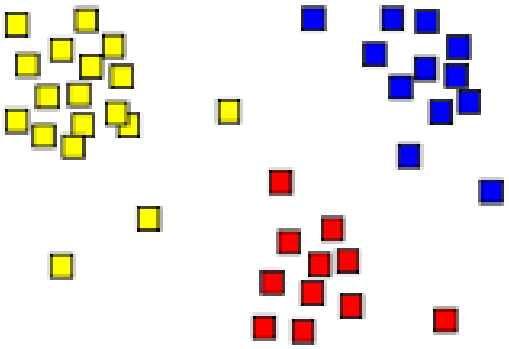
Clustering

It is basically a type of *unsupervised learning method*. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples. It is a main task of exploratory data analysis, and a common technique for statistical data analysis, used in many fields, including pattern recognition, image, analysis, information retrieval, bioinformatics, data compression, computer graphics and machine learning.



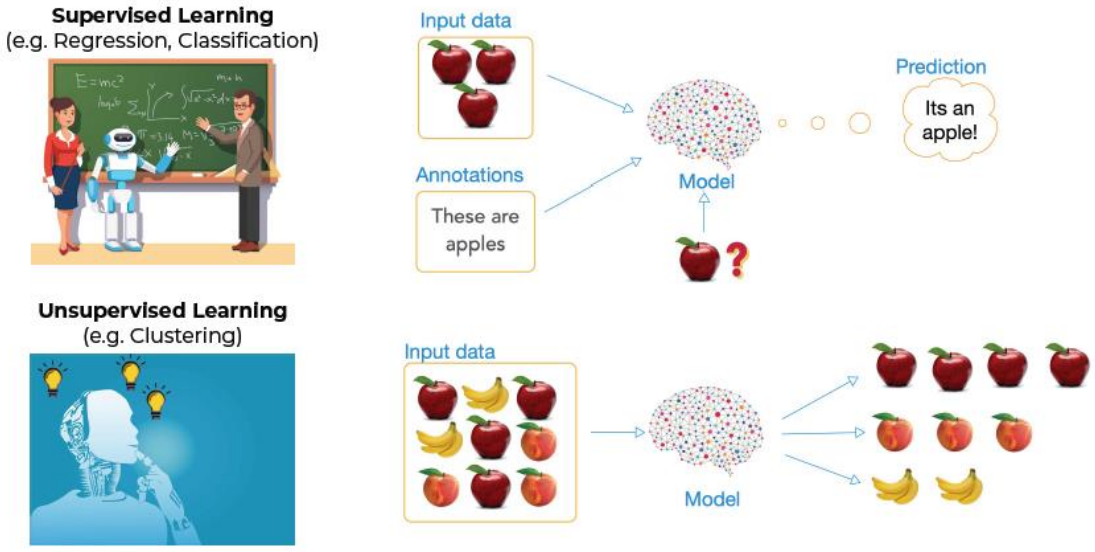
The result of a cluster analysis shown as the coloring of the squares into three clusters.

Clustering is similar to classification, but the basis is different. In Clustering you don't know what you are looking for, and you are trying to identify some segments or clusters in your data. When you use clustering algorithms on your dataset, unexpected things can suddenly pop up like structures, clusters and groupings you would have never thought of otherwise.

In this part, you will understand and learn how to implement the following Machine Learning Clustering models:

1. K-Means Clustering
2. Hierarchical Clustering

What is Clustering?



Supervised Learning (Regression, Classification): Already have training data and answer. Answer of the training data supply to the model. Above example already have supplied to the Model as input data of Apple and annotation, these are apples. Now supply new image and ask for is this? Model provide answer as it's an apple.

Un-supervised model is different, here don't have answer and model has to think for itself. Here input data send as different types of fruits images and ask to model group these fruits in different categories even we don't supply the categories. Machine has to know or understand apples, bananas and oranges. It can make conclusion and create own group. machine can understand what is banana, apple and orange. So, supervised learning you give the model and opportunity to train where has the answer and unsupervised you don't supply the answer.

Clustering Methods:

- **Density-Based Methods:** These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example *DBSCAN (Density-Based*

Spatial Clustering of Applications with Noise), *OPTICS (Ordering Points to Identify Clustering Structure)*, etc.

