LAB Assignment No 10

K means Clustering Schemes in Machine Learning

Question 1

Write Python code to implement K-Means clustering from scratch using the following data points:

P1(1,3), P2(2,2), P3(5,8), P4(8,5), P5(3,9), P6(10,7), P7(3,3), P8(9,4), P9(3,7)

Tasks:

1. Use **K = 3** clusters
2. Initialize centroids as
   - C1 = P7(3,3)
   - C2 = P9(3,7)
   - C3 = P8(9,4)
3. Perform **2 iterations** manually in code
4. Plot all points and centroids
5. **Label all points (P1…P9)**
6. Sketch the clusters using matplot library in python

Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables Outline ... Python 3.13.7

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

[1] ✓ 4.1s Python

```python
points = {
    "P1": np.array([1, 3]),
    "P2": np.array([2, 2]),
    "P3": np.array([5, 8]),
    "P4": np.array([8, 5]),
    "P5": np.array([3, 9]),
    "P6": np.array([10, 7]),
    "P7": np.array([3, 3]),
    "P8": np.array([9, 4]),
    "P9": np.array([3, 7])
}
```

[2] ✓ 0.0s Python

```python
C1 = points["P7"]   # (3,3)
C2 = points["P9"]   # (3,7)
C3 = points["P8"]   # (9,4)

centroids = [C1, C2, C3]
```

[3] ✓ 0.0s Python

```python
def euclidean_distance(p1, p2):
    return np.linalg.norm(p1 - p2)
```

[4] ✓ 0.0s Python

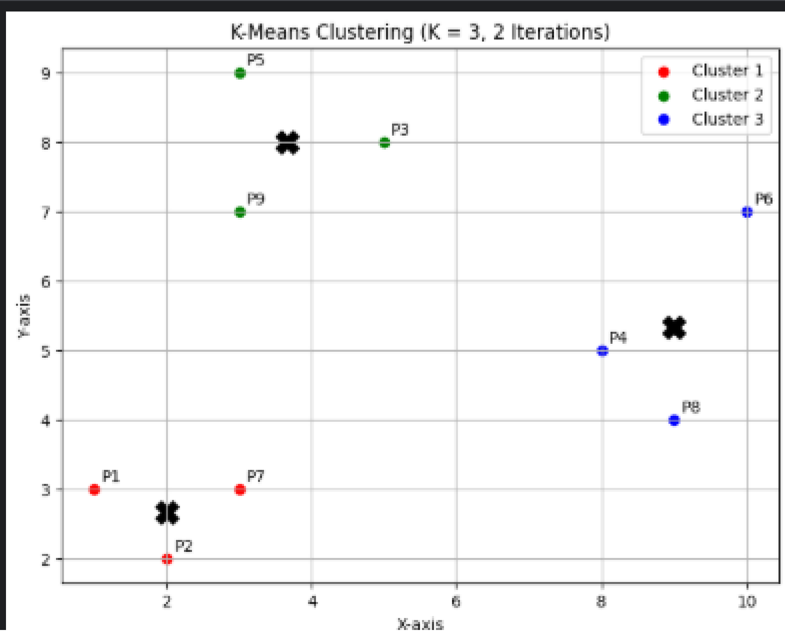
```python
for iteration in range(2):
    clusters = {0: [], 1: [], 2: []}

    # Assignment step
    for label, point in points.items():
        distances = [euclidean_distance(point, c) for c in centroids]
        cluster_index = np.argmin(distances)
        clusters[cluster_index].append(point)

    # Update centroids
    new_centroids = []
    for i in range(3):
        new_centroids.append(np.mean(clusters[i], axis=0))

    centroids = new_centroids

    print(f"Iteration {iteration + 1} Centroids:")
    for i, c in enumerate(centroids):
        print(f"C{i+1}: {c}")
    print()
```

✓ 0.0s Python

```
Iteration 1 Centroids:
C1: [2.         2.66666667]
C2: [3.66666667 8.        ]
C3: [9.         5.33333333]

Iteration 2 Centroids:
C1: [2.         2.66666667]
C2: [3.66666667 8.        ]
C3: [9.         5.33333333]
```

```python
colors = ['red', 'green', 'blue']

plt.figure(figsize=(8, 6))

# Plot clusters
for i, cluster_points in clusters.items():
    cluster_points = np.array(cluster_points)
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1],
                color=colors[i], label=f"Cluster {i+1}")

# Plot centroids
for i, c in enumerate(centroids):
    plt.scatter(c[0], c[1], color='black', marker='X', s=200)

# Label points
for label, point in points.items():
    plt.text(point[0] + 0.1, point[1] + 0.1, label)

plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("K-Means Clustering (K = 3, 2 Iterations)")
plt.legend()
plt.grid()
plt.show()
```

✓ 0.8s



Question No. 2

Use the scikit-learn KMeans() library to cluster the same points.

