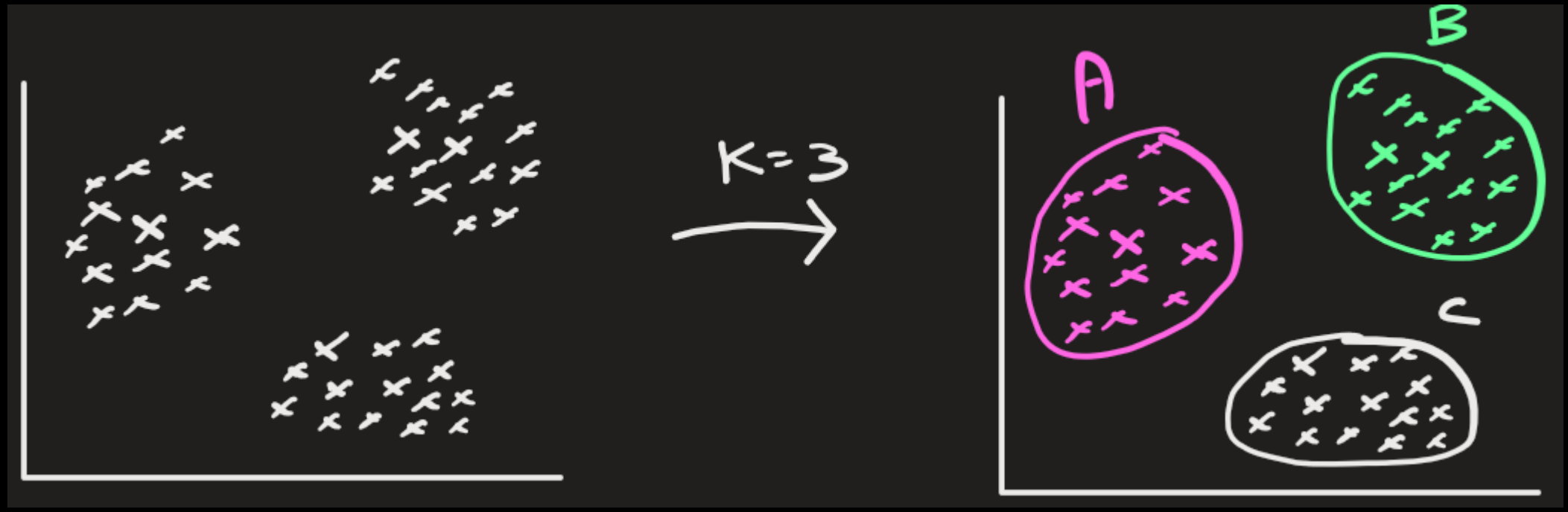
K-Means

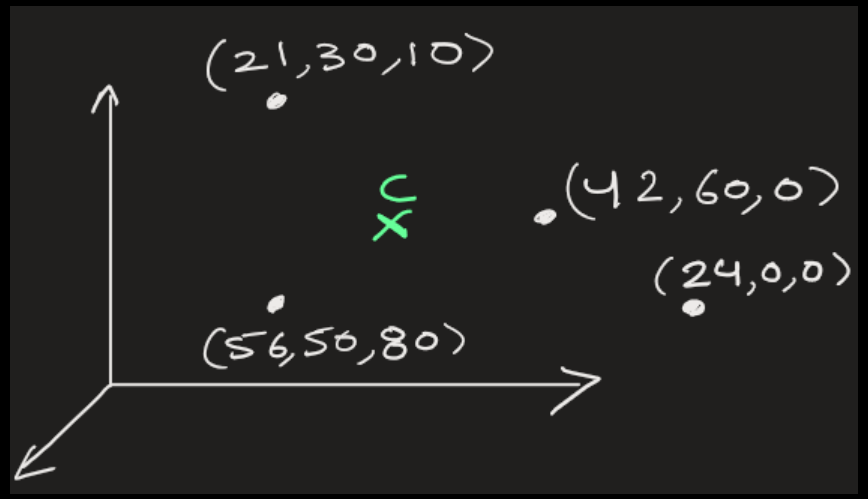
K-Means is an **unsupervised machine learning algorithm** used for **clustering**.

Its goal is simple:
Group similar data points together into *K clusters*.



Key Term: Centroid

age	Income in K	Debt in K
21	30	10
42	60	0
56	50	80
24	0	0



Centroid of above dataset:

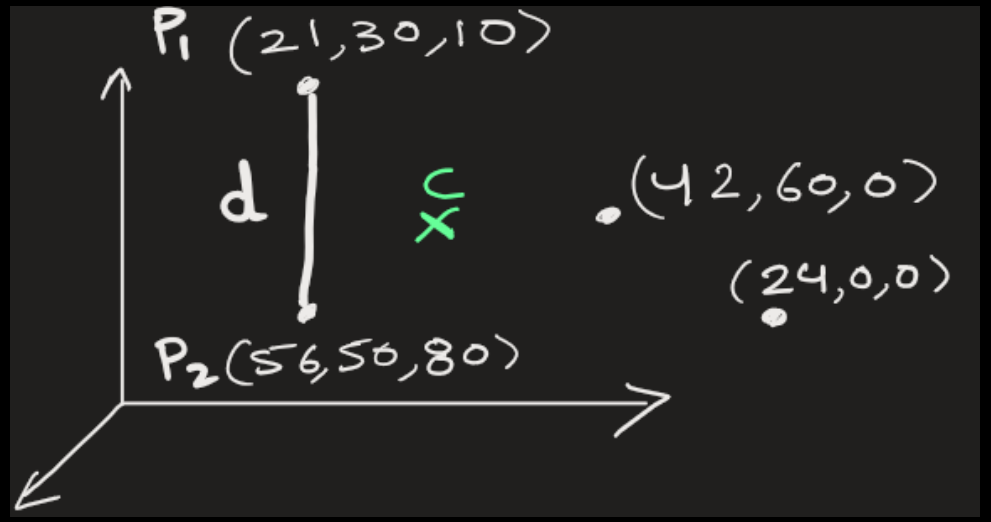
$$\left(\frac{21+42+56+24}{4}, \quad \frac{30+60+50+0}{4}, \quad \frac{10+0+80+0}{4}\right)$$

$$= (35.75, 35, 22.5)$$

K-Means

Key Term: Distance between 2 data points.

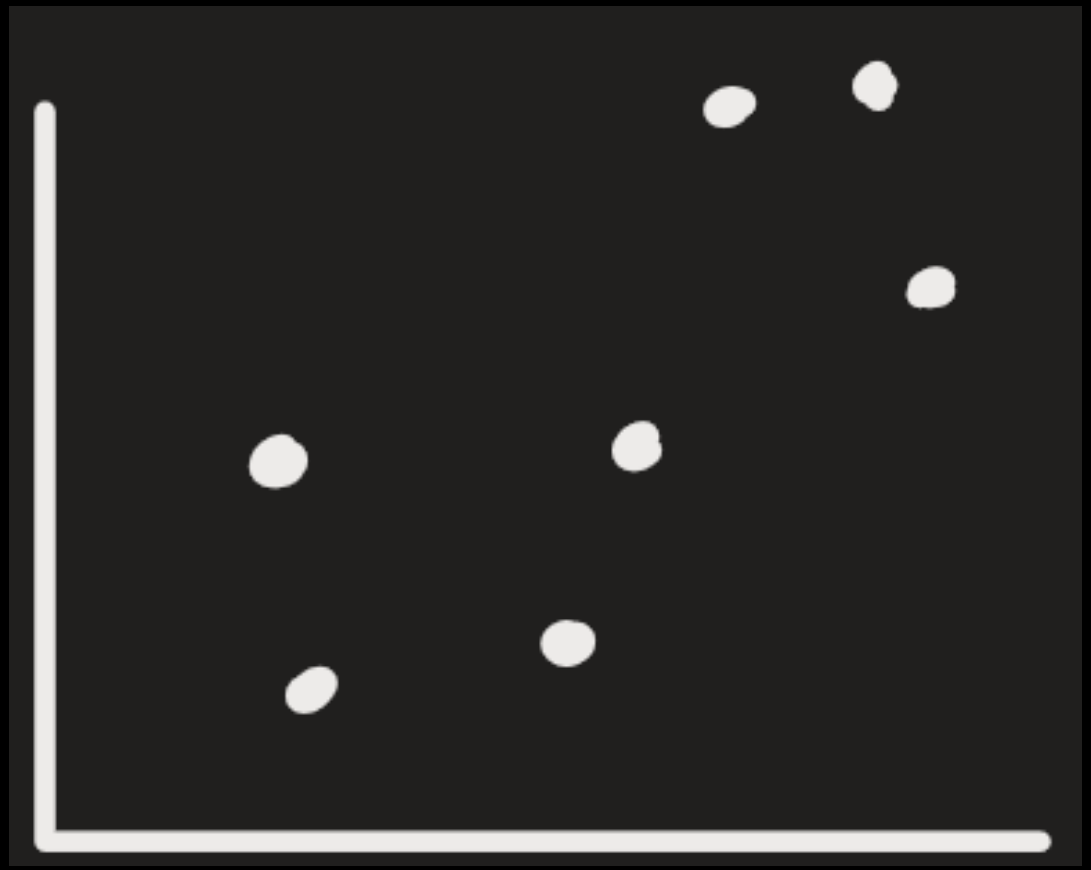
age	Income in K	Debt in K
21	30	10
42	60	0
56	50	80
24	0	0



$$d(P_1, P_2) = \sqrt{(21-56)^2 + (30-50)^2 + (10-80)^2}$$

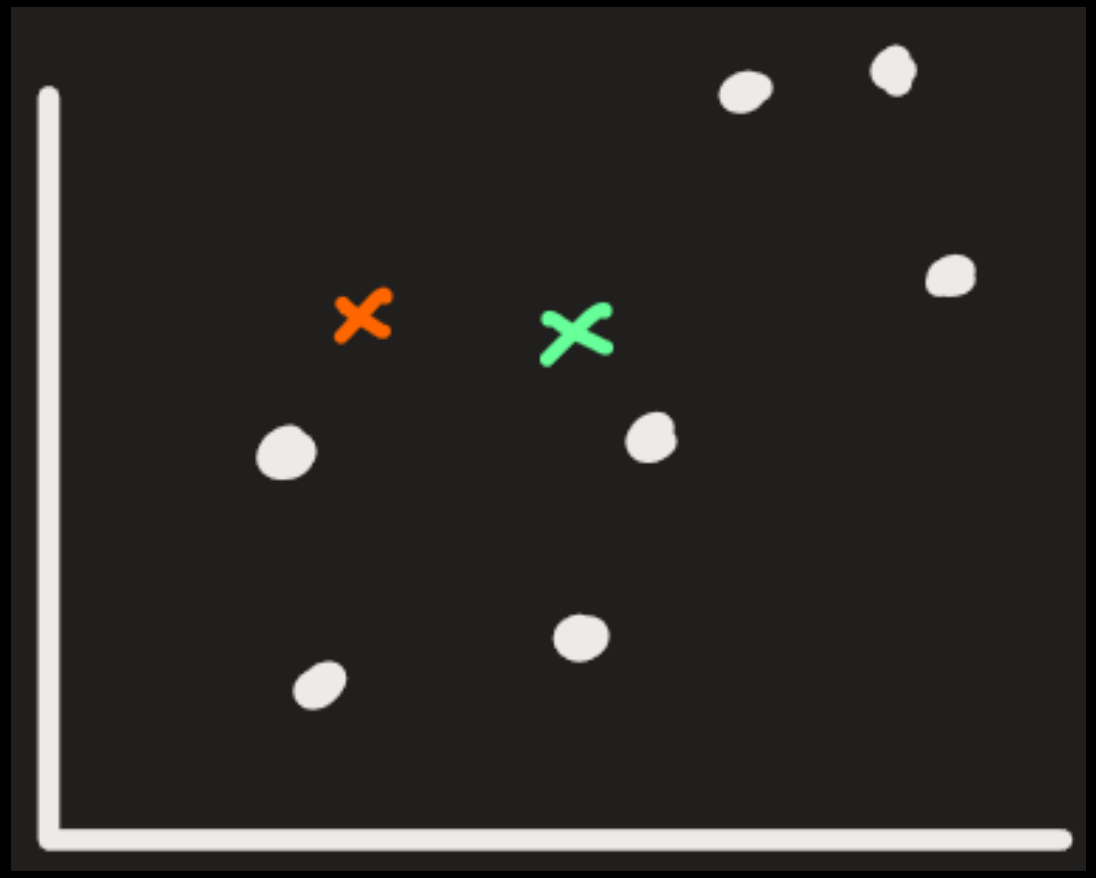
K-Means

Step1: Suppose you want to find k=2 clusters. You start with all points.



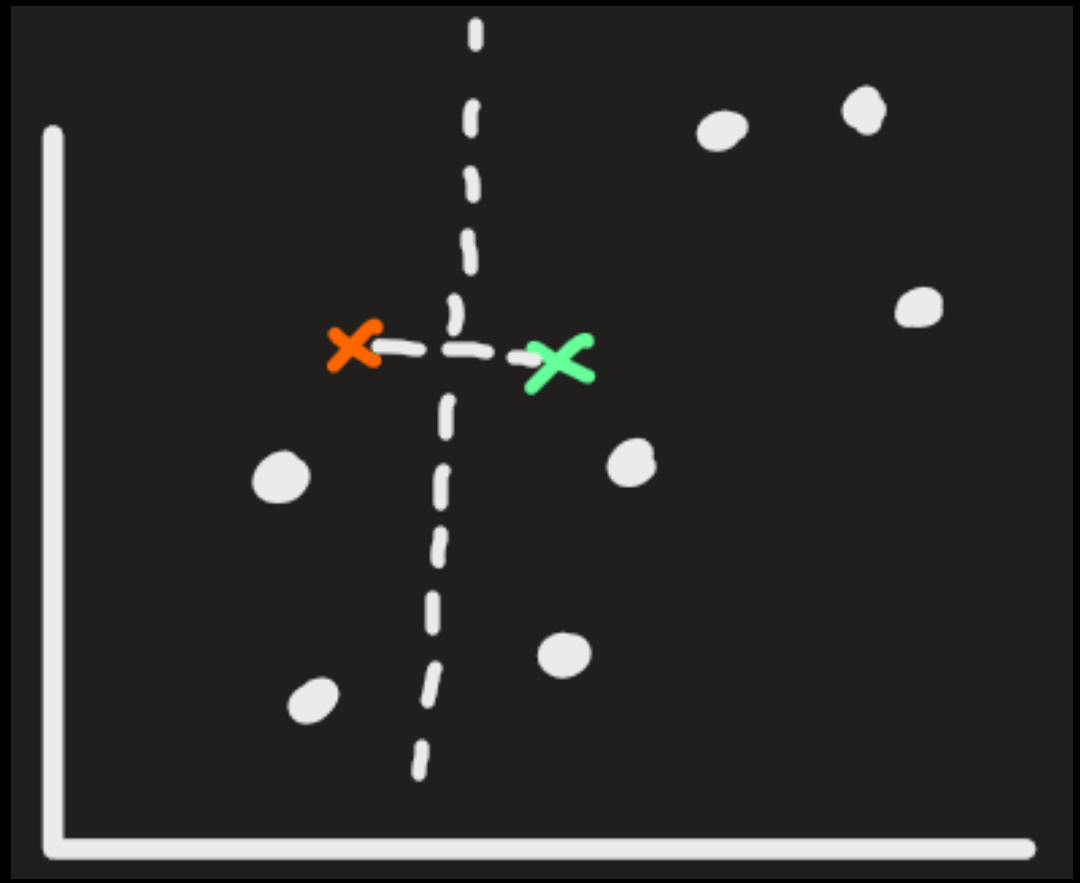
K-Means

Step2: Randomly choose k=2 centroid



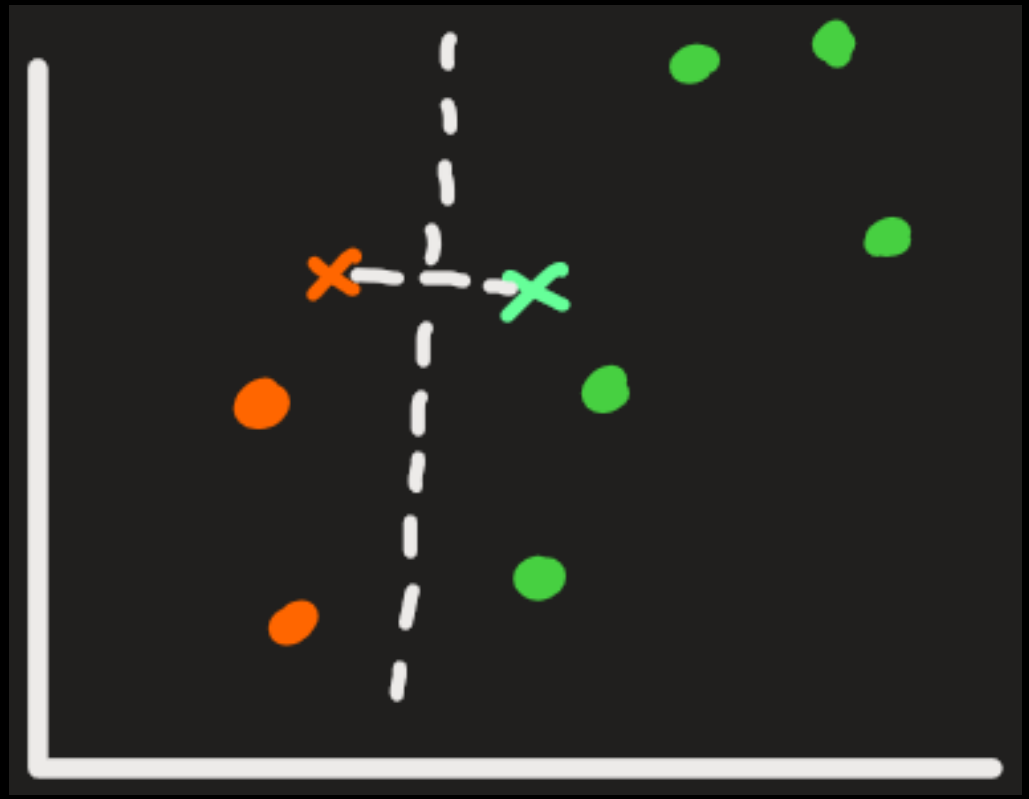
K-Means

Step3: Draw a half line between 2 centroids.



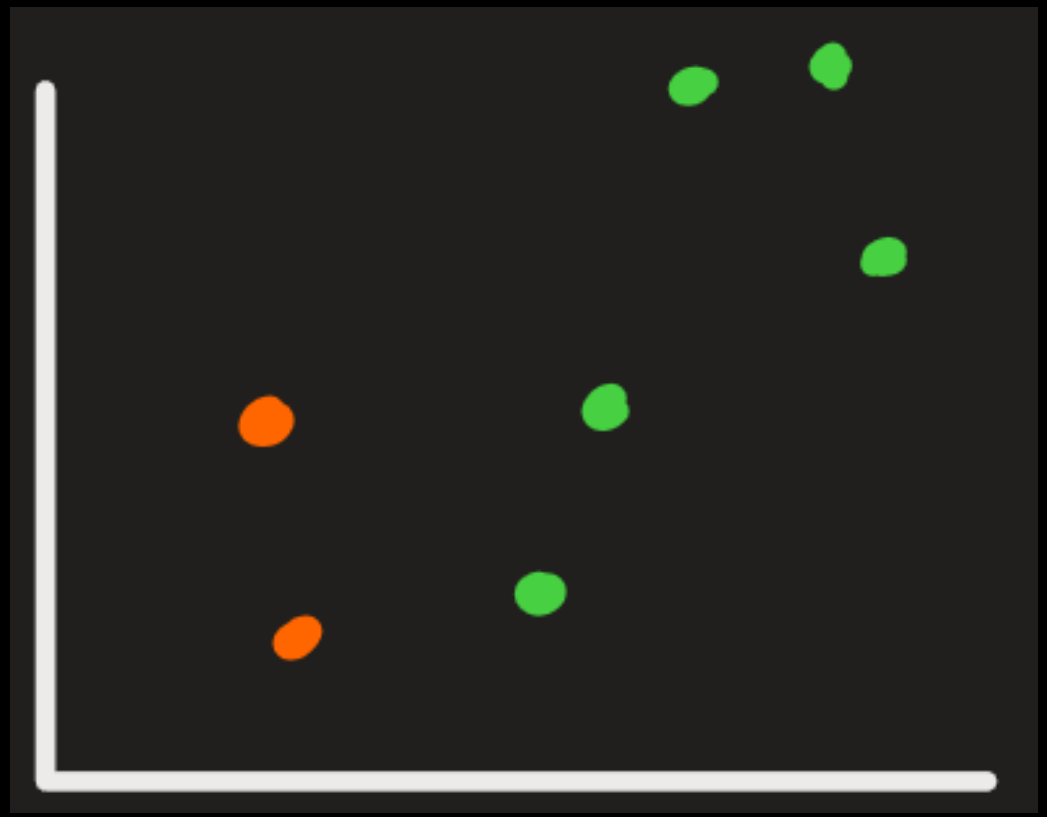
K-Means

Step4: Assign points to clusters that are closest to centroids. This can be easily done by assigning points on LHS of half-line to red cluster and RHS to green cluster



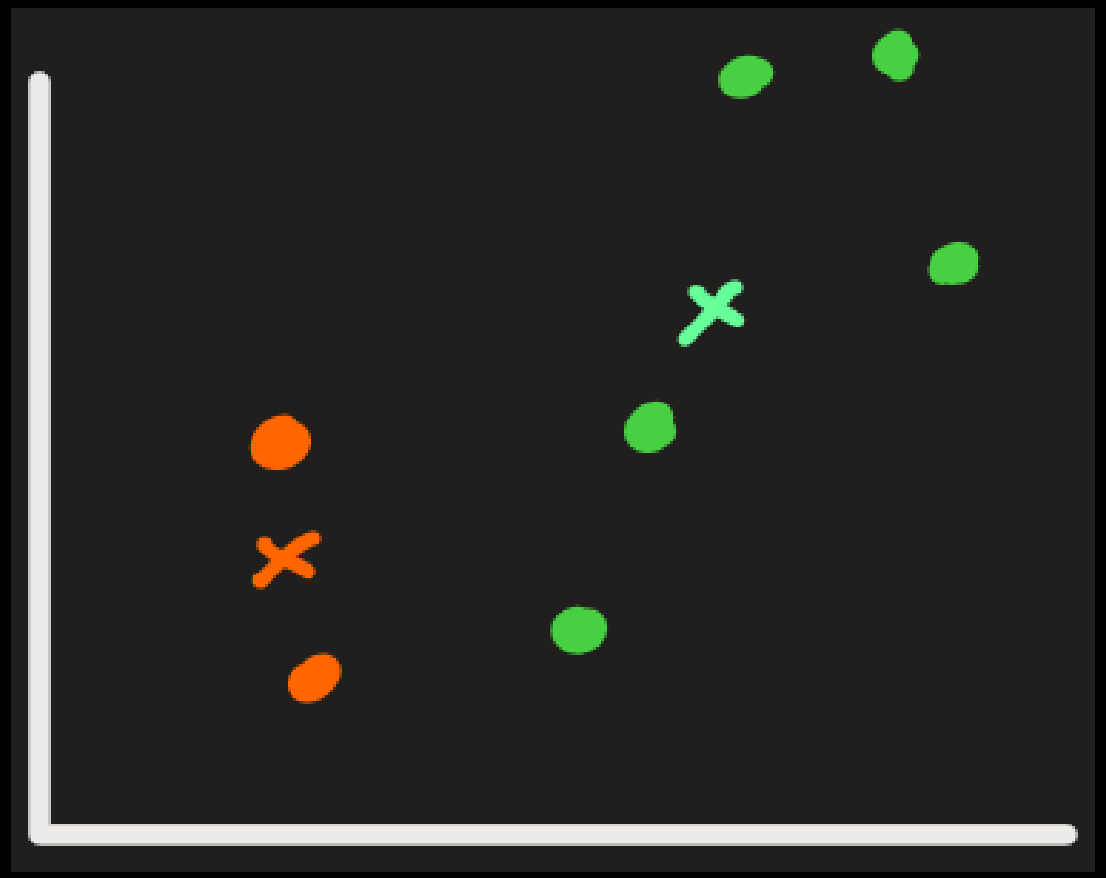
K-Means

Step5: Now we have 2 clusters



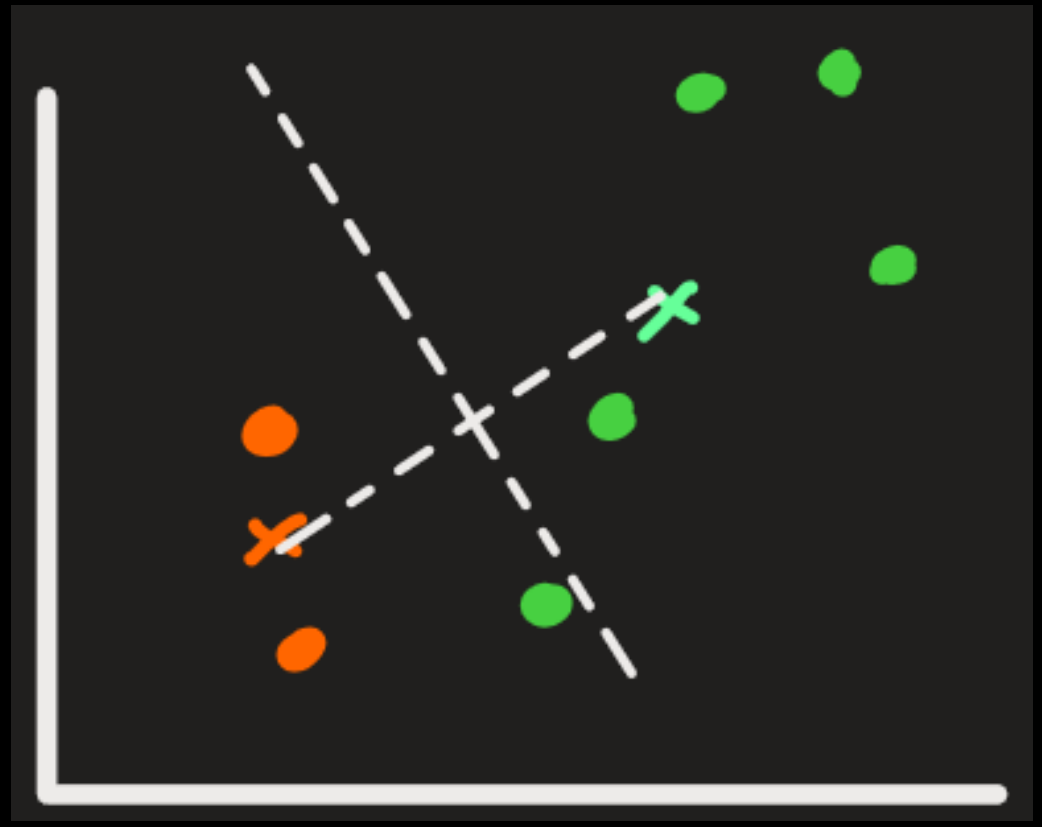
K-Means

Step6: Find the centroids of these 2 new clusters.



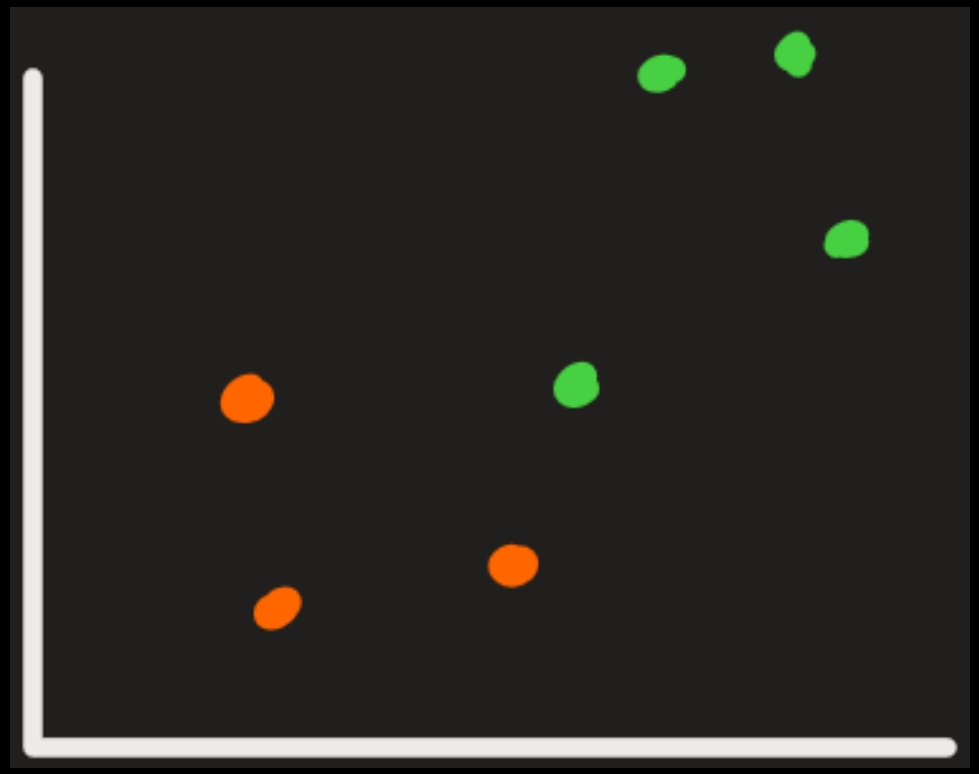
K-Means

Step7: Now repeat the process: Draw a half line between 2 centroids.
Re-assign the points to each nearest closest cluster-centroid



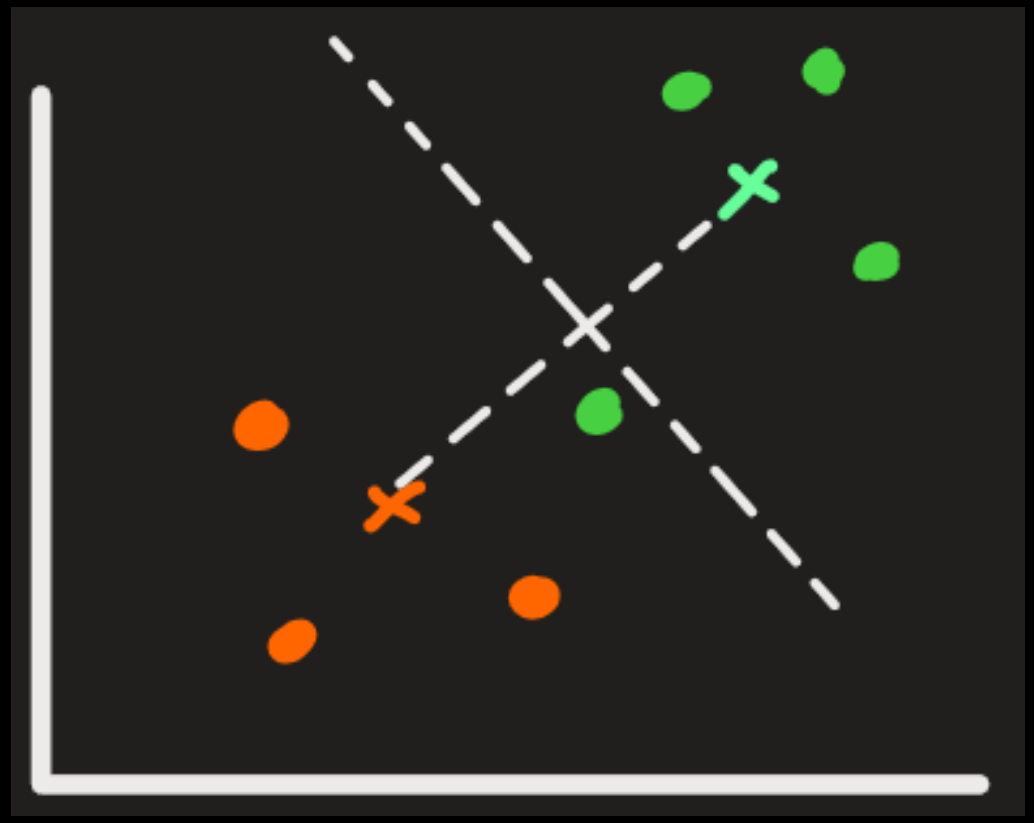
K-Means

Step8: Now you have new cluster



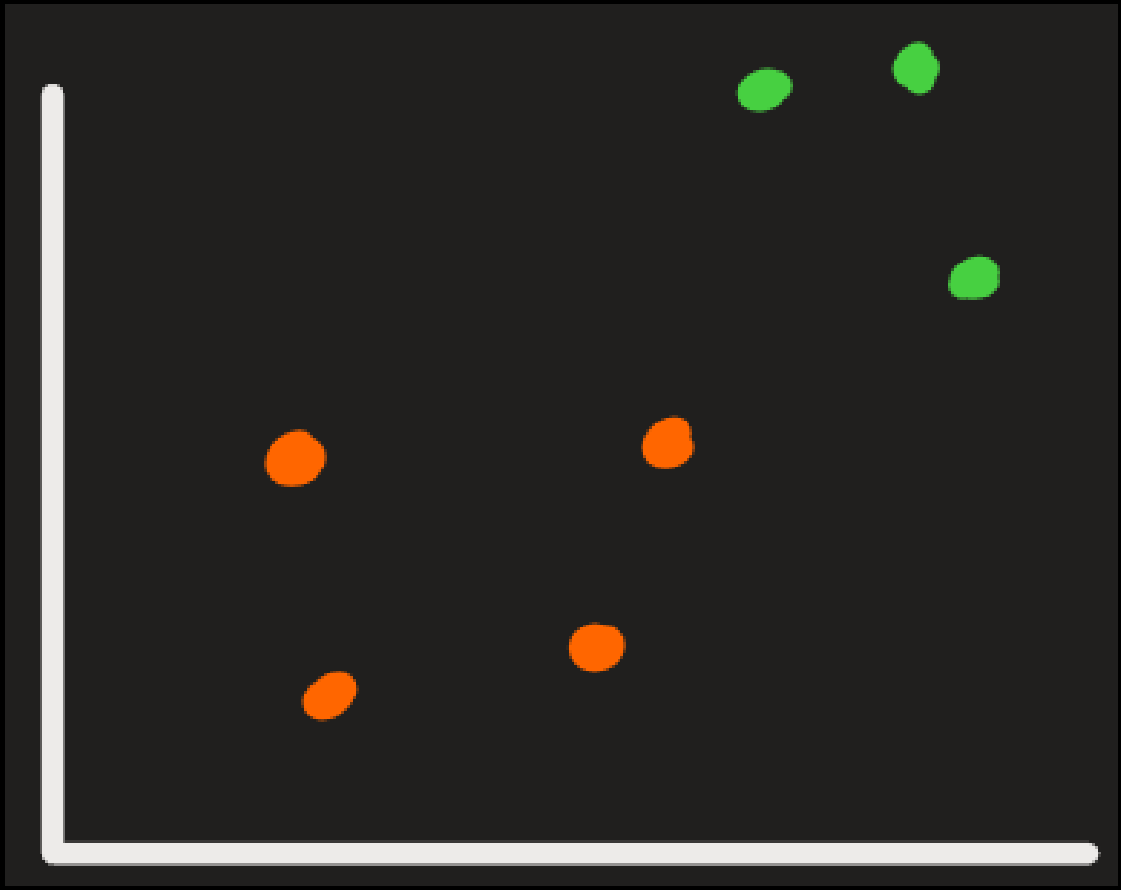
K-Means

Step9: Now repeat the process: Find the centroid →Draw a half line between 2 centroids. Re-assign the points to each nearest closest cluster-centroid



K-Means

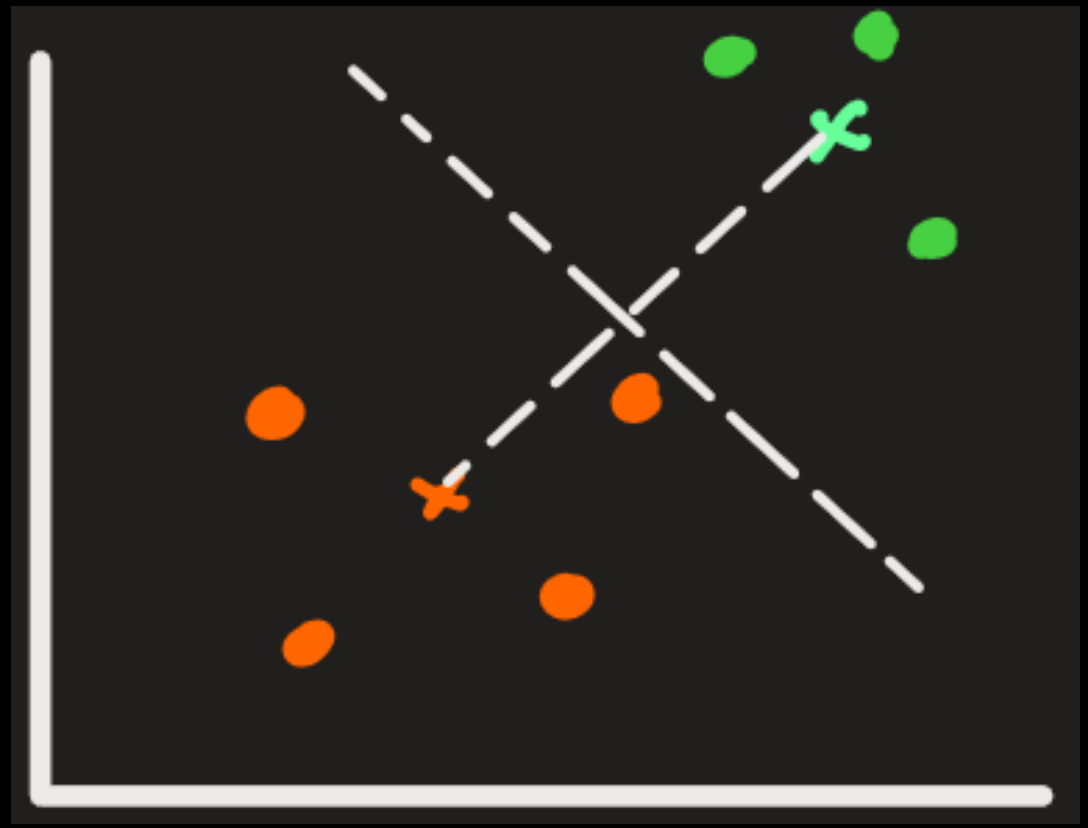
Step10: Now you have new cluster.



K-Means

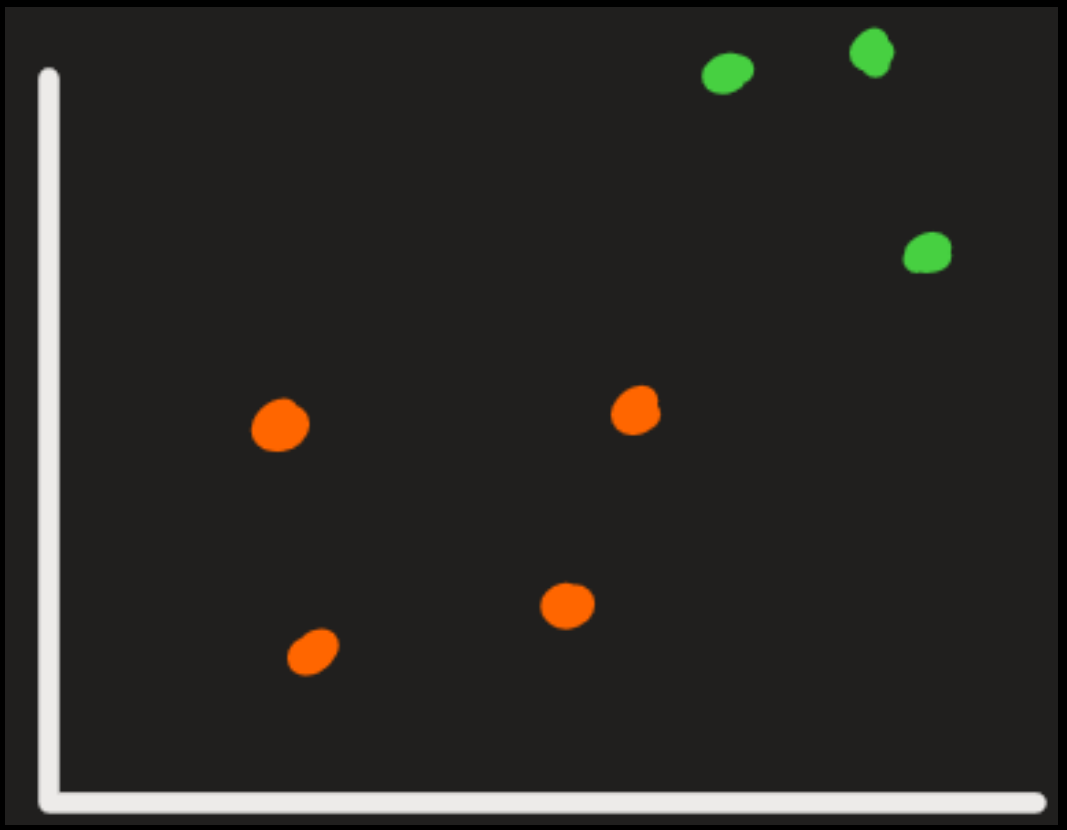
Step11: Find centroid -> Reassign points based on closest distance to centroid.

All points are closest to their centroids → No change → STOP



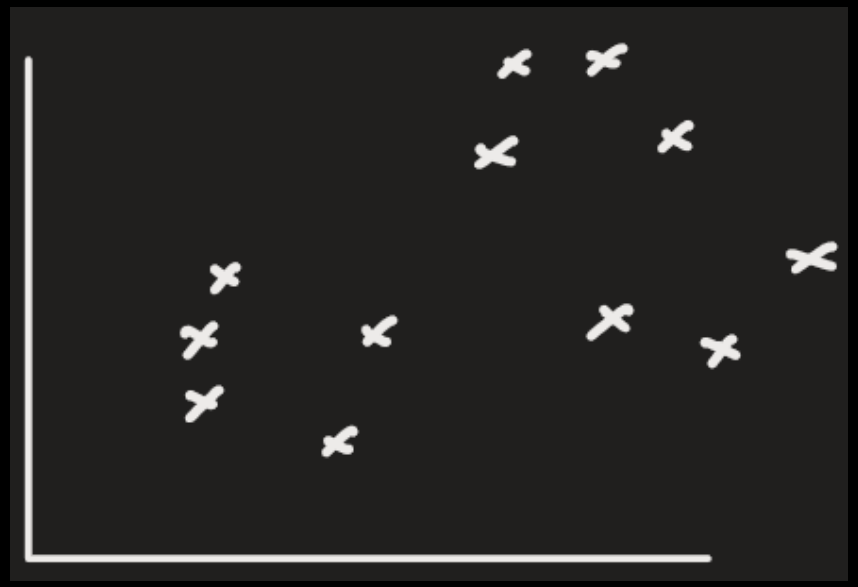
K-Means

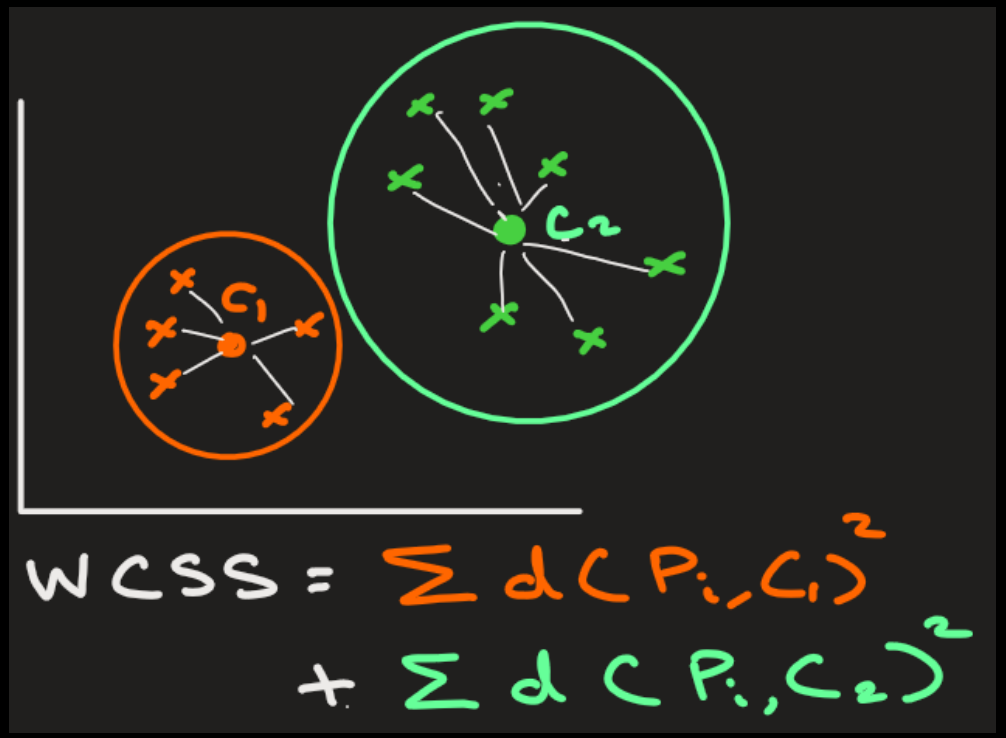
FINAL 2 clusters.



How to find K ? Elbow method

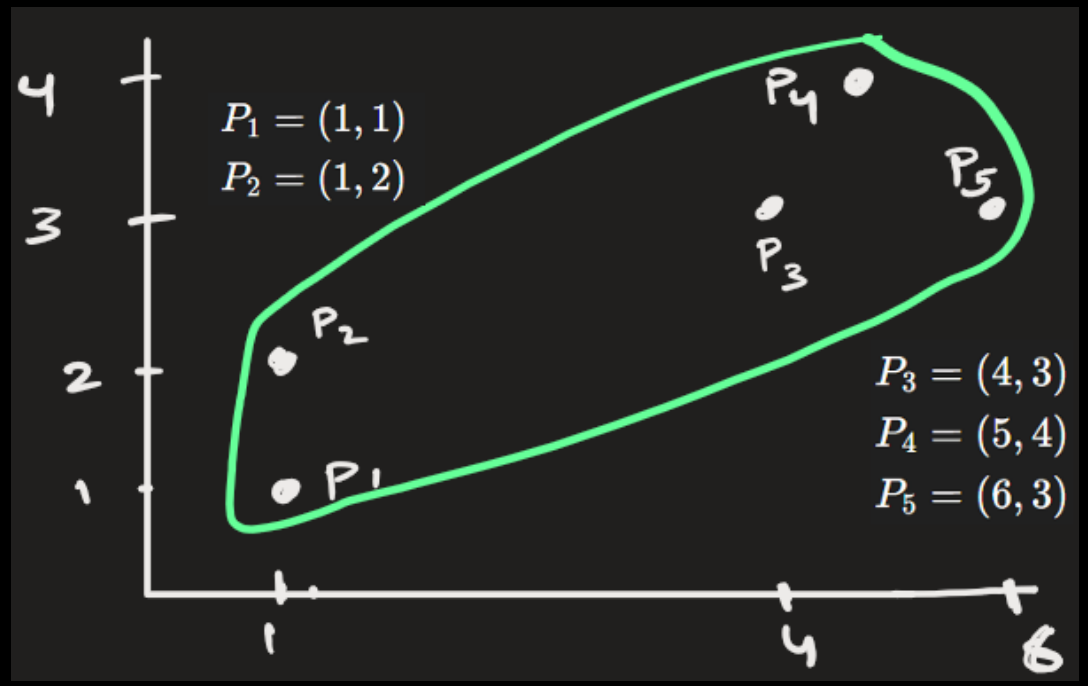
Within-cluster sum of squares (WCSS): It's the sum of the squared distances between each data point and its cluster's centroid. It’s **AKA SSE**(Sum of Squared Error).





How to find K ? Elbow method

Example: WCSS for K = 1 cluster



Centroid:

$$C = \left(\frac{1+1+4+5+6}{5}, \frac{1+2+3+4+3}{5}\right)$$

$$= (3.4,\ 2.6)$$

Formula:

$$d^2 = (x_i - x_c)^2 + (y_i - y_c)^2$$

Point P1 = (1,1)

$$(1-3.4)^2 + (1-2.6)^2 = 5.76 + 2.56 = 8.32$$

Point P2 = (1,2)

$$(1-3.4)^2 + (2-2.6)^2 = 5.76 + 0.36 = 6.12$$

Point P3 = (4,3)

$$(4-3.4)^2 + (3-2.6)^2 = 0.36 + 0.16 = 0.52$$

Point P4 = (5,4)

$$(5-3.4)^2 + (4-2.6)^2 = 2.56 + 1.96 = 4.52$$

Point P5 = (6,3)

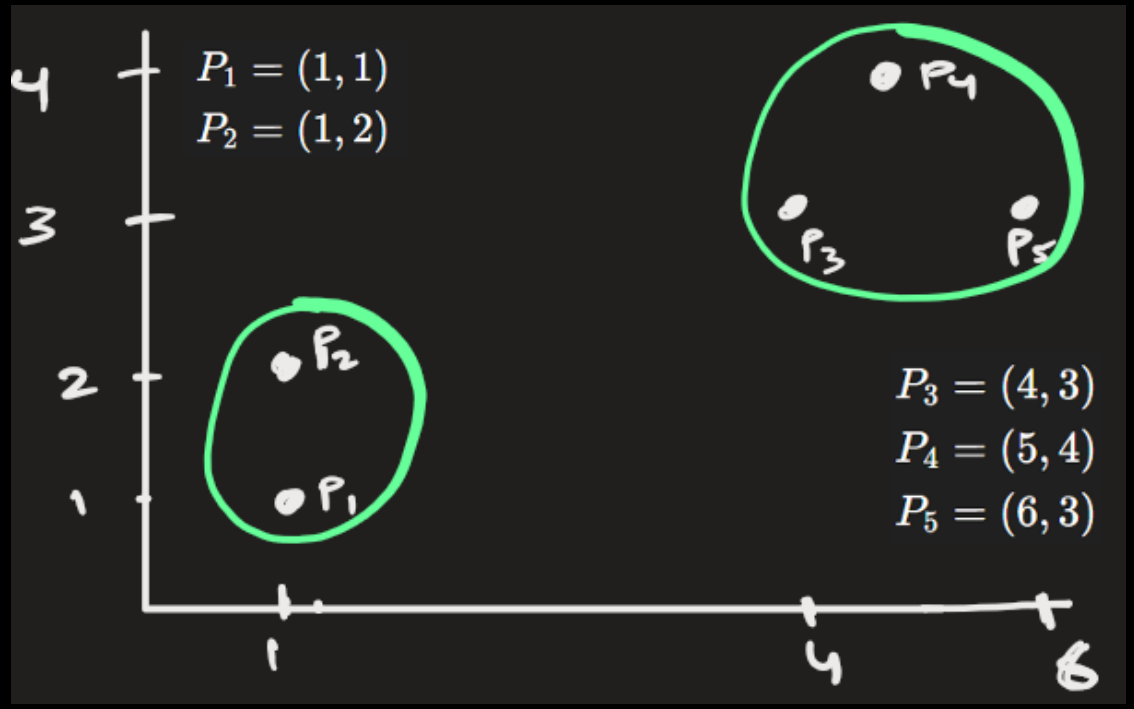
$$(6-3.4)^2 + (3-2.6)^2 = 6.76 + 0.16 = 6.92$$

$$WCSS_1 = 8.32 + 6.12 + 0.52 + 4.52 + 6.92$$

$$\boxed{WCSS_1 = 26.40}$$

How to find K ? Elbow method

Example: WCSS for K = 2 cluster



Centroid:

$$C_A = \left(\frac{1+1}{2}, \frac{1+2}{2}\right) = (1,\ 1.5)$$

$$C_B = \left(\frac{4+5+6}{3}, \frac{3+4+3}{3}\right) = (5,\ 3.33)$$

P1 = (1,1)

$$(1-1)^2 + (1-1.5)^2 = 0 + 0.25 = 0.25$$

P2 = (1,2)

$$(1-1)^2 + (2-1.5)^2 = 0 + 0.25 = 0.25$$

WCSS for Cluster A

$$WCSS_A = 0.25 + 0.25 = 0.50$$

P3 = (4,3)

$$(4-5)^2 + (3-3.33)^2 = 1 + 0.11 = 1.11$$

P4 = (5,4)

$$(5-5)^2 + (4-3.33)^2 = 0 + 0.44 = 0.44$$

P5 = (6,3)

$$(6-5)^2 + (3-3.33)^2 = 1 + 0.11 = 1.11$$

WCSS for Cluster B

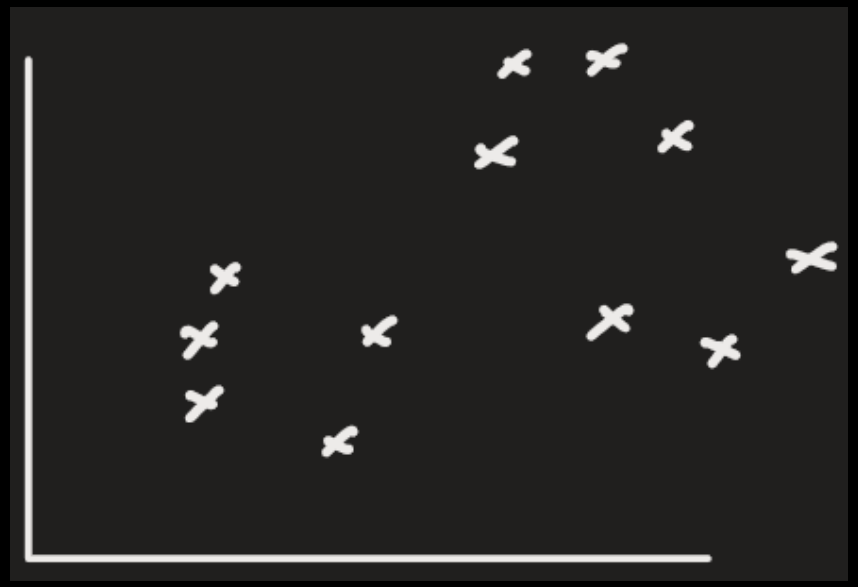
$$WCSS_B = 1.11 + 0.44 + 1.11 = 2.66$$

$$WCSS_2 = WCSS_A + WCSS_B$$

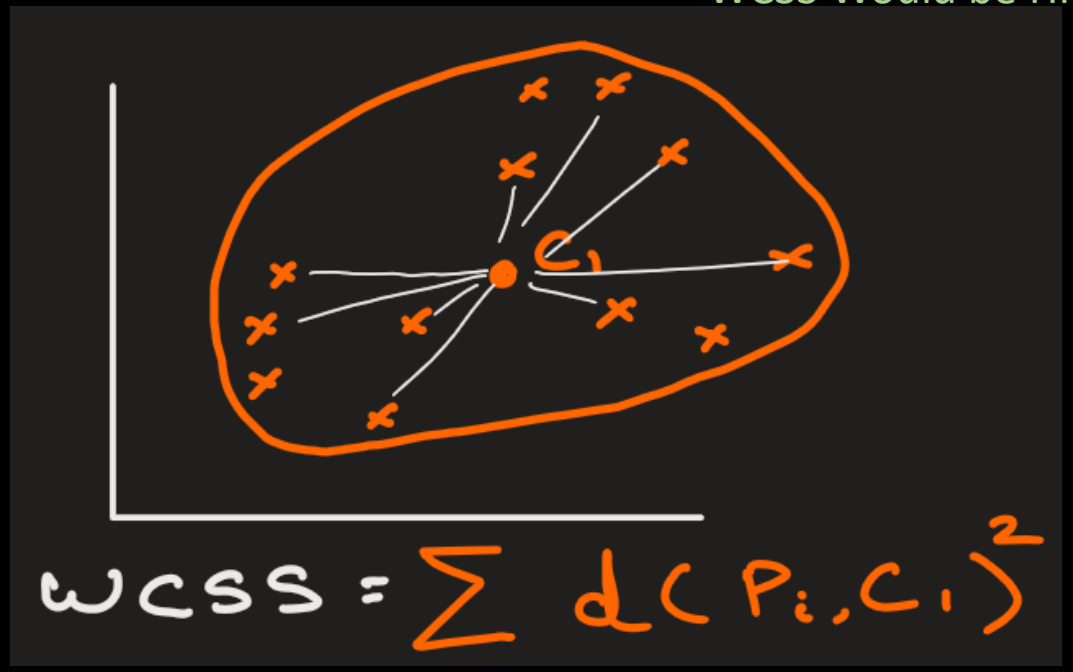
$$WCSS_2 = 0.50 + 2.66 = \boxed{3.16}$$

How to find K ? Elbow method

WCSS decreases as K increases.



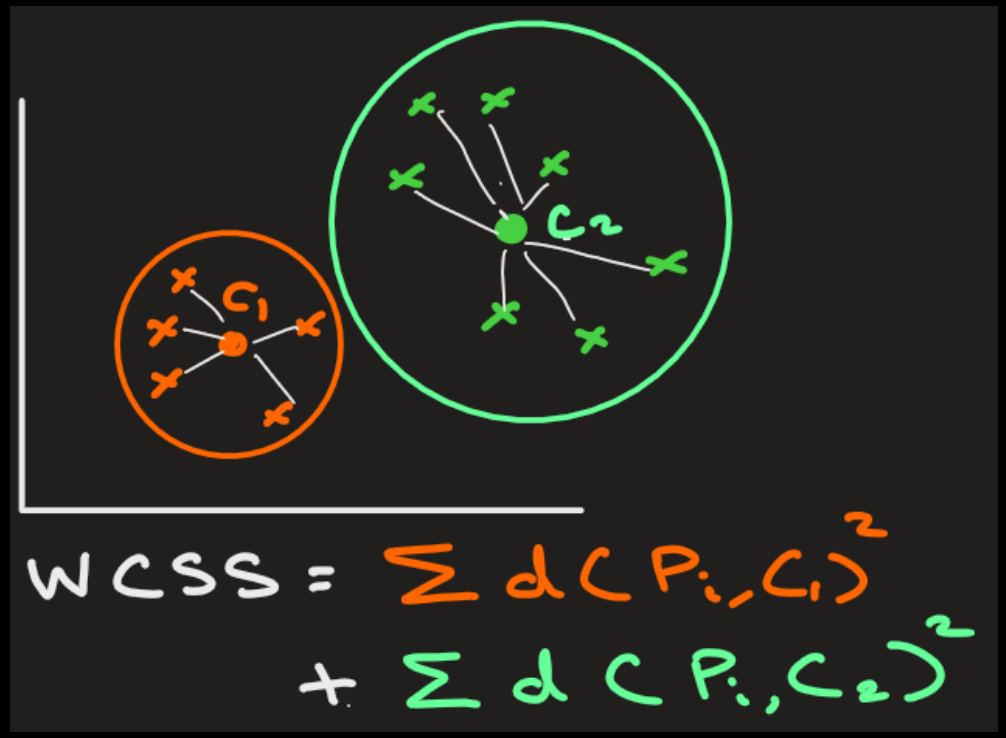
K=1.
WCSS Would be HIGH



How to find K ? Elbow method

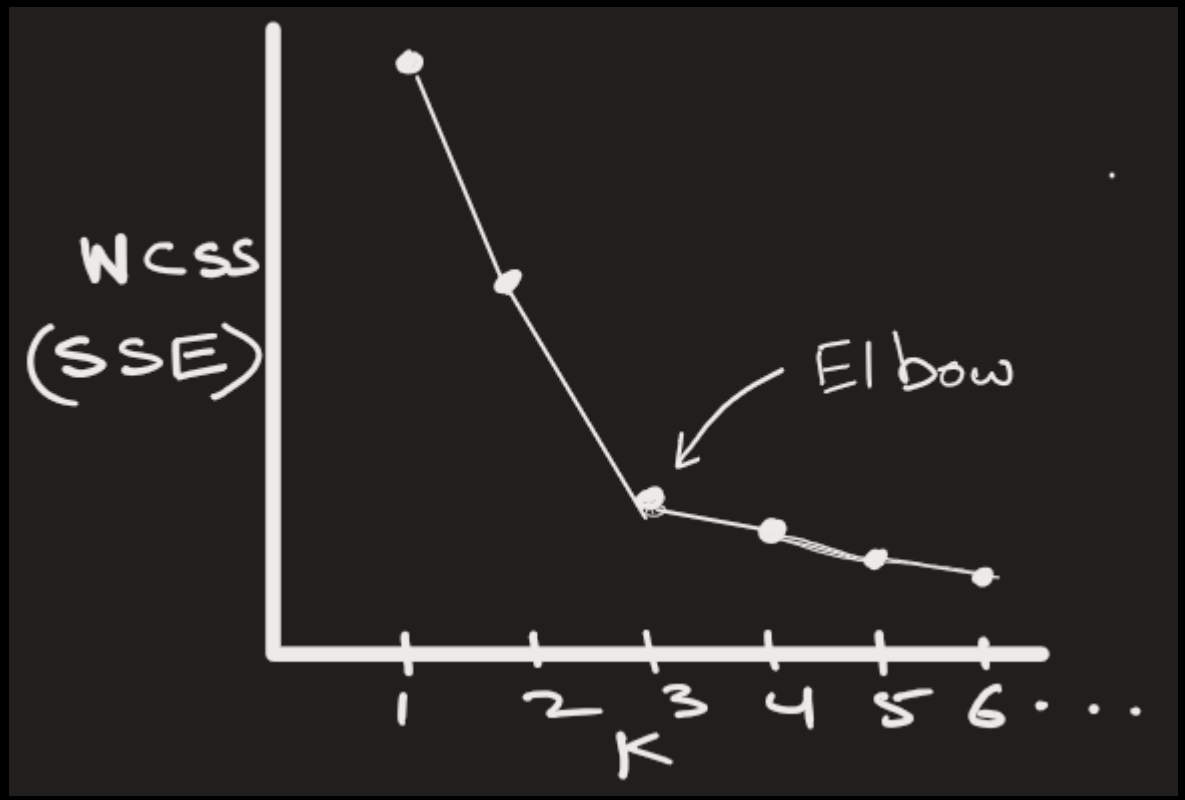
K=2

WCSS decreases as K increases



K=3.

How to find K ? Elbow method



If we plot WCSS vs K, we see that there is
- Big drop in WCSS as we go from k=1 to k=2
- Big drop in WCSS as we go from k=2 to k=3
- Small drop in WCSS as we go from k=3 to k=4
- Drop becomes smaller as we k increases.

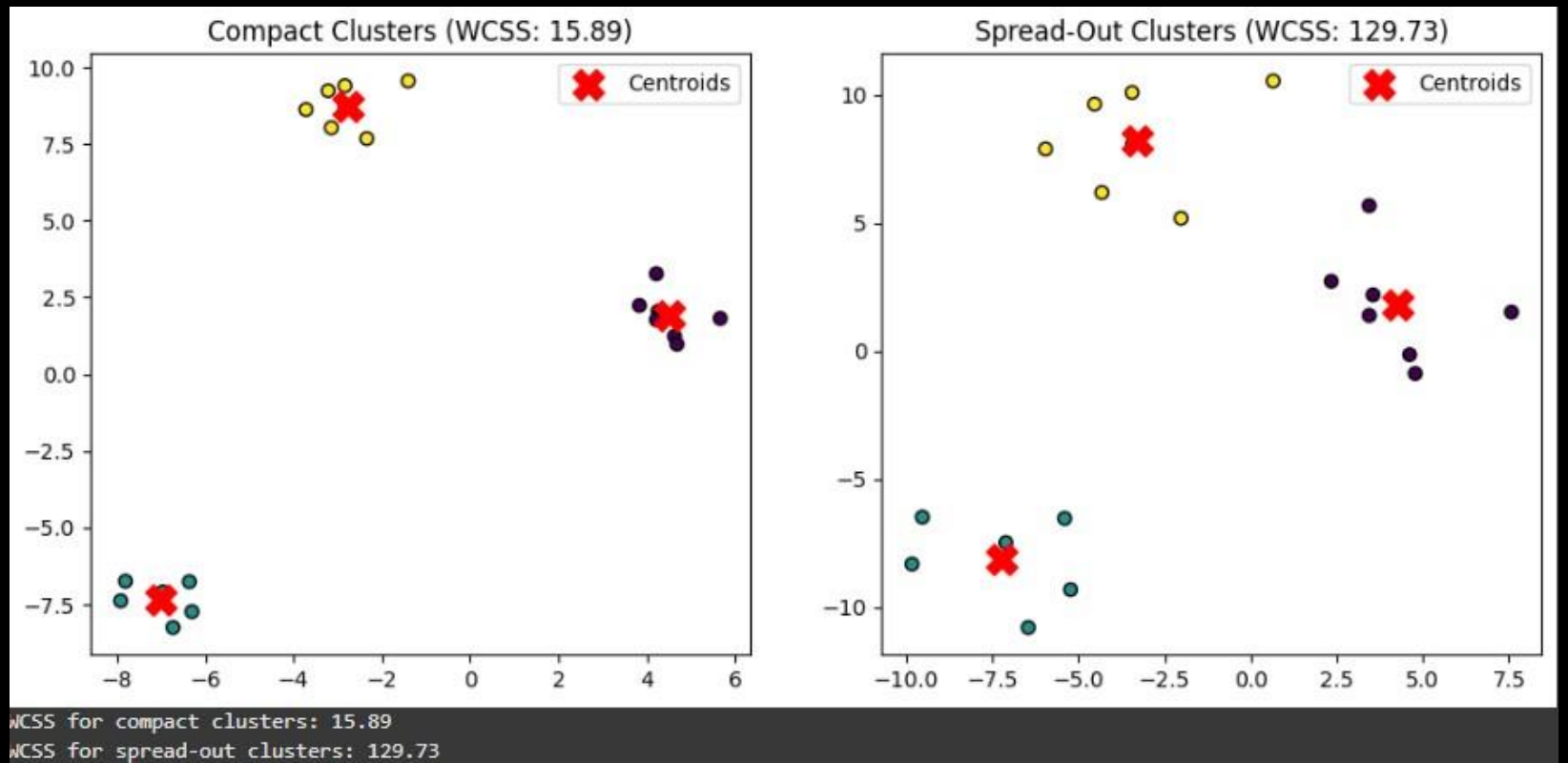
So, k=3 is a threshold that gives us a value of **how many clusters to include in our dataset.**

The point with k=3 is at the elbow.

Why is WCSS important?

1. WCSS is a measure of how compact a cluster is. **A cluster with a small WCSS is more compact and better** than a cluster with a large WCSS.

2. WCSS is used to find the optimal number of clusters. **The value of K that has the smallest WCSS is the best value**.



Always scale/standardize your data in K Means, otherwise features with big numbers would influence the K-Mean clusters.

Case1: Not standardized/scaled:

age	Income
21	30,000
56	50,000

$$d(x,y) = \sqrt{(21-56)^2 + (30{,}000-50{,}000)^2}$$

small (age feature is ignored)

big (income feature dominates)

Case2: Standardized/scaled to be between 10 - 100:

age	Income
21	30
56	50

$$d(x,y) = \sqrt{(21-56)^2 + (30-50)^2}$$

age

income

same level of influence

Scikit-Learn Code

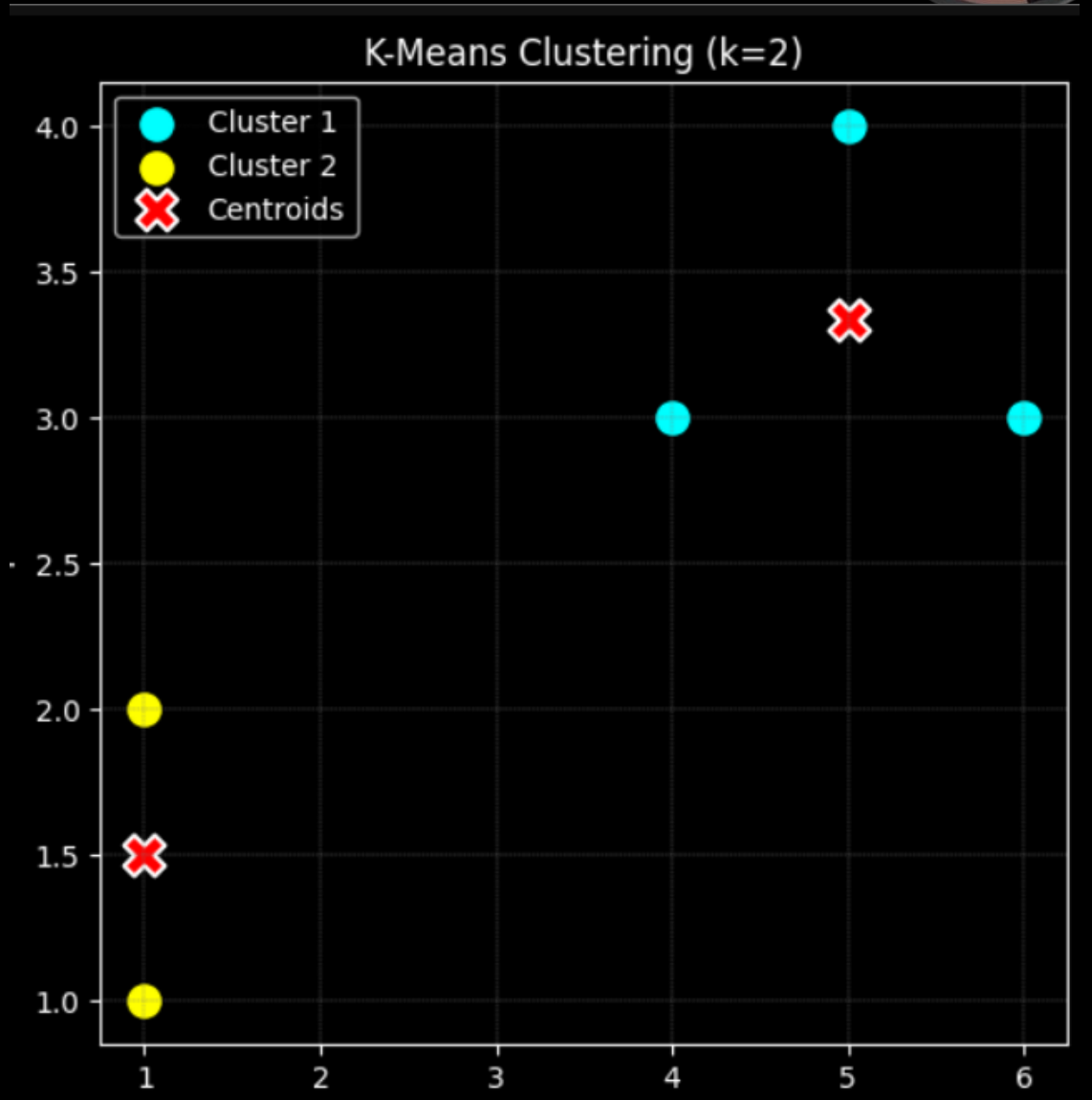
```python
from sklearn.cluster import KMeans
import numpy as np

X = np.array([
    [1, 1],
    [1, 2],
    [4, 3],
    [5, 4],
    [6, 3]
])

kmeans = KMeans(n_clusters=2)
kmeans.fit(X)

print("Cluster labels:", kmeans.labels_)
print("Cluster centroids:\n", kmeans.cluster_centers_)
print("WCSS for K=2:", kmeans.inertia_)
```

```
Cluster labels: [1 1 0 0 0]
Cluster centroids:
 [[5.         3.33333333]
 [1.         1.5       ]]
WCSS for K=2: 3.166666666666667
```



EXTRA

Fhdsklf
Fjdsklf
Fjskldf

K-mean Algorithm:

This step dictates the shape/accuracy of clusters

- Step 1) Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
- Step 2) Iterate until the cluster assignments stop changing:

  - (a) For each of the K clusters, compute the cluster centroid. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster

  - (b) Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance)

In other words, we minimize following:

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}.$$

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

W function is the measure of variation within the cluster

2. Theory: Distance used in clustering(p1)

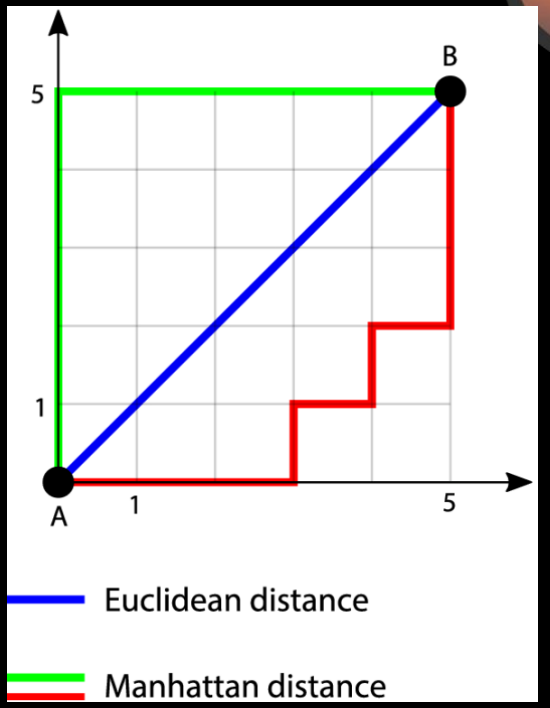
Some common distance measures used in clustering include:

1. Manhattan distance The sum of the absolute differences between the corresponding coordinates of two points. It's also known as the "city block" distance.

In the example shown:

Man. Distance for red path = 3+1+1+1+1+3=10

Man. Distance for green path = 5+5=10



2. Euclidean distance The shortest distance between two points in a Euclidean space. It's calculated using the Pythagorean theorem.

In the example shown, the Euclidean distance = sqrt(25+25)

2. Theory: Distance used in clustering(p2)

3. Minkowski distance; A generalized form of the Euclidean and Manhattan distances. It's also known as the Lp norm distance.

The L^p norm can be mathematically written as:

$$||x_p|| = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \quad where \;\; p > 0$$

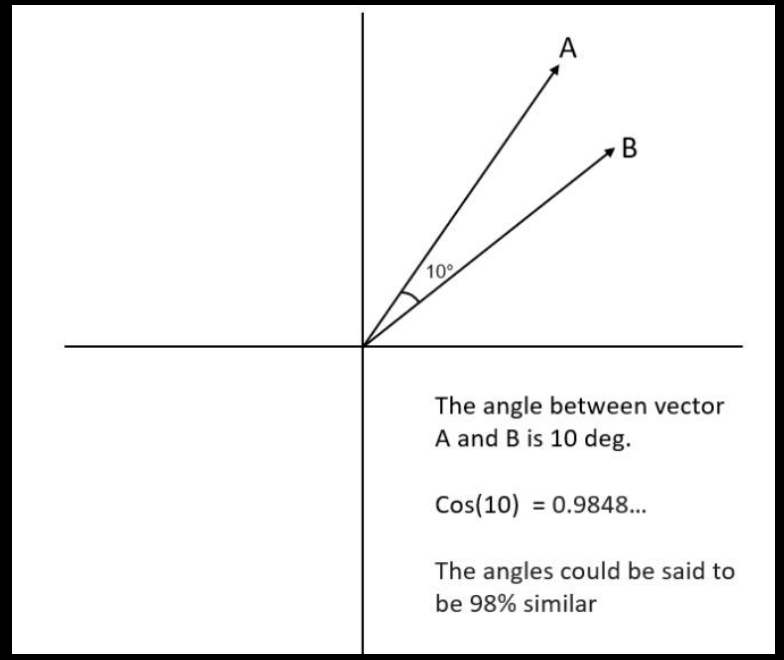
2. Theory: Distance used in clustering(p3)

4. Cosine similarity: A measure of the angle between two vectors. It's useful for clustering high-dimensional sparse data, like text or images.

Cosine similarity is a value that ranges from -1 to 1

- +1 indicates the vectors are identical and perfectly aligned (with no angle between them)
- 0 indicates the vectors are orthogonal (90 degrees to each other) with no match
- -1 indicates completely opposite vectors (with a 180 degree angle between them).

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$



Example next page

2. Theory: Distance used in clustering(p4)

4. Cosine similarity: Example

1. Document Similarity

A scenario that involves the requirement of identifying the similarity between pairs of a document is a good use case for the utilization of cosine similarity as a quantification of the measurement of similarity between two objects.

To find the quantification of the similarity between two documents, you need to convert the words or phrases within the document or sentence into a vectorized form of representation.

Here are the two documents we will compare:

- Document 1: "Deep Learning can be hard"
- Document 2: “Machine Learning can be simple“

Step 1: Obtain a Vectorized Representation of the Texts

Lets say the vectors are represented as [Deep, Machine, Learning, can, be, hard, easy]

- Document 1: [1, 0, 1, 1, 1, 1, 0] - let’s refer to this as A.
- Document 2: [0, 1, 1, 1, 1, 0, 1] - let’s refer to this as B.

Step 2: Find the Cosine Similarity

cosine similarity (CS) = (A . B) / (||A|| ||B||)

•Calculate the dot product between A and B: $(1\times0) + (0\times1) + (1\times1) + (1\times1) + (1\times1) + (1\times0) + (0\times1) = 3$

•Calculate the magnitude of the vector A: $\sqrt{(1^2 + 0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2)} = 2.2360679775$.

•Calculate the magnitude of the vector B: $\sqrt{(0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 1^2)} = 2.2360679775$.

•Calculate the cosine similarity: (3) / (2.2360679775*2.2360679775) = 0.71 (71 percent similarity between the sentences in both document).

2. Pose Matching (optional...TODO: Understand this)

Pose matching involves comparing the poses containing key points of joint locations.

Pose estimation is a computer vision task, and it's typically solved using deep learning approaches such as convolutional pose machines, stacked hourglasses and PoseNet, etc.

Pose estimation is the process where the position and orientation of the vital body parts and joints of a body are derived from an image or sequence of images.

In a scenario where there is a requirement to quantify the similarity between two poses in Image A and Image B, here is the process that would be taken:

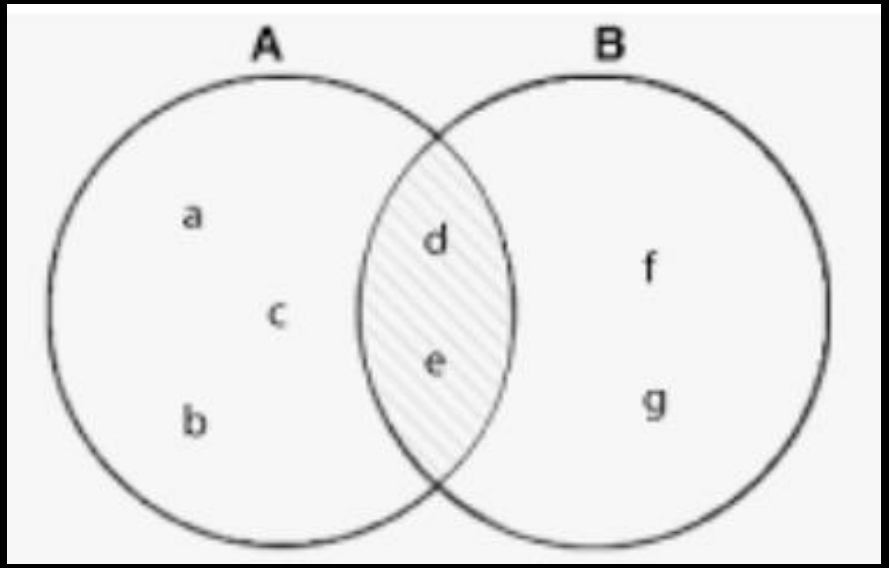
1. Identify the pose information and derive the location key points (joints) in Image A.
2. Identify the x,y location of all the respective joints required for comparison. Deep Learning solution to pose estimation usually provides information on the location of the joints within a particular pose, along with an estimation confidence score.
3. Repeat steps one and two for Image B.
4. Place all x,y positions of Image A in a vector.
5. Place all x,y positions of Image B in a vector.
6. Ensure the order of the x,y positions of each joint is the same in both vectors.
7. Perform cosine similarity using both vectors to obtain a number between 0 and 1

2. Theory: Distance used in clustering(p3)

5. Jaccard Index: Ideal for **categorical data**.. It calculates the ratio of the number of features shared by two data points to the total number of features.

Jaccard similarity measures the similarity between two sets by calculating the ratio of the size of their intersection to the size of their union; a higher ratio indicates greater similarity. The Jaccard Similarity score is in a range of **0 to 1**. If the two documents are identical, Jaccard Similarity is **1**. The Jaccard similarity score is **0** if there are no common words between two documents.

Here's a simple example:
Set A: {a, b, c, d, e}
Set B: {d, e, f, g}
Intersection (A ∩ B): {d, e} (elements present in both sets)
Union (A ∪ B): {a, b, c, d, e, f, g} (all unique elements from both sets)
Jaccard Similarity: |A ∩ B| / |A ∪ B| = 2 / 7 ≈ 0.28
Therefore, the Jaccard similarity between sets A and B is approximately 0.28, indicating a moderate level of similarity.



2. Theory: Distance used in clustering(p3)

5. Jaccard Index: Example Jaccard index of 2 docs.

$$J(doc_1, doc_2) = \frac{doc_1 \cap doc_2}{doc_1 \cup doc_2}$$

doc_1 = "Data is the new oil of the digital economy"

doc_2 = "Data is a new oil“

Let’s get the set of unique words for each document.

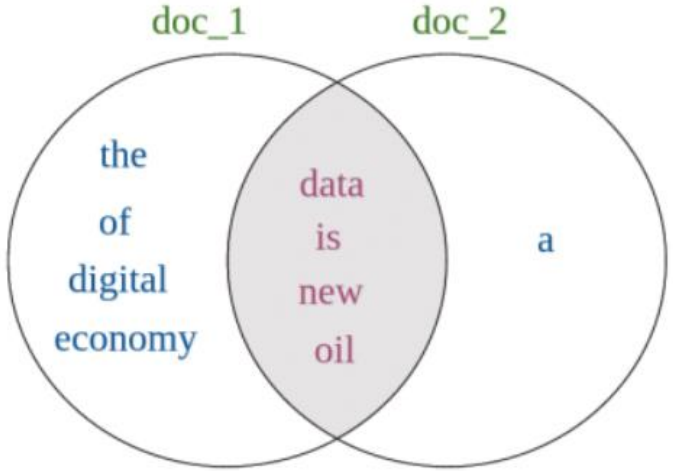
words_doc1 = {'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'}

words_doc2 = {'data', 'is', 'a', 'new', 'oil’}

$$J(doc_1, doc_2) = \frac{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cap \{'data', 'is', 'a', 'new', 'oil'\}}{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cup \{'data', 'is', 'a', 'new', 'oil'\}}$$

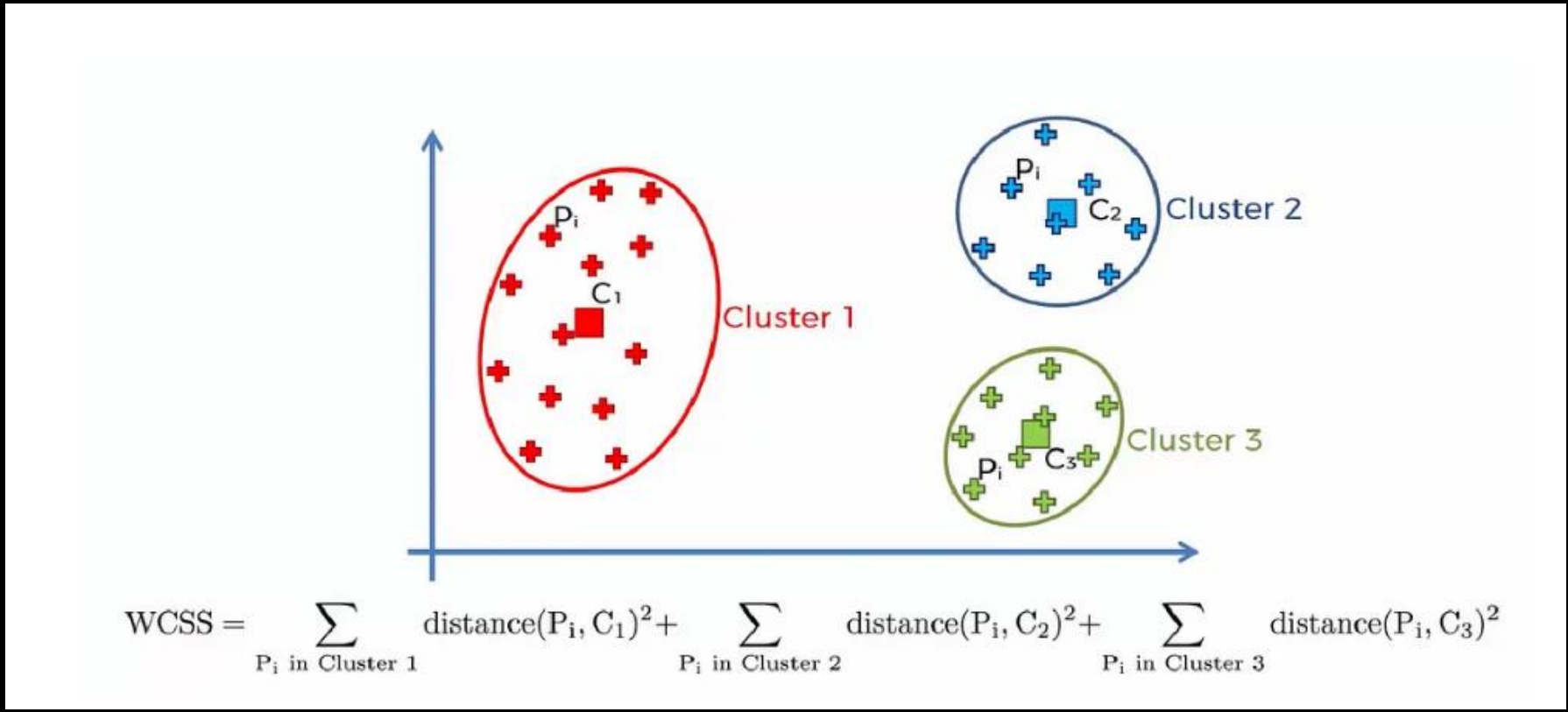
$$= \frac{\{'data', 'is', 'new', 'oil'\}}{\{'data', 'a', 'of', 'is', 'economy', 'the', 'new', 'digital', 'oil'\}}$$

$$= \frac{4}{9} = 0.444$$

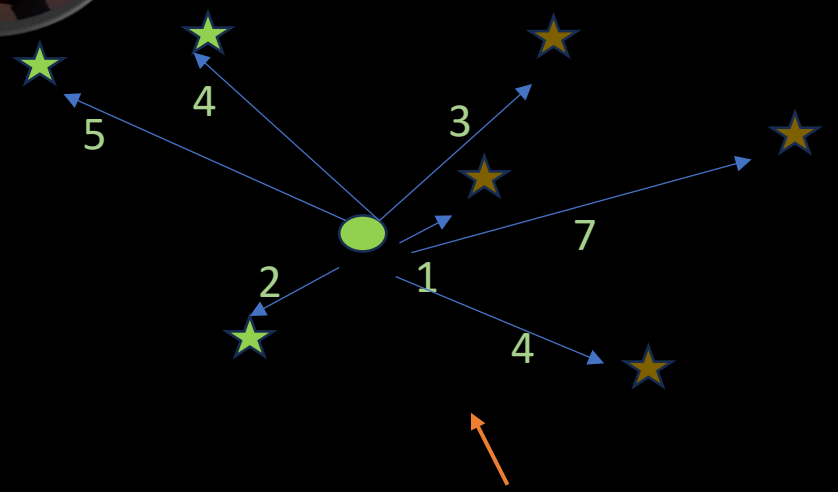


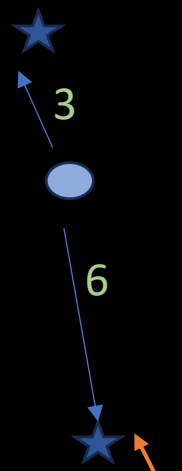
WCSS(p2)

- **Within-cluster sum of squares (WCSS)** is a measurement used in the K-means clustering algorithm to assess the quality of a model.
- It's **AKA SSE**(Sum of Squared Error)
- It's the sum of the squared distances between each data point and its cluster's centroid.



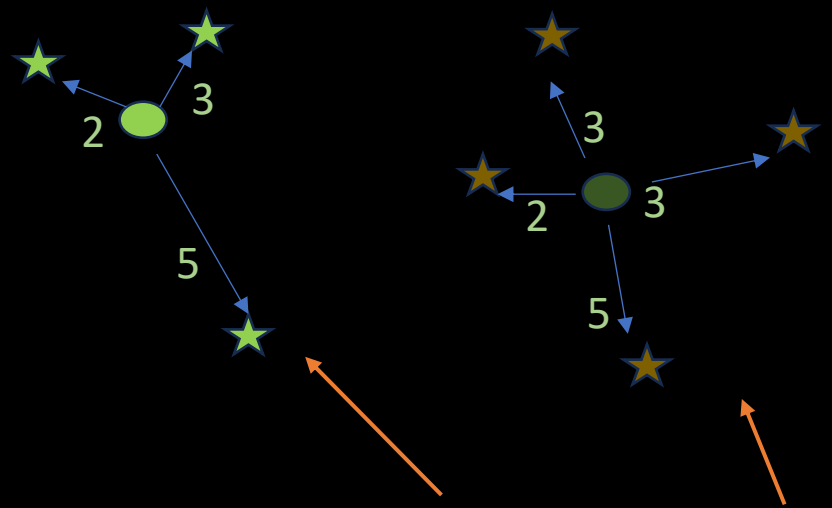
WCSS(p2): How to calculate WCSS for a model with 2 and 3 centroids





K=2 centroids

WCSS = (2^2 + 5^2 + 4^2 + 3^2 + 1^2 + 7^2 + 4^2) + (3^2 + 6^2) = 165



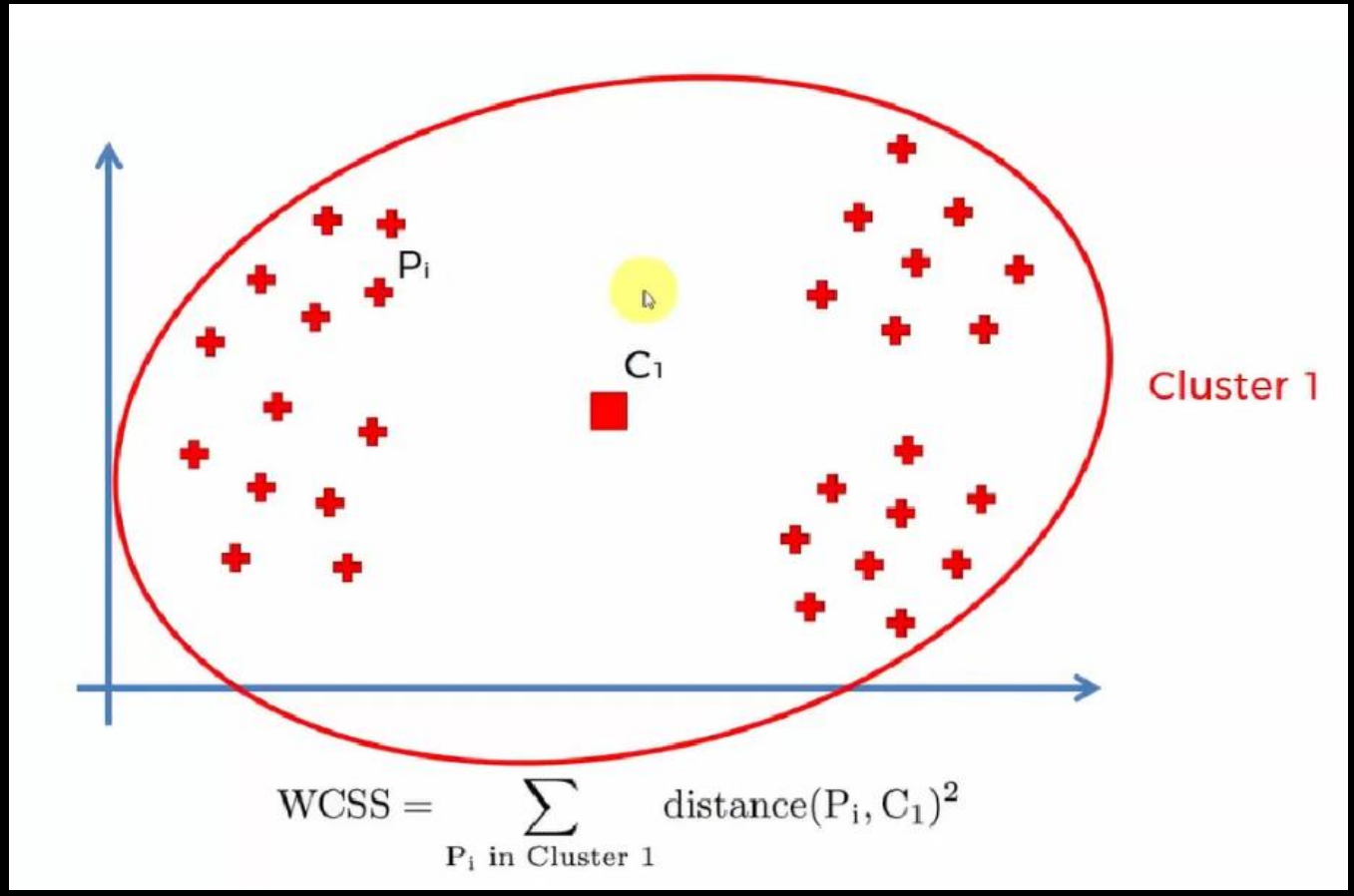
K=3 centroids

WCSS = (2^2 + 3^2 + 5^2) + (2^2 + 3^2 + 3^2 + 5^2) + (3^2 + 6^2) = 130

As K increases, WCSS decreases.

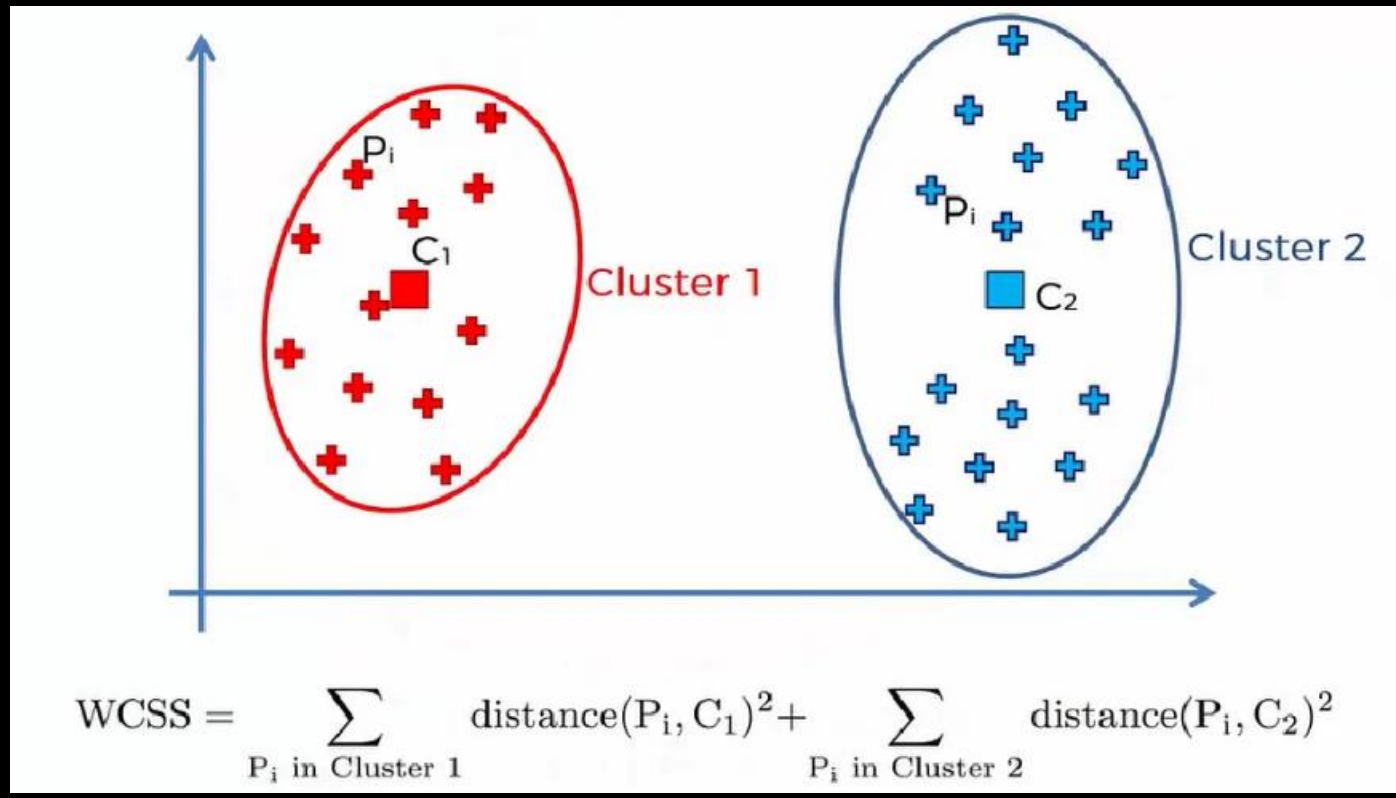
WCSS: How to find right K

Step 1. Let's include **1 centroi**d in our dataset. Here the value of **WCSS is very high** because if we do the calculation the sum of distances of observations from their cluster centroids gives a very big result.
Say, WCSS = 8000



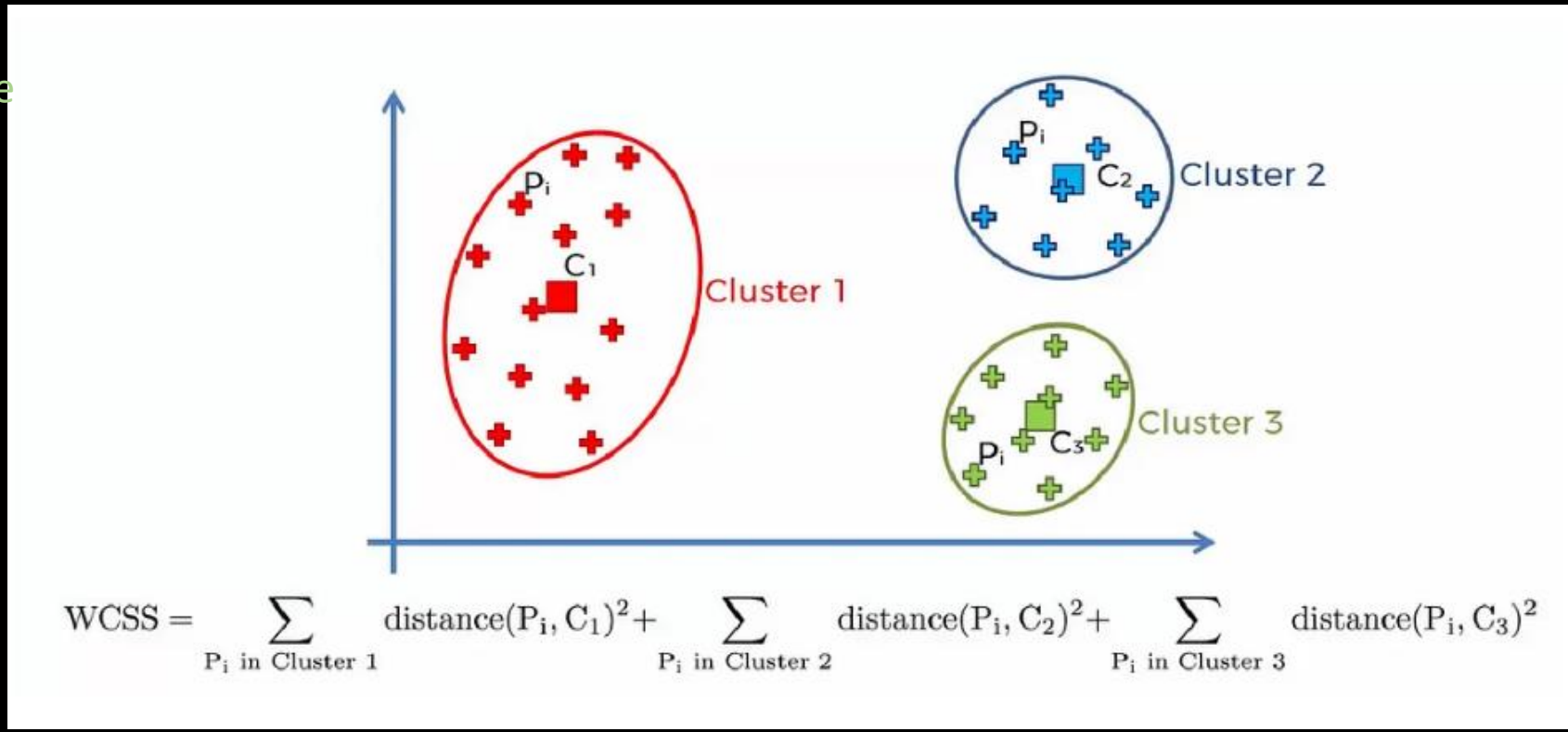
WCSS: How to find right K

Step 2. Now include 2 centroids in the dataset. Here we would see
WCSS result is less as compared to the previous result.
Say, WCSS = 3000



WCSS: How to find right K

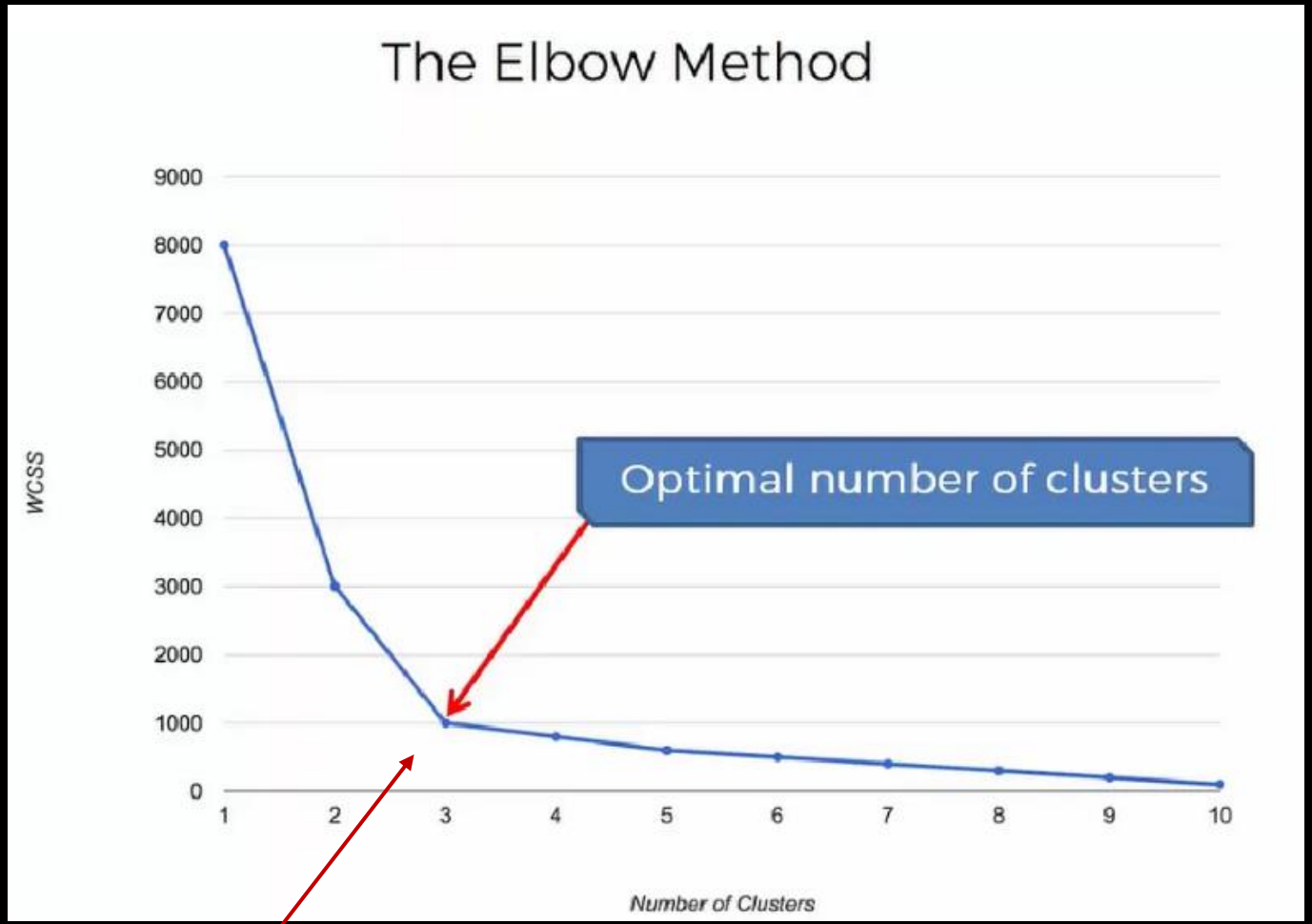
Step 3. Now include 3 centroids in the dataset. It gives a much lower result of **WCSS** than previous result.
Say, WCSS = 1000



WCSS: How to find right K

Step 4. Now, the question is w**hen to stop adding the centroids into the dataset?**

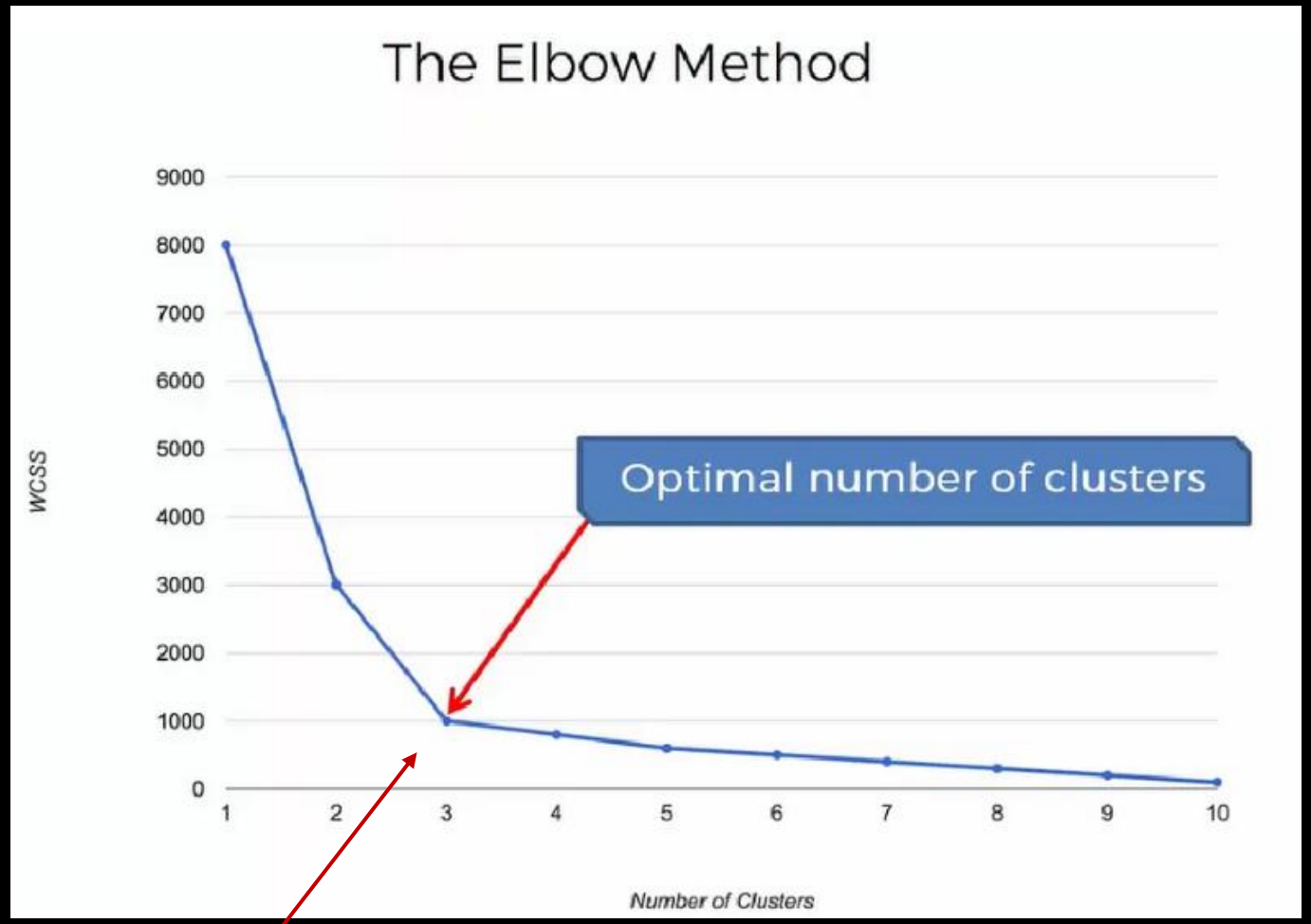
In order to answer this, let's analyze the sequential steps of adding the centroid.
So, According to the above graph, we can analyze the **substantial change in the value of WCSS by adding 2 centroids from 1 centroid.**
Again, see the **abrupt change by adding 3 centroids from 2 centroids.**
By adding centroids from **3 to 10, we can see that there is no abrupt change** but a small difference observed while adding new centroids.
So centroid 3 is a threshold that gives us a value of **how many clusters to include in our dataset. The point with k=3 is at the elbow.**



WCSS: How to find right K

How is the optimal number of clusters found using WCSS?

1. The Elbow Method can be used to find the optimal number of clusters.
2. Before the elbow, increasing K significantly reduces WCSS.
3. **After the elbow, adding more clusters has a minimal effect on WCSS, which could indicate overfitting**



WCSS(p2)

How does WCSS work in K-means clustering?

1. The K-means algorithm randomly chooses cluster centers.

2. The WCSS is calculated after each iteration.

3. The algorithm continues until the WCSS becomes stable, or until the assignments no longer change.

Silhoutte score(p1)

- The silhouette score is commonly used to assess the performance of clustering algorithms like K-Means.
- The silhouette score is a metric that measures how well data points fit into their clusters and how far they are from other clusters. It ranges from -1 to 1.

What does a silhouette score for a data point indicate?

- Silhouette score ~ 1: The data point is close to the average distance of its cluster and far from other clusters. This indicates that the data point is **well matched** to its cluster. If silhouette value is close to 1, sample is well-clustered and already assigned to a very appropriate cluster
- Silhouette score ~ 0: Data point could be assigned to another cluster closest to it and the sample lies equally far away from both the clusters. That means it indicates overlapping clusters
- Silhouette score ~ -1 : The data point is closer to other clusters than it is to its own cluster. This indicates that the clustering configuration may have too many or too few clusters. Also suggests **overlapping or poorly separated clusters**

Calculation steps:

1. For each data point, calculate the average distance to other points in its own cluster ("a"). Aka **cohesion**
2. For each data point, calculate the average distance to the **nearest** neighboring cluster ("b"). Aka **separation**
3. Compute the silhouette score for each data point using the formula: **(b - a) / max(a, b).**
4. **Average the silhouette scores across all data points** to get the overall silhouette score.

Silhoutte score(p2)

Data Points and Their Clusters:

Cluster Red:
Point [4 18]
Point [5 18]

Cluster Green:
Point [15 10]
Point [13 12]

Cluster Blue:
Point [8 4]
Point [7 8]

EXAMPLE:
Lets find Silhoutte score for DP a1=(7,8):
a (cohesion) value for DP:
Distance from a1 to DP (8,4): 4.123 = a
i.e. Mean intra-cluster distance (a): 4.123

A separation value of DP a1:
Distance from a1 to DP (4,18): 10.44
Distance from a1 to DP (5,18): 10.20

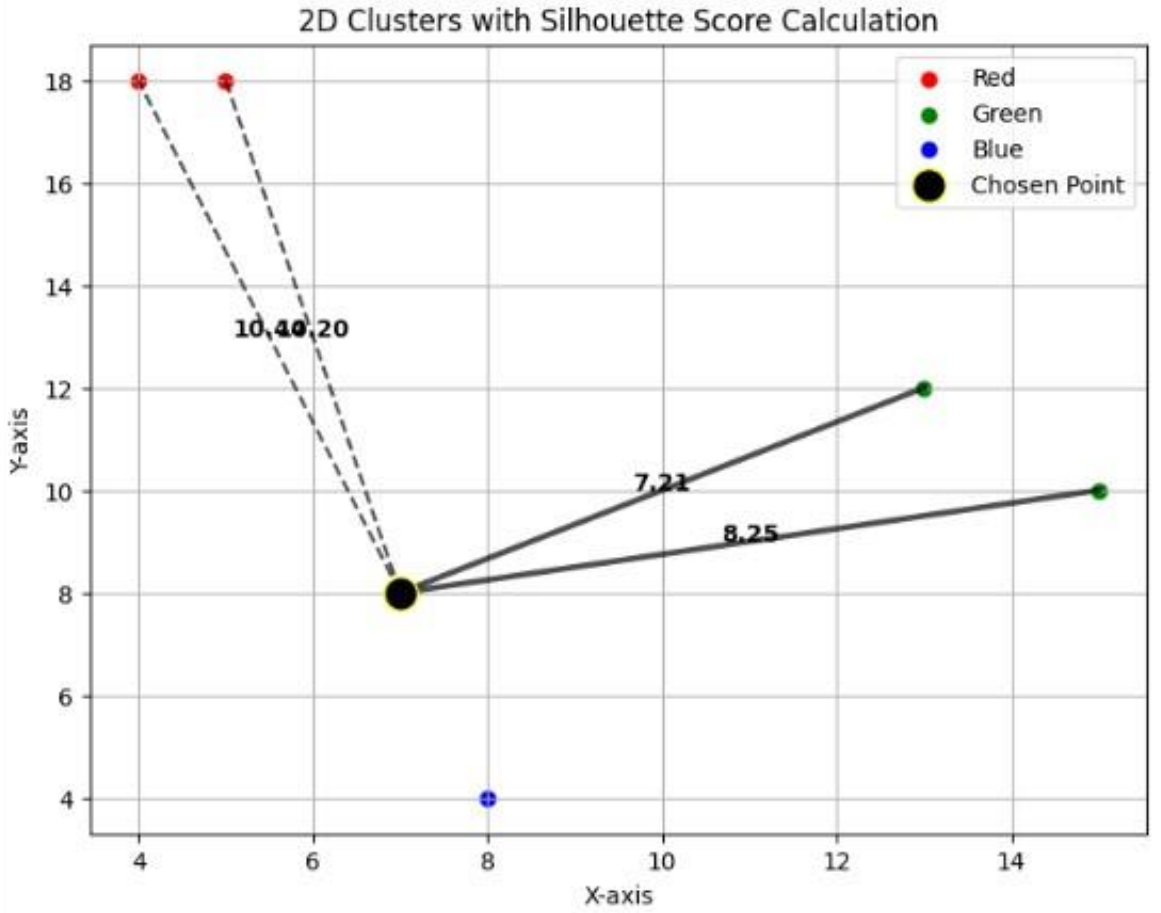
Distance from a1 to DP (13,12): 7.21
Distance from a1 to DP (15,10): 8.246

The nearest cluster is the one in blue.
Average separation for Point a1 from cluster blue = (7.21+8.246)/ 2 ≈ 7.728 =b
i.e. Mean nearest inter-cluster distance (b): 7.728

So the silhouette score for this DP is (b - a) / max(a, b) = (7.728-4.123)/7.728 = 0.466

This way we can calculate sil. score for all points and take their average.

```
Data Points and Their Clusters:
Point [ 4 18] -> Cluster Red
Point [ 5 18] -> Cluster Red
Point [15 10] -> Cluster Green
Point [13 12] -> Cluster Green
Point [8 4] -> Cluster Blue
Point [7 8] -> Cluster Blue
```




```
Silhouette Score Calculation:
Randomly chosen point: [7 8]
Cluster name of chosen point: Blue
Mean intra-cluster distance (a): 4.1231
Mean nearest inter-cluster distance (b): 7.7287
Silhouette score for the point: 0.4665
```

Silhoutte score(p3)

Limitations

- **Distance Metric Sensitivity**: Works best with Euclidean distance; struggles with non-Euclidean metrics.
- **Cluster Shape Sensitivity**: Assumes spherical clusters; less effective for irregular shapes (e.g., in DBSCAN).
- **Scalability**: Computationally expensive for large datasets due to pairwise distance calculations.

Davies-Bouldin Index (TODO)

KMeans++ (p1)

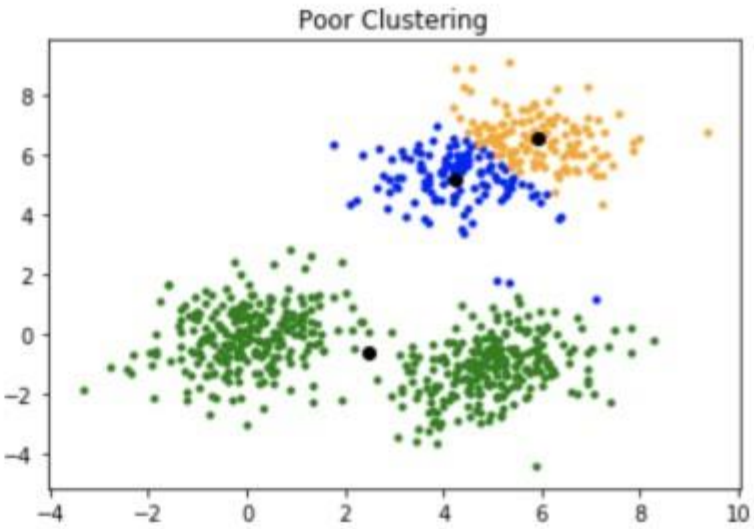
Drawback of standard K-means algorithm:

One disadvantage of the K-means algorithm is that **it is sensitive to the initialization of the centroids** or the mean points.

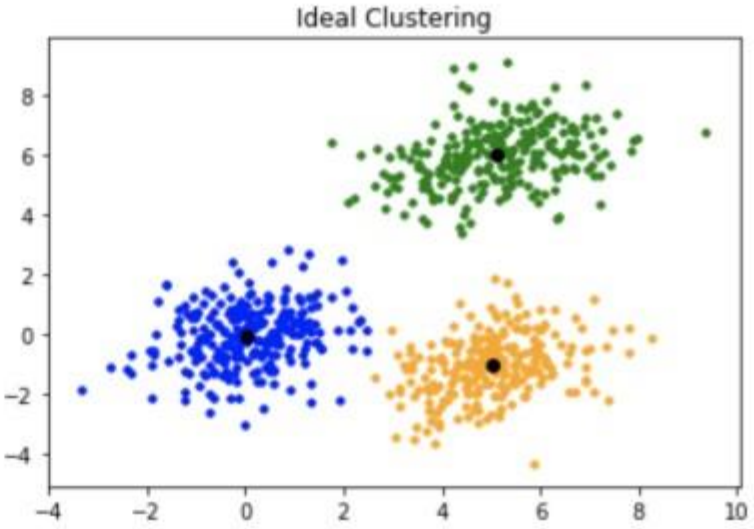
1. If a centroid is initialized to be a “far-off” point, it might just end up with no points associated with it, and at the same time, more than one cluster might end up linked with a single centroid.
2. Similarly, more than one centroid might be initialized into the same cluster resulting in poor clustering. For example, consider the images.

A poor initialization of centroids resulted in poor clustering.

A poor initialization of centroids resulted in poor clustering.



This is how the clustering should have been:



KMeans++ (p2)

To overcome the above-mentioned drawback we use K-means++. This algorithm ensures a smarter initialization of the centroids and improves the quality of the clustering. Apart from initialization, the rest of the algorithm is the same as the standard K-means algorithm:

The steps involved are:

1. **Randomly select the first centroid** from the data points.

2. For each data point compute its distance from the nearest, previously chosen centroid.

3. **Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid**. (i.e. the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid)

4. Repeat steps 2 and 3 until k centroids have been sampled

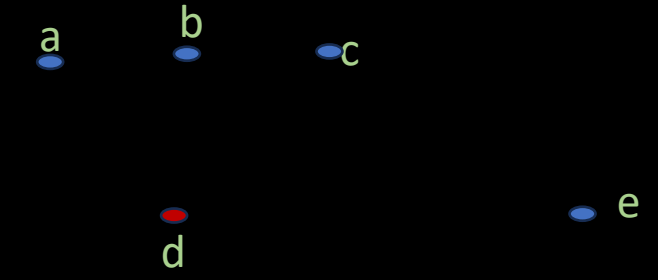
5. After the K centroids have been selected, then apply standard K-means algorithm

KMeans++ (p3)

The data points given are **a, b, c, d, e** and we need k=3 centroids.

1. Randomly 1st centroid: say its point **d**.

2. Calculate distance of each point from the previously chosen centroid(point **d**)



3.

Prob(**a** is next centroid) $\propto \sqrt{2}$

Prob(**b** is next centroid) $\propto 1$

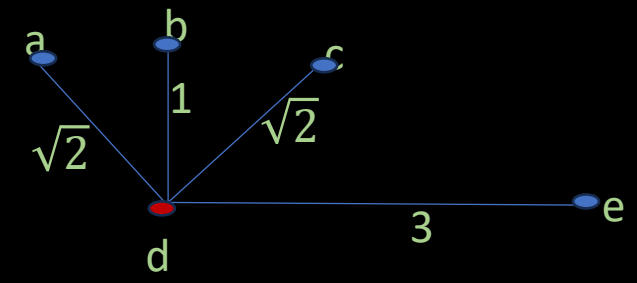
Prob(**c** is next centroid) $\propto \sqrt{2}$

Prob(**e** is next centroid) $\propto 3$

So,

Prob(**a** is next centroid) = $\sqrt{2}$ / (1+2$\sqrt{2+3}$) = 0.258

Prob(**b** is next centroid) = 1 / (1+2$\sqrt{2+3}$) = 0.182 and so on.



Say the next centroid selected is **c**. (Note: it could have selected other point too based on their probability)

4. Repeat steps 2 and 3 until k=3 centroids have been sampled

5. After the K=3 centroids have been selected, then apply standard K-means algorithm