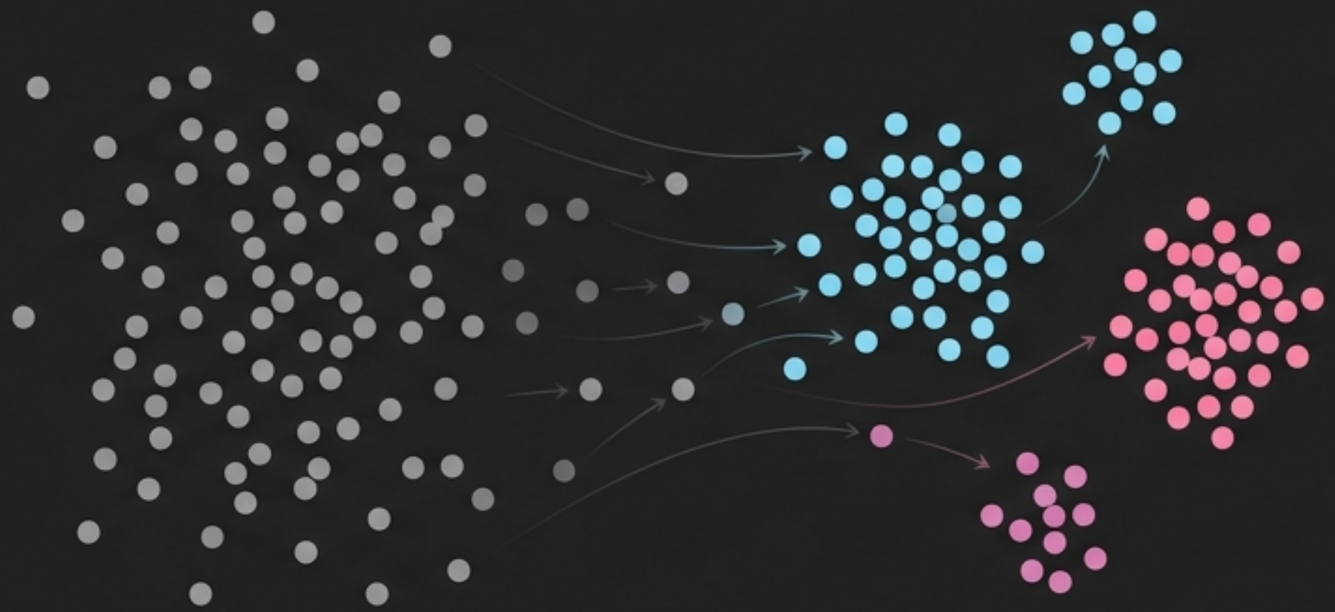
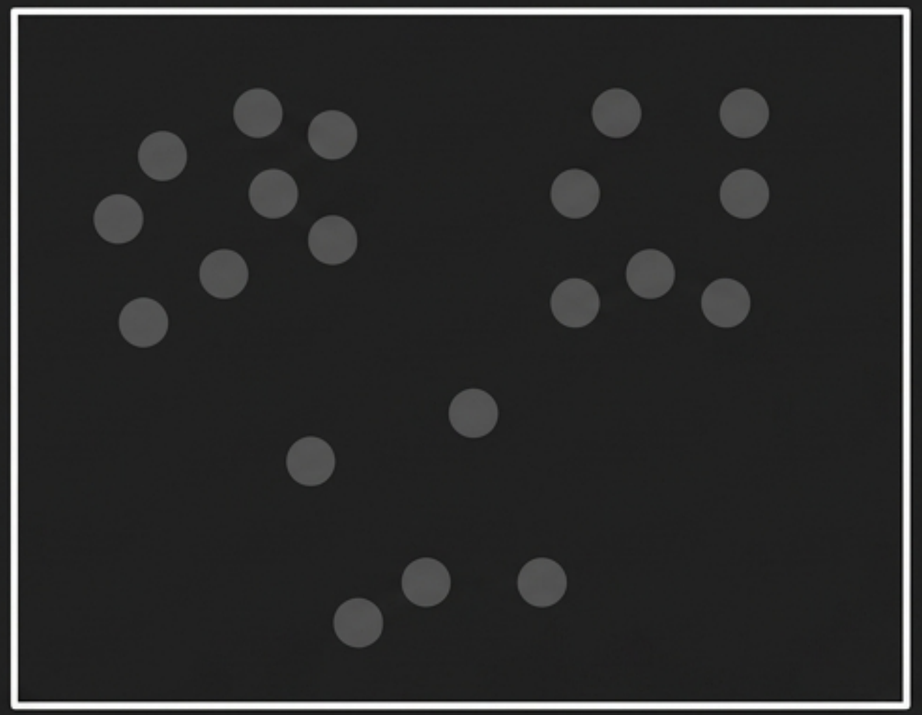
K-Means Clustering: From Chaos to Structure