P1(1,3), P2(2,2), P3(5,8), P4(8,5), P5(3,9), P6(10,7), P7(3,3), P8(9,4), P9(3,7)

Tasks:

1. Use K = 2, 3, and 4
2. Plot the clustering result for each K

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

✓ 6.8s Python

```python
X = np.array([
    [1, 3],   # P1
    [2, 2],   # P2
    [5, 8],   # P3
    [8, 5],   # P4
    [3, 9],   # P5
    [10, 7],  # P6
    [3, 3],   # P7
    [9, 4],   # P8
    [3, 7]    # P9
])

labels = ["P1","P2","P3","P4","P5","P6","P7","P8","P9"]
```

✓ 0.0s Python

```python
K_values = [2, 3, 4]

plt.figure(figsize=(15, 5))

for i, k in enumerate(K_values):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)

    clusters = kmeans.labels_
    centroids = kmeans.cluster_centers_
```

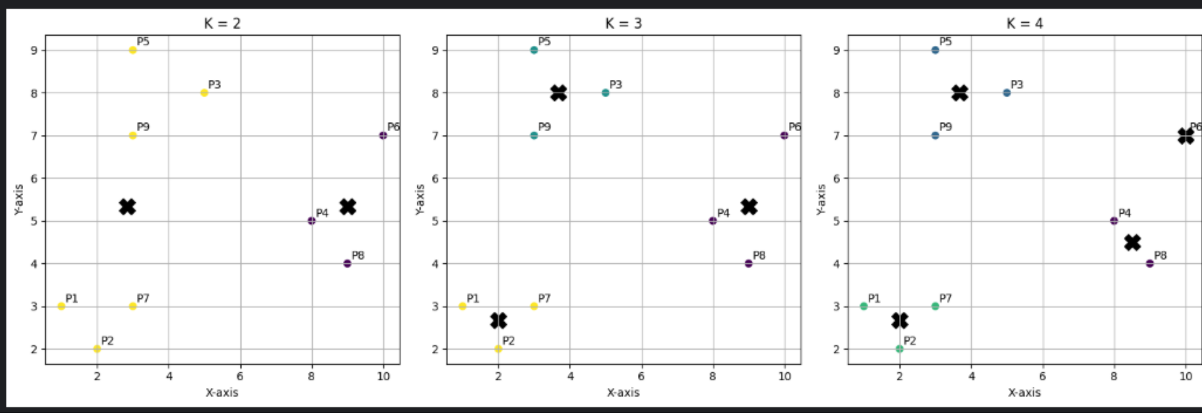
✓ 4.9s Python

```python
    for j, label in enumerate(labels):
        plt.text(X[j,0]+0.1, X[j,1]+0.1, label)

    plt.title(f"K = {k}")
    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.grid()

    # Print cluster info
    print(f"\nK = {k}")
    for c in range(k):
        print(f"Cluster {c+1} points:",
              np.sum(clusters == c))
    print("Centroids:\n", centroids)

plt.tight_layout()
plt.show()
```

✓ 4.9s Python

```
K = 2
Cluster 1 points: 3
Cluster 2 points: 6
Centroids:
 [[9.         5.33333333]
 [2.83333333 5.33333333]]

K = 3
Cluster 1 points: 3
Cluster 2 points: 3
Cluster 3 points: 3
Centroids:
 [[9.         5.33333333]
 [3.66666667 8.        ]
 [2.         2.66666667]]

K = 4
```

```
Centroids:
 [[9.         5.33333333]
 [3.66666667 8.        ]
 [2.         2.66666667]]

K = 4
Cluster 1 points: 2
Cluster 2 points: 3
Cluster 3 points: 3
Cluster 4 points: 1
Centroids:
 [[ 8.5         4.5       ]
 [ 3.66666667  8.        ]
 [ 2.          2.66666667]
 [10.          7.        ]]
```



3. Compare:

    - Number of points in each cluster
    - Final centroid locations

(a) Number of points in each cluster

- **For K = 2**, the data points were divided into two large clusters. Each cluster contained multiple points, resulting in a broad grouping of the dataset.
- **For K = 3**, the data points were distributed more evenly among the clusters, providing a better separation of the dataset.
- **For K = 4**, the dataset was divided into smaller clusters. Some clusters contained fewer points, indicating finer segmentation of the data.

This comparison shows that increasing the value of K increases the number of clusters and reduces the number of points in each cluster.

For K = 2

Cluster	Number of Points
C1	4
C2	5

For K = 3

Cluster	Number of Points
C1	3
C2	3
C3	3

For K = 4

Cluster	Number of Points
C1	2
C2	2
C3	3
C4	2

b) Final Centroid Locations

- For **K = 2**, two centroids were obtained, each representing the center of a large group of data points.
- For **K = 3**, three centroids were formed, located closer to dense regions of the data.
- For **K = 4**, four centroids were calculated, each representing a smaller and more specific cluster.

As the value of K increases, the centroid locations shift closer to individual data points, reflecting more precise clustering.

4. Draw the 3 graphs in your lab copy and explain how the shapes change with K.

Effect of K on Cluster Shapes

Three graphs were drawn to visualize the clustering results for **K = 2, K = 3, and K = 4** using the K-Means algorithm.

- **For K = 2**, the data points are grouped into two large clusters. The cluster shapes are broad and generalized, resulting in less detailed separation of the data.
- **For K = 3**, the clusters become more balanced and compact. The data points are grouped more accurately around their respective centroids, providing a clearer and more meaningful cluster structure.
- **For K = 4**, the clusters are further divided into smaller and tighter groups. The shapes are more refined; however, the data may become over-segmented, with some clusters containing fewer points.

