Open in Colab

Introduction to Clustering

Clustering is an *unsupervised machine learning* technique used to group similar data points together based on their features. Unlike classification, clustering doesn't rely on predefined labels—instead, it discovers inherent patterns or groupings in the data.

Each group formed is called a **cluster**, and the aim is to ensure that:

- Data points **within** the same cluster are more similar to each other.
- Data points **between** clusters are as different as possible.

Common Clustering Algorithms

- **K-Means**
- **Hierarchical Clustering**
- **DBSCAN**
- **Gaussian Mixture Models (GMM)**

Clustering Business Cases

#	Use Case	Goal	Example Dataset	Example / Description
1	Customer Segmentation	Group customers based on behavior to target marketing more effectively.	E-commerce data with features like purchase frequency, total spend, product categories, etc.	Retail company uses K-Means clustering to segment customers into: **High-spenders**, **Occasional buyers**, **Deal seekers** for tailored promotions and offers.
2	Anomaly Detection	Detect unusual behavior, such as fraud or outlier transactions.	Banking transaction data including amount, time, location, and device used.	If most customer behaviors fall into well-defined clusters, outliers can be flagged for investigation (e.g., fraud checks).
3	Product or Inventory Segmentation	Identify groups of products with similar sales patterns, profit margins, or lifecycles.	SKU-level sales, seasonality, return rates.	Retailer clusters products into: **Fast-moving items**, **Seasonal products**, **High-margin items**, **Slow-moving/dead-stock** to

#	Use Case	Goal	Example Dataset	Example / Description
				guide stocking, pricing, and liquidation.
4	Churn Prediction Support (Behavioral Profiles)	Group users based on engagement or usage patterns to identify segments likely to churn.	Login frequency, feature usage, subscription renewal behavior.	SaaS company clusters customers into: **Highly active**, **Moderately active**, **At-risk (low activity)** to target retention campaigns.
5	Supply Chain Optimization	Cluster suppliers, shipments, or routes for efficiency.	Delivery times, cost, reliability, shipment destinations.	Logistics firms cluster delivery destinations to optimize routes, reduce fuel costs, and consolidate shipments.
6	Image or Sensor Data Segmentation (Manufacturing)	Group patterns in quality-control images or sensor readings.	Machine vibration data, product inspection images.	Factory clusters sensor signals into: **Normal operation**, **Early-stage wear**, **Malfunction** to support predictive maintenance and reduce downtime.
7	Healthcare Patient Segmentation	Group patients for targeted care paths, personalized medicine, or resource planning.	Symptoms, medical history, lab results, comorbidities.	Hospitals cluster patients into: **High-risk chronic disease**, **Low-risk routine care**, **Emergency-prone** groups to improve care and resource allocation.
8	Financial Portfolio Segmentation	Group financial assets based on volatility, returns, and risk profiles.	Historical prices, risk measures, correlations.	Wealth management firms cluster stocks to build diversified portfolios or offer customized investment products.
9	Energy Usage Profiling	Group customers or equipment based on electricity consumption patterns.	Smart meter hourly usage, peak load times.	Utility companies cluster consumers into: **High daytime usage**, **Evening peak users**, **Low, steady users** for demand forecasting and dynamic pricing.
10	Workforce or HR Segmentation	Cluster employees to understand performance, career paths, or training needs.	Years of experience, performance scores, skills, education.	Companies cluster employees into: **High performers ready for leadership**, **Undertrained but high potential**, **At-risk employees** to plan development programs.
12	Pricing Segmentation	Group customers or products to identify	Purchase history, market demand,	Company clusters customers based on price sensitivity into **Premium**,

#	Use Case	Goal	Example Dataset	Example / Description
		differentiated pricing tiers.	customer willingness-to-pay.	**Standard**, and **Budget** segments to optimize revenue.

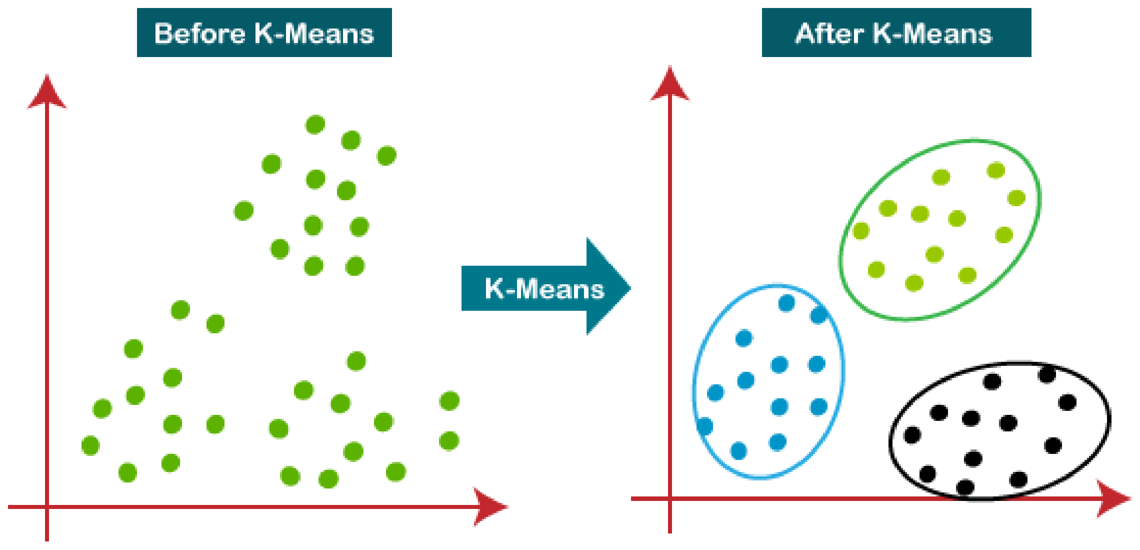
More Examples

Introduction to KMeans Clustering

K-Means Clustering is an unsupervised learning algorithm used to group similar data points into a predefined number of clusters, K. The objective is to partition the dataset into K clusters, where each data point belongs to the cluster with the nearest mean, minimizing the variance within each cluster.

