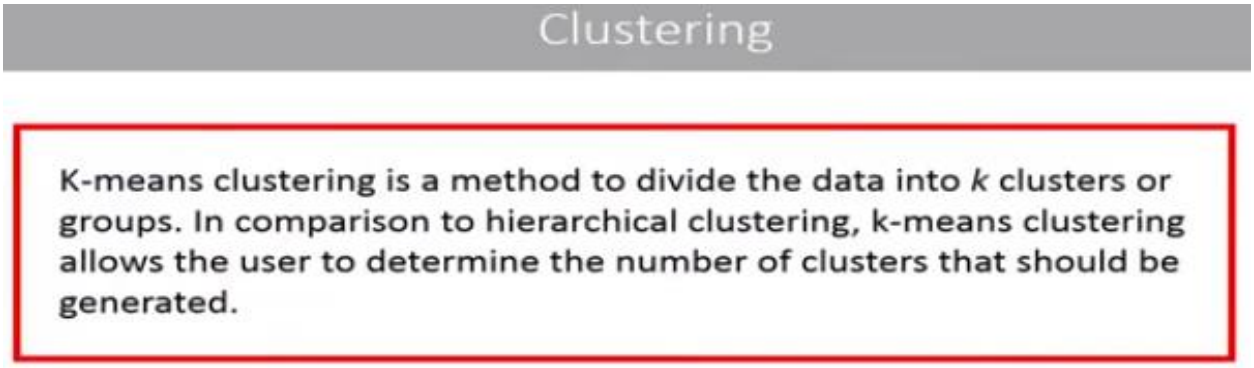
Lesson 9:K-means Clustering unsupervised learning algorithm in Data science.

Clustering is a set of techniques used to partition data into groups, or clusters. ***Clusters*** *are loosely defined as groups of data objects that are more similar to other objects in their cluster than they are to data objects in other clusters. In practice, clustering helps identify two qualities of data:*

- **Meaningful** clusters expand domain knowledge. For example, in the medical field, researchers applied clustering to gene expression experiments. The clustering results identified groups of patients who respond differently to medical treatments.
- **Useful** clusters, on the other hand, serve as an intermediate step in a data pipeline. For example, businesses use clustering for customer segmentation. The clustering results segment customers into groups with similar purchase histories, which businesses can then use to create targeted advertising campaigns.



K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

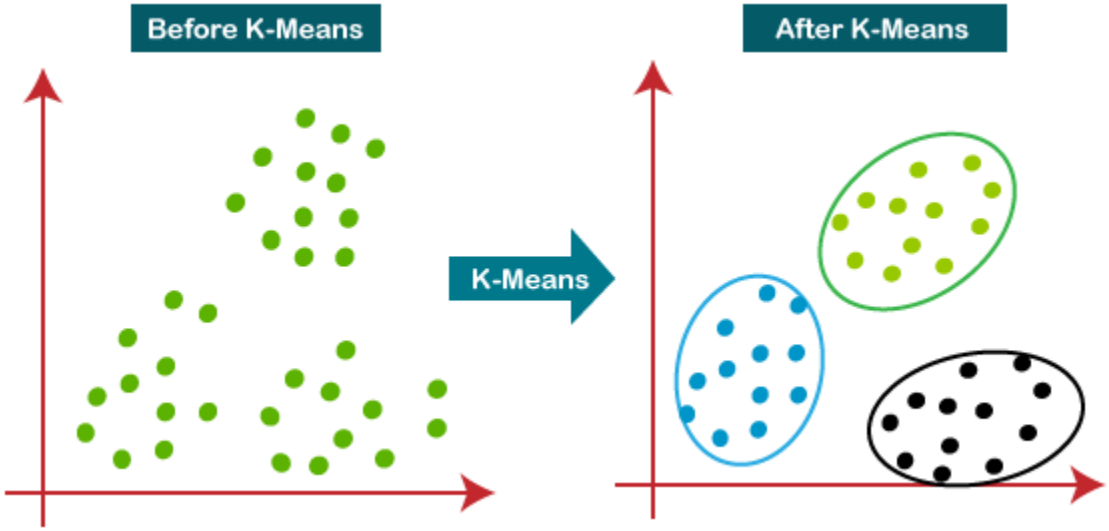
- It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
- It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.
- The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

1. Determines the best value for K center points or centroids by an iterative process.
2. Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



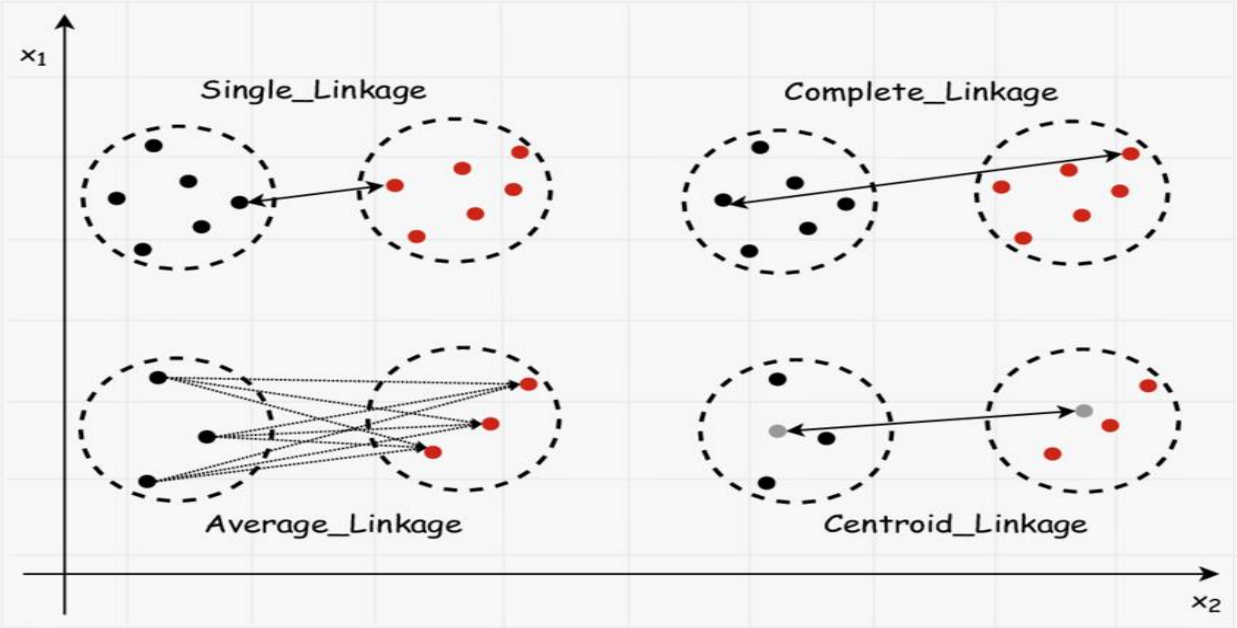
- we are interested in finding out commonalities among our data observations.
    - One way to determine that commonality or similarity is through a measure of distance among the data points. The shorter the distance, the more similar the observations are.

How does the K-Means Algorithm Work?

- ✓ **Step-1:** Select the number K to decide the number of clusters.
- ✓ **Step-2:** Select random K points or centroids. (It can be other from the input dataset).
- ✓ **Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.
- ✓ **Step-4:** Calculate the variance and place a new centroid of each cluster.
- ✓ **Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- ✓ **Step-6:** If any reassignment occurs, then go to step-4 else go to FINISH.
- ✓ **Step-7:** The model is ready.

Measuring similarities between Clusters:

There are different methods for measuring the similarity between clusters. These methods include single linkage, complete linkage, average linkage, and centroid linkage.



1. *Single linkage:* Single linkage, also known as the nearest neighbor method, is a method that focuses on the minimum distances between clusters. In other words, the similarity between two clusters is determined by the shortest distance between any two points in the two clusters. This method tends to produce elongated clusters as it joins the two closest data points at each step, regardless of whether they belong to the same cluster or not.
2. *Complete linkage:* Complete linkage, also known as the furthest neighbor method, is a method that focuses on the maximum distances between clusters. In other words, the similarity between two clusters is determined by the longest distance between any two points in the two clusters. This method tends to produce more compact clusters as it joins clusters based on their overall similarity, rather than just the closest pair of points.
3. *Average linkage:* Average linkage is a method that focuses on the average distance between clusters. In other words, the similarity between two clusters is determined by the average distance between all points in the two clusters. This method strikes a balance between single linkage and complete linkage, producing clusters that are neither too elongated nor too compact.
4. *Centroid linkage:* Centroid linkage is a method that focuses on the distance between the centroids of clusters. In other words, the similarity between two clusters is determined by the distance between the mean vectors of the two clusters. This method is similar to average linkage, but it is more sensitive to outliers as it uses the mean of the data points instead of the average distance between all pairs of points.

The choice of method for measuring the similarity between clusters depends on the specific application. For example;

- ✓ Single linkage is a good choice for identifying clusters of outliers.
- ✓ Complete linkage is a good choice for identifying clusters of tightly packed points.
- ✓ Average linkage is a good choice for a general-purpose clustering algorithm.
- ✓ Centroid linkage is a good choice for applications where the mean of the data points is a meaningful representation of the cluster.