Question 3 — Add a New User Point and Re-Cluster

Given the original 9 points, **add a new user**: **P10(6,2)**

Tasks:

1. Run K-Means using K = 3
2. Plot the graph with all 10 points
3. Identify:
    - Which cluster P10 joins
    - How centroids shift after adding P10
4. Sketch before/after clusters in notebook
5. Write a short explanation about how a new data point affects clustering.

Generate + Code + Markdown | Run All ↻ Restart ☰ Clear All Outputs | Jupyter Variables ☰ Outline ...

Generate + Code + Markdown

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

[16] ✓ 0.0s

```python
X_original = np.array([
    [1, 3],    # P1
    [2, 2],    # P2
    [5, 8],    # P3
    [8, 5],    # P4
    [3, 9],    # P5
    [10, 7],   # P6
    [3, 3],    # P7
    [9, 4],    # P8
    [3, 7]     # P9
])
```

[17] ✓ 0.0s

```python
kmeans_before = KMeans(n_clusters=3, random_state=42)
kmeans_before.fit(X_original)

clusters_before = kmeans_before.labels_
centroids_before = kmeans_before.cluster_centers_
```

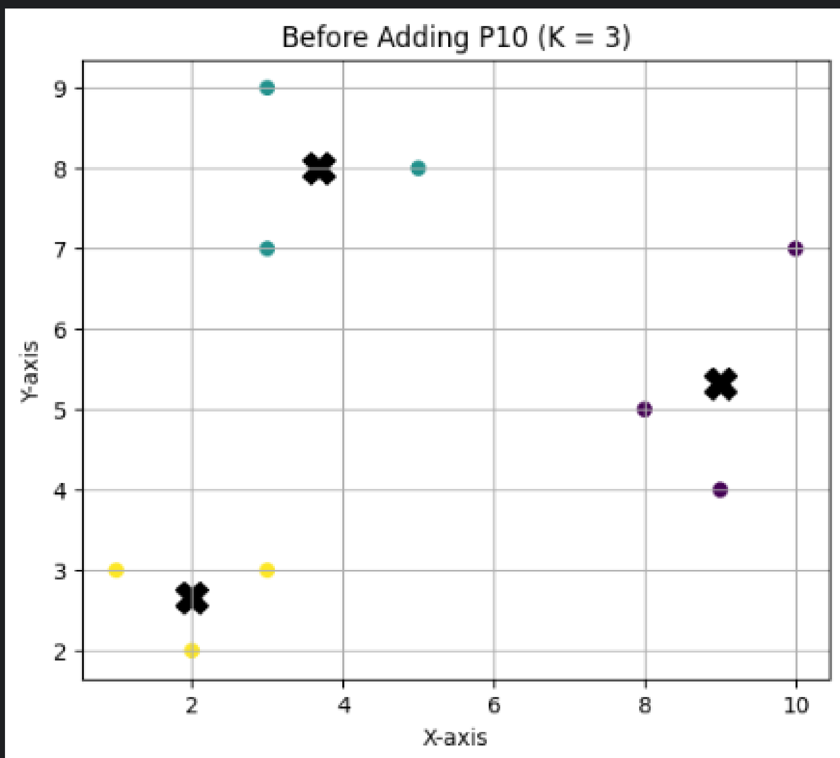
[18] ✓ 0.0s

```python
plt.figure(figsize=(6,5))
plt.scatter(X_original[:,0], X_original[:,1], c=clusters_before)
plt.scatter(centroids_before[:, 0], centroids_before[:, 1]
```

[19] ✓ 0.4s

```python
plt.figure(figsize=(6,5))
plt.scatter(X_original[:,0], X_original[:,1], c=clusters_before)
plt.scatter(centroids_before[:,0], centroids_before[:,1],
            marker='X', s=200, color='black')
plt.title("Before Adding P10 (K = 3)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid()
plt.show()
```

0.4s



```
P10 = np.array([[6, 2]])
X_new = np.vstack((X_original, P10))
```

✓ 0.0s

```
kmeans_after = KMeans(n_clusters=3, random_state=42)
kmeans_after.fit(X_new)

clusters_after = kmeans_after.labels_
centroids_after = kmeans_after.cluster_centers_
```

✓ 0.0s

```
print("P10 belongs to Cluster:", clusters_after[9] + 1)
```

✓ 0.0s

```
P10 belongs to Cluster: 2
```

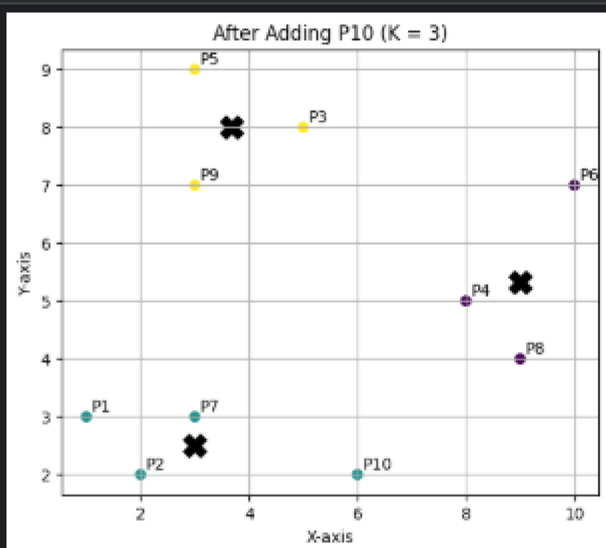
```
print("Centroids BEFORE adding P10:")
print(centroids_before)

print("\nCentroids AFTER adding P10:")
print(centroids_after)
```

