LAB No 11

Agglomerative Hierarchical Clustering

In this lab, students will learn how to perform Agglomerative Hierarchical Clustering (AHC), a method used to group similar data points into clusters. The lab involves:

- Understanding hierarchical clustering and dendrograms.
- Performing clustering on datasets using Python (scikit-learn, scipy).
- Visualizing clusters using dendrograms.
- Interpreting the clustering results for practical data analysis.

Objectives:

1. Understand hierarchical clustering concepts and linkage methods.
2. Perform agglomerative clustering on sample datasets.
3. Visualize the clustering process using dendrograms.
4. Analyze cluster assignments and validate results.

Theory

1. Introduction to Hierarchical Clustering

Hierarchical clustering is an **unsupervised learning** method that builds a hierarchy of clusters. It can be:

- **Agglomerative (bottom-up)**:
  Each observation starts as its own cluster, and pairs of clusters are merged step by step until only one cluster remains.
- **Divisive (top-down)**:
  Start with all observations in one cluster and recursively split them into smaller clusters.

2. Agglomerative Hierarchical Clustering

- Start with each data point as a separate cluster.
- Compute a **distance matrix** between all clusters.

- Merge the **two closest clusters** at each step.
- Repeat until all points belong to a single cluster.

Distance Metrics:

- **Euclidean Distance:** Most common for continuous data.
- **Manhattan Distance:** Sum of absolute differences.
- **Cosine Distance:** Measures angular distance for high-dimensional data.

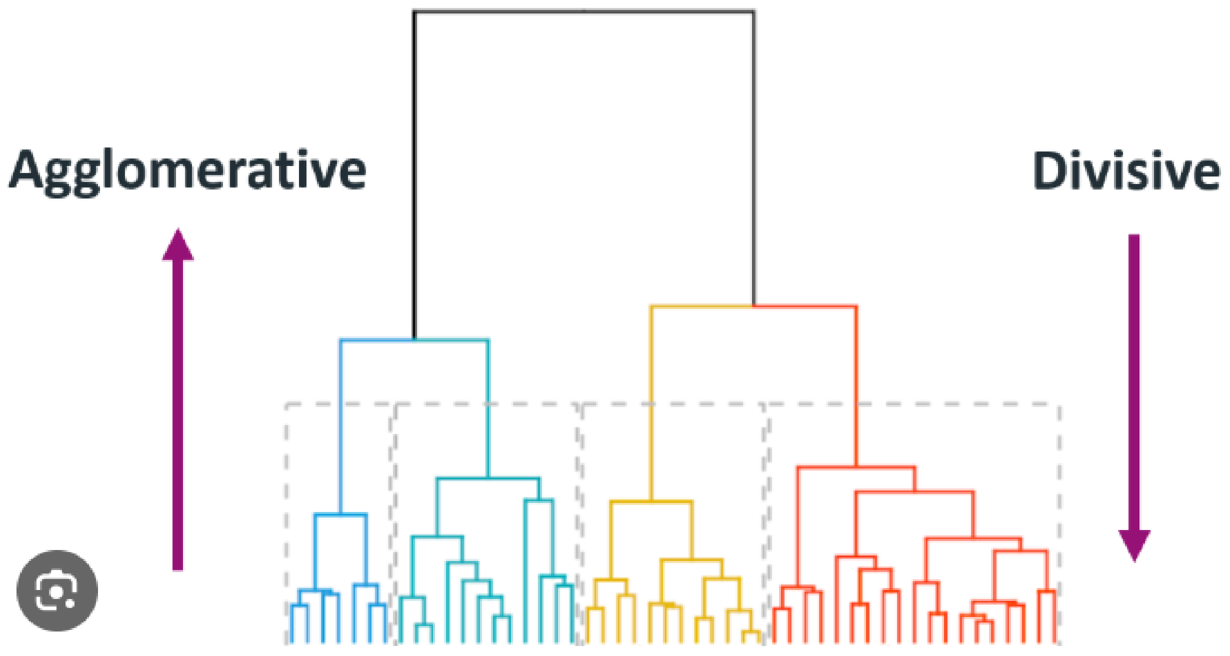
Linkage Methods:

- **Single Linkage:** Distance between closest points of two clusters.
- **Complete Linkage:** Distance between farthest points of two clusters.
- **Average Linkage:** Average distance between all points in two clusters.
- **Ward's Method:** Minimizes variance within clusters.

3. Dendrogram

A **dendrogram** is a tree-like diagram showing the order of cluster merges. It helps to:

- Visualize the hierarchy of clusters.
- Decide the optimal number of clusters by cutting the dendrogram.



4. Applications

- Customer segmentation in marketing.
- Document clustering in NLP.
- Gene expression analysis in bioinformatics.
- Image segmentation.

Python Libraries Required

```
import numpy as np
import pandas as pd
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

Solved Examples

Example 1: Clustering Simple 2D Points

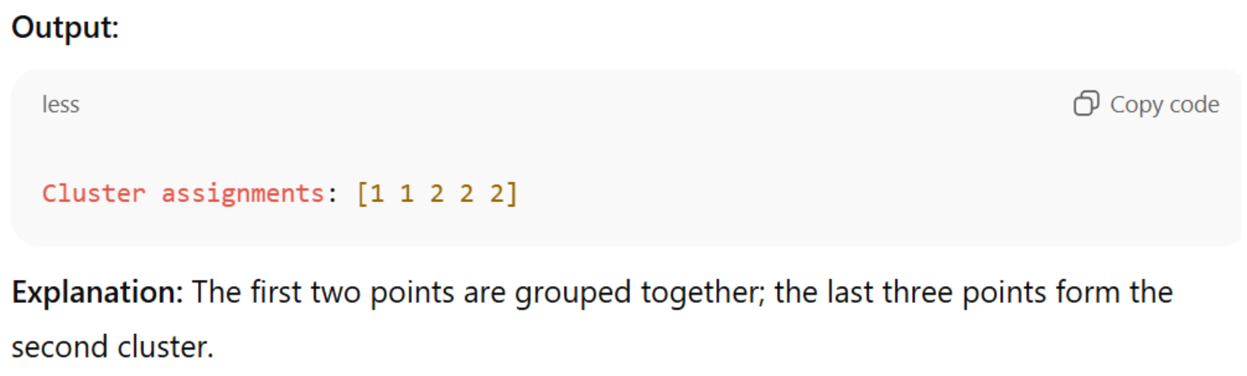
Dataset:

```
data = np.array([[1, 2], [2, 3], [5, 8], [6, 9], [10, 12]])
```

Solution

```
# Step 1: Import libraries
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
import matplotlib.pyplot as plt

# Step 2: Linkage matrix
Z = linkage(data, method='ward')  # Using Ward's method

# Step 3: Plot dendrogram
plt.figure(figsize=(6,4))
dendrogram(Z)
plt.title("Dendrogram - Example 1")
plt.show()

# Step 4: Form clusters (choose 2 clusters)
```

```
clusters = fcluster(Z, t=2, criterion='maxclust')
print("Cluster assignments:", clusters)
```



Output:

less Copy code

```
Cluster assignments: [1 1 2 2 2]
```

Explanation: The first two points are grouped together; the last three points form the second cluster.

Example 2: Agglomerative Clustering on Random Dataset

Solution

```
from sklearn.datasets import make_blobs
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

# Generate random data
X, _ = make_blobs(n_samples=8, centers=3, random_state=42)

# Linkage
Z = linkage(X, method='complete')

# Dendrogram
plt.figure(figsize=(6,4))
dendrogram(Z)
plt.title("Dendrogram - Example 2")
plt.show()

# Form clusters
clusters = fcluster(Z, t=3, criterion='maxclust')
print("Cluster assignments:", clusters)
```

Explanation: The dendrogram shows three distinct clusters; cluster labels indicate the group each point belongs to.

Example 3: Agglomerative Clustering on Iris Dataset (subset)

Solution

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
import matplotlib.pyplot as plt

# Load Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Use only sepal length and width
X = StandardScaler().fit_transform(X)

# Linkage
Z = linkage(X, method='average', metric='euclidean')

# Plot dendrogram
plt.figure(figsize=(8,5))
dendrogram(Z)
plt.title("Dendrogram - Iris Example")
plt.show()

# Form clusters (3 clusters)
clusters = fcluster(Z, t=3, criterion='maxclust')
print("Cluster assignments:", clusters)
```

Explanation:

- Standardization is important to normalize features.
- The dendrogram helps to visualize clusters of similar iris species.
- fcluster assigns each data point to a cluster.

LAB No 11

Implementation of Fuzzy C means Clustering

Fuzzy C-Means (FCM) clustering is an unsupervised soft-clustering technique that assigns data points to multiple clusters with varying degrees of membership, making it ideal for handling overlapping or uncertain data. In this practice program, students implement FCM in Python to generate clusters, analyze membership values, visualize results, and compare its performance with traditional hard-clustering methods like K-means.

Install Required libraries

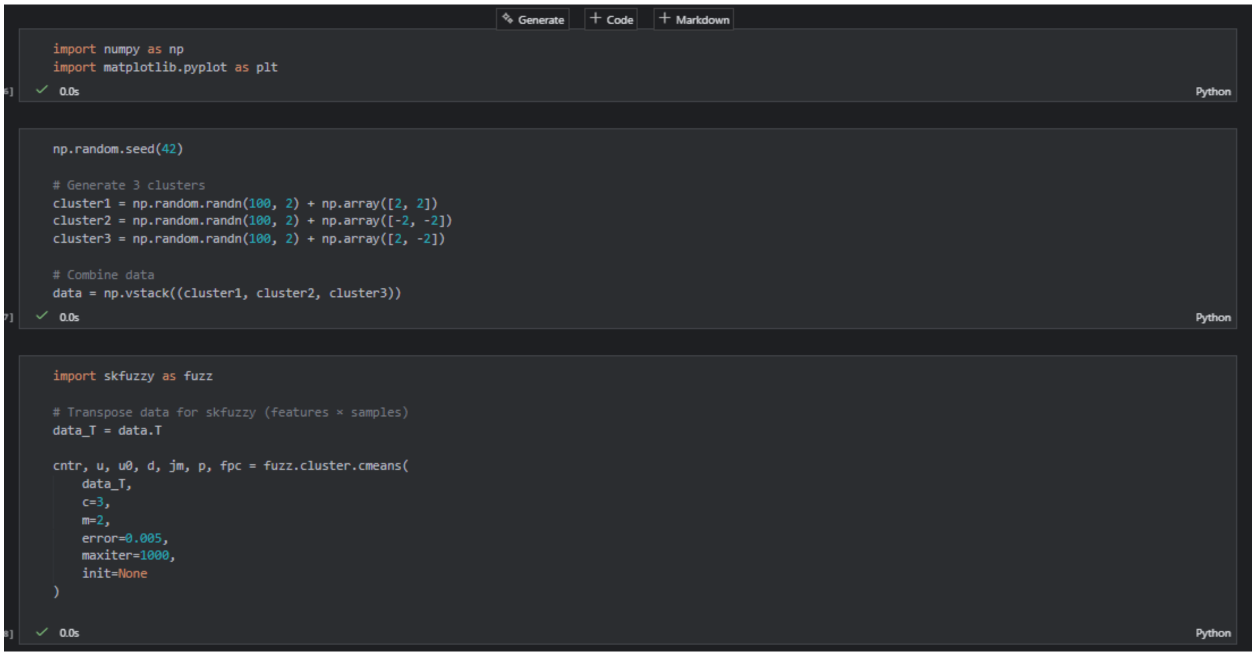
```
pip install scikit-fuzzy
```

Question No. 1

Task:
Generate a synthetic 2-dimensional dataset consisting of three clusters. Apply **Fuzzy C-Means clustering** and analyze the results.

