Machine Learning (ML) with Python

Unsupervised Learning (K-means Clustering)

Dr. Aeshah Alsughayyir

Collage of Computer Science and Engineering

Taibah University

2021-2022

Outline:

- What is Clustering?
- Applications of Clustering in Real-World Scenarios.
- Properties of Clusters.
- Understanding the Different Evaluation Metrics for Clustering.
- K-Means vs. K-Nearest Neighbor (KNN)
- K-Means Clustering
    - What is K-Means Clustering?
    - How does K-Means work?
    - Challenges with K-Means Algorithm
    - How to choose the Right Number of Clusters in K-Means?
    - Implementing K-Means Clustering in Python
    - Pros and Cons associated with K-Means Clustering.
        - Dealing with Outliers

What is Clustering?

- Cluster analysis, or clustering, is an unsupervised machine learning task.
- It involves automatically discovering natural grouping in data.
- Unlike supervised learning (like predictive modeling), clustering algorithms only interpret the input data and find natural groups or clusters in feature space.

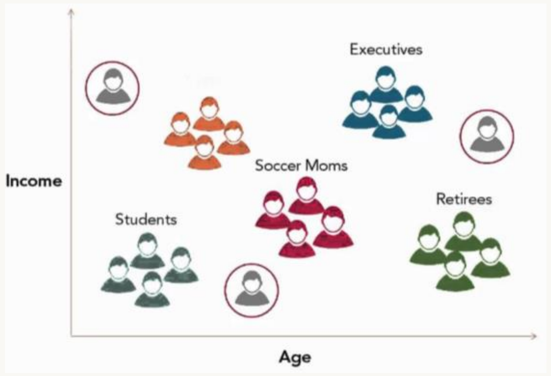
Clustering is the process of dividing the entire data into groups (also known as clusters) based on the patterns in the data.

"Clustering techniques apply when there is no class to be predicted but rather when the instances are to be divided into natural groups."
- *Page 141, Data Mining: Practical Machine Learning Tools and Techniques, 2016*

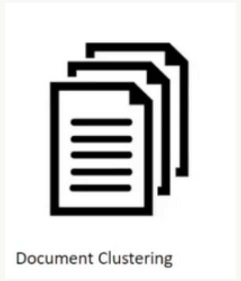
Applications of Clustering in Real-World Scenarios

Clustering is a widely used technique in the industry and one of the most utilized data mining techniques. It is being used in almost every domain

- **Customer Segmentation**
  - It is one of the most common applications of clustering to do targeted marketing.



- **Document Clustering**
  - This is another common application of clustering. Let's say you have multiple documents, and you need to cluster similar documents together. Clustering helps us group these documents such that similar documents are in the same clusters.



- **Image Segmentation**
  - This process trying to club similar pixels in the image together. It applies clustering to create clusters that having similar pixels in the same group.

Example:

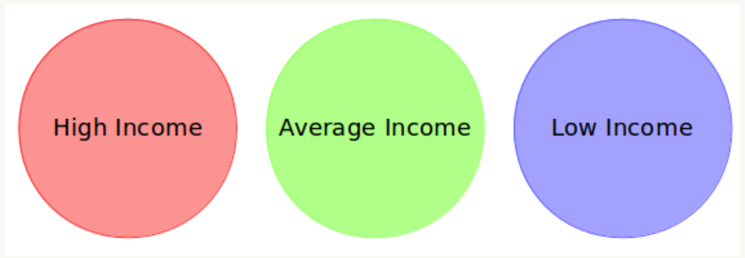
- A bank wants to give credit card offers to its customers. Currently, they look at the details of each customer and based on this information, decide which offer should be given to which customer..

Now, the bank can potentially have millions of customers.
Does it make sense to look at the details of each customer separately and then make a decision?

Certainly not!

So, what can the bank do?

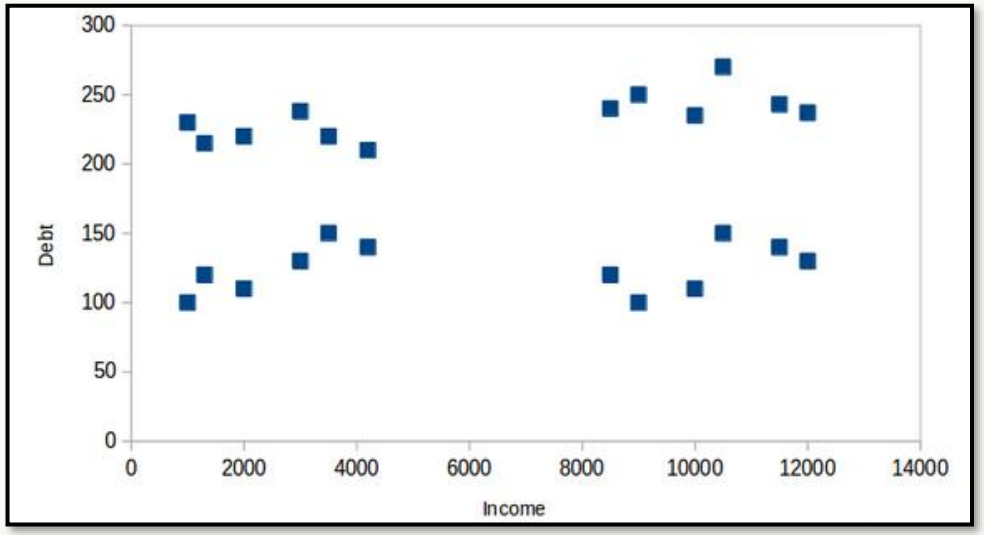
- One option is to segment its customers into different groups. For instance, the bank can group the customers based on their income:



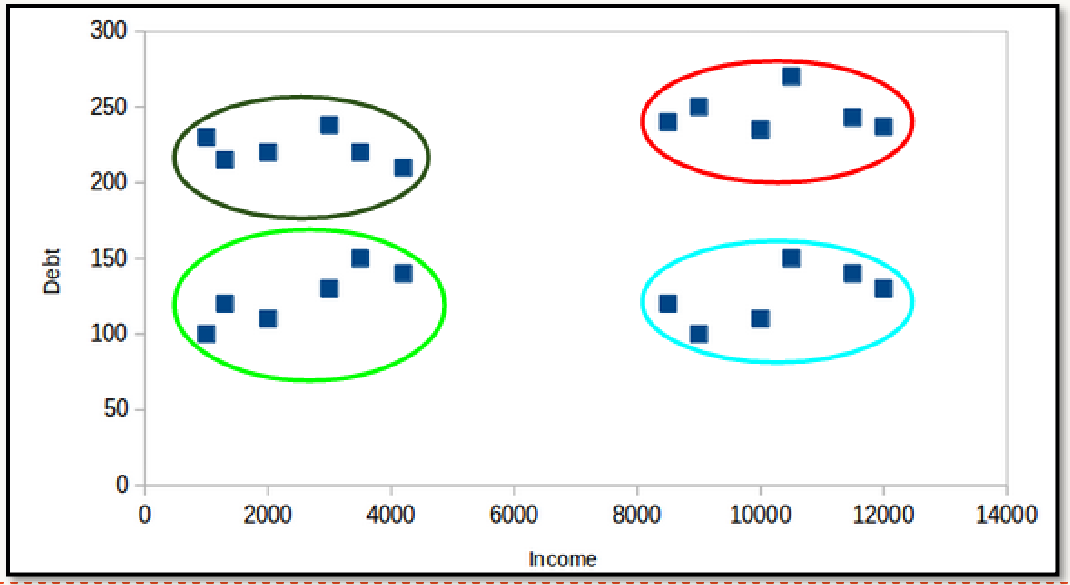
- The bank can now make three different strategies or offers, one for each group. Here, instead of creating different strategies for individual customers, they only have to make 3 strategies. This will reduce the effort as well as the time.

Properties of Clusters

- Let's say the bank only wants to use the income and debt to make the segmentation. They collected the customer data and visualize it:

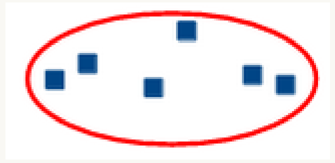


Here, we can clearly visualize that these customers can be segmented into 4 different clusters as shown below:



Properties of Clusters (Cont.)

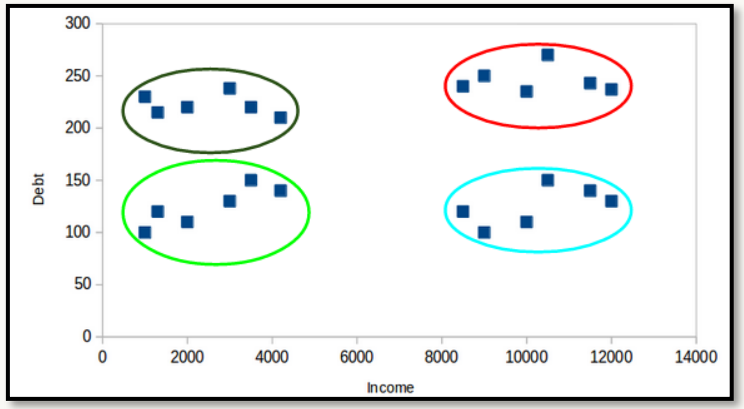
- **Property 1**
  - All the data points in a cluster should be closely similar to each other.



For example, if the customers in a particular cluster are not similar to each other, then their requirements might vary, right? If the bank gives them the same offer, they might not like it and their interest in the bank might reduce.

- **Property 2**
  - The data points from different clusters should be as different as possible.

For example, points in the red cluster are completely different from the customers in the blue cluster. All the customers in the red cluster have high income and high debt and customers in the blue cluster have high income and low debt value. Clearly, we have a good clustering of customers in this case.

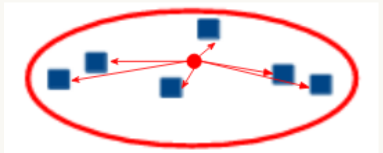


Understanding the Different Evaluation Metrics for Clustering

The primary aim of clustering is not just to make clusters, but to make good and meaningful ones. This is where we can make use of evaluation metrics.

Recall the first property of clusters we covered above. This is what inertia evaluates.

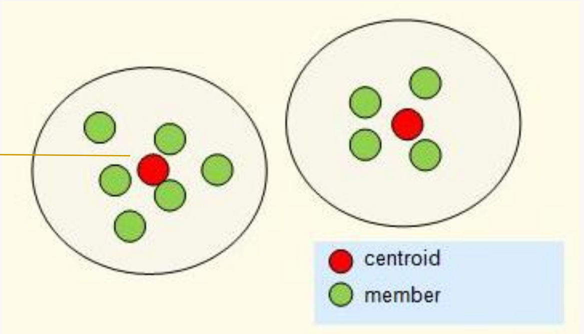
- **Inertia**
  - ***Inertia*** is a metric that is used to estimate how close the data points in a cluster are.



Intra cluster distance

  - This is **calculated as** the sum of squared distance for each point to its closest **centroid**.

centroid is the assigned cluster center.

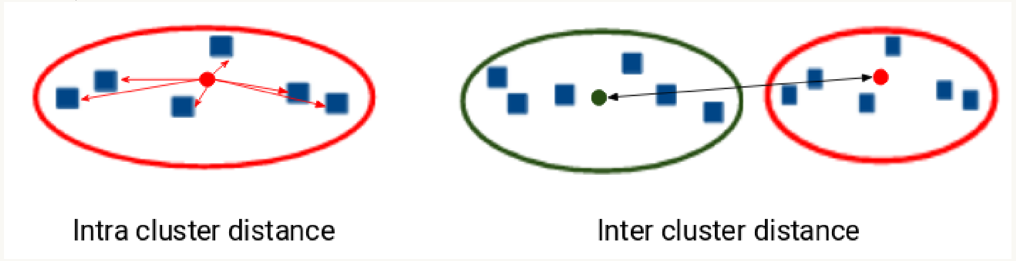


  - The intuition behind Inertia is that **clusters with lower Inertia are better, as it means closely related points form a cluster**.

Understanding the Different Evaluation Metrics for Clustering (Cont.)

So, inertia makes sure that the first property of clusters is satisfied. But it does not care about the second property – that different clusters should be as different from each other as possible. This is where Dunn index can come into action.

- **Dunn Index**
  - ***Dunn Index*** (or Davies–Bouldin index or Silhouette index) measures the distance between two clusters (how closely created various clusters are). **This distance** between the centroids of two different clusters is known as inter-cluster distance.



  - This is **calculated by:**

$$\text{Dunn Index} = \frac{\min(\text{Inter cluster distance})}{\max(\text{Intra cluster distance})}$$

  - We want to maximize the Dunn index, **The more the value of the Dunn index, the better will be the clusters.**

K-Means Clustering

- What is K-Means Clustering?
- How does K-Means work?
- Challenges with K-Means Algorithm
- How to choose the Right Number of Clusters in K-Means?
- Implementing K-Means Clustering in Python
- Pros and Cons associated with K-Means Clustering.
    - **Dealing with Outliers**

Remember!

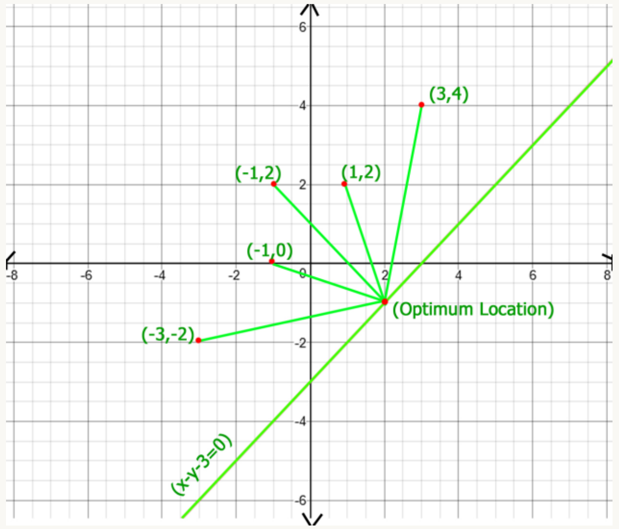
K-Nearest Neighbour	K-Means Clustering
▪ Supervised Technique	▪ Unsupervised Technique
▪ Used for Classification or Regression	▪ Used for Clustering
▪ ‘K’ in KNN represents the number of nearest neighbours used to classify or predict in case of continuous variable/regression	▪ ‘K’ in K-Means represents the number of clusters the algorithm is trying to identify or learn from the data

What is K-Means Clustering?

- **K-means** is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid. Also, it is a partitional clustering algorithm

Its main objective is to minimize the sum of Euclidean distance between the points and their respective cluster centroid. (minimizing the sum of squared error (SSE))

$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2$$



$\boldsymbol{C_j}$: is the j th cluster

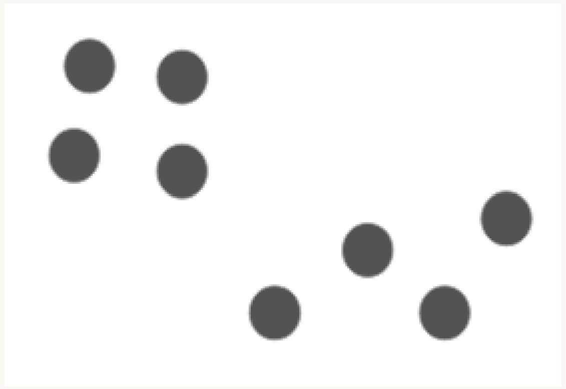
$\mathbf{m}_j$: is the centroid of cluster C_j (the mean vector of all the data points in C_j)

$dist(\mathbf{x}, \mathbf{m}_j)$ is the distance between data point $\mathbf{x}$ and centroid $\mathbf{m}_j$.

How does K-Means work?

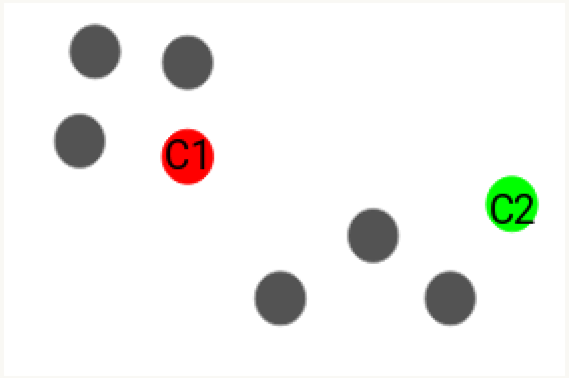
- Let’s take an example to understand how K-Means actually works:

We have these 8 points, and we want to apply k-means to create clusters for these points. Here’s how we can do it.



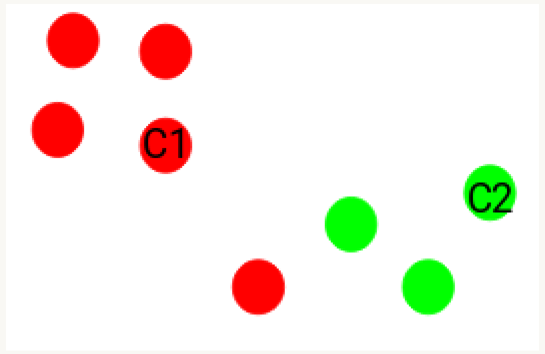
- **Step 1: Choose the number of clusters k**
  - The first step in k-means is to pick the number of clusters, k.

- **Step 2: Select k random points from the data as centroids**
  - We randomly select the centroid for each cluster.
  - Let’s say we want to have 2 clusters, so k is equal to 2 here.
  - Then randomly select the centroid.

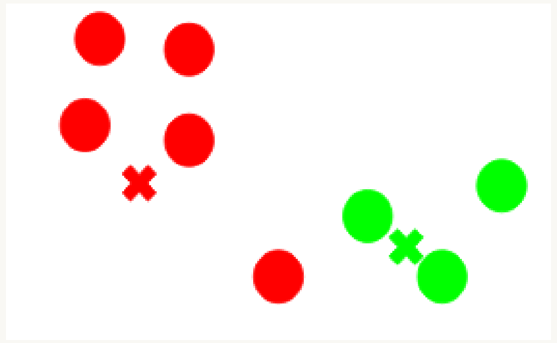


How does K-Means work? (Cont.)

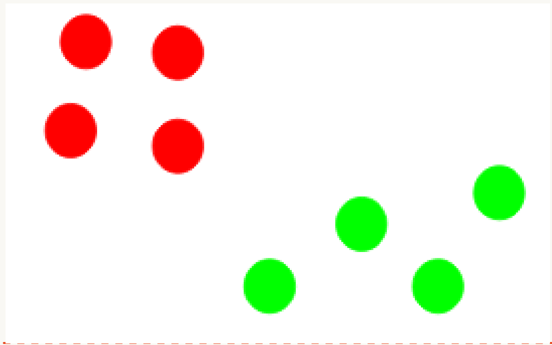
- **Step 3: Assign all the points to the closest cluster centroid**
  - Once we have initialized the centroids, we assign each point to the closest cluster centroid:



- **Step 4: Recompute the centroids of newly formed clusters**
  - Now, once we have assigned all of the points to either cluster, the next step is to compute the centroids of newly formed clusters:



- **Step 5: Repeat steps 3 and 4**
  - The step of computing the centroid and assigning all the points to the cluster based on their distance from the centroid is a single iteration.



How does K-Means work? (Cont.)

- **Stopping Criteria for K-Means Clustering**

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

  - **Centroids** of newly formed clusters do not change

    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.

  - **Points** remain in the same cluster

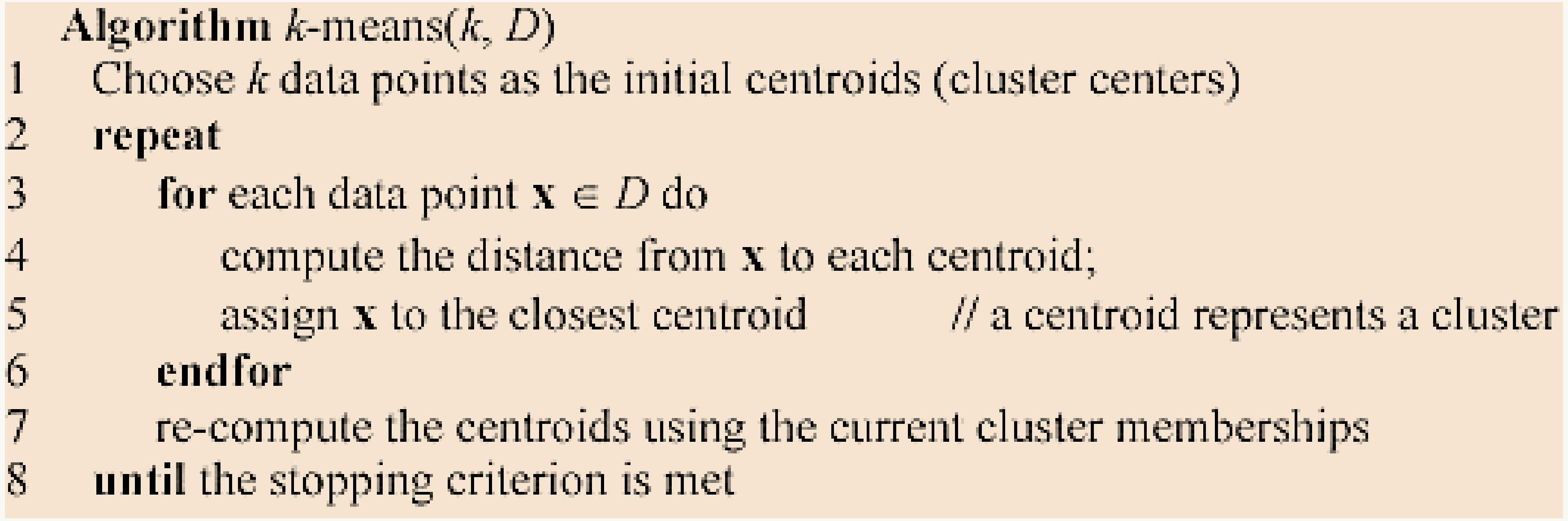
    Another clear sign that we should stop the training process if the points remain in the same cluster even after training the algorithm for multiple iterations.