Here is a table summarizing the key differences between the four methods:

Method	*Focus*	*Similarity metric*	*Advantages*	*Disadvantages*
Single linkage	Minimum distance	Shortest distance between any two points in the two clusters	Easy to implement	Sensitive to outliers, can produce elongated clusters
Complete linkage	Maximum distance	Longest distance between any two points in the two clusters	Robust to outliers, can produce compact clusters	Can produce clusters that are too tightly packed
Average linkage	Average distance	Average distance between all points in the two clusters	Produces a balance between single linkage and complete linkage	Can be sensitive to outliers
Centroid linkage	Distance between centroids	Distance between the mean vectors of the two clusters	Insensitive to outliers	Can produce clusters that are not well-separated

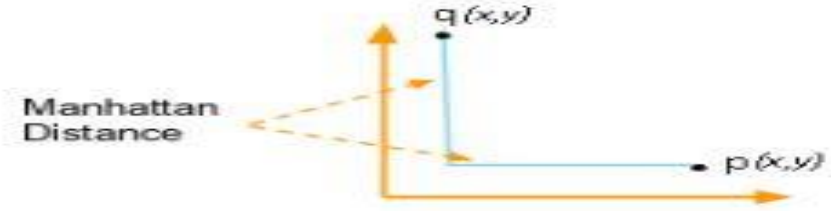
K-Means clustering supports various kinds of distance measures, such as:

- Manhattan distance measure (*Also called Rectilinear distance*).
- Euclidean distance measure.

1. ***Manhattan distance measure (Also called Rectilinear distance).***

The Manhattan distance is the simple sum of the horizontal and vertical components or the distance between two points measured along axes at right angles.
Note that we are taking the absolute value so that the negative values don't come into play.
The formula is shown below:

$$d = \sum_{i=1}^{n} | q_x - p_x | + | q_x - p_y |$$



Calculating Manhattan distance (Also called Rectilinear distance).

K-means Clustering

- **Example**

Cluster the following eight points (with (x, y) representing locations) into three clusters A1(2, 10) A2(2, 5) A3(8, 4) A4(5, 8) A5(7, 5) A6(6, 4) A7(1, 2) A8(4, 9).

K-means Clustering

- **Example**

Cluster the following eight points (with (x, y) representing locations) into three clusters A1(2, 10) A2(2, 5) A3(8, 4) A4(5, 8) A5(7, 5) A6(6, 4) A7(1, 2) A8(4, 9).

Initial cluster centers are: A1(2, 10), A4(5, 8) and A7(1, 2).

The distance function between two points $a=(x1, y1)$ and $b=(x2, y2)$ is defined as: $\rho(a, b) = |x2 - x1| + |y2 - y1|$.

- Use k-means algorithm to find the three cluster centers after the second iteration.

Solution

Iteration 1

		(2, 10)	(5, 8)	(1, 2)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	Cluster
A1	(2, 10)				
A2	(2, 5)				
A3	(8, 4)				
A4	(5, 8)				
A5	(7, 5)				
A6	(6, 4)				
A7	(1, 2)				
A8	(4, 9)				

Example Continued

point	mean1
$x1, y1$	$x2, y2$
(2, 10)	(2, 10)

$$\rho(a, b) = |x2 - x1| + |y2 - y1|$$

$$\rho(point, mean1) = |x2 - x1| + |y2 - y1|$$
$$= |2 - 2| + |10 - 10|$$
$$= 0 + 0$$
$$= 0$$

point	mean2
$x1, y1$	$x2, y2$
(2, 10)	(5, 8)

$$\rho(a, b) = |x2 - x1| + |y2 - y1|$$

$$\rho(point, mean2) = |x2 - x1| + |y2 - y1|$$
$$= |5 - 2| + |8 - 10|$$
$$= 3 + 2$$
$$= 5$$

Example Continued

Iteration 1

		(2, 10)	(5, 8)	(1, 2)	
	Point	**Dist Mean 1**	**Dist Mean 2**	**Dist Mean 3**	**Cluster**
A1	(2, 10)	0	5	9	1
A2	(2, 5)	5	6	4	3
A3	(8, 4)	12	7	9	2
A4	(5, 8)	5	0	10	2
A5	(7, 5)	10	5	9	2
A6	(6, 4)	10	5	7	2
A7	(1, 2)	9	10	0	3
A8	(4, 9)	3	2	10	2

Cluster 1
(2, 10)

Cluster 2
(8, 4)
(5, 8)
(7, 5)
(6, 4)
(4, 9)

Cluster 3
(2, 5)
(1, 2)

Example Continued

- Next, we need to re-compute the new cluster centers (means). We do so, by taking the mean of all points in each cluster.
- For Cluster 1, we only have one point A1(2, 10), which was the old mean, so the cluster center remains the same.
- For Cluster 2, we have ((8+5+7+6+4)/5, (4+8+5+4+9)/5) = (6, 6)
- For Cluster 3, we have ((2+1)/2, (5+2)/2) = (1.5, 3.5)

Example Continued

		(2, 10)	(6, 6)	(1.5, 3.5)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	**Cluster**
A1	(2, 10)	0	8	7	1
A2	(2, 5)	5	5	2	3
A3	(8, 4)	12	4	7	2
A4	(5, 8)	5	3	8	2
A5	(7, 5)	10	2	7	2
A6	(6, 4)	10	2	5	2
A7	(1, 2)	9	9	2	3
A8	(4, 9)	3	5	8	1

Example Continued

- Next, we need to re-compute the new cluster centers (means). We do so, by taking the mean of all points in each cluster.

- In Cluster 1, we have points 1 and 8. Therefore the centroid is: $((2+4)/2,(10+9)/2)=(3,9.5)$
- In Cluster 2, we have points 3,4,5 and 6. Therefore, the centroid is: $((8+5+7+6)/4,(4+8+5+4)/4)=(6.5,5.25)$
- For Cluster 3, we have points 2 and 7. Therefore, the centroid is: $((2+1)/2,(5+2)/2)=(1.5,3.5)$

Example Continued

		(3, 9.5)	(6.5 ,5.25)	(1.5, 3.5)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	**Cluster**
A1	(2, 10)	1.5	9.25	7	**1**
A2	(2, 5)	5.5	4.75	2	**3**
A3	(8, 4)	10.5	2.75	7	**2**
A4	(5, 8)	3.5	4.25	8	**1**
A5	(7, 5)	8.5	0.75	7	**2**
A6	(6, 4)	8.5	1.75	5	**2**
A7	(1, 2)	9.5	8.75	2	**3**
A8	(4, 9)	1.5	6.25	8	**1**

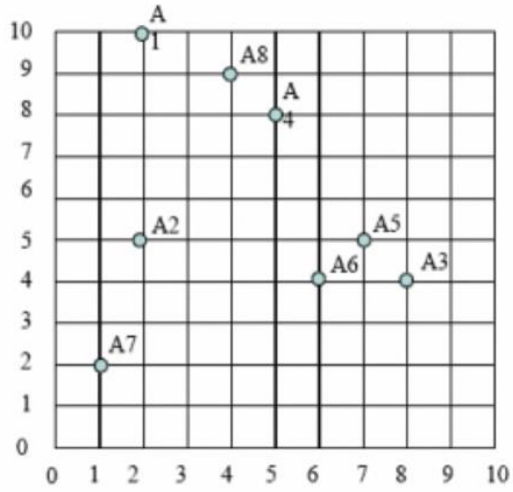
Example Continued

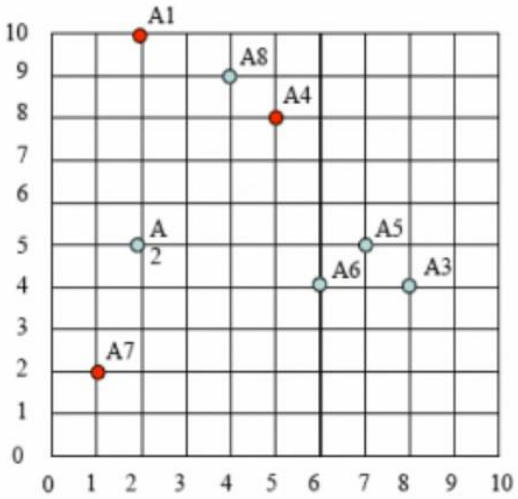
- Next, we need to re-compute the new cluster centers (means). We do so, by taking the mean of all points in each cluster.