Questions:

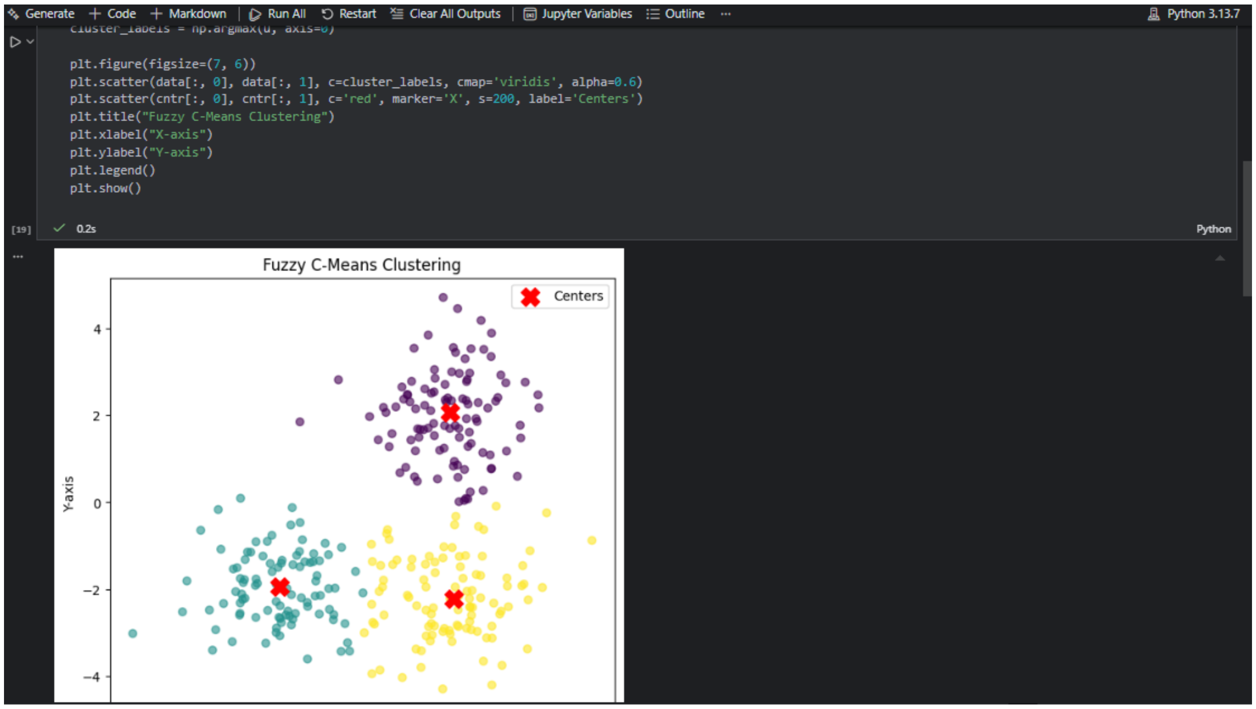
1. Generate a 2D dataset with three groups of points using Gaussian noise.
2. Apply Fuzzy C-Means (FCM) clustering using **skfuzzy** with 3 clusters.
3. Plot the clustered data points and cluster centers.
4. Display the **membership values** for any 5 randomly selected points.
5. Compute and interpret the **Fuzzy Partition Coefficient (FPC)**.
6. Compare the results with K-means clustering.



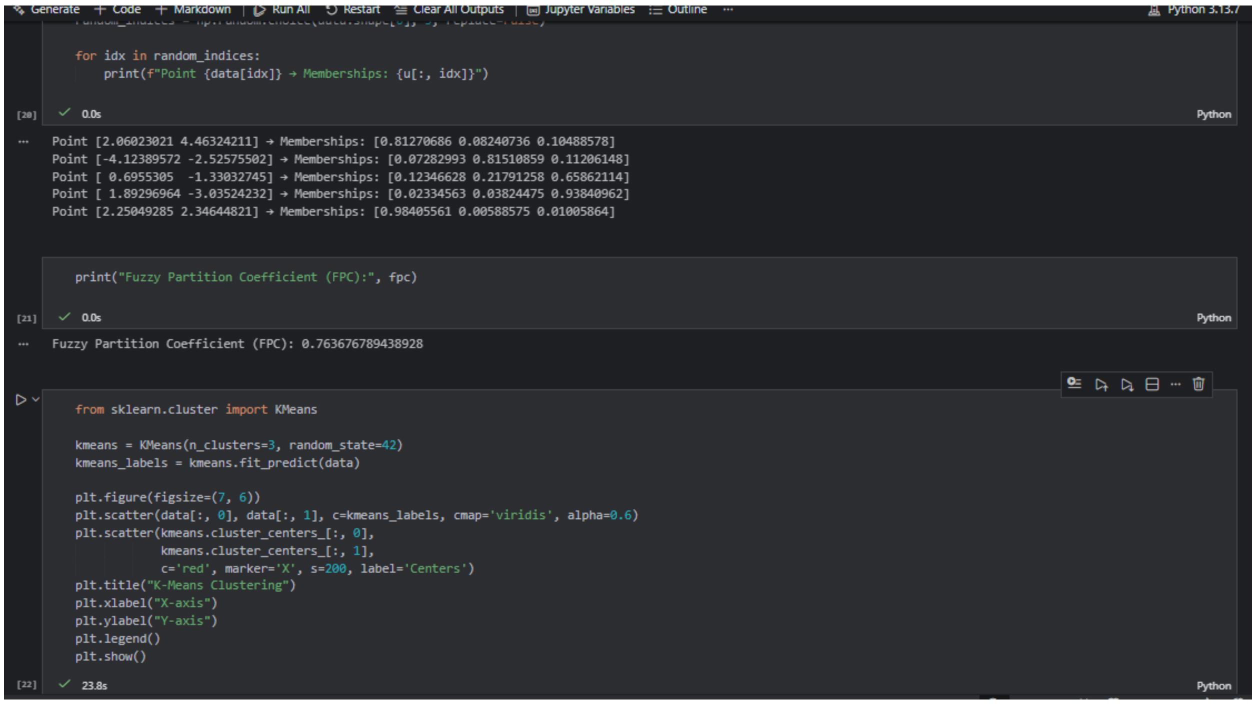
```
import numpy as np
import matplotlib.pyplot as plt
```
```
✓ 0.0s
```

```
np.random.seed(42)

# Generate 3 clusters
cluster1 = np.random.randn(100, 2) + np.array([2, 2])
cluster2 = np.random.randn(100, 2) + np.array([-2, -2])
cluster3 = np.random.randn(100, 2) + np.array([2, -2])

# Combine data
data = np.vstack((cluster1, cluster2, cluster3))
```
```
✓ 0.0s
```

```
import skfuzzy as fuzz

# Transpose data for skfuzzy (features × samples)
data_T = data.T

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    data_T,
    c=3,
    m=2,
    error=0.005,
    maxiter=1000,
    init=None
)
```
```
✓ 0.0s
```



```
  cluster_labels = np.argmax(u, axis=0)

  plt.figure(figsize=(7, 6))
  plt.scatter(data[:, 0], data[:, 1], c=cluster_labels, cmap='viridis', alpha=0.6)
  plt.scatter(cntr[:, 0], cntr[:, 1], c='red', marker='X', s=200, label='Centers')
  plt.title("Fuzzy C-Means Clustering")
  plt.xlabel("X-axis")
  plt.ylabel("Y-axis")
  plt.legend()
  plt.show()
```
```
[19] ✓ 0.2s
```



```
for idx in random_indices:
    print(f"Point {data[idx]} → Memberships: {u[:, idx]}")
```

[20] ✓ 0.0s

```
Point [2.06023021 4.46324211] → Memberships: [0.81270686 0.08240736 0.10488578]
Point [-4.12389572 -2.52575502] → Memberships: [0.07282993 0.81510859 0.11206148]
Point [ 0.6955305  -1.33032745] → Memberships: [0.12346628 0.21791258 0.65862114]
Point [ 1.89296964 -3.03524232] → Memberships: [0.02334563 0.03824475 0.93840962]
Point [2.25049285 2.34644821] → Memberships: [0.98405561 0.00588575 0.01005864]
```

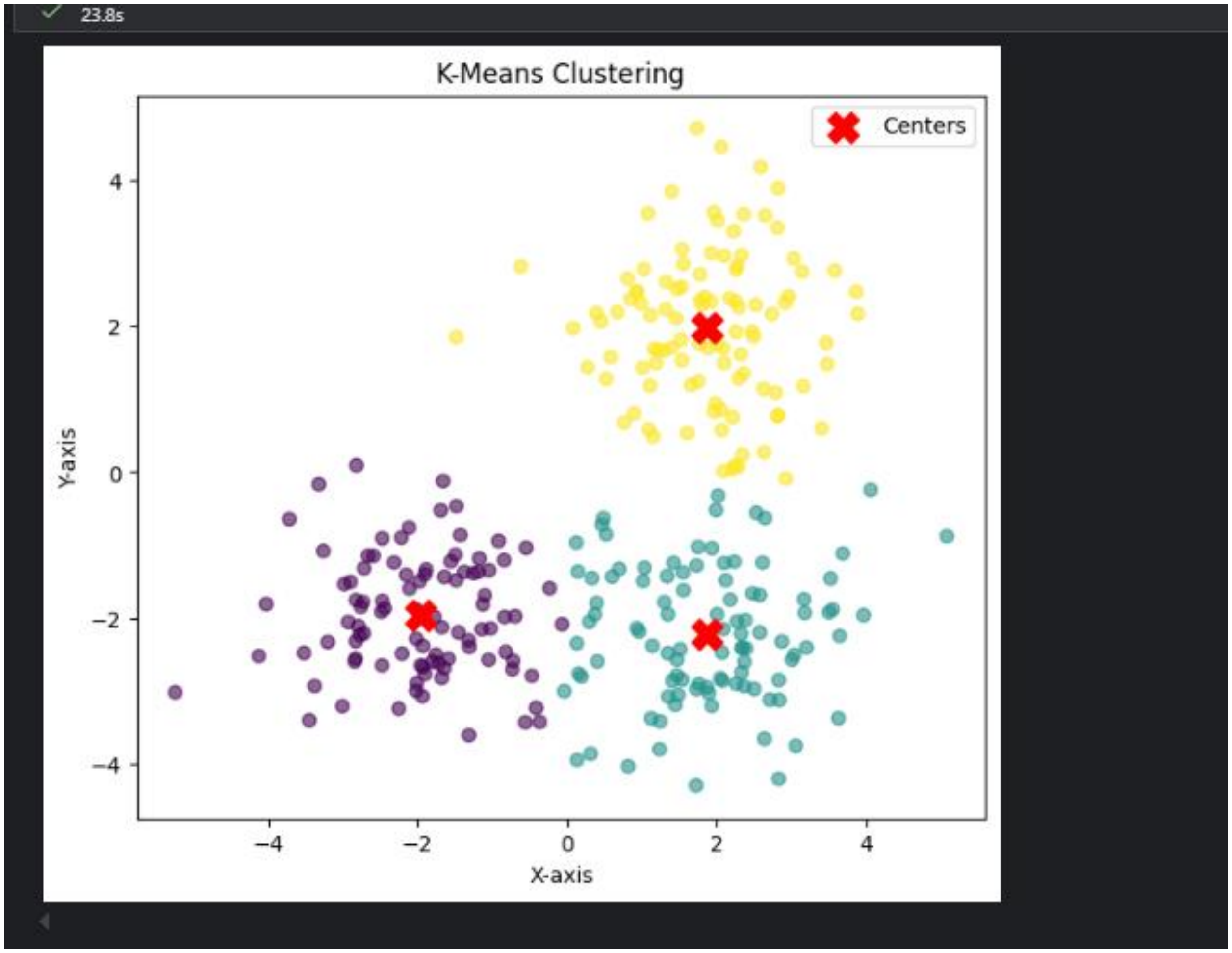
```
print("Fuzzy Partition Coefficient (FPC):", fpc)
```

[21] ✓ 0.0s

```
Fuzzy Partition Coefficient (FPC): 0.763676789438928
```

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(data)

plt.figure(figsize=(7, 6))
plt.scatter(data[:, 0], data[:, 1], c=kmeans_labels, cmap='viridis', alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1],
            c='red', marker='X', s=200, label='Centers')
plt.title("K-Means Clustering")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

[22] ✓ 23.8s



7. Explain why FCM is more suitable for overlapping clusters than K-means.

Answer:

Fuzzy C-Means (FCM) is more suitable for overlapping clusters because it uses **soft clustering**, where each data point is assigned a **degree of membership** to all clusters rather than being assigned to only one cluster. This allows FCM to effectively handle uncertainty and overlap between clusters by representing partial belonging.

