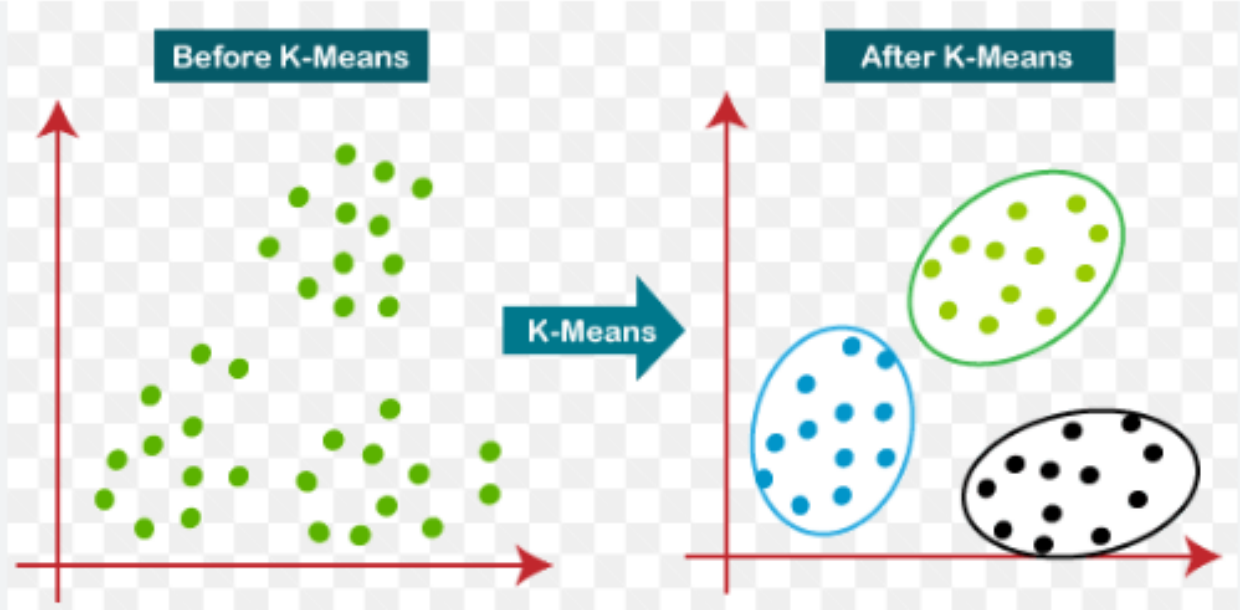
LAB Assignment No 10

K means Clustering Schemes in Machine Learning

In this lab, students will learn and implement K-Means Clustering, an unsupervised machine learning algorithm used to group data points into K distinct clusters based on similarity. Unlike supervised learning methods, K-Means does not require labeled data. Students will first apply K-Means on a small numerical dataset to understand clustering behavior, then use it on real-world datasets to visualize cluster formation and analyze results. Model performance will be interpreted using visualization and cluster characteristics.

Introduction

K-Means Clustering is an **unsupervised learning algorithm** that partitions a dataset into **K clusters**, where each data point belongs to the cluster with the nearest mean (centroid).



Working of K-Means:

1. Select the number of clusters **K**
2. Initialize K centroids randomly
3. Assign each data point to the nearest centroid
4. Update centroids by computing the mean of assigned points
5. Repeat steps 3 and 4 until convergence

Key Concepts:

- **Centroid** – center of a cluster
- **Inertia** – sum of squared distances within clusters

- **Elbow Method** – technique to choose optimal K

Applications:

- Customer segmentation
- Image compression
- Market basket analysis
- Document clustering

Solved Examples

Example 1: K-Means Clustering on a Simple Dataset

Apply K-Means clustering to group students based on **marks and attendance**

Solution:

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

# Sample data: Marks vs Attendance
X = np.array([
   [40, 60],
   [45, 65],
   [70, 80],
   [75, 85],
   [90, 95],
   [85, 90]
])
# Apply K-Means
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)
labels = kmeans.labels_

