Words of Concern

Point to be As Allan Bloom has said "Education is the movement from darkness to light". Through this handbook, I have tried to illuminate what might otherwise appear as black boxes to some. In doing so, I have used references from several other authors to synthesize or simplify or elaborate information. This is not possible without omitting details that I deem trivial while dilating the data that I consider relevant to topic. Every effort has been made to avoid errors. In spite of this, some errors might have crept in. Any errors or discrepancies noted maybe brought to my notice which I shall rectify in my next revision.

This handbook is solely for educational purpose and is not for sale. This handbook shall not be reproduced or distributed or used for commercial purposes in any form or by any means.

Thanks,
Deeksha.V.
Interface College of Computer Applications (ICCA)
Davangere

Bibliography

1. Stuart Russel, Peter Norvig: Artificial Intelligence A Modern Approach, 2nd Edition, Pearson Education 2003
2. Tom Mitchell, "Machine Learning", 1st Edition, McGraw-Hill,2017.
3. Elaine Rich, Kevin Knight, Shivashankar B Nair: Artificial Intelligence, Tata McGraw Hill 3rd edition

Part A

1. Write a Program to Implement Breadth First Search using Python.
2. Write a Program to Implement Depth First Search using Python.
3. Write a Program to Implement Tic-Tac-Toe game using Python.
4. Write a Program to Implement 8-Puzzle problem using Python.
5. Write a Program to Implement Water-Jug problem using Python.
6.Write a Program to Implement Travelling Salesman Problem using Python.
7. Write a Program to Implement Tower of Hanoi using Python.
8. Write a Program to Implement Monkey Banana Problem using Python.
9. Write a Program to Implement Alpha-Beta Pruning using Python.
10. Write a Program to Implement 8-Queens Problem using Python.

Part B

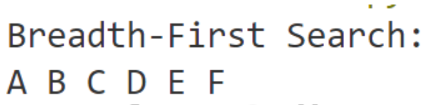
1. Write a program to implement Hill Climbing Algorithm
2. Write a program to implement A* Algorithm
3. Implementation of Python basic Libraries such as Math, Numpy and Scipy
4. Implementation of Python Libraries for ML application such as Pandas and Matplotlib
5. Creation AND Loading different datasets in Python.
6. Write a python program to compute Mean, Median, Mode, Variance and Standard Deviation using Dataset
7.Implementation of Find S Algorithm
8.Implementation of Candidate elimination Algorithm
9. Write a program to implement simple Linear and Plot the graph
10. Write a Python program to implement stemming for a given sentence using NLTK.

Part A

1. Write a Program to Implement Breadth First Search using Python.

```
def bfs(graph, start):
    visited = set()  # Set to keep track of visited nodes
    queue = [start]  # Queue for BFS, initialized with the starting node

    while queue:
        vertex = queue.pop(0)  # Dequeue the next vertex
        if vertex not in visited:
            visited.add(vertex)
            print(vertex, end=" ")  # Print the vertex

            # Enqueue all unvisited neighbors
            for neighbor in graph[vertex]:
                if neighbor not in visited:
                    queue.append(neighbor)

# Example usage
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
start_node = 'A'
print("Breadth-First Search:")
bfs(graph, start_node)
```

Output



```
Breadth-First Search:
A B C D E F
```

2. Write a Program to Implement Depth First Search using Python.

```
def dfs(graph, start):
    visited = set()
    stack = [start]

    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            print(vertex, end=" ")  # Print the visited vertex

            # Add unvisited neighbors to the stack
            for neighbor in graph[vertex]:
                if neighbor not in visited:
                    stack.append(neighbor)

    return visited
```

```
# Example usage
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

start_vertex = 'A'
print("Depth-First Search traversal:")
dfs(graph, start_vertex)
```

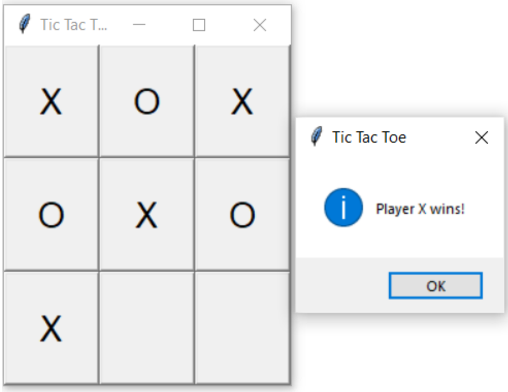
Output

```
Depth-First Search traversal:
A C F B E D
```

3. Write a Program to Implement Tic-Tac-Toe game using Python.

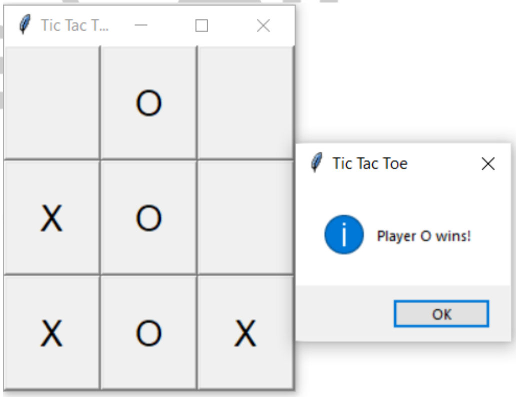
```
import tkinter as tk
from tkinter import messagebox

# Create the main window
root = tk.Tk()
root.title("Tic Tac Toe")

# Global variables
player = "X"
board = [" " for _ in range(9)]

# Function to check for a winner
def check_winner():
    winning_combinations = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],  # Rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8],  # Columns
        [0, 4, 8], [2, 4, 6]              # Diagonals
    ]

    for combo in winning_combinations:
        if board[combo[0]] == board[combo[1]] == board[combo[2]] != " ":
            return board[combo[0]]

    if " " not in board:
        return "Tie"

    return None

# Function to handle button click
def button_click(index):
    global player

    if board[index] == " ":
        board[index] = player
        buttons[index].config(text=player)

        winner = check_winner()
        if winner:
            if winner == "Tie":
                messagebox.showinfo("Tic Tac Toe", "It's a tie!")
            else:
                messagebox.showinfo("Tic Tac Toe", f"Player {winner} wins!")
            reset_game()
        else:
            player = "O" if player == "X" else "X"

# Function to reset the game
def reset_game():
    global board, player
    board = [" " for _ in range(9)]
    player = "X"
    for button in buttons:
        button.config(text=" ")

# Create buttons
buttons = []
for i in range(9):
    button = tk.Button(root, text=" ", font=("Arial", 20), width=4, height=2,
                       command=lambda idx=i: button_click(idx))
    button.grid(row=i // 3, column=i % 3)
    buttons.append(button)
# Start the main loop
root.mainloop()
```