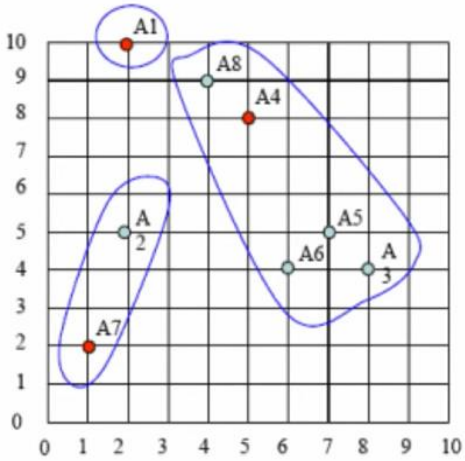
- In Cluster 1, we have points 1, 4, and 8. Therefore the centroid is: $((2+5+4)/2,(10+8+9)/2)=(3.67,9)$
- In Cluster 2, we have points 3,5 and 6. Therefore, the centroid is: $((8+7+6)/4,(4+5+4)/4)=(7,4.3)$
- For Cluster 3, we have points 2 and 7. Therefore, the centroid is: $((2+1)/2,(5+2)/2)=(1.5,3.5)$

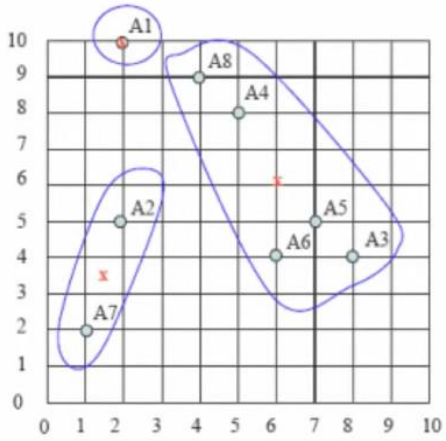
Example Continued

		(3.67, 9)	(7 ,4.3)	(1.5, 3.5)	
	Point	Dist Mean 1	Dist Mean 2	Dist Mean 3	**Cluster**
A1	(2, 10)	2.67	10.7	7	**1**
A2	(2, 5)	5.67	5.7	2	**3**
A3	(8, 4)	9.33	1.3	7	**2**
A4	(5, 8)	2.33	5.7	8	**1**
A5	(7, 5)	7.33	0.7	7	**2**
A6	(6, 4)	7.33	1.3	5	**2**
A7	(1, 2)	9.67	8.3	2	**3**
A8	(4, 9)	0.33	7.7	8	**1**

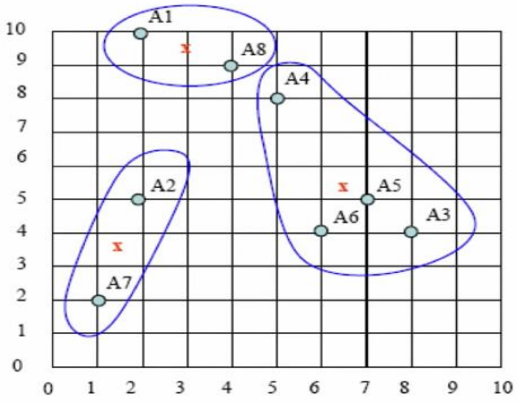


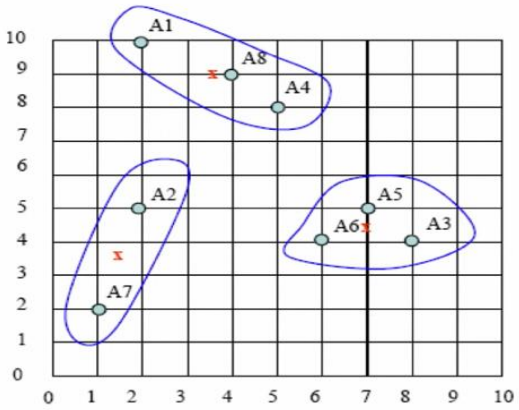




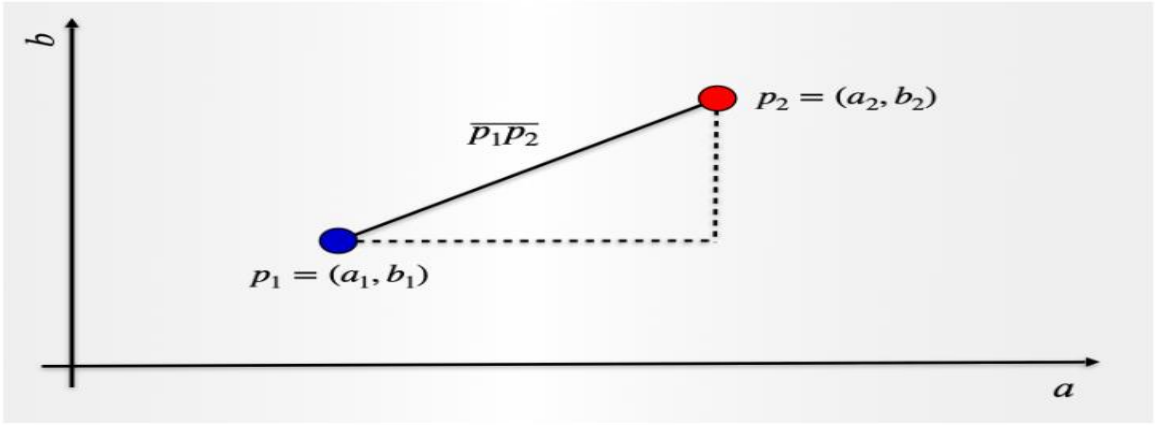


Next two Iterations





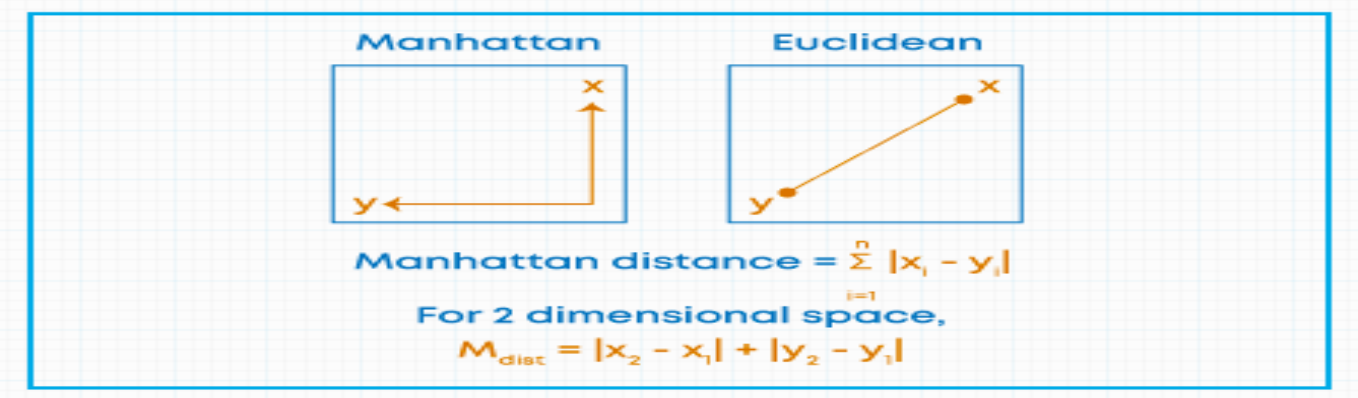
2. *Euclidean distance measure.*

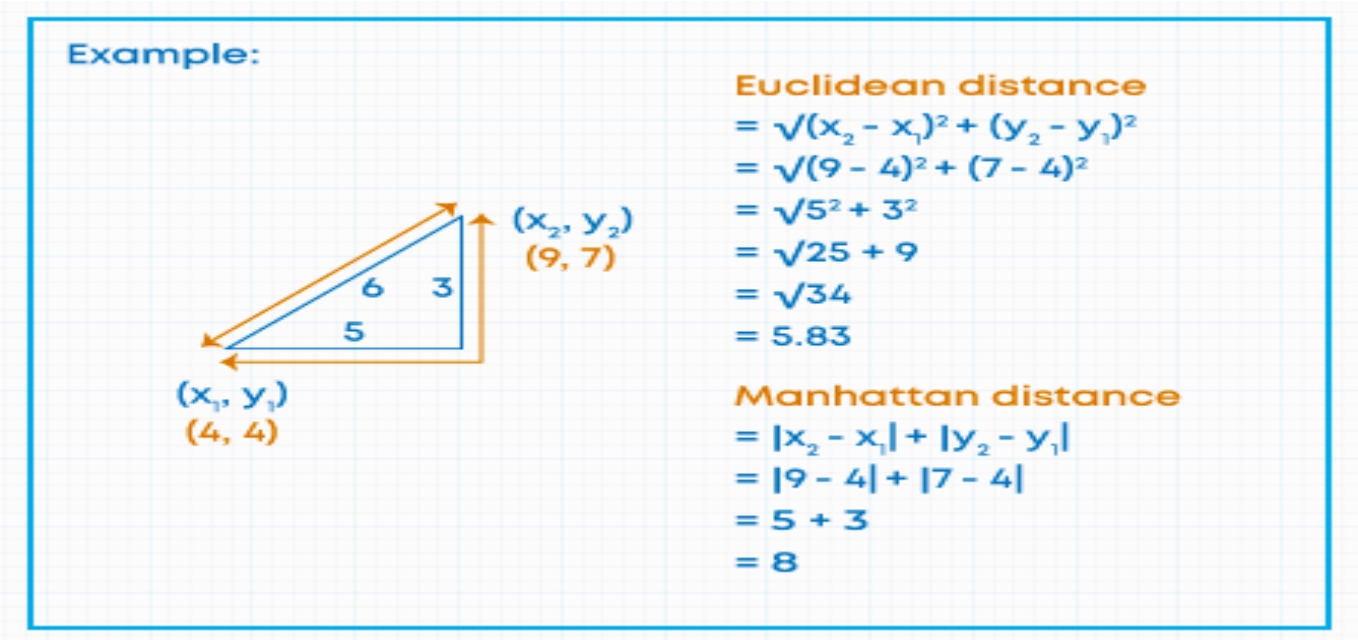


The distance $\overline{p_1p_2}$ is given by:

$$\overline{p_1p_2} = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$$

Compare the two here:





Calculating Euclidian Distance:

K-Means Clustering – Solved Example

- Suppose that the data mining task is to cluster points into three clusters,
- where the points are
- *A1*(2, 10), *A2*(2, 5), *A3*(8, 4), *B1*(5, 8), *B2*(7, 5), *B3*(6, 4), *C1*(1, 2), *C2*(4, 9).
- The distance function is Euclidean distance.
- Suppose initially we assign *A1*, *B1*, and *C1* as the center of each cluster, respectively.