# Plot clusters
plt.scatter(X[:,0], X[:,1], c=labels)
plt.scatter(kmeans.cluster_centers_[:,0],       kmeans.cluster_centers_[:,1],
marker='X')
plt.xlabel("Marks")
plt.ylabel("Attendance")
plt.title("K-Means Clustering (K=2)")
plt.show()
```

Explanation

The algorithm groups students into clusters based on similarity in marks and attendance.

Example 2: Choosing Optimal K Using Elbow Method

Use the **Elbow Method** to determine the optimal number of clusters.

Solution:

```
inertia = []

for k in range(1, 6):
   kmeans = KMeans(n_clusters=k, random_state=42)
   kmeans.fit(X)
   inertia.append(kmeans.inertia_)

# Plot Elbow Curve
plt.plot(range(1,6), inertia, marker='o')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method")
plt.show()
```

Explanation

The “elbow point” in the graph suggests the best value of **K** where further increase does not significantly reduce inertia.

Example 3: K-Means Clustering on Iris Dataset

Apply K-Means clustering to the **Iris dataset** and visualize the clusters.

Solution:

```
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # Use first two features for visualization

# Apply K-Means
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(X)

# Visualize clusters
plt.scatter(X[:,0], X[:,1], c=labels)
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
        marker='X')
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.title("K-Means Clustering on Iris Dataset")
plt.show()
```

Explanation

K-Means successfully groups data into three clusters corresponding to Iris species.

Lab Assignment No. 10

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

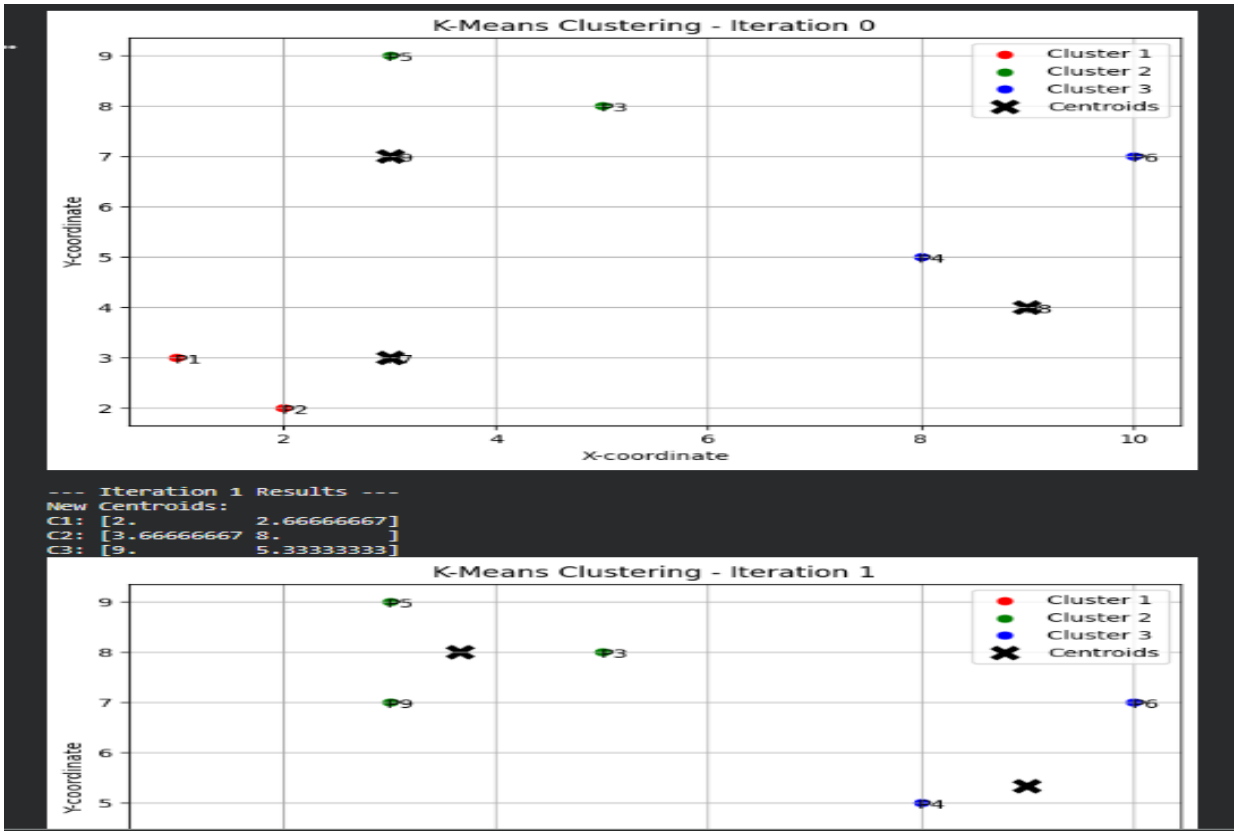
Code:

```
import numpy as np
import matplotlib.pyplot as plt
points = np.array([ [1, 3], [2, 2], [5, 8], [8, 5], [3, 9], [10, 7], [3, 3], [9, 4], [3, 7]])
point_labels = [f'P{i+1}' for i in range(len(points))]K = 3 centroids = np.array([
points[6], points[8], points[7] ])
def assign_clusters(data, centroids):
```

```
def euclidean_distance(a, b):
"""Computes the Euclidean distance between two points."""
return np.sqrt(np.sum((a - b)**2))
assignments = []
distances = []
for point in data:
dist_to_centroids = [euclidean_distance(point, c)
for c in centroids]
assignments.append(np.argmin(dist_to_centroids))
distances.append(dist_to_centroids)
return np.array(assignments),
np.array(distances)
def update_centroids(data, assignments, K):
new_centroids = []
for k in range(K):
cluster_points = data[assignments == k]
if len(cluster_points) > 0:
new_centroids.append(cluster_points.mean(axis=0))
else: new_centroids.append(centroids[k])
return np.array(new_centroids)
def plot_clusters(data, assignments, centroids, iteration):
plt.figure(figsize=(8, 6))
colors = ['r', 'g', 'b', 'y', 'c', 'm']
for i in range(K):
cluster_points = data[assignments == i]
```

```
plt.scatter(cluster_points[:, 0], cluster_points[:, 1],

c=colors[i], label=f'Cluster {i+1}')

plt.scatter(centroids[:, 0],

centroids[:, 1],

marker='X', s=200, c='k',

label='Centroids', edgecolor='w')

for i, txt in enumerate(point_labels):

plt.annotate(txt, (data[i, 0], data[i, 1]),

textcoords="offset points", xytext=(5,-5), ha='center')

plt.title(f'K-Means Clustering - Iteration {iteration}') plt.xlabel('X-coordinate')

plt.ylabel('Y-coordinate')

plt.legend()

plt.grid(True) plt.show()

print("--- Initial State ---")

assignments_0, _ = assign_clusters(points, centroids)

plot_clusters(points, assignments_0, centroids, 0)

assignments_1, _ = assign_clusters(points, centroids) centroids_1 = update_centroids(points, assignments_1, K) print(f"\n--- Iteration 1 Results ---")

print(f"New Centroids:\nC1: {centroids_1[0]}\nC2: {centroids_1[1]}\n

C3: {centroids_1[2]}") centroids = centroids_1 # Update centroids for the next iteration

plot_clusters(points, assignments_1, centroids_1, 1)

assignments_2, _ = assign_clusters(points, centroids)

centroids_2 = update_centroids(points, assignments_2, K)

print(f"\n--- Iteration 2 Results ---")

