Weekly Assignment Report

Question 1:

Python

```python
# The following code implements the fuzzy c-means clustering algorithm.

# Fuzzy C-Means is a soft clustering algorithm where each data point can belong to multiple clusters with varying degrees (membership values).

# The function fuzzy_c_means performs the clustering by iteratively updating the membership values and cluster centers until convergence.



def fuzzy_c_means(X, Y, C1, C2, m=2, max_iter=100, tol=1e-5):

    """

    Fuzzy C-Means clustering algorithm.



    This algorithm clusters the data points (X, Y) into two clusters, with each point having a membership value for both clusters (C1 and C2).

    The clustering process is based on minimizing the distance between each point and the cluster centers, with a fuzziness parameter (m)

    controlling the degree of membership. The algorithm converges when the membership values stop changing significantly.



    Parameters:

    X : list - List of x-coordinates of the data points.

    Y : list - List of y-coordinates of the data points.
```

```
    C1 : list - List of initial membership values for cluster 1.

    C2 : list - List of initial membership values for cluster 2.

    m : int - Fuzziness parameter (default is 2).

    max_iter : int - Maximum number of iterations (default is 100).

    tol : float - Tolerance for convergence (default is 1e-5).


    Returns:

    C1 : list - Final membership values for cluster 1 after convergence.

    C2 : list - Final membership values for cluster 2 after convergence.

    """


    # Initialize cluster centers based on weighted averages of the data points,
weighted by the initial membership values.

    C1_center = [

        sum(x * c**m for x, c in zip(X, C1)) / sum(c**m for c in C1),

        sum(y * c**m for y, c in zip(Y, C1)) / sum(c**m for c in C1)

    ]

    C2_center = [

        sum(x * c**m for x, c in zip(X, C2)) / sum(c**m for c in C2),

        sum(y * c**m for y, c in zip(Y, C2)) / sum(c**m for c in C2)

    ]


    # Iterate through the maximum number of iterations or until convergence.
```

```
    for _ in range(max_iter):
        # Update membership values for each data point based on its distance to the current cluster centers.
        new_C1 = []
        new_C2 = []

        for i in range(len(X)):
            d1 = ((X[i] - C1_center[0])**2 + (Y[i] - C1_center[1])**2)**0.5  # Distance to cluster 1 center
            d2 = ((X[i] - C2_center[0])**2 + (Y[i] - C2_center[1])**2)**0.5  # Distance to cluster 2 center

            # Handle division by zero in case of a perfect match with the center.
            if d1 == 0:
                new_C1.append(1)
                new_C2.append(0)
            elif d2 == 0:
                new_C1.append(0)
                new_C2.append(1)
            else:
                # Calculate the membership values based on the distances (fuzziness parameter m controls the degree of membership).
                c1 = 1 / (1 + (d1/d2)**(2/(m-1)))
```

```
                new_C1.append(c1)

                new_C2.append(1 - c1)


        # Check for convergence: if the membership values don't change 
significantly, stop the algorithm.

        if all(abs(nc1 - c1) < tol and abs(nc2 - c2) < tol

              for nc1, c1, nc2, c2 in zip(new_C1, C1, new_C2, C2)):

            break


        C1 = new_C1

        C2 = new_C2


        # Update the cluster centers based on the new membership values.

        C1_center = [

            sum(x * c**m for x, c in zip(X, C1)) / sum(c**m for c in C1),

            sum(y * c**m for y, c in zip(Y, C1)) / sum(c**m for c in C1)

        ]

        C2_center = [

            sum(x * c**m for x, c in zip(X, C2)) / sum(c**m for c in C2),

            sum(y * c**m for y, c in zip(Y, C2)) / sum(c**m for c in C2)

        ]


    # Return the final membership values for both clusters.
```

```python
    return C1, C2


# Test data points and initial membership values

X = [1, 2, 3, 4, 5, 6]

Y = [6, 5, 8, 4, 7, 9]

C1 = [0.8, 0.9, 0.7, 0.3, 0.5, 0.2]

C2 = [0.2, 0.1, 0.3, 0.7, 0.5, 0.8]


# Run the fuzzy c-means algorithm

C1, C2 = fuzzy_c_means(X, Y, C1, C2)


# Print the final membership values for each cluster

print("Final membership values for cluster 1:", C1)

print("Final membership values for cluster 2:", C2)
```