Output



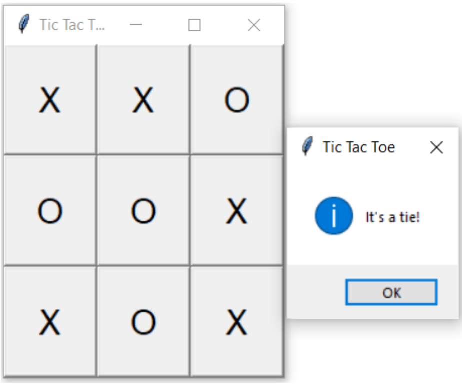
```
X O X
O X O
X
```

Player X wins!



```
  O
X O
X O X
```

Player O wins!



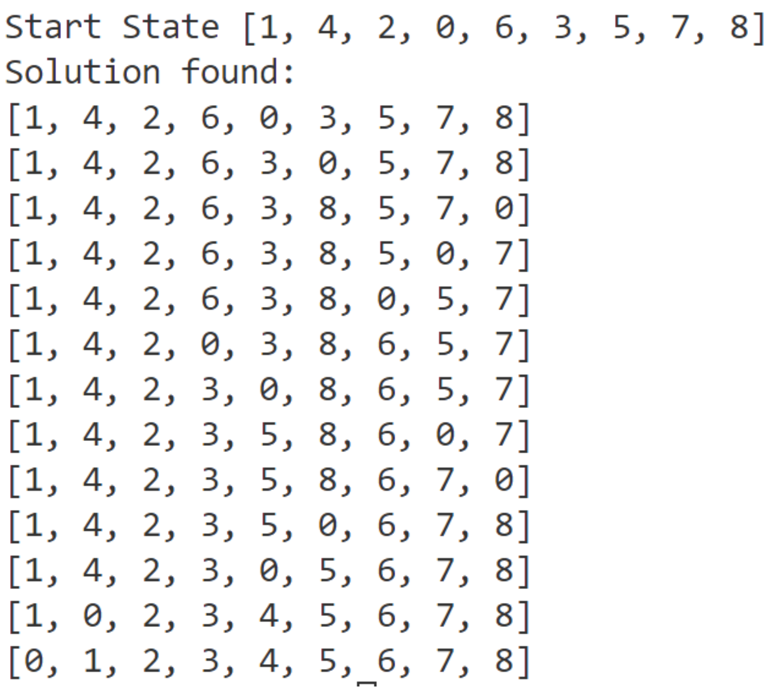
```
X X O
O O X
X O X
```

It's a tie!

4. Write a Program to Implement 8-Puzzle problem using Python.

```
goal_state = (0, 1, 2, 3, 4, 5, 6, 7, 8)

moves = {
    0: (1, 3),
    1: (0, 2, 4),
    2: (1, 5),
    3: (0, 4, 6),
    4: (1, 3, 5, 7),
    5: (2, 4, 8),
    6: (3, 7),
    7: (4, 6, 8),
    8: (5, 7)
}

def solve(start_state):

    queue = [(tuple(start_state),[])
    visited = set([tuple(start_state)])

    while queue:
        current_state, path = queue.pop(0)

        if current_state == goal_state:
            return path

        zero_idx = current_state.index(0)
        for move in moves[zero_idx]:
            new_state = list(current_state)
            new_state[zero_idx], new_state[move] = new_state[move],
                                                               new_state[zero_idx]
            new_state = tuple(new_state)

            if new_state not in visited:
                visited.add(new_state)
                new_path = path + [list(new_state)]
                queue.append((new_state, new_path))

    return None

start_state = [1,4,2,0,6,3,5,7,8]
solution = solve(start_state)
if solution:
    print(f"Start State {start_state}")
    print("Solution found:")
    for state in solution:
        print(state)
else:
    print("No solution found.")
```