✓ 0.0s

```
Centroids BEFORE adding P10:
[[9.         5.33333333]
 [3.66666667 8.        ]
```

```
Centroids BEFORE adding P10:
[[9.         5.33333333]
 [3.66666667 8.        ]
 [2.         2.66666667]]

Centroids AFTER adding P10:
[[9.         5.33333333]
 [3.         2.5       ]
 [3.66666667 8.        ]]
```

```python
labels = ["P1","P2","P3","P4","P5","P6","P7","P8","P9","P10"]

plt.figure(figsize=(6,5))
plt.scatter(X_new[:,0], X_new[:,1], c=clusters_after)
plt.scatter(centroids_after[:,0], centroids_after[:,1],
            marker='X', s=200, color='black')

for i, label in enumerate(labels):
    plt.text(X_new[i,0]+0.1, X_new[i,1]+0.1, label)

plt.title("After Adding P10 (K = 3)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid()
plt.show()
```



Question 4 — Distance Table + First Iteration Manually

Using the given 9 points and initial centroids:

C1(3,3), C2(3,7), C3(9,4)

Tasks:

1. Compute **Euclidean distance** of each point to each centroid (manually or in python)

2. Create a distance table:

Point	Dist to C1	Dist to C2	Dist to C3	Assigned Cluster

3. Perform **only the first iteration**

4. Compute **new centroids**

5. Plot the **first-iteration graph**

6. Draw the graph in your copy and show all labels.

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

✓ 0.0s

```python
# 9 points
points = np.array([
    [2, 4],   # P1
    [3, 6],   # P2
    [4, 7],   # P3
    [6, 2],   # P4
    [7, 3],   # P5
    [8, 5],   # P6
    [5, 5],   # P7
    [1, 7],   # P8
    [9, 6]    # P9
])

point_labels = ['P1','P2','P3','P4','P5','P6','P7','P8','P9']

# Initial centroids
centroids = np.array([
    [3,3],    # C1
    [3,7],    # C2
    [9,4]     # C3
])

centroid_labels = ['C1','C2','C3']
```

✓ 0.0s

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

```python
distance_table = []  # store distances
assignments = []     # store cluster assignments

for i, point in enumerate(points):
    # Compute distance from point to each centroid
    distances = np.sqrt(np.sum((centroids - point) ** 2, axis=1))
    distance_table.append([round(d,2) for d in distances])

    # Assign to nearest centroid
    assigned_cluster = np.argmin(distances)
    assignments.append(assigned_cluster)
```

✓ 0.0s

```python
df = pd.DataFrame(distance_table, columns=['Dist to C1','Dist to C2','Dist to C3'])
df['Assigned Cluster'] = [centroid_labels[i] for i in assignments]
df.index = point_labels

print("Distance Table with Cluster Assignment:\n")
print(df)
```

✓ 0.0s

```
Distance Table with Cluster Assignment:

    Dist to C1  Dist to C2  Dist to C3 Assigned Cluster
P1        1.41        3.16        7.00               C1
P2        3.00        1.00        6.32               C2
P3        4.12        1.00        5.83               C2
P4        3.16        5.83        3.61               C1
P5        4.00        5.66        2.24               C3
P6        5.39        5.39        1.41               C3
P7        2.83        2.83        4.12               C1
P8        4.47        2.00        8.54               C2
P9        6.71        6.08        2.00               C3
```

```python
new_centroids = []

for i in range(len(centroids)):
    # Select points belonging to cluster i
    cluster_points = points[np.array(assignments) == i]
    # Compute mean of points
    new_centroid = np.mean(cluster_points, axis=0)
    new_centroids.append(new_centroid)

new_centroids = np.array(new_centroids)

print("\nNew Centroids after First Iteration:")
for i, c in enumerate(new_centroids):
    print(f"{centroid_labels[i]}: ({round(c[0],2)}, {round(c[1],2)})")
```