- **Hierarchical Based Methods:** The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

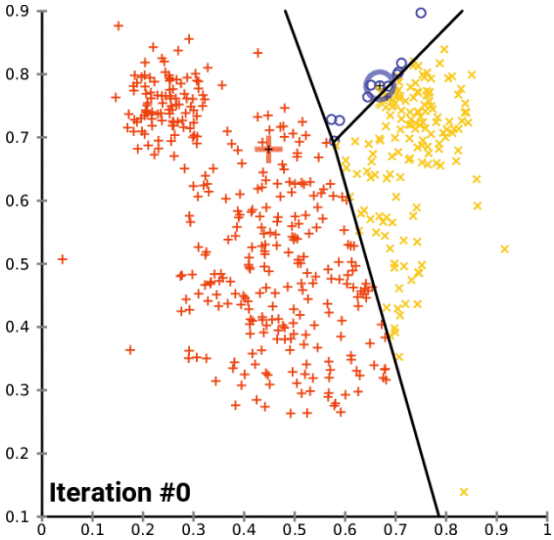
    - **Agglomerative** (bottom-up *approach*)
    - **Divisive** (top-down *approach*)

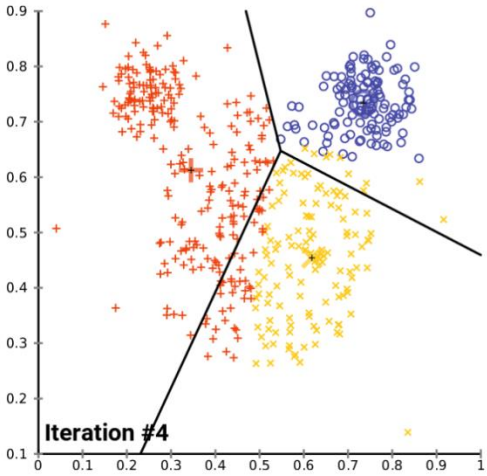
- **Partitioning Methods:** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example *K-means, CLARANS (Clustering Large Applications based upon Randomized Search)*, etc.
- **Grid-based Methods:** In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects example *STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest)*, etc.

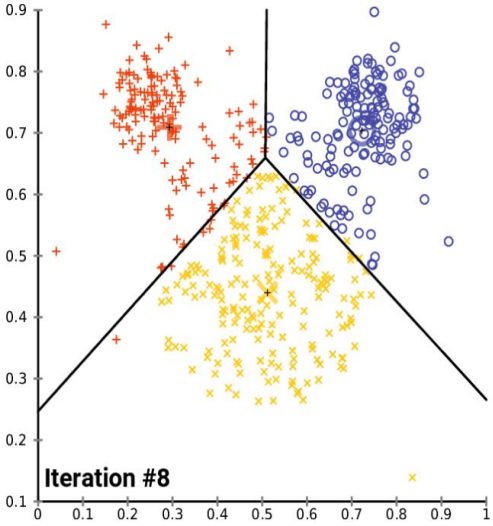
Clustering Algorithms: K-means clustering algorithm – It is the simplest unsupervised learning algorithm that solves clustering problem. K-means algorithm partitions n observations into k clusters where each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster.

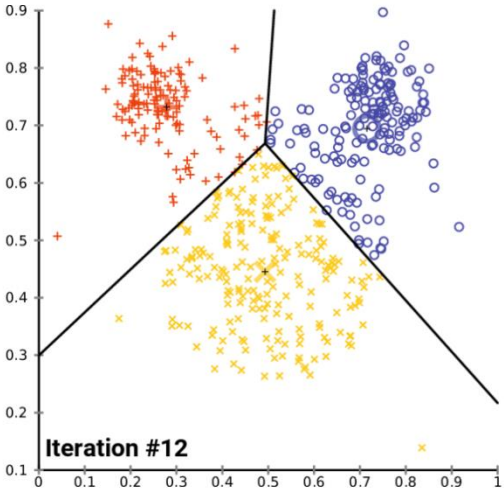
K **MEANS** CLUSTERING

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. *k*-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, better Euclidean solutions can be found using *k*-medians and *k*-medoids.