Output

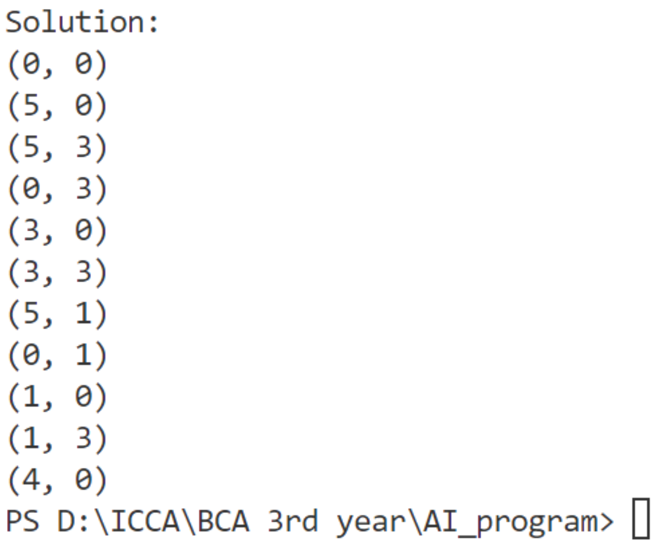


```
Start State [1, 4, 2, 0, 6, 3, 5, 7, 8]
Solution found:
[1, 4, 2, 6, 0, 3, 5, 7, 8]
[1, 4, 2, 6, 3, 0, 5, 7, 8]
[1, 4, 2, 6, 3, 8, 5, 7, 0]
[1, 4, 2, 6, 3, 8, 5, 0, 7]
[1, 4, 2, 6, 3, 8, 0, 5, 7]
[1, 4, 2, 0, 3, 8, 6, 5, 7]
[1, 4, 2, 3, 0, 8, 6, 5, 7]
[1, 4, 2, 3, 5, 8, 6, 0, 7]
[1, 4, 2, 3, 5, 8, 6, 7, 0]
[1, 4, 2, 3, 5, 0, 6, 7, 8]
[1, 4, 2, 3, 0, 5, 6, 7, 8]
[1, 0, 2, 3, 4, 5, 6, 7, 8]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

5. Write a Program to Implement Water-Jug problem using Python.

```
def water_jug_solver(jug1, jug2, target):
    def solve(amt1, amt2):
        if (amt1, amt2) == (0, target) or (amt1, amt2) == (target, 0):
            path.append((amt1, amt2))
            return True
        if (amt1, amt2) in visited:
            return False
        visited.add((amt1, amt2))
        # Try all possible operations
        operations = [
            (jug1, amt2),   # Fill jug1
            (amt1, jug2),   # Fill jug2
            (0, amt2),      # Empty jug1
            (amt1, 0),      # Empty jug2
            (amt1 + min(amt2, jug1 - amt1), amt2 - min(amt2, jug1 - amt1)),  # Transfer
from jug2 to jug1
            (amt1 - min(amt1, jug2 - amt2), amt2 + min(amt1, jug2 - amt2))   # Transfer
from jug1 to jug2
        ]
        for new_amt1, new_amt2 in operations:
            if solve(new_amt1, new_amt2):
                path.append((amt1, amt2))
                return True
        return False
    visited = set()
    path = []
    if solve(0, 0):
        return path[::-1]
    else:
        return None
# Example usage
jug1_capacity = 5
jug2_capacity = 3
target_amount = 4
solution = water_jug_solver(jug1_capacity, jug2_capacity, target_amount)
if solution:
    print("Solution:")
    for step in solution:
        print(step)
else:
    print("No solution found.")
```

Output

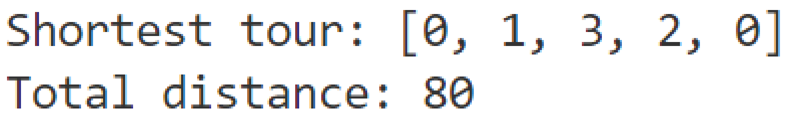


```
Solution:
(0, 0)
(5, 0)
(5, 3)
(0, 3)
(3, 0)
(3, 3)
(5, 1)
(0, 1)
(1, 0)
(1, 3)
(4, 0)
PS D:\ICCA\BCA 3rd year\AI_program> 
```

6.Write a Program to Implement Travelling Salesman Problem using Python.

```
import itertools
def tsp_brute_force(distances, start=0):
    n = len(distances)
    cities = list(range(n))
    best = min(
        ([start] + list(p) + [start] for p in itertools.permutations(cities[1:])),
        key=lambda tour: sum(distances[i][j] for i, j in zip(tour, tour[1:]))
    )
    return best, sum(distances[i][j] for i, j in zip(best, best[1:]))
distances = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]
tour, distance = tsp_brute_force(distances)
print("Shortest tour:", tour)
print("Total distance:", distance)
```

Output

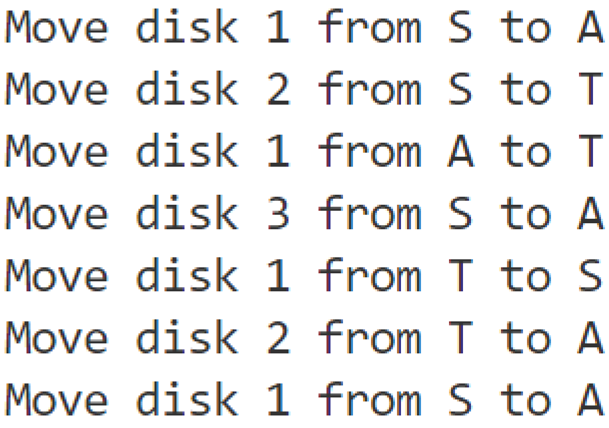


```
Shortest tour: [0, 1, 3, 2, 0]
Total distance: 80
```

7. Write a Program to Implement Tower of Hanoi using Python.

```
def tower_of_hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
        return
    tower_of_hanoi(n-1, source, auxiliary, target)
    print(f"Move disk {n} from {source} to {target}")
    tower_of_hanoi(n-1, auxiliary, target, source)

# Example usage:
num_disks = 4
tower_of_hanoi(num_disks, 'S','A','T')
```

Output

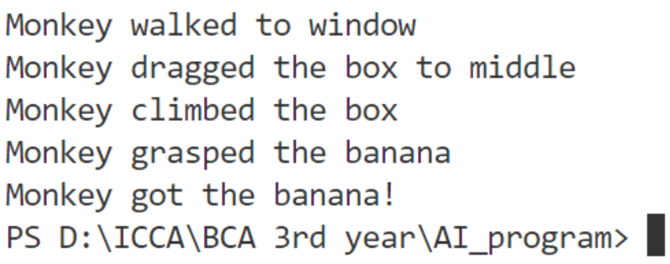


```
Move disk 1 from S to A
Move disk 2 from S to T
Move disk 1 from A to T
Move disk 3 from S to A
Move disk 1 from T to S
Move disk 2 from T to A
Move disk 1 from S to A
```

8. Write a Program to Implement Monkey Banana Problem using Python.

