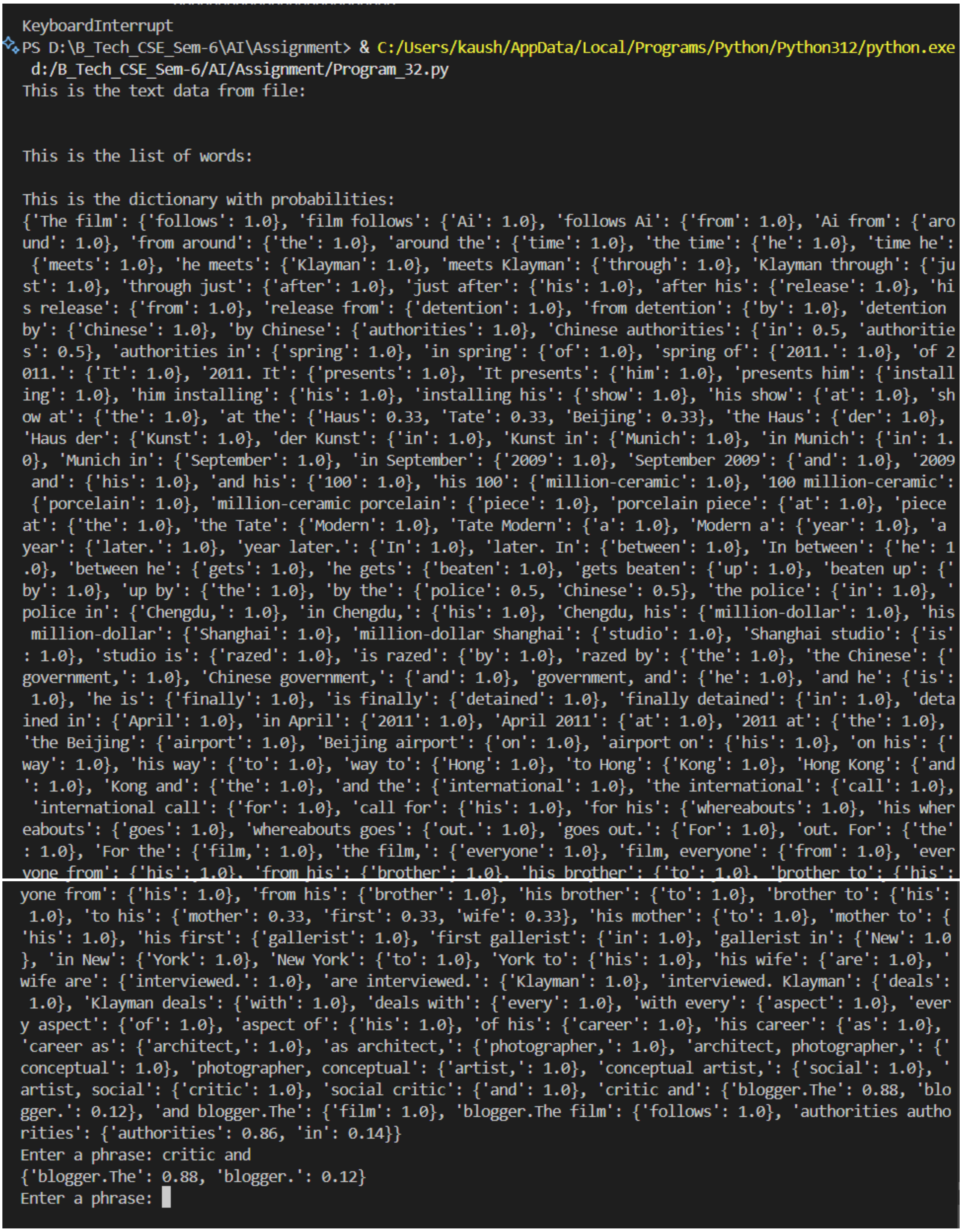
```
'''
@author: 22000409 Kaushal Ramoliya
@description: 32. - Write a program for next word prediction using N-Gram
conditional probability.
'''
file = open("Program_32_data_file.txt", "r", encoding="utf-8")
text = file.read()
file.close()


print("This is the text data from file:")
#print(text)
print("\n")


words = text.split()
print("This is the list of words:")
#print(words)


di = {}


for i in range(len(words) - 2):
  key = words[i] + " " + words[i+1]
  value = words[i+2]


  if key not in di:
    di[key] = {}
    if value in di[key]:
        di[key][value] += 1
    else:
        di[key][value] = 1

for key, value_dict in di.items():
    total_count = sum(value_dict.values())
    for value in value_dict:
        value_dict[value] = round(value_dict[value] / total_count, 2)

print("\nThis is the dictionary with probabilities:")
print(di)

while True:
    user_input = input("Enter a phrase: ")

    if user_input.lower() == "exit":
        break

    print(di.get(user_input, "No matching phrase found"))
```