Algorithms: **Standard algorithm (naive k-means)**

WCSS (Within-Cluster Sum of Squares) is a crucial concept in **K-Means Clustering**, an unsupervised learning algorithm used for grouping unlabeled datasets into different clusters. Let's dive into how it works:

1. **K-Means Clustering Overview**:
   - K-Means aims to divide an unlabeled dataset into **K predefined clusters**. Each data point belongs to only one cluster based on its similarity to other data points.
   - The algorithm iteratively refines the clusters by minimizing the sum of distances between data points and their corresponding centroids.
2. **Working of K-Means Algorithm**:
   - **Step 1: Choose K**:
     - Decide the number of clusters (**K**). This value determines how many centroids the algorithm will create.
   - **Step 2: Initialize Centroids**:
     - Randomly select **K** points as initial centroids (they can be from the dataset or other points).
   - **Step 3: Assign Data Points**:
     - Assign each data point to the closest centroid. This forms the predefined **K clusters**.
   - **Step 4: Update Centroids**:
     - Calculate the variance within each cluster (WCSS) and place a new centroid at the center of each cluster.
   - **Step 5: Repeat Iterations**:
     - Reassign each data point to the closest new centroid.
     - If any reassignment occurs, go back to Step 4; otherwise, proceed to the next step.
   - **Step 6: Model Ready**:
     - The algorithm converges when no further reassignments occur.
     - The final centroids represent the clusters.
3. **Visual Explanation**:
   - Imagine a scatter plot with two variables (let's call them M1 and M2).
   - Suppose we want to create **2 clusters (K=2)**.
   - Randomly choose initial centroids.
   - Assign data points to the nearest centroid.
   - Recalculate centroids.
   - Repeat until convergence.

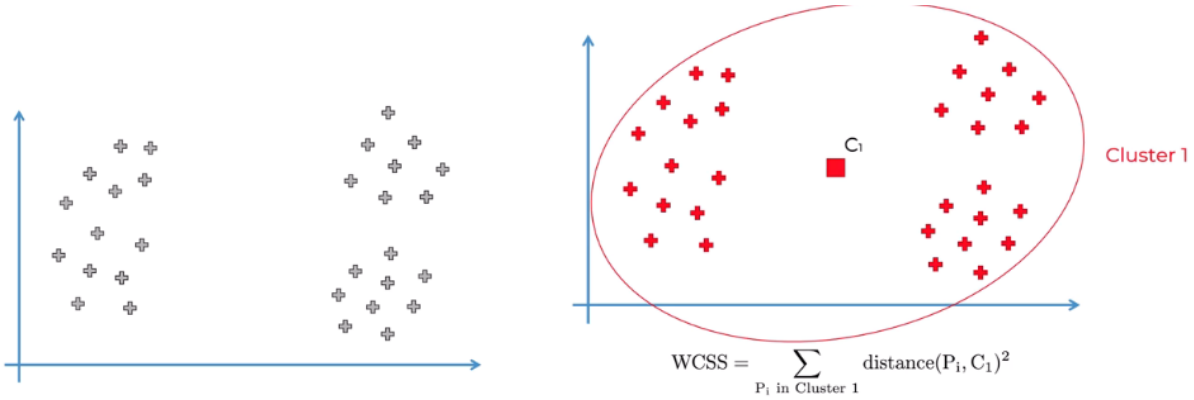
   !K-Means Clustering
4. **Elbow Method**:
   - To determine the optimal **K**, we use the **elbow method**.
   - It plots the **WCSS** against different values of **K**.
   - The "elbow" point (where WCSS starts to level off) indicates the best **K**.

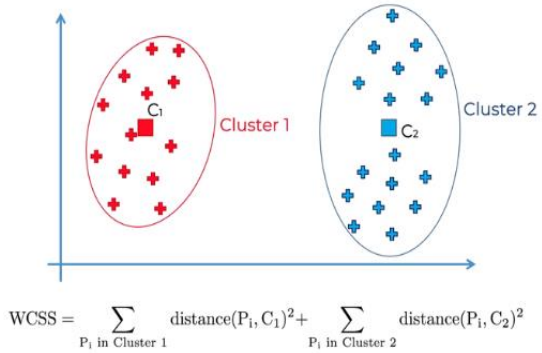
K-Means Clustering iteratively refines clusters by minimizing distances, and WCSS plays a crucial role in this process.