```
class Monkey:
    def __init__(self, position, has_banana=False):
        self.position = position
        self.has_banana = has_banana

    def move(self, new_position):
        self.position = new_position

    def walk(self, new_position):
        if self.position!= new_position:
            print("Monkey walked to", new_position)
            self.move(new_position)
        else:
            print("Monkey is at", new_position)
    def drag(self, box_position, new_position):
        if self.position == box_position:
            print("Monkey dragged the box to", new_position)
            self.move(new_position)
            return True
        return False
    def climb(self, box_position):
        if self.position == box_position:
            print("Monkey climbed the box")
            return True
        return False
    def grasp(self, banana_position):
        if self.position == banana_position and not self.has_banana:
            print("Monkey grasped the banana")
            self.has_banana = True
            return True
        return False

    def has_banana(self):
        return self.has_banana

def solve_monkey_banana_problem(monkey_position, banana_position,
box_position):
    monkey = Monkey(monkey_position)
    monkey.walk(box_position)
    monkey.drag(box_position, banana_position)
    monkey.climb(banana_position)
    monkey.grasp(banana_position)

    if monkey.has_banana:
        print("Monkey got the banana!")
    else:
        print("Monkey didn't get the banana")

# Example usage:
solve_monkey_banana_problem("window", "middle", "window")
```

Output



```
Monkey walked to window
Monkey dragged the box to middle
Monkey climbed the box
Monkey grasped the banana
Monkey got the banana!
PS D:\ICCA\BCA 3rd year\AI_program>
```

9. Write a Program to Implement Alpha-Beta Pruning using Python.

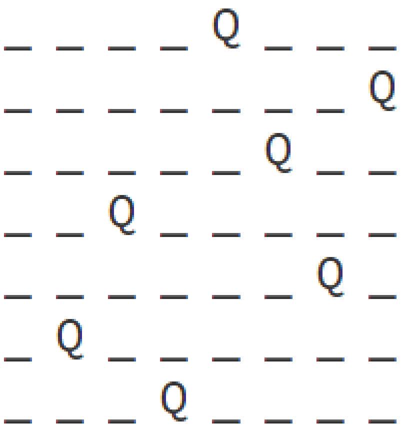
```
def alpha_beta_pruning(node, depth, alpha, beta, maximizing_player):
    if depth == 0 or isinstance(node, int):
        return node
    if maximizing_player:
        max_eval = float('-inf')
        for child in node:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, False)
            max_eval = max(max_eval, eval)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break  # beta cut-off
        return max_eval
    else:
        min_eval = float('inf')
        for child in node:
            eval = alpha_beta_pruning(child, depth - 1, alpha, beta, True)
            min_eval = min(min_eval, eval)
            beta = min(beta, eval)
            if beta <= alpha:
                break  # alpha cut-off
        return min_eval
# Example usage
if __name__ == "__main__":
    # The tree structure is represented as nested lists
    game_tree = [[[3, 5], [6, 9]],[ [1, 2], [0, -1]]]
    # Running alpha-beta pruning
    result = alpha_beta_pruning(game_tree, depth=3, alpha=float('-inf'),
beta=float('inf'), maximizing_player=True)
    print(f"Optimal value: {result}")
```

Output

10. Write a Program to Implement 8-Queens Problem using Python.

```
def solve_queens(n):
    board = [[0]*8 for _ in range(8)]
    solutions = []
    def place_queens(row):
        if row == n:    # Base
            solutions.append(board[:])
            return True
        for col in range(n):
            if is_safe(board, row, col):
                board[row] = col
                if place_queens(row + 1):
                    return True
                board[row] = -1  # Backtrack
        return False  # No solution found for the current row
    def is_safe(board, row, col):
        for i in range(row):
            if board[i] == col or abs(board[i] - col) == row - i:
                return False
        return True
    def print_board(board):
        for row in range(n):
            line = " _" * board[row] + " Q" + " _" * (n - board[row] - 1)
            print(line)
    place_queens(0)
    if not solutions:
        print("No solution exists")
    else:
        for solution in solutions:
            print_board(solution)
            print()
solve_queens(8)
```

<u>Output</u>



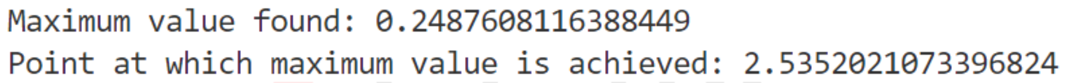
```
 _ _ _ _ Q _ _ _
 _ _ _ _ _ _ _ Q
 _ _ _ _ _ Q _ _
 _ _ Q _ _ _ _ _
 _ _ _ _ _ _ Q _
 _ Q _ _ _ _ _ _
 _ _ _ Q _ _ _ _
```

<u>Part B</u>

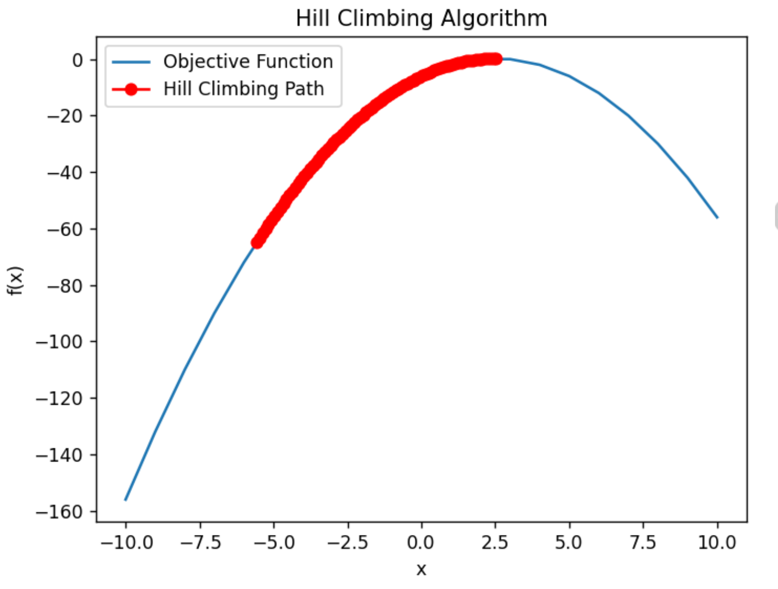
1. Write a program to implement Hill Climbing Algorithm.