K-Means Clustering – Solved Example

Initial Centroids:
A1: (2, 10)
B1: (5, 8)
C1: (1, 2)

Data Points			Distance to						Cluster	New Cluster
			2	10	5	8	1	2		
A1	2	10								
A2	2	5								
A3	8	4								
B1	5	8								
B2	7	5								
B3	6	4								
C1	1	2								
C2	4	9								

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- To calculate the distance of the new cluster we have to use the Euclidean Distance
- Euclidean distance is the shortest path between source and destination which is a straight line.
    - such as finding the nearest hospital for an emergency helicopter flight

K-Means Clustering – Solved Example

Initial Centroids:
A1: (2, 10)
B1: (5, 8)
C1: (1, 2)

Data Points			Distance to						Cluster	New Cluster
			2	10	5	8	1	2		
A1	2	10	0							
A2	2	5	5							
A3	8	4								
B1	5	8								
B2	7	5								
B3	6	4								
C1	1	2								
C2	4	9								

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

K-Means Clustering – Solved Example

Initial Centroids:
A1: (2, 10)
B1: (5, 8)
C1: (1, 2)

Data Points			Distance to						Cluster	New Cluster
			2	10	5	8	1	2		
A1	2	10	0.00		3.61		8.06			
A2	2	5	5.00		4.24		3.16			
A3	8	4	8.49		5.00		7.28			
B1	5	8	3.61		0.00		7.21			
B2	7	5	7.07		3.61		6.71			
B3	6	4	7.21		4.12		5.39			
C1	1	2	8.06		7.21		0.00			
C2	4	9	2.24		1.41		7.62			

Once the centroids calculations are through. this is how the distance points should look Fig Above.

K-Means Clustering – Solved Example

Initial Centroids:
A1: (2, 10)
B1: (5, 8)
C1: (1, 2)

Data Points			Distance to						Cluster	New Cluster
			2	10	5	8	1	2		
A1	2	10	0.00		3.61		8.06		1	
A2	2	5	5.00		4.24		3.16		3	
A3	8	4	8.49		5.00		7.28		2	
B1	5	8	3.61		0.00		7.21		2	
B2	7	5	7.07		3.61		6.71		2	
B3	6	4	7.21		4.12		5.39		2	
C1	1	2	8.06		7.21		0.00		3	
C2	4	9	2.24		1.41		7.62		2	

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Next step: Iterate by calculating the new centroid.

K-Means Clustering – Solved Example

Current Centroids:
A1: (2, 10)
B1: (6, 6)
C1: (1.5, 3.5)

	Data Points		Distance to 2, 10	Distance to 6, 6	Distance to 1.5, 3.5	Cluster	New Cluster
A1	2	10	0.00	5.66	6.52	1	
A2	2	5	5.00	4.12	1.58	3	
A3	8	4	8.49	2.83	6.52	2	
B1	5	8	3.61	2.24	5.70	2	
B2	7	5	7.07	1.41	5.70	2	
B3	6	4	7.21	2.00	4.53	2	
C1	1	2	8.06	6.40	1.58	3	
C2	4	9	2.24	3.61	6.04	2	

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

K-Means Clustering – Solved Example

Current Centroids:
A1: (2, 10)
B1: (6, 6)
C1: (1.5, 3.5)

	Data Points		Distance to 2, 10	Distance to 6, 6	Distance to 1.5, 3.5	Cluster	New Cluster
A1	2	10	0.00	5.66	6.52	1	1
A2	2	5	5.00	4.12	1.58	3	3
A3	8	4	8.49	2.83	6.52	2	2
B1	5	8	3.61	2.24	5.70	2	2
B2	7	5	7.07	1.41	5.70	2	2
B3	6	4	7.21	2.00	4.53	2	2
C1	1	2	8.06	6.40	1.58	3	3
C2	4	9	2.24	3.61	6.04	2 →	1

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

We need to iterate again until we get the original and the new cluster points same grouping.

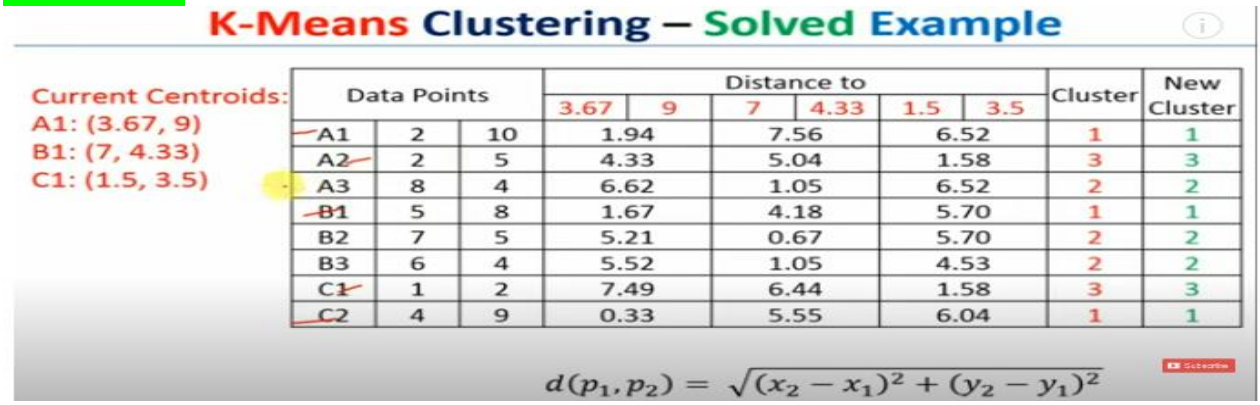
K-Means Clustering – Solved Example

Current Centroids:
A1: (3, 9.5)
B1: (6.5, 5.25)
C1: (1.5, 3.5)

	Data Points		Distance to 3, 9.5	Distance to 6.5, 5.25	Distance to 1.5, 3.5	Cluster	New Cluster
A1	2	10	1.12	6.54	6.52	1	1
A2	2	5	4.61	4.51	1.58	3	3
A3	8	4	7.43	1.95	6.52	2	2
B1	5	8	2.50	3.13	5.70	2	1
B2	7	5	6.02	0.56	5.70	2	2
B3	6	4	6.26	1.35	4.53	2	2
C1	1	2	7.76	6.39	1.58	3	3
C2	4	9	1.12	4.51	6.04	1	1

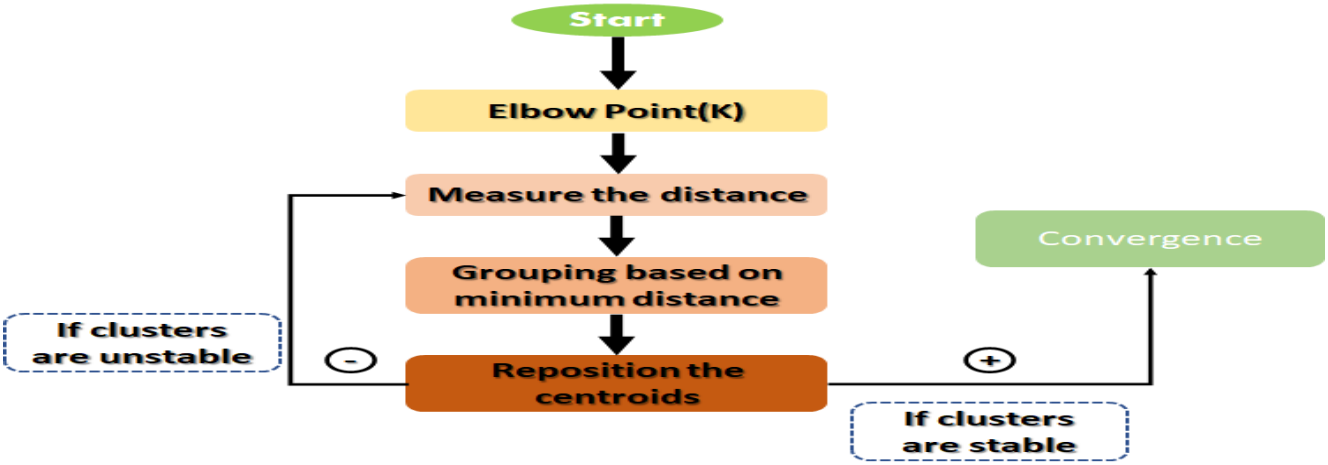
$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Final centroid



How K Means Clustering works.