In contrast, K-Means uses **hard clustering**, where each data point belongs to exactly one cluster. This rigid assignment can lead to incorrect clustering when cluster boundaries overlap, as points near the boundaries are forced into a single cluster.

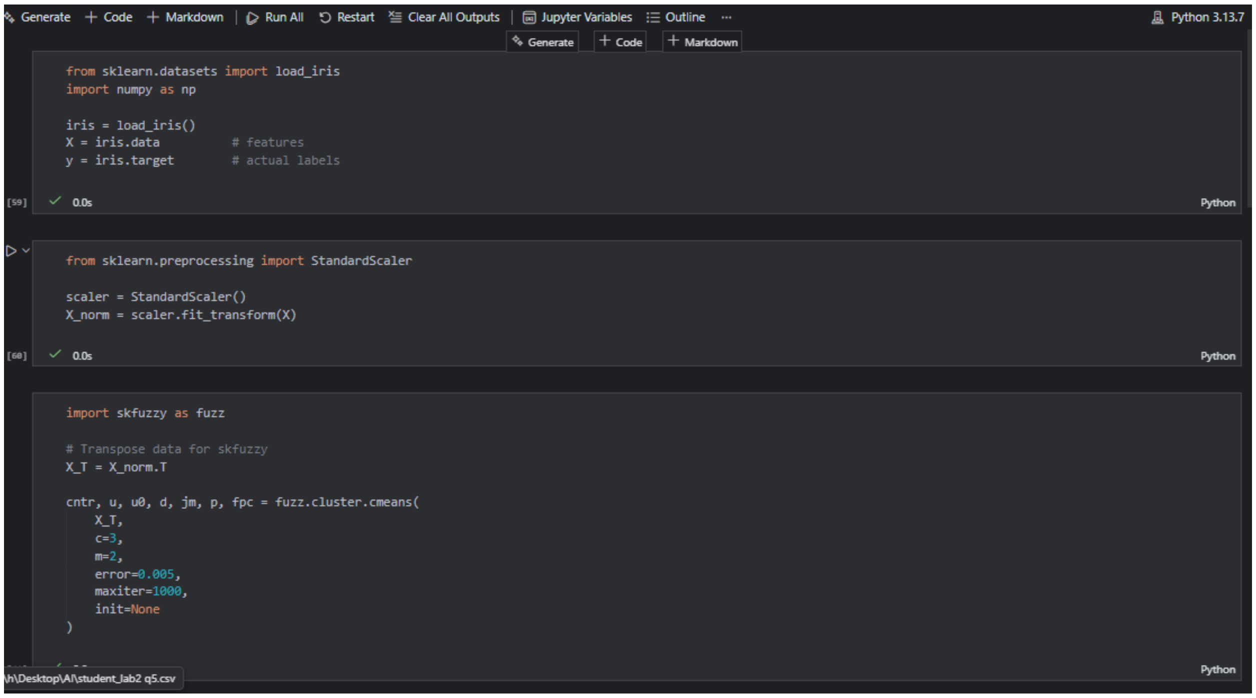
Therefore, FCM provides a more flexible and realistic clustering approach for overlapping data, making it superior to K-Means in scenarios involving ambiguous or uncertain cluster boundaries.

Question No. 2

Task:
Use the Iris dataset to cluster samples into 3 fuzzy classes and compare them with the actual species labels.

Questions:

1. Load the Iris dataset from sklearn.
2. Apply normalization and then use **FCM** to form 3 clusters.
3. Identify the predicted cluster for the first 20 samples.
4. Compare the predicted clusters with the actual labels.
5. Compute the **accuracy of FCM** (use majority-mapping method).
6. Report the **FPC value** and explain what it indicates about cluster quality.
7. Compare FCM results with K-means clustering on the same dataset.



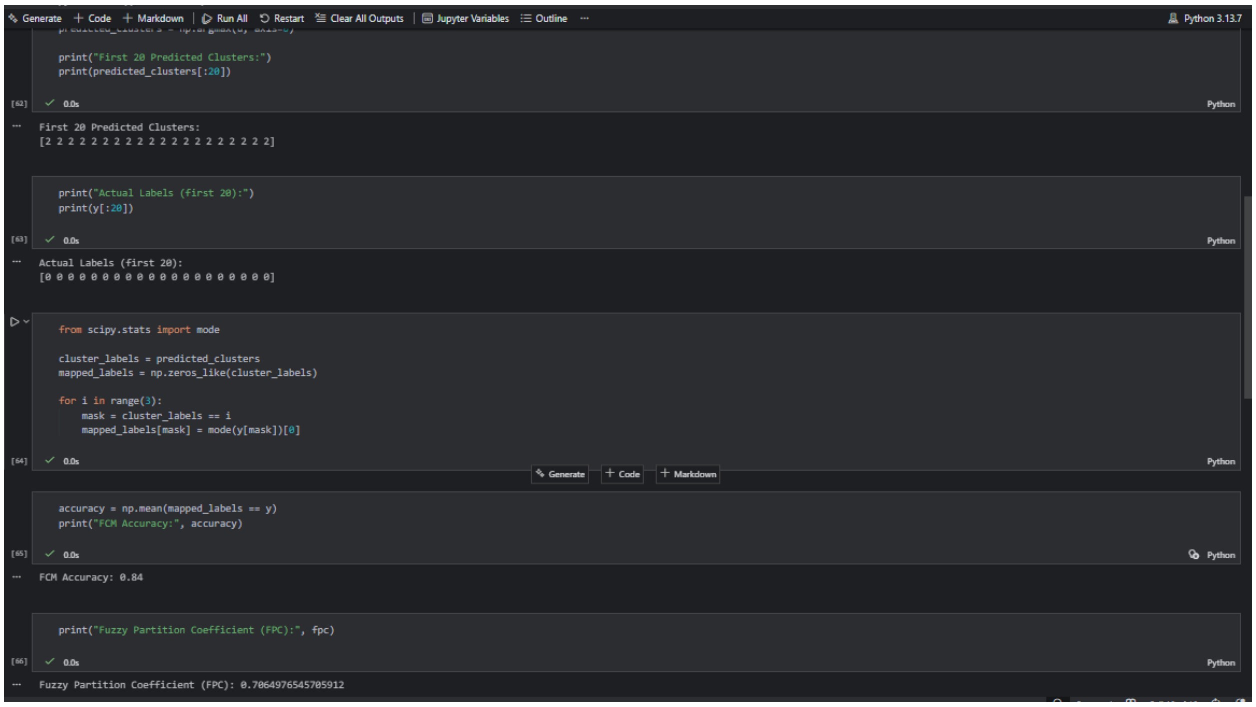
```
from sklearn.datasets import load_iris
import numpy as np

iris = load_iris()
X = iris.data          # features
y = iris.target        # actual labels
```

```
[59] ✓ 0.0s
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_norm = scaler.fit_transform(X)
```

```
[60] ✓ 0.0s
```

```
import skfuzzy as fuzz

# Transpose data for skfuzzy
X_T = X_norm.T

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X_T,
    c=3,
    m=2,
    error=0.005,
    maxiter=1000,
    init=None
)
```



```
print("First 20 Predicted Clusters:")
print(predicted_clusters[:20])
```

```
[62] ✓ 0.0s
```

```
First 20 Predicted Clusters:
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

```
print("Actual Labels (first 20):")
print(y[:20])
```

```
[63] ✓ 0.0s
```

```
Actual Labels (first 20):
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
from scipy.stats import mode

cluster_labels = predicted_clusters
mapped_labels = np.zeros_like(cluster_labels)

for i in range(3):
    mask = cluster_labels == i
    mapped_labels[mask] = mode(y[mask])[0]
```

```
[64] ✓ 0.0s
```

```
accuracy = np.mean(mapped_labels == y)
print("FCM Accuracy:", accuracy)
```

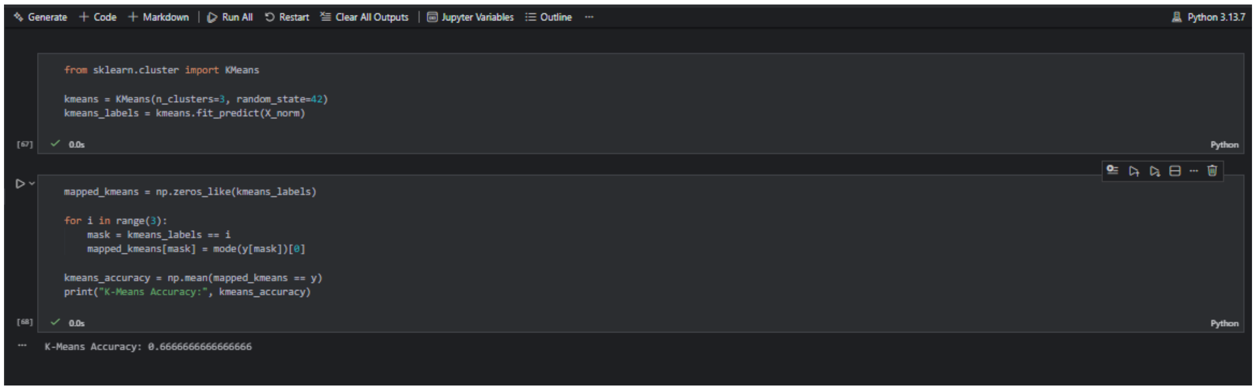
```
[65] ✓ 0.0s
```

```
FCM Accuracy: 0.84
```

```
print("Fuzzy Partition Coefficient (FPC):", fpc)
```

```
[66] ✓ 0.0s
```

```
Fuzzy Partition Coefficient (FPC): 0.7064976545705912
```



```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_norm)
```

```
[67] ✓ 0.0s
```

```
mapped_kmeans = np.zeros_like(kmeans_labels)

for i in range(3):
    mask = kmeans_labels == i
    mapped_kmeans[mask] = mode(y[mask])[0]

kmeans_accuracy = np.mean(mapped_kmeans == y)
print("K-Means Accuracy:", kmeans_accuracy)
```