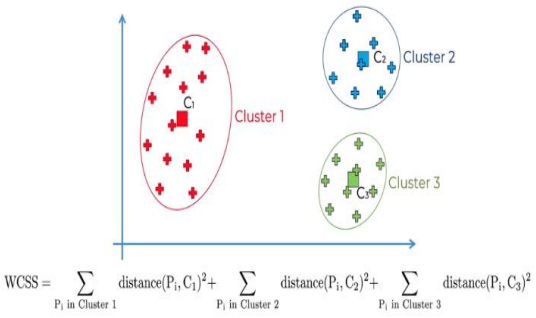
Elbow Method

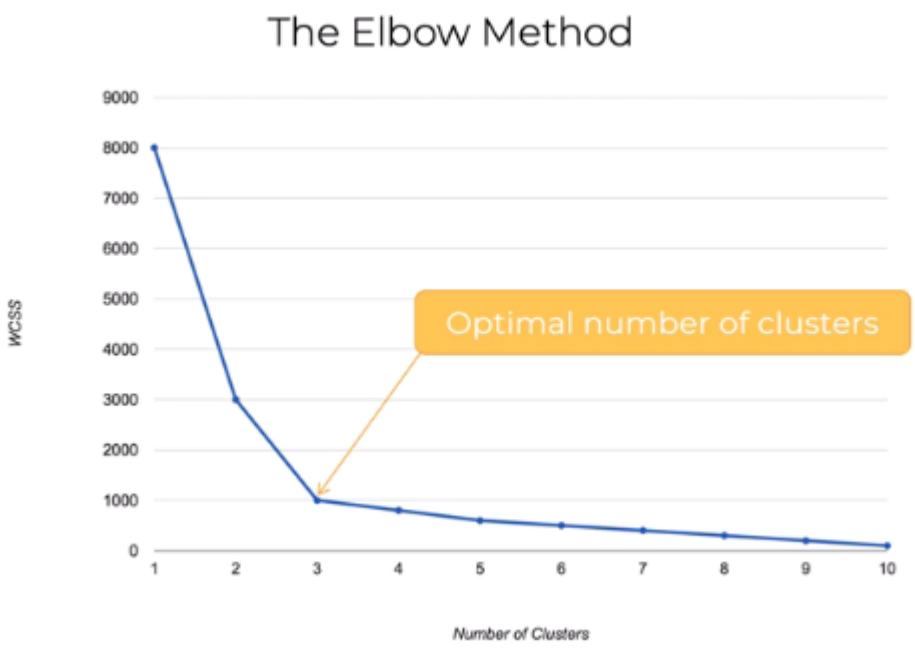
the elbow method involves finding the optimal k via a graphical representation. It works by finding the within-cluster sum of square (WCSS), i.e. the sum of the square distance between points in a cluster and the cluster centroid.

The elbow graph shows WCSS values on the y-axis corresponding to the different values of K on the x-axis. When we see an elbow shape in the graph, we pick the K-value where the elbow gets created. We can call this the elbow point. Beyond the elbow point, increasing the value of 'K' does not lead to a significant reduction in WCSS.