```
import random
import matplotlib.pyplot as plt

def objective_function(x):
    return -(x**2 - 5*x + 6)

def hill_climbing(start_point, step_size, max_iterations):
    current_point = start_point
    current_value = objective_function(current_point)
    points = [current_point]
    values = [current_value]

    for _ in range(max_iterations):
        neighbors = [current_point + step_size, current_point - step_size]
        neighbor_values = [objective_function(x) for x in neighbors]

        if max(neighbor_values) > current_value:
            current_point = neighbors[neighbor_values.index(max(neighbor_values))]
            current_value = max(neighbor_values)
            points.append(current_point)
            values.append(current_value)
        else:
            break

    return current_point, current_value, points, values

start_point = random.uniform(-10, 10)
step_size = 0.1
max_iterations = 1000

solution, max_value, points, values = hill_climbing(start_point, step_size,
max_iterations)

print(f"Maximum value found: {max_value}")
print(f"Point at which maximum value is achieved: {solution}")

x_range = range(-10, 11)
y_range = [objective_function(x) for x in x_range]

plt.plot(x_range, y_range, label="Objective Function")
plt.plot(points, values, 'ro-', label="Hill Climbing Path")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Hill Climbing Algorithm")
plt.legend()
plt.show()
```

Output



```
Maximum value found: 0.2487608116388449
Point at which maximum value is achieved: 2.5352021073396824
```

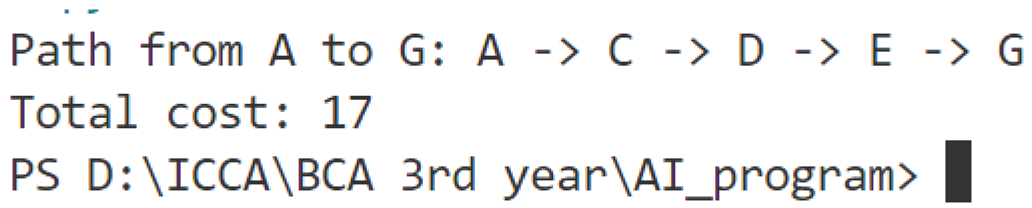


2. Write a program to implement A* Algorithm

```
import heapq

def astar(graph, heuristics, start, goal):
    # Initialize the open list with the start node
    open_list = [(heuristics[start], 0, start, [start])]
    closed_set = set()

    while open_list:
        # Pop the node with the lowest f-score (heuristic + cost) from the open list
        current_fscore, current_cost, current_node, current_path =
heapq.heappop(open_list)

        if current_node == goal:
            # Goal node found, return the path and cost
            return current_path, current_cost

        closed_set.add(current_node)

        # Explore neighbors of the current node
        for neighbor, weight in graph[current_node].items():
            if neighbor in closed_set:
                continue

            new_cost = current_cost + weight
            new_path = current_path + [neighbor]
            new_fscore = new_cost + heuristics[neighbor]

            # Add the neighbor to the open list or update its f-score if already present
            heapq.heappush(open_list, (new_fscore, new_cost, neighbor, new_path))

    # No path found
    return None, None

# Example graph: {node: {neighbor: cost}}
graph = {
    'A': {'B': 4, 'C': 3},
    'B': {'E': 12, 'F': 5},
    'C': {'D': 7, 'E': 10},
    'D': {'E': 2},
    'E': {'G': 5},
    'F': {'G': 16}
}

# Direct heuristic values for each node to the goal 'G'
heuristics = {
    'A': 14,
    'B': 12,
    'C': 11,
    'D': 6,
    'E': 4,
    'F': 11,
    'G': 0
}
start = 'A'
goal = 'G'
path, cost = astar(graph, heuristics, start, goal)
if path:
    print(f"Path from {start} to {goal}: {' -> '.join(path)}")
    print(f"Total cost: {cost}")
```

```
else:
    print("No path found.")
```

Output

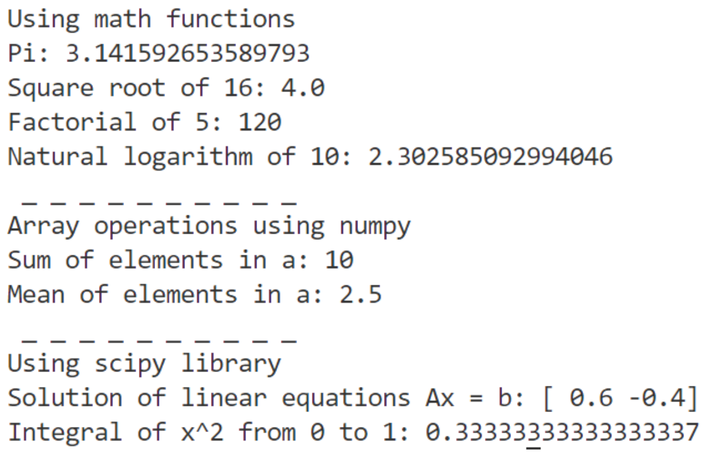