```
[68] ✓ 0.0s
```

```
K-Means Accuracy: 0.6666666666666666
```

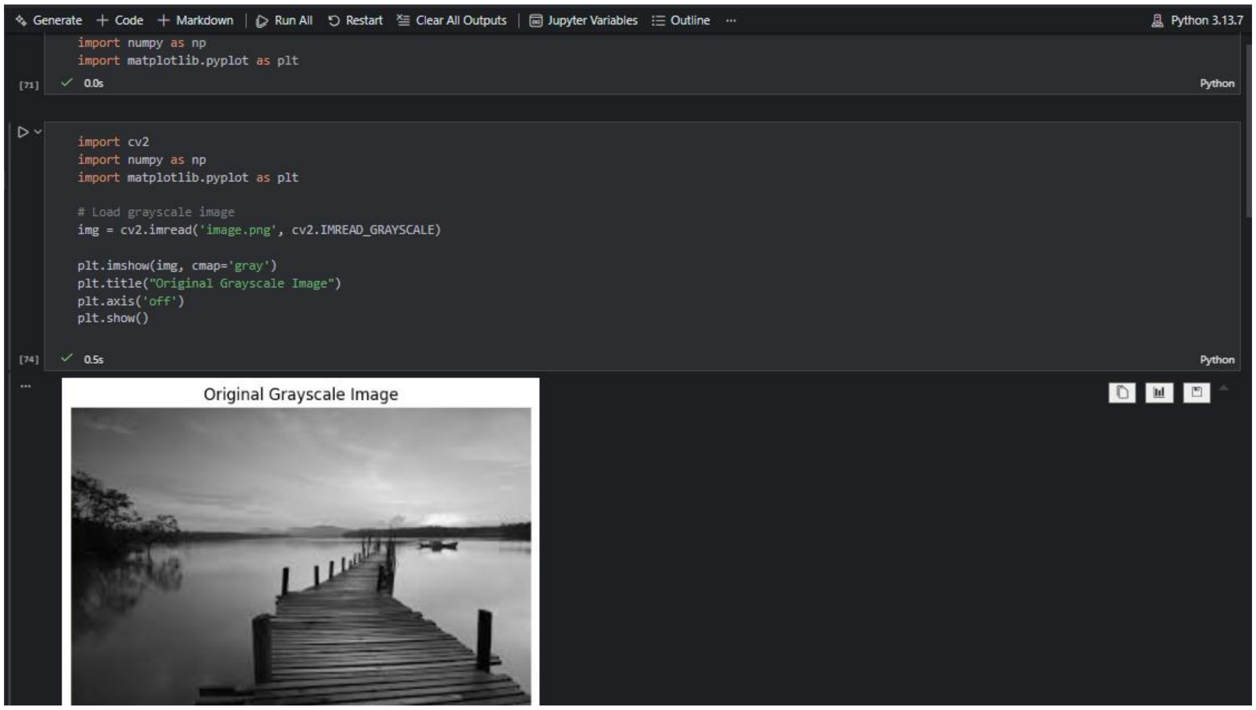
Question No. 3

Task:

Segment a grayscale image into meaningful regions using fuzzy clustering.

Questions:

1. Load any grayscale image (or the one provided by the teacher).
2. Convert the image into a 1D pixel array.
3. Apply Fuzzy C-Means clustering to segment the image into **three clusters**.
4. Reconstruct the segmented image and display it.

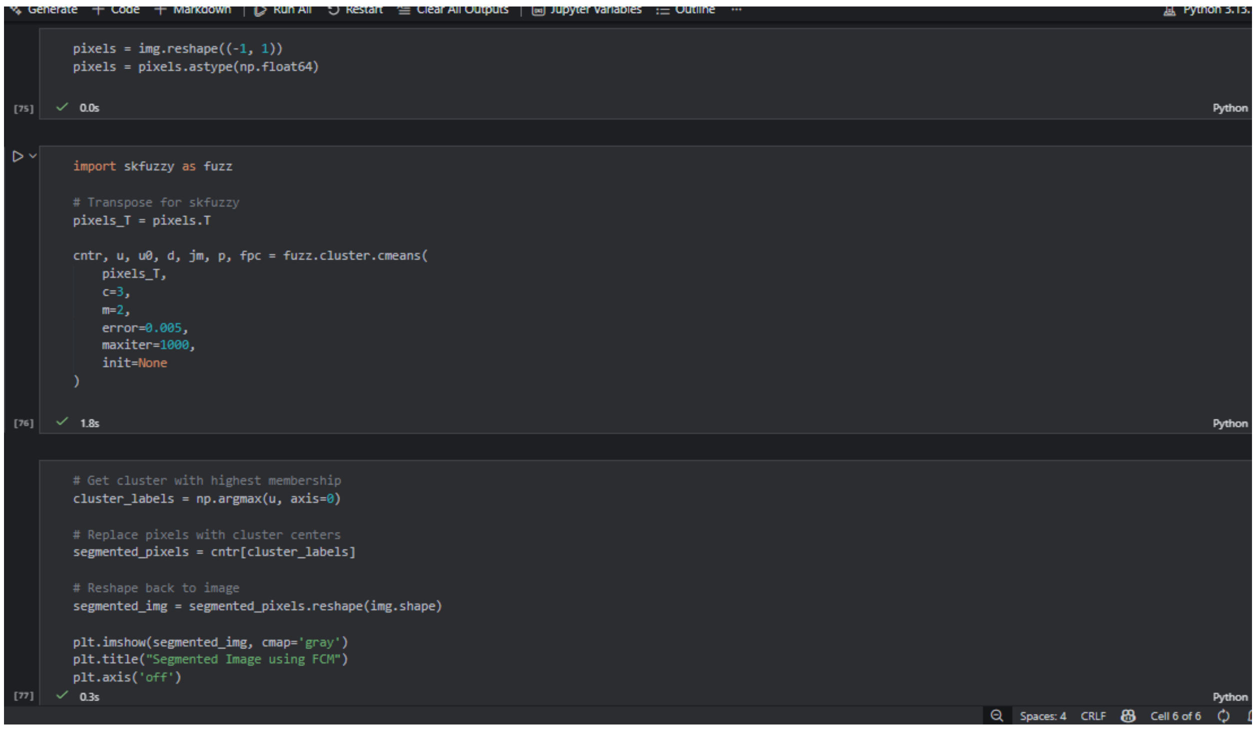


```
import numpy as np
import matplotlib.pyplot as plt
```

```
[71] ✓ 0.0s
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load grayscale image
img = cv2.imread('image.png', cv2.IMREAD_GRAYSCALE)

plt.imshow(img, cmap='gray')
plt.title("Original Grayscale Image")
plt.axis('off')
plt.show()
```

```
[74] ✓ 0.5s
```



Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ... Python 3.13.

```
pixels = img.reshape((-1, 1))
pixels = pixels.astype(np.float64)
```

[75] ✓ 0.0s Python

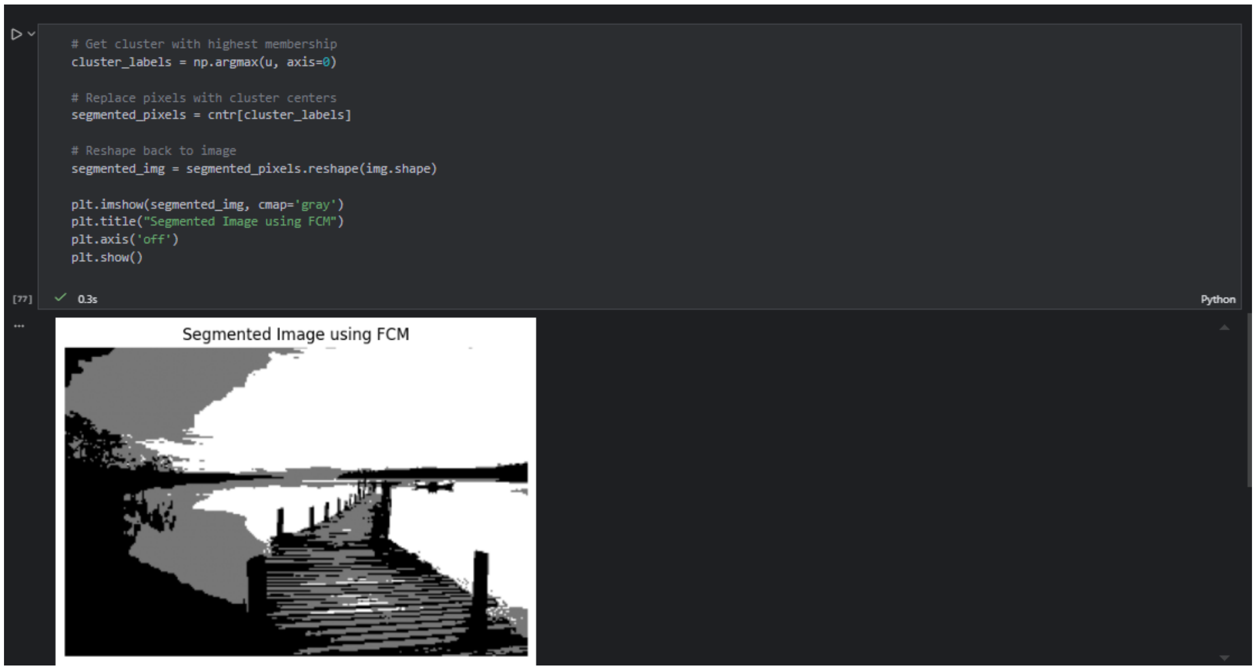
```
import skfuzzy as fuzz

# Transpose for skfuzzy
pixels_T = pixels.T

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    pixels_T,
    c=3,
    m=2,
    error=0.005,
    maxiter=1000,
    init=None
)
```

[76] ✓ 1.8s Python

```
# Get cluster with highest membership
cluster_labels = np.argmax(u, axis=0)

# Replace pixels with cluster centers
segmented_pixels = cntr[cluster_labels]

# Reshape back to image
segmented_img = segmented_pixels.reshape(img.shape)

plt.imshow(segmented_img, cmap='gray')
plt.title("Segmented Image using FCM")
plt.axis('off')
```

[77] ✓ 0.3s Python

Spaces: 4 CRLF Cell 6 of 6



```
# Get cluster with highest membership
cluster_labels = np.argmax(u, axis=0)

# Replace pixels with cluster centers
segmented_pixels = cntr[cluster_labels]

# Reshape back to image
segmented_img = segmented_pixels.reshape(img.shape)

plt.imshow(segmented_img, cmap='gray')
plt.title("Segmented Image using FCM")
plt.axis('off')
plt.show()
```

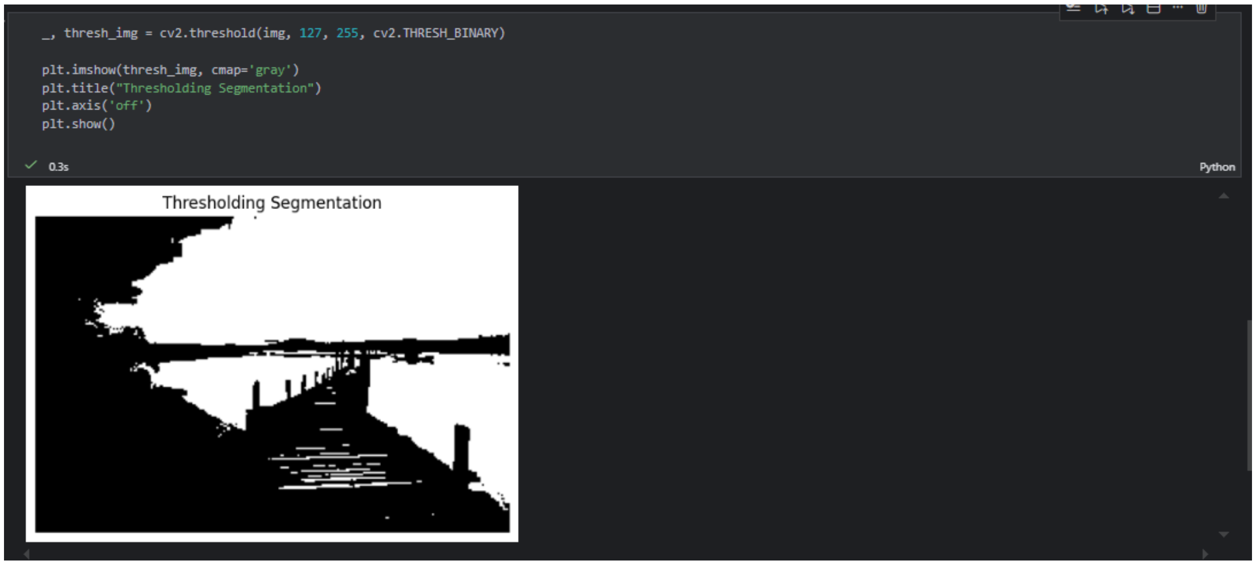
[77] ✓ 0.3s Python

5. Explain how pixel membership values differ across regions.

Answer:

In Fuzzy C-Means segmentation, each pixel is assigned a **membership value** for every cluster instead of a hard label. Pixels in **homogeneous regions** (clear dark or bright areas) show high membership for a single cluster. However, pixels near **edges or boundaries** exhibit distributed membership values across multiple clusters, representing uncertainty. This allows FCM to produce smoother transitions between regions compared to hard segmentation methods.

6. Compare your segmented image with segmentation from **thresholding** or **K-means**.