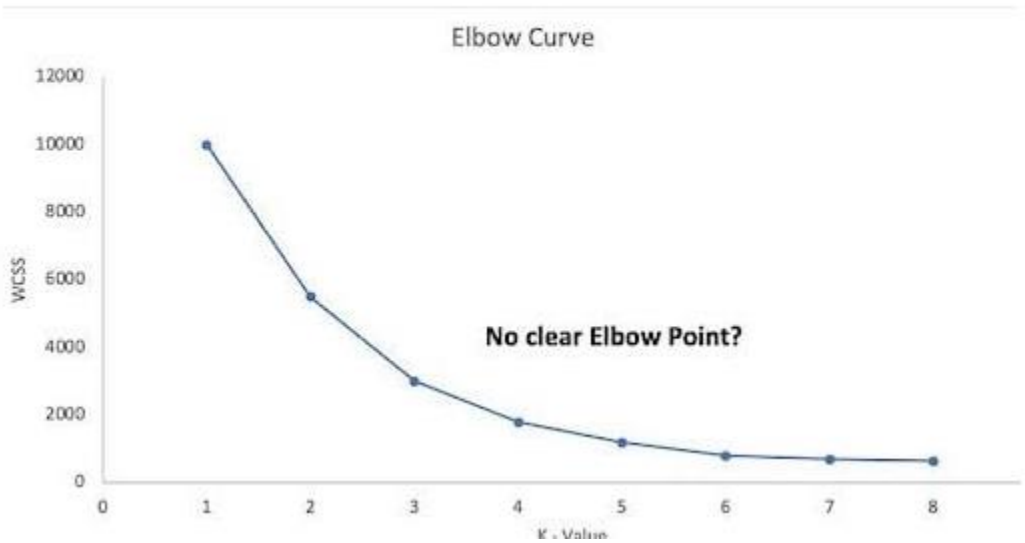




How to Use the Elbow Method in Python

```
#install yellowbrick to vizualize the Elbow curve
!pip install yellowbrick

from sklearn import datasets
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer

# Load the IRIS dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Instantiate the clustering model and visualizer
km = KMeans(random_state=42)
visualizer = KElbowVisualizer(km, k=(2,10))

visualizer.fit(X)        # Fit the data to the visualizer
visualizer.show()        # Finalize and render the figure
```

But here's what it typically looks like:



Actual elbow curve with no clear elbow.

So, in the majority of the real-world data sets, there's not a clear elbow inflection point to identify the right 'K' using the elbow method. This makes it easier to find the wrong K.

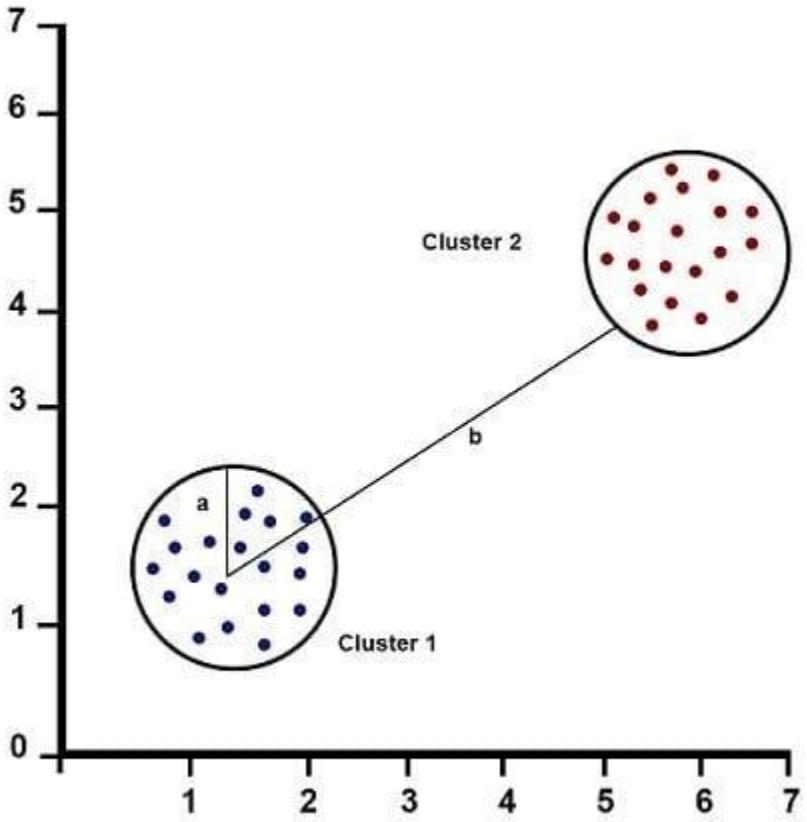
Silhouette Method Is Better Than the Elbow Method

The Silhouette score is a very useful method to find the number of K when the elbow method doesn't show the elbow point.

The value of the Silhouette score ranges from -1 to 1. Following is the interpretation of the Silhouette score.

- **1:** Points are perfectly assigned in a cluster and clusters are easily distinguishable.

- **0:** Clusters are overlapping.
- **-1:** Points are wrongly assigned in a cluster.



Silhouette scores for two clusters.

Silhouette Score = (b-a)/max(a,b)

Where:

- a = average intra-cluster distance, i.e the average distance between each point within a cluster.