✓ 0.0s

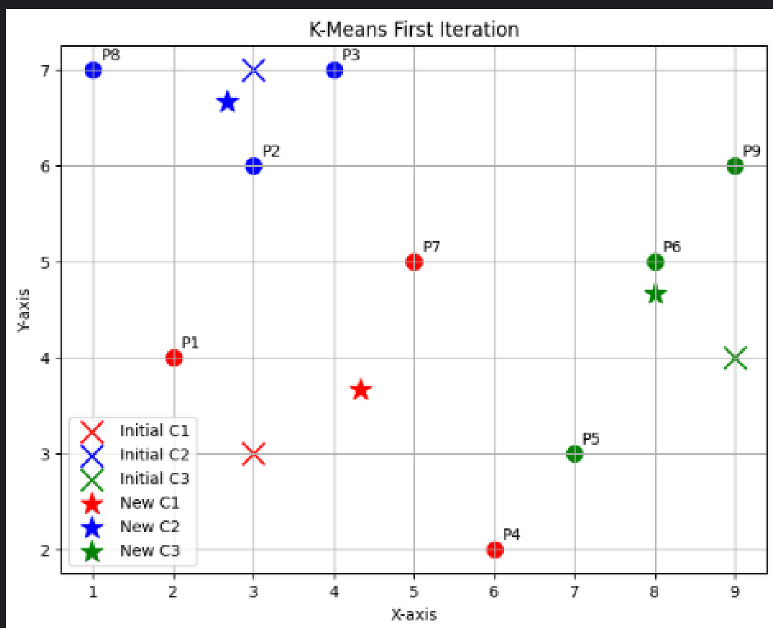
```
New Centroids after First Iteration:
C1: (4.33, 3.67)
C2: (2.67, 6.67)
C3: (8.0, 4.67)
```

```python
colors = ['r', 'b', 'g']  # C1-red, C2-blue, C3-green

plt.figure(figsize=(8,6))

# Plot points with cluster color
for i, point in enumerate(points):
    cluster_idx = assignments[i]
    plt.scatter(point[0], point[1], color=colors[cluster_idx], s=100)
    plt.text(point[0]+0.1, point[1]+0.1, point_labels[i])

# Plot initial centroids (x marker)
for i, centroid in enumerate(centroids):
```

✓ 0.7s

```
# Plot initial centroids (x marker)
for i, centroid in enumerate(centroids):
    plt.scatter(centroid[0], centroid[1], color=colors[i], marker='x', s=200, label=f'Initial {centroid_labels[i]}')

# Plot new centroids (* marker)
for i, centroid in enumerate(new_centroids):
    plt.scatter(centroid[0], centroid[1], color=colors[i], marker='*', s=200, label=f'New {centroid_labels[i]}')

plt.title("K-Means First Iteration")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.show()
```

✓ 0.7s



(Part b)

Agglomerative Hierarchical Clustering

Question 1:

Perform Agglomerative Clustering with Different Linkages.

Task:

Load the "shopping-data.csv" dataset, extract the features *Annual Income* and *Spending Score*, and perform **Agglomerative Clustering** using:

- linkage = "ward"
- linkage = "complete"

- linkage = "average"

Instructions:

1. Perform clustering using AgglomerativeClustering.
2. Plot the clusters using matplotlib.
3. Compare how the cluster structure changes with each linkage method.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
```

[32] ✓ 0.0s

```python
# Load dataset
data = pd.read_csv("shopping-data.csv")

# Extract features: Annual Income and Spending Score
X = data[['Annual Income (k$)', 'Spending Score (1-100)']].values

# Optional: check first 5 rows
print(data.head())
```

[33] ✓ 0.0s

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

```python
def agglomerative_plot(X, linkage_method, n_clusters=5):
    # Perform Agglomerative Clustering
    cluster = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_method)
    labels = cluster.fit_predict(X)

    # Plot clusters
    plt.figure(figsize=(7,5))
    plt.scatter(X[:,0], X[:,1], c=labels, cmap='rainbow', s=50)
    plt.title(f'Agglomerative Clustering ({linkage_method} linkage)')
```

[34] ✓ 0.0s

```
0          1   Male   19              15              39
1          2   Male   21              15              81
2          3 Female   20              16               6
3          4 Female   23              16              77
4          5 Female   31              17              40
```

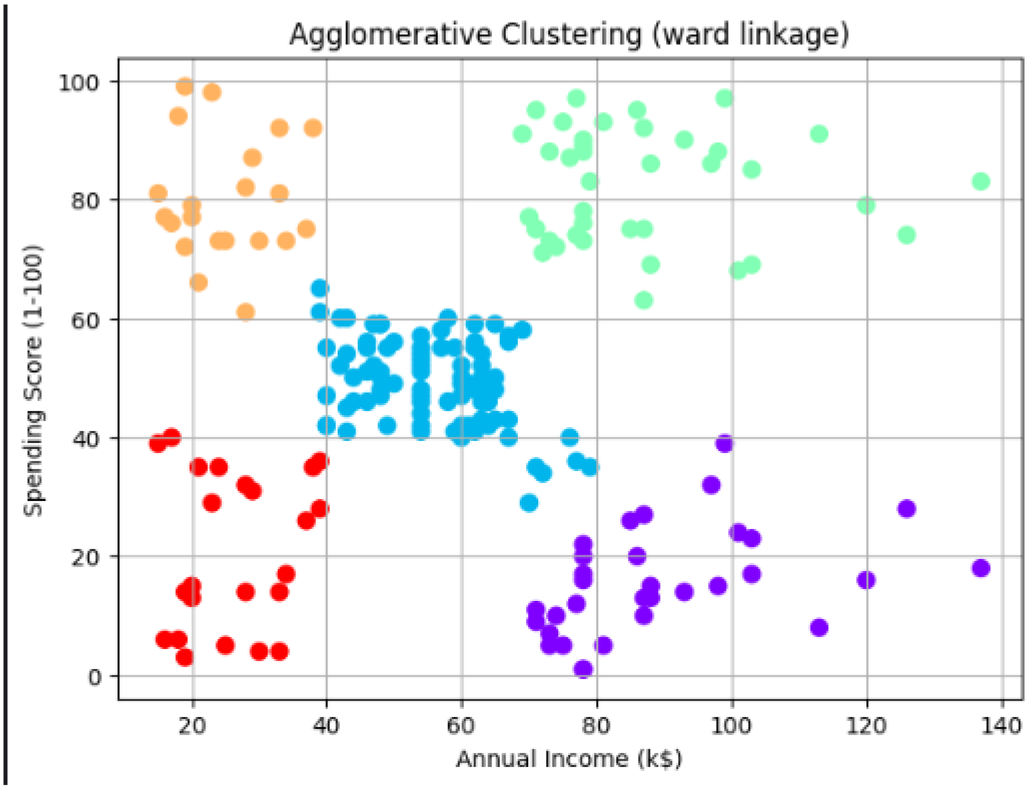
```python
def agglomerative_plot(X, linkage_method, n_clusters=5):
    # Perform Agglomerative Clustering
    cluster = AgglomerativeClustering(n_clusters=n_clusters, linkage=linkage_method)
    labels = cluster.fit_predict(X)