```
_, thresh_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

plt.imshow(thresh_img, cmap='gray')
plt.title("Thresholding Segmentation")
plt.axis('off')
plt.show()
```

```
✓ 0.3s                                                                 Python
```

7. Discuss how changing the number of clusters (c = 2, 4, 5) affects segmentation quality.

Answer:

Effect of changing number of clusters (c = 2, 4, 5)

- **c = 2:**
  Image is over-simplified; important details may be lost.
- **c = 3:**
  Balanced segmentation; main regions clearly separated.
- **c = 4 or 5:**
  Finer segmentation; captures subtle intensity variations but may introduce noise and over-segmentation.

Optimal number of clusters depends on image complexity and application requirements.

Question No. 4

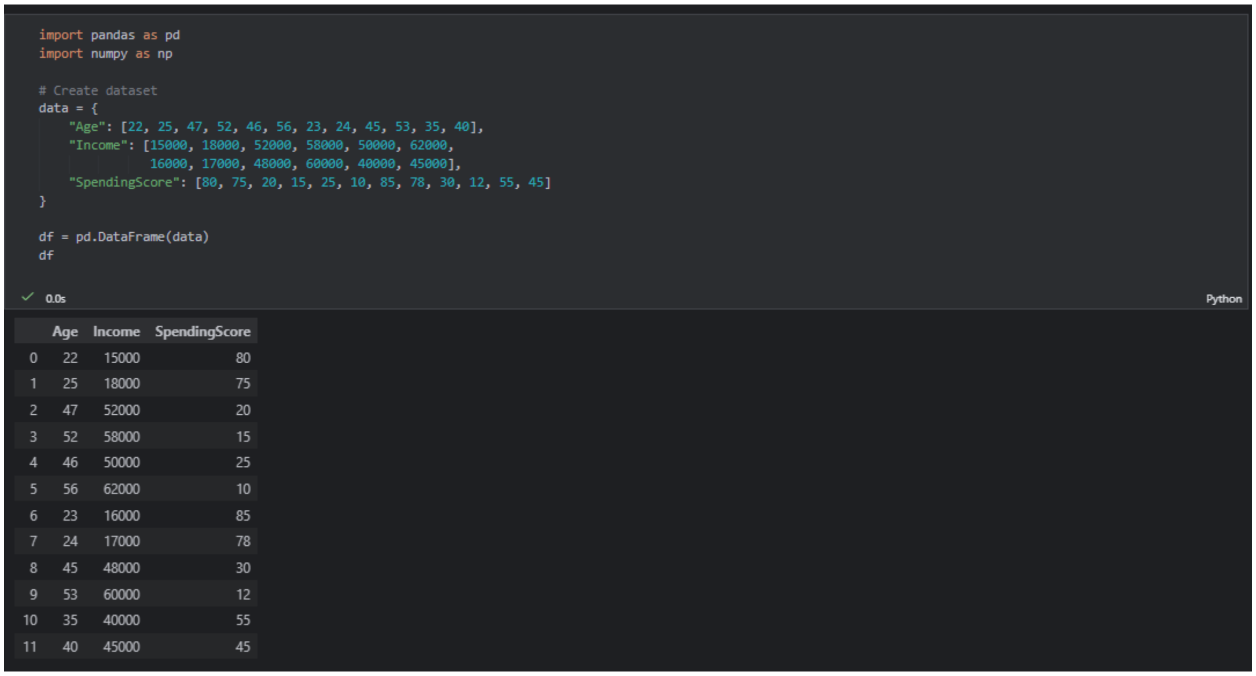
Task:
Perform market segmentation on a small customer dataset using Fuzzy C-Means clustering.

Dataset Fields:

- Age
- Income
- Spending Score

Questions:

1. Create or load the given dataset of 10–20 customers.
2. Normalize the features using MinMaxScaler.
3. Apply FCM to generate **3 customer clusters**.
4. Assign each customer to the cluster with maximum membership.
5. Display the membership matrix and cluster centers.
6. Interpret each cluster (e.g., high income–low spending).
7. Compare the results with K-means segmentation and justify whether FCM is better.

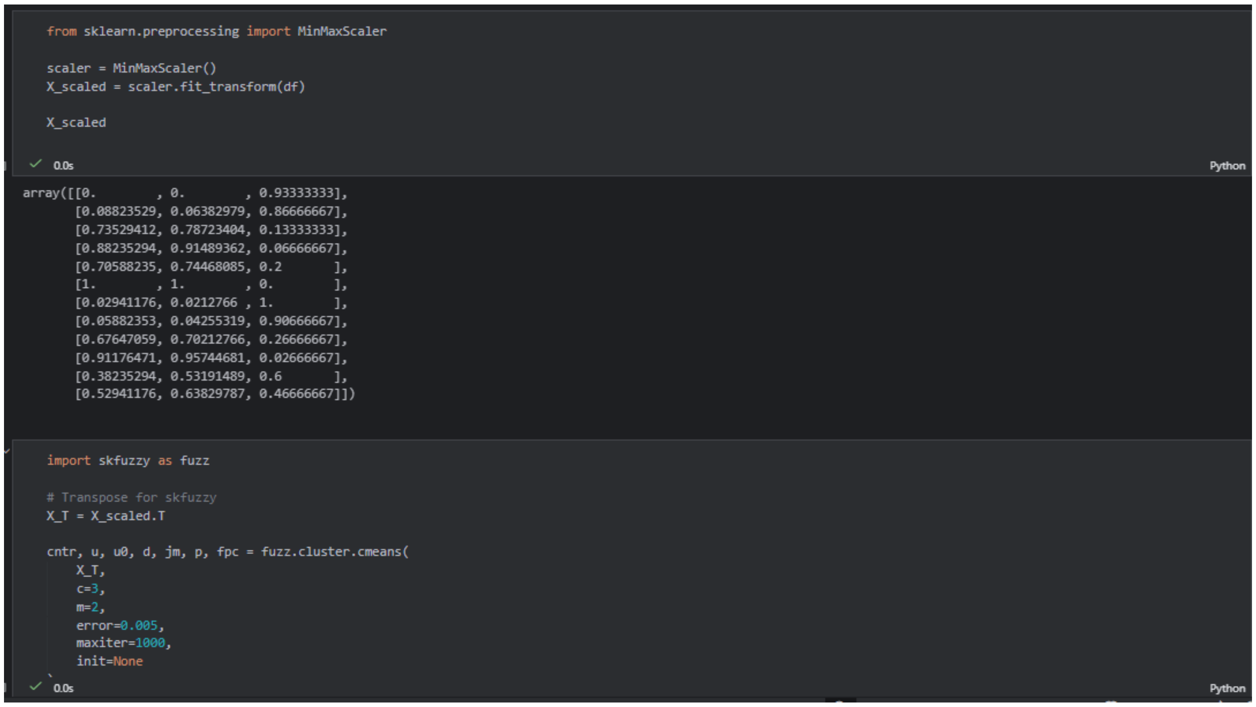


```python
import pandas as pd
import numpy as np

# Create dataset
data = {
    "Age": [22, 25, 47, 52, 46, 56, 23, 24, 45, 53, 35, 40],
    "Income": [15000, 18000, 52000, 58000, 50000, 62000,
               16000, 17000, 48000, 60000, 40000, 45000],
    "SpendingScore": [80, 75, 20, 15, 25, 10, 85, 78, 30, 12, 55, 45]
}

df = pd.DataFrame(data)
df
```

✓ 0.0s

Python

```
    Age  Income  SpendingScore
0    22   15000             80
1    25   18000             75
2    47   52000             20
3    52   58000             15
4    46   50000             25
5    56   62000             10
6    23   16000             85
7    24   17000             78
8    45   48000             30
9    53   60000             12
10   35   40000             55
11   40   45000             45
```



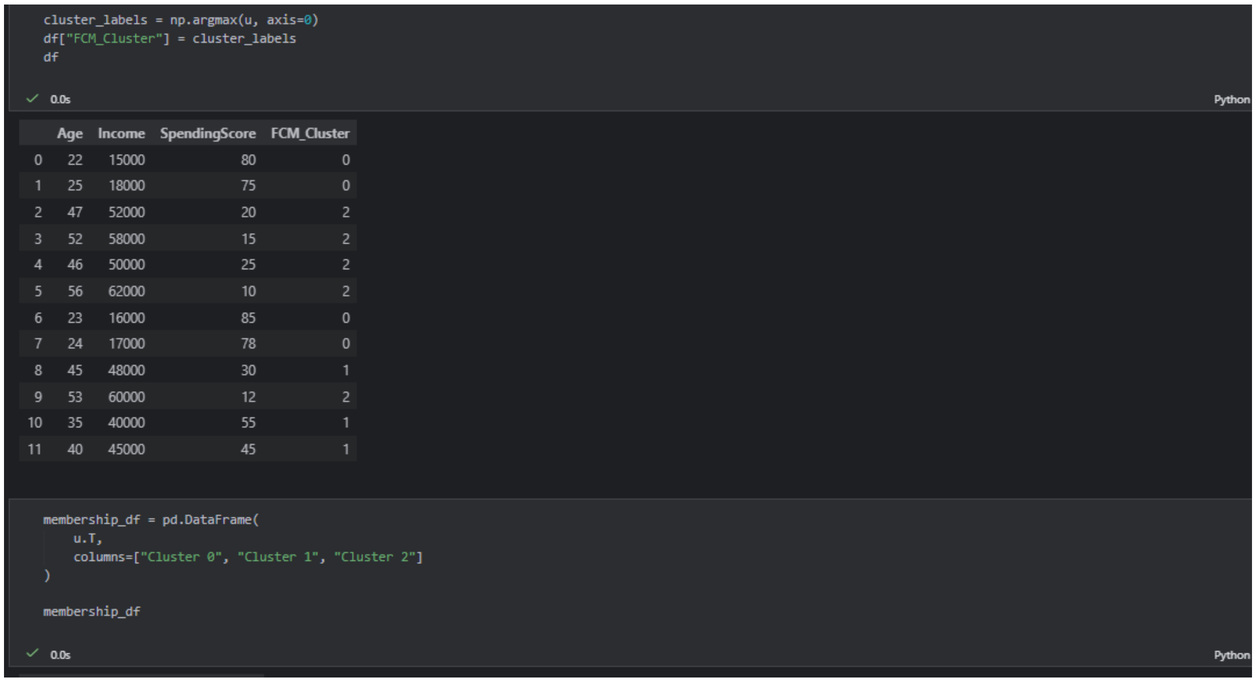
```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(df)

X_scaled
```

✓ 0.0s Python

```
array([[0.        , 0.        , 0.93333333],
       [0.08823529, 0.06382979, 0.86666667],
       [0.73529412, 0.78723404, 0.13333333],
       [0.88235294, 0.91489362, 0.06666667],
       [0.70588235, 0.74468085, 0.2       ],
       [1.        , 1.        , 0.        ],
       [0.02941176, 0.0212766 , 1.        ],
       [0.05882353, 0.04255319, 0.90666667],
       [0.67647059, 0.70212766, 0.26666667],
       [0.91176471, 0.95744681, 0.02666667],
       [0.38235294, 0.53191489, 0.6       ],
       [0.52941176, 0.63829787, 0.46666667]])
```