```
Path from A to G: A -> C -> D -> E -> G
Total cost: 17
PS D:\ICCA\BCA 3rd year\AI_program>
```

3. Implementation of Python basic Libraries such as Math, Numpy and Scipy

```
from scipy import linalg         #pip install scipy
from scipy import integrate
import numpy as np
import math

print("Using math functions")
print("Pi:", math.pi)
print("Square root of 16:", math.sqrt(16))
print("Factorial of 5:", math.factorial(5))
print("Natural logarithm of 10:", math.log(10))
print(" _"*10)

A = np.array([[3, 2], [4, 1]])
b = np.array([1, 2])

print("Array operations using numpy")
print("Sum of elements in a:", np.sum(A))
print("Mean of elements in a:", np.mean(A))
print(" _"*10)
print("Using scipy library")
x = linalg.solve(A, b)  # Solving linear equations Ax = b
print("Solution of linear equations Ax = b:", x)

def f(x):   # Integration with scipy.integrate
    return x**2
result, error = integrate.quad(f, 0, 1)
print("Integral of x^2 from 0 to 1:", result)
```

Output

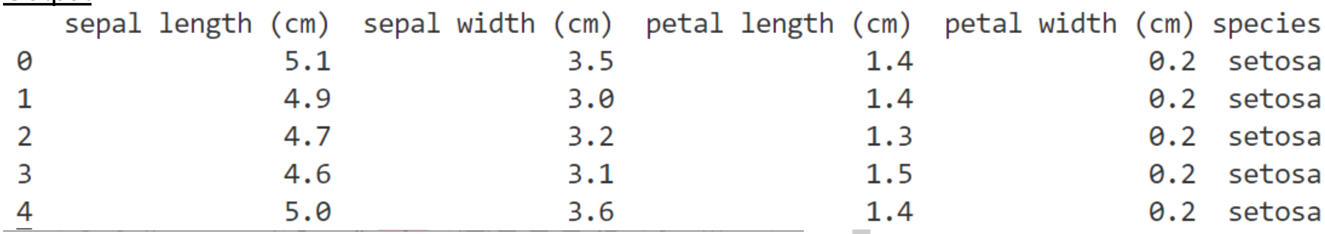


```
Using math functions
Pi: 3.141592653589793
Square root of 16: 4.0
Factorial of 5: 120
Natural logarithm of 10: 2.302585092994046
 _ _ _ _ _ _ _ _ _ _
Array operations using numpy
Sum of elements in a: 10
Mean of elements in a: 2.5
 _ _ _ _ _ _ _ _ _ _
Using scipy library
Solution of linear equations Ax = b: [ 0.6 -0.4]
Integral of x^2 from 0 to 1: 0.33333333333333337
```

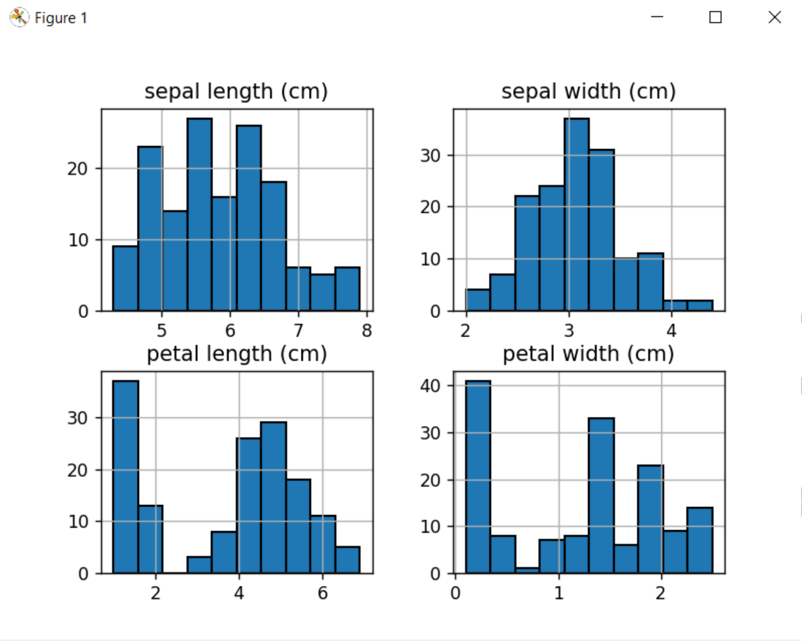
4. Implementation of Python Libraries for ML application such as Pandas and Matplotlib

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load the iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target_names[iris.target]
print(df.head())

# Histograms
df.hist(edgecolor='black', linewidth=1.2)
plt.show()
```

Output

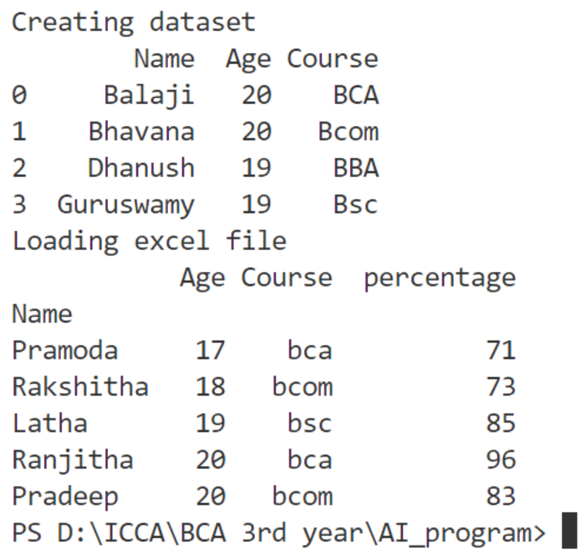


```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm) species
0                5.1               3.5                1.4               0.2  setosa
1                4.9               3.0                1.4               0.2  setosa
2                4.7               3.2                1.3               0.2  setosa
3                4.6               3.1                1.5               0.2  setosa
4                5.0               3.6                1.4               0.2  setosa
```