INPUT: -



```
The film follows Ai from around the time he meets Klayman through just after his release
detention by Chinese authorities in spring of 2011. It presents him installing his show a
Haus der Kunst in Munich in September 2009 and his 100 million-ceramic porcelain piece at
Tate Modern a year later. In between he gets beaten up by the police in Chengdu, his
million-dollar Shanghai studio is razed by the Chinese government, and he is finally deta
in April 2011 at the Beijing airport on his way to Hong Kong and the international call
whereabouts goes out. For the film, everyone from his brother to his mother to his first
gallerist in New York to his wife are interviewed. Klayman deals with every aspect of his
career as architect, photographer, conceptual artist, social critic and blogger.The film
follows Ai from around the time he meets Klayman through just after his release from dete
by Chinese authorities authorities authorities authorities authorities authorities author
authorities in spring of 2011. It presents him installing his show at the Haus der Kunst
Munich in September 2009 and his 100 million-ceramic porcelain piece at the Tate Modern a
later. In between he gets beaten up by the police in Chengdu, his million-dollar Shanghai
studio is razed by the Chinese government, and he is finally detained in April 2011 at th
Beijing airport on his way to Hong Kong and the international call for his whereabouts go
out. For the film, everyone from his brother to his mother to his first gallerist in New
to his wife are interviewed. Klayman deals with every aspect of his career as architect,
```

