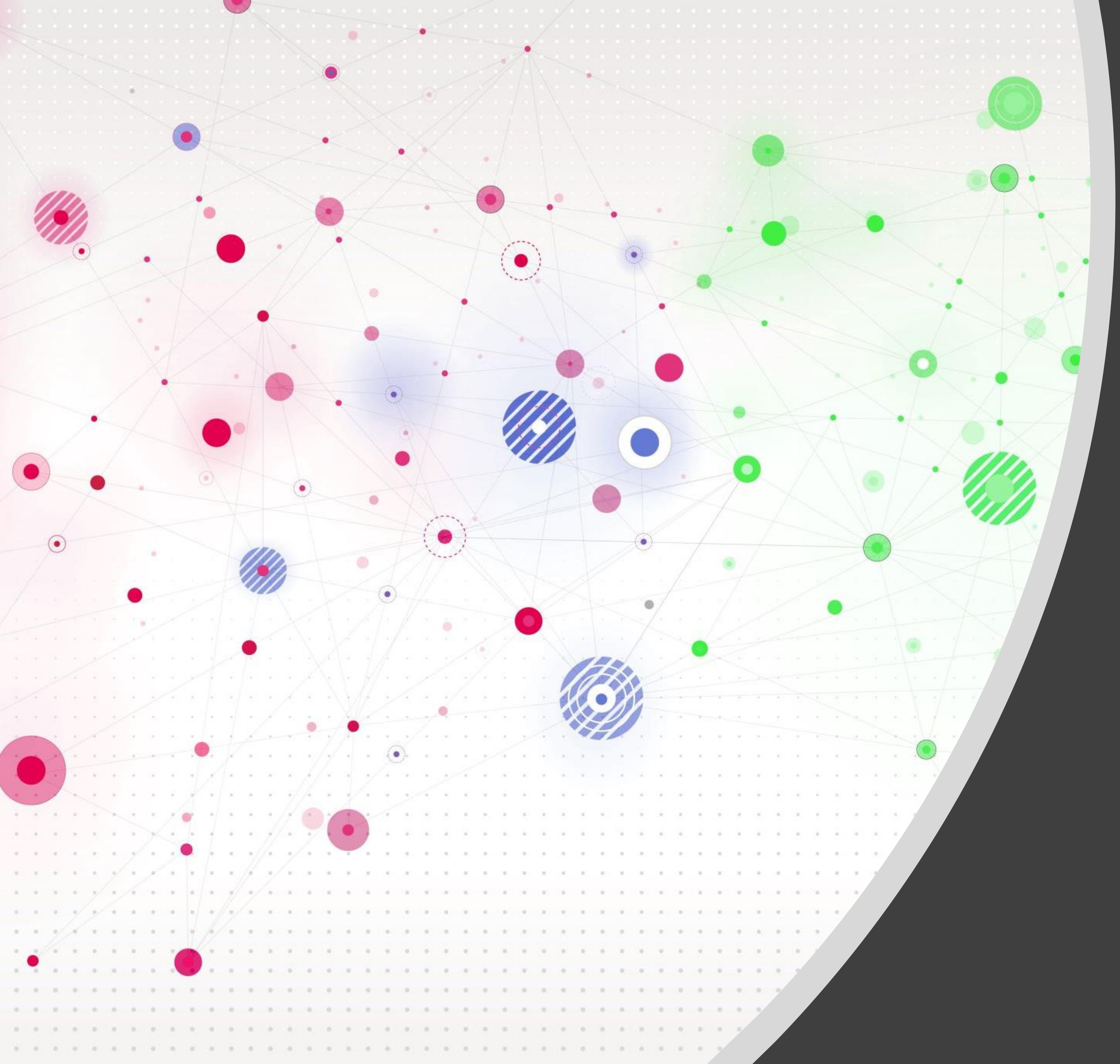


Clustering in Machine Learning

Arockiaraj S

Introduction

- In real world, not every data we work upon has a target variable.
  - This kind of data cannot be analyzed using supervised learning algorithms.
- We need the help of unsupervised algorithms.
- One of the most popular type of analysis under unsupervised learning is [Cluster analysis].

Clustering comes under unsupervised learning.

Goal is to group similar data points in a dataset.

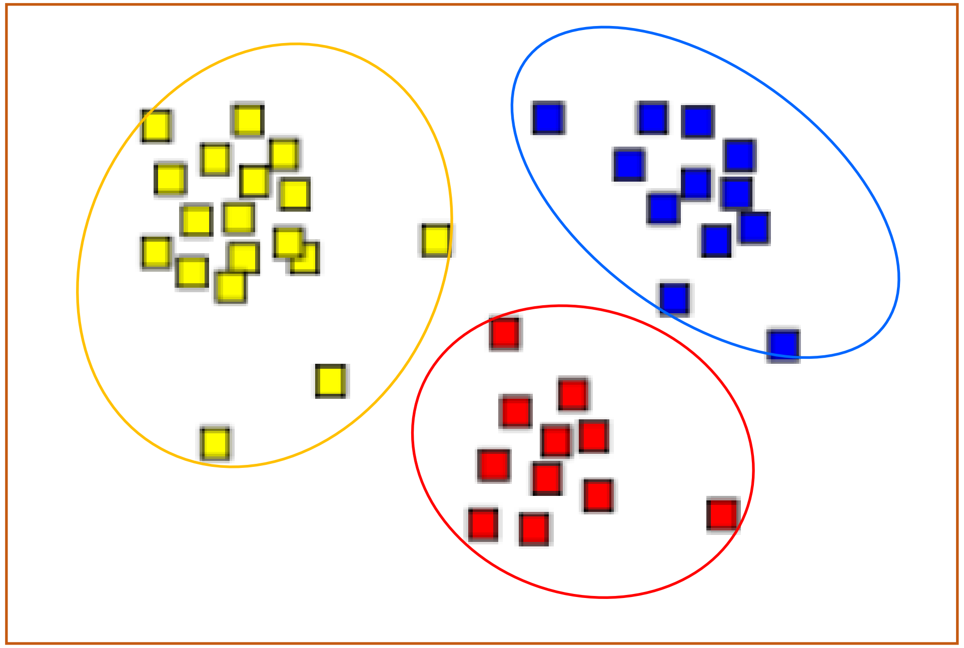
What is clustering?

The organization of unlabeled data into similarity groups called clusters.

Data items within a cluster are "*similar*"

Data items between clusters are "*dissimilar*"

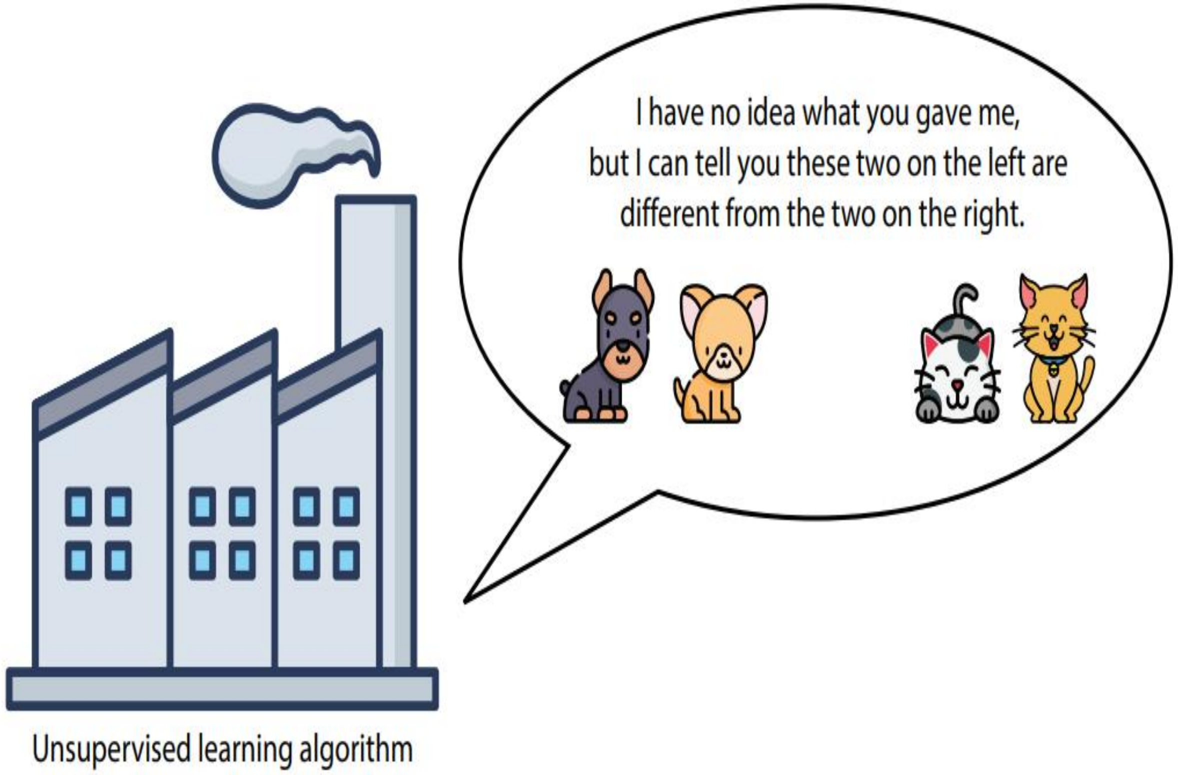
- Clustering is an unsupervised learning method
- Used for statistical data analysis in many fields.



Unsupervised learning

- Machine learning algorithms that works with unlabeled data.

- MLA must extract as much information as possible from a dataset (has no labels, or targets) to predict.
- Determine - two pictures are **similar or different**



Unsupervised learning

- Even if the labels are there, we can still use unsupervised learning techniques on our data **to preprocess** it and apply supervised learning methods more effectively.

- **clustering algorithms** The algorithms that group data into clusters based on similarity.

- **dimensionality reduction algorithms** The algorithms that simplify our data and describe it with fewer features

- **generative algorithms** The algorithms that can generate new data points that resemble the existing data

Clustering

- Consider the email dataset.

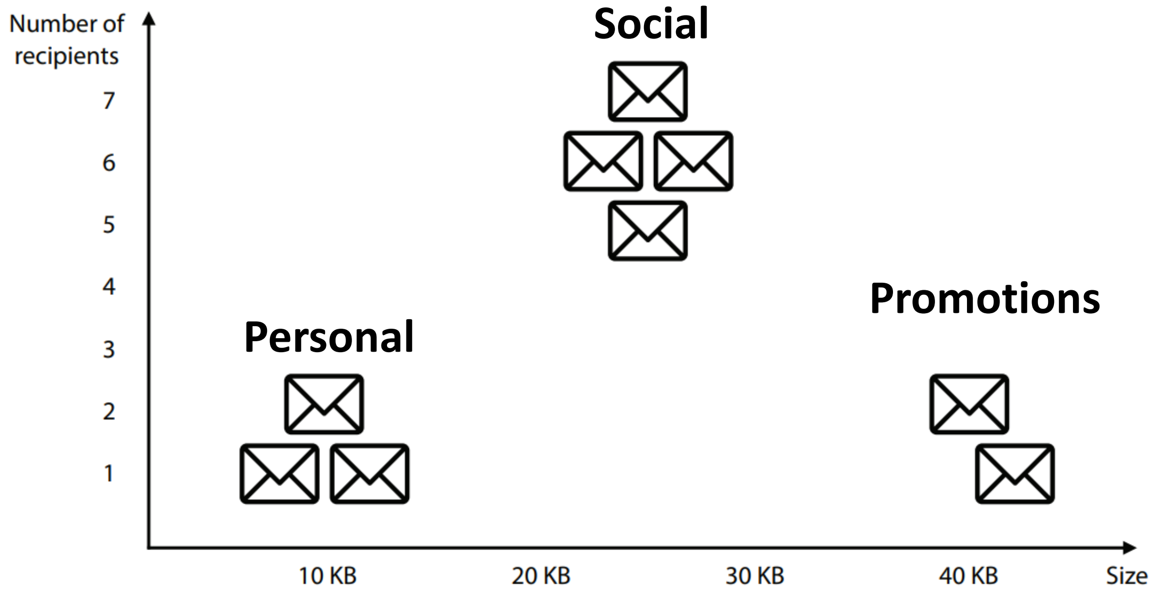
- Imagine that there is no labels
  - Email classification – **spam or ham is not available**.
  - The dataset is unlabeled, we don't know whether each email is spam or ham.
  - We can apply some clustering to this dataset.

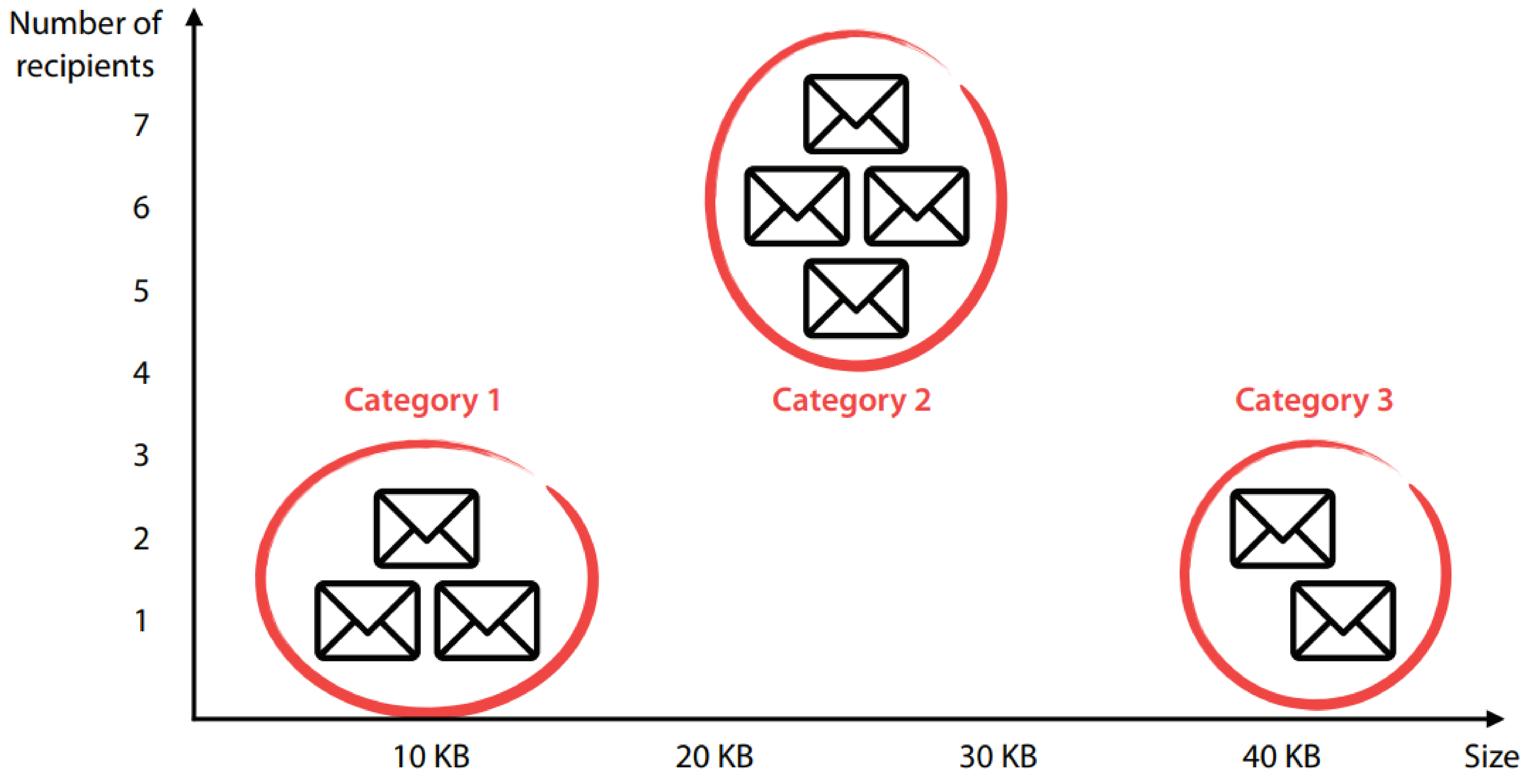
Clustering

- Group the emails – based on the number of words in the message, the sender, the number and size of the attachments, or the types of links inside the email.
- After clustering the dataset, a human (or a combination of a human and a supervised learning algorithm) could label these clusters by categories such as “**Personal**,” “**Social**,” and “**Promotions**.”

cluster the emails into three categories based on size and number of recipients

Email	Size	Recipients
1	8	1
2	12	1
3	43	1
4	10	2
5	40	2
6	25	5
7	23	6
8	28	6
9	26	7




Number of recipients
7
6
5
4
3
2
1
Category 1
Category 2
Category 3
10 KB
20 KB
30 KB
40 KB
Size

Other applications of clustering

- **Market segmentation**: dividing customers into groups based on demographics and previous purchasing behavior to create different marketing strategies for the groups
- **Genetics**: clustering species into groups based on gene similarity
- **Medical imaging**: splitting an image into different parts to study different types of tissue
- **Video recommendations**: dividing users into groups based on demographics and previous videos watched and using this to recommend to a user the videos that other users in their group have watched

Popular clustering algorithms

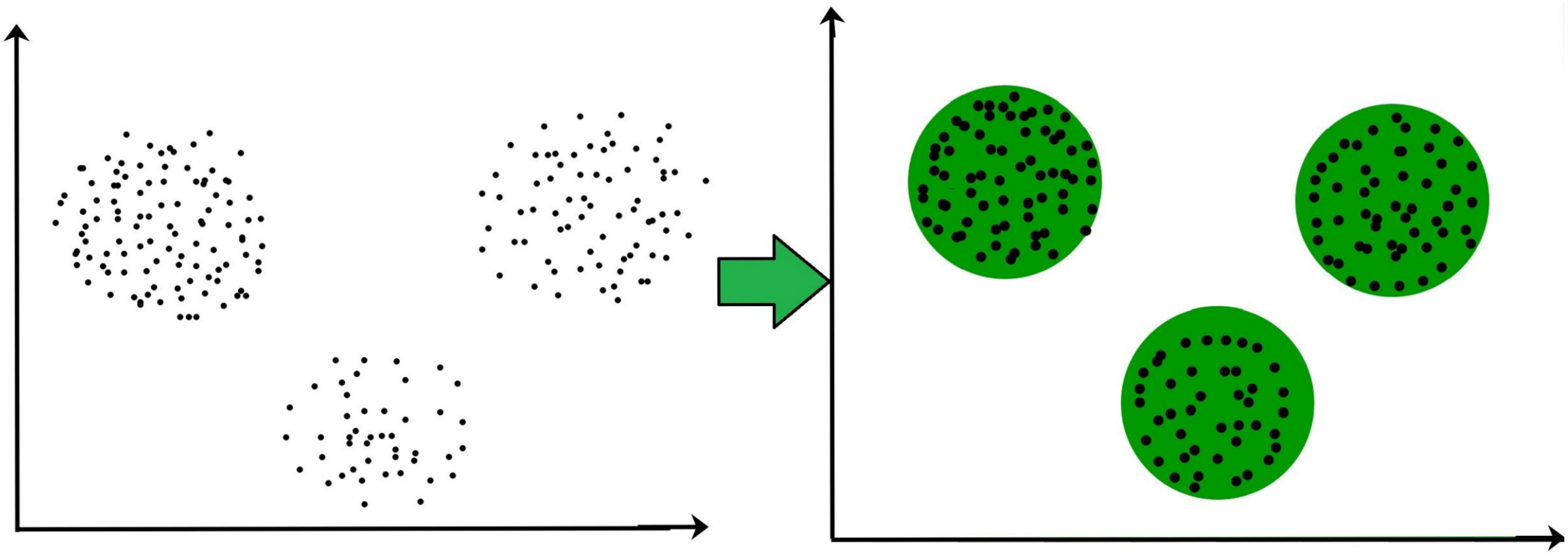
- **K-means clustering**: this algorithm groups points by picking some **random centers** of mass and moving them closer and closer to the points until they are at the right spots.

- **Hierarchical clustering**: this algorithm starts by **grouping the closest points together** and continuing in this fashion, until we have some well-defined groups.

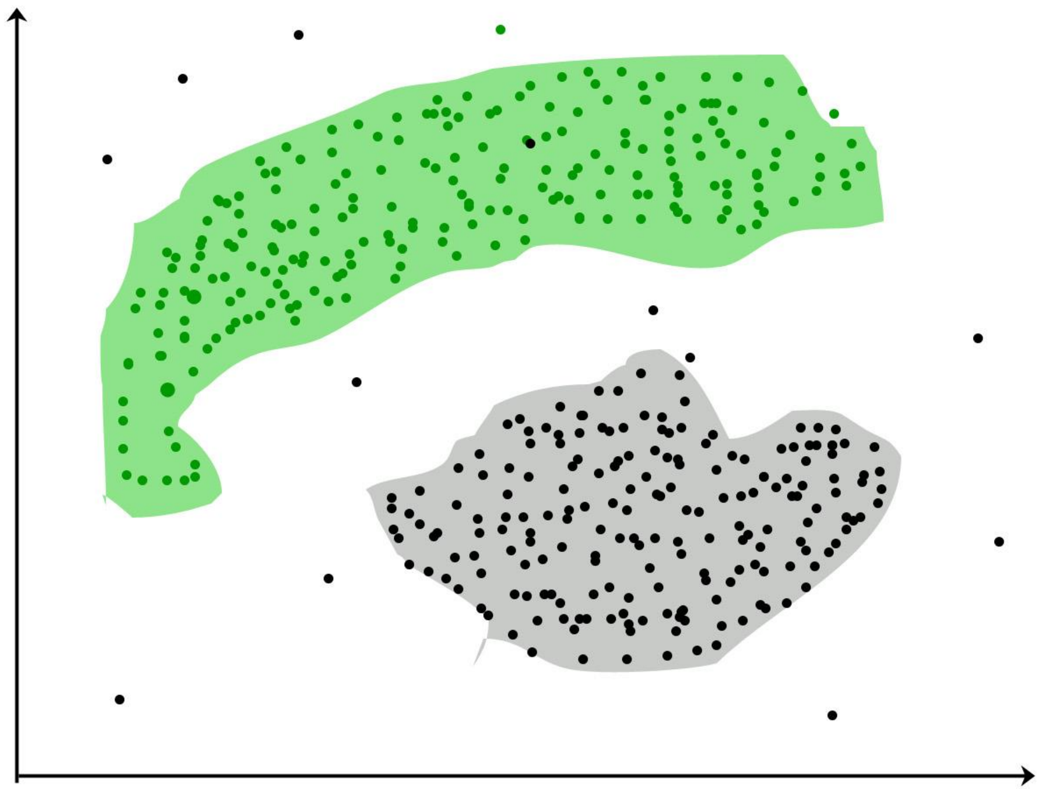
- **Density-based spatial clustering (DBSCAN):** this algorithm starts **grouping points together in places with high density**, while labeling the isolated points as noise.

- **Gaussian mixture models**: this algorithm does not assign a point to one cluster but instead assigns fractions of the point to each of the existing clusters.
- For example, if there are three clusters, A, B, and C, then the algorithm could determine that 60% of a particular point belongs to group A, 25% to group B, and 15% to group C.

It evaluates the similarity based on a metric like **Euclidean distance, Cosine similarity, Manhattan distance,** etc. and then group the points with highest similarity score together.



It is not necessary that the clusters formed must be circular in shape. The shape of clusters can be arbitrary.



Types of Clustering

Clustering can be divided into two subgroups

- **Hard Clustering:**
  - In this type of clustering, each data point belongs to a cluster completely or not.
  - Example: there are 4 data points, and we must cluster them into 2 clusters.
  - So, each data point will either belong to cluster 1 or cluster 2.

Data Points	Clusters
A	C1
B	C2
C	C2
D	C1

Types of Clustering

- **Soft Clustering**:
  - In soft clustering, instead of putting each data point into a separate cluster, a ***probability or likelihood*** of that data point to be in those clusters is assigned.
  - For example, each datapoint is assigned a probability to be in either in C1 or C2.

Data Points	Probability of C1	Probability of C2
A	0.91	0.09
B	0.3	0.7
C	0.17	0.83
D	1	0

Distance and Similarity measures in Clustering

Distance Measures:

- Used to determine the dissimilarity between two data points.

- There are several distance measures commonly used in clustering.

1. Minkowski distance family
    1. Euclidean distance
    2. Manhattan distance

2. Cosine similarity

3. Mahalanobis distance

1. Euclidean Distance:

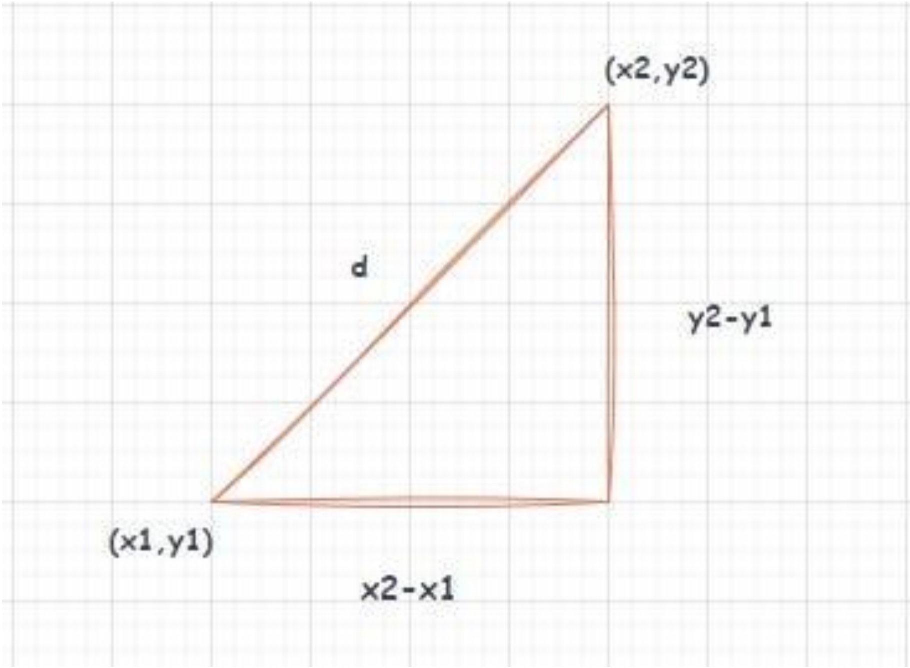
- This is the most common distance measure used in clustering.
- Euclidean distance can be used for data with continuous variables, such as height, weight, or age.
- if $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ are two points in Euclidean *n*-space, then the distance (*d*) from *p* to *q*, or from *q* to *p* is given by the Pythagorean formula:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Calculate Euclidean distance using Math Import function:

```
import math

def euclidean_distance(point1, point2):
# Calculate the sum of the squared differences between the
coordinates
squared_diffs = [(point1[i] - point2[i]) ** 2 for i in
range(len(point1))]
sum_squared_diffs = sum(squared_diffs)
# Take the square root of the sum to get the Euclidean distance
distance = math.sqrt(sum_squared_diffs)
return distance

point1 = (1, 2, 3)
point2 = (4, 5, 6)
distance = euclidean_distance(point1, point2)
print(distance)
```



Output: 5.196152422706632

2. **Manhattan Distance:**

- This distance measure is also known as city block distance or taxicab distance.
- Manhattan distance can be used for data with discrete variables, such as the number of bedrooms in a house.

Manhattan Distance Formula

$$D_{mn}(x_i, x_j) = \sum_{l=1}^{d} |x_{il} - x_{jl}|$$

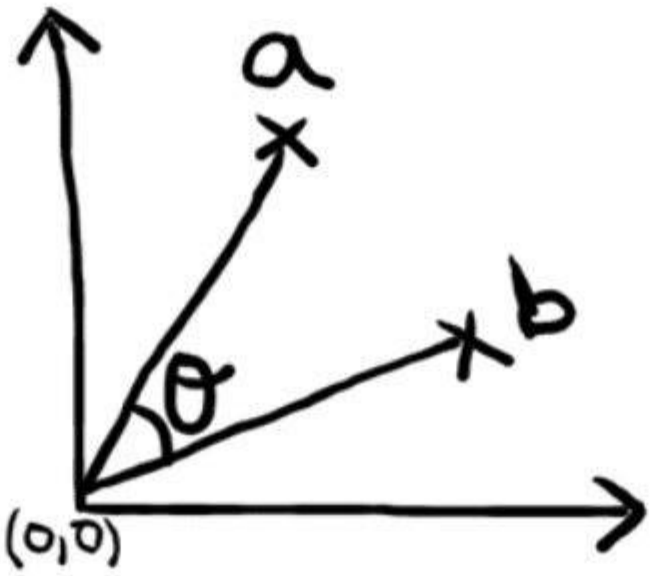
```
from math import sqrt

#create function to calculate Manhattan distance
def manhattan(a, b):
return sum(abs(val1-val2) for val1, val2 in zip(a,b))

#define vectors
A = [2, 4, 4, 6]
B = [5, 5, 7, 8]

#calculate Manhattan distance between vectors
manhattan(A, B)
```

$$D_{mn}(x_i, x_j) = \sum_{l=1}^{d} |x_{il} - x_{jl}|$$

Output: 9

3. **Cosine Similarity:**

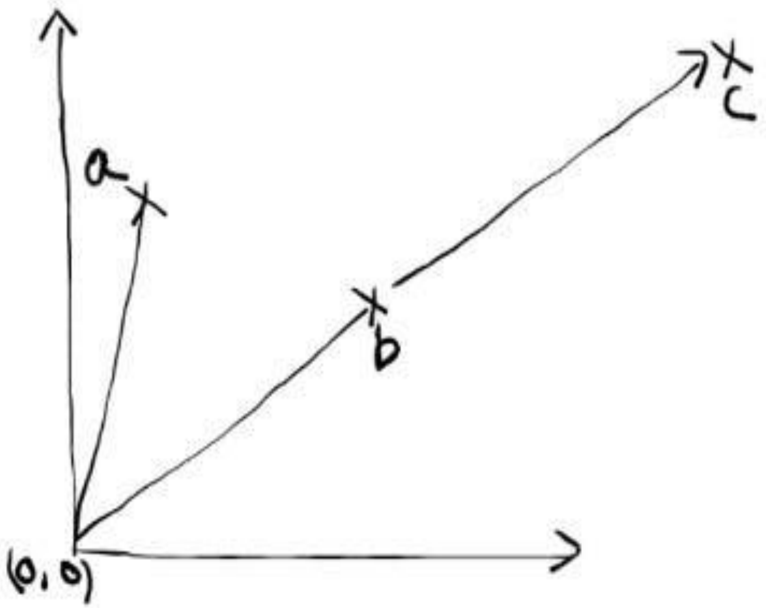
- This similarity measure is commonly used for text-based data or other high-dimensional data.
- It measures the cosine of the angle between two vectors in a multidimensional space.
- The cosine similarity ranges from -1 to 1, with 1 indicating that the vectors are identical a value of 0 indicating that they are orthogonal (perpendicular) to each other.



$$\cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$$

3. Cosine Similarity

- Cosine similarity cares only about the angle between the two vectors and not the distance between them. Assume there's another vector c in the direction of b.
- Now, the cosine similarity between b and c is 1 since the angle between b and c is 0 and cos(0) = 1.
- Even though the distance between b and c is large compared to a and b cosine similarity cares only about the direction of the vector and not the distance.



```python
def cos_sim(a, b):
"""Takes 2 vectors a, b and returns the cosine similarity according
to the definition of the dot product
"""
dot_product = np.dot(a, b)
norm_a = np.linalg.norm(a)
norm_b = np.linalg.norm(b)
return dot_product / (norm_a * norm_b)

# the counts we computed above
sentence_m = np.array([1, 1, 1, 1, 0, 0, 0, 0, 0])
sentence_h = np.array([0, 0, 1, 1, 1, 1, 0, 0, 0])
sentence_w = np.array([0, 0, 0, 1, 0, 0, 1, 1, 1])

# We should expect sentence_m and sentence_h to be more similar
print(cos_sim(sentence_m, sentence_h)) # 0.5
print(cos_sim(sentence_m, sentence_w)) # 0.25
```

Output: 0.5, 0.25

K-Means Clustering

MacQueen, 1967

K-means is a Partitional Clustering algorithm – dividing a dataset into distinct groups or clusters.

The *k*-means algorithm partitions the given data into *k* clusters: Each cluster has a cluster **center**, called **centroid**.

The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.

Partitioning-based

- Partitional clustering divides data objects into nonoverlapping groups – i.e., no object can be a member of more than one cluster, and every cluster must have at least one object – hard clustering.

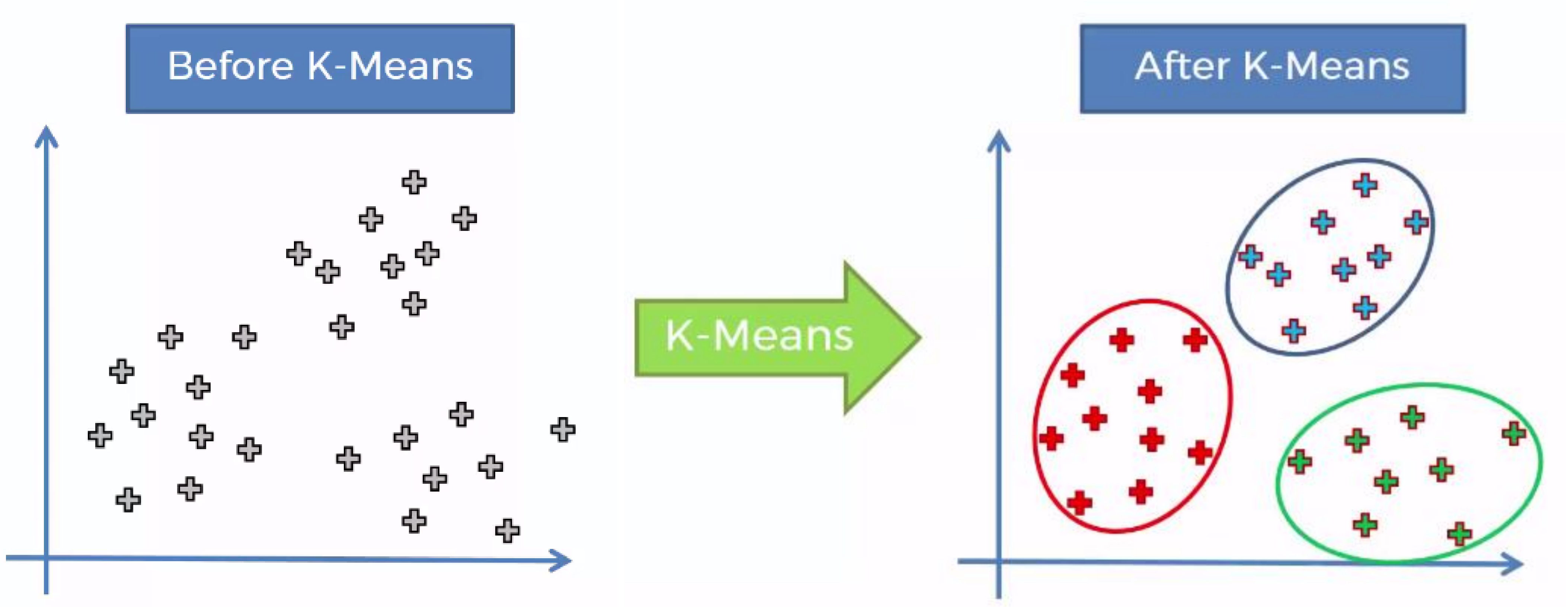
- K-means clustering require the user to specify the number of clusters, indicated by the variable ***k***.

- Partitional clustering algorithms work through an iterative process to assign subsets of data points into ***k clusters***.

Partitioning-based

- These algorithms are both nondeterministic, meaning they could produce different results from two separate runs even if the runs were based on the same input.
- Partitional clustering - strengths:
    1. They work well when clusters have a spherical shape.
    2. They're scalable with respect to algorithm complexity.
- Weaknesses:
    1. They're not well suited for clusters with complex shapes and different sizes.
    2. They break down when used with clusters of different densities.

What does K-means clustering accomplish for us?



K-Means Algorithm

Step-1: Choose the number of clusters (K)

Step-2: Select at random k points, the centroids (not necessarily from your dataset)

Step-3: Assign each data point to the closest centroid → that forms k clusters

Step-4: Compute and place the new centroid of each cluster

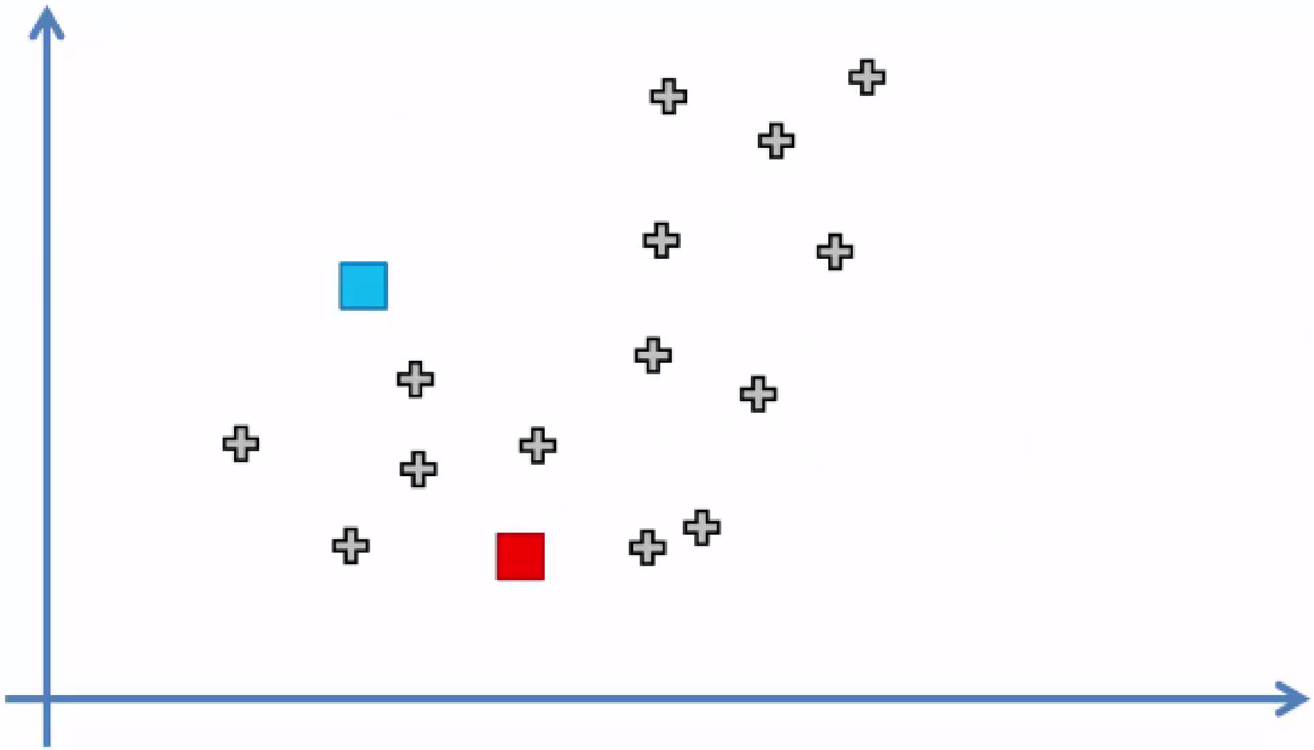
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.

Stop: your Model is ready

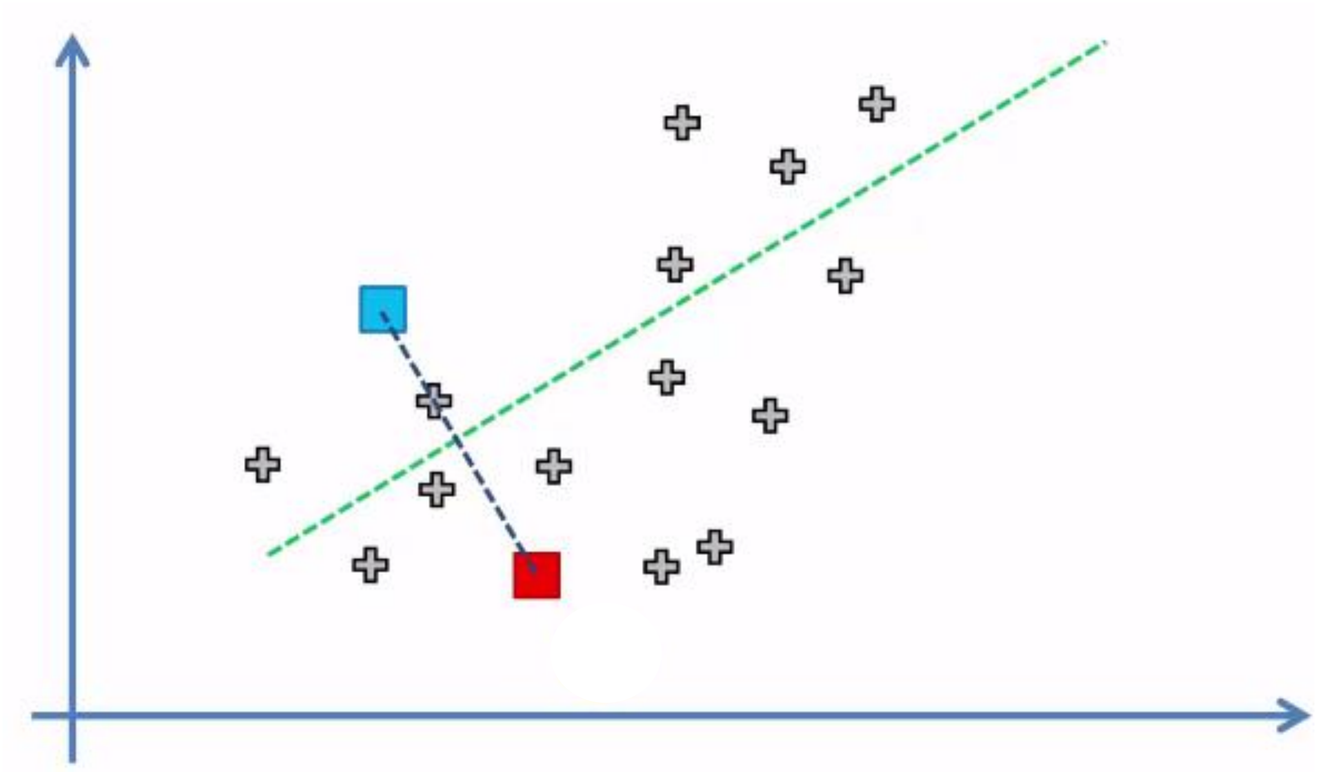
Example

STEP 1: Choose the number K of clusters: K = 2

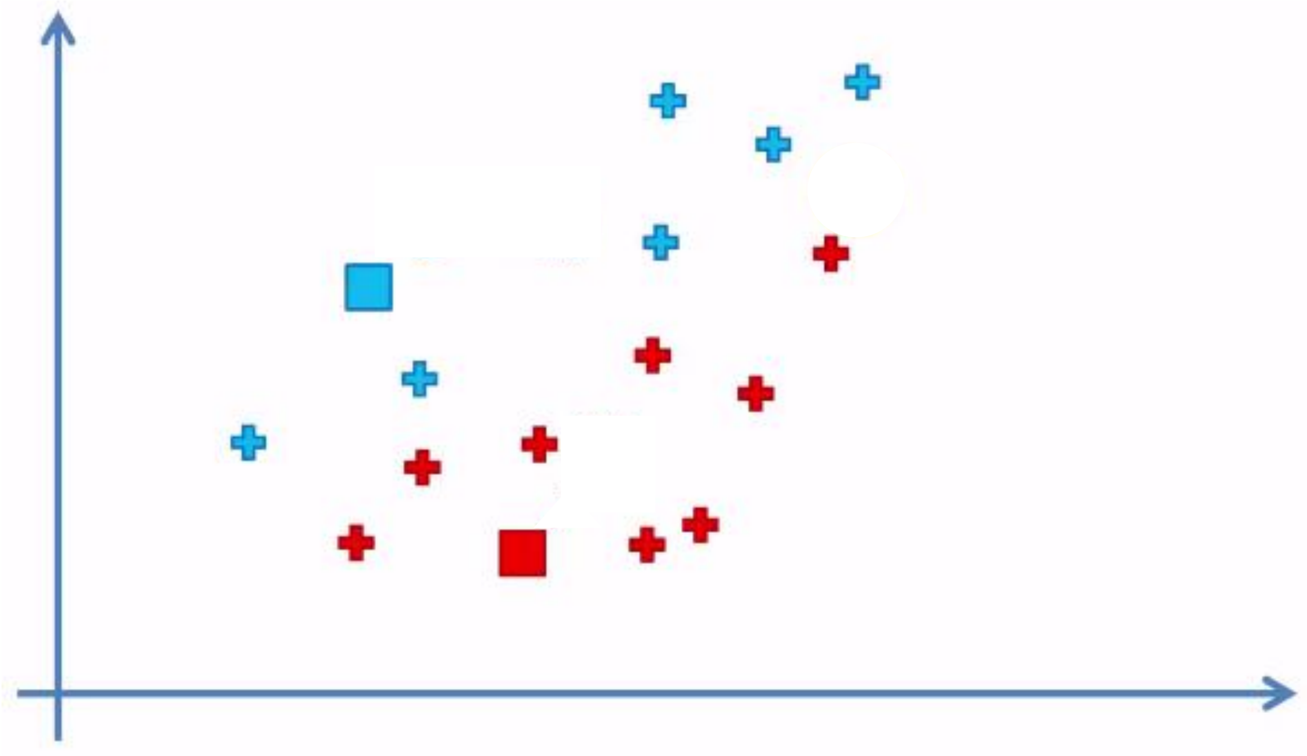
Step-2: Select at random k points, the centroids (not necessarily from your dataset)



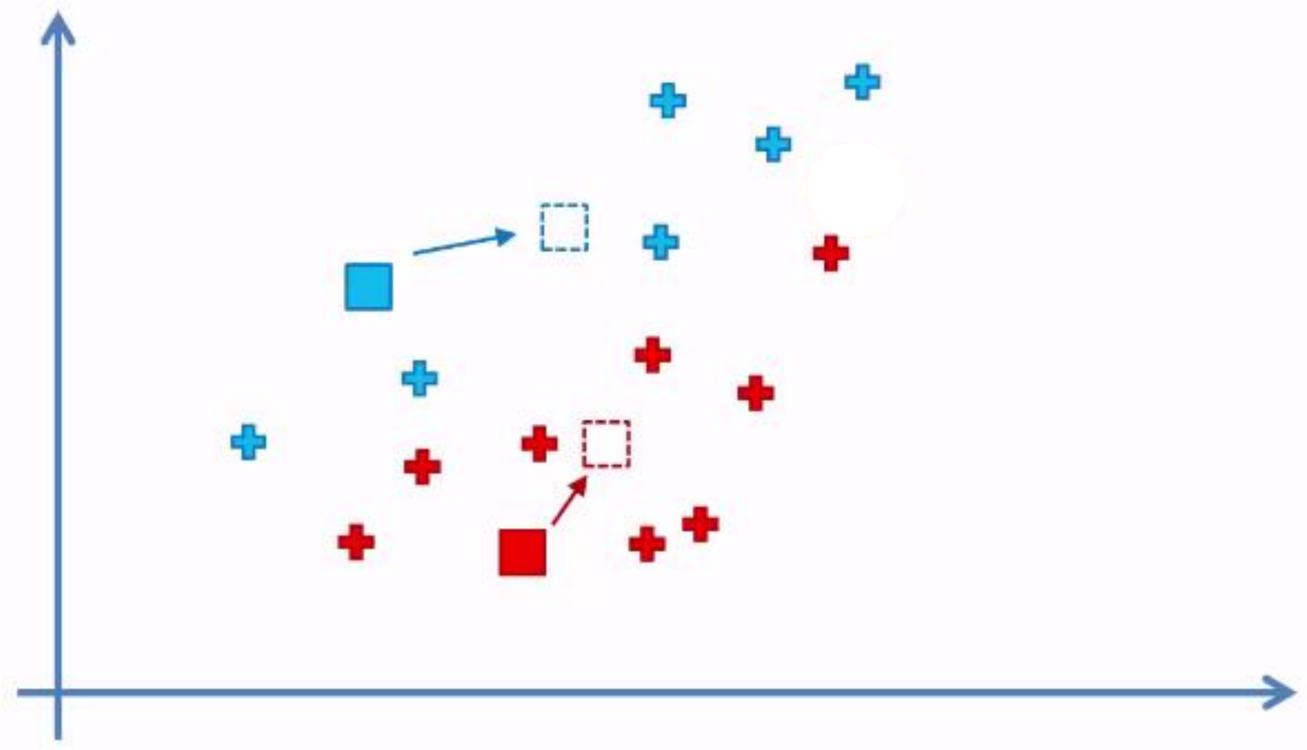
Step-3: Assign each data point to the closest centroid → that forms k clusters



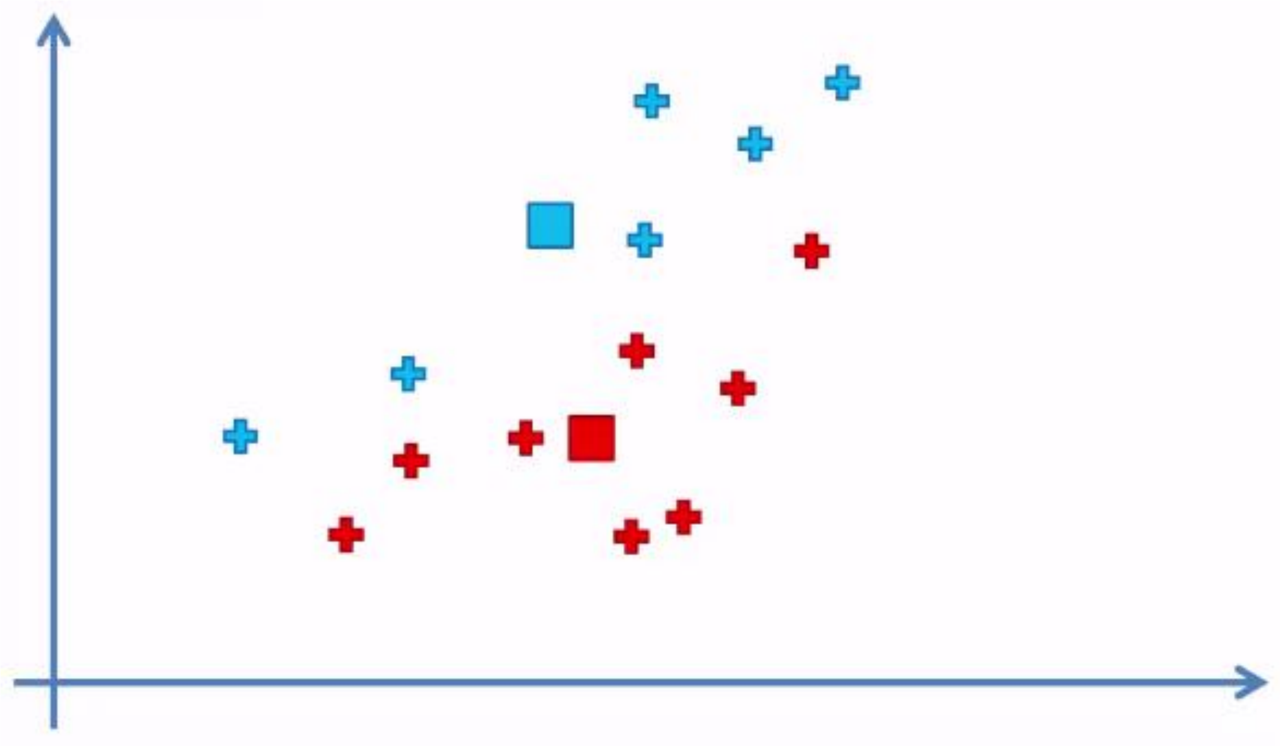
Step-4: Compute and place the new centroid of each cluster



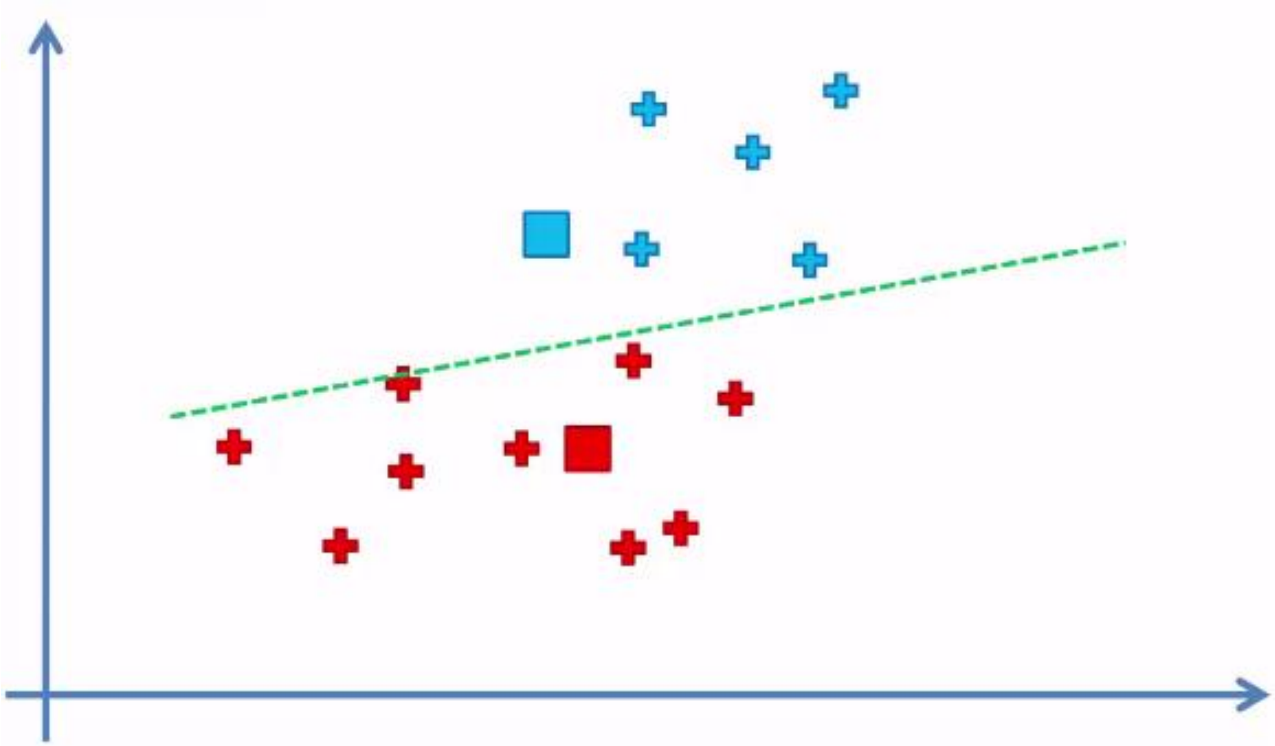
Step-4: Compute and place the new centroid of each cluster



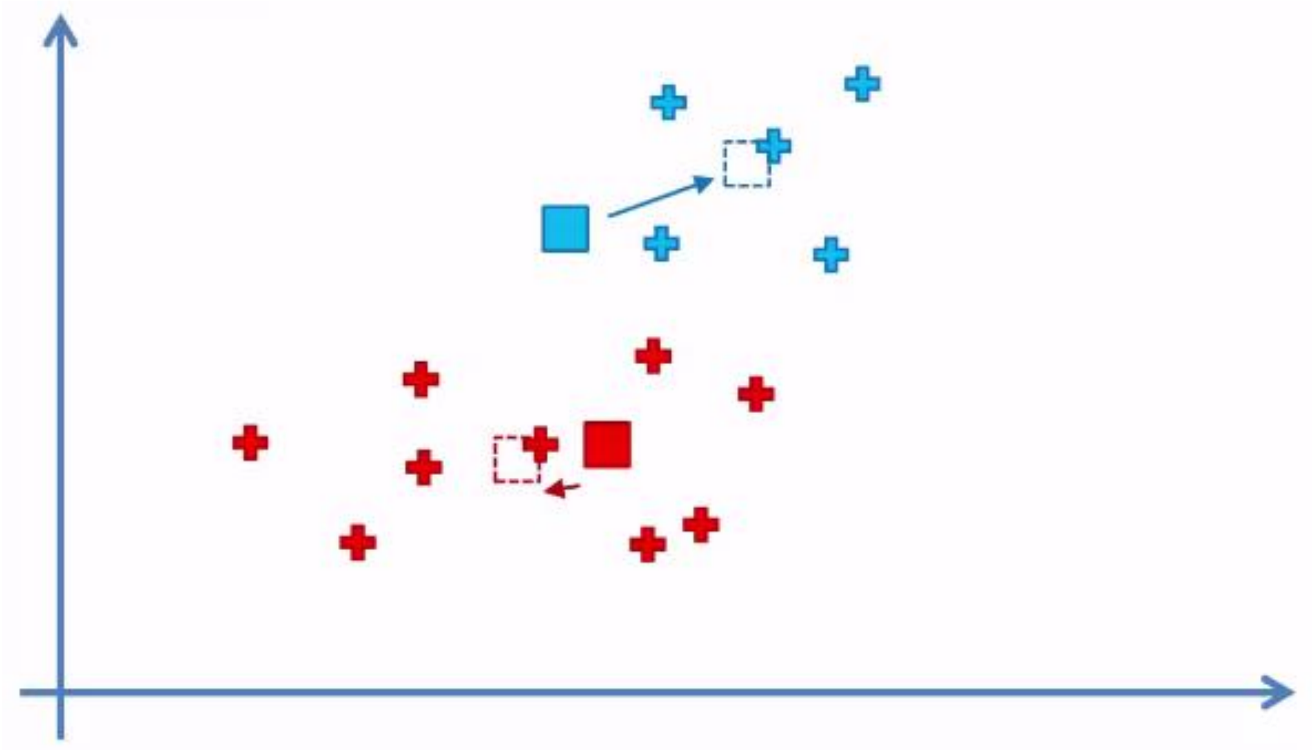
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



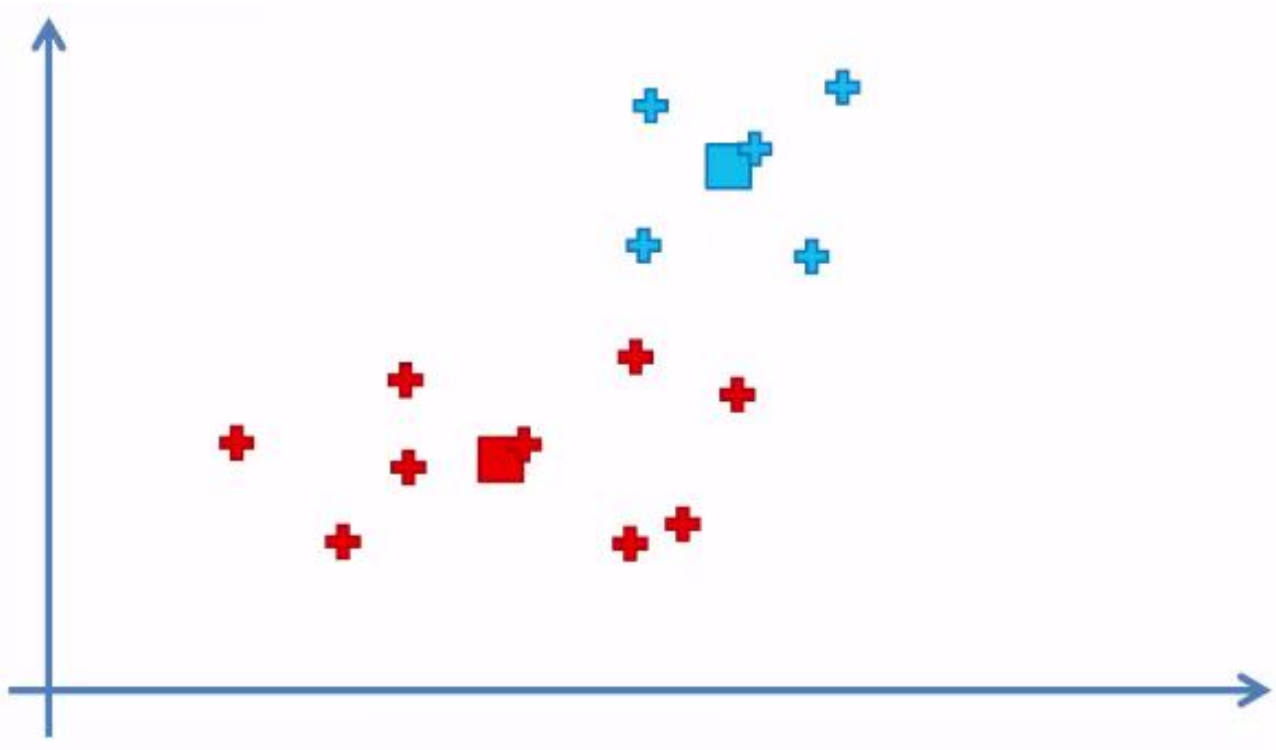
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



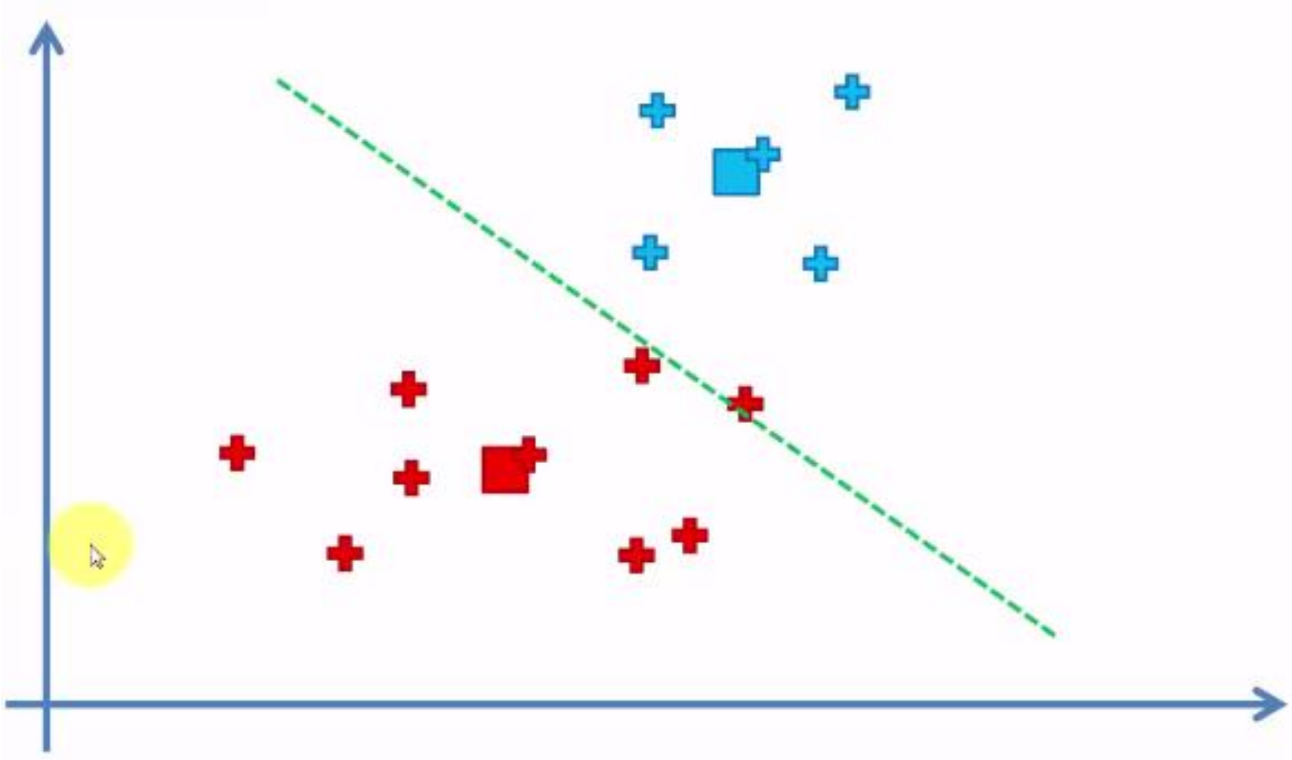
Step-4: Compute and place the new centroid of each cluster



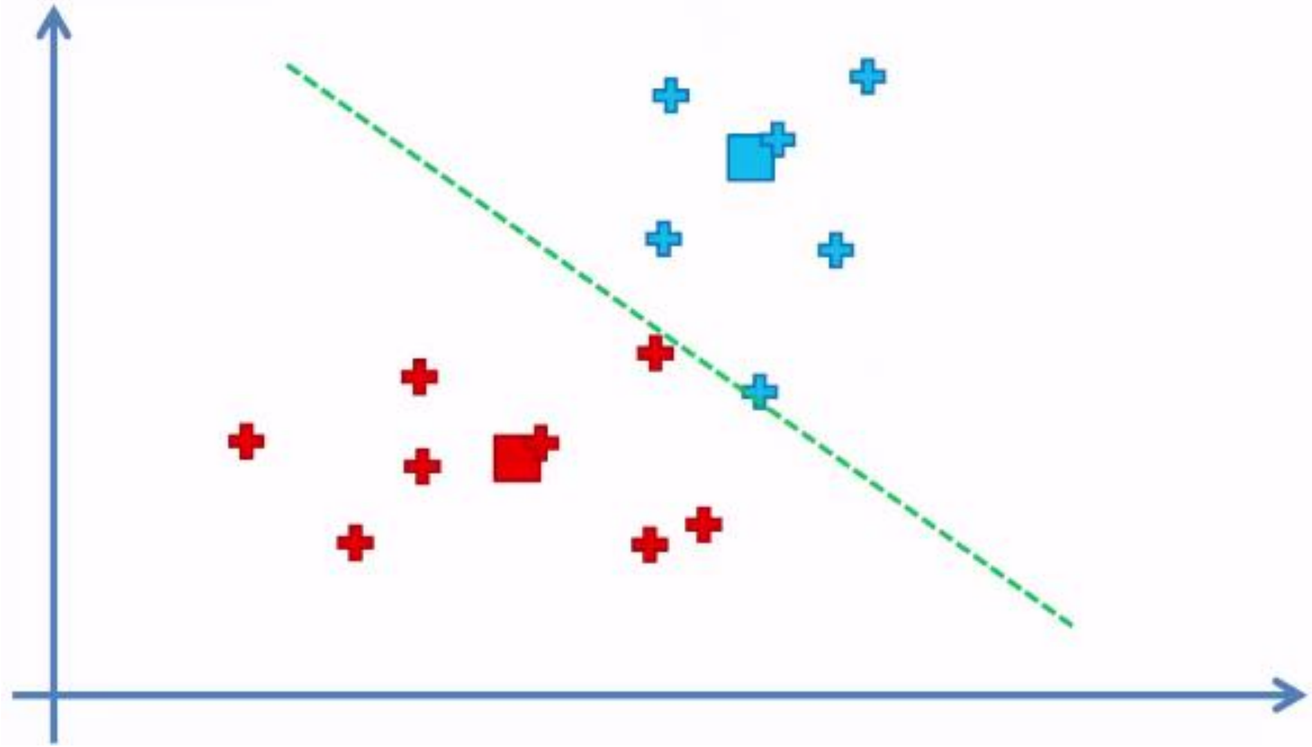
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



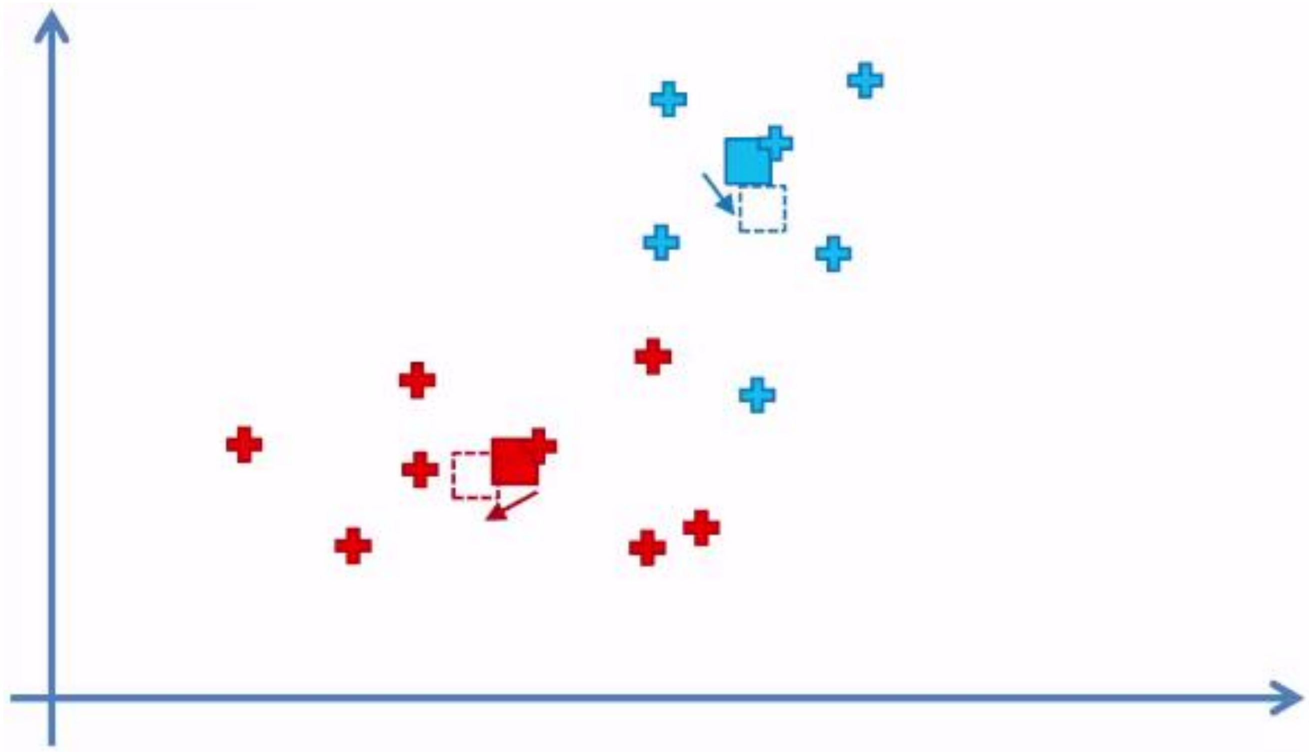
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



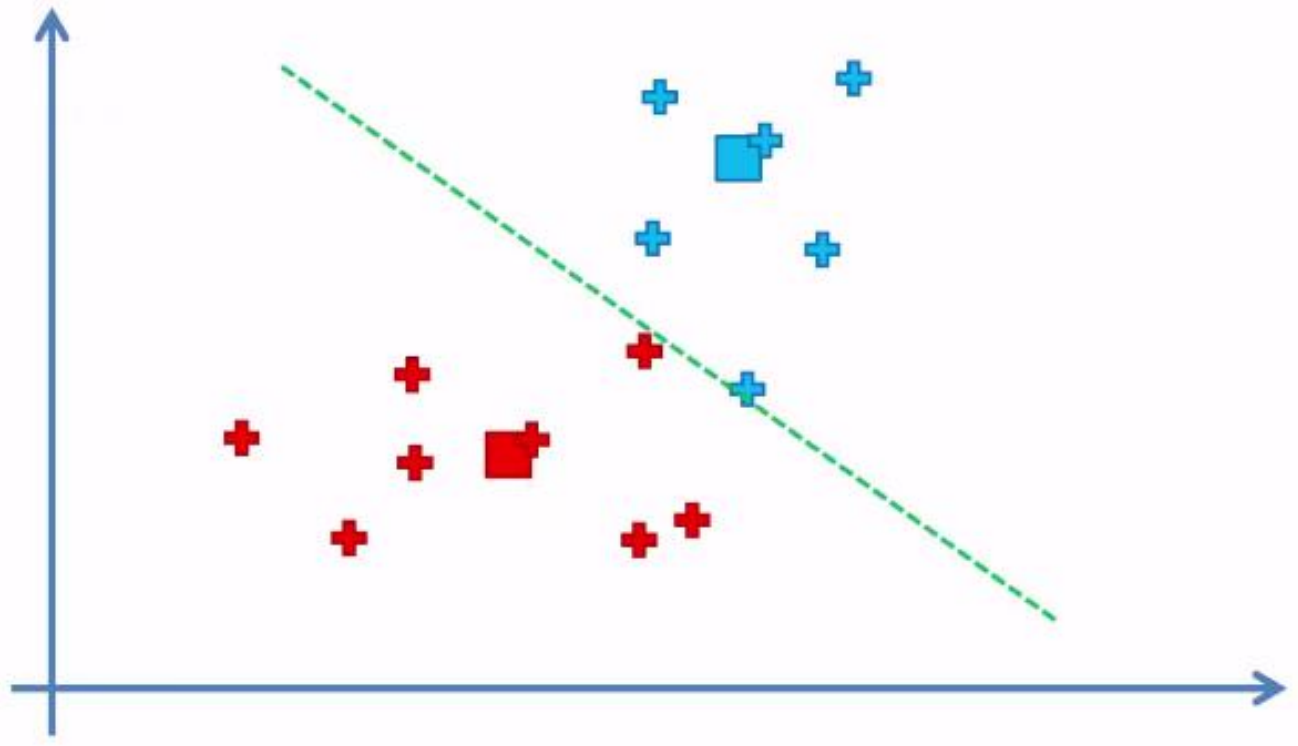
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



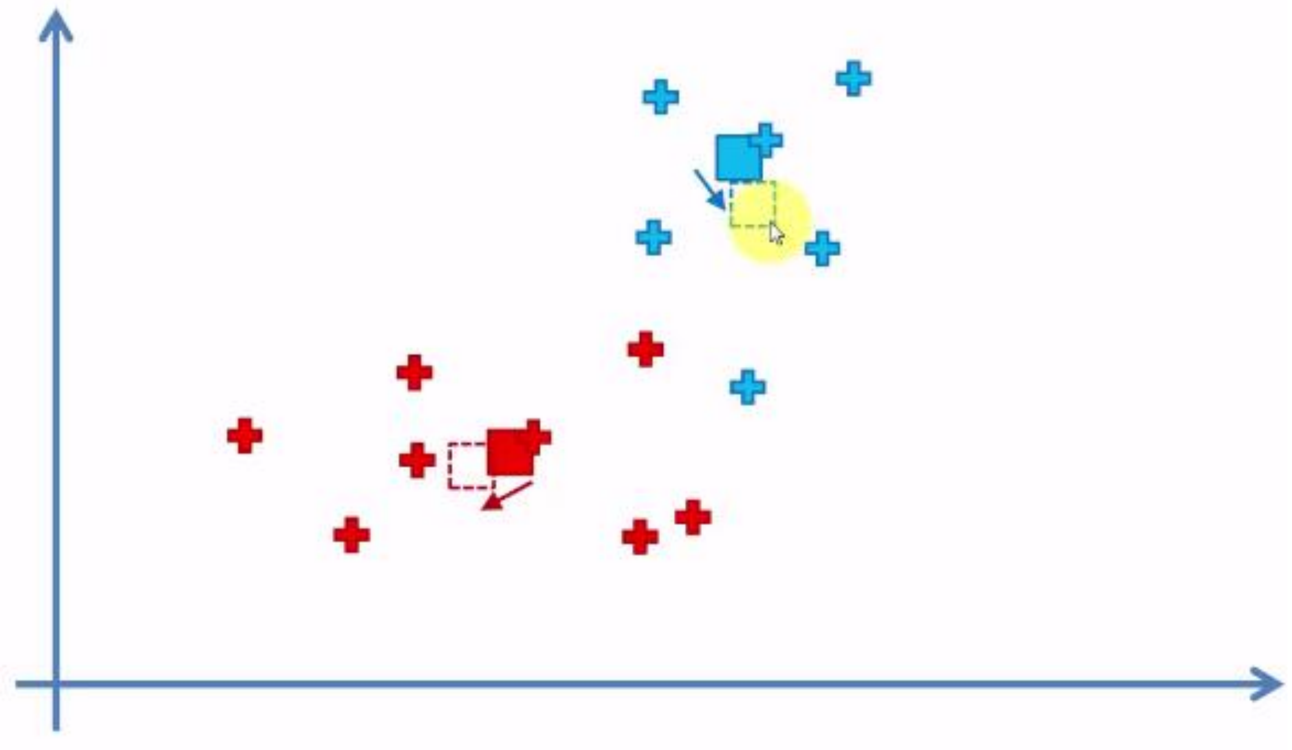
Step-4: Compute and place the new centroid of each cluster



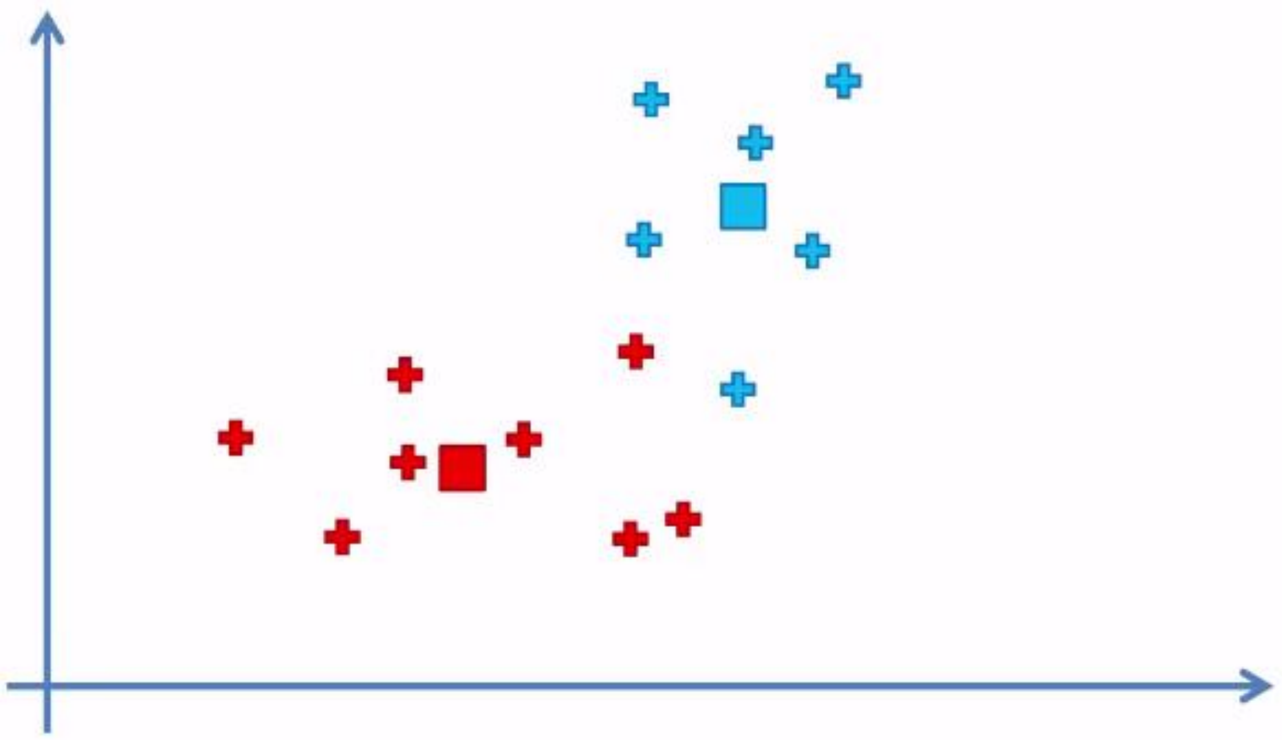
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



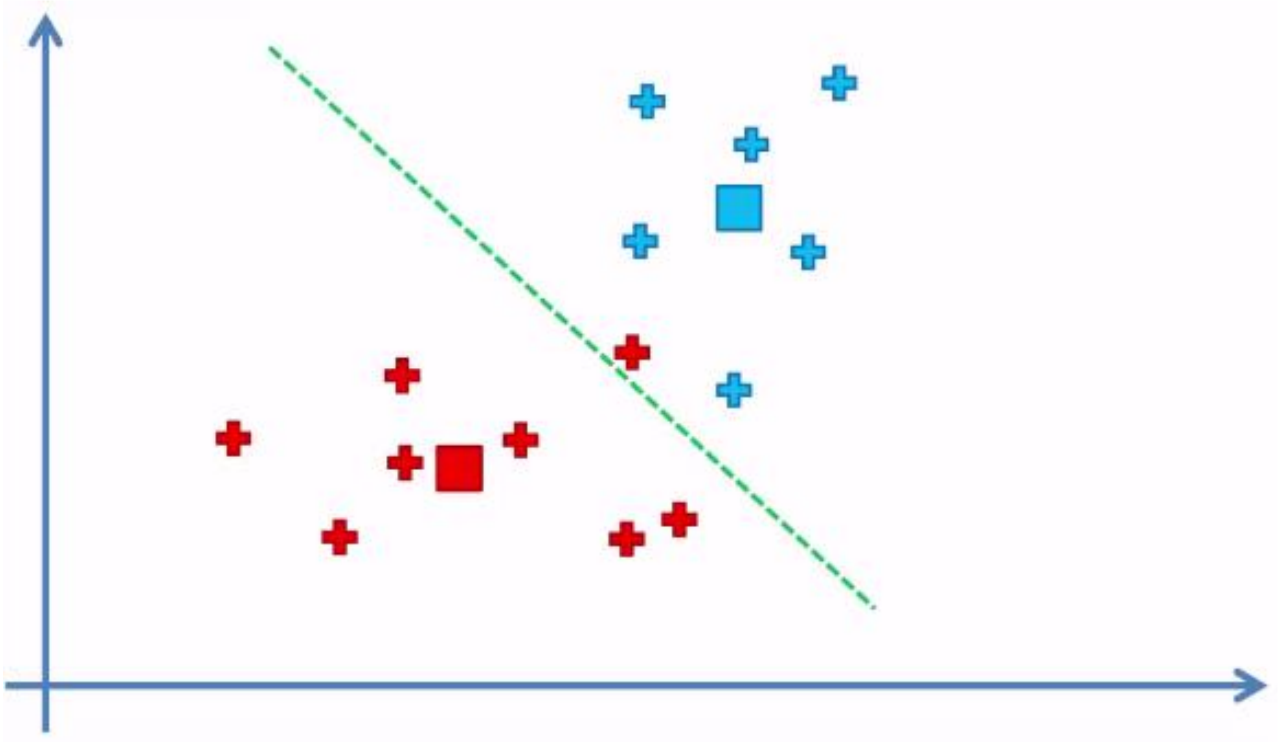
Step-4: Compute and place the new centroid of each cluster



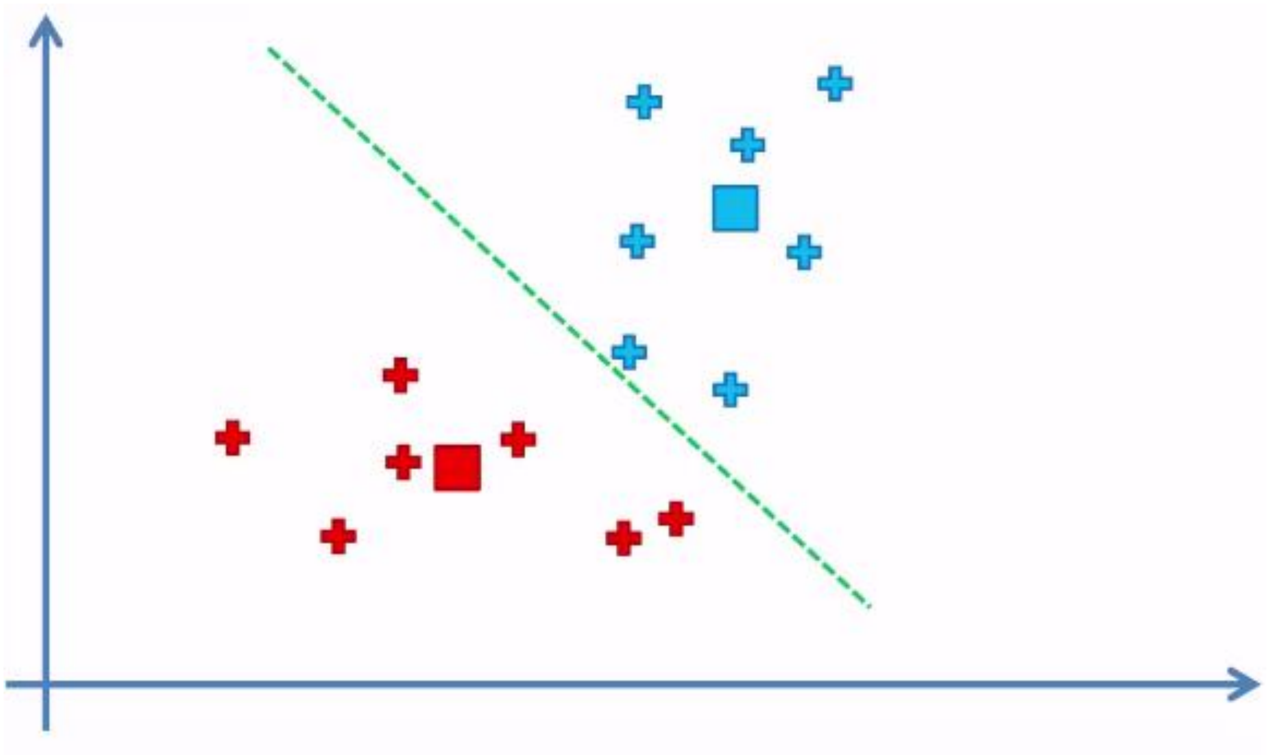
Step-4: Compute and place the new centroid of each cluster



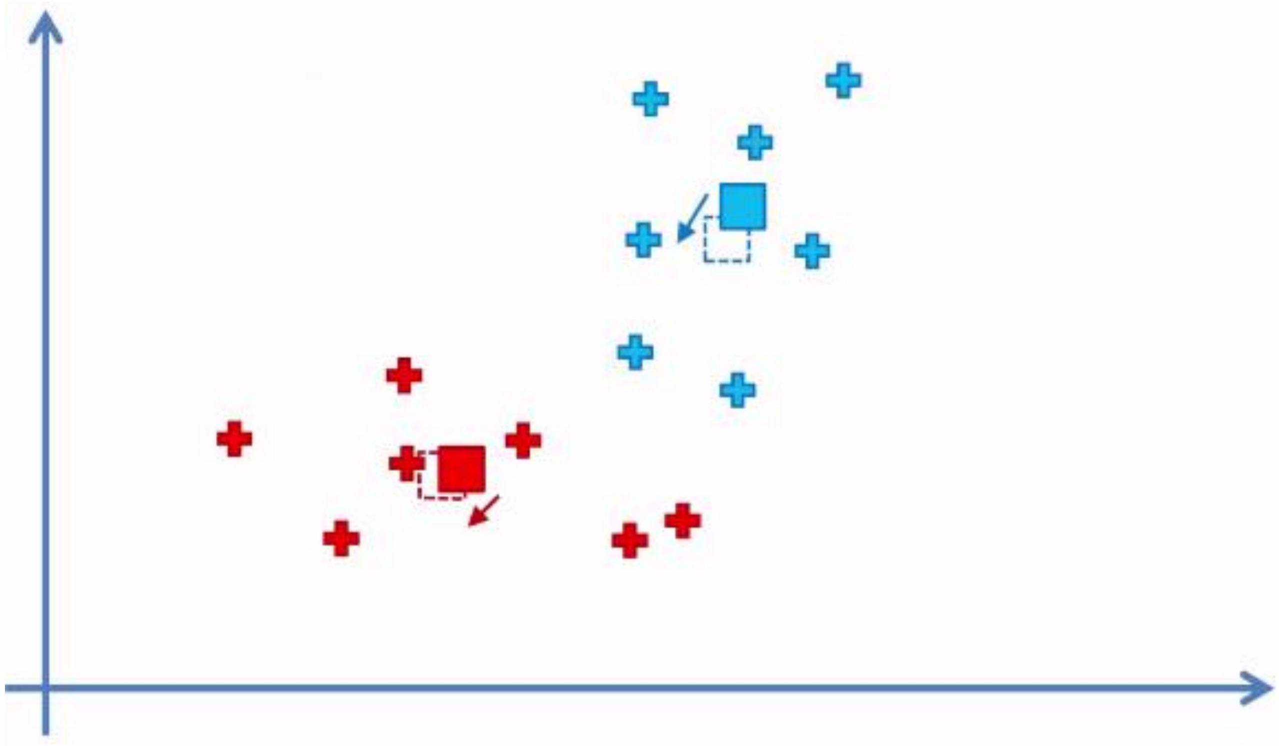
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



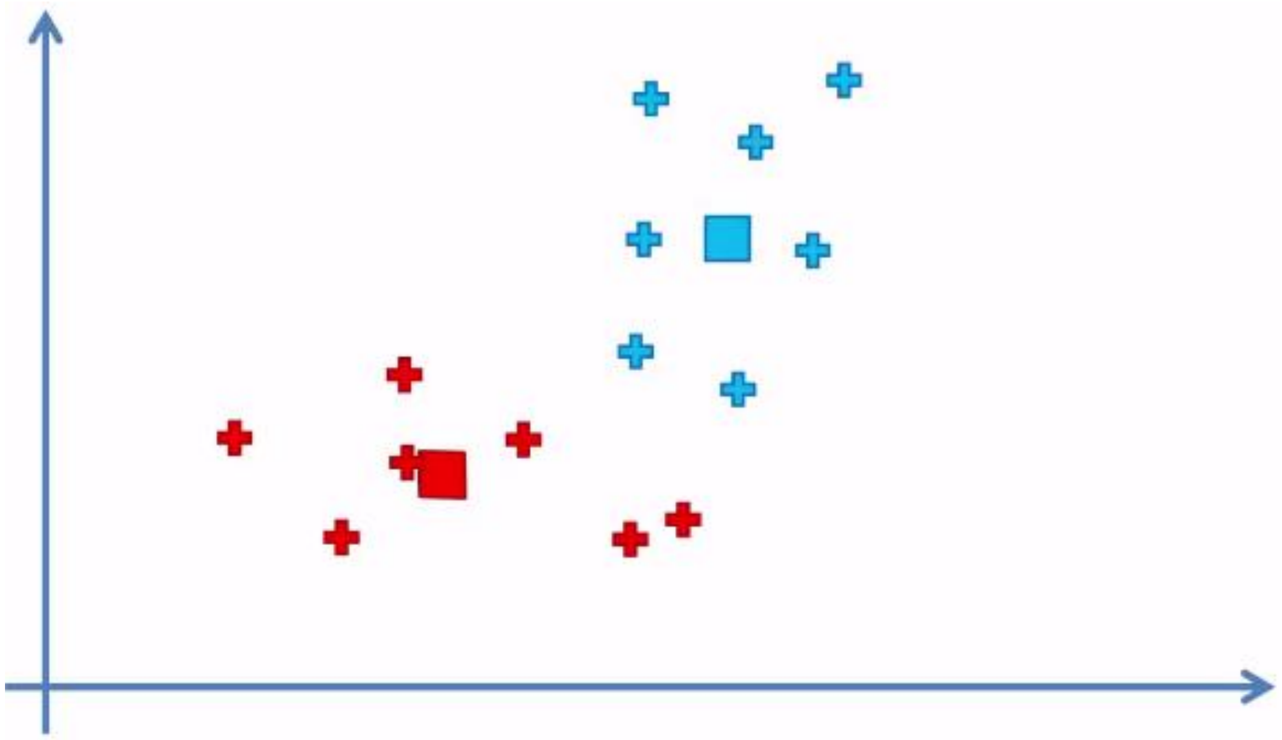
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



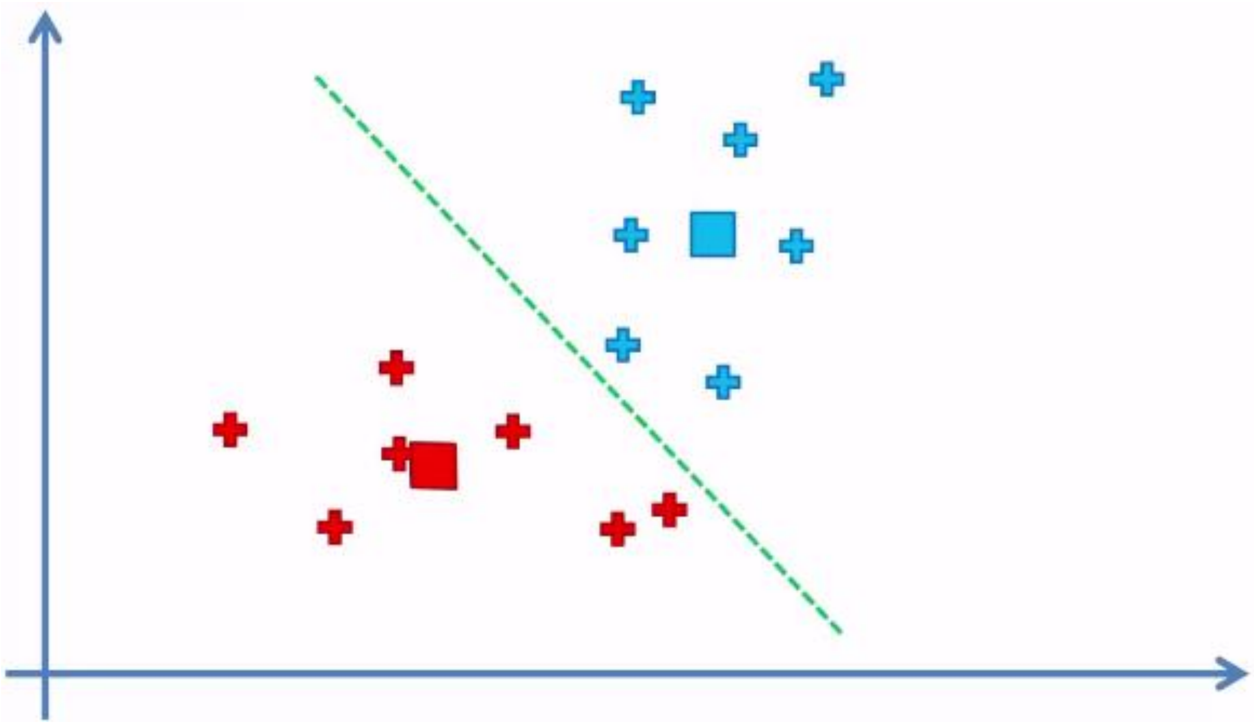
Step-4: Compute and place the new centroid of each cluster



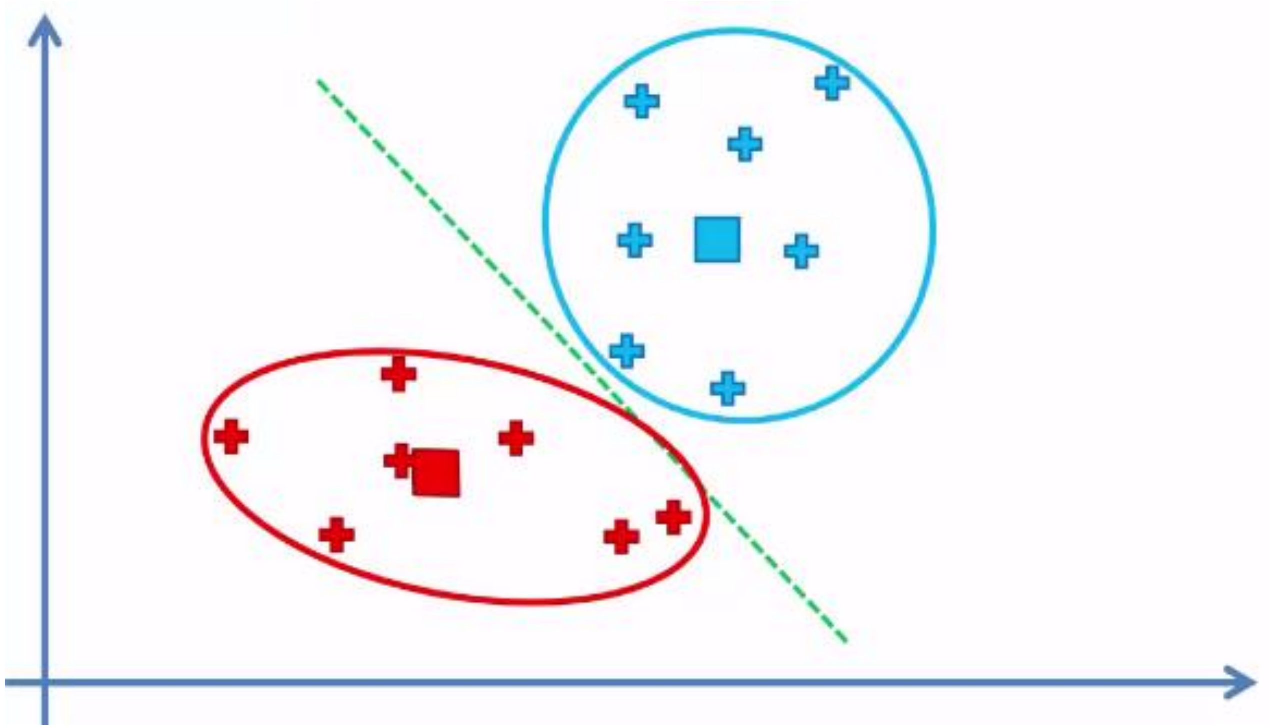
Step-4: Compute and place the new centroid of each cluster



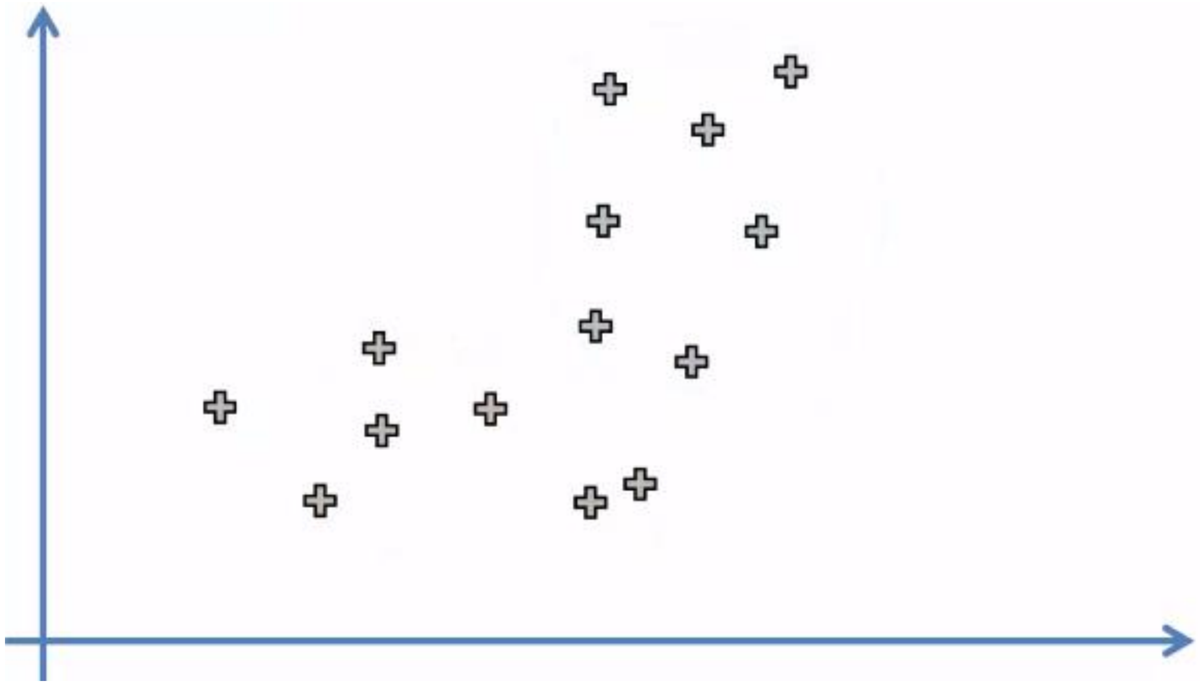
Step-5: Reassign each data points to the new closest centroid.
If any reassignment took place, go to Step-4, otherwise go to Stop.



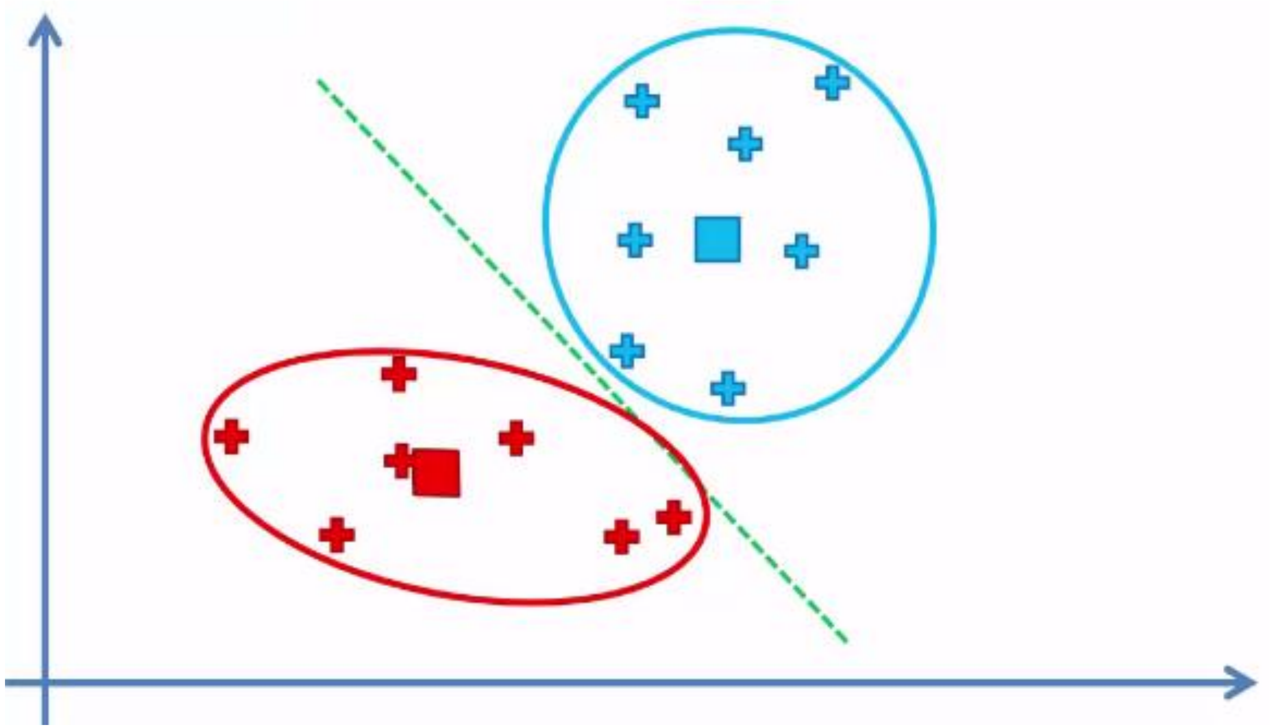
Stop: your model is ready



Step-2: Select at random k points, the centroids (not necessarily from your dataset)



Stop: your model is ready



K-Means Clustering: Example

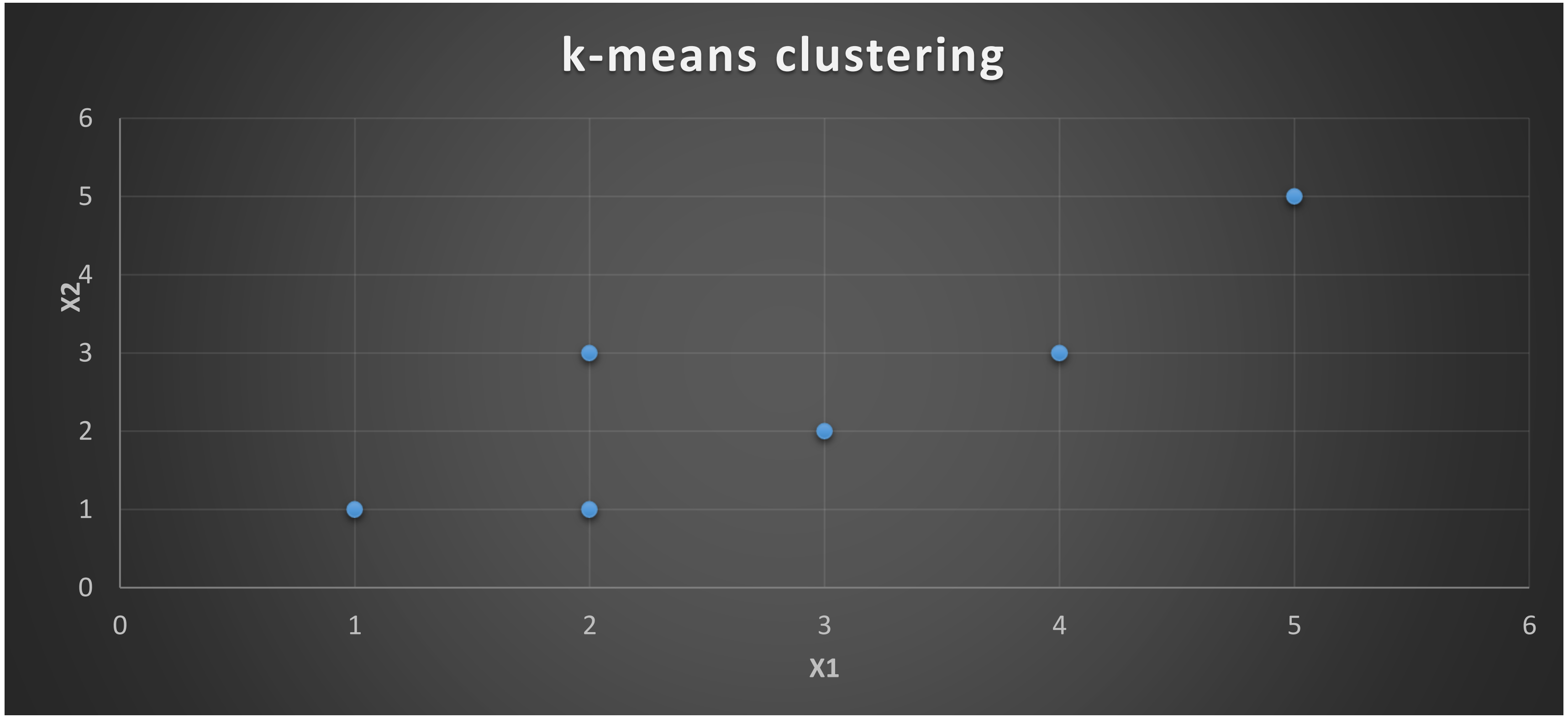
- These objects belong to two groups of medicine.

- Determine which medicines belong to cluster 1 and which medicines belong to the other cluster 2.

Objects	Attribute 1 (X):weight index	Attribute 2 (Y): pH
Medicine A	**1**	**1**
Medicine B	**2**	**1**
Medicine C	**4**	**3**
Medicine D	**5**	**4**

Use k-means clustering algorithm to divide the following data points into 2 clusters

X1	1	2	2	3	4	5
X2	1	1	3	2	3	5



Day 2
Clustering

Choosing right centroid

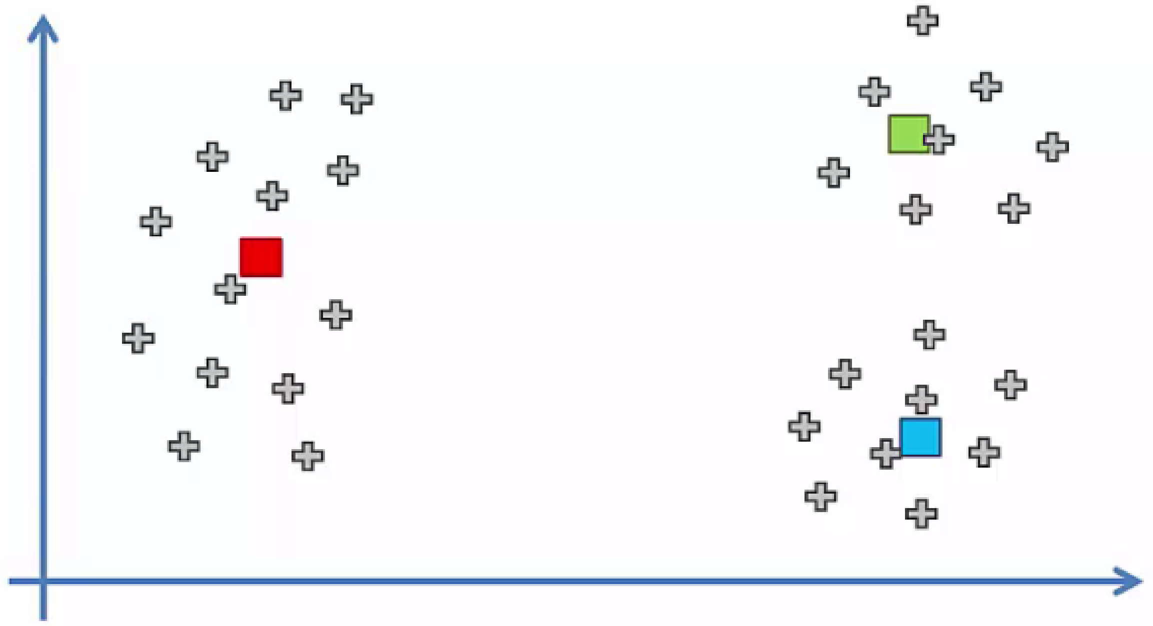
WCSS – elbow method

K-means code

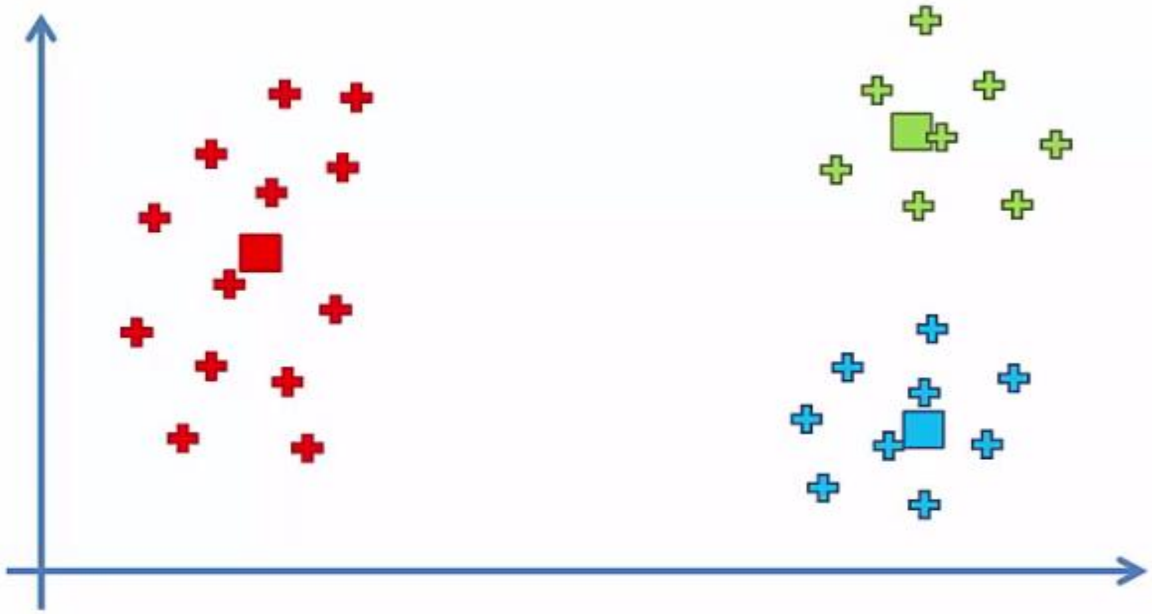
Random Initialization of Centroids

If we choose K = 3 clusters...

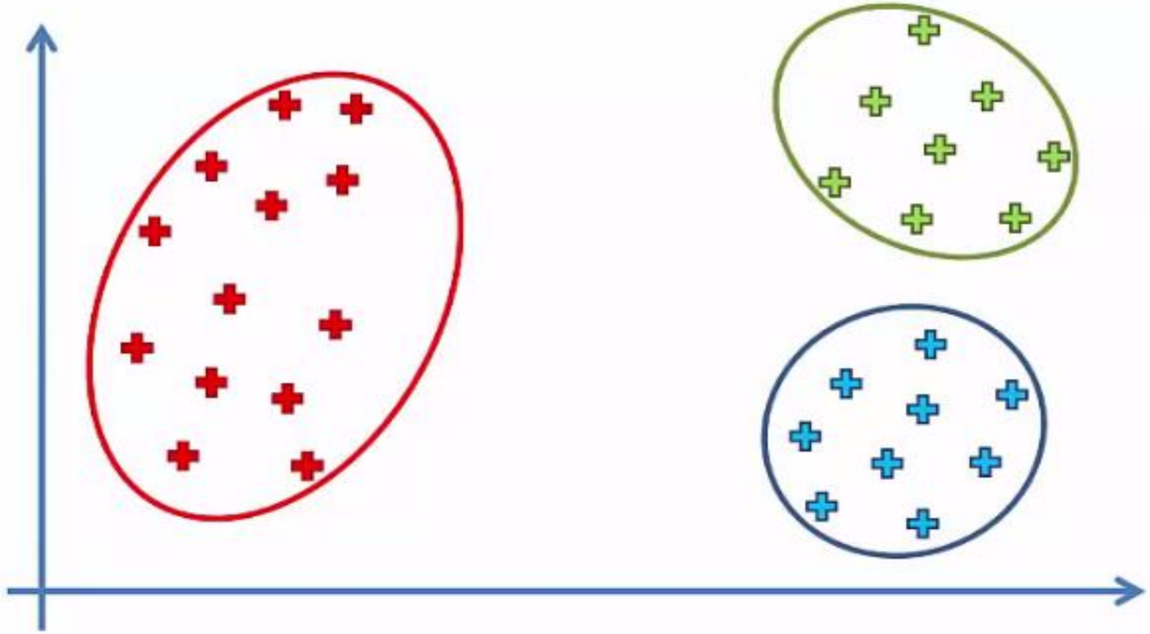
If I chose correct random initialization of centroids …



…this correct random initialisation would lead us to…



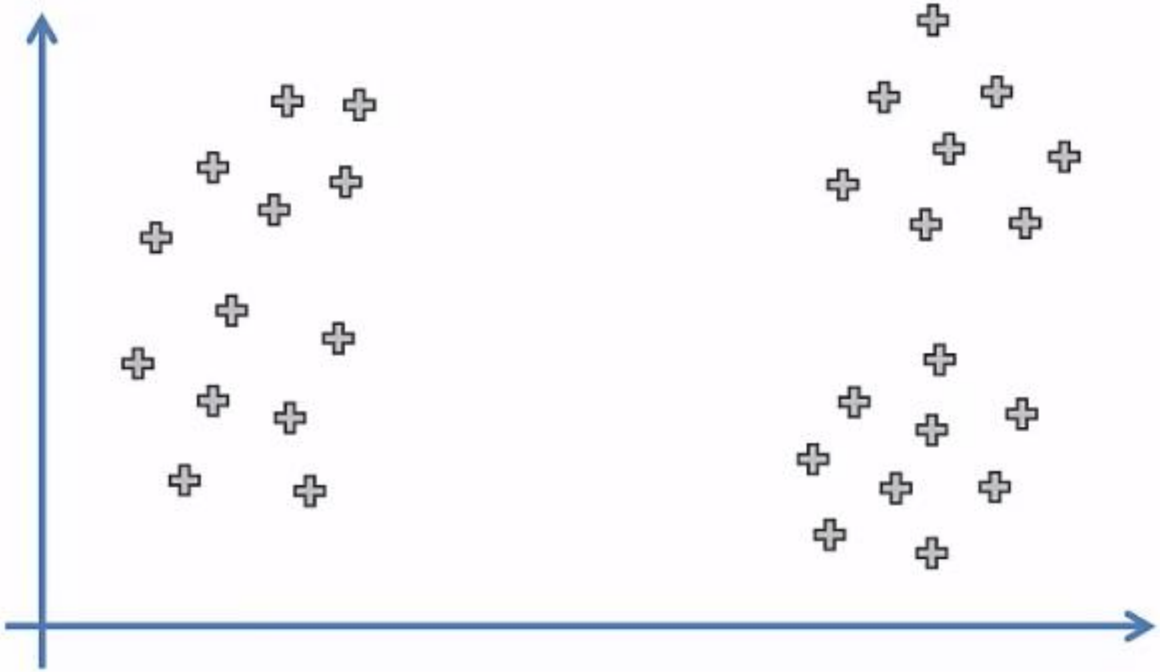
...the following three clusters



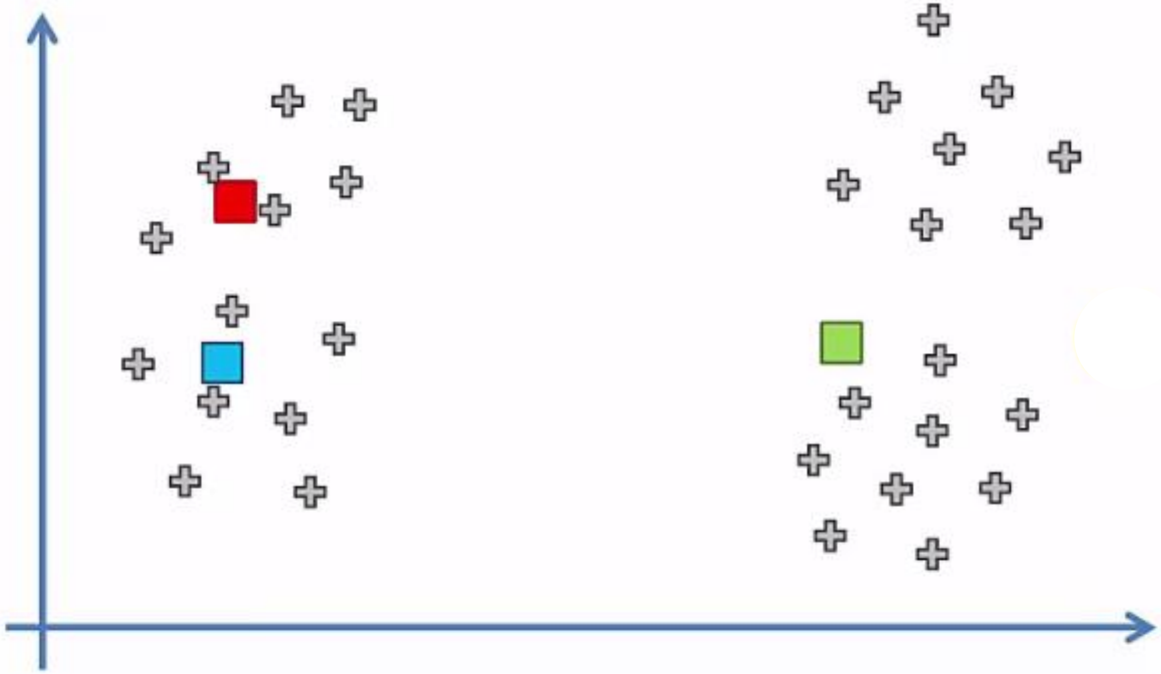
...the following three clusters

What will happen if we choose a bad random initialization?

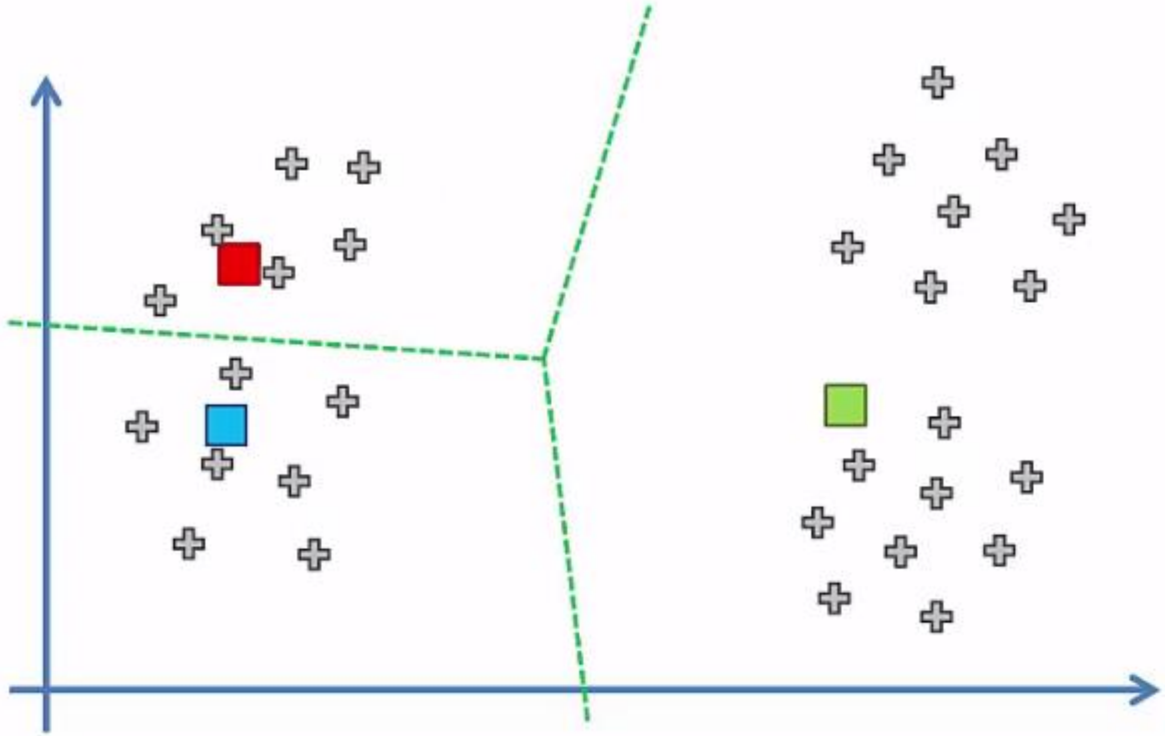
STEP 1: Choose the number K of clusters: K = 3



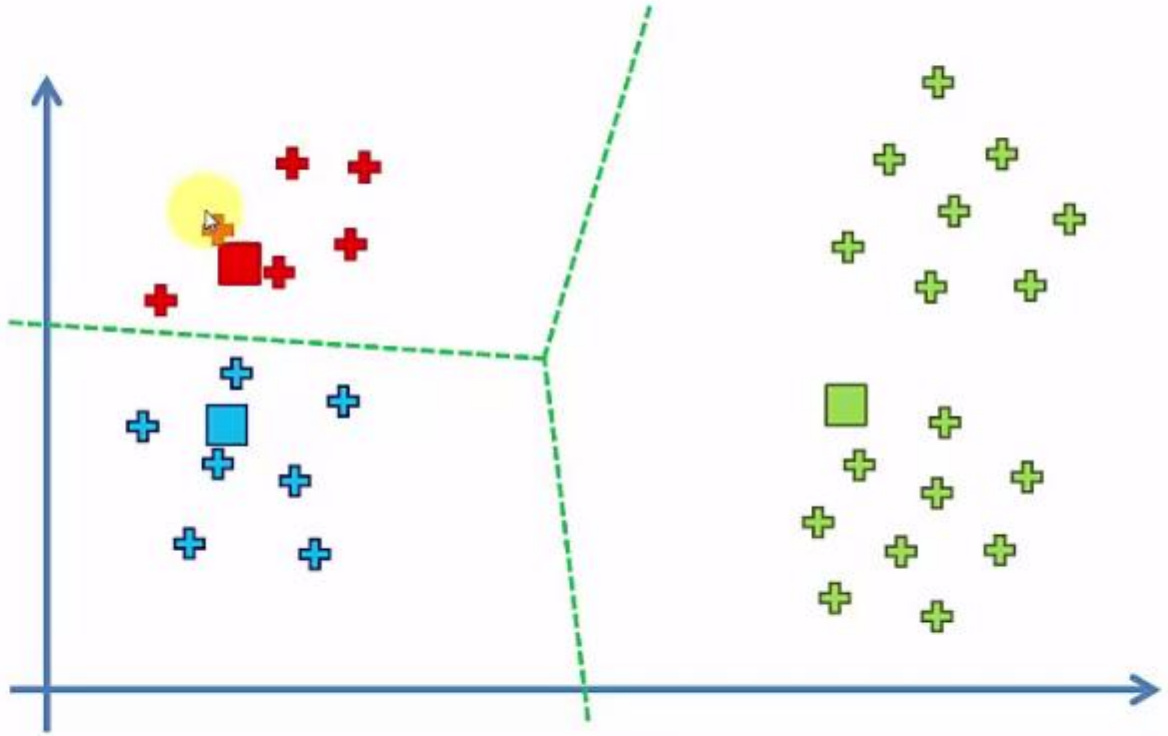
STEP 2: Select at random K points, the centroids (not necessarily from your dataset)



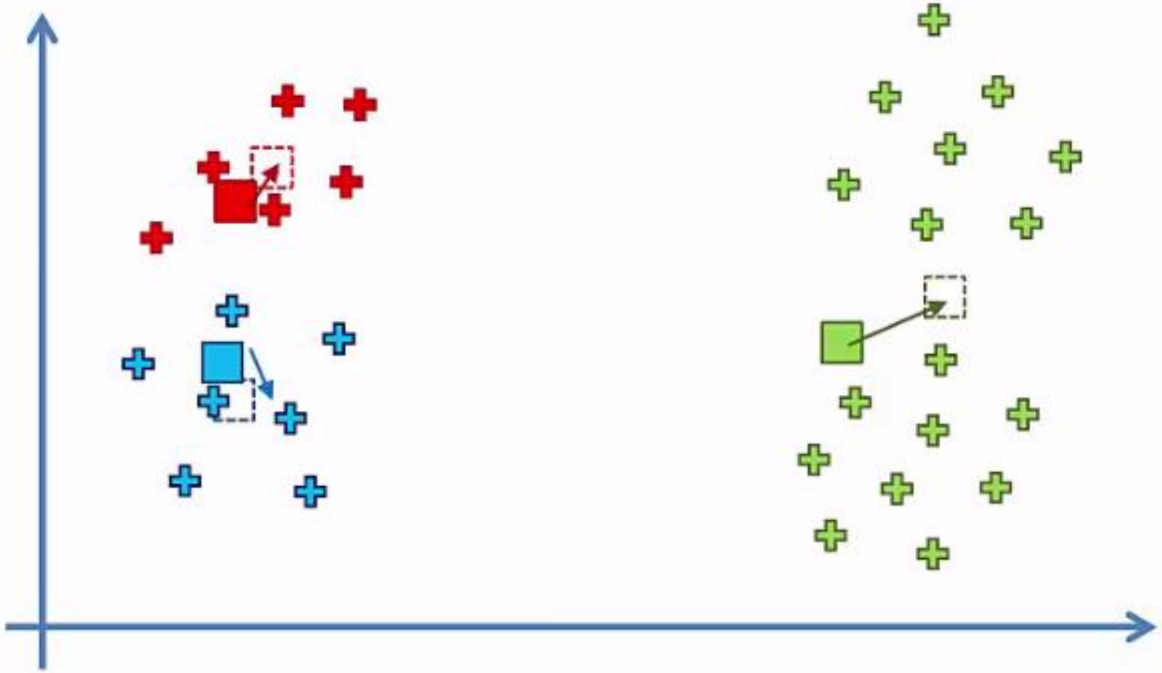
STEP 2: Select at random K points, the centroids (not necessarily from your dataset)



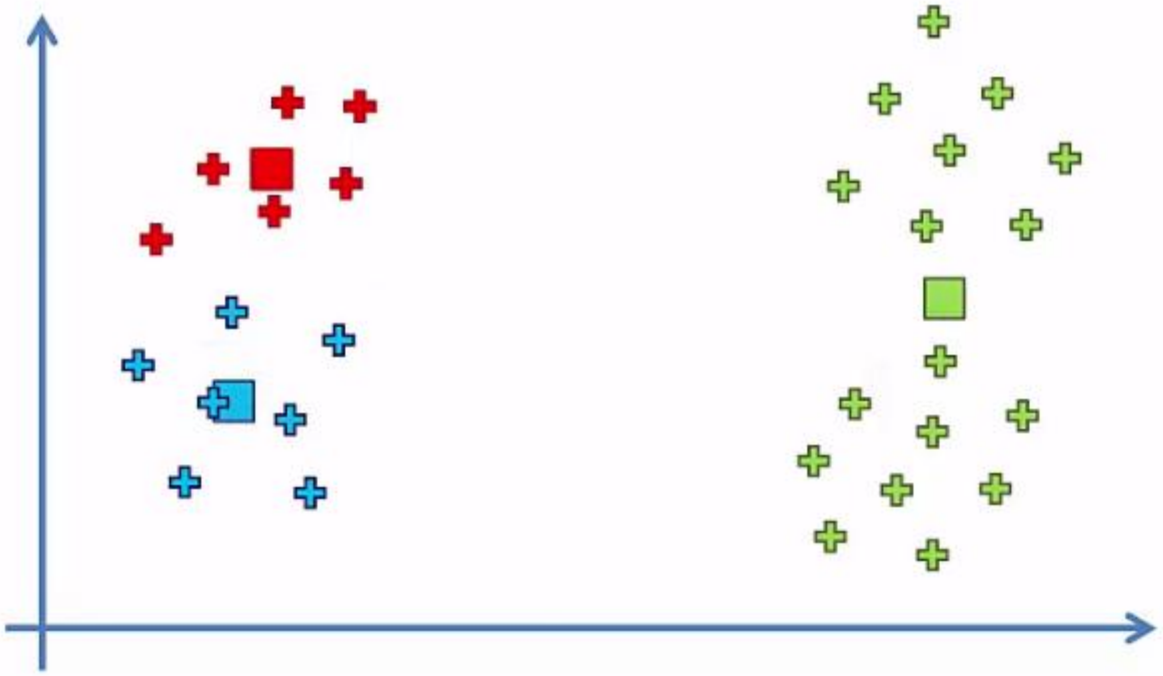
STEP 3: Assign each data point to the closest centroid ➡ That forms K clusters



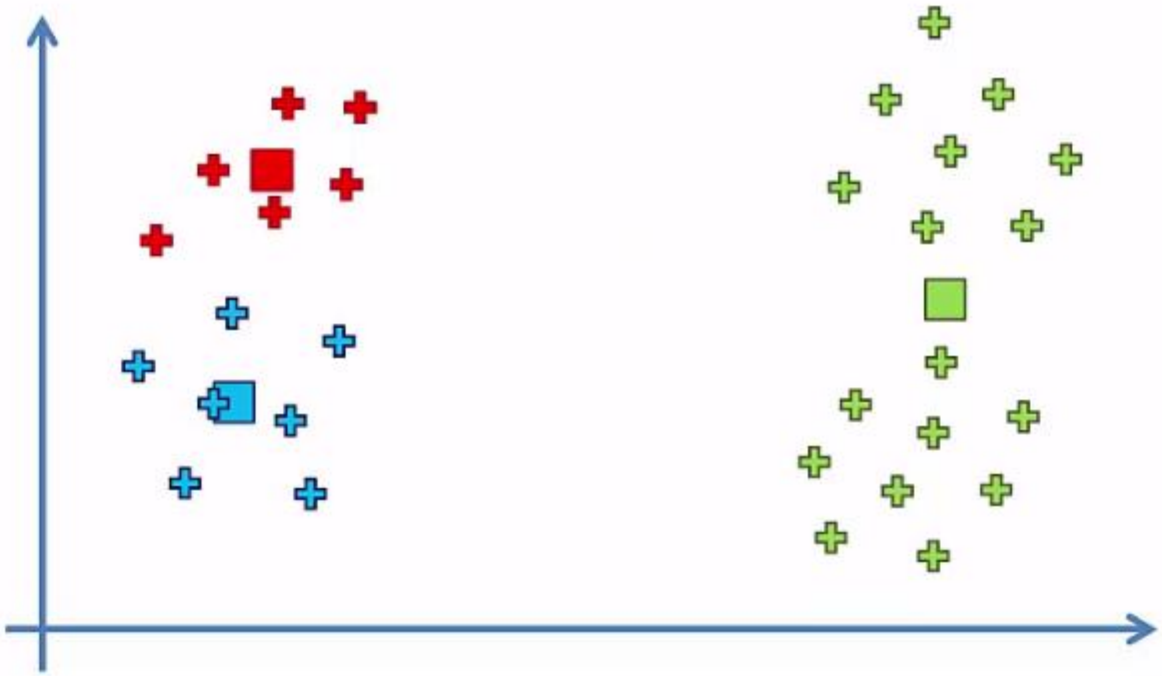
STEP 3: Assign each data point to the closest centroid ➡ That forms K clusters



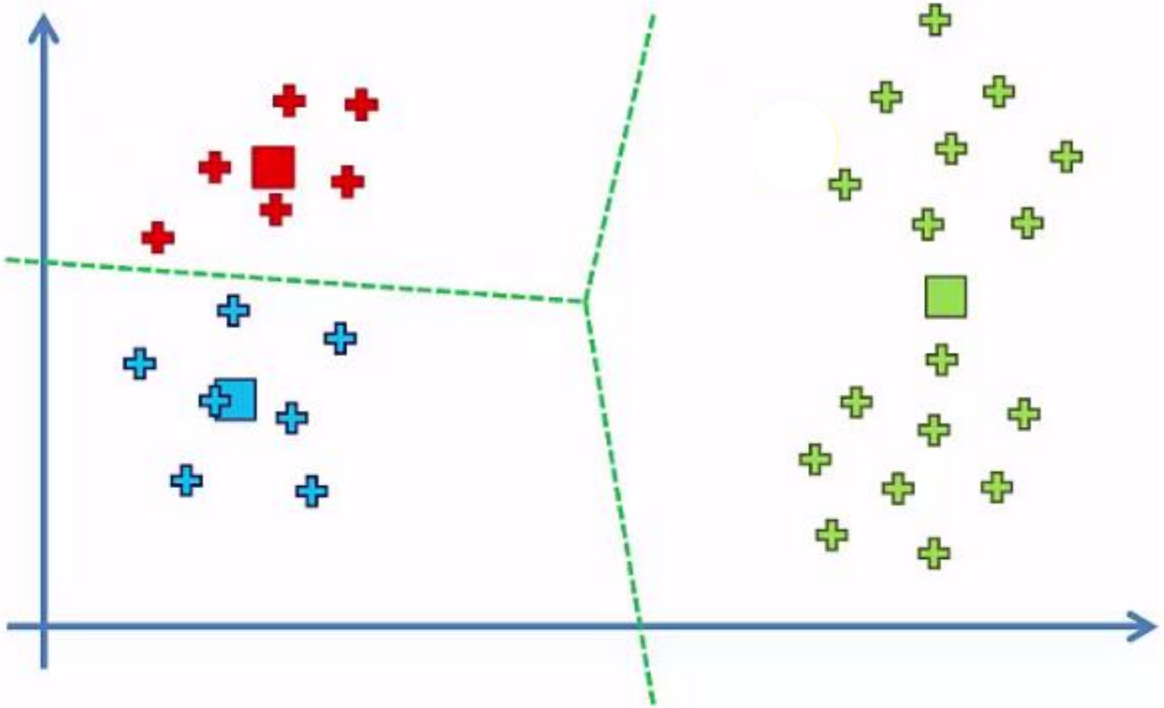
STEP 4: Compute and place the new centroid of each cluster



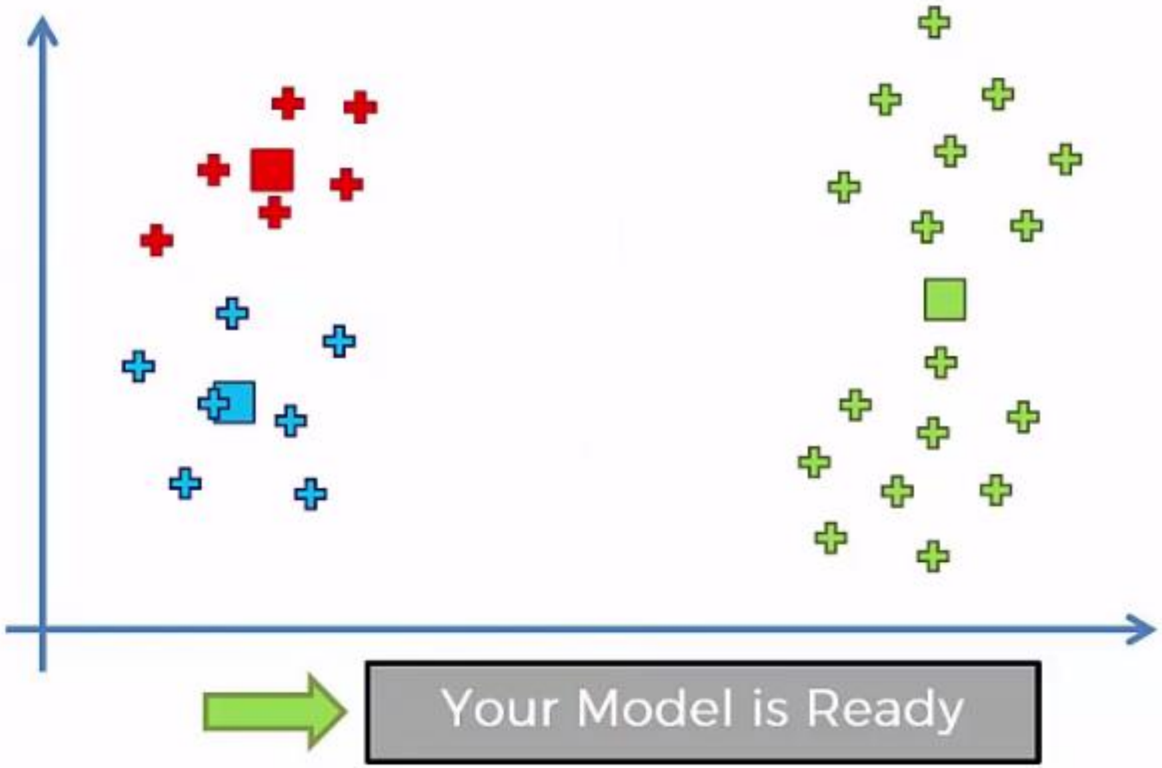
STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.



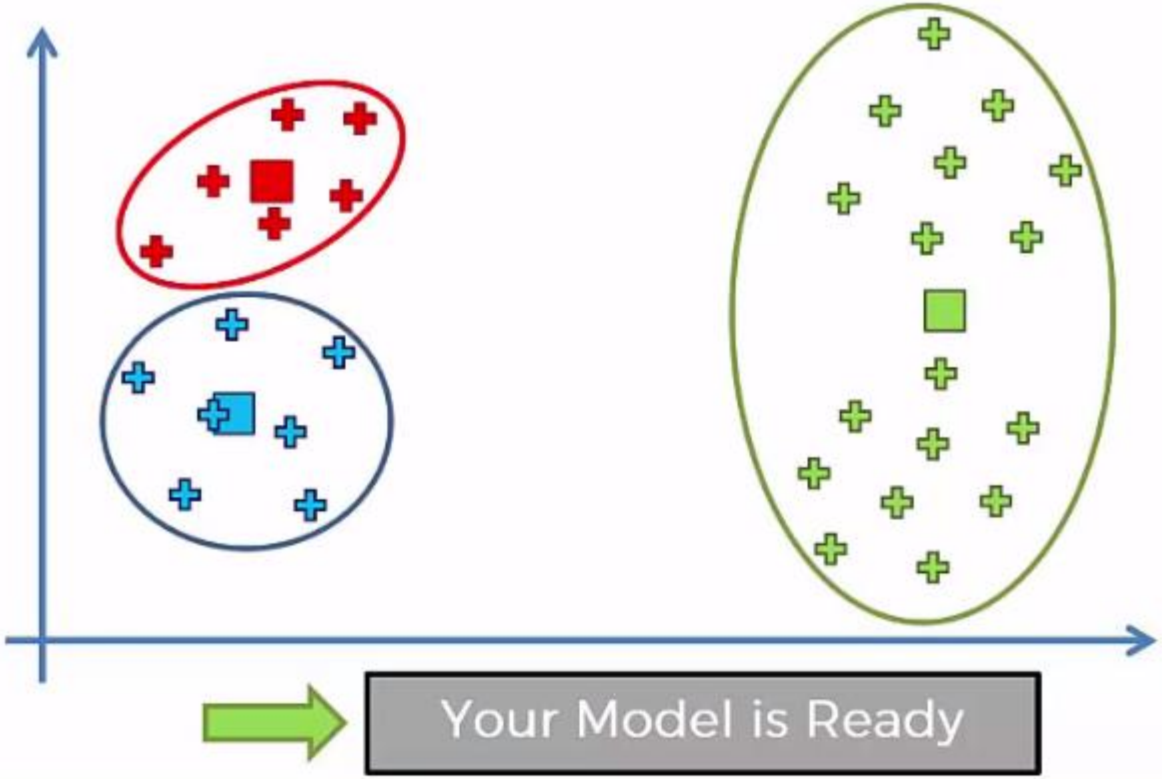
STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.



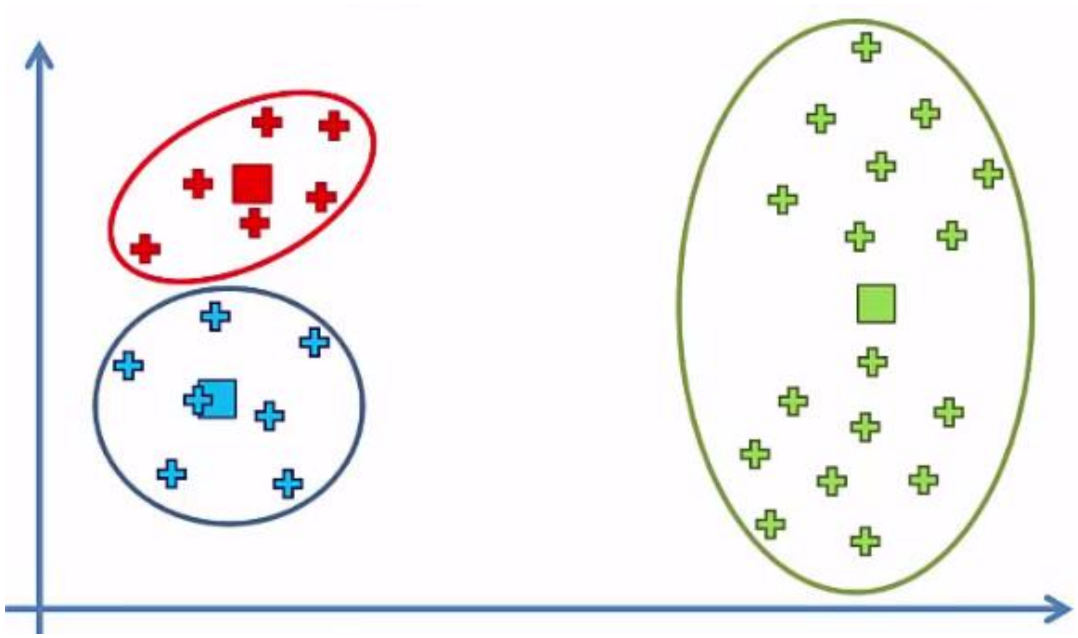
STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.



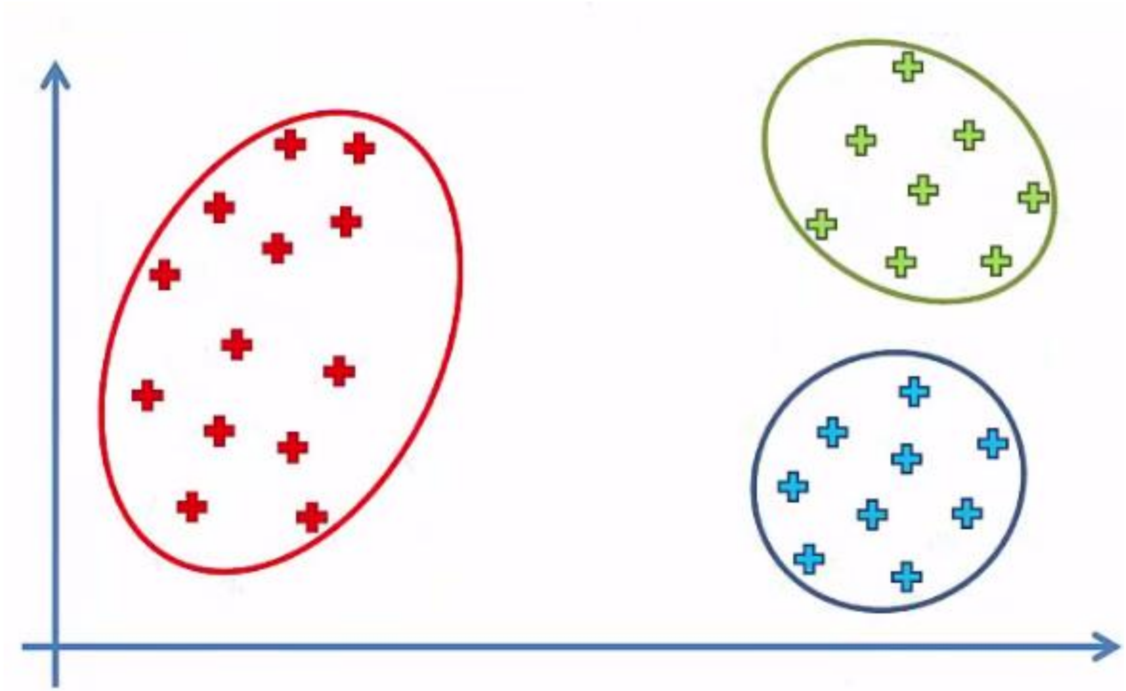
STEP 5: Reassign each data point to the new closest centroid.
If any reassignment took place, go to STEP 4, otherwise go to FIN.



Incorrect one



Correct one

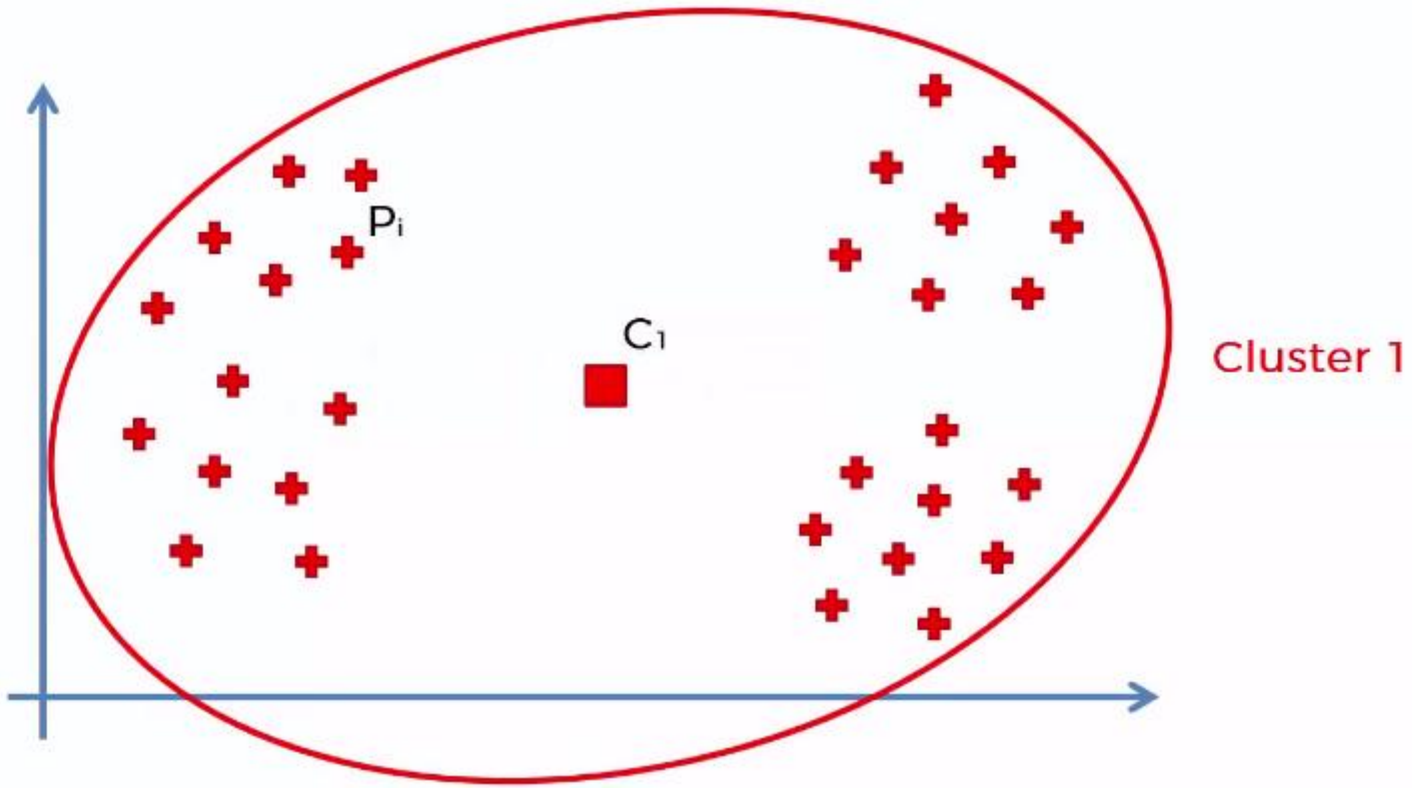


Solution → K-Means++

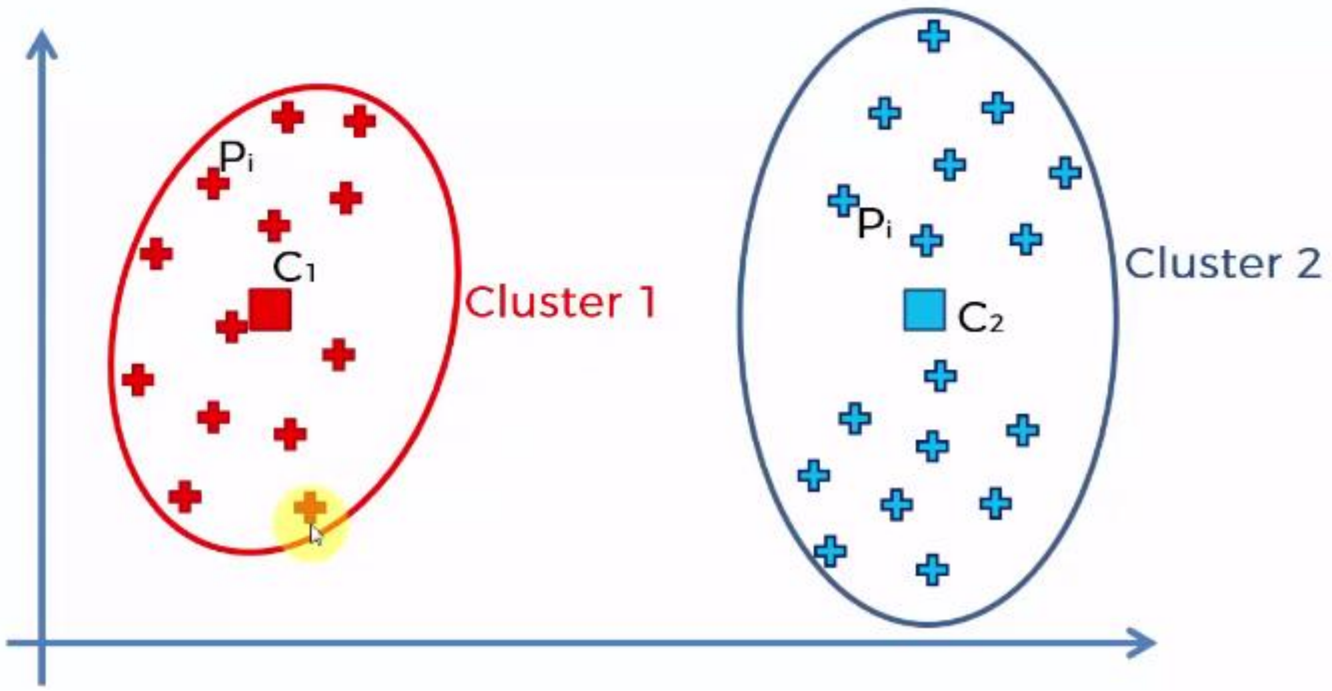
Choosing the Right Number of Clusters

→ Within Cluster Sum of Square (WCSS)

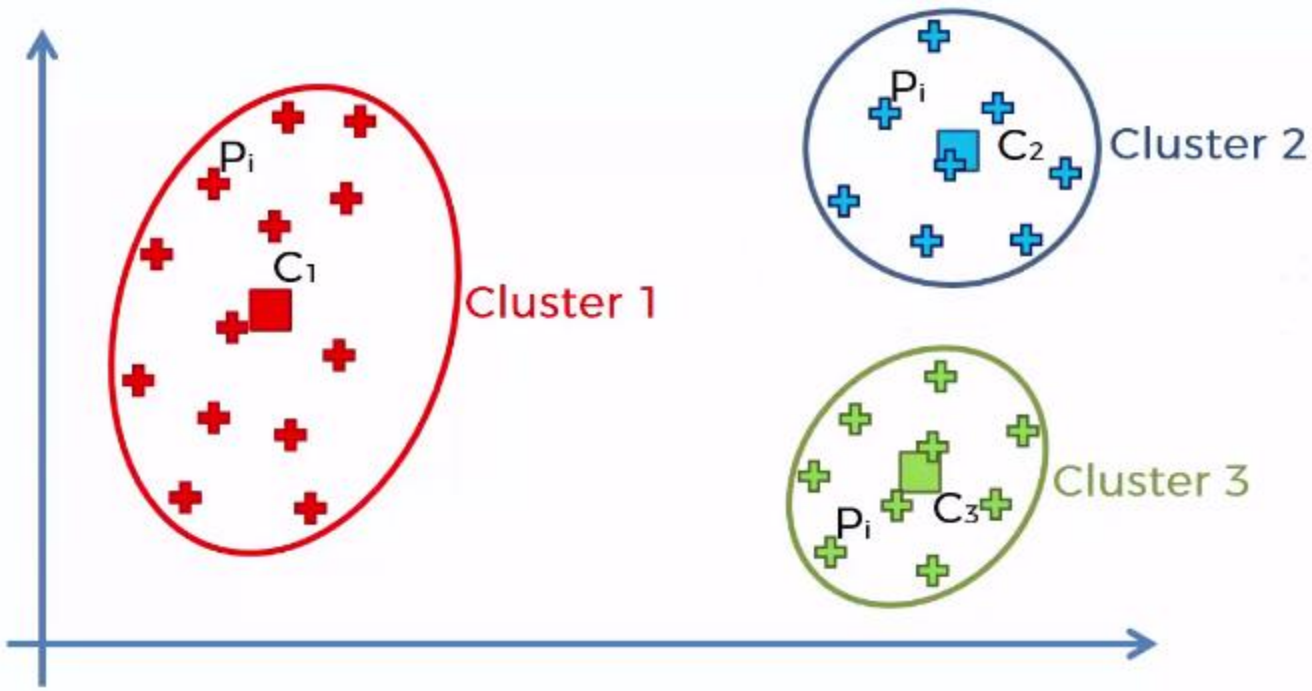
Let us consider the following scenarios…



$$\mathbf{WCSS_1} = \sum_{\mathbf{p_i\ in\ cluster\ 1}} \mathbf{distance\ (P_i, C1)^2}$$

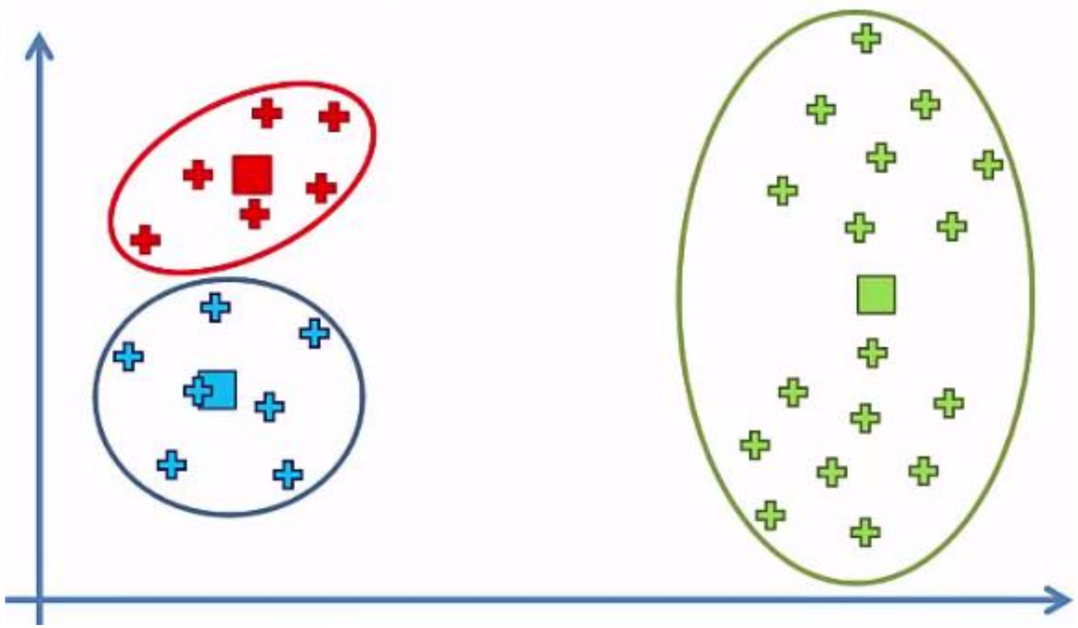


$$\text{WCSS} = \sum_{p_i \text{ in cluster } 1} \text{distance}\,(P_i, C1)^2 + \sum_{p_i \text{ in cluster } 2} \text{distance}\,(P_i, C_2)^2$$

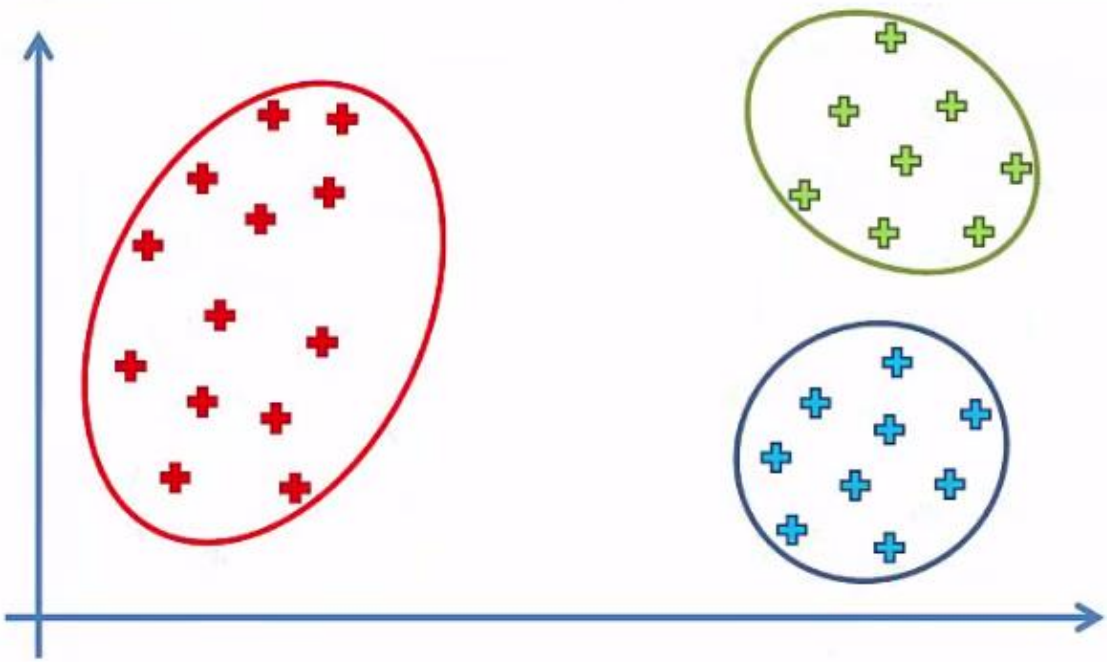


$$\text{WCSS} = \sum_{p_i \text{ in cluster 1}} \text{distance}(P_i, C1)^2 + \sum_{p_i \text{ in cluster 2}} \text{distance}(P_i, C_2)^2 + \sum_{p_i \text{ in cluster 3}} \text{distance}(P_i, C_3)^2$$

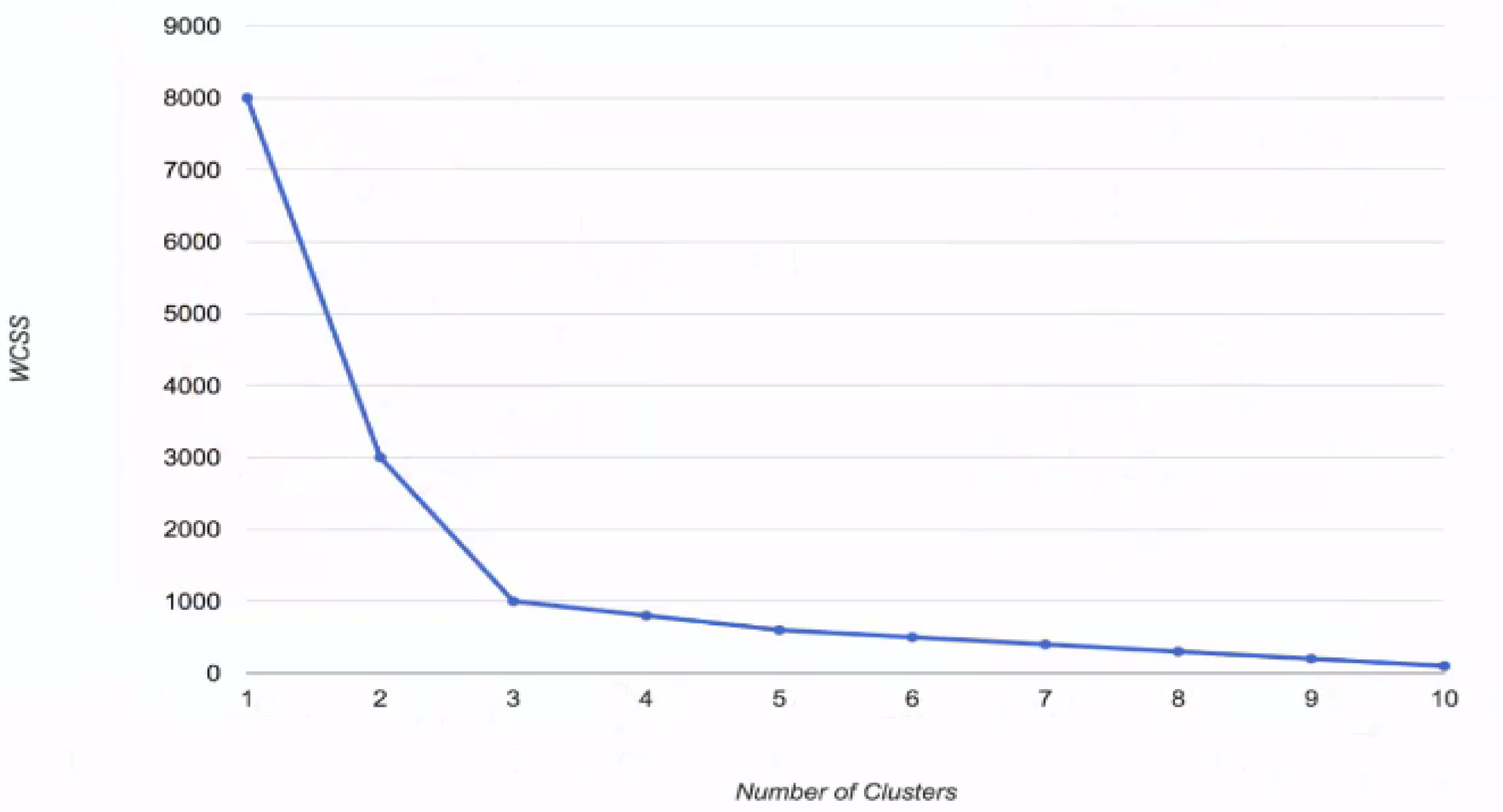
Incorrect one



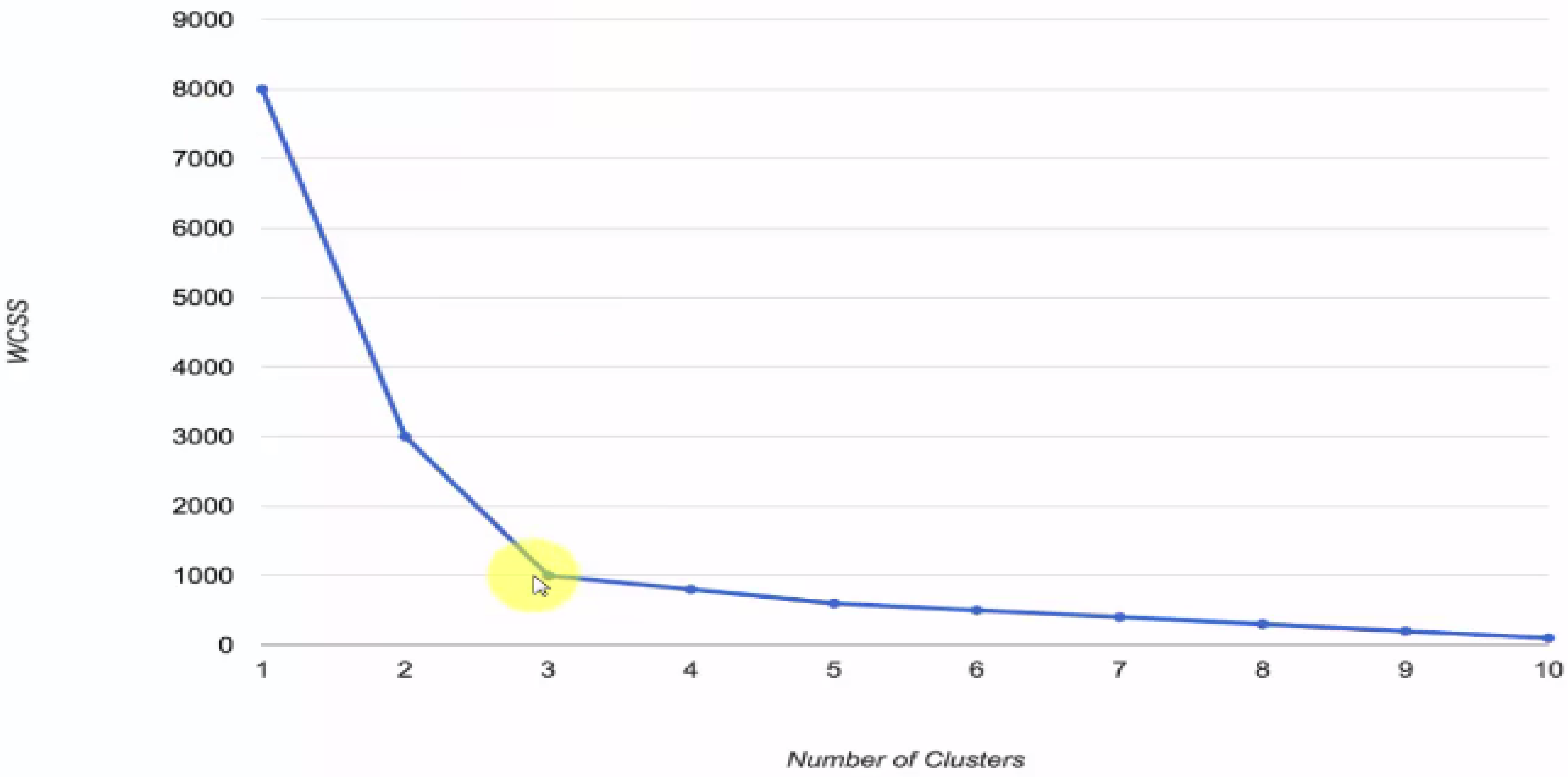
Correct one



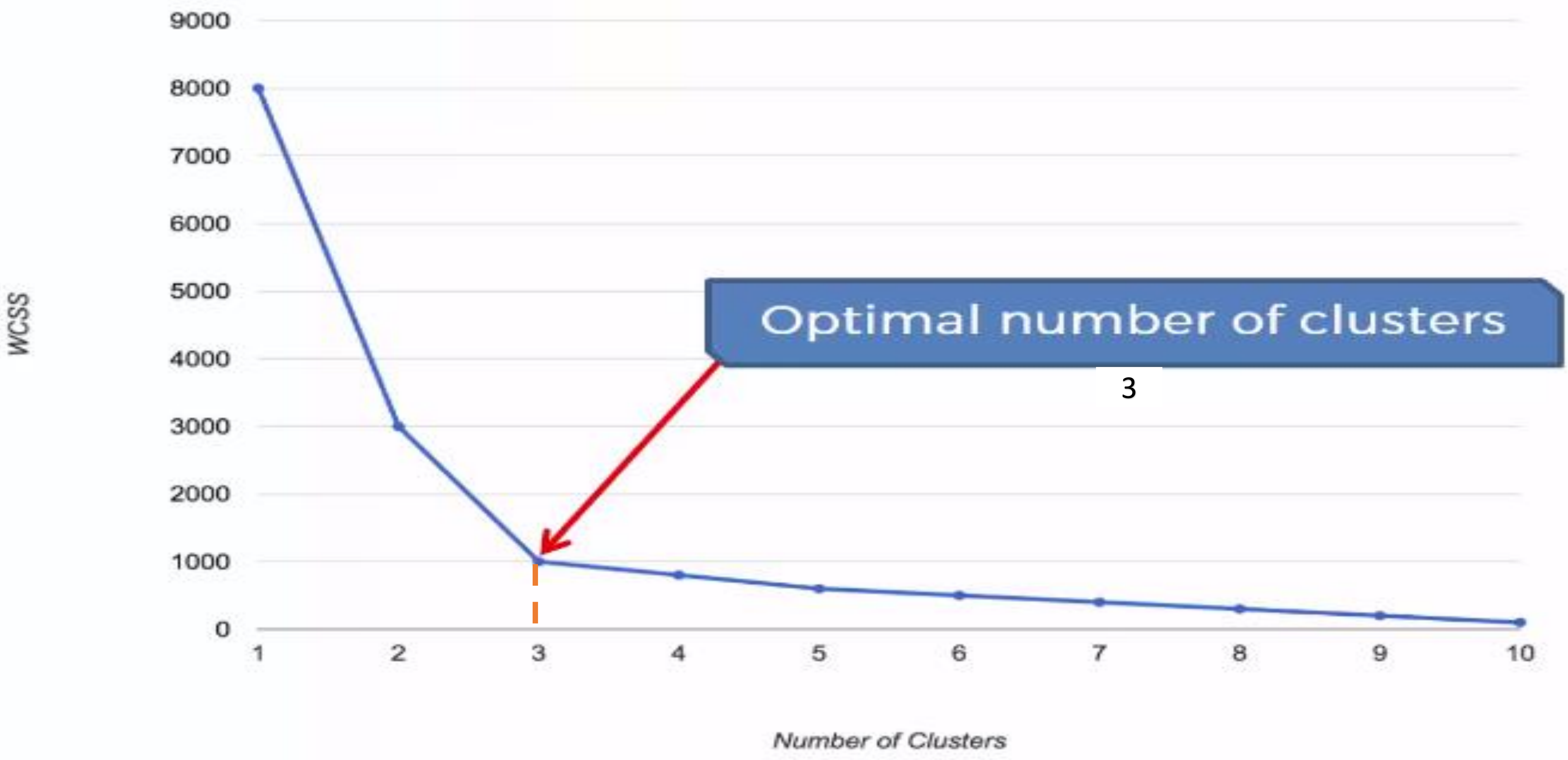
$$WCSS = \sum_{p_i \text{ in cluster } 1} \text{distance}(P_i, C1)^2 + \sum_{p_i \text{ in cluster } 2} \text{distance}(P_i, C_2)^2 + \sum_{p_i \text{ in cluster } 3} \text{distance}(P_i, C_3)^2$$


9000
8000
7000
6000
5000
4000
3000
2000
1000
0
WCSS
1
2
3
4
5
6
7
8
9
10
Number of Clusters

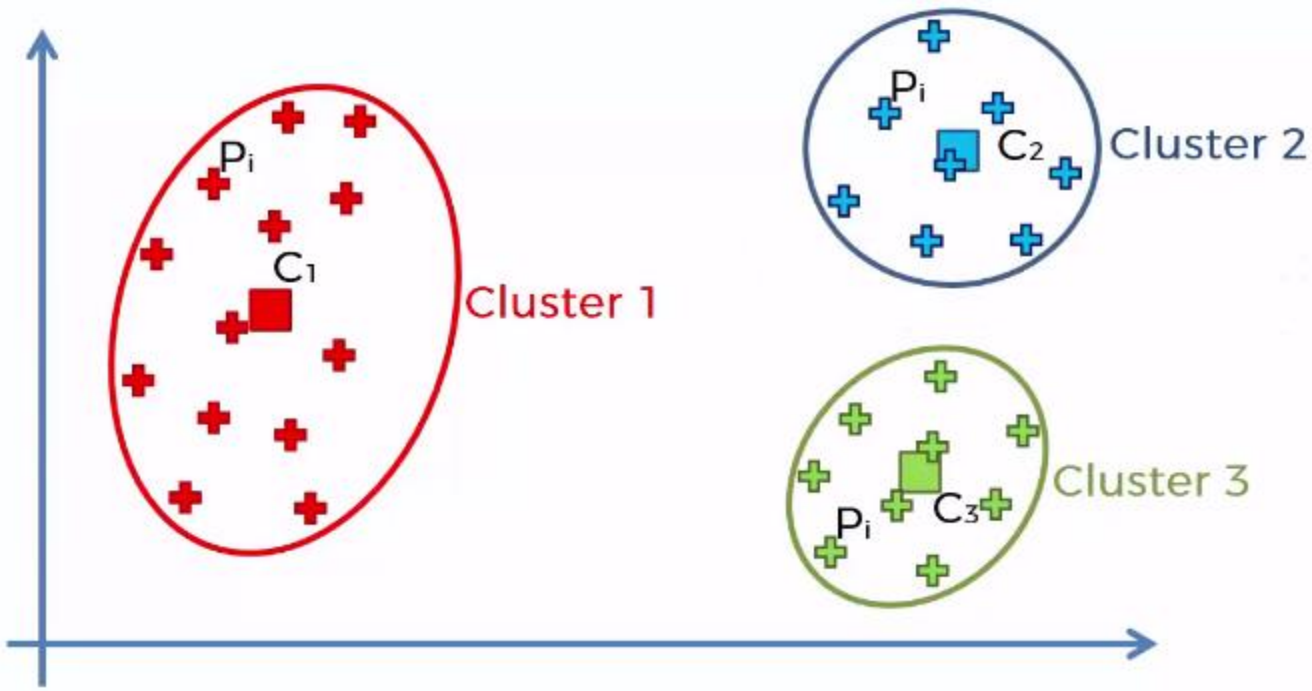
The Elbow Method



The Elbow Method

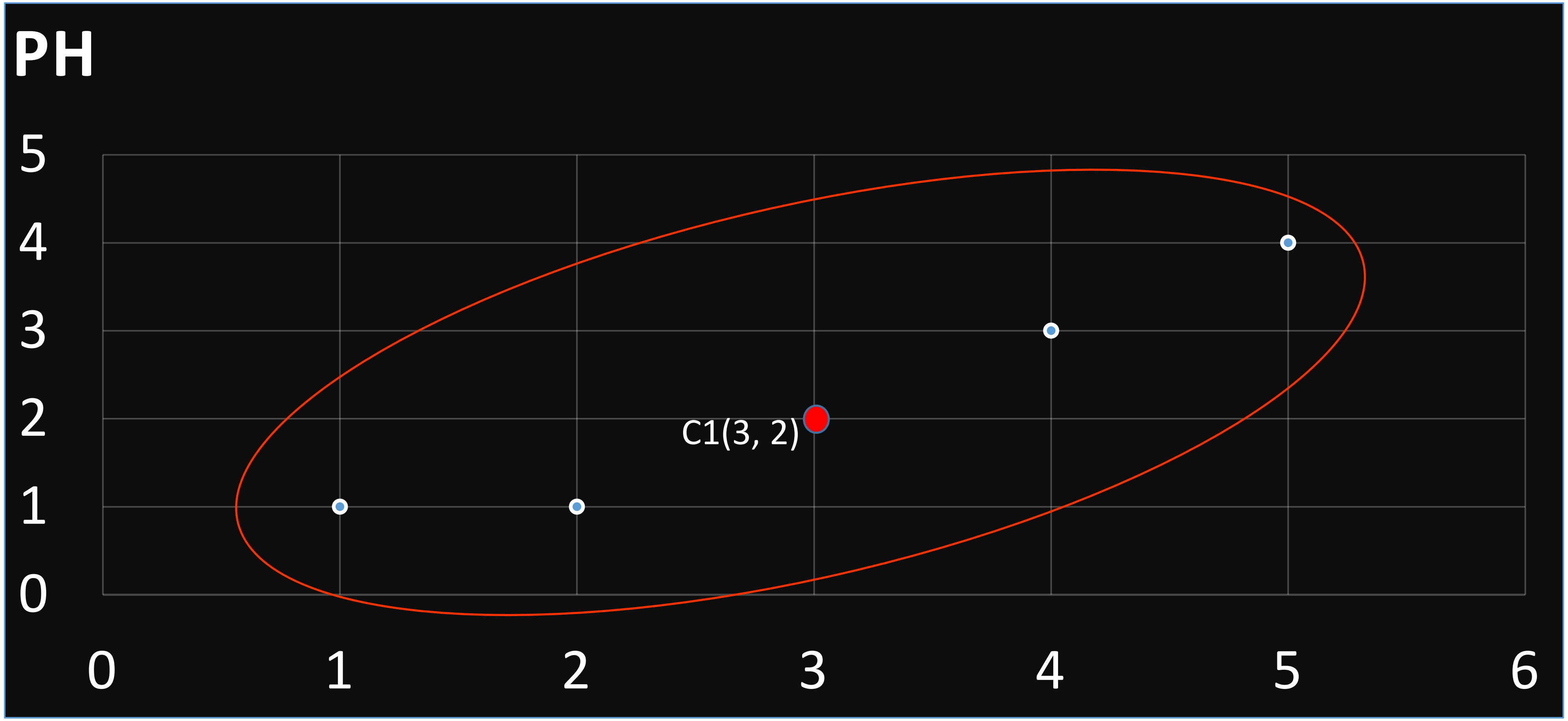


Calculate WCSS

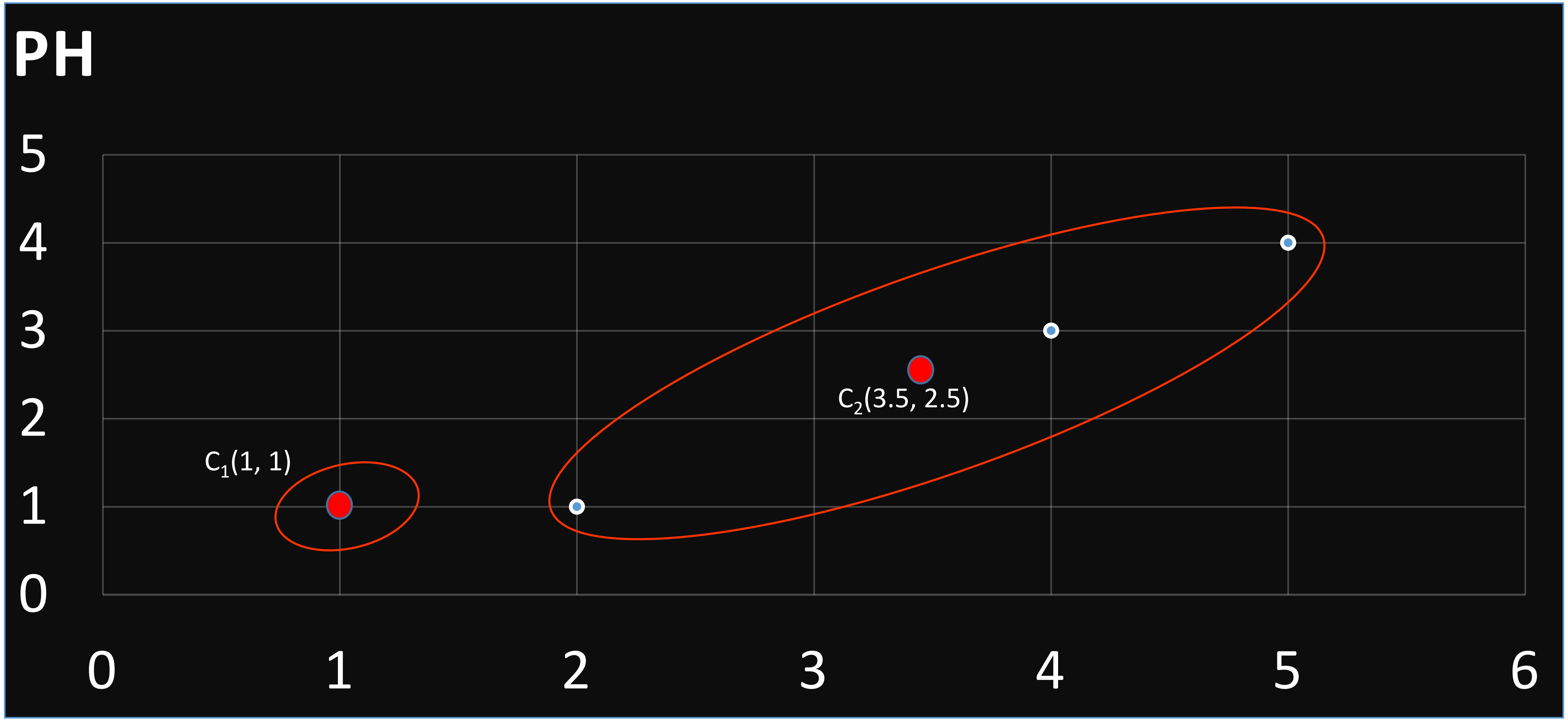


$$\mathbf{WCSS} = \sum_{p_i \text{ in cluster 1}} \text{distance}\,(P_i, C1)^2 + \sum_{p_i \text{ in cluster 2}} \text{distance}\,(P_i, C_2)^2 + \sum_{p_i \text{ in cluster 3}} \text{distance}\,(P_i, C_3)^2$$

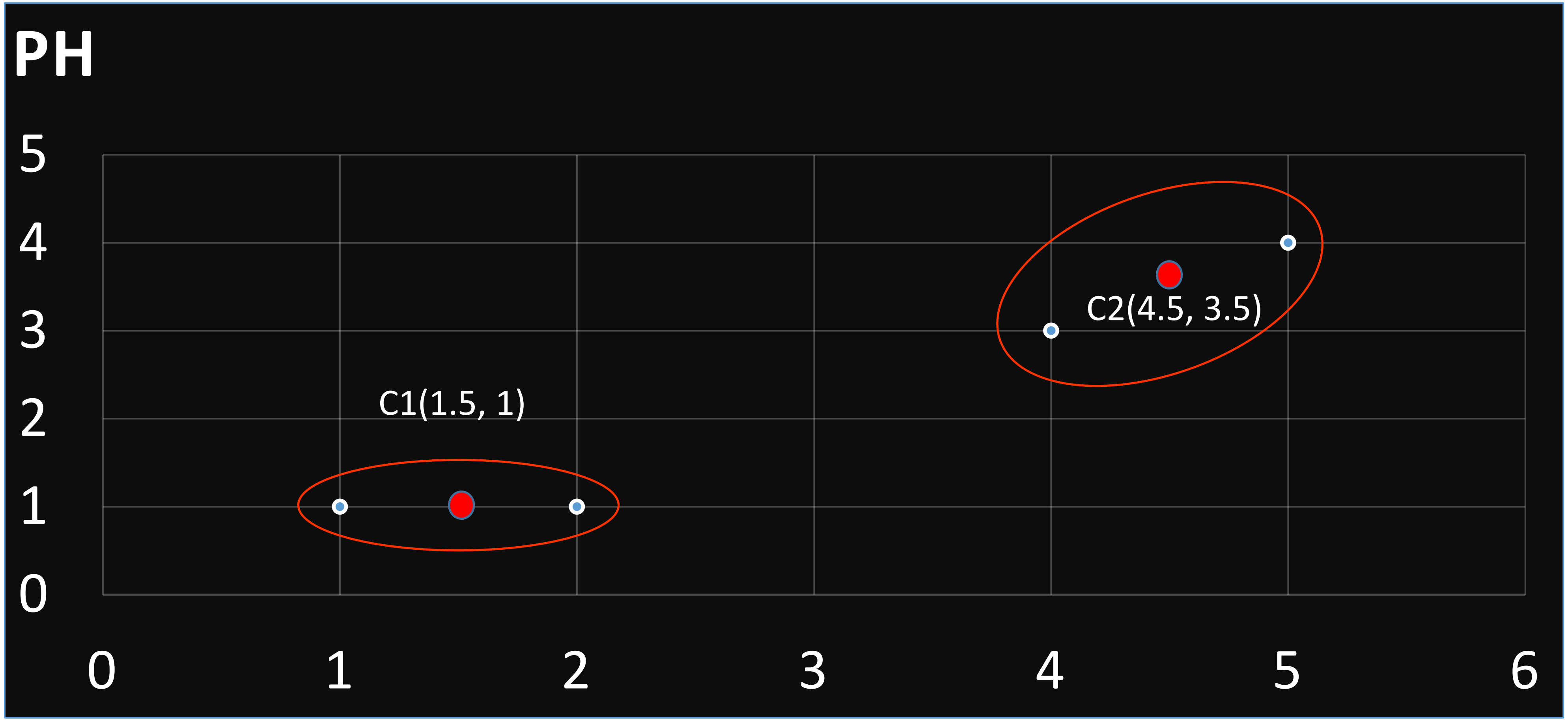
WCSS = 17



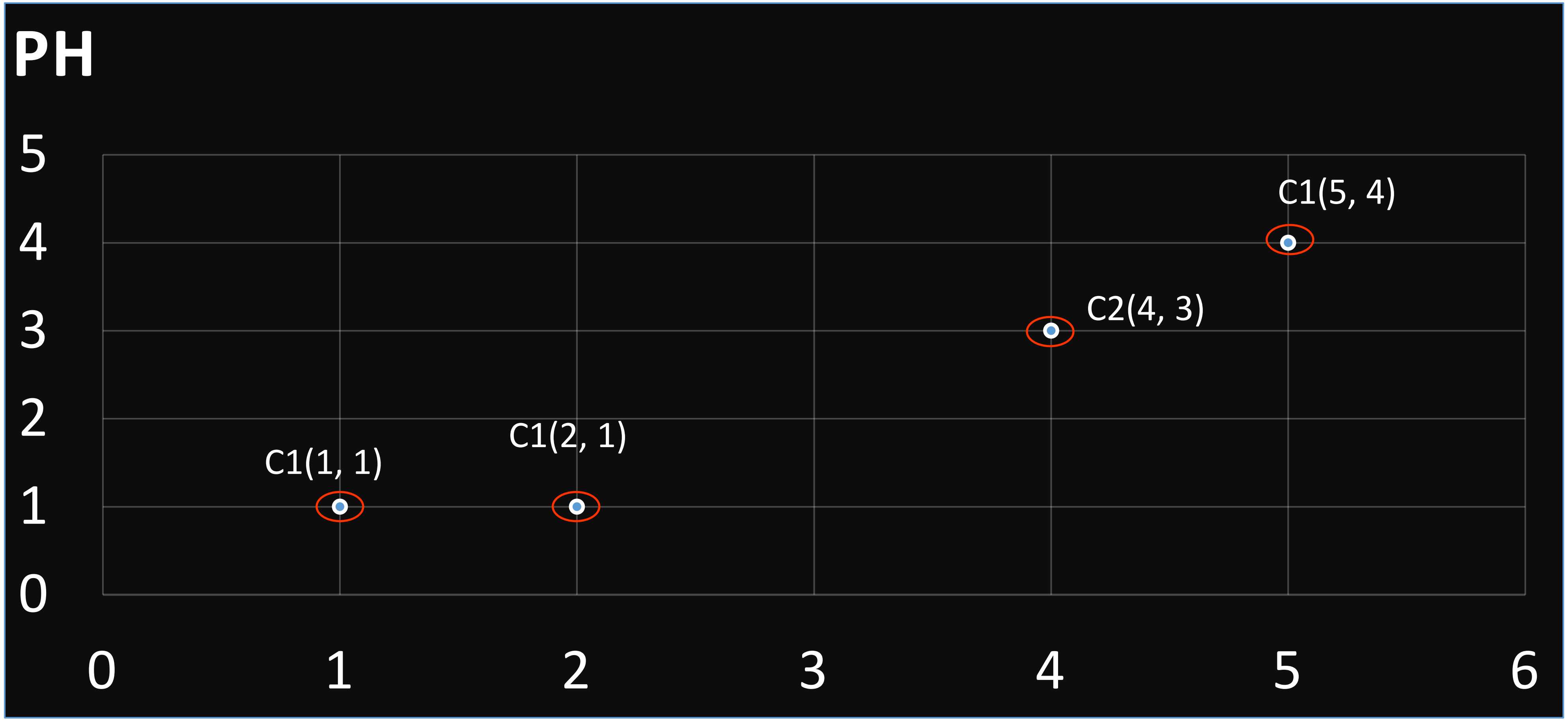
WCSS = 9.5



WCSS = 1.5



WCSS = 0




No of Clusters = 2
18
16
14
12
10
8
6
4
2
0
C1
C2
C3
C4

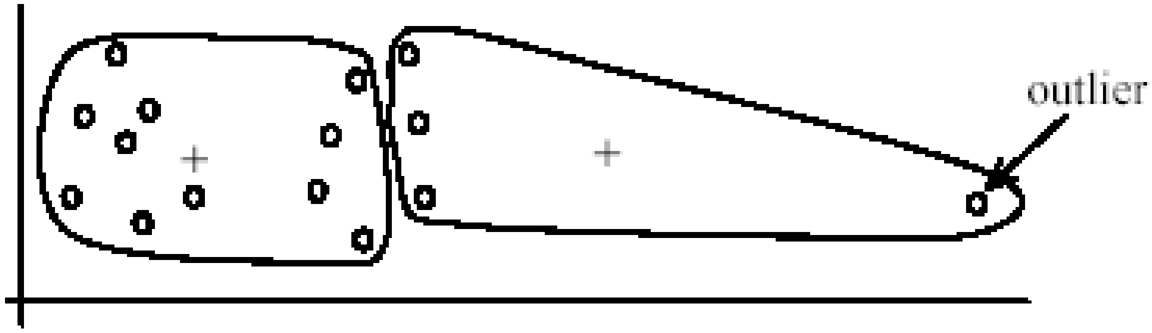
Strengths of K-means

- **Simple**: easy to understand and to implement
- **Efficient:** Time complexity: $O(tkn)$
  - *n*-is the number of data points
  - *k-is the number of clusters*
  - *t- is the number of iterations*
  - Since both *k* and *t* are small. *k*-means is considered a linear algorithm.
- K-means is the most popular clustering algorithm.

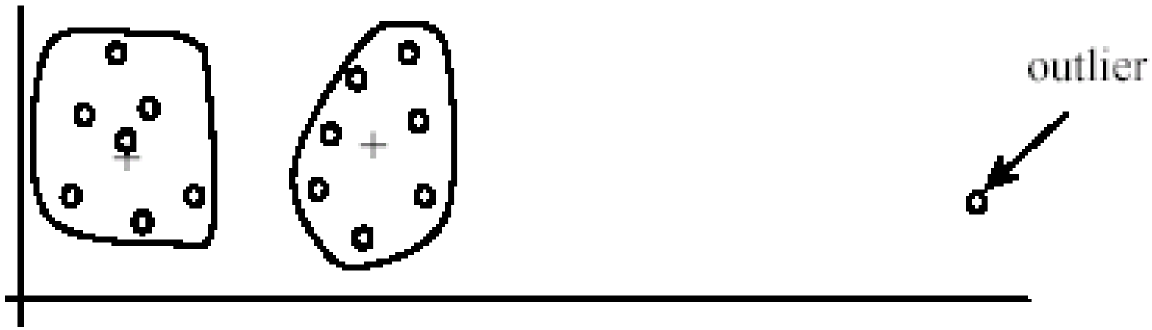
Weaknesses of K-means

- The user needs to specify number of clusters (K).
- The algorithm is sensitive to **outliers**
  - Outliers are data points that are **very far away from other data points**.
  - Outliers could be **errors** in the data recording or some **special data points** with very different values.

Outliers



(A): Undesirable clusters



(B): Ideal clusters

Handling with outliers

- Remove some data points that are much further away from the centroids
- To be safe, monitor these possible outliers over a few iterations and then decide to remove them.

K-means clustering implementation in Python

```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

Create arrays that resemble two variables in a dataset.

```python
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

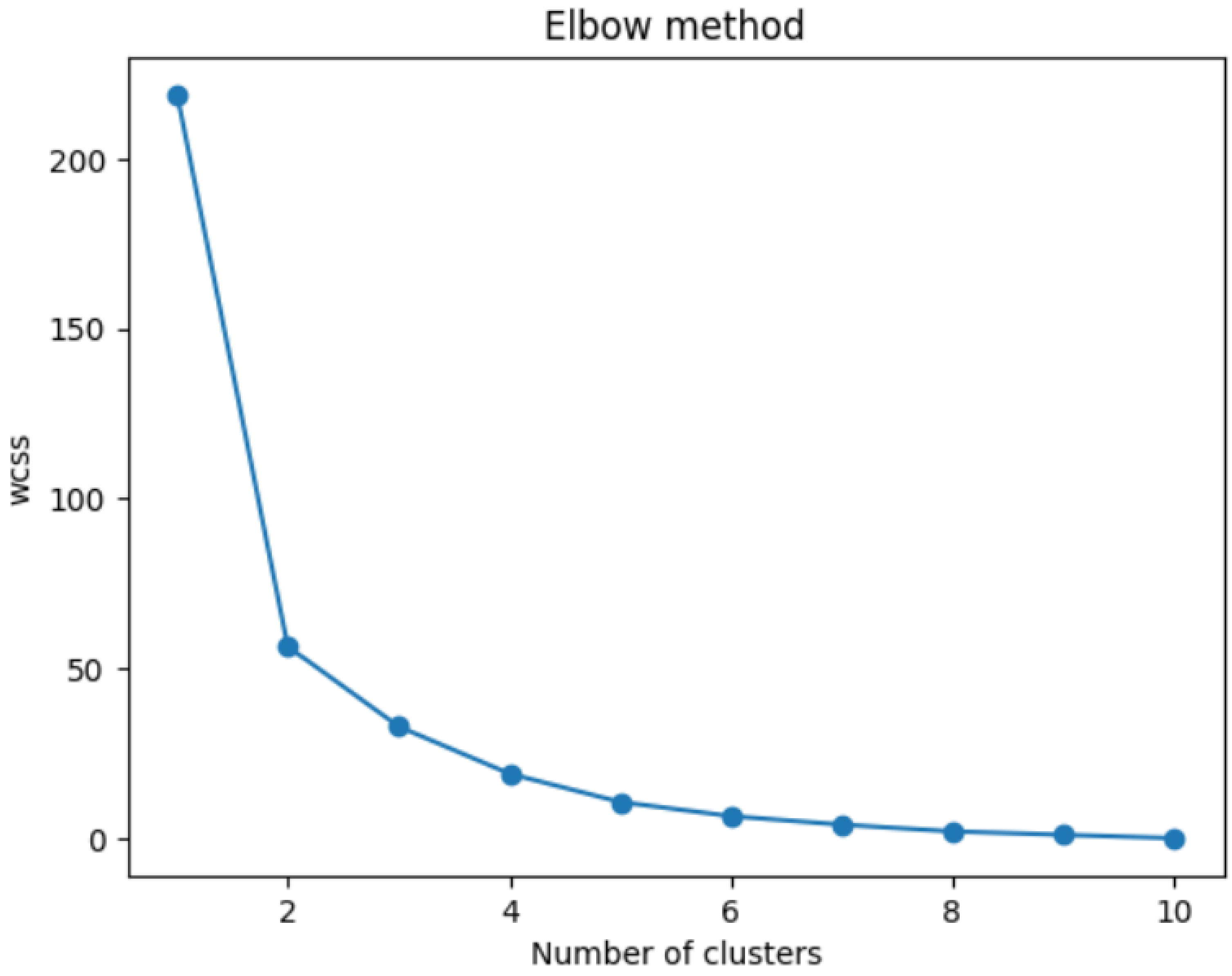
Turn the data into a set of points:

```
data = list(zip(x, y))
print(data)
```

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11,
25), (14, 24), (6, 22), (10, 21), (12, 21)]
```

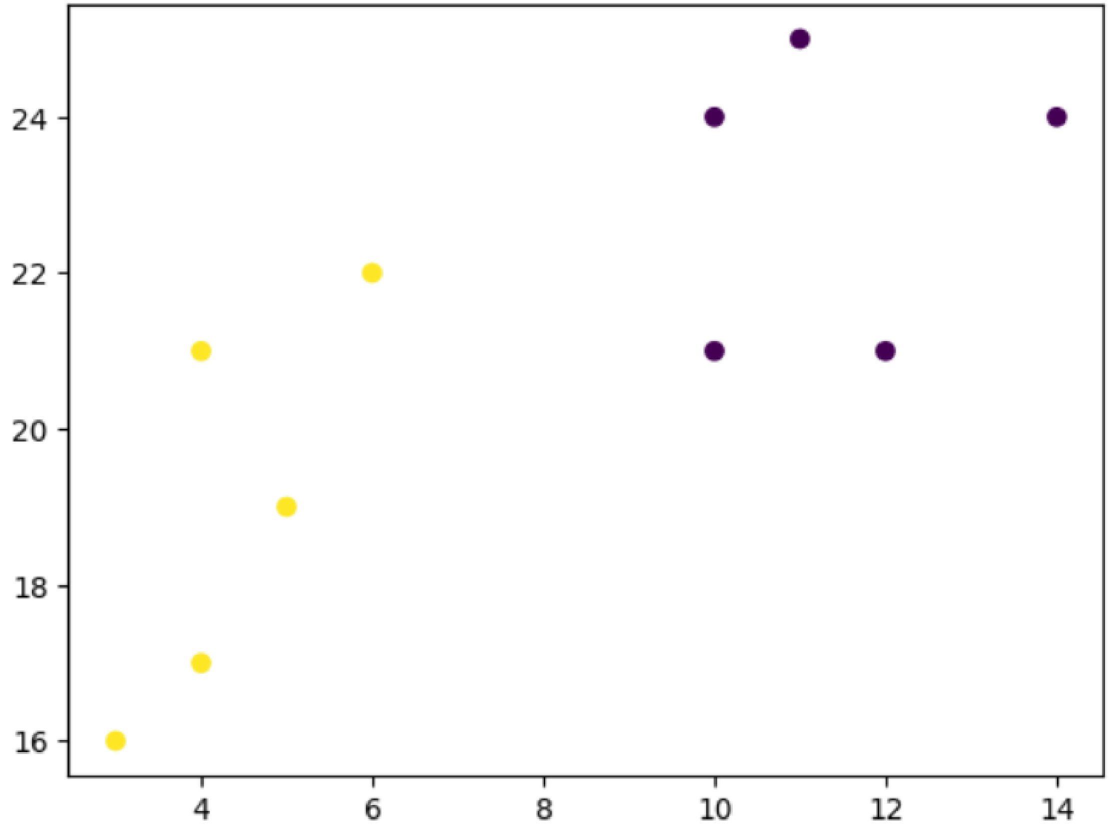
- To find the best value for K, run K-means across the data for a range of possible values.
- We have 10 data points, so the maximum number of clusters is 10.
- For each value K in range(1,11), we train a K-means model and plot the **intertia** at that number of clusters:

```
wcss = []

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(data)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11), wcss, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('wcss')
plt.show()
```



We can see that the "elbow" on the graph, where the interia becomes more linear is at K=2.

```
kmeans = KMeans(n_clusters =2, init = 'k-means++', max_iter = 300, n_init = 10, random_state =0)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```



K-means clustering – implementation for mall.csv

	CustomerID	Genre	Age	Annual.Income..k..	Spending.Score..1.100.
1	1	Male	19	15	39
2	2	Male	21	15	81
3	3	Female	20	16	6
4	4	Female	23	16	77
5	5	Female	31	17	40
6	6	Female	22	17	76
7	7	Female	35	18	6
8	8	Female	23	18	94
9	9	Male	64	19	3
10	10	Female	30	19	72
11	11	Male	67	19	14
12	12	Female	35	19	99

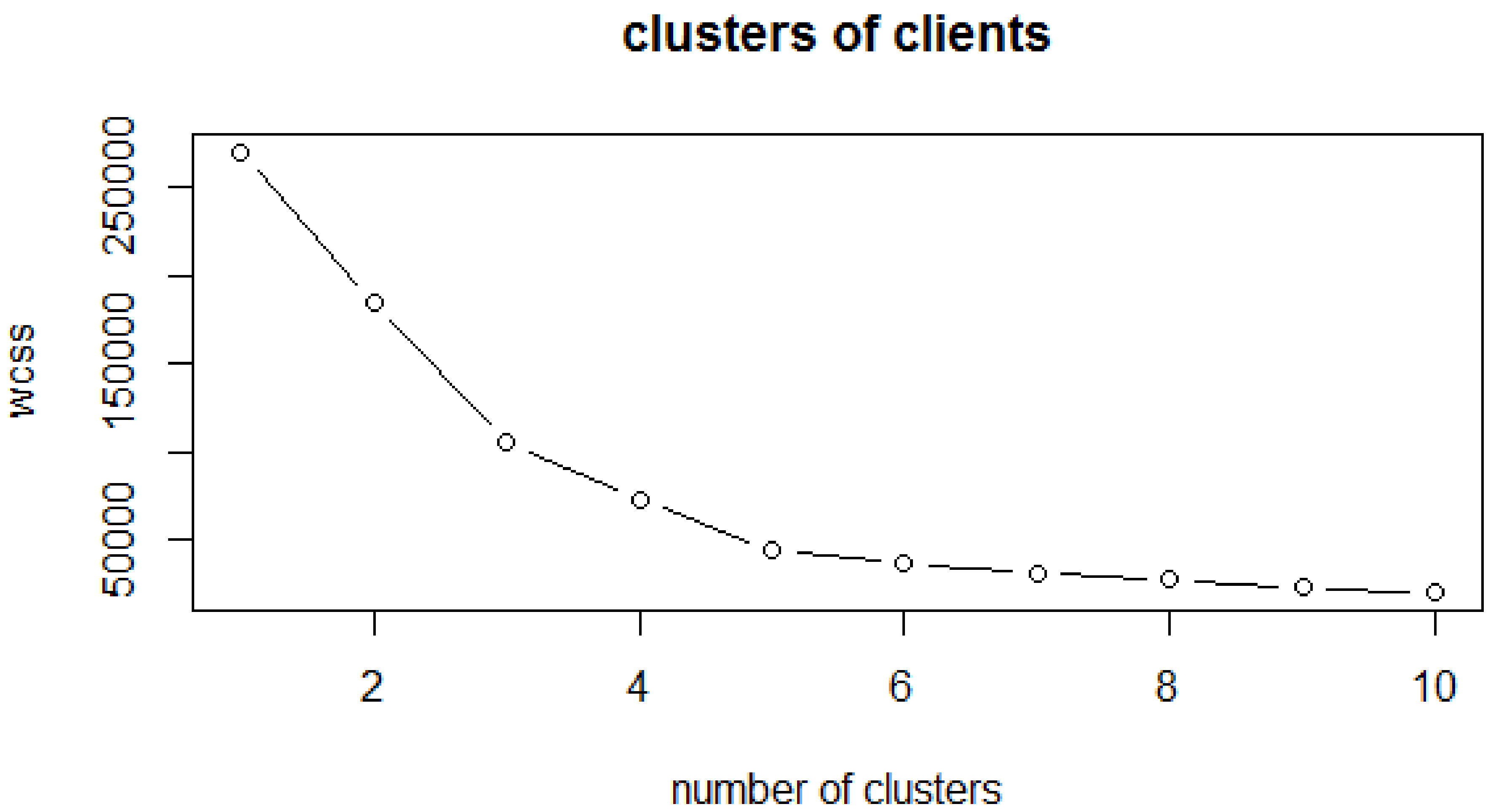
Showing 1 to 12 of 200 entries

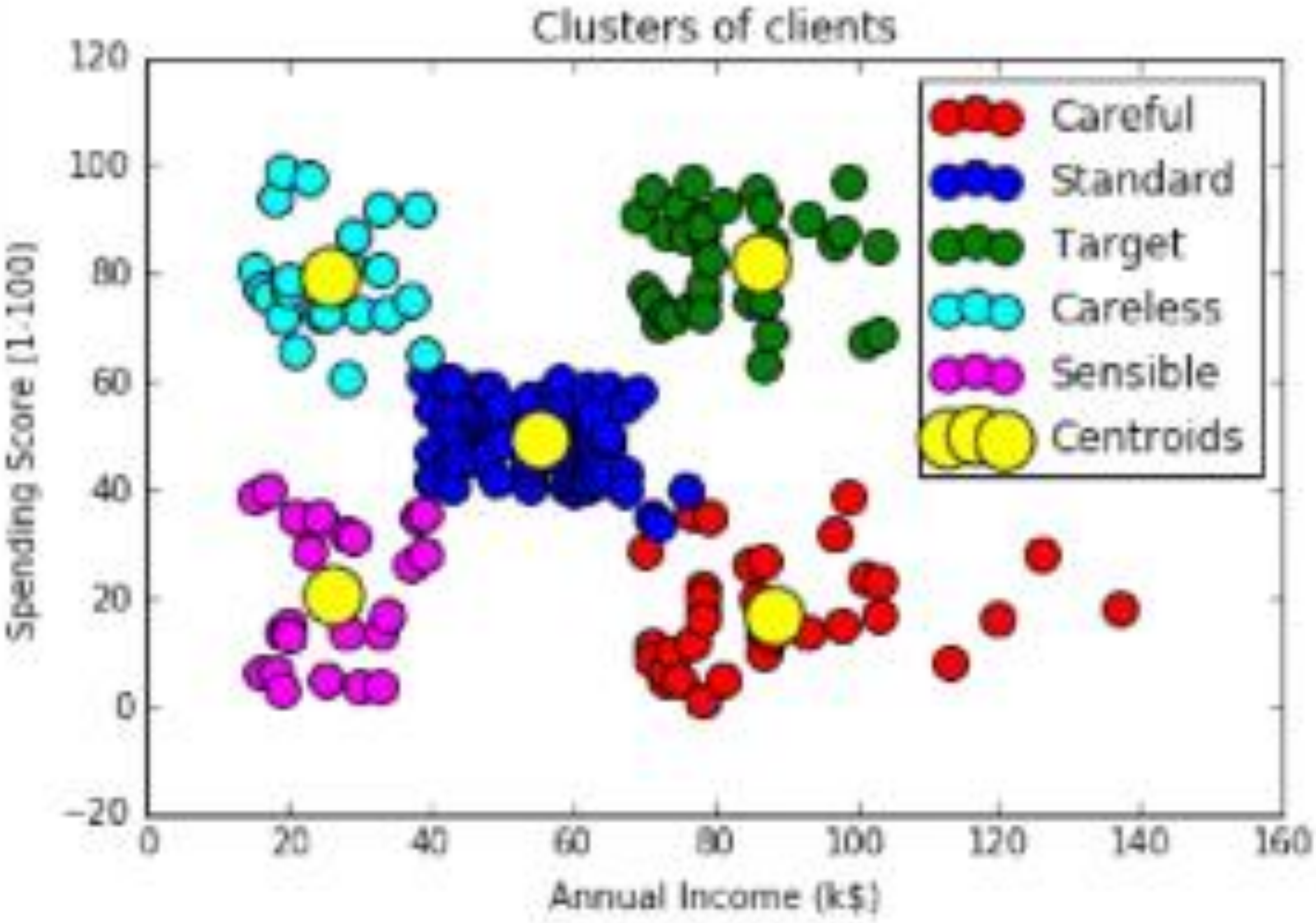
K-means clustering – implementation

	Annual.Income..k..	Spending.Score..1.100.
1	15	39
2	15	81
3	16	6
4	16	77
5	17	40
6	17	76
7	18	6
8	18	94
9	19	3
10	19	72
11	19	14
12	19	99

Showing 1 to 12 of 200 entries

K-means clustering – implementation




Clusters of clients
Careful
Standard
Target
Careless
Sensible
Centroids
Spending Score (1-100)
Annual Income (k$)
120
100
80
60
40
20
0
−20
0
20
40
60
80
100
120
140
160

K-means clustering in python

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the mall dataset with pandas
dataset = pd.read_csv(‘mall.csv’)
X = dataset.iloc[:, [3, 4]]. values
```

K-means clustering in python

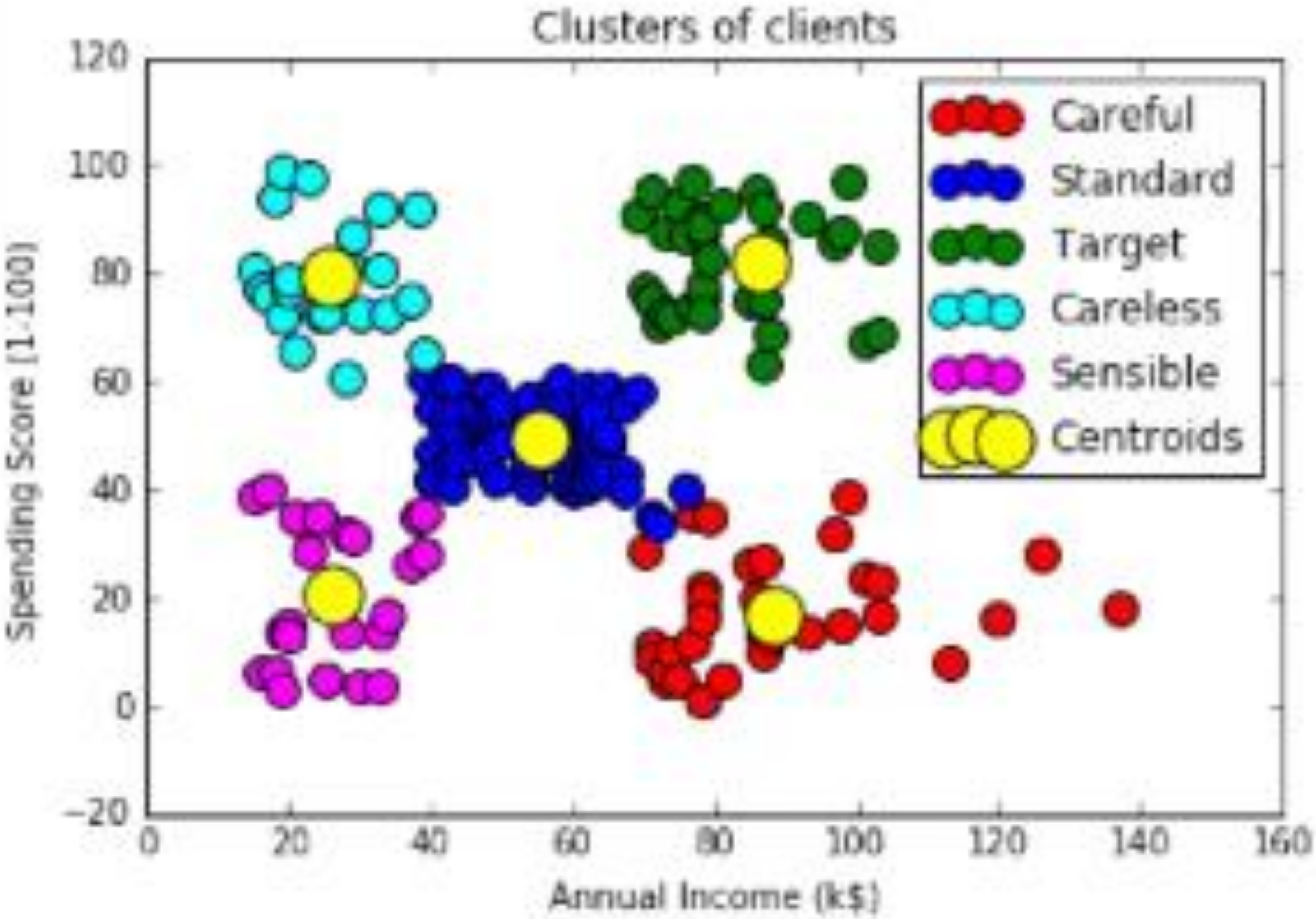
```
#Using the elbow method to find the optimal number of clusters
from sklearn.cluster import Kmeans
wcss = []
for i in range (1, 11):
    kmeans = KMeans(n_clusters = i, init = ‘k-means++’, max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title(‘The elbow Method’)
plt.xlabel(‘Number of clusters’)
plt.ylabel(‘WCSS’)
plt.show()
```

K-means clustering in python

```
# Apply K-Means to the mall dataset
kmeans = KMeans(n_cluster = 5, init =‘k-means++’, max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(X)
```

K-means clustering in python

```
plt.scatter(X[y_kmeans==0, 0], X[y_kmeans ==0, 1], s=100, c='red', label = 'Careful')
plt.scatter(X[y_kmeans==1, 0], X[y_kmeans ==1, 1], s=100, c='blue', label = 'Standard')
plt.scatter(X[y_kmeans==2, 0], X[y_kmeans ==2, 1], s=100, c='green', label = 'Target')
plt.scatter(X[y_kmeans==3, 0], X[y_kmeans ==3, 1], s=100, c='cyan', label = 'Careless')
plt.scatter(X[y_kmeans==4, 0], X[y_kmeans ==4, 1], s=100, c='magenta', label = 'Sensible')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='yellow', label = 'Centroids')

plt.title('Cluster of Clients')
plt.xlabel('Annual Income K$')
plt.ylabel ('Spenting Score (1-100)')
plt.legend()
plt.show()
```


Clusters of clients
Careful
Standard
Target
Careless
Sensible
Centroids
Spending Score (1-100)
Annual Income (k$)
120
100
80
60
40
20
0
−20
0
20
40
60
80
100
120
140
160


Thank you