K Means Clustering works by dividing a dataset into a predetermined number of clusters, or groups, based on the similarity of the data points within each cluster. The algorithm begins by randomly selecting a number of data points as the initial cluster centroids. It then assigns each data point to the closest cluster centroid, based on the distance between the data point and the centroid.



Elbow Method.

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$WCSS = \sum_{Pi \text{ in Cluster1}} \text{distance } (P_i\ C_1)^2 + \sum_{Pi \text{ in Cluster2}} \text{distance } (P_i\ C_2)^2 + \sum_{Pi \text{ in CLuster3}} \text{distance } (P_i\ C_3)^2$

- ✓ *WCSS:* Within-Cluster Sum of Squares.
- ✓ *Cluster1, Cluster2, Cluster3:* The clusters you are considering.
- ✓ *Pi:* Data points within each cluster.
- ✓ *distance (Pi, Ci) ^2:* The squared Euclidean distance between data point Pi and the centroid Ci of its cluster.

In the above formula of WCSS,

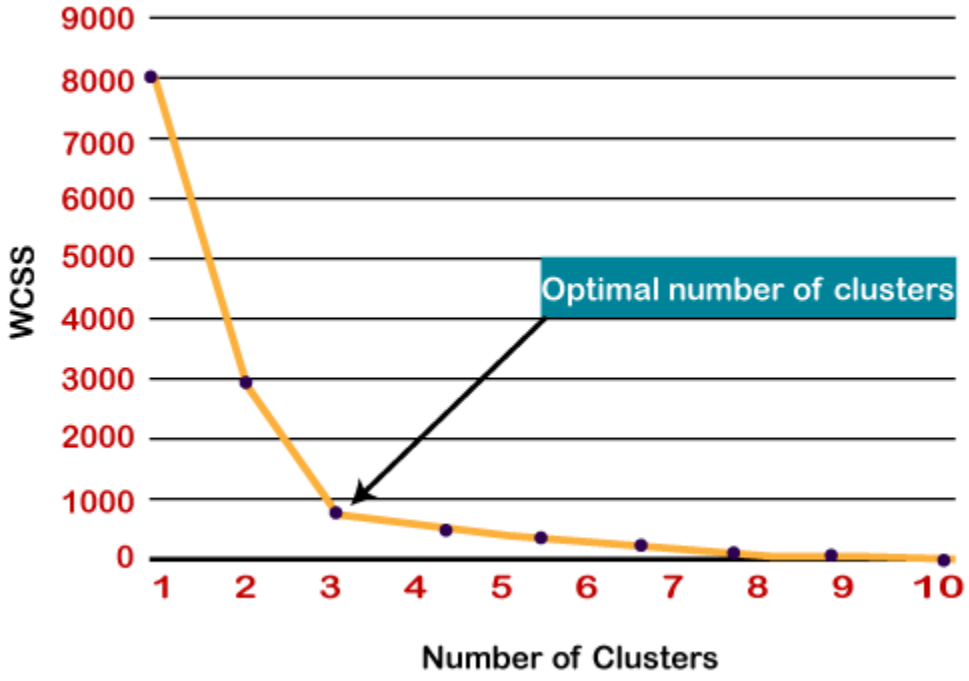
$\sum_{Pi \text{ in Cluster1}}$ distance$(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



IMPLEMENTATION IN GOOGLE COLAB:

EXAMPLE 1: KMeans Clustering with Python and Scikit-learn.

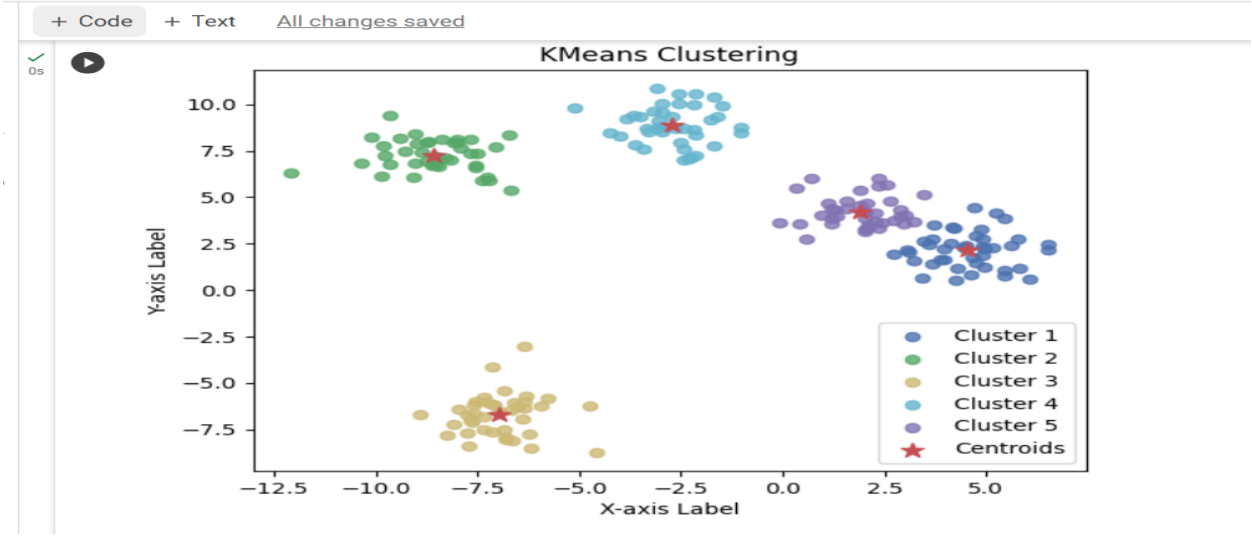
- Create dummy data for clustering
- Train and cluster data using KMeans
- Plot the clustered data
- Pick the best value for K using the Elbow method.