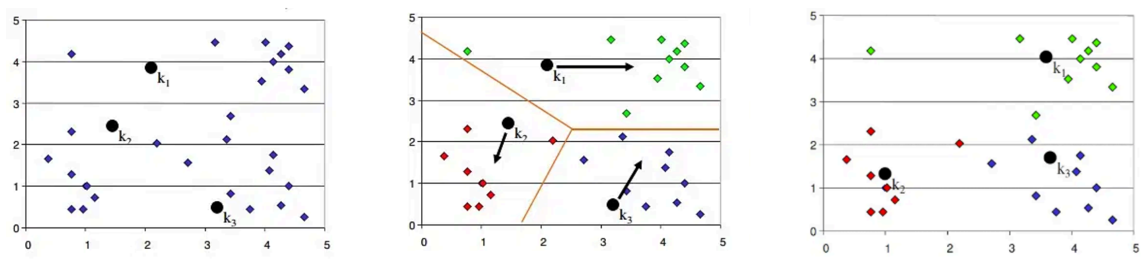
How does K-Means Work

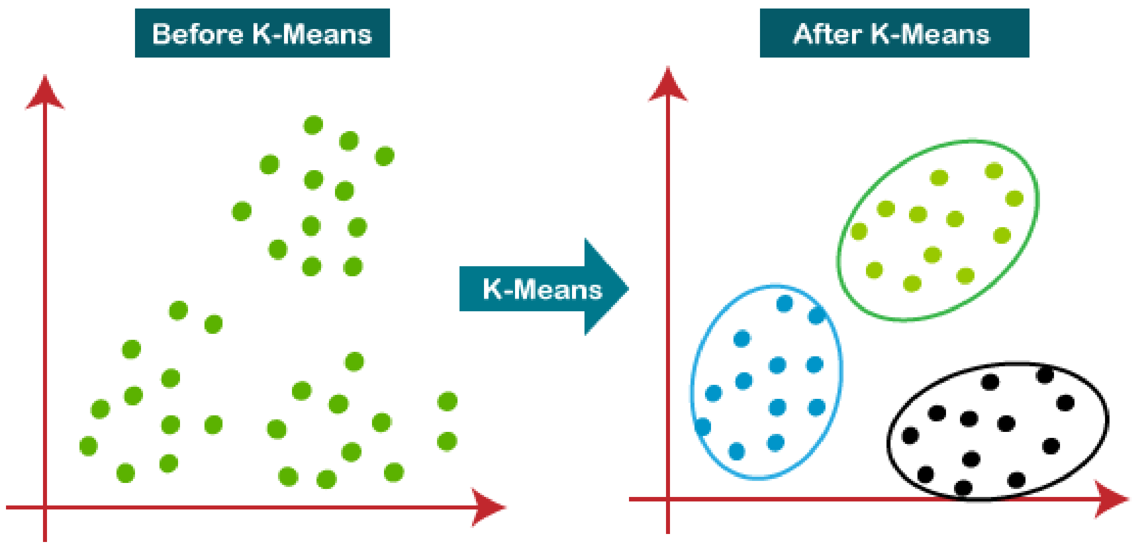
The K-Means algorithm works iteratively to partition a dataset into K clusters.

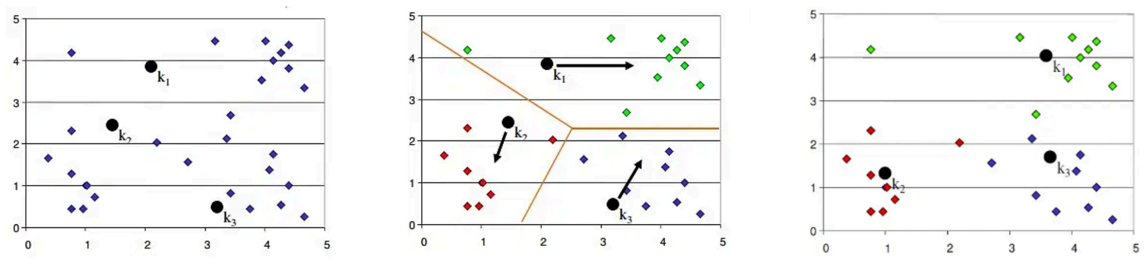


Here's how the algorithm proceeds:

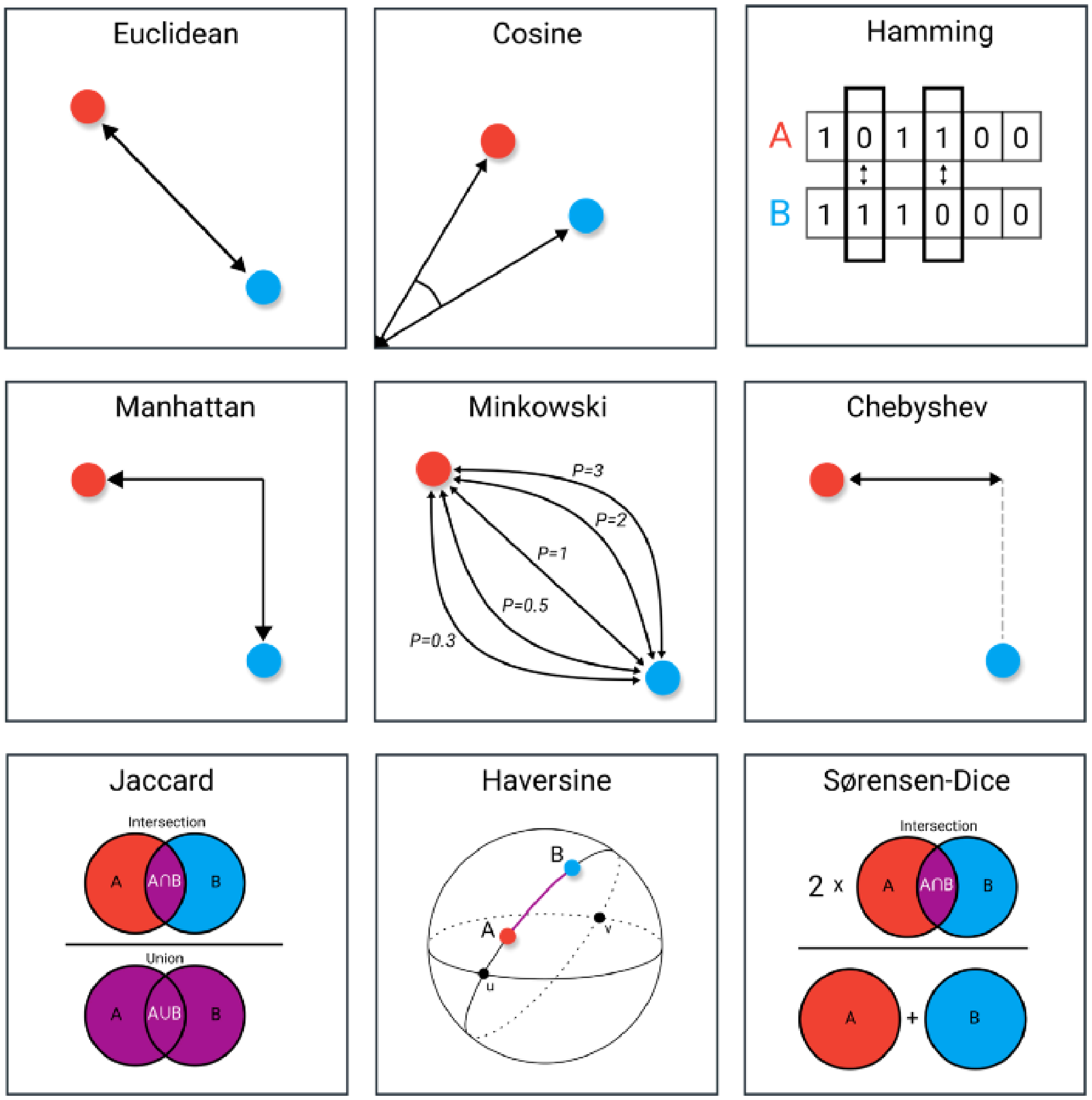
- **Step 1**: Define the number K to decide the number of clusters.
- **Step 2**: Initialize K centroids **randomly**.
- **Step 3**: Assign each data point to the nearest centroid, forming K clusters.
- **Step 4**: Recalculate the centroids as the mean of all data points assigned to each cluster.
- **Step 5**: Repeat steps 2 and 3 until the centroids no longer change, or the algorithm converges.







Common Distance Measures



4.1. Euclidean Distance

- Definition: The straight-line distance between two points in Euclidean space.
- Formula:

$$\text{Euclidean Distance} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Here, (x_i) and (y_i) are the (i)-th components (features) of the two points (x) and (y); the term ((x_i - y_i)^2) is the **squared difference** for that component, which (a) makes all contributions non-negative and (b) penalizes larger differences more strongly than smaller ones.
- For example, if (x = (x_1, x_2, x_3, x_4)) and (y = (y_1, y_2, y_3, y_4)), then [\sum_{i=1}^{4} (x_i - y_i)^2 = (x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + (x_4 - y_4)^2.]

- Use Case: Commonly used in clustering (e.g., K-means) and nearest neighbor algorithms.

4.2. Cosine Similarity/Distance

- Definition: Measures the cosine of the angle between two vectors. It evaluates orientation rather than magnitude.
- Formula:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}||\mathbf{B}|}$$

- Use Case: Used in text similarity and high-dimensional spaces (e.g., document similarity).

4.3. Hamming Distance

- Definition: Counts the number of positions at which corresponding symbols differ between two strings of equal length.
- Formula: Count mismatches between (A) and (B).
- Use Case: Applied in binary and categorical data, like error detection in coding.