```python
import skfuzzy as fuzz

# Transpose for skfuzzy
X_T = X_scaled.T

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X_T,
    c=3,
    m=2,
    error=0.005,
    maxiter=1000,
    init=None
)
```

✓ 0.0s Python



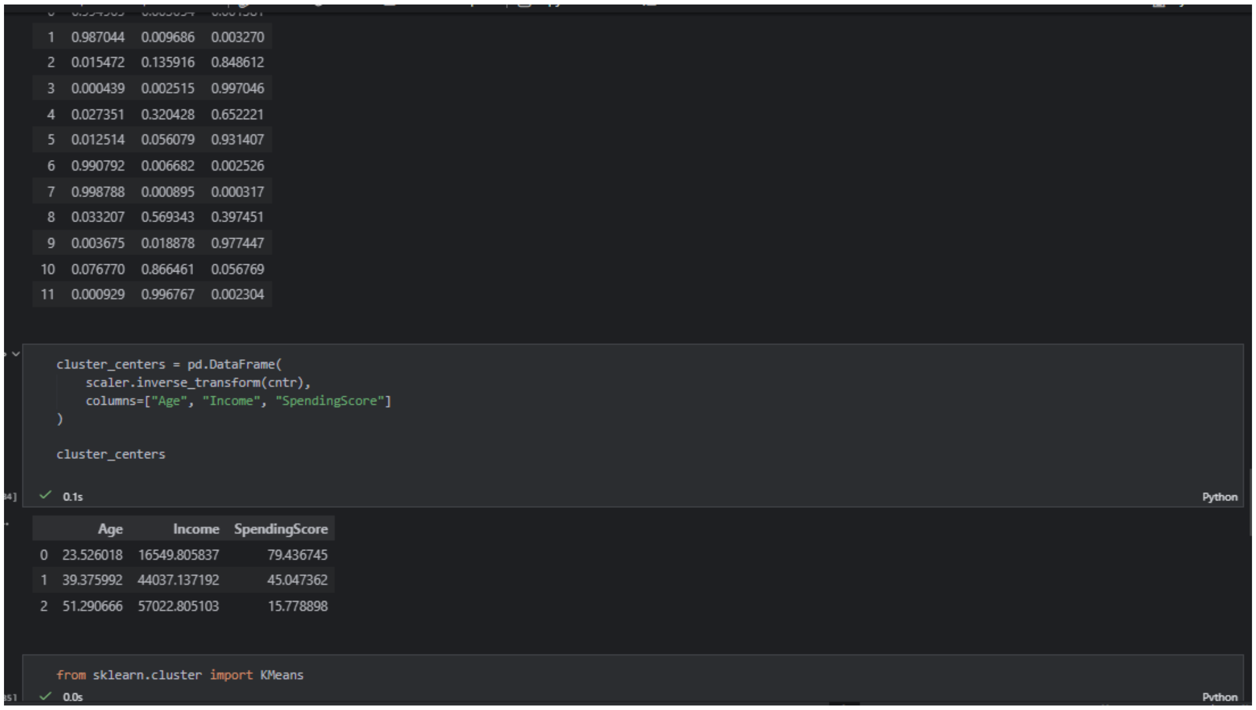
```python
cluster_labels = np.argmax(u, axis=0)
df["FCM_Cluster"] = cluster_labels
df
```

✓ 0.0s Python

	Age	Income	SpendingScore	FCM_Cluster
0	22	15000	80	0
1	25	18000	75	0
2	47	52000	20	2
3	52	58000	15	2
4	46	50000	25	2
5	56	62000	10	2
6	23	16000	85	0
7	24	17000	78	0
8	45	48000	30	1
9	53	60000	12	2
10	35	40000	55	1
11	40	45000	45	1

```python
membership_df = pd.DataFrame(
    u.T,
    columns=["Cluster 0", "Cluster 1", "Cluster 2"]
)

membership_df
```

✓ 0.0s Python



1	0.987044	0.009686	0.003270
2	0.015472	0.135916	0.848612
3	0.000439	0.002515	0.997046
4	0.027351	0.320428	0.652221
5	0.012514	0.056079	0.931407
6	0.990792	0.006682	0.002526
7	0.998788	0.000895	0.000317
8	0.033207	0.569343	0.397451
9	0.003675	0.018878	0.977447
10	0.076770	0.866461	0.056769
11	0.000929	0.996767	0.002304

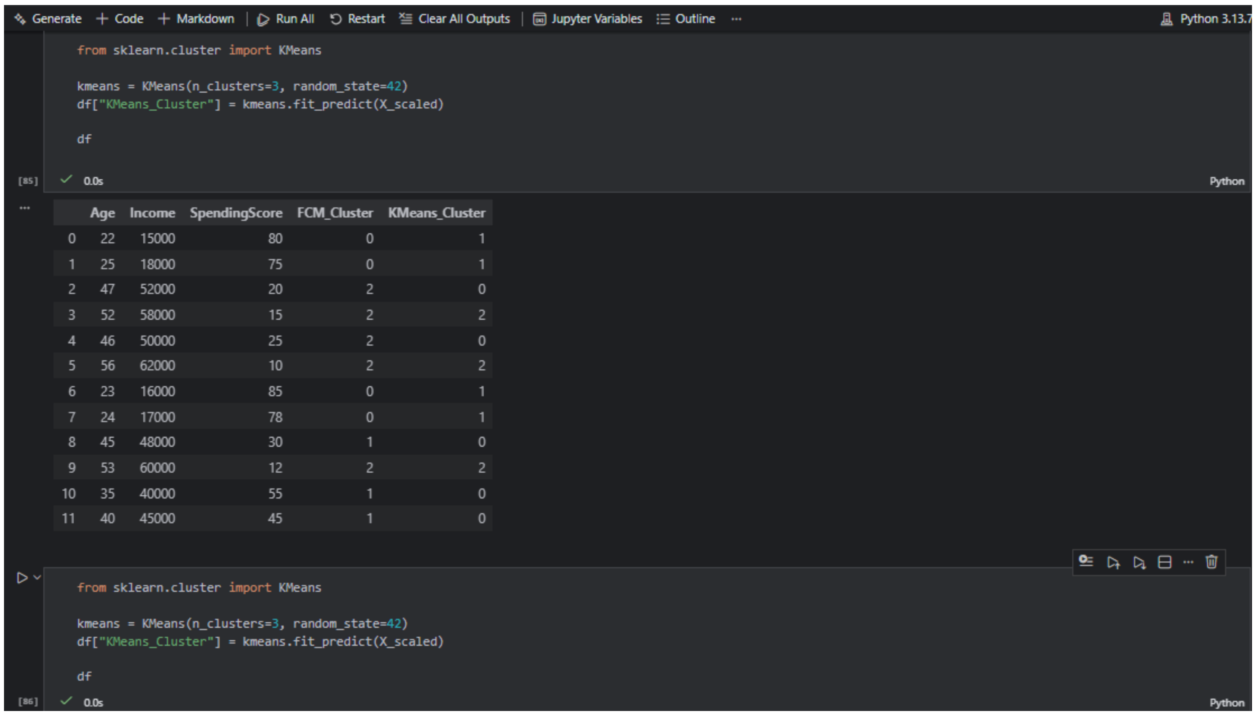
```python
cluster_centers = pd.DataFrame(
    scaler.inverse_transform(cntr),
    columns=["Age", "Income", "SpendingScore"]
)

cluster_centers
```

[84] ✓ 0.1s Python

	Age	Income	SpendingScore
0	23.526018	16549.805837	79.436745
1	39.375992	44037.137192	45.047362
2	51.290666	57022.805103	15.778898

```python
from sklearn.cluster import KMeans
```

✓ 0.0s Python



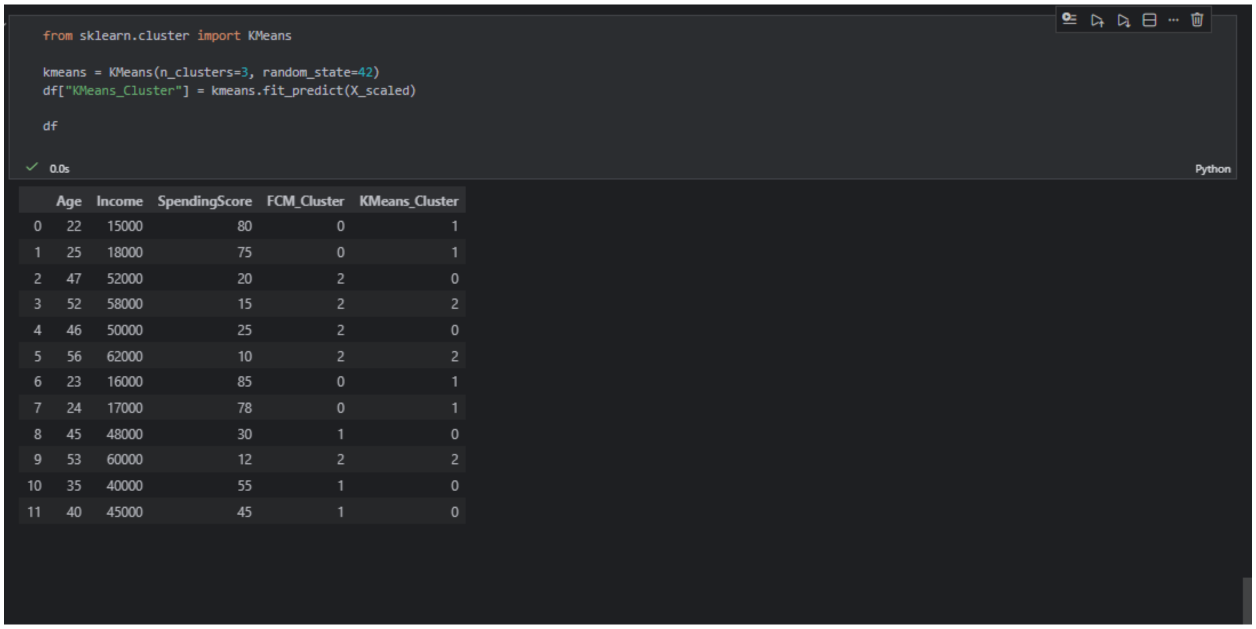
```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
df["KMeans_Cluster"] = kmeans.fit_predict(X_scaled)

df
```

[85] ✓ 0.0s Python

	Age	Income	SpendingScore	FCM_Cluster	KMeans_Cluster
0	22	15000	80	0	1
1	25	18000	75	0	1
2	47	52000	20	2	0
3	52	58000	15	2	2
4	46	50000	25	2	0
5	56	62000	10	2	2
6	23	16000	85	0	1
7	24	17000	78	0	1
8	45	48000	30	1	0
9	53	60000	12	2	2
10	35	40000	55	1	0
11	40	45000	45	1	0

```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
df["KMeans_Cluster"] = kmeans.fit_predict(X_scaled)

df
```

[86] ✓ 0.0s Python



```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
df["KMeans_Cluster"] = kmeans.fit_predict(X_scaled)