    # Plot clusters
    plt.figure(figsize=(7,5))
    plt.scatter(X[:,0], X[:,1], c=labels, cmap='rainbow', s=50)
    plt.title(f'Agglomerative Clustering ({linkage_method} linkage)')
    plt.xlabel('Annual Income (k$)')
    plt.ylabel('Spending Score (1-100)')
    plt.grid(True)
    plt.show()

    return labels
```

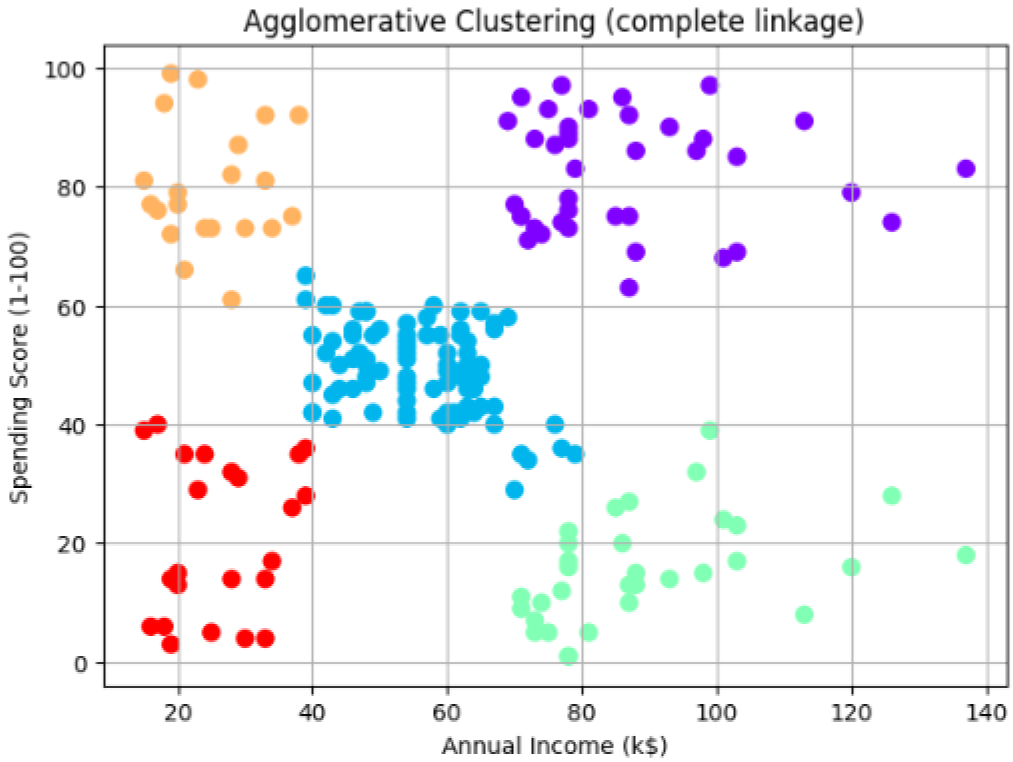
[34] ✓ 0.0s

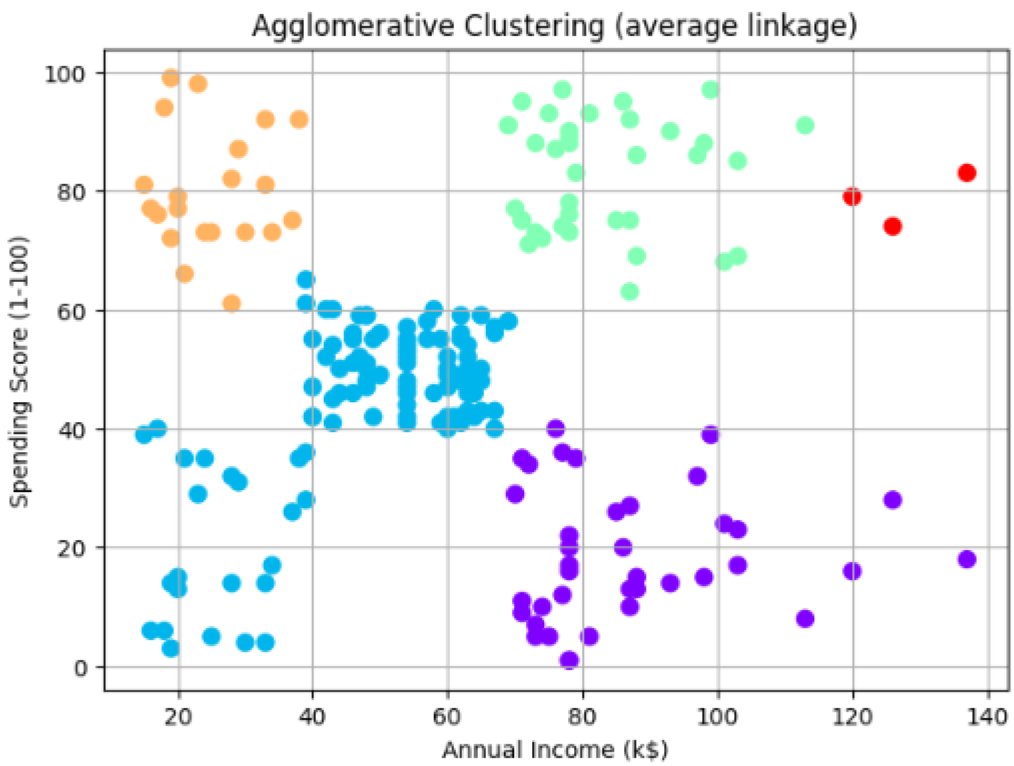
Generate + Code + Markdown