Output:

```
~/Doc/CS354-Lab/Lab-10 〉 on main ?1          ✔ 〈 base Py 〈 at 02:19:10
python q1.py
Final membership values for cluster 1: [0.9004245384645562, 0.994941864329955, 0
.3015792825013346, 0.7735352401044389, 0.0681914145420l451, 0.0777171135831387 8]
Final membership values for cluster 2: [0.09957546153544383, 0.005058135670044983, 0.6984207174986654, 0.22646475989556114, 0.9318085854579855, 0.922282886416861 2]
```

Question 2:

```python
import numpy as np

from sklearn.datasets import load_wine

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.metrics import adjusted_rand_score

import matplotlib.pyplot as plt


# Load and preprocess the wine dataset

data = load_wine()

X_raw = data.data

y = data.target


# Standardize the data

scaler = StandardScaler()

X = scaler.fit_transform(X_raw)


# Parameters for Fuzzy C-Means

k = 5

m = 1.7

max_iter = 100

tol = 1e-5
```

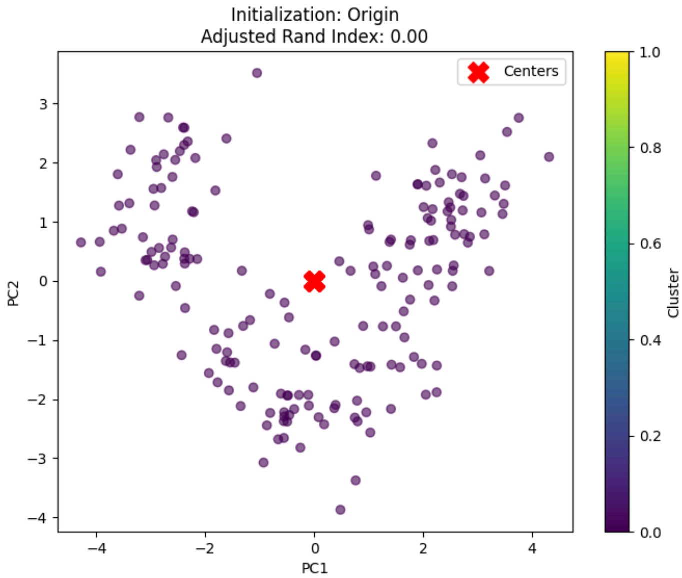
```python
# Fuzzy C-Means clustering function

def fuzzy_c_means(X, k, m, initial_centers, max_iter=100, tol=1e-5):

    n, d = X.shape

    centers = initial_centers.copy()

    U = np.zeros((n, k))


    # Initialize membership matrix U based on initial centers

    for i in range(n):

        distances = np.linalg.norm(X[i] - centers, axis=1)

        distances = np.clip(distances, 1e-10, None)  # Prevent division by zero

        U[i] = (distances**(-2 / (m - 1))) / np.sum(distances**(-2 / (m - 1)))


    # Iterate until convergence or max_iter

    for _ in range(max_iter):

        # Update centers

        new_centers = np.array([np.dot(U[:, j]**m, X) / np.sum(U[:, j]**m) for j
in range(k)])


        # Update membership matrix U

        new_U = np.zeros((n, k))

        for i in range(n):

            distances = np.linalg.norm(X[i] - new_centers, axis=1)
```

```
            distances = np.clip(distances, 1e-10, None)  # Prevent division by
zero

            new_U[i] = (distances**(-2 / (m - 1))) / np.sum(distances**(-2 / (m
- 1)))


        # Check for convergence

        if np.max(np.abs(new_U - U)) < tol:

            break

        U, centers = new_U.copy(), new_centers.copy()


    return U, centers


# PCA for 2D visualization

pca = PCA(n_components=2)

X_pca = pca.fit_transform(X)


# Initialize cluster centers using different methods

np.random.seed(42)

initializations = [

    ('Origin', np.zeros((k, X.shape[1]))),

    ('Gaussian', np.random.multivariate_normal(np.zeros(X.shape[1]),
np.eye(X.shape[1]), k)),

    ('Data Points', X[np.random.choice(X.shape[0], k, replace=False)])

]
```

