Name : Mozeb Ahmed Khan

Roll No: 20F-0161

Sec: BS(CS)-6A

Assignment: 05

Course: Artificial Intelligence

Question 1:

AI Assignment # 05

Question # 01

K-Means Clustering

Answer :-

⇒ Let us select two centroids randomly.

$C1 = (4, 3)$, $C2 = (7, 8)$.

⇒ Using Manhattan distance, calculate distance of all data points using the centroid C1,

1) $C1 = (4, 3)$

$P_1 = (2, 3)$, $D_1 = |(4-2)| + |(3-3)| = 2 - 0 = 2$

$P_2 = (3, 4)$, $D_2 = |(4-3)| + |(3-4)| = 1 + 1 = 2$

$P_3 = (5, 6)$, $D_3 = |(4-5)| + |(3-6)| = 1 + 3 = 4$

$P_4 = (6, 7)$, $D_4 = |(4-6)| + |(3-7)| = 2 + 4 = 6$

$P_5 = (8, 9)$, $D_5 = |(4-8)| + |(3-9)| = 4 + 6 = 10$

⇒ Using Manhattan distance, calculate distance of all data points using the centroid C2,

2) $C2 = (7, 8)$

$P_1 = (2, 3)$, $D_1 = |(2-7)| + |(3-8)| = 5 + 5 = 10$

$P_2 = (3, 4)$, $D_2 = |(3-7)| + |(4-8)| = 4 + 4 = 8$

$P_3 = (5,6)$, $D_3 = |(5-7)|+|(6-8)| = 2+2 = 4$

$P_4 = (6,7)$, $D_4 = |(6-7)|+|(7-8)| = 1+1 = 2$

$P_5 = (8,9)$, $D_5 = |(8-7)|+|(9-8)| = 1+1 = 2$

Result:- ⇒ For $C1 = (4,3)$, we have $(2,3)$, $(3,4)$

⇒ For $C2 = (7,8)$, we have $(5,6)$, $(6,7)$, $(8,9)$.

Updated Centroids:-

$C_1 = \left(\frac{2+3}{2}, \frac{3+4}{2}\right) = \left(\frac{5}{2}, \frac{7}{2}\right) = (2.5, 3.5)$.

$C_2 = \left(\frac{5+6+8}{3}, \frac{6+7+9}{3}\right) = \left(\frac{19}{3}, \frac{22}{3}\right) = (6.33, 7.33)$.

$C_1 = (2.5, 3.5)$, $C_2 = (6.33, 7.33)$

⇒ Using manhattan distance, calculate distance of all the data points of from centroid ~~C1(2.5,3.5)~~ $C_2 = (6.33, 7.33)$

$P_1 = (2,3)$, $D_1 = |(6.33-2)| + |(7.33-3)| = 4.33 + 4.33 = 8.66.$

$P_2 = (3,4)$, $D_2 = |(6.33-3)| + |(7.33-4)| = 3.33 + 3.33 = 6.66$

$P_3 = (5,6)$, $D_3 = |(6.33-5)| + |(7.33-6)| = 1.33 + 1.33 = 2.66.$

$P_4 = (6,7)$, $D_4 = |(6.33-6)| + |(7.33-7)| = 0.33 + 0.33 = 0.66.$

$P_5 = (8,9)$, $D_5 = |(6.33-8)| + |(7.33-9)| = 1.66 + 1.66 = 3.33.$

=) Using manhattan distance, calculate distance of all the data points ~~of~~ from centroid $C_1 = (2.5, 3.5)$.

$P_1 = (2,3)$, $D_1 = |(2.5-2)| + |(3.5-3)| = 0.5 + 0.5 = 1.$

$P_2 = (3,4)$, $D_2 = |(2.5-3)| + |(3.5-4)| = 0.5 + 0.5 = 1.$

$P_3 = (5,6)$, $D_3 = |(2.5-5)| + |(3.5-6)| = 2.5 + 2.5 = 5.$

$P_4 = (6,7)$, $D_4 = |(2.5-6)| + |(3.5-7)| = 3.5 + 3.5 = 7.$

$P_5 = (8,9)$, $D_5 = |(2.5-8)| + |(3.5-9)| = 5.5 + 5.5 = 11.$

Result:-

=) For $C1 = (2.5, 3.5)$, we have $(2,3), (3,4)$.