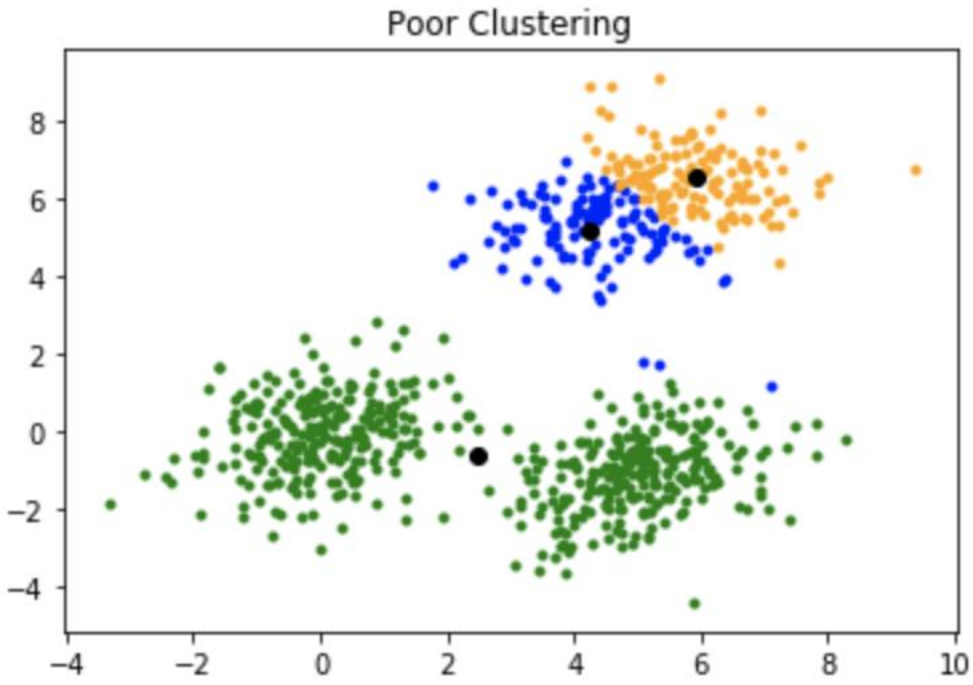
- b = average inter-cluster distance i.e the average distance between all clusters.

k-means++

***k*-means clustering** is a method of vector quantization, originally from signal processing, that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. *k*-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, better Euclidean solutions can be found using *k*-medians and *k*-medoids.

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. So, 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. Similarly, more than one centroid might be initialized into the same cluster resulting in poor clustering. For example, consider the images shown below.

A poor initialization of centroids resulted in poor clustering.



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. That is K-means++ is the standard K-means algorithm coupled with a smarter initialization of the centroids.

K-Means++ Initialization Algorithm:

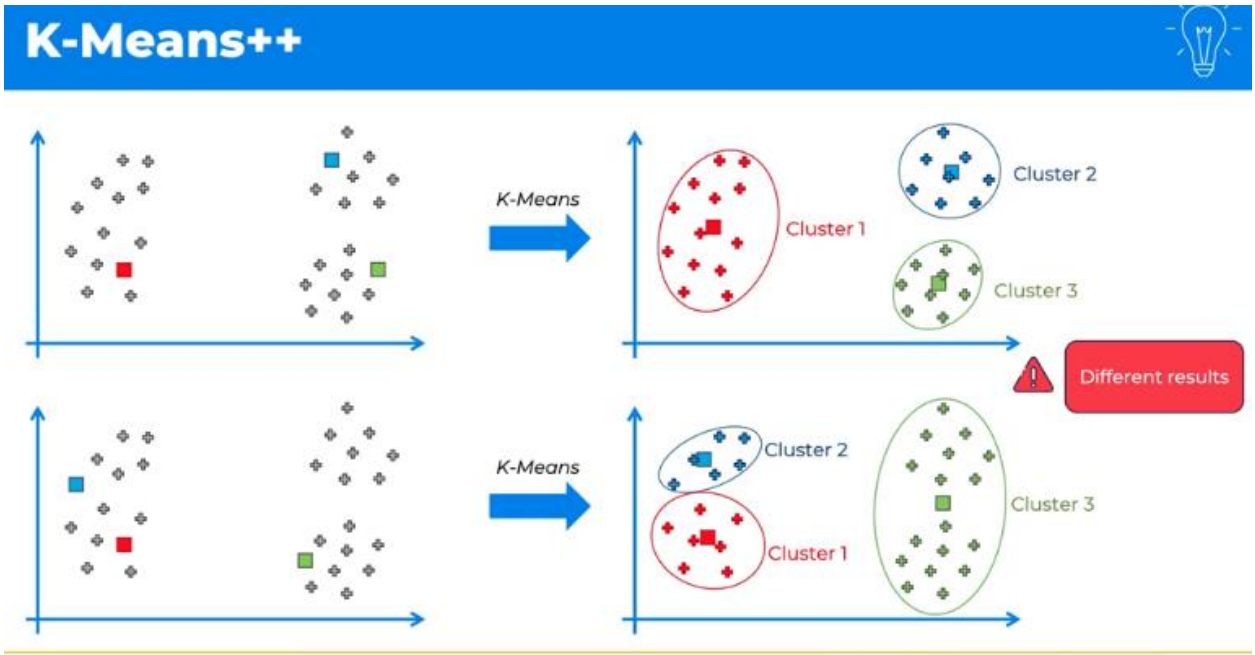
Step 1: Choose first centroid at random among data points