SOURCE CODE:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Create a dataset of 200 samples and 5 clusters
features, labels = make_blobs(
   n_samples=200,
   centers=5,
   random_state=42  # Add a random state for reproducibility
)
# Instantiate the model with 5 'K' clusters and 10 iterations with different centroid seed
model = KMeans(
   n_clusters=5,
   n_init=10,
   random_state=42
)
# Train the model
model.fit(features)
# Make a prediction on the data
p_labels = model.predict(features)
plt.style.use('default')
# Define a list of colors for the clusters
cluster_colors = ['b', 'g', 'y', 'c', 'm']
# Iterate through the clusters and plot them with different colors and labels
for cluster_num in range(5):
   cluster_data = features[p_labels == cluster_num]
   plt.scatter(cluster_data[:, 0], cluster_data[:, 1], alpha=0.8, label=f'Cluster {cluster_num
+ 1}', c=cluster_colors[cluster_num])
cluster_centers = model.cluster_centers_
cs_x = cluster_centers[:, 0]
cs_y = cluster_centers[:, 1]
# Plot centroids with a different color (e.g., yellow)
plt.scatter(cs_x, cs_y, marker='*', s=100, c='r', label='Centroids')
plt.title('KMeans Clustering')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.legend()
plt.show()
```

Line(s)	*Code*	*Explanation*
1	import matplotlib.pyplot as plt	Import the pyplot module from the matplotlib library as plt to create plots and visualizations.
2	from sklearn.cluster import KMeans	Import the KMeans class from the sklearn.cluster module to perform K-Means clustering.
3	from sklearn.datasets import make_blobs	Import the make_blobs function from sklearn.datasets to create a synthetic dataset with blobs for clustering.
5	features, labels = make_blobs(	Use the make_blobs function to create a synthetic dataset with 200 samples and 5 clusters. n_samples specifies the number of samples, and centers specifies the number of clusters. A random state (random_state) is set for reproducibility. The data is stored in the features variable, and the labels for each data point are stored in the labels variable.
9	model = KMeans(	Instantiate a K-Means clustering model with 5 clusters (n_clusters=5), 10 iterations with different centroid seeds (n_init=10), and a random state for reproducibility. The model is stored in the model variable.
13	model.fit(features)	Train the K-Means model on the features data using the fit method. This step clusters the data into the specified number of clusters.
16	p_labels = model.predict(features)	Make predictions on the data to assign each data point to one of the clusters using the predict method. The cluster labels for each data point are stored in the p_labels variable.
17	plt.style.use('default')	Set the plot style to the default style using plt.style.use('default').
19-23	Define a list of colors (cluster_colors)	Define a list of colors in the variable cluster_colors. This list will be used to assign different colors to each cluster.
25-32	Iterate through the clusters and plot them	Iterate through each cluster (from 1 to 5) using a for loop. For each cluster, select the data points that belong to that cluster and scatter plot them with a distinct color from cluster_colors. The alpha parameter controls the transparency of the points, and the label specifies the label for the cluster.
34-38	Plot the centroids with a different color	Plot the centroids of the clusters with a different color (e.g., yellow) using the plt.scatter function. The marker parameter specifies that the centroids are plotted as stars ('*'), the s parameter controls the size of the stars, and the c parameter sets the color.
39-42	Set the plot title and axis labels	Set the title of the plot using plt.title(), and label the X and Y axes using plt.xlabel() and plt.ylabel().
44	plt.legend()	Add a legend to the plot using plt.legend() to label the clusters and centroids.
45	plt.show()	Display the plot using plt.show().

OUTPUT:



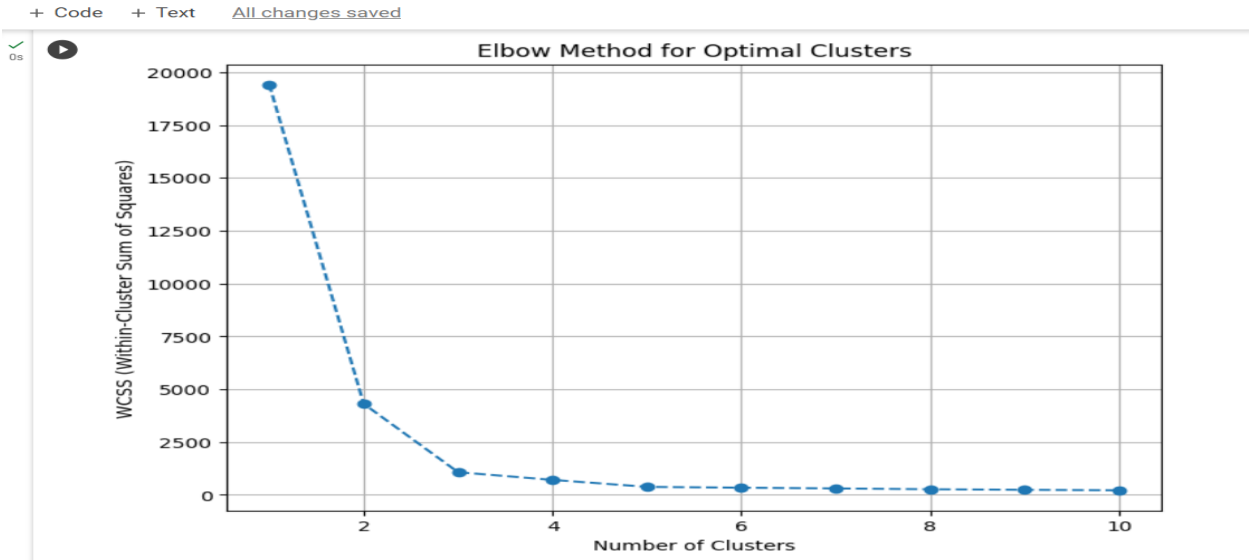
To generate an elbow method graph to determine the optimal number of clusters, you calculate the within-cluster sum of squares (WCSS) for a range of cluster numbers and then plot it. The "elbow" point in the graph represents the optimal cluster number.

SOURCE CODE:

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
# Create a dataset of 200 samples and 5 clusters
features, labels = make_blobs(
    n_samples=200,
    centers=5
)
# Create an empty list to store WCSS (within-cluster sum of squares) values
wcss = []
# Try different numbers of clusters from 1 to 10
for i in range(1, 11):
    model = KMeans(n_clusters=i, n_init=10, random_state=42)
    model.fit(features)
    wcss.append(model.inertia_)
# Plot the elbow method graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.grid()
plt.show()
```

Line(s)	*Code*	*Explanation*
1	import matplotlib.pyplot as plt	Import the pyplot module from the matplotlib library as plt to create plots and visualizations.
2	from sklearn.cluster import KMeans	Import the KMeans class from the sklearn.cluster module to perform K-Means clustering.
3	from sklearn.datasets import make_blobs	Import the make_blobs function from sklearn.datasets to create a synthetic dataset with blobs for clustering.
5	features, labels = make_blobs(	Create a synthetic dataset with 200 samples and 5 clusters using the make_blobs function. The n_samples parameter specifies the number of samples, and the centers parameter specifies the number of clusters.
7-11	Create an empty list to store WCSS values	Initialize an empty list wcss (within-cluster sum of squares) to store the WCSS values for different numbers of clusters.
13-18	Try different numbers of clusters	Iterate through different numbers of clusters from 1 to 10 using a for loop. For each iteration, create a K-Means model with the specified number of clusters, set the number of initializations to 10, and provide a random state for reproducibility.
19-21	Train the K-Means model	Train the K-Means model on the features data using the fit method. This step clusters the data into the specified number of clusters.
22-23	Calculate and store the WCSS	Calculate and store the within-cluster sum of squares (WCSS) using the model.inertia_ attribute, and append it to the wcss list. WCSS is a measure of the compactness of the clusters.
25-31	Plot the elbow method graph	Create a figure with a specific size using plt.figure(). Plot the WCSS values on the Y-axis and the number of clusters on the X-axis. Customize the plot with a marker, linestyle, title, and axis labels. Finally, display the plot using plt.show().

OUTPUT:



EXAMPLE 2: A Python program to perform K-Means clustering on a dataset containing information about Mall Customers.
The goal is to make you to understand the customers like who can be easily converge Target Customers so that the sense can be given to marketing team and plan the strategy accordingly.
https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python

Step 1: Load the data set and check basic information (EDA).

SOURCE CODE:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
wainaina = pd.read_csv(r'/content/Mall_Customers.csv')
print(wainaina) # or use wainaina.head(20)
wainaina.info()
wainaina.duplicated().sum()
wainaina.describe()
print('shape=', wainaina.shape)
print('Number of Duplicated values=',wainaina.duplicated().sum())
wainaina.describe()
```

Line(s)	*Code*	*Explanation*
1	import numpy as np	Import the NumPy library with the alias np. NumPy is commonly used for numerical and array operations in Python.
2	import pandas as pd	Import the Pandas library with the alias pd. Pandas is used for data manipulation and analysis, particularly with DataFrames and Series.
3	wainaina = pd.read_csv (r'/content/Mall_Customers.csv')	Read a CSV file located at the path /content/Mall_Customers.csv into a Pandas DataFrame named wainaina. This line reads the data from the CSV file and stores it in a structured tabular format.
4	print(wainaina)	Display the entire DataFrame wainaina to the console. This provides a visual representation of the data in the DataFrame.
5	wainaina.info()	Display an overview of the DataFrame's information, including the data types of each column, the number of non-null values, and memory usage. It's a useful method to get a summary of the DataFrame's structure.
6	wainaina.duplicated().sum()	Check for duplicated rows in the DataFrame using the duplicated() method. The sum() method is then used to count the number of duplicated rows. This line provides the count of duplicated rows in the DataFrame.

7	print('shape=', wainaina.shape)	Print the shape of the DataFrame using the shape attribute. This line displays the number of rows and columns in the DataFrame.
8	print('Number of Duplicated values=', wainaina.duplicated().sum())	Display the count of duplicated values again, but this time as part of a printed string. This line provides the count of duplicated rows in a human-readable format.
9	wainaina.describe()	Generate summary statistics of the DataFrame, including count, mean, standard deviation, minimum, and maximum values for each numeric column. It provides an overview of the data distribution.

OUTPUT:

```
+ Code  + Text   All changes saved

     CustomerID  Gender  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
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
shape= (200, 5)
Number of Duplicated values= 0
```