=) For $C2 = (6.33, 7.33)$, we have $(5,6), (6,7), (8,9)$.

Updated Centroids:-

$$C_1 = \left(\frac{2+3}{2}, \frac{3+4}{2}\right) = \left(\frac{5}{2}, \frac{7}{2}\right) = (2.5, 3.5).$$

$$C_2 = \left(\frac{5+6+8}{2}, \frac{6+7+9}{2}\right) = \left(\frac{19}{3}, \frac{22}{3}\right) = (6.33, 7.33)$$

=) As centroids remain same, these are final centroids.

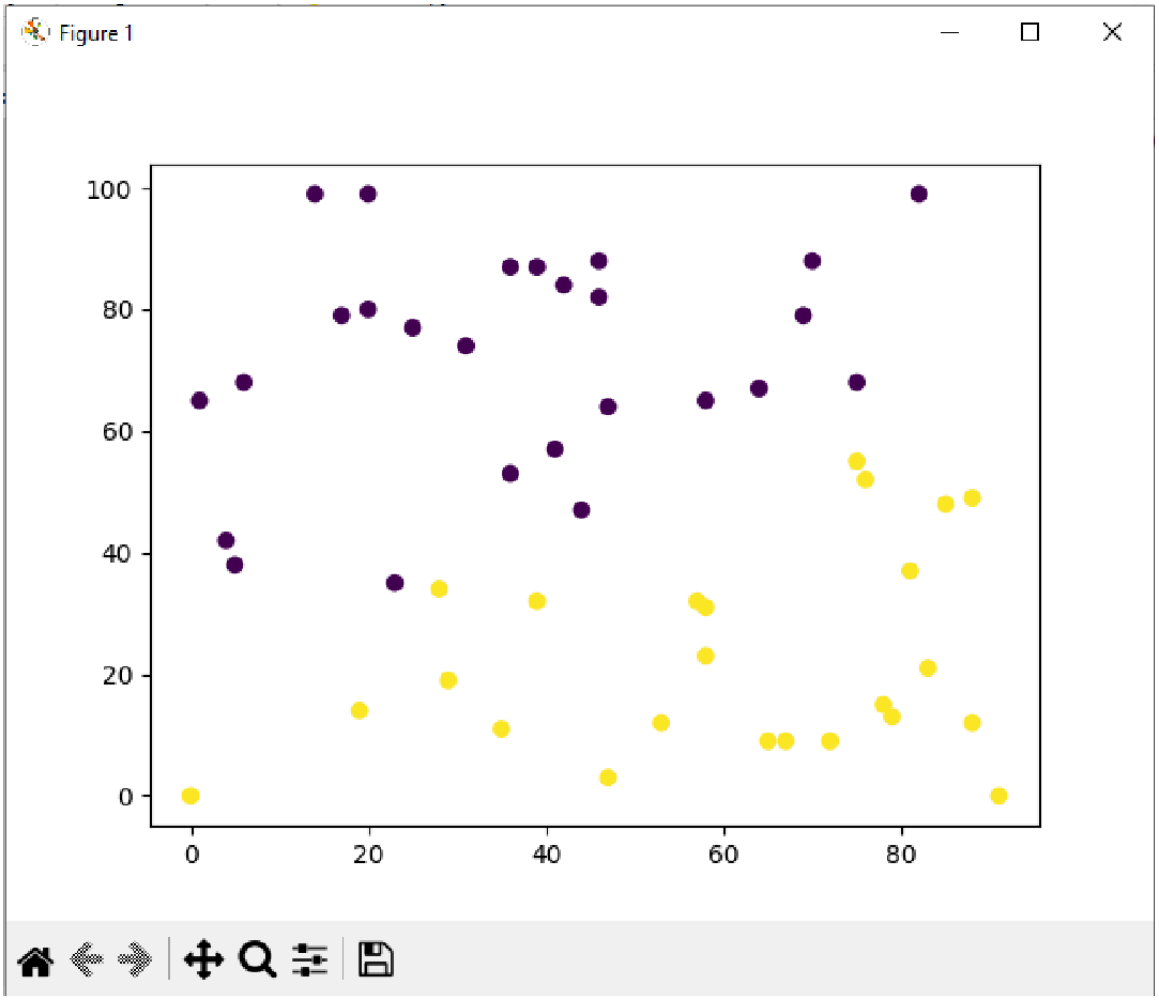
Question 2:

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

def kmeans_clustering(X, k, max_iterations=100):
    # convert data to numpy array
    X = np.array(X)
    # randomly choose k initial centroids
    centroids = X[np.random.choice(X.shape[0], k, replace=False)]
    # initialize cluster labels
    cluster_labels = np.zeros(X.shape[0], dtype=np.int32)

    for _ in range(max_iterations):
        # assign data points to the nearest centroid
        for i in range(X.shape[0]):
            distances = np.linalg.norm(X[i] - centroids, axis=1)
            cluster_labels[i] = np.argmin(distances)

        # calculate new centroids based on assigned data points
        new_centroids = np.empty((k, X.shape[1]))
        for j in range(k):
            new_centroids[j] = np.mean(X[cluster_labels == j], axis=0)

        # check if centroids have converged
        if np.allclose(centroids, new_centroids):
            break

        # update centroids
        centroids = new_centroids

    return cluster_labels

# read data and preprocess
df = pd.read_csv('data.csv', usecols=[0,1])
```

```python
X = df.values
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# perform k-means clustering
num_clusters = 2
cluster_labels = kmeans_clustering(X_scaled, num_clusters)

# inverse transform scaled data for visualization
X_unscaled = scaler.inverse_transform(X_scaled)

# plot data points colored by their assigned cluster
plt.scatter(X_unscaled[:, 0], X_unscaled[:, 1], c=cluster_labels)

# show plot
plt.show()
```

Output:



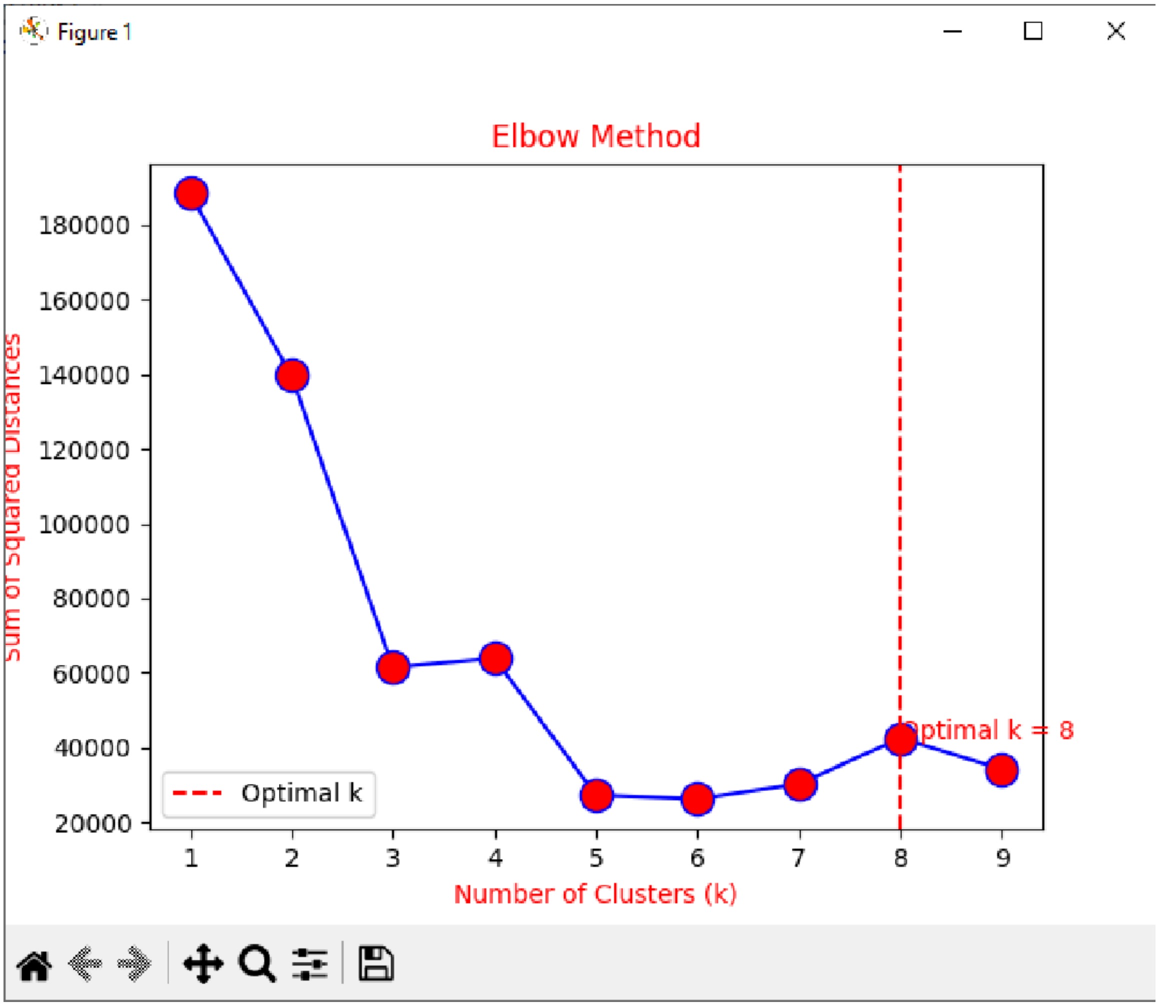
Question 3:

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset from CSV file
df = pd.read_csv('data.csv', usecols=[0, 1])

# Convert the dataset to a NumPy array
data = df.values
```

```python
# Define the range of k values to test
k_range = range(1, 10)

# Initialize an empty list to store the sum of squared errors
sse_list = []
for k in k_range:
    centroids = data[np.random.choice(range(len(data)), k)]
    # Assign each data point to its nearest centroid
    distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=-1)