4.4. Manhattan Distance

- Definition: The sum of absolute differences between the coordinates of two points.
- Formula:

$$\text{Manhattan Distance} = \sum_{i=1}^{n} |x_i - y_i|$$

- In this formula, (x_i) and (y_i) are again the (i)-th components of the points, and (|x_i - y_i|) is the **absolute difference** for that component, which treats all deviations linearly and ignores direction (only the magnitude of the difference matters).
- For example, if (x = (x_1, x_2, x_3, x_4)) and (y = (y_1, y_2, y_3, y_4)), then [\sum_{i=1}^{4} |x_i - y_i| = |x_1 - y_1| + |x_2 - y_2| + |x_3 - y_3| + |x_4 - y_4|.]
- Use Case: Used when movement is restricted to horizontal and vertical paths (e.g., grid-based games or city blocks).

4.5. Minkowski Distance

- Definition: Generalized distance metric, including Euclidean ((p=2)) and Manhattan ((p=1)) distances as special cases.
- Formula:

$$\text{Minkowski Distance} = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

- Use Case: Offers flexibility to adjust distance metrics based on parameter (p).

4.6. Chebyshev Distance

- Definition: The maximum absolute difference between two coordinates.
- Formula:

$$\text{Chebyshev Distance} = \max_i(|x_i - y_i|)$$

- Use Case: Used in chess for king's moves and grid-based pathfinding.

4.7. Jaccard Index/Distance

- Definition: Measures similarity or dissimilarity between two sets.
- Formula:

$$\text{Jaccard Index} = \frac{|A \cap B|}{|A \cup B|}$$

$$\text{Jaccard Distance} = 1 - \text{Jaccard Index}$$

- Use Case: Applied in text analysis, image segmentation, and set comparison.

4.8. Haversine Distance

- Definition: Calculates the shortest distance between two points on a sphere, considering curvature (e.g., Earth).
- Formula:

$$d = 2r \arcsin \sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}$$

- Use Case: Used in geographic applications like GPS.

4.9. Sørensen-Dice Coefficient

- Definition: Measures the similarity between two sets.
- Formula:

$$\text{Sørensen-Dice Coefficient} = \frac{2|A \cap B|}{|A| + |B|}$$

- Use Case: Common in biology and text analysis.

K-Means Clustering Step by Step Example

We will use the Euclidean distance formula to compute the distance between each point and the centroids and assign each point to the closest cluster.
The Euclidean distance measures the straight-line distance between two points in a 2D space. The formula is:

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

- **Data Points**: (x, y)
  - 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 Centroids**:
  - Mean 1: (2, 10)
  - Mean 2: (5, 8)
  - Mean 3: (1, 2)

Iteration 1

1. **Compute Distances**: Use the Euclidean distance formula to calculate the distance from each point to all centroids.
2. **Assign Clusters**: Each point is assigned to the cluster of the nearest centroid.

Distances and cluster assignments are shown in the table below.

Point	Coordinates	Dist Mean 1 ($\sqrt{(2-x)^2 + (10-y)^2}$)	Dist Mean 2 ($\sqrt{(5-x)^2 + (8-y)^2}$)	Dist Mean 3 ($\sqrt{(1-x)^2 + (2-y)^2}$)	Cluster
A1	(2, 10)	$\sqrt{(2-2)^2 + (10-10)^2}$ = 0.00	$\sqrt{(5-2)^2 + (8-10)^2}$ = 3.61	$\sqrt{(1-2)^2 + (2-10)^2}$ = 8.25	1
A2	(2, 5)	$\sqrt{(2-2)^2 + (10-5)^2}$ = 5.00	$\sqrt{(5-2)^2 + (8-5)^2}$ = 3.61	$\sqrt{(1-2)^2 + (2-5)^2}$ = 3.16	3
A3	(8, 4)	$\sqrt{(2-8)^2 + (10-4)^2}$ = 9.43	$\sqrt{(5-8)^2 + (8-4)^2}$ = 5.00	$\sqrt{(1-8)^2 + (2-4)^2}$ = 7.28	2
A4	(5, 8)	$\sqrt{(2-5)^2 + (10-8)^2}$ = 3.61	$\sqrt{(5-5)^2 + (8-8)^2}$ = 0.00	$\sqrt{(1-5)^2 + (2-8)^2}$ = 6.71	2
A5	(7, 5)	$\sqrt{(2-7)^2 + (10-5)^2}$ = 8.06	$\sqrt{(5-7)^2 + (8-5)^2}$ = 3.61	$\sqrt{(1-7)^2 + (2-5)^2}$ = 6.08	2
A6	(6, 4)	$\sqrt{(2-6)^2 + (10-4)^2}$ = 8.94	$\sqrt{(5-6)^2 + (8-4)^2}$ = 4.47	$\sqrt{(1-6)^2 + (2-4)^2}$ = 5.39	2
A7	(1, 2)	$\sqrt{(2-1)^2 + (10-2)^2}$ = 8.00	$\sqrt{(5-1)^2 + (8-2)^2}$ = 7.62	$\sqrt{(1-1)^2 + (2-2)^2}$ = 1.00	3

Point	Coordinates	Dist Mean 1 (√((2−x)² + (10−y)²))	Dist Mean 2 (√((5−x)² + (8−y)²))	Dist Mean 3 (√((1−x)² + (2−y)²))	Cluster
A8	(4, 9)	√((2−4)² + (10−9)²) = 2.24	√((5−4)² + (8−9)²) = 1.41	√((1−4)² + (2−9)²) = 7.07	2

Iteration 2

1. **Recalculate Centroids**: For each cluster, compute the new centroid.

The new centroid is calculated as the mean of the (x)- and (y)-coordinates of all points in the cluster:

$$\text{Centroid}_i = \left(\frac{\sum x_{\text{cluster}}}{\text{n}}, \frac{\sum y_{\text{cluster}}}{\text{n}}\right)$$

New Centroids: $(2.0, 10.0)$, $(6.0, 6.0)$, $(1.5, 3.5)$

2. **Reassign Clusters**: Recompute distances to the updated centroids and assign points to the nearest cluster.

Point	Coordinates	Dist Mean 1 (√((x1−x)² + (y1−y)²))	Dist Mean 2 (√((x2−x)² + (y2−y)²))	Dist Mean 3 (√((x3−x)² + (y3−y)²))	Cluster
A1	(2, 10)	√((2−2)² + (10−10)²) = 0.00	√((6−2)² + (6−10)²) = 5.39	√((1.5−2)² + (3.5−10)²) = 8.83	1
A2	(2, 5)	√((2−2)² + (10−5)²) = 5.00	√((6−2)² + (6−5)²) = 4.47	√((1.5−2)² + (3.5−5)²) = 2.50	3
A3	(8, 4)	√((2−8)² + (10−4)²) = 9.22	√((6−8)² + (6−4)²) = 3.61	√((1.5−8)² + (3.5−4)²) = 6.86	2
A4	(5, 8)	√((2−5)² + (10−8)²) = 3.61	√((6−5)² + (6−8)²) = 2.24	√((1.5−5)² + (3.5−8)²) = 6.92	2
A5	(7, 5)	√((2−7)² + (10−5)²) = 8.06	√((6−7)² + (6−5)²) = 2.24	√((1.5−7)² + (3.5−5)²) = 5.92	2
A6	(6, 4)	√((2−6)² + (10−4)²) = 8.94	√((6−6)² + (6−4)²) = 2.83	√((1.5−6)² + (3.5−4)²) = 5.22	2
A7	(1, 2)	√((2−1)² + (10−2)²) = 8.06	√((6−1)² + (6−2)²) = 7.21	√((1.5−1)² + (3.5−2)²) = 1.80	3
A8	(4, 9)	√((2−4)² + (10−9)²) = 2.24	√((6−4)² + (6−9)²) = 3.61	√((1.5−4)² + (3.5−9)²) = 6.80	1

Comments:

- Notice how centroids shift toward the center of their clusters.
- Some points might change clusters as centroids update.