print(f"New Centroids:\nC1: {centroids_2[0]}\n

C2: {centroids_2[1]}\nC3: {centroids_2[2]}")

centroids = centroids_2 # Final centroids

plot_clusters(points, assignments_2, centroids_2, 2)
```

output:



Question No. 2

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

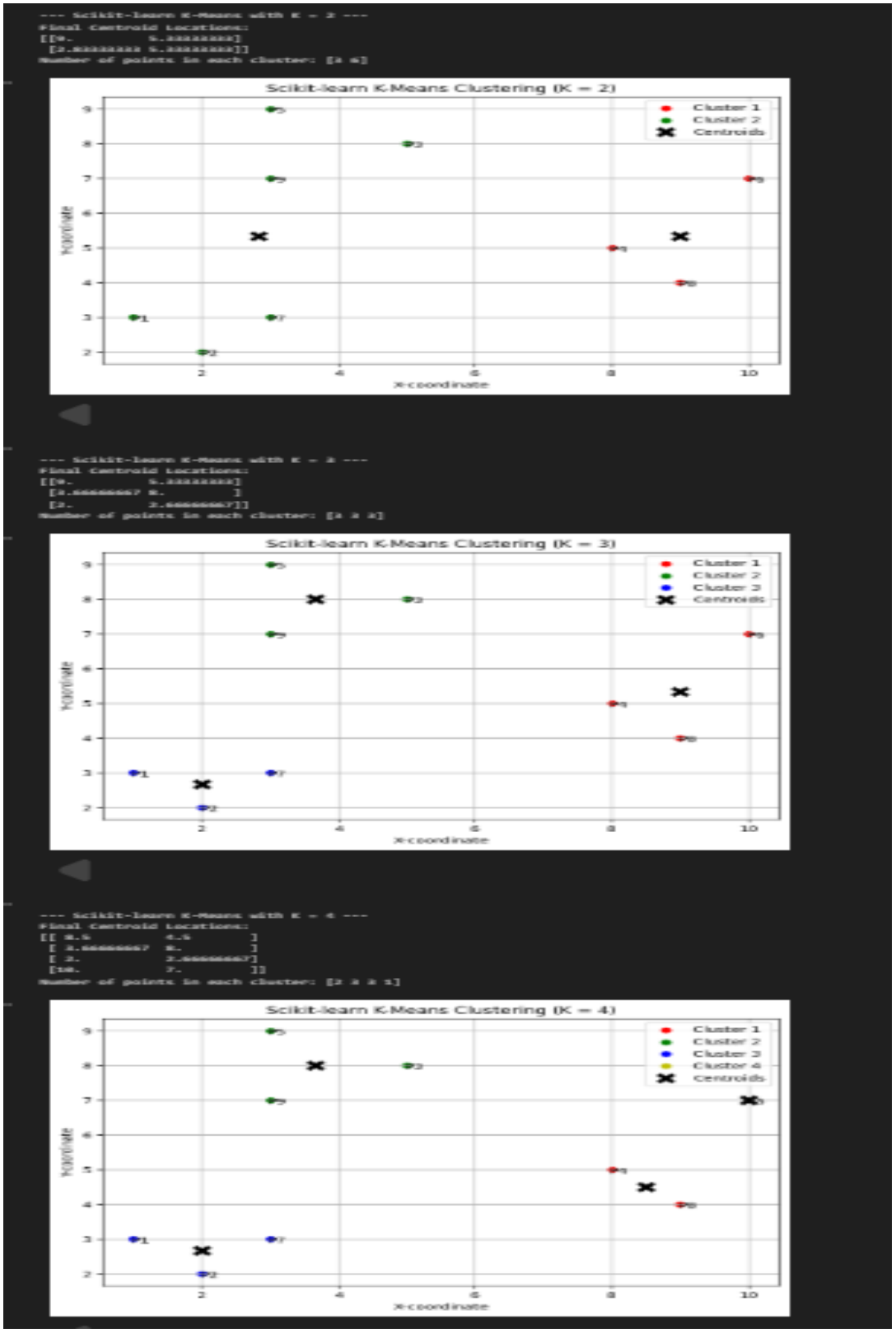
P1(1,3), P2(2,2), P3(5,8), P4(8,5), P5(3,9), P6(10,7), P7(3,3), P8(9,4), P9(3,7)

Tasks:

1. Use K = 2, 3, and 4
2. Plot the clustering result for each K
3. Compare:
    - Number of points in each cluster
    - Final centroid locations
4. Draw the 3 graphs in your lab copy and explain how the shapes change with K.