5. Creation AND Loading different datasets in Python.

```
import pandas as pd
print("Creating dataset")
data = {'Name': ['Balaji', 'Bhavana', 'Dhanush', 'Guruswamy'],
        'Age': [20, 20, 19, 19],
        'Course': ['BCA', 'Bcom', 'BBA', 'Bsc']}

df = pd.DataFrame(data)
print(df)
print("Loading excel file")
book = pd.read_excel('D:\ICCA\BCA 3rd
year\AI_program\Programs\Book1.xlsx',index_col=0)
print(book.tail())
```

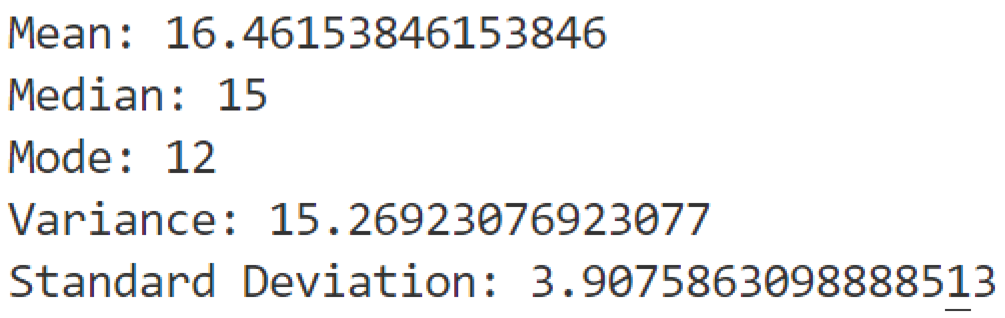


```
Creating dataset
        Name  Age Course
0     Balaji   20    BCA
1    Bhavana   20   Bcom
2    Dhanush   19    BBA
3  Guruswamy   19    Bsc
Loading excel file
           Age Course  percentage
Name
Pramoda     17    bca          71
Rakshitha   18   bcom          73
Latha       19    bsc          85
Ranjitha    20    bca          96
Pradeep     20   bcom          83
PS D:\ICCA\BCA 3rd year\AI_program>
```

6. Write a python program to compute Mean, Median, Mode, Variance and Standard Deviation using Dataset

```
import pandas as pd
import numpy as np
from statistics import mean, median, mode, variance, stdev
data = {
    'values': [12, 15, 12, 18, 21, 12, 15, 21, 18, 22, 15, 12, 21]
}
df = pd.DataFrame(data)
values = df['values']
mean_value = mean(values)
median_value = median(values)
mode_value = mode(values)
variance_value = variance(values)
stdev_value = stdev(values)
print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Mode: {mode_value}")
print(f"Variance: {variance_value}")
print(f"Standard Deviation: {stdev_value}")
```

Output

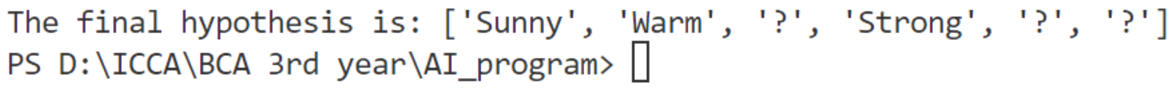


```
Mean: 16.46153846153846
Median: 15
Mode: 12
Variance: 15.26923076923077
Standard Deviation: 3.907586309888513
```

7.Implementation of Find S Algorithm

```
import numpy as np

def find_s_algorithm(training_data):
    # Extract the feature vectors and the labels from the training data
    features = np.array([data[:-1] for data in training_data])
    labels = np.array([data[-1] for data in training_data])

    # Initialize the hypothesis to the most specific hypothesis
    hypothesis = ['0'] * len(features[0])

    # Iterate over the training examples
    for i, instance in enumerate(features):
        if labels[i] == 'Yes':  # Only consider positive examples
            for j, feature in enumerate(instance):
                if hypothesis[j] == '0':  # If hypothesis is most specific, replace it with the
feature value
                    hypothesis[j] = feature
                elif hypothesis[j] != feature:  # If feature values differ, generalize with '?'
                    hypothesis[j] = '?'

    return hypothesis

# Example usage
training_data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
]

hypothesis = find_s_algorithm(training_data)
print("The final hypothesis is:", hypothesis)
```

Output



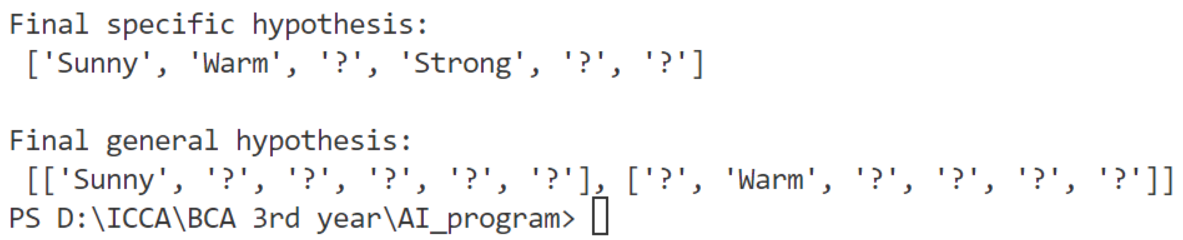
```
The final hypothesis is: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
PS D:\ICCA\BCA 3rd year\AI_program> 
```