Iteration 3

1. **Recompute Centroids**: Repeat the process of recalculating centroids.

New Centroids: $(3.0, 9.5)$, $(6.5, 5.25)$, $(1.5, 3.5)$

2. **Reassign Clusters**: Update cluster assignments based on new centroids.

Point	Coordinates	Dist Mean 1	Dist Mean 2	Dist Mean 3	Cluster
A1	(2, 10)	1.12	5.94	8.83	1
A2	(2, 5)	4.53	4.18	2.50	3
A3	(8, 4)	8.86	3.00	6.86	2
A4	(5, 8)	2.55	2.28	6.92	2
A5	(7, 5)	7.90	2.05	5.92	2
A6	(6, 4)	8.55	2.65	5.22	2
A7	(1, 2)	7.55	7.07	1.80	3
A8	(4, 9)	1.80	3.74	6.80	1

Comments:

- Observe how centroids continue to adjust.
- Points begin to stabilize in their respective clusters.

Iteration 4

1. **Recalculate Centroids**: Adjust centroids as the mean of their cluster points.
2. **Check for Convergence**: If centroids do not change, the algorithm stops.

Results:

- Distances and final cluster assignments are shown in the table below.
- **Final Centroids**: $(3.67, 9.0)$, $(7.0, 4.33)$, $(1.5, 3.5)$

Point	Coordinates	Dist Mean 1	Dist Mean 2	Dist Mean 3	Cluster
A1	(2, 10)	1.60	6.12	8.83	1
A2	(2, 5)	4.12	4.01	2.50	3

Point	Coordinates	Dist Mean 1	Dist Mean 2	Dist Mean 3	Cluster
A3	(8, 4)	8.64	2.62	6.86	2
A4	(5, 8)	1.66	4.17	5.7	1
A5	(7, 5)	7.68	1.75	5.92	2
A6	(6, 4)	8.28	2.35	5.22	2
A7	(1, 2)	7.23	7.07	1.80	3
A8	(4, 9)	1.41	3.77	6.80	1

Comments:

- The algorithm converges as centroids stabilize and no further changes occur.

Final Results

- Each point has been assigned to its nearest cluster.
- The process has converged after 4 iterations.

Key Observations

1. **Centroid Movement**: Centroids progressively adjust to the cluster's center during each iteration.
2. **Convergence**: K-Means stops when the centroids no longer move.

Python Example

Here's how to implement K-Means clustering in Python using `scikit-learn` and Mall Customers Information
We will use a using Mall Customers Dataset to demonstrate kmeans clustering using scikit Learn Library.

Index	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
5	6	Female	22	17	76
6	7	Female	35	18	6

Index	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	20	99
12	13	Male	58	20	15
13	14	Female	24	20	77
14	15	Male	22	20	79

Import Libraries and Dataset

In [ ]:
```
# Import necessary libraries
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Step 1: Get the data
df = pd.read_csv('https://raw.githubusercontent.com/msfasha/307304-Data-Mining/refs
df
```

Out[ ]:

	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 × 5 columns

Apply K-Means with K = 3

In [ ]:
```python
# Step 2: Select relevant features (Annual Income and Spending Score) for clusterin
features = df[["Annual Income (k$)", "Spending Score (1-100)"]]

# Step 3: Apply K-Means with K=3
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(features)

# Step 4: Add the cluster labels to the DataFrame
df["Cluster"] = kmeans.labels_
```

Out[ ]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	Male	19	15	39	2
1	2	Male	21	15	81	2
2	3	Female	20	16	6	2
3	4	Female	23	16	77	2
4	5	Female	31	17	40	2
...	...	...	...	...	...	...
195	196	Female	35	120	79	1
196	197	Female	45	126	28	0
197	198	Male	32	126	74	1
198	199	Male	32	137	18	0
199	200	Male	30	137	83	1

200 rows × 6 columns

Print the updated dataframe showing the assigned cluster for each record

In [ ]:
```python
df
```

Out[ ]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	Male	19	15	39	2
1	2	Male	21	15	81	2
2	3	Female	20	16	6	2
3	4	Female	23	16	77	2
4	5	Female	31	17	40	2
...	...	...	...	...	...	...
195	196	Female	35	120	79	1
196	197	Female	45	126	28	0
197	198	Male	32	126	74	1
198	199	Male	32	137	18	0
199	200	Male	30	137	83	1

200 rows × 6 columns

Print out the centroids of the clusters

In [ ]:
```
kmeans.cluster_centers_
```

Out[ ]:
```
array([[87.        , 18.63157895],
       [86.53846154, 82.12820513],
       [44.15447154, 49.82926829]])
```

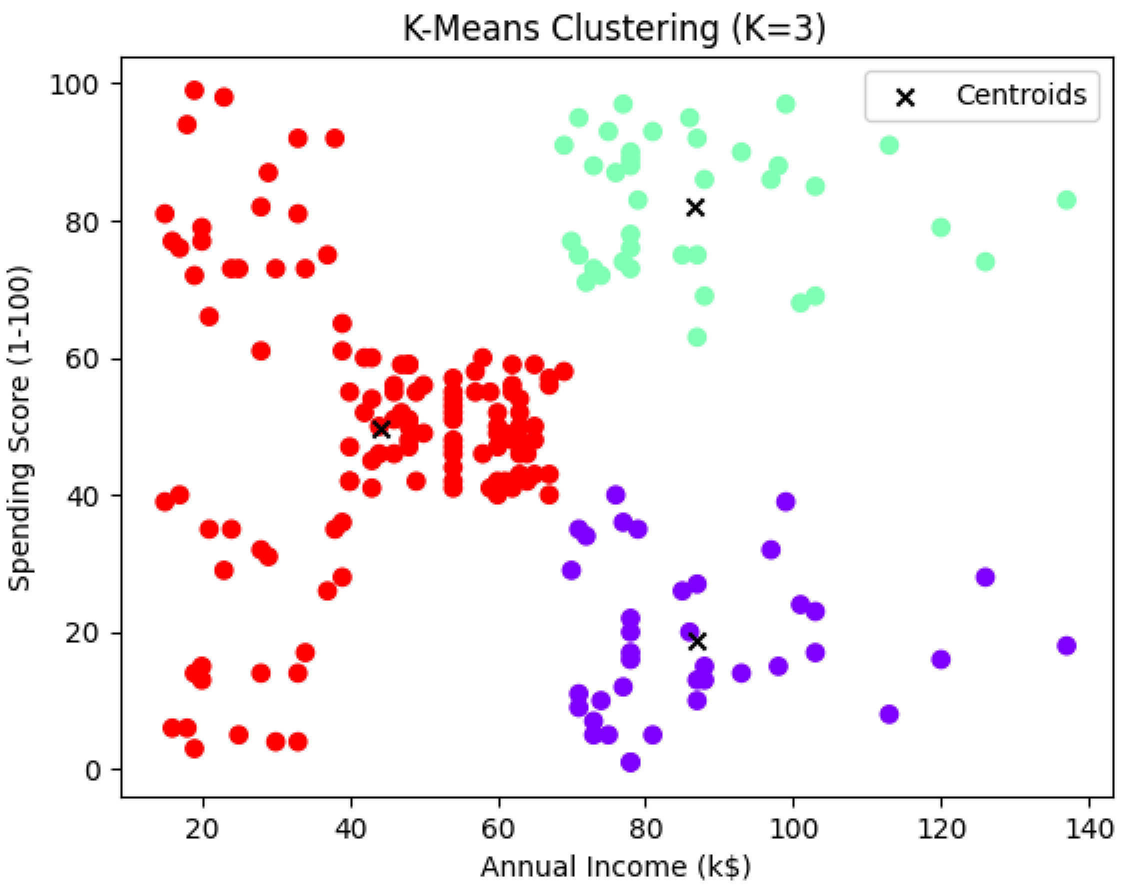
Format centroids in a dataframe for better visualization

In [ ]:
```
# Assuming features and KMeans object already defined
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=features.columns)

# Display centroids for interpretation
print("Centroids for each cluster:")
print(centroids)
```

```
Centroids for each cluster:
   Annual Income (k$)  Spending Score (1-100)
0           87.000000               18.631579
1           86.538462               82.128205
2           44.154472               49.829268
```

Display The Scatter Plot and the DataFrame

In [ ]:
```
# Step 5: Plot the clusters and centroids
plt.scatter(features["Annual Income (k$)"], features["Spending Score (1-100)"], c=d
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color="bl
plt.title("K-Means Clustering (K=3)")
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.legend()
plt.show()
```



Explanation:

- **Step 1**: We create a simple 2D dataset.
- **Step 2**: We apply K-Means clustering with $K = 3$.
- **Step 3**: We assign each data point to a cluster and store the result in the DataFrame.
- **Step 4**: We plot the clusters with different colors and mark the centroids with black 'x's.