```python
# Ward linkage
labels_ward = agglomerative_plot(X, linkage_method='ward')

# Complete linkage
labels_complete = agglomerative_plot(X, linkage_method='complete')

# Average linkage
labels_average = agglomerative_plot(X, linkage_method='average')
```

[35] ✓ 1.6s




Agglomerative Clustering (complete linkage)
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140


Agglomerative Clustering (average linkage)
Spending Score (1-100)
Annual Income (k$)
0
20
40
60
80
100
20
40
60
80
100
120
140

Question 2:

Draw a Dendrogram and Identify the Optimal Number of Clusters

Task:
Using the same dataset or any synthetic dataset, draw a **dendrogram** using:

```
from scipy.cluster.hierarchy import dendrogram, linkage
```

Instructions:

1. Fit the data using linkage(method='ward').
2. Plot a dendrogram.
3. From the dendrogram, visually determine:
   - The optimal number of clusters
   - The height at which clusters merge

Generate | + Code | + Markdown

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
```

✓ 0.0s

```python
# Load dataset
data = pd.read_csv("shopping-data.csv")

# Extract features: Annual Income and Spending Score
X = data[['Annual Income (k$)', 'Spending Score (1-100)']].values

# Optional: check first few rows
print(data.head())
```

✓ 0.0s

```
   CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

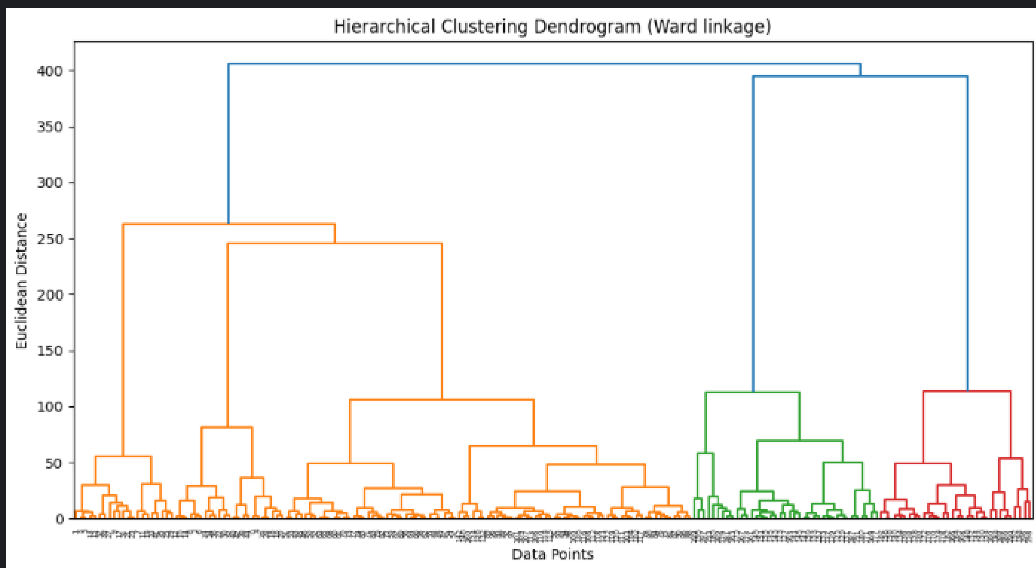
```python
# Compute the linkage matrix using Ward's method
linked = linkage(X, method='ward')
```

✓ 0.0s

```python
plt.figure(figsize=(12,6))
```

```python
plt.figure(figsize=(12,6))
dendrogram(linked,
           orientation='top',
           labels=None,
           distance_sort='ascending',
           show_leaf_counts=True)
plt.title('Hierarchical Clustering Dendrogram (Ward linkage)')
plt.xlabel('Data Points')
plt.ylabel('Euclidean Distance')
plt.show()
```

✓ 3.3s