Step 2: For each of the remaining data points compute the distance (D) to the nearest out of already selected centroids

Step 3: Choose next centroid among remaining data points using weighted random selection – weighted by D^2

Step 4: Repeat Steps 2 and 3 until all k centroids have been selected

Step 5: Proceed with standard k-means clustering



K-Mean Clustering in Python

Dataset:

Key point: We will not to predict here because we will have no idea what to predict. We want to identify some pattern. We will create dependent variable and that will take finite number of variables.

Mall_Customers.csv:

CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-1
1	Male	19	15	39
2	Male	21	15	81
3	Female	20	16	6
4	Female	23	16	77
5	Female	31	17	40
6	Female	22	17	76
7	Female	35	18	6
8	Female	23	18	94
9	Male	64	19	3
10	Female	30	19	72
11	Male	67	19	14
12	Female	35	19	99
13	Female	58	20	15
14	Female	24	20	77
15	Male	37	20	13
16	Male	22	20	79
17	Female	35	21	35
18	Male	20	21	66
19	Male	52	23	29

K-Means Clustering (Python)

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

```
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
```

** customer_id no impact on whatever we will create dependent variable in future. Process the K-Mean algorithm identify the cluster of data and also create dependent variable. customer_id just identification the customer. Two features (Annual Income & Spending Score (1-100)) we are using here out four matrices of features because visualize the cluster 2D features. If we use four features then very hard to visualize the all features. Every axis is one feature then may be possible to visualize three dimensions but nice graph 2D therefore only going with column and identify the cluster. Python starting index is 0 like customer_id. Therefore, Annual Income K$ as index 3 & Spending Score (1-100) as index 4.

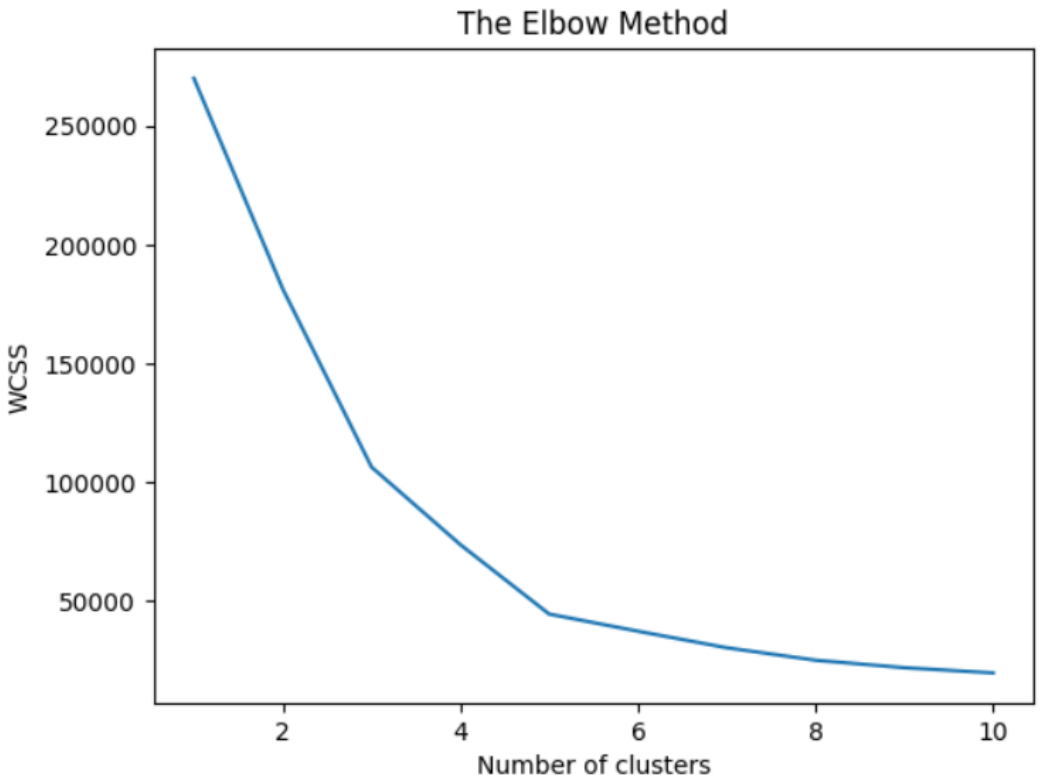
Using the elbow method to find the optimal number of clusters