```
Def run_sk_learn _kmeans(data, k_val, point_labels):
"""Runs Scikit-learn KMeans for a given K and plots the result."""
 # n_init='auto' and random_state for reproducibility
kmeans = KMeans(n_clusters=k_val,
random_state=42, n_init='auto')
kmeans.fit(data) labels = kmeans.labels_ centroids = kmeans.cluster_centers_
  cluster_counts = np.bincount(labels)
print(f"\n--- Scikit-learn K-Means with K = {k_val} ---")
print(f"Final Centroid Locations:\n{centroids}")
print(f"Number of points in each cluster: {cluster_counts}")
plt.figure(figsize=(8, 6)) colors = ['r', 'g', 'b', 'y']
for i in range(k_val):
cluster_points = data[labels == i]
plt.scatter(cluster_points[:, 0], cluster_points[:, 1],
c=colors[i], label=f'Cluster {i+1}') plt.scatter(centroids[:, 0], centroids[:, 1],
marker='X', s=200, c='k', label='Centroids',
edgecolor='w', zorder=10) for i, txt in enumerate(point_labels):
plt.annotate(txt, (data[i, 0], data[i, 1]),
textcoords="offset points",
xytext=(5,-5), ha='center')
plt.title(f'Scikit-learn K-Means Clustering (K = {k_val})')
plt.xlabel('X-coordinate')
plt.ylabel('Y-coordinate')
plt.legend()
plt.grid(True)
plt.show()
run_sklearn_kmeans(POINTS_9, 2, POINT_LABELS_9)
run_sklearn_kmeans(POINTS_9, 3, POINT_LABELS_9)
run_sklearn_kmeans(POINTS_9, 4, POINT_LABELS_9)
```

Output:


--- Scikit-learn K-Means with K = 2 ---
Final Centroid Locations:
Number of points in each cluster: [4 6]
Scikit-learn K-Means Clustering (K = 2)
Cluster 1
Cluster 2
Centroids
Y-coordinate
X-coordinate
--- Scikit-learn K-Means with K = 3 ---
Final Centroid Locations:
Number of points in each cluster: [3 3 4]
Scikit-learn K-Means Clustering (K = 3)
Cluster 1
Cluster 2
Cluster 3
Centroids
Y-coordinate
X-coordinate
--- Scikit-learn K-Means with K = 4 ---
Final Centroid Locations:
Number of points in each cluster: [2 2 2 5]