OUTPUT: -



```
KeyboardInterrupt
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/python.exe
 d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_32.py
This is the text data from file:


This is the list of words:

This is the dictionary with probabilities:
{'The film': {'follows': 1.0}, 'film follows': {'Ai': 1.0}, 'follows Ai': {'from': 1.0}, 'Ai from': {'aro
und': 1.0}, 'from around': {'the': 1.0}, 'around the': {'time': 1.0}, 'the time': {'he': 1.0}, 'time he':
 {'meets': 1.0}, 'he meets': {'Klayman': 1.0}, 'meets Klayman': {'through': 1.0}, 'Klayman through': {'ju
st': 1.0}, 'through just': {'after': 1.0}, 'just after': {'his': 1.0}, 'after his': {'release': 1.0}, 'hi
s release': {'from': 1.0}, 'release from': {'detention': 1.0}, 'from detention': {'by': 1.0}, 'detention
by': {'Chinese': 1.0}, 'by Chinese': {'authorities': 1.0}, 'Chinese authorities': {'in': 0.5, 'authoritie
s': 0.5}, 'authorities in': {'spring': 1.0}, 'in spring': {'of': 1.0}, 'spring of': {'2011.': 1.0}, 'of 2
011.': {'It': 1.0}, '2011. It': {'presents': 1.0}, 'It presents': {'him': 1.0}, 'presents him': {'install
ing': 1.0}, 'him installing': {'his': 1.0}, 'installing his': {'show': 1.0}, 'his show': {'at': 1.0}, 'sh
ow at': {'the': 1.0}, 'at the': {'Haus': 0.33, 'Tate': 0.33, 'Beijing': 0.33}, 'the Haus': {'der': 1.0},
'Haus der': {'Kunst': 1.0}, 'der Kunst': {'in': 1.0}, 'Kunst in': {'Munich': 1.0}, 'in Munich': {'in': 1.
0}, 'Munich in': {'September': 1.0}, 'in September': {'2009': 1.0}, 'September 2009': {'and': 1.0}, '2009
 and': {'his': 1.0}, 'and his': {'100': 1.0}, 'his 100': {'million-ceramic': 1.0}, '100 million-ceramic':
 {'porcelain': 1.0}, 'million-ceramic porcelain': {'piece': 1.0}, 'porcelain piece': {'at': 1.0}, 'piece
at': {'the': 1.0}, 'the Tate': {'Modern': 1.0}, 'Tate Modern': {'a': 1.0}, 'Modern a': {'year': 1.0}, 'a
year': {'later.': 1.0}, 'year later.': {'In': 1.0}, 'later. In': {'between': 1.0}, 'In between': {'he': 1
.0}, 'between he': {'gets': 1.0}, 'he gets': {'beaten': 1.0}, 'gets beaten': {'up': 1.0}, 'beaten up': {'
by': 1.0}, 'up by': {'the': 1.0}, 'by the': {'police': 0.5, 'Chinese': 0.5}, 'the police': {'in': 1.0}, '
police in': {'Chengdu,': 1.0}, 'in Chengdu,': {'his': 1.0}, 'Chengdu, his': {'million-dollar': 1.0}, 'his
 million-dollar': {'Shanghai': 1.0}, 'million-dollar Shanghai': {'studio': 1.0}, 'Shanghai studio': {'is'
: 1.0}, 'studio is': {'razed': 1.0}, 'is razed': {'by': 1.0}, 'razed by': {'the': 1.0}, 'the Chinese': {'
government,': 1.0}, 'Chinese government,': {'and': 1.0}, 'government, and': {'he': 1.0}, 'and he': {'is':
 1.0}, 'he is': {'finally': 1.0}, 'is finally': {'detained': 1.0}, 'finally detained': {'in': 1.0}, 'deta
ined in': {'April': 1.0}, 'in April': {'2011': 1.0}, 'April 2011': {'at': 1.0}, '2011 at': {'the': 1.0},
'the Beijing': {'airport': 1.0}, 'Beijing airport': {'on': 1.0}, 'airport on': {'his': 1.0}, 'on his': {'
way': 1.0}, 'his way': {'to': 1.0}, 'way to': {'Hong': 1.0}, 'to Hong': {'Kong': 1.0}, 'Hong Kong': {'and
': 1.0}, 'Kong and': {'the': 1.0}, 'and the': {'international': 1.0}, 'the international': {'call': 1.0},
 'international call': {'for': 1.0}, 'call for': {'his': 1.0}, 'for his': {'whereabouts': 1.0}, 'his wher
eabouts': {'goes': 1.0}, 'whereabouts goes': {'out.': 1.0}, 'goes out.': {'For': 1.0}, 'out. For': {'the'
: 1.0}, 'For the': {'film,': 1.0}, 'the film,': {'everyone': 1.0}, 'film, everyone': {'from': 1.0}, 'ever
yone from': {'his': 1.0}, 'from his': {'brother': 1.0}, 'his brother': {'to': 1.0}, 'brother to': {'his':
 1.0}, 'to his': {'mother': 0.33, 'first': 0.33, 'wife': 0.33}, 'his mother': {'to': 1.0}, 'mother to': {
'his': 1.0}, 'his first': {'gallerist': 1.0}, 'first gallerist': {'in': 1.0}, 'gallerist in': {'New': 1.0
}, 'in New': {'York': 1.0}, 'New York': {'to': 1.0}, 'York to': {'his': 1.0}, 'his wife': {'are': 1.0}, '
wife are': {'interviewed.': 1.0}, 'are interviewed.': {'Klayman': 1.0}, 'interviewed. Klayman': {'deals':
 1.0}, 'Klayman deals': {'with': 1.0}, 'deals with': {'every': 1.0}, 'with every': {'aspect': 1.0}, 'ever
y aspect': {'of': 1.0}, 'aspect of': {'his': 1.0}, 'of his': {'career': 1.0}, 'his career': {'as': 1.0},
'career as': {'architect,': 1.0}, 'as architect,': {'photographer,': 1.0}, 'architect, photographer,': {'
conceptual': 1.0}, 'photographer, conceptual': {'artist,': 1.0}, 'conceptual artist,': {'social': 1.0}, '
artist, social': {'critic': 1.0}, 'social critic': {'and': 1.0}, 'critic and': {'blogger.The': 0.88, 'blo
gger.': 0.12}, 'and blogger.The': {'film': 1.0}, 'blogger.The film': {'follows': 1.0}, 'authorities autho
rities': {'authorities': 0.86, 'in': 0.14}}
Enter a phrase: critic and
{'blogger.The': 0.88, 'blogger.': 0.12}
Enter a phrase: 
```

CONCLUSION: - This program implements a next-word prediction model using N-Gram conditional probabilities. It processes text data to calculate the likelihood of a word following a given phrase and stores these probabilities in a dictionary. The program demonstrates the use of N-Gram models for predictive text generation, enabling users to query and predict the next word based on input phrases.