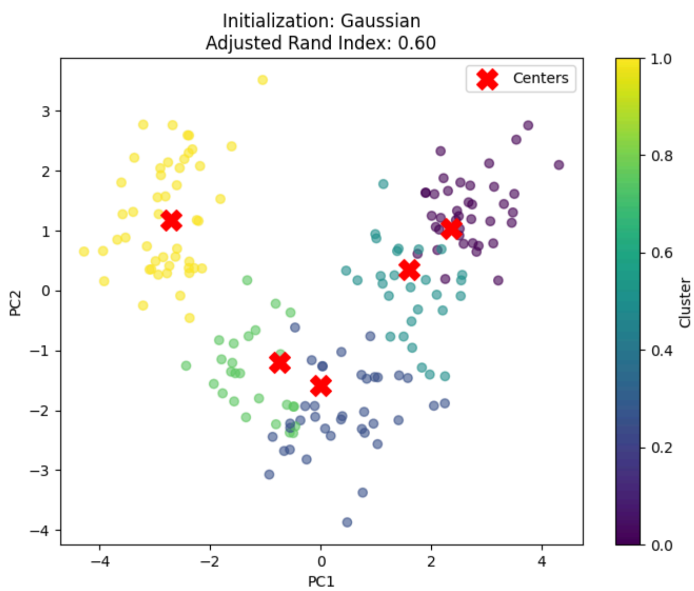
```
# Run FCM for each initialization and compute ARI score
results = []
for name, centers in initializations:
    U, final_centers = fuzzy_c_means(X, k, m, centers, max_iter, tol)
    clusters = np.argmax(U, axis=1)
    ari = adjusted_rand_score(y, clusters)
    results.append((name, clusters, final_centers, ari))

# Plot clusters and centers
for name, clusters, centers, ari in results:
    # Check for NaN values in centers
    if np.any(np.isnan(centers)):
        print(f"Warning: NaN values found in centers for initialization:
{name}")
        continue

    # Perform PCA transformation on centers
    centers_pca = pca.transform(centers)
    plt.figure(figsize=(8, 6))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis', alpha=0.6)
    plt.scatter(centers_pca[:, 0], centers_pca[:, 1], c='red', marker='X',
s=200, label='Centers')
```

```
    plt.title(f'Initialization: {name}\nAdjusted Rand Index: {ari:.2f}')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.legend()
    plt.colorbar(label='Cluster')
    plt.show()

# Print ARI scores for each initialization
print("Adjusted Rand Index Scores:")
for name, _, _, ari in results:
    print(f"{name}: {ari:.3f}")
```

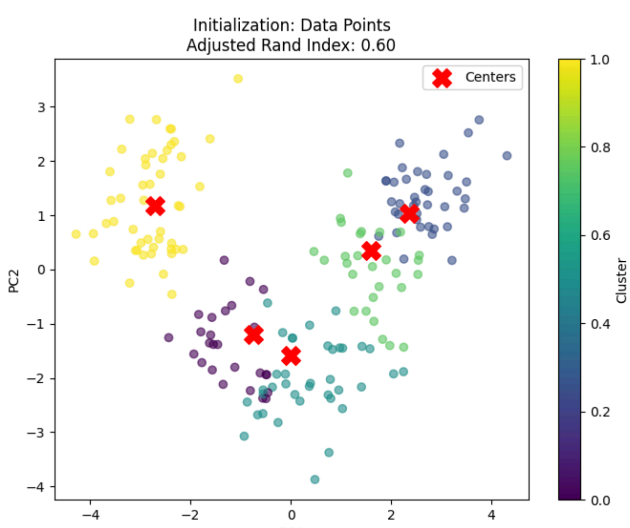
Output:



Initialization: Origin
Adjusted Rand Index: 0.00
Centers
PC1
PC2
Cluster



Initialization: Gaussian
Adjusted Rand Index: 0.60
Centers
PC1
PC2
Cluster



Initialization: Data Points
Adjusted Rand Index: 0.60
Centers
PC1
PC2
Cluster