Scikit-learn K-Means Clustering (K = 4)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids
Y-coordinate
X-coordinate

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

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

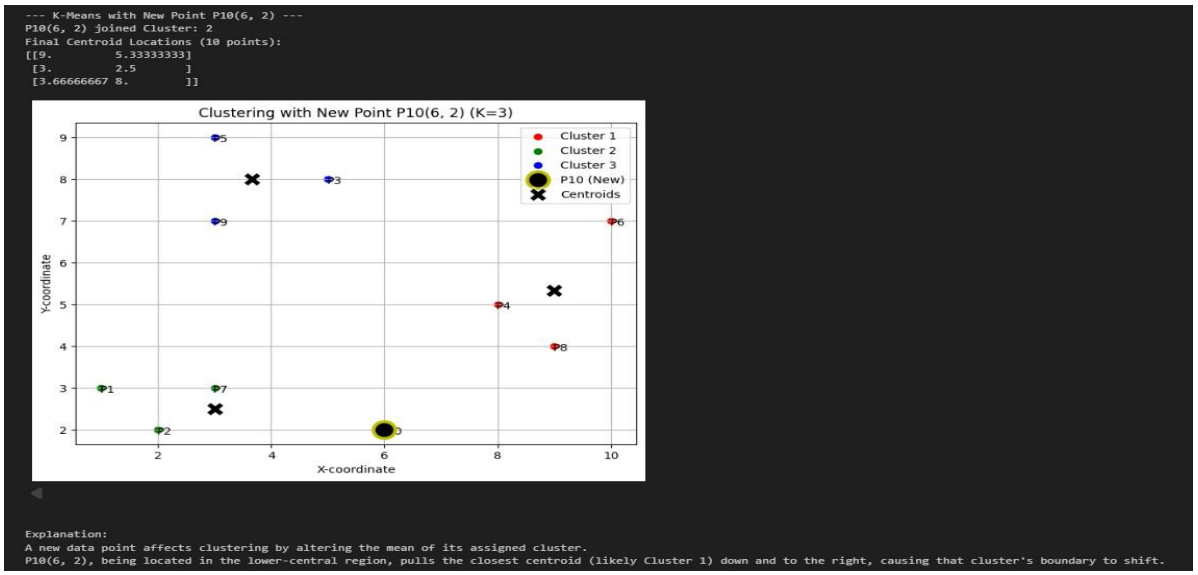
Tasks:

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.

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

points_10 = np.vstack([POINTS_9, P10]) point_labels_10 = POINT_LABELS_9 + ['P10']

kmeans_q3 = KMeans(n_clusters=K_VALUE, random_state=42, n_init='auto')
kmeans_q3.fit(points_10)

labels_q3 = kmeans_q3.labels_ centroids_q3 = kmeans_q3.cluster_centers_
P10_cluster_index = labels_q3[-1]

print(f"--- K-Means with New Point P10(6, 2) ---")

print(f"P10(6, 2) joined Cluster: {P10_cluster_index + 1}")

print(f"Final Centroid Locations (10 points):\n{centroids_q3}") plt.figure(figsize=(8, 6))

colors = ['r', 'g', 'b']

for i in range(K_VALUE):

cluster_points = points_10[labels_q3 == i]

plt.scatter(cluster_points[:, 0], cluster_points[:, 1], c=colors[i],

label=f'Cluster {i+1}')
```

```
plt.scatter(P10[0], P10[1],
marker='o', s=300, c='k', edgecolor='y', linewidth=3,
label='P10 (New)', zorder=10)
plt.scatter(centroids_q3[:, 0], centroids_q3[:, 1], marker='X', s=200, c='k',
label='Centroids', edgecolor='w', zorder=10)
for i, txt in enumerate(point_labels_10):
plt.annotate(txt, (points_10[i, 0], points_10[i, 1]),
textcoords="offset points", xytext=(5,-5), ha='center')
plt.title('Clustering with New Point P10(6, 2) (K=3)')
 plt.xlabel('X-coordinate')
plt.ylabel('Y-coordinate') plt.legend()
plt.grid(True) plt.show()
print("\nExplanation:")
print("A new data point affects clustering by altering the mean of its assigned cluster.")
print("P10(6, 2), being located in the lower-central region, pulls the closest centroid (likely Cluster 1) down and to the right, causing that cluster's boundary to shift.")
```