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    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()
```

** The **sklearn.cluster** module gathers popular unsupervised clustering algorithms. Kmeans is class. wcss [ ] list for each number of cluster. range(1, 11) → 10 different KMeans clusters. n_clusters is number of clusters i=1,2,3,….10. init = 'k-means++' initialization so save from random initialization trap. random_state = 42 any number you can take here 42 is the bring luck mathematics number. kmeans.fit(X) method trained kmeans algorithm data as feature of Annual Income & Spending Score (1-100). `It is way identify some clusters. wcss.append(kmeans.inertia_)` → `append new value inside the list. wcss computed one cluster. some of the square distances all the observation points and centre point. Attribute of the Kmeans class which is called inertia and here kmeans is object.`
`plt.plot(range(1, 11), wcss)` → `plot function X Co-ordinate is range(1,11) and Y Co-ordinate is wcss.`

O/P:



So, optimal number of clusters is 5. After this point curve started being almost flat and decreasing very slowly. Next step train the K-Mean Model on the dataset that's why we want to build it, trained it and run it to identify the 5 clusters.

Training the K-Means model on the dataset

```
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

****kmeans.fit_predict(X)→ fit_predict method not only trained it also returned exactly 5 clusters different group of customers. values taken from customer make a group of Each group similar information. fit_predict(X)creates the dependent variable y_kmeans.**

```
print(y_kmeans)
```

O/P:

```
[2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 0 2 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 4 1 4 0 4 1 4 1 4 0 4 1 4 1 4 1 4 1 4 0 4 1 4 1 4
 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1
 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4]
```

****We can see 1st customer belongs to cluster 3 (Index 2 because cluster 1 start with index 0)**

Visualising the clusters

```
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c =
'green', label = 'Cluster 3')
```

```
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s = 300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')

plt.show()
```

**** details of code: plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1') → X contains exactly different customers mean matrix of two columns Annual Income(k$) & Spending Score (1-100); X co-ordinate is Annual Income(k$) and Y co-ordinate is Spending Score (1-100); X[y_kmeans == 0, 0] → y_kmeans == 0 is the 1st cluster and 0 refers to 1st column mean index 0 related to Annual Income(k$);**
X[y_kmeans == 0, 1] → y_kmeans == 0 is the 1st cluster and 1 refers to 2nd column mean index 1 related to Spending Score (1-100);
Simple stuff size S=100 mean display began of the points, each of the point is different customer in 1st cluster.

Same way for second cluster :

```
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
```

upto five cluster.

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroids') → kmeans.cluster_centers_ refer different centre input take different values of the cluster and 0 to correspondent to 1st column inside the cluster centre array;
kmeans.cluster_centers_[:, 1] → Y co-ordinate 1 correspond to second column inside the cluster centre array;
s=300 larger size clearly highlight the centre point among all the observation points of the customers in a cluster.

O/P :

Green cluster Annual Income low and Spending Score also low.

Magenta cluster high income and high spending.

Red cluster medium income and medium spend.