df
```

```
    Age  Income  SpendingScore  FCM_Cluster  KMeans_Cluster
0    22   15000             80            0               1
1    25   18000             75            0               1
2    47   52000             20            2               0
3    52   58000             15            2               2
4    46   50000             25            2               0
5    56   62000             10            2               2
6    23   16000             85            0               1
7    24   17000             78            0               1
8    45   48000             30            1               0
9    53   60000             12            2               2
10   35   40000             55            1               0
11   40   45000             45            1               0
```

Question No. 5

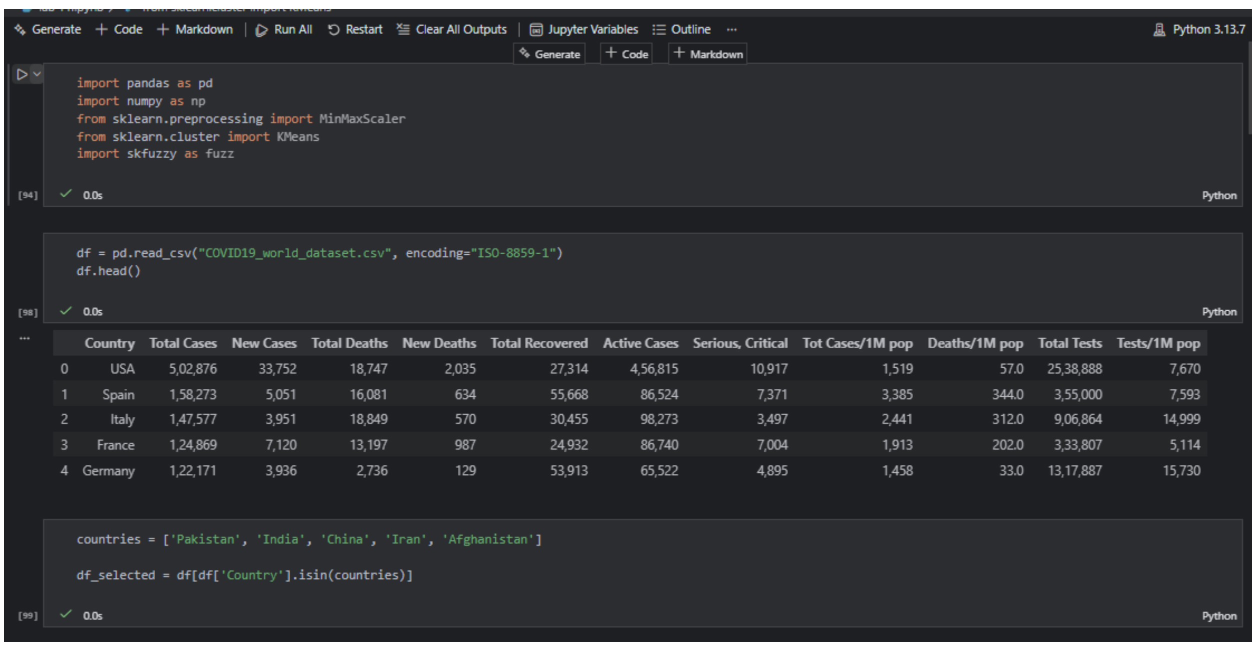
(Dataset: Kaggle → "COVID-19 World Dataset")

Task:
Cluster Pakistan and its neighboring countries based on COVID-19 indicators.

Questions:

1. Select the countries:
   - Pakistan
   - India
   - China
   - Iran
   - Afghanistan
2. Extract the following variables from the dataset:
   - Total Cases
   - Total Deaths
   - Population

3. Normalize the selected features.
4. Apply FCM with **2 clusters** and report the cluster membership values.
5. Show the final clusters for each country.
6. Interpret results (e.g., high-impact vs. low-impact countries).
7. Compute the **FPC** and discuss cluster quality.
8. Compare FCM-based clustering with K-means clustering and comment on differences.



```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import skfuzzy as fuzz
```

[94] 0.0s

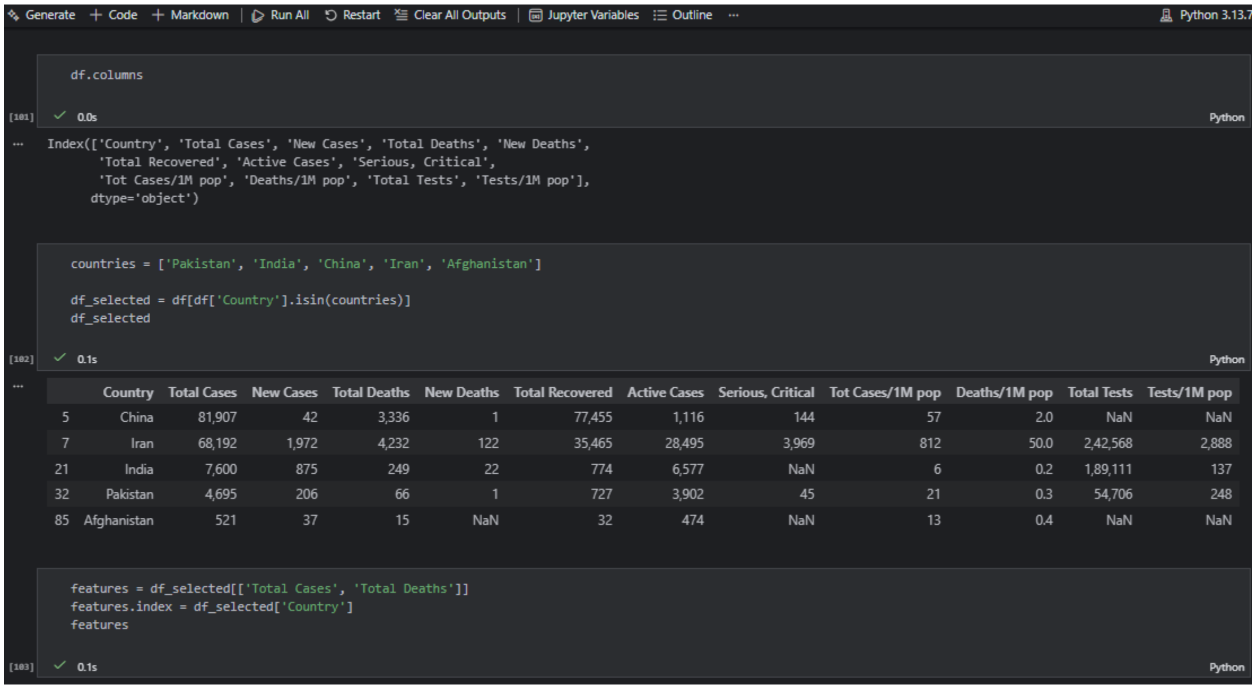
```python
df = pd.read_csv("COVID19_world_dataset.csv", encoding="ISO-8859-1")
df.head()
```

[98] 0.0s

	Country	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	Active Cases	Serious, Critical	Tot Cases/1M pop	Deaths/1M pop	Total Tests	Tests/1M pop
0	USA	5,02,876	33,752	18,747	2,035	27,314	4,56,815	10,917	1,519	57.0	25,38,888	7,670
1	Spain	1,58,273	5,051	16,081	634	55,668	86,524	7,371	3,385	344.0	3,55,000	7,593
2	Italy	1,47,577	3,951	18,849	570	30,455	98,273	3,497	2,441	312.0	9,06,864	14,999
3	France	1,24,869	7,120	13,197	987	24,932	86,740	7,004	1,913	202.0	3,33,807	5,114
4	Germany	1,22,171	3,936	2,736	129	53,913	65,522	4,895	1,458	33.0	13,17,887	15,730

```python
countries = ['Pakistan', 'India', 'China', 'Iran', 'Afghanistan']

df_selected = df[df['Country'].isin(countries)]
```

[99] 0.0s



```python
df.columns
```

[101] ✓ 0.0s Python

```
Index(['Country', 'Total Cases', 'New Cases', 'Total Deaths', 'New Deaths',
       'Total Recovered', 'Active Cases', 'Serious, Critical',
       'Tot Cases/1M pop', 'Deaths/1M pop', 'Total Tests', 'Tests/1M pop'],
      dtype='object')
```

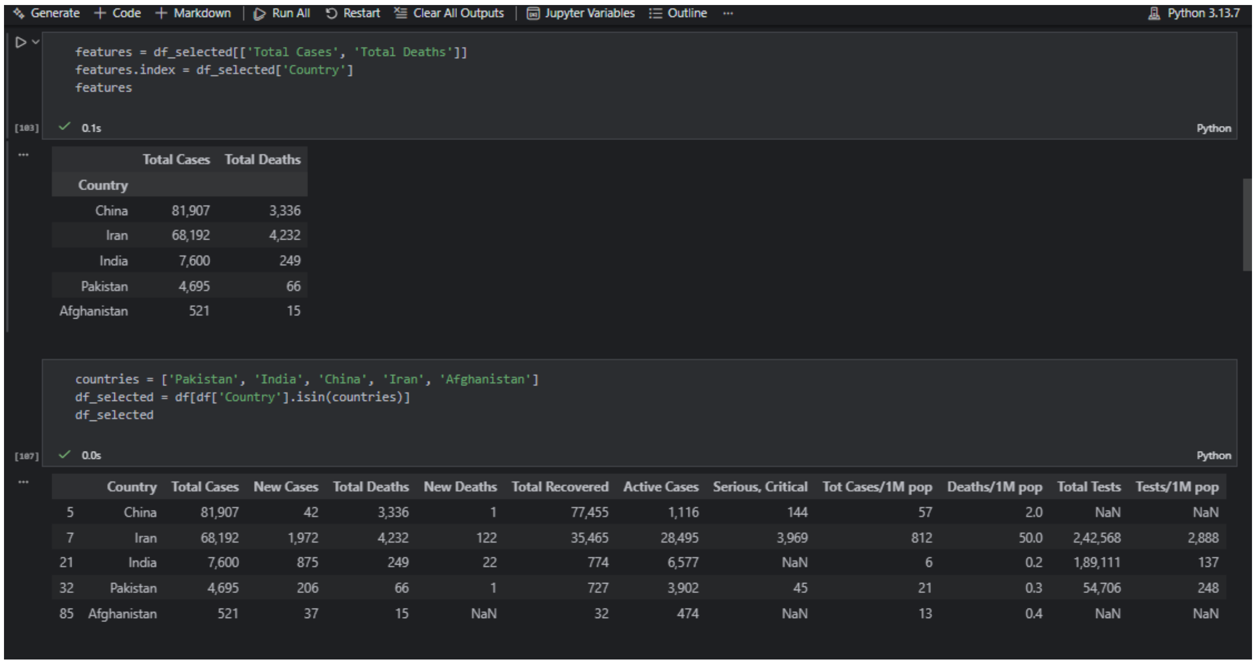
```python
countries = ['Pakistan', 'India', 'China', 'Iran', 'Afghanistan']

df_selected = df[df['Country'].isin(countries)]
df_selected
```

[102] ✓ 0.1s Python

	Country	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	Active Cases	Serious, Critical	Tot Cases/1M pop	Deaths/1M pop	Total Tests	Tests/1M pop
5	China	81,907	42	3,336	1	77,455	1,116	144	57	2.0	NaN	NaN
7	Iran	68,192	1,972	4,232	122	35,465	28,495	3,969	812	50.0	2,42,568	2,888
21	India	7,600	875	249	22	774	6,577	NaN	6	0.2	1,89,111	137
32	Pakistan	4,695	206	66	1	727	3,902	45	21	0.3	54,706	248
85	Afghanistan	521	37	15	NaN	32	474	NaN	13	0.4	NaN	NaN

```python
features = df_selected[['Total Cases', 'Total Deaths']]
features.index = df_selected['Country']
features
```

[103] ✓ 0.1s Python



```python
features = df_selected[['Total Cases', 'Total Deaths']]
features.index = df_selected['Country']
features
```

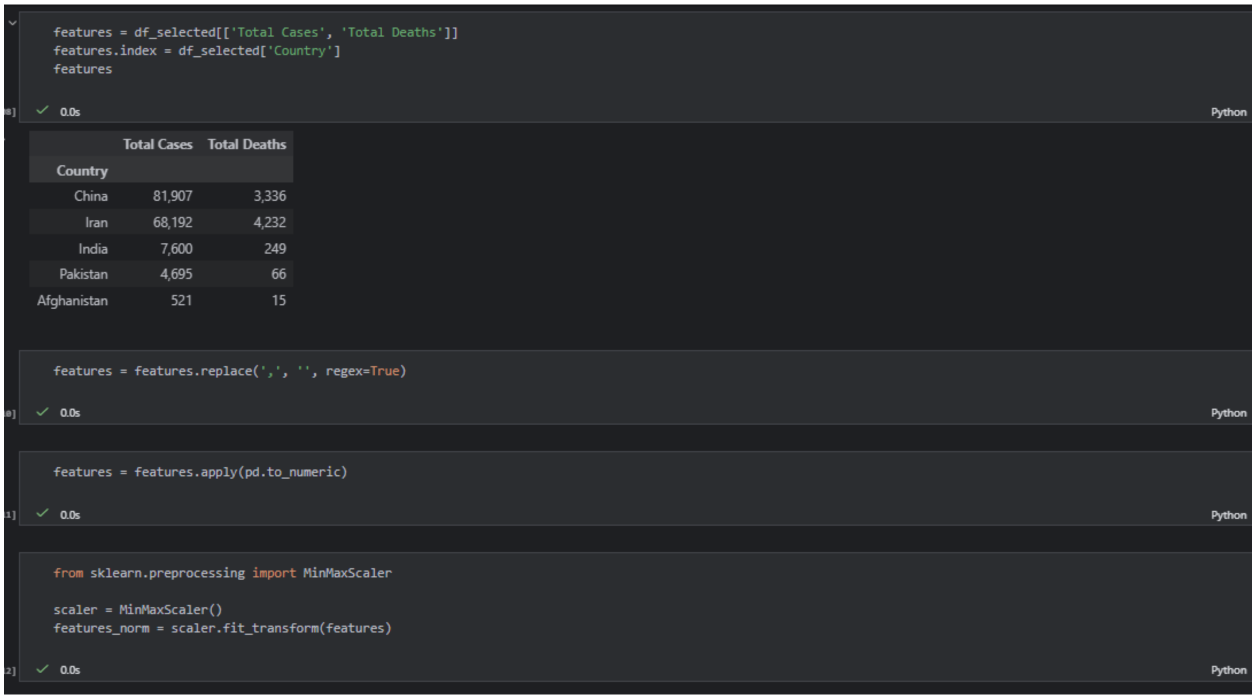
[103] ✓ 0.1s Python

	Total Cases	Total Deaths
Country		
China	81,907	3,336
Iran	68,192	4,232
India	7,600	249
Pakistan	4,695	66
Afghanistan	521	15