Output:



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:

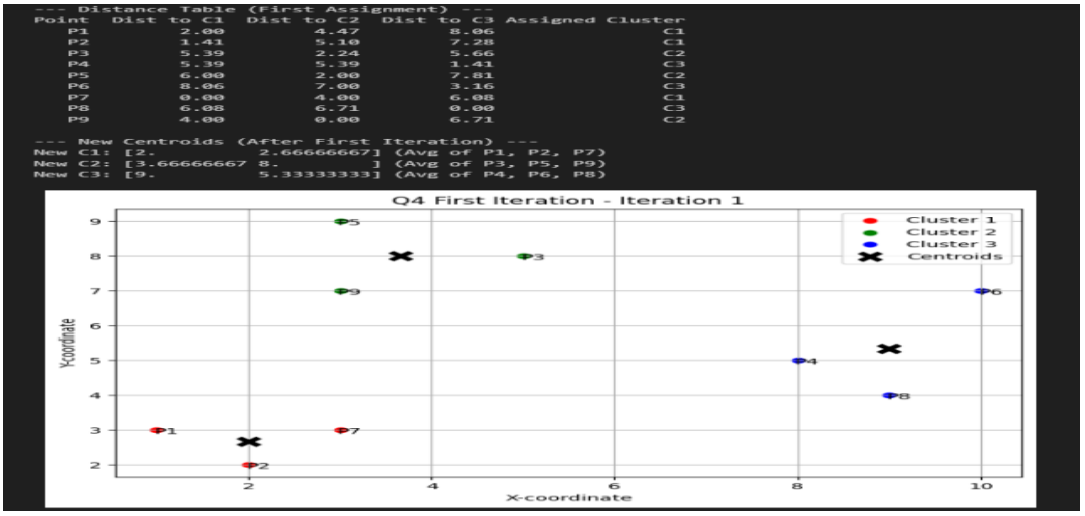
Point Dist to C1 Dist to C2 Dist to C3 Assigned Cluster

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.

```
C1_Q4 = np.array([3, 3])
C2_Q4 = np.array([3, 7])
C3_Q4 = np.array([9, 4])
centroids_Q4 = np.array([C1_Q4, C2_Q4, C3_Q4]
) distances_Q4 = []
assignments_Q4 = []
for i,
 point in enumerate(POINTS_9):
dist_c1 = euclidean_distance(point, C1_Q4) dist_c2 = euclidean_distance(point, C2_Q4)
dist_c3 = euclidean_distance(point, C3_Q4)
 dists = [dist_c1, dist_c2, dist_c3]
assigned_cluster_index = np.argmin(dists)
 distances_Q4.append(dists)
assignments_Q4.append(assigned_cluster_index)
```

```
df_distances = pd.DataFrame(distances_Q4,

columns=['Dist to C1', 'Dist to C2', 'Dist to C3'])

df_distances['Point'] = POINT_LABELS_9

df_distances['Assigned Cluster'] = [f'C{i+1}'

for i in assignments_Q4]

df_distances = df_distances[['Point', 'Dist to C1', 'Dist to C2', 'Dist to C3', 'Assigned Cluster']]

def highlight_min(s):is_min = s == s.min()

return ['background-color: yellow' if v else ''

for v in is_min] print("--- Distance Table (First Assignment) ---")

print(df_distances.to_string(index=False, float_format="%.2f")) assignments_Q4 = np.array(assignments_Q4)

new_centroids_Q4 = update_centroids(POINTS_9, assignments_Q4, K_VALUE) print("\n--- New Centroids (After First Iteration) ---")

print(f"New C1: {new_centroids_Q4[0]} (Avg of P1, P2, P7)") print(f"New C2: {new_centroids_Q4[1]} (Avg of P3, P5, P9)") print(f"New C3: {new_centroids_Q4[2]} (Avg of P4, P6, P8)")

plot_clusters(POINTS_9, assignments_Q4, new_centroids_Q4, 1, title_prefix="Q4 First Iteration")
```

Output:



LAB Assessment

Student Name		**LAB Rubrics**	CLO3 , P5, PLO5
		Total Marks	10
Registration No		**Obtained Marks**	
		Teacher Name	Dr. Syed M Hamedoon
Date		**Signature**	