An Unsupervised Learning Approach to Partitioning Unlabeled Data.



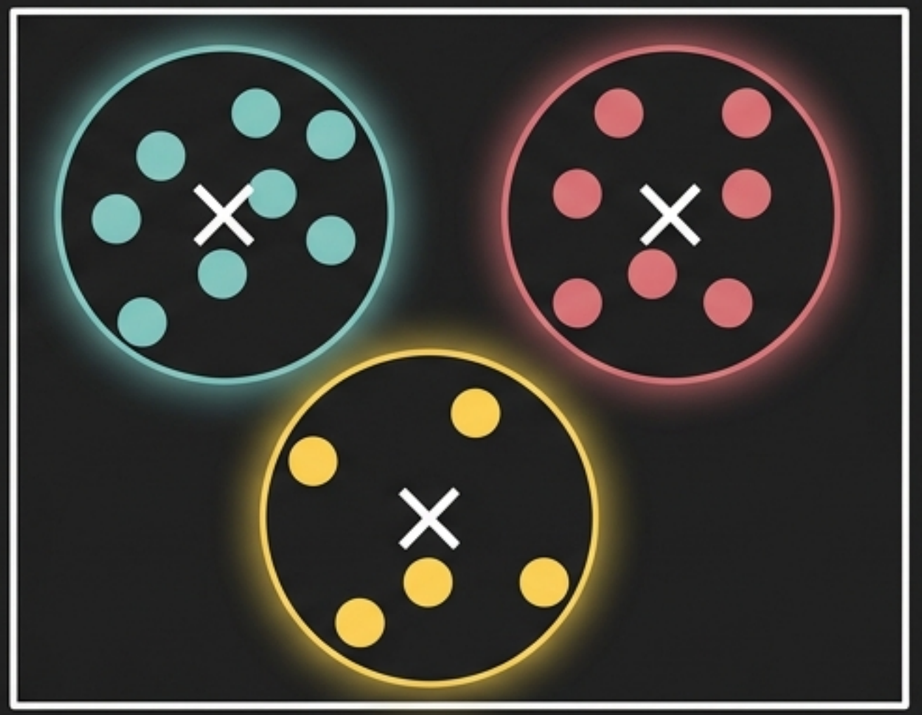
Module: Concept → Optimization → Code

Input: Unlabeled Data



K-Means Algorithm

Output: Labeled Clusters



“Organizing the Unknown”

Core Definition: K-Means is an unsupervised algorithm. It partitions data into distinct, non-overlapping clusters based strictly on feature similarity, without pre-existing labels.

Goal: Minimize the distance between points and their assigned cluster center (**Centroid**).