PRACTICAL: - 33

AIM: Write an script to build Bag-of-Word and TF-IDF model from English text.

PROGRAM CODE: -

```
'''
@author: 22000409 Kaushal Ramoliya
@description: 33. - Write a script to build Bag-of-Word and TF-IDF model from English text.
'''

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

#Sample English text data
documents = [
    "Natural Language Processing is a fascinating field.",
    "It involves the interaction between computers and human language.",
    "NLP techniques are used in applications like chatbots and sentiment analysis.",
    "The goal is to enable computers to understand and generate human language."
]

#Build Bag-of-Words (BoW) model
print("Bag-of-Words Model:")
vectorizer_bow = CountVectorizer()
bow_matrix = vectorizer_bow.fit_transform(documents)
print("Feature Names:", vectorizer_bow.get_feature_names_out())
print("BoW Matrix:\n", bow_matrix.toarray())

#Build TF-IDF model
print("\nTF-IDF Model:")
vectorizer_tfidf = TfidfVectorizer()
```

```
tfidf_matrix = vectorizer_tfidf.fit_transform(documents)

print("Feature Names:", vectorizer_tfidf.get_feature_names_out())

print("TF-IDF Matrix:\n", tfidf_matrix.toarray())
```

INPUT: -

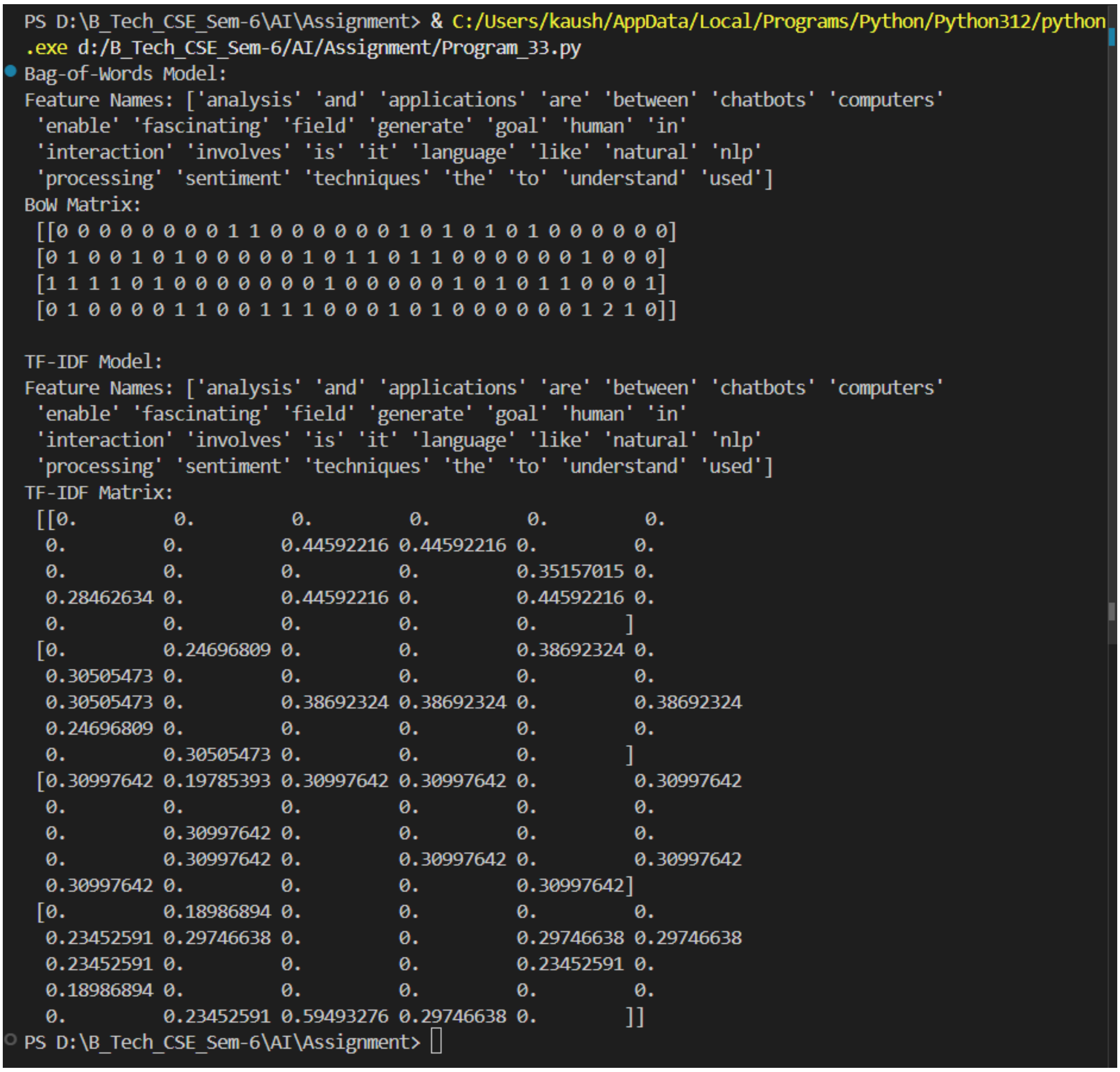
"Natural Language Processing is a fascinating field.",