Laboratory Work Assessment Rubrics

Sr. No.	Performance Indicator	Excellent (5)	Good (4)	Average (3)	Fair (2)	Poor (1)
1	**Theoretical knowledge 10%**	Student knows all the related concepts about the theoretical background of the experiment and rephrase those concepts in written and oral assessments	Student knows most of the related concepts about the theoretical background of the experiment and partially rephrase those concepts in written and oral assessments	Student knows few of the related concepts about the theoretical background of the experiment and partially rephrase those concepts in written and oral assessments	Student knows very little about the related concepts about the theoretical background of the experiment and poorly rephrase those concepts in written and oral assessments	Student has poor understanding of the related concepts about the theoretical background of the experiment and unable to rephrase those concepts in written and oral assessments
2	**Application Functionality 10%**	Application runs smoothly and operation of the application runs efficiently	Application compiles with no warnings. Robust operation of the application, with good recovery.	Application compiles with few or no warnings. Consideration given to unusual conditions with reasonable	Application compiles and runs without crashing. Some attempt at detecting and correcting errors.	Application does not compile or compiles but crashes. Confusing. Little or no error detection or correction.
3	**Specifications 10%**	The program works very efficiently and meets all of the required specifications.	The program works and meets some of the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly.	The program is producing incorrect results.
4	**Level of understanding of the learned skill 10%**	Provide complete and logical answers based upon accurate technical content to the questions asked by examiner	Provide complete and logical answers based upon accurate technical content to the questions asked by examiner with few errors	Provide partially correct and logical answers based upon minimum technical content to the questions asked by examiner	Provide very few and illogical answers to the questions asked by examiner.	Provide no answer to the questions asked by examiner.
5	**Readability and Reusability 10%**	The code is exceptionally well organized and very easy to follow and reused	The code is fairly easy to read. The code could be reused as a whole or each class could be reused.	Most of the code could be reused in other programs.	Some parts of the code require change before they could be reused in other programs.	The code is poorly organized and very difficult to read and not organized for reusability.

6	**AI System Design 10%**	Well-designed AI models. Code is highly maintainable	Good designed AI models and Little code duplications	Some attempt to make AI models. Code can be maintained with significant effort	Little attempt to design AI models and less understanding of code	Very poor attempt to design AI models and its code
7	**Responsiveness to Questions/ Accuracy 10%**	1. Responds well, quick and very accurate all the time. 2. Effectively uses eye contact, speaks clearly, effectively and confidently using suitable volume	1. Generally Responsive and accurate most of the times. 2. Maintains eye contact, speaks clearly with suitable volume and pace.	1. Generally Responsive and accurate few times. 2. Some eye contact, speaks clearly and unclearly in different portions.	1. Not much Responsive and accurate most of the times. 2. Uses eye contact ineffectively and fails to speak clearly and audibly	. 1. Non Responsive and inaccurate all the times. 2. No eye contact and unable to speak 3. Dresses inappropriately
8	**Efficiency** **10%**	The code is extremely efficient without sacrificing readability and understanding	The code is fairly efficient without sacrificing readability and understanding	Some part of the code is efficient and other part of the code is not understandable and work properly	The code is brute force and unnecessarily long	The code is huge and appears to be patched together
9	**Delivery** **10%**	The program was delivered in time during lab.	The program was delivered in Lab before the end time.	The program was delivered within the due date.	The code was delivered within a day after the due date.	The code was delivered more than 2 days overdue.
10	**Awareness of Safety Guidelines 10%**	Student has sufficient knowledge of the laboratory safety SOPs and protocol and is fully compliant to the guidelines	Student has sufficient knowledge of the laboratory safety SOPs and protocol and is Partially compliant to the guidelines	Student has little knowledge of the laboratory safety SOPs and protocol and is Partially compliant to the guidelines	Student has little knowledge of the laboratory safety SOPs and protocol and is non-compliant to the guidelines	Student has no knowledge of the laboratory safety SOPs and protocol and is non-compliant to the guidelines