    cluster_ids = np.argmin(distances, axis=-1)
    # Calculate the sum of squared errors for the current k
    sse = np.sum((data - centroids[cluster_ids]) ** 2)
    sse_list.append(sse)

# Plot the SSE values for different k values
plt.plot(k_range, sse_list, color='blue', linestyle='-', marker='o',
markerfacecolor='red', markersize=13)
plt.xlabel('Number of Clusters (k)', color='red')
plt.ylabel('Sum of Squared Distances', color='red')
plt.title('Elbow Method', color='red')

def find_optimal_k(k_range, sse_list):
    # Calculate the differences between consecutive SSE values
    sse_diffs = np.diff(sse_list)
    # Calculate the optimal k value as the point of maximum curvature
    optimal_k = np.argmax(sse_diffs) + 2
    return optimal_k

def display_graph(k_range, sse_list):
    optimal_k = find_optimal_k(k_range, sse_list)
    plt.axvline(x=optimal_k, color='red', linestyle='--', label='Optimal k')
    plt.legend()
    plt.text(optimal_k, sse_list[optimal_k - 1], f'Optimal k = {optimal_k}',
color='red')
    plt.show()

display_graph(k_range, sse_list)
```

Output:



Question 4:

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from Ass5Task3 import find_optimal_k, k_range, sse_list
from sklearn.preprocessing import StandardScaler

num_clusters = find_optimal_k(k_range, sse_list)

def kmeans_clustering(data, k=10, max_iterations=100):
    # convert data to numpy array
```

```python
    X = np.array(data)
    # randomly choose k initial centroids
    centroids = X[np.random.choice(X.shape[0], k, replace=False)]
    # initialize cluster labels
    cluster_labels = np.zeros(X.shape[0], dtype=np.int32)

    for _ in range(max_iterations):
        # assign data points to the nearest centroid
        for i in range(X.shape[0]):
            distances = np.linalg.norm(X[i] - centroids, axis=1)
            cluster_labels[i] = np.argmin(distances)

        # calculate new centroids based on assigned data points
        new_centroids = np.empty((k, X.shape[1]))
        for j in range(k):
            new_centroids[j] = np.mean(X[cluster_labels == j], axis=0)