Observations for (k=3):

1. **Cluster Formation**:
   - The data points are divided into **three distinct clusters**, differentiated by their spending scores and annual incomes.
   - The clusters reflect distinct patterns in customer behavior.
2. **Cluster Characteristics**:
   - **Cluster 1 (e.g., Red)**:

- Represents customers with **low to moderate income** and a **wide range of spending scores**.
- Includes customers who might be more price-conscious or have varied spending behaviors.

- **Cluster 2 (e.g., Purple)**:
  - Comprises customers with **high income** but **low spending scores**.
  - Indicates potentially frugal or savings-oriented individuals, despite higher incomes.
- **Cluster 3 (e.g., Green)**:
  - Includes customers with **high income** and **high spending scores**.
  - Likely represents high-value customers who are ideal for premium marketing campaigns.

3. **Centroid Placement**:

- The centroids (black "X" markers) represent the mean position of each cluster.
- The distance between centroids suggests clear separation between the groups.

4. **Insights**:

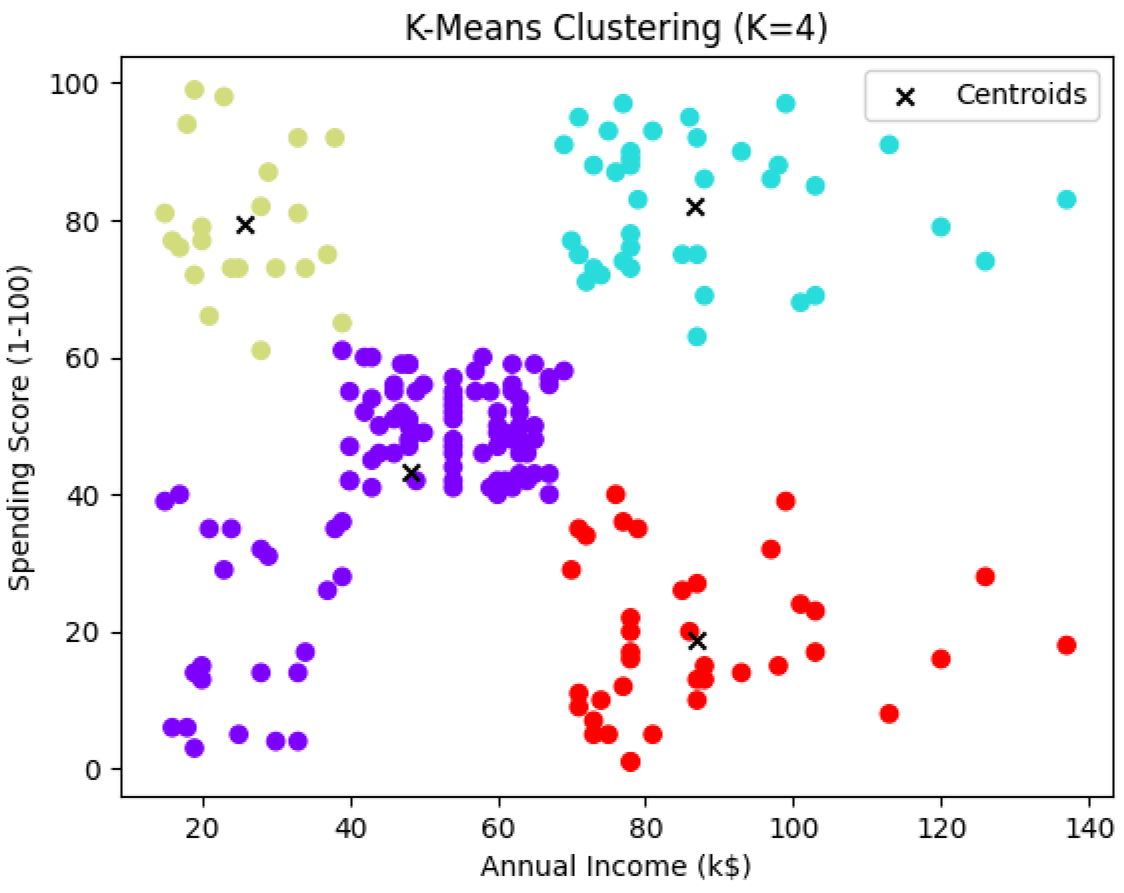
- This segmentation aligns well with typical customer segmentation strategies in marketing:
  - **Cluster 1**: Target with discounts or budget-friendly options.
  - **Cluster 2**: Upsell higher-end products or encourage spending.
  - **Cluster 3**: Retain as loyal high-value customers with exclusive offers.

5. **Potential Next Steps**:

- Validate the clusters with real-world customer behavior (e.g., demographic or shopping preferences).
- Experiment with **k=4** to check for further sub-clusters.
- Perform **3D clustering** by adding another feature (e.g., Age or Gender).

Apply K-Means with K = 4

In [ ]:
```
# Select relevant features (Annual Income and Spending Score) for clustering
features = df[["Annual Income (k$)", "Spending Score (1-100)"]]

# Apply K-Means with K=4
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(features)

# Add assigned clusters to each row
df["Cluster"] = kmeans.labels_

# Plot the clusters and centroids
plt.scatter(features["Annual Income (k$)"], features["Spending Score (1-100)"], c=d
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color="bl
plt.title("K-Means Clustering (K=4)")
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.legend()
plt.show()
```



With (k=4) clusters, the customer segmentation becomes more refined. Here are the observations and insights:

Observations

1. **Cluster Formation**:
   - The data is divided into four distinct clusters, with clear separation along **Annual Income (k$)** and **Spending Score (1-100)**.
2. **Cluster Characteristics**:
   - **Cluster 1 (Light Green)**:
     - **Low Annual Income**, **High Spending Score**.
     - Represents customers who spend a large proportion of their income, possibly impulsive spenders or younger customers.
   - **Cluster 2 (Purple)**:
     - **Moderate Annual Income**, **Moderate Spending Score**.
     - Likely represents the "average" customer with balanced income and spending behavior.

- **Cluster 3 (Light Blue)**:
  - **High Annual Income**, **High Spending Score**.
  - Represents premium, high-value customers with significant purchasing power.
- **Cluster 4 (Purple)**:
  - **High Annual Income**, **Low Spending Score**.
  - Likely represents frugal or savings-oriented individuals who may prioritize saving over spending despite their high income.

3. **Centroids**:

- The centroids (black "X" markers) represent the average characteristics of each cluster.
- Clear separation of centroids confirms meaningful clusters.

Insights

1. **Marketing Strategies**:

- **Cluster 1 (Light Green)**:
  - Target with discounts, promotions, or budget-friendly offers to maximize spending potential.
- **Cluster 2 (Purple)**:
  - Maintain regular engagement, as they represent a balanced customer segment.
- **Cluster 3 (Light Blue)**:
  - Focus on premium products and services. These customers are ideal for exclusive campaigns.
- **Cluster 4 (Purple)**:
  - Consider introducing tailored promotions or incentivizing spending for this segment.

