Assignment 5: Practical Application of DB-SCAN and A* Algorithm.

Instructions: This assignment engages you in the practical application of DB-SCAN and A* Algorithm. Follow the steps carefully, and submit your work as a single PDF file that includes all required code and graphs. **Attach** supplementary code files (.py and .ipynb) as used in this assignment.

Objective: Learn DB-SCAN and A* Algorithm and understand the effects of DB-SCAN and A* Algorithm in any problem.

Step-by-Step Instructions

Q1. DB-SCAN

Step 1: Visualize Data Before Clustering

Using the provided code,

```
# ====================== Initial ================================
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

# Generate sample data
centers = [[2, 2], [-2, -2], [2, -2], [-2, 2], [0, 0]]
X, _ = make_blobs(n_samples= 1250, centers=centers, cluster_std=0.55,
random_state=2)
X = StandardScaler().fit_transform(X)

# Plotting before clustering
plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], color='gray', marker='o')
plt.title('Sample Data Before Clustering')
plt.grid(True)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
# ============================================================
```

- Create a graph to show how your data looks before you apply clustering. This helps you see what the data looks like normally.
- **Attach your plotting code** with your report.

Step 2: Implement K-means Clustering and Determine Optimal Clusters

- Use the K-means algorithm to find groups in your data. To figure out the best number of groups, use the elbow method: **plot** the within-cluster sum of squares (WCSS) against different numbers of clusters and look for the point where the line starts to flatten out.
- **Attach your K-means and elbow method code** with your report.

Elbow Method Example Code:

==========

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

# Calculate WCSS for different numbers of clusters
wcss = []
for i in range(1, 11):  # Testing 1 to 10 clusters
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)  # inertia_ is the WCSS for the model

# Plotting the WCSS to observe the 'Elbow'
plt.figure(figsize=(10, 8))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.grid(True)
plt.show()
```

==========

Step 3: Visualize Data After Clustering

- Make another graph to show your data with the groups found by best K-means with K value from step2. This shows how the algorithm has organized the data.
- **Attach your post-clustering plotting code** with your report.

Step 4: Applied data from step1 with DB-SCAN

- Use the DB-SCAN algorithm to identify clusters in your data. Set the epsilon parameter to 0.19 and the minimum number of points required to form a dense region (min_samples) to 9.
- Make another graph to show your data with the groups found by DB-SCAN. This shows how the algorithm has organized the data.
- **Attach your post-clustering plotting code** with your report.

Step 5: Interpret Your Results

- Explain the different the results from the K-means clustering show about your data and DBSCAN. What is pros and cons of each algorithm.
- **Attach your interpretation and any additional code** you used to analyze the results.

Q2. A* algrorithm

Task: Implement the A Algorithm for the 8-Puzzle Problem*

STEP 1:

Task Setup:

- Define a Node class to encapsulate the **state of the puzzle**, its **parent state**, and the cost: g, h, and f.
    - **g (cost):** Represents the cost from the start node to the current node, where each move incurs a cost of one.
    - **h (heuristic):** The estimated cost from the current node to the goal, calculated using the Manhattan Distance between the current state and the goal state.
    - **f (total cost):** The sum of g and h, determining the path priority in the A* algorithm.

Implementation Details:

- Start with the initial puzzle state: ((1, 2, 3), (4, 0, 6), (7, 5, 8)).
- Set the goal state as: ((1, 2, 3), (4, 5, 6), (7, 8, 0)).
- Implement the A* search to explore paths, expand nodes based on the lowest f value, and efficiently find the shortest path to the goal state.
- Attach your python code and your result that your get.

Main Function for Running:

```
def main():
    """Main function to run A* algorithm on a given 8-puzzle problem."""
    start = (
        (1, 2, 3),
        (4, 0, 6),
        (7, 5, 8)
    )

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

```
    path, total_cost = astar(start, end)

    if path:
        print(f"Solution found in {len(path) - 1} moves with total cost
{total_cost}:\n")
        for i, board in enumerate(path):
            print(f"Move {i}:")
            print_board(board)
    else:
        print("No solution found.")

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

Example code for Manhattan distance:

```
def manhattan_distance(start_state, goal_state):
    """Calculate the Manhattan Distance heuristic for the 8-puzzle."""
    distance = 0
    for num in range(1, 9):  # Loop through numbers 1 to 8 (tiles of the 8-
puzzle)
        # Find the position of the number in the start state
        x1, y1 = -1, -1
        for ix, row in enumerate(start_state):
            for iy, value in enumerate(row):
                if value == num:
                    x1, y1 = ix, iy
                    break
            if x1 != -1:
                break

        # Find the position of the number in the goal state
        x2, y2 = -1, -1
        for ix, row in enumerate(goal_state):
            for iy, value in enumerate(row):
                if value == num:
                    x2, y2 = ix, iy
                    break
            if x2 != -1:
                break

        # Calculate the Manhattan distance for this number
        distance += abs(x1 - x2) + abs(y1 - y2)
    return distance
```

Result Expectation:

```
Move 0:
1 2 3
4   6
7 5 8

Move 1:
1 2 3
4 5 6
7   8

Move 2:
1 2 3
4 5 6
7 8
```

STEP 2:

Testing:

Test the A* algorithm with alternative start points, such as: ((2, 5, 4), (0, 8, 3), (1, 7, 6)).

```python
def main():
    """Main function to run A* algorithm on a given 8-puzzle problem."""
    start = (
        (2, 5, 4),
        (0, 8, 3),
        (1, 7, 6)
    )
```

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

    path, total_cost = astar(start, end)

    if path:
        print(f"Solution found in {len(path) - 1} moves with total cost
{total_cost}:\n")
        for i, board in enumerate(path):
            print(f"Move {i}:")
            print_board(board)
    else:
        print("No solution found.")

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

Result Expectation:

- Document how many moves were needed to reach the goal state for each test case.
- Include all code snippets, testing scenarios, and results in your report.

Submission Requirements:

- Submit your completed assignment as a PDF containing all necessary code snippets and graphs.
- Extensive explanations or theoretical discussions should be included in the main text of your PDF, not in code comments.
- Attach corresponding .py and .ipynb files containing executable code.
- Ensure all files are clearly labeled and organized.
- Late submissions will be subject to deductions according to the policy.

Evaluation Criteria: Your assignment will be graded based on the accuracy of your implementation, the clarity of your presentation, and the completeness of your submission. Specific attention will be given to how well you follow these guidelines:

- **Step-by-Step Explanation:** This assignment requires clear, step-by-step explanations. You must explicitly label and explain each step (e.g., Step 1, Step 2, Step 3, Step 4). Failure to provide detailed explanations for each required step will result in a proportional deduction in your score. For example, if the assignment requires four steps and you complete only three, this will result in a score of 75 out of 100.