"It involves the interaction between computers and human language.",

"NLP techniques are used in applications like chatbots and sentiment analysis.",

"The goal is to enable computers to understand and generate human language."

OUTPUT: -



```
PS D:\B_Tech_CSE_Sem-6\AI\Assignment> & C:/Users/kaush/AppData/Local/Programs/Python/Python312/python
.exe d:/B_Tech_CSE_Sem-6/AI/Assignment/Program_33.py
Bag-of-Words Model:
Feature Names: ['analysis' 'and' 'applications' 'are' 'between' 'chatbots' 'computers'
 'enable' 'fascinating' 'field' 'generate' 'goal' 'human' 'in'
 'interaction' 'involves' 'is' 'it' 'language' 'like' 'natural' 'nlp'
 'processing' 'sentiment' 'techniques' 'the' 'to' 'understand' 'used']
BoW Matrix:
 [[0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0]
 [0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0 1 0 0 0]
 [1 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1]
 [0 1 0 0 0 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 0 0 1 2 1 0]]

TF-IDF Model:
Feature Names: ['analysis' 'and' 'applications' 'are' 'between' 'chatbots' 'computers'
 'enable' 'fascinating' 'field' 'generate' 'goal' 'human' 'in'
 'interaction' 'involves' 'is' 'it' 'language' 'like' 'natural' 'nlp'
 'processing' 'sentiment' 'techniques' 'the' 'to' 'understand' 'used']
TF-IDF Matrix:
 [[0.         0.         0.         0.         0.         0.
  0.         0.         0.44592216 0.44592216 0.         0.
  0.         0.         0.         0.         0.35157015 0.
  0.28462634 0.         0.44592216 0.         0.44592216 0.
  0.         0.         0.         0.         0.        ]
 [0.         0.24696809 0.         0.         0.38692324 0.
  0.30505473 0.         0.         0.         0.         0.
  0.30505473 0.         0.38692324 0.38692324 0.         0.38692324
  0.24696809 0.         0.         0.         0.         0.
  0.         0.30505473 0.         0.         0.        ]
 [0.30997642 0.19785393 0.30997642 0.30997642 0.         0.30997642
  0.         0.         0.         0.         0.         0.
  0.         0.30997642 0.         0.         0.         0.
  0.         0.30997642 0.         0.30997642 0.         0.30997642
  0.30997642 0.         0.         0.         0.30997642]
 [0.         0.18986894 0.         0.         0.         0.
  0.23452591 0.29746638 0.         0.         0.29746638 0.29746638
  0.23452591 0.         0.         0.         0.23452591 0.
  0.18986894 0.         0.         0.         0.         0.
  0.         0.23452591 0.59493276 0.29746638 0.        ]]
PS D:\B_Tech_CSE_Sem-6\AI\Assignment>
```

CONCLUSION: - This program builds both Bag-of-Words (BoW) and TF-IDF models from a set of English text documents. It demonstrates how to extract features (unique words) and represent text data as numerical matrices, enabling further analysis or machine learning tasks. The BoW model captures word frequencies, while the TF-IDF model highlights the importance of words relative to the entire corpus.