8.Implementation of Candidate elimination Algorithm

```
import csv

def candidate_elimination(data):
    # Find the first positive example to initialize S
    for example in data:
        if example[-1] == "Yes":
            s = example[:-1]
            break

    # Initialize G to the most general hypothesis
    g = [['?' for _ in range(len(s))] for _ in range(len(s))]

    for example in data:
        if example[-1] == "Yes":
            for i in range(len(s)):
                if example[i] != s[i]:
                    s[i] = '?'
                    g[i][i] = '?'
        elif example[-1] == "No":
            for i in range(len(s)):
                if example[i] != s[i]:
```

```
                g[i][i] = s[i]
            else:
                g[i][i] = '?'

    # Filter out general hypotheses that are too specific
    gh = [hypothesis for hypothesis in g if any(attribute != '?' for attribute in
hypothesis)]

    return s, gh

# Read the training data from a CSV file
with open("D:\\ICCA\\BCA 3rd year\\AI_program\\Programs\\training_data.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)[1:]  # Skip the header row

# Apply the Candidate Elimination algorithm
specific_hypothesis, general_hypothesis = candidate_elimination(data)

# Print the results
print("\nFinal specific hypothesis:\n", specific_hypothesis)
print("\nFinal general hypothesis:\n", general_hypothesis)
```

Output



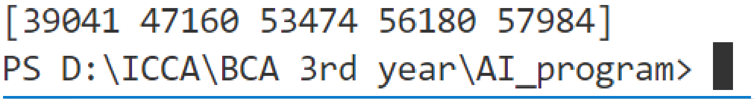
```
Final specific hypothesis:
 ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final general hypothesis:
 [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
PS D:\ICCA\BCA 3rd year\AI_program> 
```

9. Write a program to implement simple Linear and Plot the graph

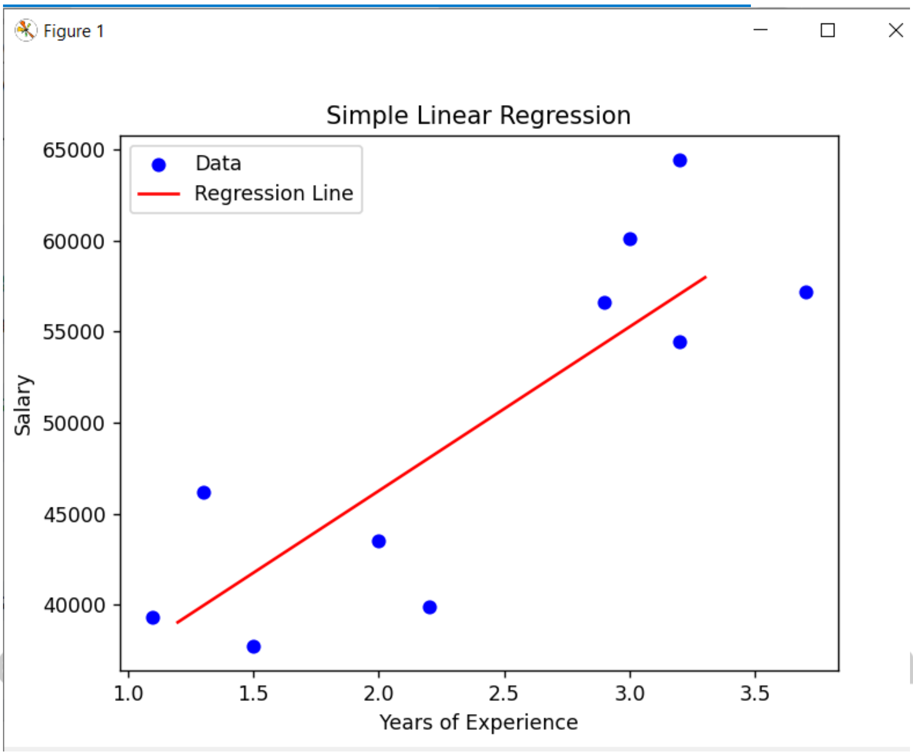
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression #pip install scikit-learn

# Sample dataset
data = {'Experience': [1.1, 1.3, 1.5, 2.0, 2.2, 2.9, 3.0, 3.2, 3.2, 3.7],
        'Salary': [39343, 46205, 37731, 43525, 39891, 56642, 60150, 54445, 64445,
57189]}
df = pd.DataFrame(data)

# Selecting features and target
X = df[['Experience']]
y = df['Salary']

# Train the model
model = LinearRegression()
model.fit(X, y)

# New data for prediction
new_experience = [[1.2], [2.1], [2.8], [3.1], [3.3]]  # 2D array

# Convert new_experience to DataFrame with same column name
new_experience_df = pd.DataFrame(new_experience, columns=['Experience'])
```

```
# Make predictions
y_pred = model.predict(new_experience_df)
print(y_pred)

# Plot the data and regression line
plt.scatter(X, y, color='blue', label='Data')
plt.plot(new_experience_df, y_pred, color='red', label='Regression Line')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

Output



```
[39041 47160 53474 56180 57984]
PS D:\ICCA\BCA 3rd year\AI_program>
```

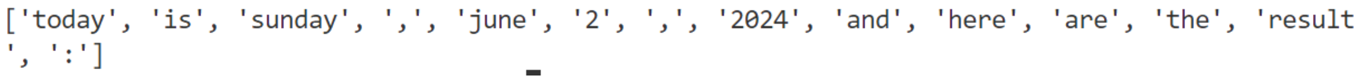


10. Write a Python program to implement stemming for a given sentence using NLTK.

```
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

# Download the required NLTK resources
nltk.download('punkt')
# Create an object of class PorterStemmer
porter = PorterStemmer()
# Input sentence
sentence = "Today is Sunday, June 2, 2024 and here are the results:"
# Tokenize the sentence into words
```

```
        words = word_tokenize(sentence)
        # Stem each word
        stemmed_words = [porter.stem(word) for word in words]
        # Print the stemmed words
        print(stemmed_words)
```

Output



```
['today', 'is', 'sunday', ',', 'june', '2', ',', '2024', 'and', 'here', 'are', 'the', 'result
', ':']
```