The Learning Loop

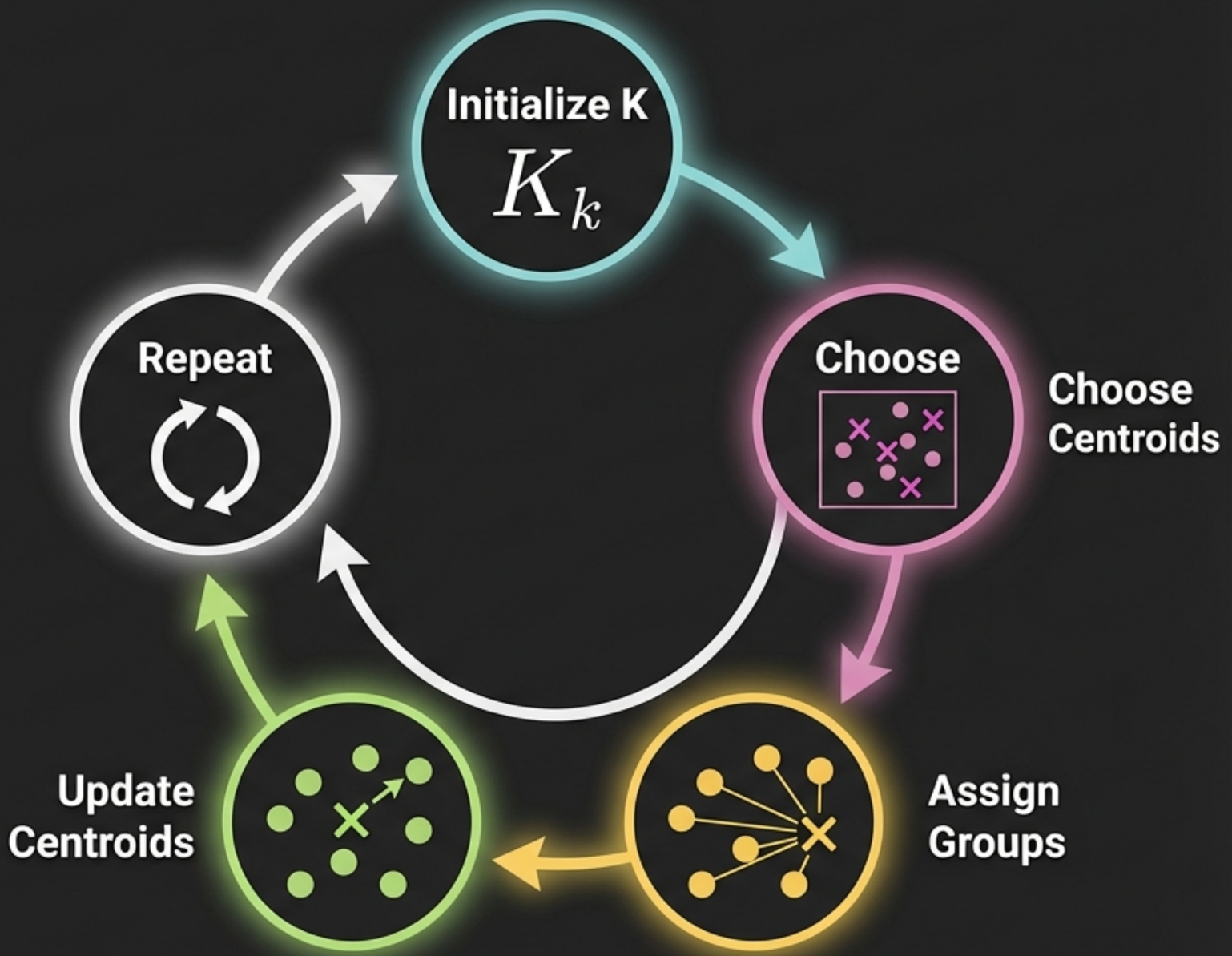
1. **Initialize**: Decide the number of clusters (K).

2. **Select**: Choose random initial centroids.