```python
countries = ['Pakistan', 'India', 'China', 'Iran', 'Afghanistan']
df_selected = df[df['Country'].isin(countries)]
df_selected
```

[107] ✓ 0.0s Python

	Country	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	Active Cases	Serious, Critical	Tot Cases/1M pop	Deaths/1M pop	Total Tests	Tests/1M pop
5	China	81,907	42	3,336	1	77,455	1,116	144	57	2.0	NaN	NaN
7	Iran	68,192	1,972	4,232	122	35,465	28,495	3,969	812	50.0	2,42,568	2,888
21	India	7,600	875	249	22	774	6,577	NaN	6	0.2	1,89,111	137
32	Pakistan	4,695	206	66	1	727	3,902	45	21	0.3	54,706	248
85	Afghanistan	521	37	15	NaN	32	474	NaN	13	0.4	NaN	NaN



```python
features = df_selected[['Total Cases', 'Total Deaths']]
features.index = df_selected['Country']
features
```

✓ 0.0s Python

Country	Total Cases	Total Deaths
China	81,907	3,336
Iran	68,192	4,232
India	7,600	249
Pakistan	4,695	66
Afghanistan	521	15

```python
features = features.replace(',', '', regex=True)
```

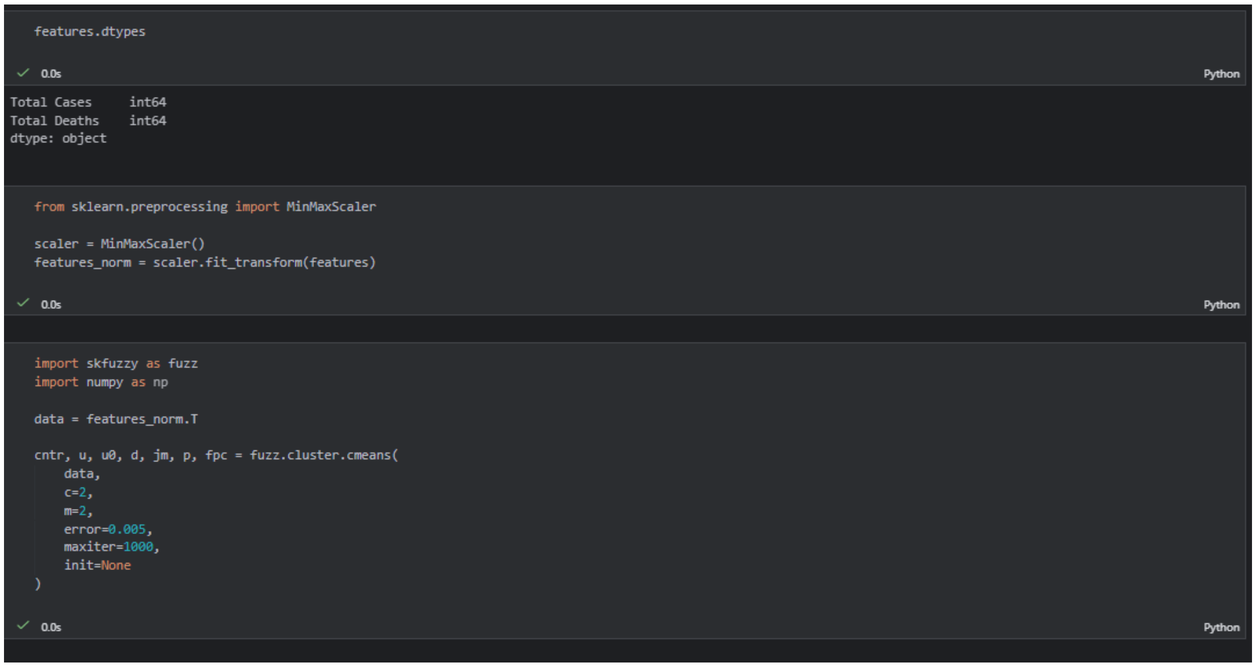
✓ 0.0s Python

```python
features = features.apply(pd.to_numeric)
```

✓ 0.0s Python

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
features_norm = scaler.fit_transform(features)
```

✓ 0.0s Python



```python
features.dtypes
```

✓ 0.0s Python

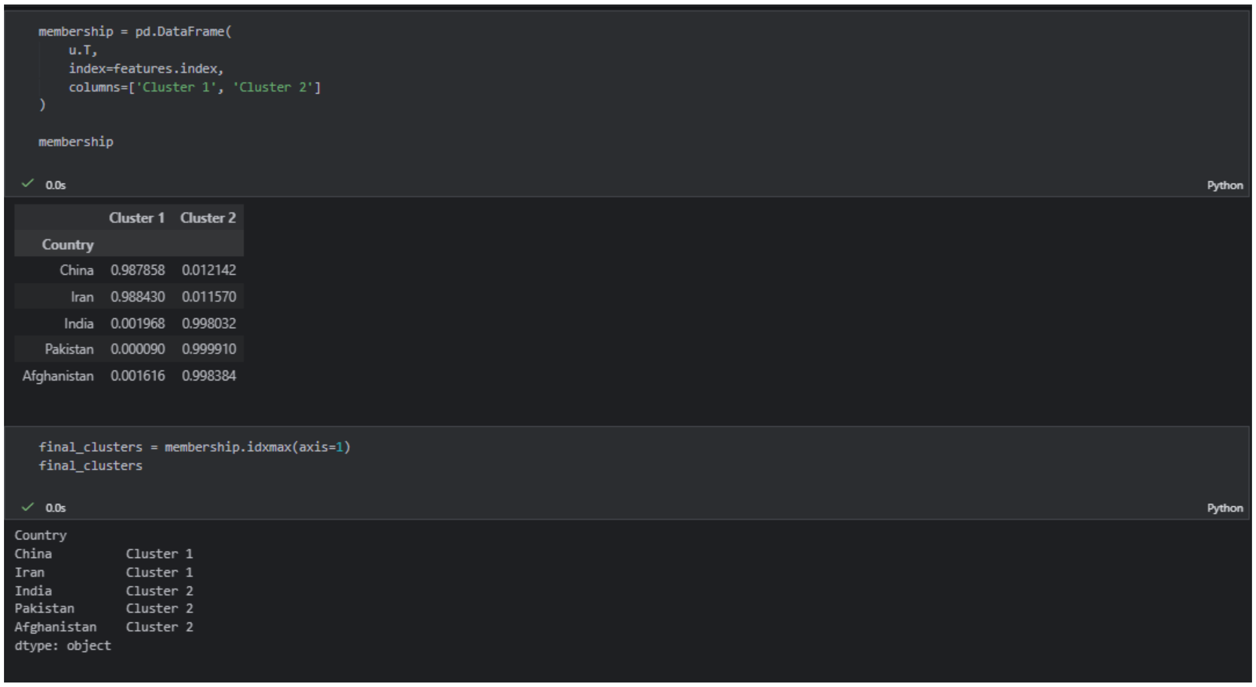
```
Total Cases     int64
Total Deaths    int64
dtype: object
```

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
features_norm = scaler.fit_transform(features)
```

✓ 0.0s Python

```python
import skfuzzy as fuzz
import numpy as np

data = features_norm.T

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    data,
    c=2,
    m=2,
    error=0.005,
    maxiter=1000,
    init=None
)
```

✓ 0.0s Python



```python
membership = pd.DataFrame(
    u.T,
    index=features.index,
    columns=['Cluster 1', 'Cluster 2']
)

membership
```

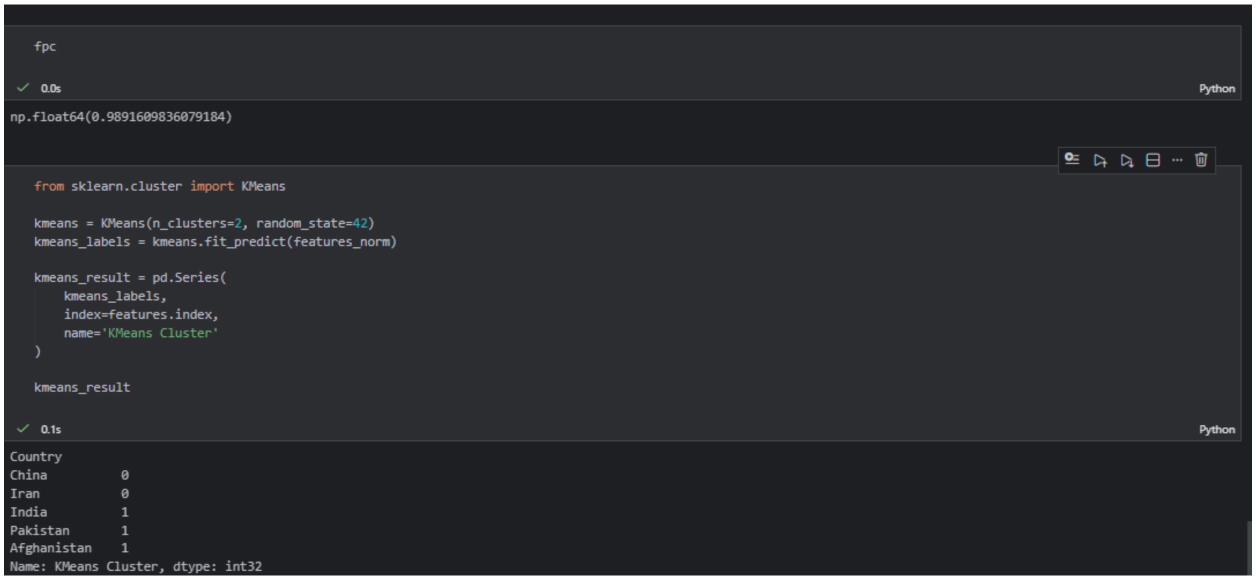
✓ 0.0s Python

	Cluster 1	Cluster 2
Country		
China	0.987858	0.012142
Iran	0.988430	0.011570
India	0.001968	0.998032
Pakistan	0.000090	0.999910
Afghanistan	0.001616	0.998384

```python
final_clusters = membership.idxmax(axis=1)
final_clusters
```

✓ 0.0s Python

```
Country
China          Cluster 1
Iran           Cluster 1
India          Cluster 2
Pakistan       Cluster 2
Afghanistan    Cluster 2
dtype: object
```



```python
fpc
```

✓ 0.0s Python

```
np.float64(0.9891609836079184)
```

```python
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans_labels = kmeans.fit_predict(features_norm)

kmeans_result = pd.Series(
    kmeans_labels,
    index=features.index,
    name='KMeans Cluster'
)

kmeans_result
```

✓ 0.1s Python

```
Country
China          0
Iran           0
India          1
Pakistan       1
Afghanistan    1
Name: KMeans Cluster, dtype: int32
```