```
+ Code  + Text   All changes saved
```

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

Step 2: Generate an elbow graph to determine the optimal number of clusters:

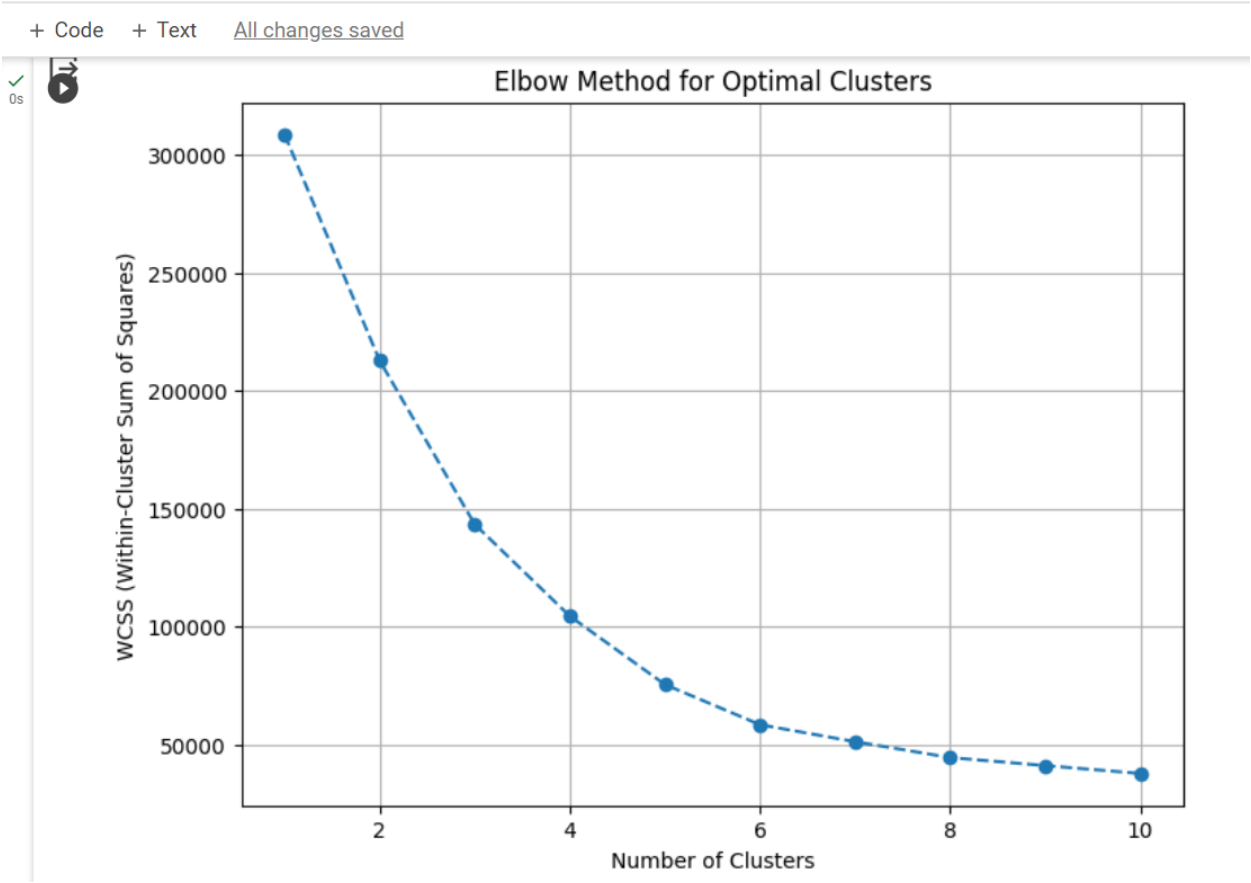
SOURCE CODE:

```
import pandas as pd
from sklearn.cluster import KMeans
wainaina = pd.read_csv(r'/content/Mall_Customers.csv')
# Select the numerical columns for clustering (Age, Annual Income, Spending Score)
X = wainaina.iloc[:, [2, 3, 4]].values
# Create an empty list to store WCSS (within-cluster sum of squares) values
wcss = []
# Try different numbers of clusters from 1 to 10
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
# Plot the elbow graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.grid()
plt.show()
```

Line(s)	***Code***	***Explanation***
1	import pandas as pd	Import the Pandas library with the alias pd. Pandas is used for data manipulation and analysis, especially with DataFrames and Series.
2	from sklearn.cluster import KMeans	Import the KMeans class from the sklearn.cluster module to perform K-Means clustering.
3	wainaina = pd.read_csv (r'/content/Mall_Customers.csv')	Read a CSV file located at the path /content/Mall_Customers.csv into a Pandas DataFrame named wainaina. This line reads the data from the CSV file and stores it in a structured tabular format.
4	X = wainaina.iloc[:, [2, 3, 4]].values	Select the numerical columns for clustering (Age, Annual Income, and Spending Score) from the DataFrame wainaina. This line extracts the values of these columns and stores them in a NumPy array X.
5	wcss = []	Create an empty list named wcss to store the within-cluster sum of squares (WCSS) values for different numbers of clusters.
6-11	Try different numbers of clusters	Use a for loop to iterate through different numbers of clusters from 1 to 10. For each iteration, create a K-Means model with the specified number of clusters, set the initialization method to 'k-means++', and perform a maximum of 300 iterations with 10 initializations. The random_state parameter is set to 0 for reproducibility.

12-13	Calculate and store the WCSS	Fit the K-Means model to the X data and calculate the WCSS using the kmeans.inertia_ attribute. Append the calculated WCSS to the wcss list for the current number of clusters.
14-19	Plot the elbow graph	Create a figure with a specific size using plt.figure(). Plot the WCSS values on the Y-axis and the number of clusters on the X-axis. Customize the plot with markers, a dashed line, title, and axis labels. Finally, display the plot using plt.show().

OUTPUT:



Step 3: Visualize the clusters.

SOURCE CODE:

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans
wainaina = pd.read_csv(r'/content/Mall_Customers.csv')
# Specify the optimal number of clusters
optimal_clusters = 5
# User-defined attributes to visualize (you can modify this)
attributes_to_visualize = ['Annual Income (k$)', 'Spending Score (1-100)']
```

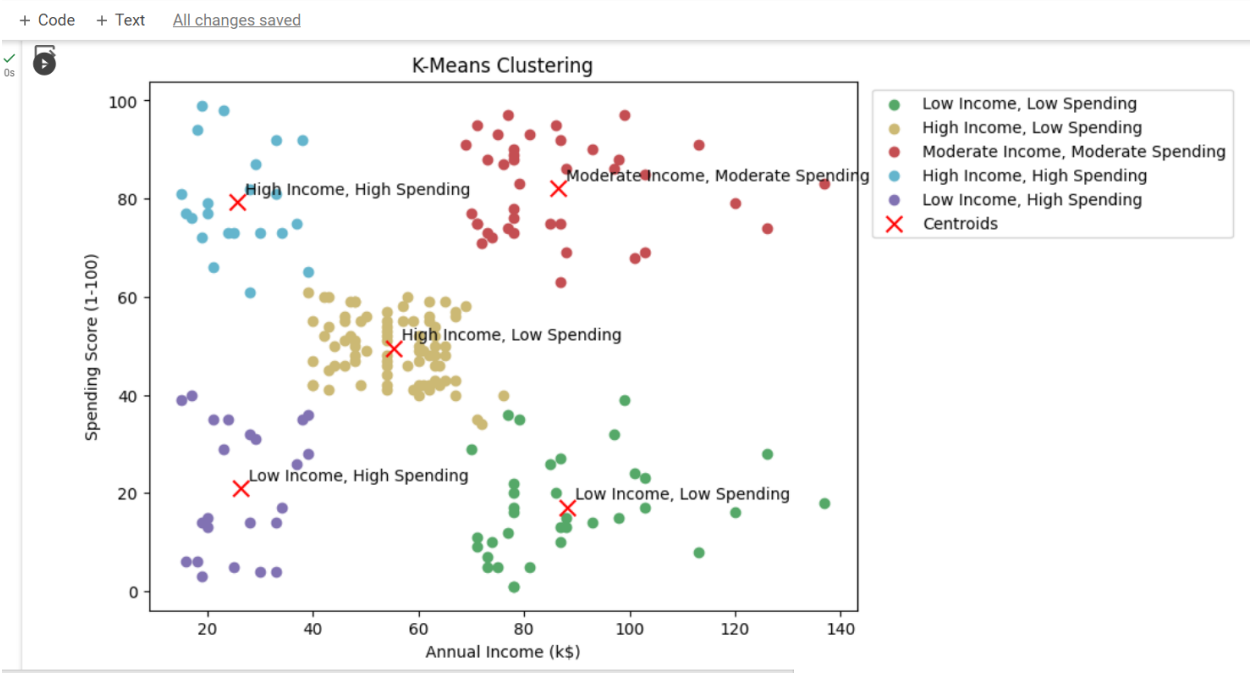
```
# Select the data for visualization based on user-defined attributes
X = wainaina[attributes_to_visualize].values
# Perform K-Means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(X)
# Add the cluster labels to the dataset
wainaina['Cluster'] = y_kmeans
# Specify colors for each cluster (modify this list with your preferred colors)
cluster_colors = ['g', 'y', 'r', 'c', 'm']
# Specify labels for each cluster (modify this list with your preferred labels)
cluster_labels = ['Low Income, Low Spending', 'High Income, Low Spending', 'Moderate Income, Moderate Spending', 'High Income, High Spending', 'Low Income, High Spending']
# Visualize the clusters with specified colors and X symbol for centroids
plt.figure(figsize=(8, 6))
for cluster_num in range(optimal_clusters):
    cluster_data = X[y_kmeans == cluster_num]
    plt.scatter(cluster_data[:, 0], cluster_data[:, 1], label=cluster_labels[cluster_num], c=cluster_colors[cluster_num])
# Annotate the cluster with the label near its centroid
    centroid = kmeans.cluster_centers_[cluster_num]
    plt.annotate(cluster_labels[cluster_num], centroid, xytext=(5, 5), textcoords='offset points', fontsize=10)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='x', s=100, c='red', label='Centroids')
# Move the legend outside
plt.legend(loc='upper left', bbox_to_anchor=(1.01, 1))
plt.title('K-Means Clustering')
plt.xlabel(attributes_to_visualize[0])  # Set X-axis label
plt.ylabel(attributes_to_visualize[1])  # Set Y-axis label
plt.show()
```

Line	***Code***	***Explanation***
1	import matplotlib.pyplot as plt	Imports the 'pyplot' module from the 'matplotlib' library for data visualization.
2	import pandas as pd	Imports the 'pandas' library for data manipulation and analysis.
3	wainaina = pd.read_csv(r'/content/Mall_Customers.csv')	Reads a CSV file ('Mall_Customers.csv') into a Pandas DataFrame named 'wainaina'.
6	optimal_clusters = 5	Defines the optimal number of clusters to create in the K-Means clustering analysis (5 in this case).
9	attributes_to_visualize = ['Annual Income (k$)', 'Spending Score (1-100)']	Specifies the attributes to be used for visualization and clustering (annual income and spending score).
12	X = wainaina[attributes_to_visualize].values	Extracts the data from the 'wainaina' DataFrame, containing the specified attributes, as a NumPy array stored in the variable 'X'.