2. **Segmentation Insights**:

- Including (k=4) provides finer segmentation, highlighting distinct behavioral groups like "impulsive spenders" and "savings-oriented high-income customers."
- This segmentation can inform personalized marketing and product offerings.

3. **Potential for Further Analysis**:

- **3D Clustering**: Add a third feature like **Age** to see if it further refines the clusters.
- **Exploring Gender**: Overlay gender information to analyze behavioral patterns by gender.

Cluster Data using 3 Features

In [ ]:
```
# Step 2: Select relevant features (Annual Income and Spending Score) for clusterin
features = df[["Annual Income (k$)", "Spending Score (1-100)", "Age"]]
```

```
# Step 3: Apply K-Means with K=3
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(features)

# Step 4: Add the cluster labels to the DataFrame
df["Cluster"] = kmeans.labels_

df
```

Out[ ]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	Male	19	15	39	2
1	2	Male	21	15	81	2
2	3	Female	20	16	6	0
3	4	Female	23	16	77	2
4	5	Female	31	17	40	2
...	...	...	...	...	...	...
195	196	Female	35	120	79	1
196	197	Female	45	126	28	1
197	198	Male	32	126	74	1
198	199	Male	32	137	18	1
199	200	Male	30	137	83	1

200 rows × 6 columns

Display Centroids

In [ ]:

```
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=features.columns)

# Display centroids for interpretation
print("Centroids for each cluster:")
print(centroids)
```

```
Centroids for each cluster:
   Annual Income (k$)  Spending Score (1-100)        Age
0           59.879032               35.427419  44.483871
1           88.731707               79.243902  32.975610
2           29.971429               68.514286  25.771429
```

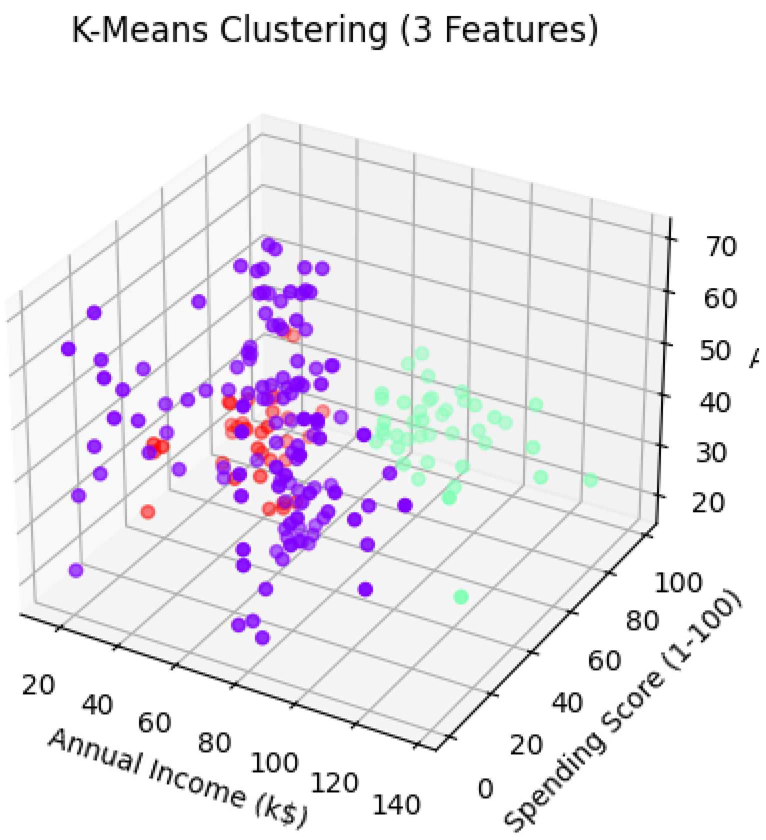
3D Scatter Plot

We can use a 3D scatter plot to visualize three features together.

In [ ]:

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')

# Plotting points
ax.scatter(
    df["Annual Income (k$)"],
    df["Spending Score (1-100)"],
    df["Age"],
    c=df["Cluster"], cmap="rainbow"
)

# Label axes
ax.set_title("K-Means Clustering (3 Features)")
ax.set_xlabel("Annual Income (k$)")
ax.set_ylabel("Spending Score (1-100)")
ax.set_zlabel("Age")
plt.show()
```



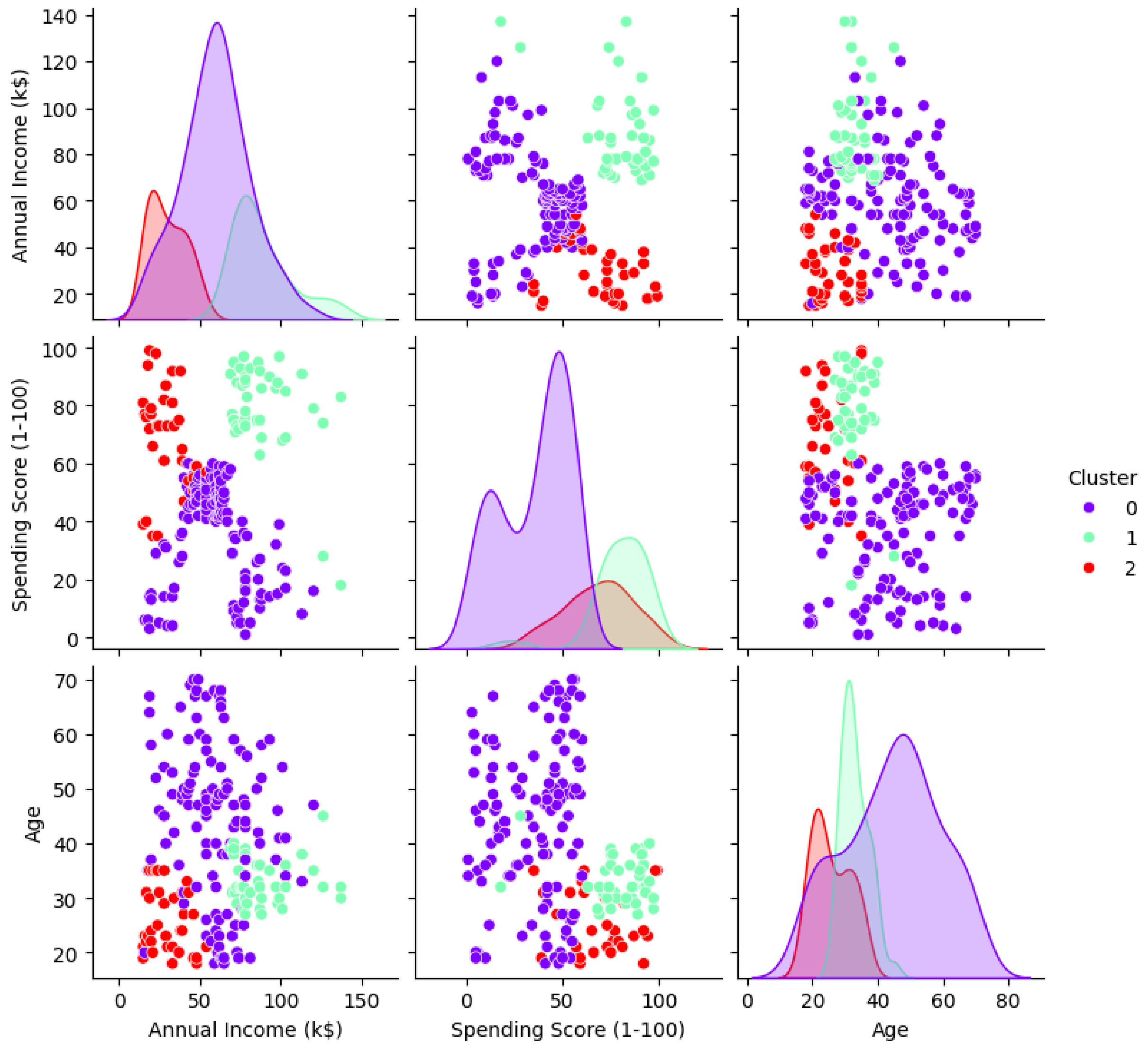
Use Plotly Package to Create Dynamic Plots

In [ ]:
```
import plotly.express as px