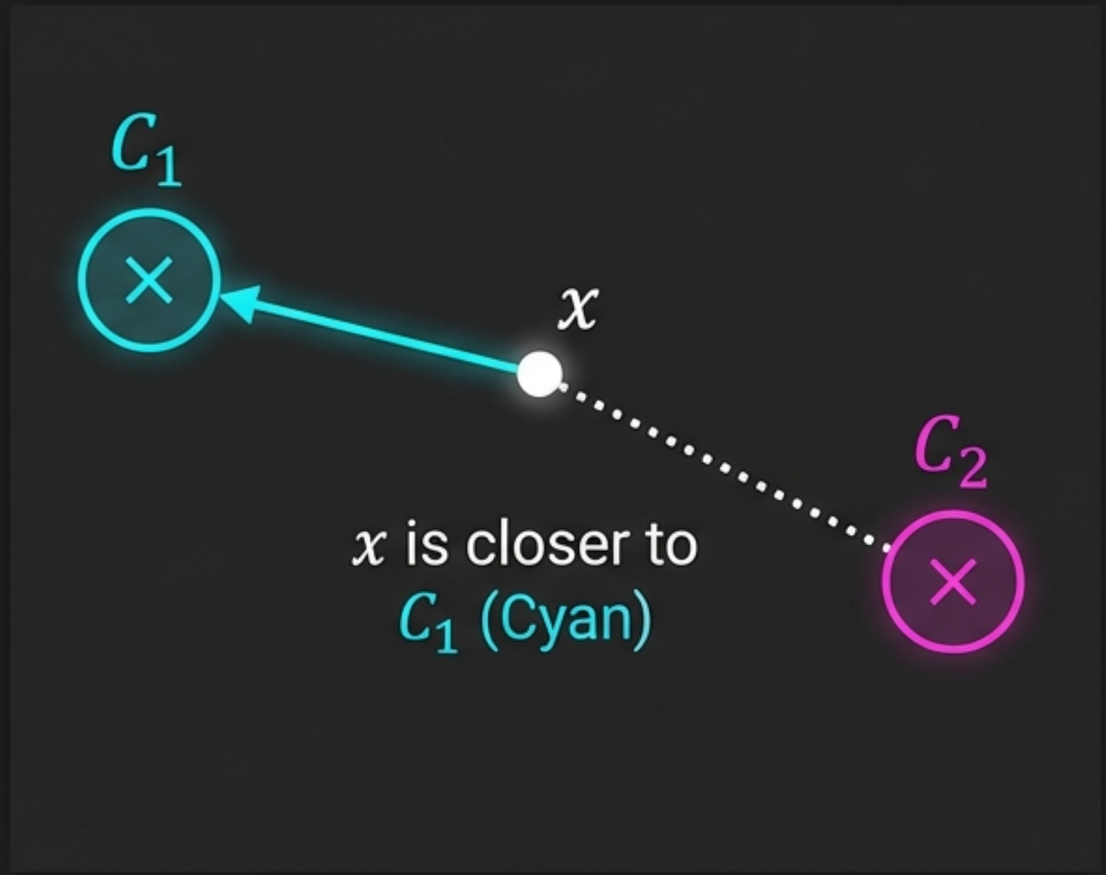
3. **Assign**: Link every point to the nearest centroid (Euclidean distance).

4. **Update**: Move the centroid to the calculated Mean of its new group.

5. **Converge**: Repeat steps 3-4 until centroids stop moving.



The Engine: Geometry & Convergence



Euclidean Distance:

$$dis(x, c) = \sqrt{\sum(x_i - c_i)^2}$$

Logic: Point x joins the cluster with the minimum distance.

Update Centroid (Mean):

$$New\ Centroid = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Logic: The centroid moves to the exact average position of all points in its team.

Optimization: The Search for ‘K’

K=2 (Underfitting)