15	kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300, n_init=10, random_state=0)	Creates a K-Means clustering model with the specified parameters: 'n_clusters' is the number of clusters to create (5 in this case), 'init' is the method for initializing centroids, 'max_iter' is the maximum number of iterations, 'n_init' is the number of times the algorithm is run with different initial centroids, and 'random_state' is the seed for random number generation.
16	y_kmeans = kmeans.fit_predict(X)	Applies the K-Means clustering algorithm to the data in 'X' and assigns each data point to a cluster. The cluster labels are stored in 'y_kmeans'.
19	wainaina['Cluster'] = y_kmeans	Adds a new column 'Cluster' to the 'wainaina' DataFrame, containing the cluster labels assigned by the K-Means algorithm.
22	cluster_colors = ['g', 'y', 'r', 'c', 'm']	Defines a list of colors to be used for visualizing different clusters (in this case, five different colors).
25	cluster_labels = ['Low Income, Low Spending', 'High Income, Low Spending', 'Moderate Income, Moderate Spending', 'High Income, High Spending', 'Low Income, High Spending']	Specifies descriptive labels for each cluster.
28	plt.figure(figsize=(8, 6))	Creates a new Matplotlib figure for plotting the clusters with a specified figure size.
29	for cluster_num in range(optimal_clusters):	Initiates a loop to iterate through each cluster for visualization.
30	cluster_data = X[y_kmeans == cluster_num]	Filters the data points in 'X' that belong to the current cluster being visualized and stores them in 'cluster_data'.
31	plt.scatter(cluster_data[:, 0], cluster_data[:, 1], label=cluster_labels[cluster_num], c=cluster_colors[cluster_num])	Scatters the data points on the plot, applying the specified color and label corresponding to the cluster.
34	centroid = kmeans.cluster_centers_[cluster_num]	Retrieves the coordinates of the centroid for the current cluster.
35	plt.annotate(cluster_labels[cluster_num], centroid, xytext=(5, 5), textcoords='offset points', fontsize=10)	Annotates the plot with the label of the cluster, positioned near its centroid.
38	plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], marker='x', s=100, c='red', label='Centroids')	Adds 'X' symbols for the centroids of each cluster in red color with a 'Centroids' label.
41	plt.legend(loc='upper left', bbox_to_anchor=(1.01, 1))	Positions the legend outside the plot on the upper left side.
42	plt.title('K-Means Clustering')	Sets the title of the plot to 'K-Means Clustering'.
43	plt.xlabel(attributes_to_visualize[0])	Labels the X-axis with the first attribute to visualize ('Annual Income (k$)').

44	plt.ylabel(attributes_to_visualize[1])	Labels the Y-axis with the second attribute to visualize ('Spending Score (1-100)').
45	plt.show()	Displays the plot with the K-Means clustering visualization.

OUTPUT:



Step 4: 3D Visualization using the same data set in K-Clustering.

SOURCE CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
import seaborn as sns
# Load the dataset
wainaina = pd.read_csv('/content/Mall_Customers.csv')
# Specify the optimal number of clusters
optimal_clusters = 5
# User-defined attributes to visualize in 3D (you can modify this)
attributes_to_visualize = ['Annual Income (k$)', 'Spending Score (1-100)', 'Age']
# Select the data for visualization based on user-defined attributes
X = wainaina[attributes_to_visualize].values
# Perform K-Means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300,
n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(X)
```

```
# Add the cluster labels to the dataset
wainaina['Cluster'] = y_kmeans
# Specify custom colors for each cluster
cluster_colors = ['b', 'g', 'y', 'c', 'k']
# Use Seaborn for color palette
palette = sns.color_palette(cluster_colors)
# Create a 3D scatter plot using Matplotlib and Seaborn for coloring
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
# Add the following two lines to set the azimuth and elevation angles
ax.view_init(elev=20, azim=70)  # Adjust the angles as needed
# Specify cluster labels
cluster_labels = ['Low Income, Low Spending', 'High Income, Low Spending', 'Moderate Income, Moderate Spending', 'High Income, High Spending', 'Low Income, High Spending']
for cluster_num in range(optimal_clusters):
    cluster_data = X[y_kmeans == cluster_num]
    scatter = ax.scatter(cluster_data[:, 0], cluster_data[:, 1], cluster_data[:, 2], c=palette[cluster_num], s=100, alpha=0.6, label=cluster_labels[cluster_num])
# Add centroids with red "X" symbols
centroids = kmeans.cluster_centers_
ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], c='red', marker='*', s=300, label='Centroids')
# Set labels, title, and color bar
ax.set_xlabel(attributes_to_visualize[0])
ax.set_ylabel(attributes_to_visualize[1])
ax.set_zlabel(attributes_to_visualize[2])
ax.set_title('K-Means Clustering in 3D')
# Move the legend outside
ax.legend(loc='center left', bbox_to_anchor=(1.1, 0.5))
# Show the plot
plt.show()
```

Line Number	***Code***	***Explanation***
1	import pandas as pd	Imports the Pandas library as 'pd' to work with data.
2	import matplotlib.pyplot as plt	Imports the Matplotlib library for creating plots as 'plt'.
3	from mpl_toolkits.mplot3d import Axes3D	Imports the 'Axes3D' toolkit from Matplotlib for creating 3D plots.
4	from sklearn.cluster import KMeans	Imports the KMeans clustering algorithm from the Scikit-Learn library.
5	import seaborn as sns	Imports the Seaborn library for enhanced data visualization.

8-9	wainaina = pd.read_csv('/content/Mall_Customers.csv')	Reads a CSV file ('Mall_Customers.csv') using Pandas and stores it in a DataFrame named 'wainaina'.
12	optimal_clusters = 5	Defines the number of clusters to be created in K-Means clustering.
15	attributes_to_visualize = ['Annual Income (k$)', 'Spending Score (1-100)', 'Age']	Defines the attributes from the dataset to visualize in 3D. These attributes will be used to create the 3D plot.
18	X = wainaina[attributes_to_visualize].values	Selects the data from the DataFrame based on the specified attributes and stores it in the 'X' variable as a NumPy array.
21-25	kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300, n_init=10, random_state=0)	Configures the K-Means clustering algorithm with the desired parameters and initializes it as 'kmeans'.
26	y_kmeans = kmeans.fit_predict(X)	Applies K-Means clustering to the data and assigns cluster labels to the data points, storing the result in 'y_kmeans'.
29	wainaina['Cluster'] = y_kmeans	Adds a new column 'Cluster' to the 'wainaina' DataFrame to store the cluster labels assigned to each data point.
32	cluster_colors = ['b', 'g', 'y', 'c', 'k']	Defines a list of custom colors to be used for each cluster in the 3D plot.
35	palette = sns.color_palette(cluster_colors)	Creates a color palette using Seaborn, which maps the custom colors to the clusters for coloring the data points.
38-39	fig = plt.figure(figsize=(8, 6))	Creates a new figure for the 3D plot with a specific size (8x6 inches) and stores it in 'fig'.
40	ax = fig.add_subplot(111, projection='3d')	Adds a 3D subplot (axes) to the figure and sets it as 'ax' for creating a 3D plot.
43	ax.view_init(elev=20, azim=70)	Adjusts the viewing angle (elevation and azimuth) of the 3D plot.
47	cluster_labels = ['Low Income, Low Spending', 'High Income, Low Spending', ...]	Defines labels for each cluster, which will be used in the legend of the 3D plot.
48-50	for cluster_num in range(optimal_clusters):	Iterates through each cluster for visualization.
51	cluster_data = X[y_kmeans == cluster_num]	Selects the data points that belong to the current cluster and stores them in 'cluster_data'.
52-55	scatter = ax.scatter(cluster_data[:, 0], cluster_data[:, 1], cluster_data[:, 2], ...	Creates a scatter plot for the current cluster in the 3D space. It uses the custom color from the palette and assigns a label for the legend.
58-61	centroids = kmeans.cluster_centers_	Retrieves the cluster centroids from the K-Means model and stores them in the 'centroids' variable.
62	ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], ...	Adds red "X" symbols at the positions of the cluster centroids in the 3D plot.
65-67	ax.set_xlabel(attributes_to_visualize[0]), ...	Sets labels for the x, y, and z axes in the 3D plot.

68	ax.set_title('K-Means Clustering in 3D')	Sets the title of the 3D plot.
71-72	ax.legend(loc='center left', bbox_to_anchor=(1.1, 0.5))	Places the legend outside the plot at the specified location.
75	plt.show()	Displays the 3D plot.

OUTPUT: