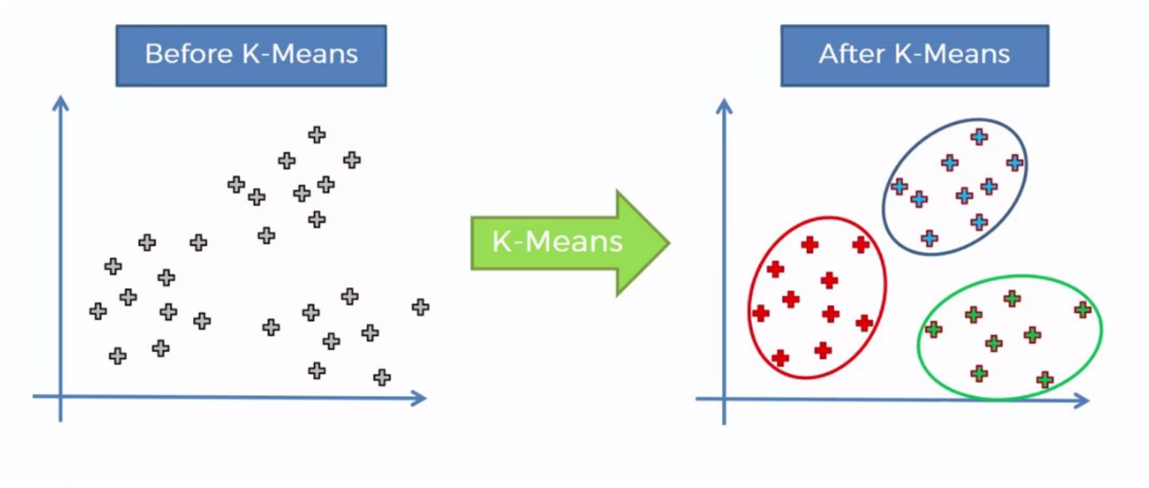
K-Means clustering

K-Means clustering is the most popular unsupervised learning algorithm. It is used when we have unlabelled data which is data without defined categories or groups. The algorithm follows an easy or simple way to classify a given data set through a certain number of clusters, fixed apriori. K-Means algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

K-Means clustering can be represented diagrammatically as follows:-



2. How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

3. K-Means Clustering intuition

K-Means clustering is used to find intrinsic groups within the unlabelled dataset and draw inferences from them. It is based on centroid-based clustering.

Centroid - A centroid is a data point at the centre of a cluster. In centroid-based clustering, clusters are represented by a centroid. It is an iterative algorithm in which the notion of similarity is derived by how close a data point is to the centroid of the cluster. K-Means clustering works as follows:-

The K-Means clustering algorithm uses an iterative procedure to deliver a final result. The algorithm requires number of clusters K and the data set as input. The data set is a collection of features for each data point. The algorithm starts with initial estimates for the K centroids. The algorithm then iterates between two steps:-

3.1 Data assignment step

Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, which is based on the squared Euclidean distance. So, if ci is the collection of centroids in set C, then each data point is assigned to a cluster based on minimum Euclidean distance.

3.2 Centroid update step

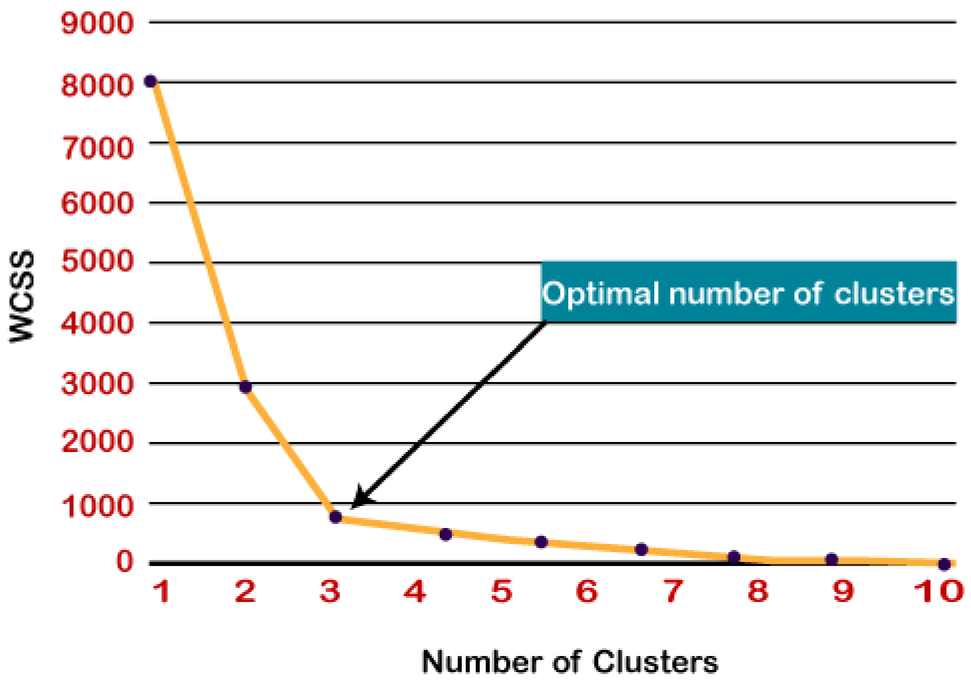
In this step, the centroids are recomputed and updated. This is done by taking the mean of all data points assigned to that centroid's cluster.

The algorithm then iterates between step 1 and step 2 until a stopping criteria is met. Stopping criteria means no data points change the clusters, the sum of the distances is minimized or some maximum number of iterations is reached. This algorithm is guaranteed to converge to a result. The result may be a local optimum meaning that assessing more than one run of the algorithm with randomized starting centroids may give a better outcome.

4. Choosing the value of K

The K-Means algorithm depends upon finding the number of clusters and data labels for a pre-defined value of K. To find the number of clusters in the data, we need to run the K-Means clustering algorithm for different values of K and compare the results. So, the performance of K-Means algorithm depends upon the value of K. We should choose the optimal value of K that gives us best performance. There are different techniques available to find the optimal value of K. The most common technique is the **elbow method** which is described below.

The elbow method is used to determine the optimal number of clusters in K-means clustering. The elbow method plots the value of the cost function produced by different values of K. The below diagram shows how the elbow method works:-



Stopping Criteria for K-Means Clustering

There are essentially three stopping criteria that can be adopted to stop the K-means algorithm:

1.Centroids of newly formed clusters do not change
2.Points remain in the same cluster
3.Maximum number of iterations is reached

We can stop the algorithm if the centroids of newly formed clusters are not changing. Even after multiple iterations, if we are getting the same centroids for all the clusters, we can say that the algorithm is not learning any new pattern, and it is a sign to stop the training.

Another clear sign that we should stop the training process is if the points remain in the same cluster even after training the algorithm for multiple iterations.

Finally, we can stop the training if the maximum number of iterations is reached. Suppose we have set the number of iterations as 100. The process will repeat for 100 iterations before stopping.

```
# importer les librairies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.metrics import confusion_matrix

#charger le dataset
iris=load_iris()
# afficher les features
print(iris.feature_names)

# stocker les données en tant que dataframe
X=pd.DataFrame(iris.data)
# définir les noms des colones
X.columns=['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width']
Y=pd.DataFrame(iris.target)
Y.columns=['Class']
print(X)

# répartition du dataset dans un scater plot 2D
plt.scatter(X.Petal_Length, X.Petal_Width)

# visualiser les classes
colorL=np.array(['green', 'red','blue'])
plt.scatter(X.Petal_Length, X.Petal_Width, c=colorL[iris.target], s=20)

# Utiliser la méthode Elbow pour trouver le nombre optimal des clusters
inert=[]
for i in range(1,10):
   model=KMeans(n_clusters=i)
   model.fit(X)
   inert.append(model.inertia)

# afficher la courbe Elbow
plt.plot(range(1,10), inert)
plt.title('La méthode Elbow')
plt.xlabel('Nombre de clusters')
plt.ylabel('Inertia')
plt.show

# cluster KMeans
model=KMeans(n_clusters=3)
```

```
model.fit(X)
model.predict(X)   #model.labels_

#visualiser les classes prédites par le modèle
colorL=np.array(['green', 'red','blue'])
plt.scatter(X.Petal_Length, X.Petal_Width, c=colorL[model.predict(X)], s=20)

# matrice de confusion
confusion_matrix(iris.target, model.labels)
```