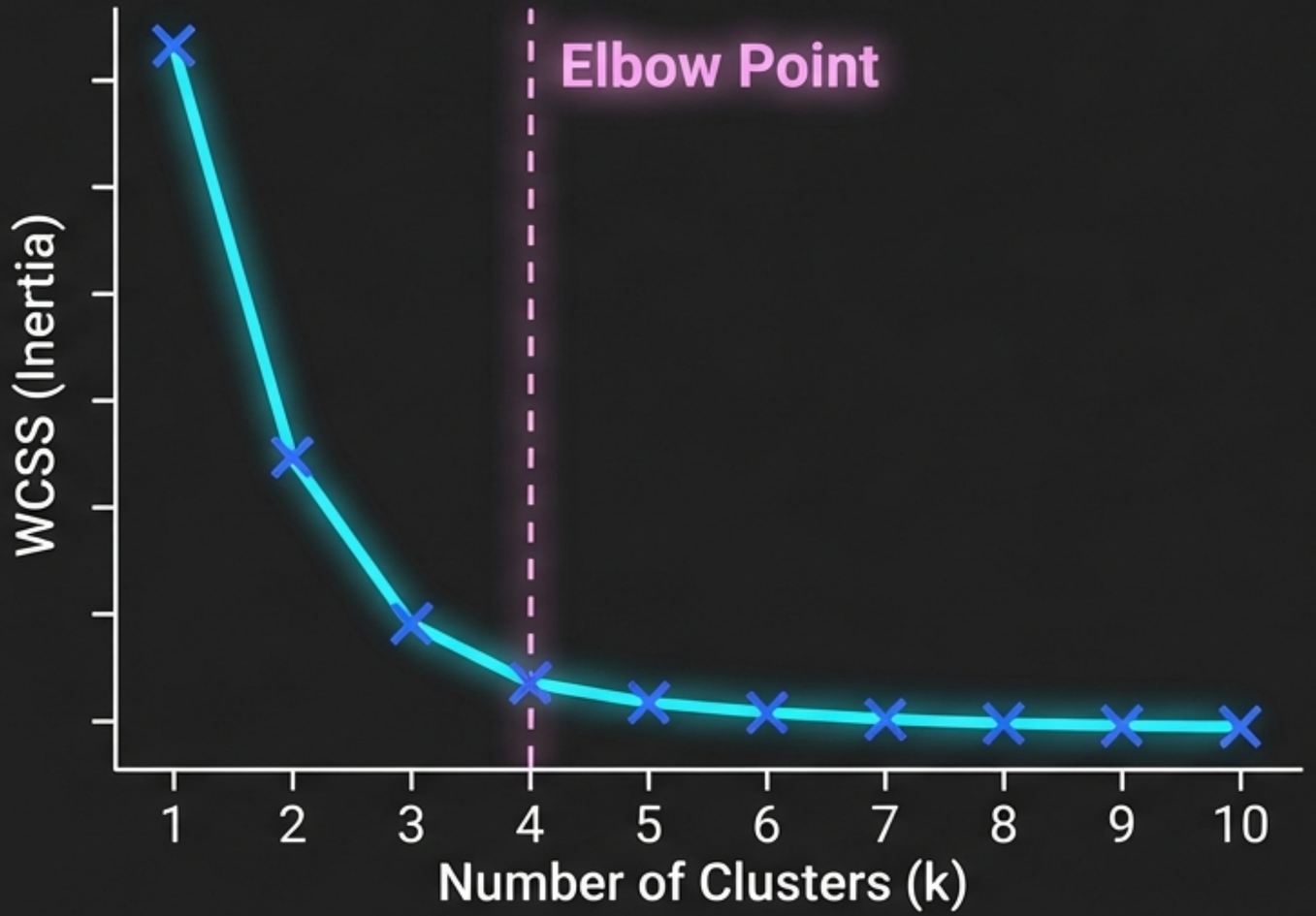
K=4 (Optimal?)

K=15 (Overfitting)

The Metric: Inertia (WCSS)