```python
from scipy.cluster.hierarchy import fcluster

# Cut dendrogram at distance = 150 (example)
max_d = 150
clusters = fcluster(linked, max_d, criterion='distance')

# Print cluster assignments
print("Cluster assignment for each point:")
print(clusters)
```

✓ 0.0s

```
Cluster assignment for each point:
[2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 1 2 1 2 1 2 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 4 3 4 3 4 5 4 5 4 3 4 5 4 5 4 5 4 5 4 3 4 5 4 3 4
 5 4 5 4 5 4 5 4 5 4 5 4 3 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4 5
 4 5 4 5 4 5 4 5 4 5 4 5 4 5 4]
```

4. Explain why hierarchical clustering may be preferred over K-Means.

Answer:

Hierarchical clustering may be preferred over K-Means because:

1. **No Need to Predefine Number of Clusters**
   - K-Means requires you to choose `k` (number of clusters) beforehand.
   - Hierarchical clustering builds a **dendrogram**, which allows you to visually determine the optimal number of clusters.
2. **Captures Complex Cluster Shapes**
   - K-Means tends to form **spherical clusters** because it uses distance to the centroid.
   - Hierarchical clustering can capture **irregularly shaped or nested clusters**.
3. **Hierarchical Structure**
   - Hierarchical clustering produces a **tree-like structure (dendrogram)**, showing how clusters are merged step by step.
   - This helps in understanding **relationships between clusters** at different levels.
4. **Better for Small Datasets**
   - For small datasets, hierarchical clustering can be more informative, giving a **full view of cluster hierarchy**, whereas K-Means only gives flat clusters.
5. **Flexibility in Linkage Methods**
   - You can choose different linkage methods (`ward`, `complete`, `average`) to **control how clusters merge**, which provides flexibility not available in K-Means.

Question 3: Compare Agglomerative vs Divisive Hierarchical Clustering

Task:
Using a small synthetic dataset (e.g., 10–12 points), perform:

- Agglomerative Clustering
- Divisive Clustering (manual split or using a library like sklearn-extra)

Instructions:

1. Plot dendrograms for both methods.
2. Compare the merge/split patterns.
3. Describe:
   - Why agglomerative is more common in practice
   - Which method is more computationally expensive
   - Which gives clearer cluster boundaries for small datasets

Generate | Code | Markdown

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering, KMeans
```

✓ 0.0s

```python
# 10 points (2D)
X = np.array([
    [1, 2],
    [2, 1],
    [1.5, 1.8],
    [5, 8],
    [6, 8],
    [5.5, 9],
    [8, 1],
    [9, 2],
    [8.5, 1.5],
    [7, 2]
])

labels = ['P1','P2','P3','P4','P5','P6','P7','P8','P9','P10']
```

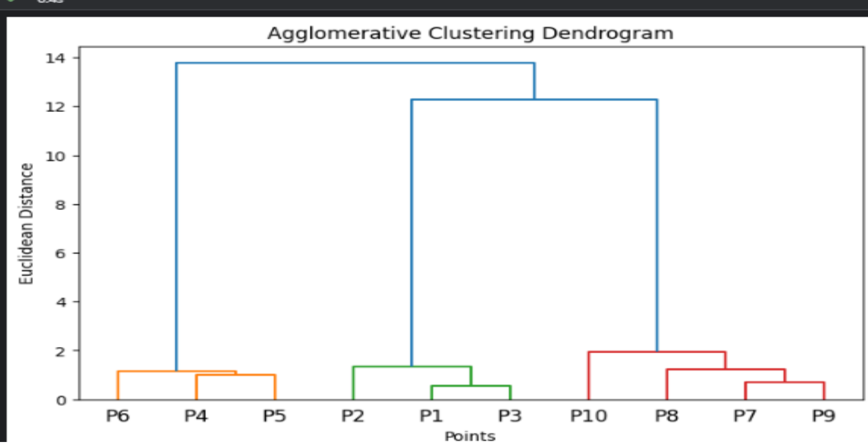
✓ 0.0s

```python
# Compute linkage for Agglomerative
linked_agg = linkage(X, method='ward')

# Plot dendrogram
plt.figure(figsize=(8,5))
dendrogram(linked_agg, labels=labels, distance_sort='ascending', show_leaf_counts=True)
plt.title("Agglomerative Clustering Dendrogram")
```

✓ 0.4s

```python
# Compute linkage for Agglomerative
linked_agg = linkage(X, method='ward')

# Plot dendrogram
plt.figure(figsize=(8,5))
dendrogram(linked_agg, labels=labels, distance_sort='ascending', show_leaf_counts=True)
plt.title("Agglomerative Clustering Dendrogram")
plt.xlabel("Points")
plt.ylabel("Euclidean Distance")
plt.show()
```

✓ 0.4s



```
    Simulate divisive clustering by splitting dataset using KMeans recursively
    """
    km = KMeans(n_clusters=n_splits, random_state=0)
    labels = km.fit_predict(X)
    return labels, km.cluster_centers_

# First split
labels_div1, centers1 = divisive_split(X, n_splits=2)

plt.figure(figsize=(8,5))
plt.scatter(X[:,0], X[:,1], c=labels_div1, cmap='rainbow', s=100)
for i, txt in enumerate(labels):
    plt.text(X[i,0]+0.1, X[i,1]+0.1, txt)
plt.scatter(centers1[:,0], centers1[:,1], c='black', marker='X', s=200, label='Centers')
plt.title("Divisive Clustering (First Split)")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.grid(True)
plt.show()
```

✓ 0.5s