# Create an interactive 3D scatter plot
fig = px.scatter_3d(
    df,
    x="Annual Income (k$)",
    y="Spending Score (1-100)",
    z="Age",
    color="Cluster",  # Color points by cluster
    title="K-Means Clustering (3 Features)",
    labels={
        "Annual Income (k$)": "Annual Income (k$)",
        "Spending Score (1-100)": "Spending Score (1-100)",
        "Age": "Age"
    },
    width=1000,  # Set the width of the plot
    height=800   # Set the height of the plot
)

# Show the interactive plot
fig.show()
```

Pairwise Scatter Plots

We can create scatter plots for each pair of the three features, highlighting clusters using different colors.

In [ ]:
```
import seaborn as sns

# Add Cluster information to the dataframe
sns.pairplot(df, vars=["Annual Income (k$)", "Spending Score (1-100)", "Age"], hue=
plt.show()
```



This pair plot visualizes the relationships between **Annual Income (k$)**, **Spending Score (1-100)**, and **Age** across the identified customer clusters.

Observations

1. **Cluster Separation**:

- **Cluster 0 (Purple)**:
  - Spreads across a wide range of **Annual Income** but tends to have **low Spending Scores**.
  - Includes a mix of ages, but younger individuals dominate.
- **Cluster 1 (Green)**:
  - Consists of customers with **high Spending Scores** and moderate to high **Annual Income**.
  - Likely includes customers who spend consistently regardless of age.
- **Cluster 2 (Red)**:

- Concentrated among customers with **low Annual Income** and **high Spending Scores**.
- Mostly young customers, indicating impulsive or budget-focused spenders.

2. **Feature Interactions**:

- **Annual Income vs. Spending Score**:
  - Cluster 0 includes customers with higher income but lower spending.
  - Cluster 1 indicates high-spending customers across various income levels.
  - Cluster 2 reflects high spending at lower income levels, highlighting a younger demographic.
- **Age**:
  - Younger customers dominate **Cluster 2**, while **Cluster 1** has a broader age range.
  - Older customers tend to appear in **Cluster 0**, indicating cautious spending habits.

Recommendations

1. **Targeted Marketing**:

- **Cluster 0 (Low Spending, Higher Income)**:
  - Encourage spending with tailored promotions or loyalty programs.
  - Highlight premium products to appeal to their income level.
- **Cluster 1 (High Spending, Various Income)**:
  - Retain these high-value customers through exclusive rewards or VIP experiences.
  - Offer premium and mid-tier products.
- **Cluster 2 (Low Income, High Spending)**:
  - Focus on budget-friendly offers and promotions.
  - Use engaging campaigns to maintain loyalty among younger customers.

2. **Product Offerings**:

- **Cluster 1**:
  - Expand premium offerings or subscription services for consistent high spenders.
- **Cluster 2**:
  - Focus on affordability and discounts to cater to their financial constraints.

3. **Explore Subclusters**:

- Analyze **Cluster 0** further to identify why some high-income customers spend less.
- Investigate potential subsegments in **Cluster 1** to tailor campaigns further.

4. **Use Age Segmentation**:

- Younger customers (in **Cluster 2**) might respond to digital campaigns.

- Older customers (in **Cluster 0**) may value traditional marketing and personalized service.

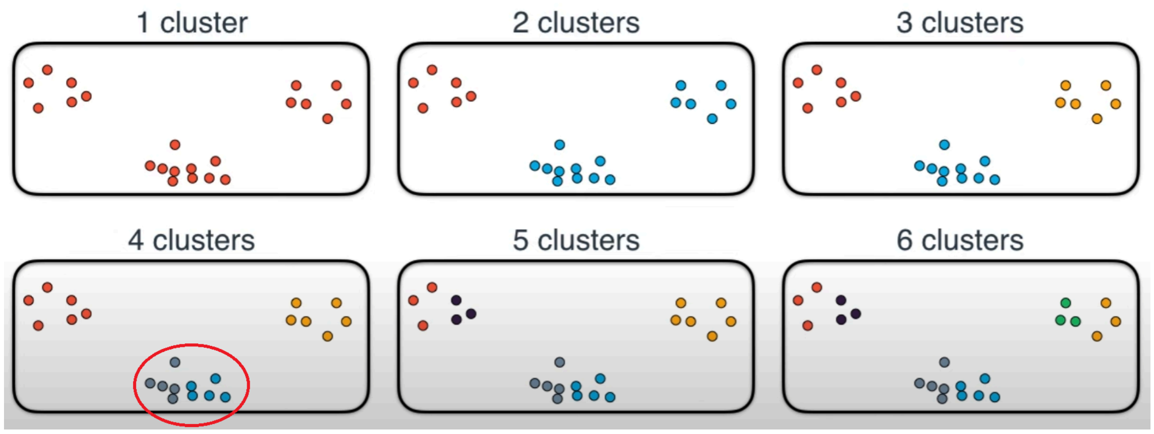
Clustering Quality Measures

Most of the time, business determine the requred clustered, but when they don't, we can use two methods/measures that can assist us to determine the best number of clusters, these are:

- The Eblow Method
- The Silhouette Metric

Choosing the Right K For K-Means Clustering (The Elbow Method)

One challenge with K-Means is selecting the appropriate number of clusters K.
We can try different cluster numbers and plot them and eye ball the goodness of the created clusters.
For example, in the image below, we can notice that we started to create unclear clusters **starting from K = 4**.



The **Elbow Method** is a technique used to determine the optimal K by plotting the **within-cluster sum of squares (WCSS)** against different values of K.

Steps of the Elbow Method:

1. Run K-Means for a range of K values.
2. Plot the WCSS for each K.
3. Look for the "elbow" point in the plot, where the decrease in WCSS slows down. This is the ideal K.

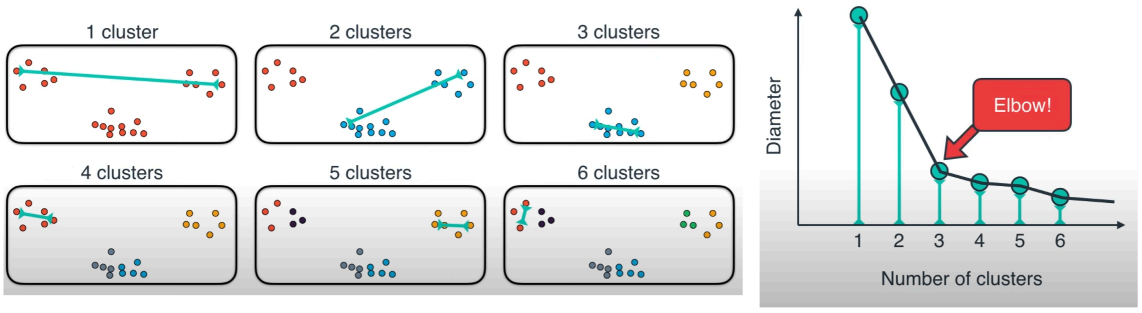
The WCSS Formula (Within-Cluster Sum of Squares)

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Explanation

- k: Number of clusters
- C_i: The set of points in cluster i
- x: A data point in cluster C_i
- μ_i: The centroid (mean) of cluster C_i
- $\|x - \mu_i\|^2$: Squared Euclidean distance between the point and the cluster centroid

This formula calculates the sum of the squared distances between each point and the centroid of its assigned cluster, and is used to evaluate the compactness of clusters in algorithms like **K-Means**.



The Elbow Method in Python

Import the required Python Libraries and Load the Dataset

```
In [ ]: # Import necessary libraries
        import pandas as pd
        from sklearn.cluster import KMeans
        import matplotlib.pyplot as plt