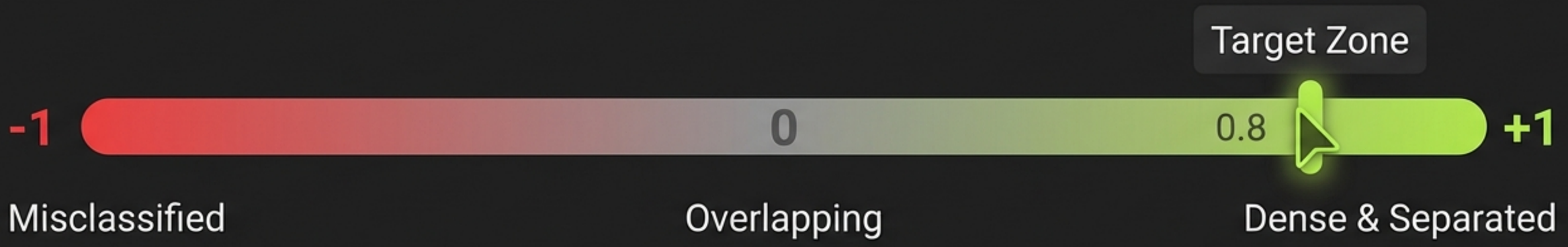
> Within-Cluster Sum of Squares measures how compact the clusters are.
> Goal: We want low Inertia (tight clusters) without artifcially fragmenting the data.

The Elbow The Elbow Method



- 1. **Run K-Means** for a range of K (1-10).
- 2. Plot **WCSS** for each run.
- 3. Identify the **bend**. This is the point of diminishing returns.
- 4. **Select K** at the elbow (Here, K=4).

Validation: The Silhouette Score



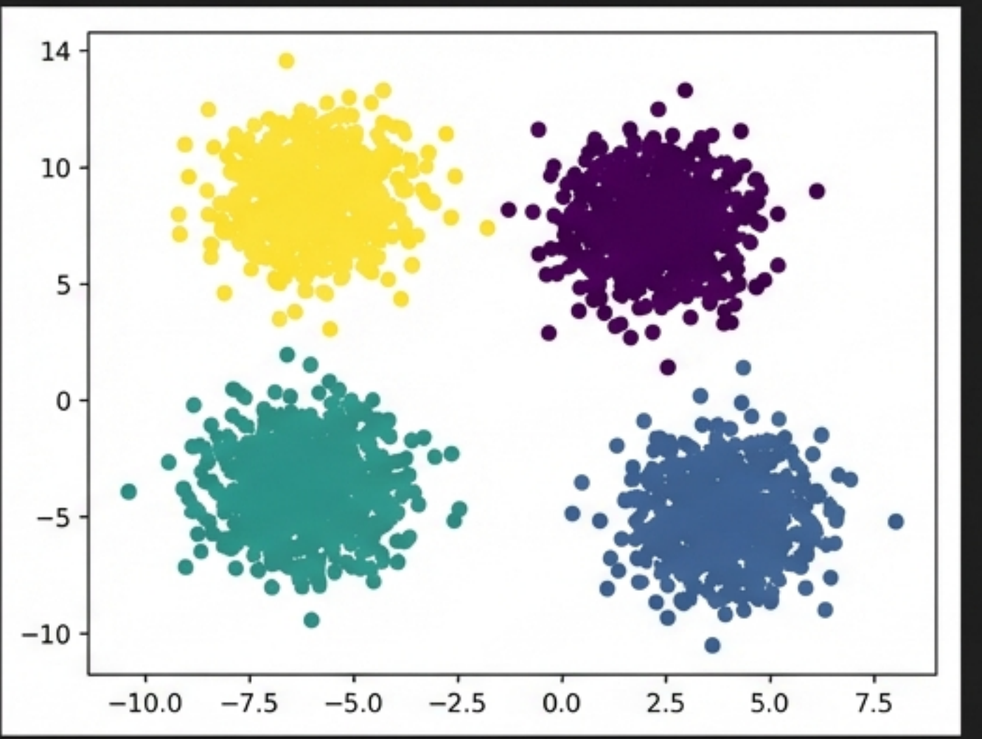
Logic: Measures Cohesion (distance to own cluster) vs. Separation (distance to nearest neighbor).