        # check if centroids have converged
        if np.allclose(centroids, new_centroids):
            break

        # update centroids
        centroids = new_centroids

    return cluster_labels

# read data and preprocess
df = pd.read_csv('data.csv', usecols=[0,1])
data = df.values
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# perform k-means clustering
cluster_labels = kmeans_clustering(data_scaled, num_clusters)

# inverse transform scaled data for visualization
data_unscaled = scaler.inverse_transform(data_scaled)

# plot data points colored by their assigned cluster
plt.scatter(data_unscaled[:, 0], data_unscaled[:, 1], c=cluster_labels)
```

```
# show plot
plt.show()
```

Output:

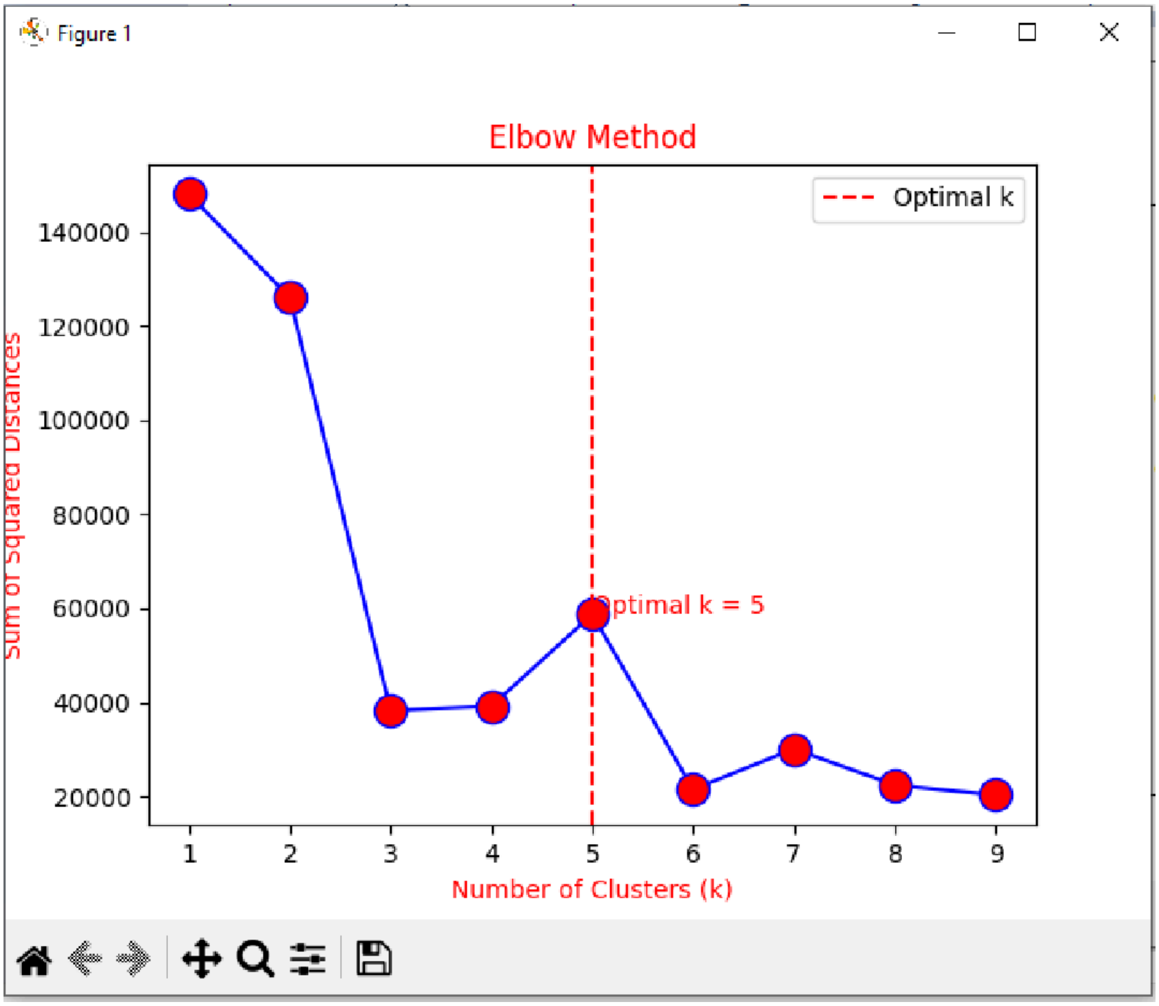
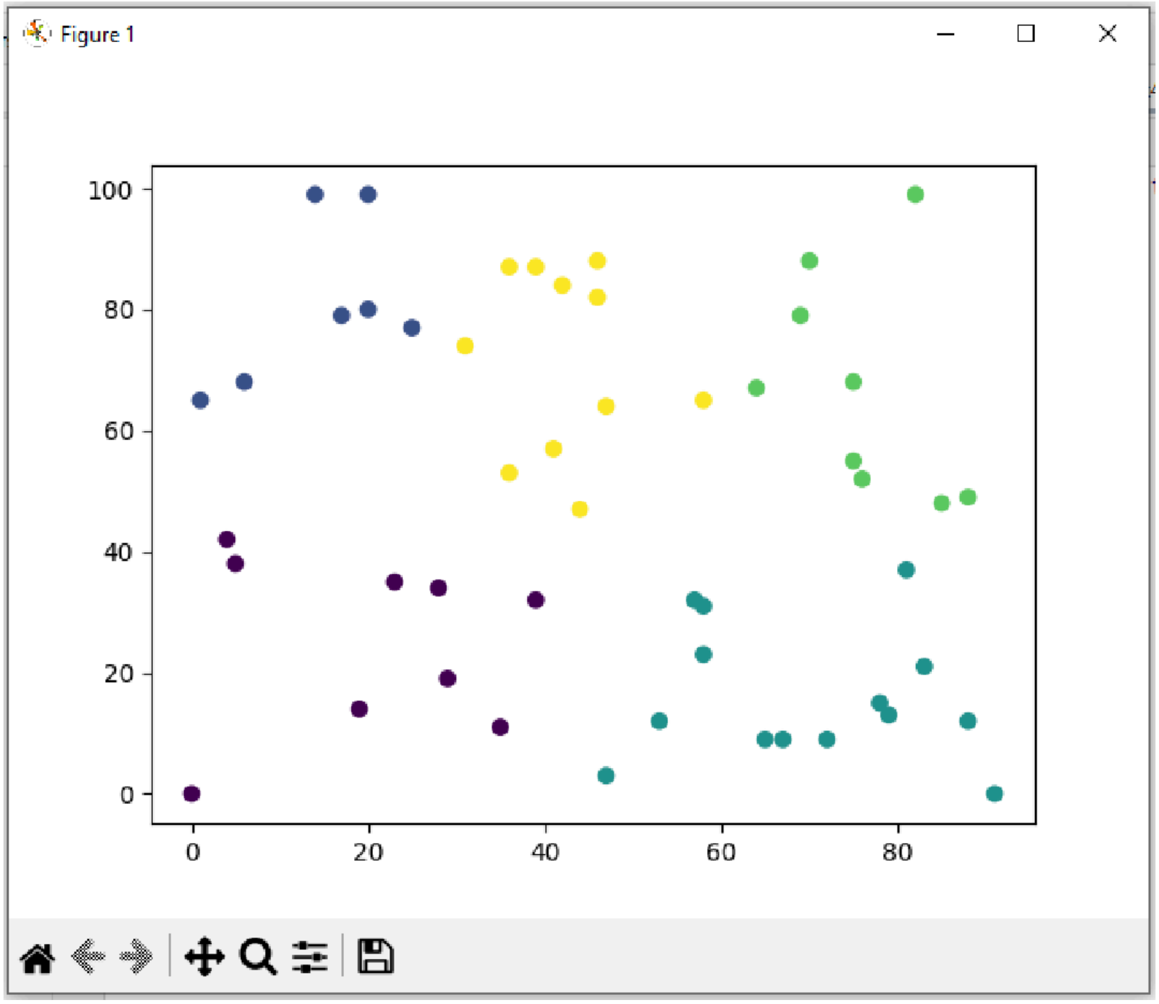


Figure 1
Elbow Method
Optimal k
Optimal k = 5
Sum of Squared Distances
Number of Clusters (k)
140000
120000
100000
80000
60000
40000
20000
1
2
3
4
5
6
7
8
9



The End.

Thank You.