        # Get the data
        df = pd.read_csv('https://raw.githubusercontent.com/msfasha/307304-Data-Mining/refs
        features = df[["Annual Income (k$)", "Spending Score (1-100)"]]
```

Implement the Elbow Method

In [ ]:
```
# Use the kmeans.inertia_() method to compute the WCSS for each K setting

# Initialize a list to store the WCSS values
wcss = []

# Using the features "Annual Income (k$)" and "Spending Score (1-100)"
for k in range(1, 11):  # Testing cluster counts from 1 to 10
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(features)  # Fit on the selected features
    wcss.append(kmeans.inertia_)  # Inertia is the WCSS
```

Display the Computed WCSS values for the Different K values

Notice how the distance is largest when K = 1 and how the difference in distance starts to shrink as K increases

In [ ]:
```
# Sort WCSS values in descending order and round to 2 decimal places, then print
sorted_wcss = sorted(wcss, reverse=True)
print("WCSS values for k from 1 to 10 (sorted, 2 decimal places):")
for i, w in enumerate(sorted_wcss, start=1):
    print(f"K={i}: {w:.2f}") # Formatted output with 2 decimal places
# Plot the WCSS values
```

Plot the Elbow Graph

In [ ]:
```
# Plot the elbow graph
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.grid()
plt.show()
```

The elbow chart above illustrates the process of determining the optimal number of clusters for customer segmentation based on the features "Annual Income (k$)" and "Spending Score (1-100)." Key observations include:

- **Purpose:** The chart plots the Within-Cluster Sum of Squares (WCSS) against the number of clusters to identify the point where adding more clusters does not significantly reduce WCSS.
- **Trend:** As the number of clusters increases, WCSS decreases due to better compactness within each cluster.
- **Elbow Point:**
  - The "elbow" is observed at **4 or 5 clusters**, where the rate of decrease in WCSS slows down noticeably.
  - This indicates the optimal number of clusters, balancing simplicity and effectiveness.

- **Interpretation:** Selecting 4 or 5 clusters will likely result in meaningful segmentation of customers into distinct groups for analysis or targeted strategies.

Validating Results

After setting the k number of clusters, we can validate the clustering using thye following methods:
i. **Number of Clusters**: Check if the number of clusters is reasonable for your data.

ii. **Visual Inspection**: Plot the clusters and noise points to ensure they align with the data's structure.

iii. **Silhouette Score**: We can also use the $Silhouette\ Score$ which tells us how well each point fits into its cluster (see below at the end of this notebook).

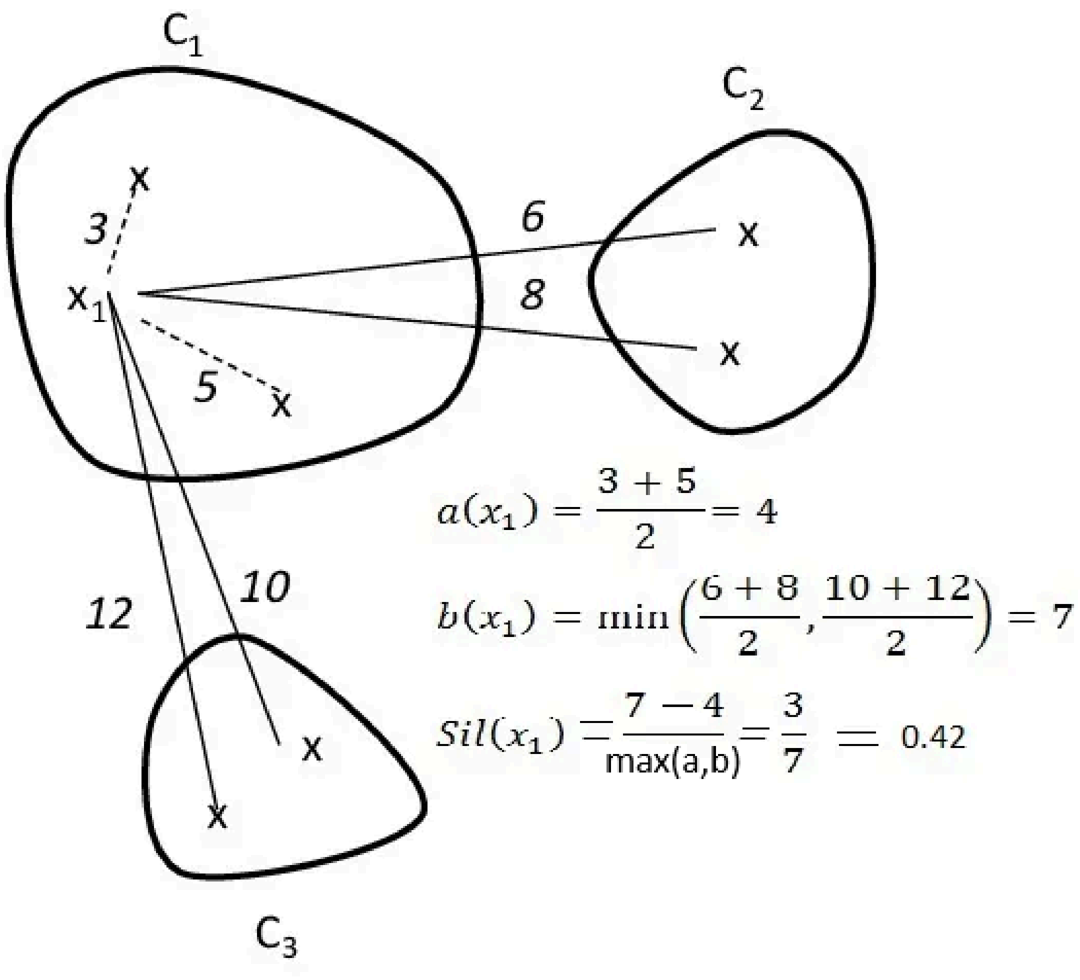
Validate Clusters using the Silhouette Score

The **silhouette score** is a metric used to evaluate the quality of clustering. It gives an idea of how similar a point is to its own cluster compared to other clusters.

What is the silhouette measure?

- For each data point, the silhouette score measures two things:
  1. **Cohesion (a):** How close the point is to other points in its own cluster.
  2. **Separation (b):** How far the point is from points in the nearest other cluster.
- The silhouette score for a point is calculated as:

$$s = \frac{b - a}{\max(a, b)}$$


C_1
C_2
C_3
x_1
3
5
6
8
10
12
x
$a(x_1) = \frac{3+5}{2} = 4$
$b(x_1) = \min\left(\frac{6+8}{2}, \frac{10+12}{2}\right) = 7$
$Sil(x_1) = \frac{7-4}{\max(a,b)} = \frac{3}{7} = 0.42$

where:

- (a): Average distance to other points in the same cluster (within-cluster distance).
- (b): Average distance to points in the nearest other cluster (nearest-cluster distance).
- The score ranges between:
  - **+1**: The point is very close to its own cluster and far from other clusters (well-clustered).
  - **0**: The point is on or near the boundary between two clusters.
  - **-1**: The point is closer to another cluster than its own (poorly clustered).

In [ ]:
```python
from sklearn.metrics import silhouette_score

# Compute Silhouette Score
if len(set(labels)) > 1:  # Avoid silhouette score error for single cluster
    score = silhouette_score(features, labels)
    print(f"Silhouette Score: {score:.2f}")
else:
    print("Silhouette Score cannot be calculated for a single cluster.")
```

Score = 0.36 — is it suitable?

A score of 0.36 is:

Moderate clustering quality.

It suggests that there is some structure, but the clusters may not be well-separated or clearly defined.

General Benchmarks:

- > 0.7: Strong structure, well-separated clusters.
- 0.5 – 0.7: Reasonable structure.
- 0.25 – 0.5: Weak structure (yours falls here).
- < 0.25: Poor structure.

So, 0.36 is acceptable, especially if your data is complex or noisy, but there is likely room for improvement (e.g., tuning number of clusters, trying different features, or using another clustering algorithm).