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

Implementation: Generating Synthetic Chaos

Step 1: Create the dataset

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate 1000 distinct points with 4 centers
X, y = make_blobs(n_samples=1000,
                  n_features=2,
                  centers=4,
                  random_state=42)
```



Implementation: Standardization

Step 2: Scale the features

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Initialize Scaler
scaler = StandardScaler()

# Fit and transform ensuring equal variance
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Why? K-Means is distance-based. If features have different scales (e.g., Age vs. Salary), the larger scale dominates the calculation. Scaling creates a level playing field.

Implementation: Coding the Elbow Search

Step 3: Find Optimal K

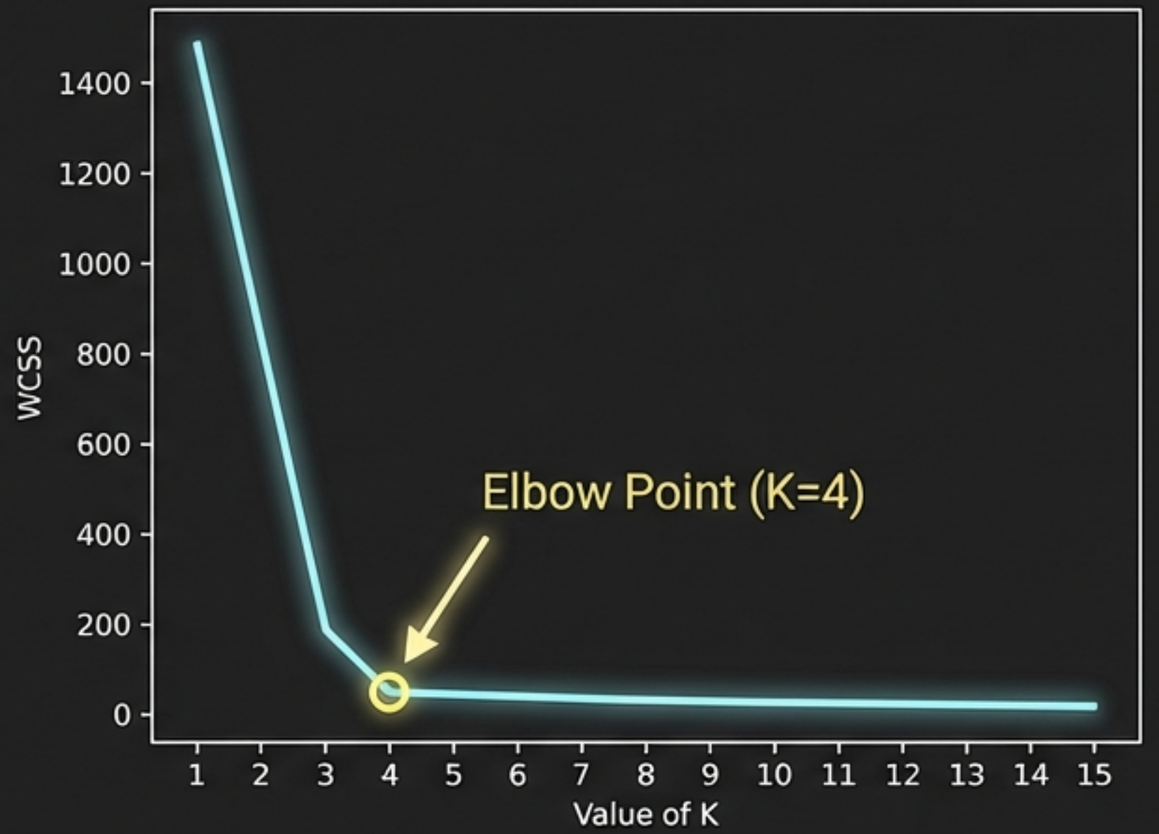
```
from sklearn.cluster import KMeans

wcss = []

# Loop through K values from 1 to 15
for k in range(1, 16):
    # k-means++ ensures smart centroid initialization
    kmeans = KMeans(n_clusters=k, init="k-means++")

    # Fit the model to the scaled training data
    kmeans.fit(X_train_scaled)

    # Store the inertia (WCSS)
    wcss.append(kmeans.inertia_)
```

Note: 'k-means++' is a smart initialization technique that speeds up convergence by spreading out initial centroids.

Implementation: Analyzing the Elbow Curve

Step 4: Plot the Elbow Curve

```
plt.plot(range(1, 16), wcss) # Plot WCSS vs K
plt.xlabel("Value of K")
plt.ylabel("WCSS")
plt.show()
```



Analysis: The curve bends sharply at K=4. This confirms 4 is the optimal cluster count.

Implementation: Training the Model

Step 4: Execute with K=4

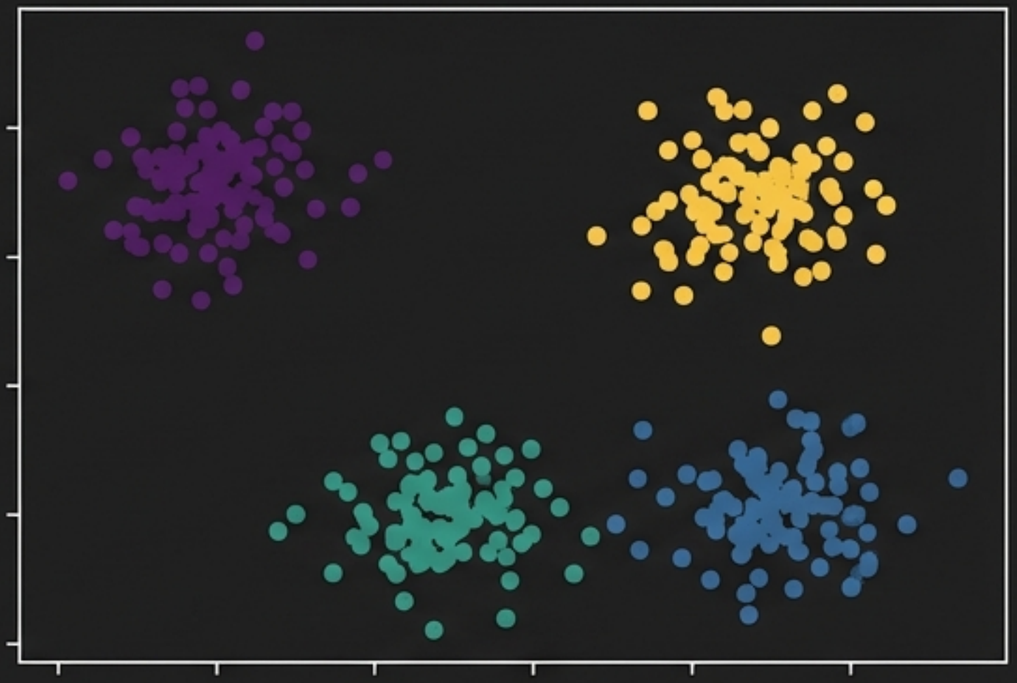
```
# Instantiate with optimal K
kmeans = KMeans(n_clusters=4, init="k-means++")

# Train the model
kmeans.fit(X_train_scaled)

# Predict cluster labels for test data
y_pred = kmeans.predict(X_test_scaled)

print(y_pred)
```

```
> Output:
array([1, 0, 2, 1, 3, 0, 2, 3, 2, 3...])
```

The model returns an array of integers (0-3), assigning each data point to one of the four clusters.

Implementation: Visualizing Structure

```
plt.scatter(X_test[:, 0],
            X_test[:, 1],
            c=y_pred)

plt.show()
```



Analysis: The scatter plot confirms the model has successfully structured the test data into four distinct clusters, matching the optimal K=4.

Implementation: Final Validation

```python
from sklearn.metrics import silhouette_score

# Calculate score
score = silhouette_score(X_train_scaled, kmeans.labels_)
print(score)
```

0.7958

Verdict: A score of ~0.80 indicates high-quality clustering. The groups are dense and well-separated.

Summary: The K-Means Workflow

01. Theory	02. Optimization	03. Preprocessing	04. Validation
Iterative algorithm that minimizes Euclidean distance to centroids.	Use the Elbow Method (WCSS Plot) to determine the correct K.	Always use StandardScaler to normalize feature variance.	Use Silhouette Score to confirm cluster separation and density.

Design based on [Source Material]. End of Presentation.