  - Maximum number of **iterations** are reached

    Finally, we can stop the training if the maximum number of iterations is reached. Suppose if we have set the number of iterations as 80. The process will repeat for 80 iterations before stopping.

How does K-Means work? (Cont.)

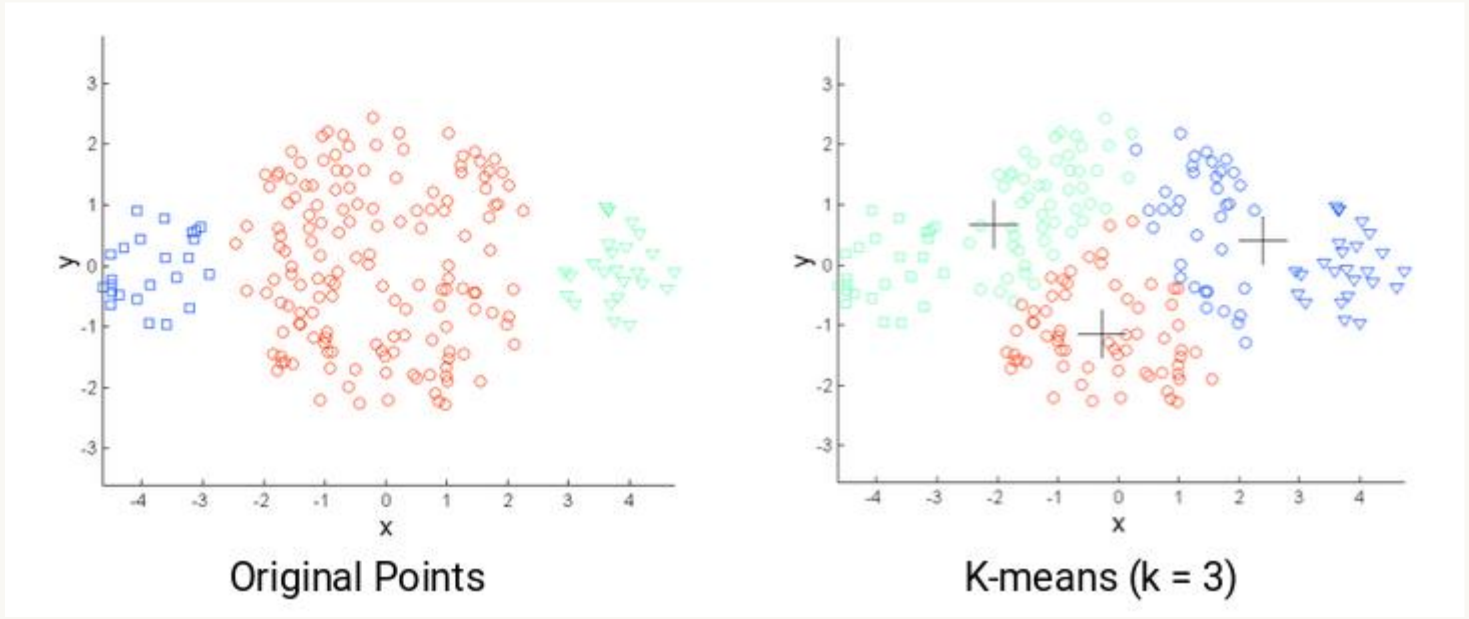
- **K-Means Clustering Algorithm**



```
  Algorithm k-means(k, D)
1   Choose k data points as the initial centroids (cluster centers)
2   repeat
3       for each data point $\mathbf{x} \in D$ do
4           compute the distance from $\mathbf{x}$ to each centroid;
5           assign $\mathbf{x}$ to the closest centroid        // a centroid represents a cluster
6       endfor
7       re-compute the centroids using the current cluster memberships
8   until the stopping criterion is met
```

Challenges with K-Means Algorithm

- One of the common challenges we face while working with K-Means is that **the size of clusters is different.**

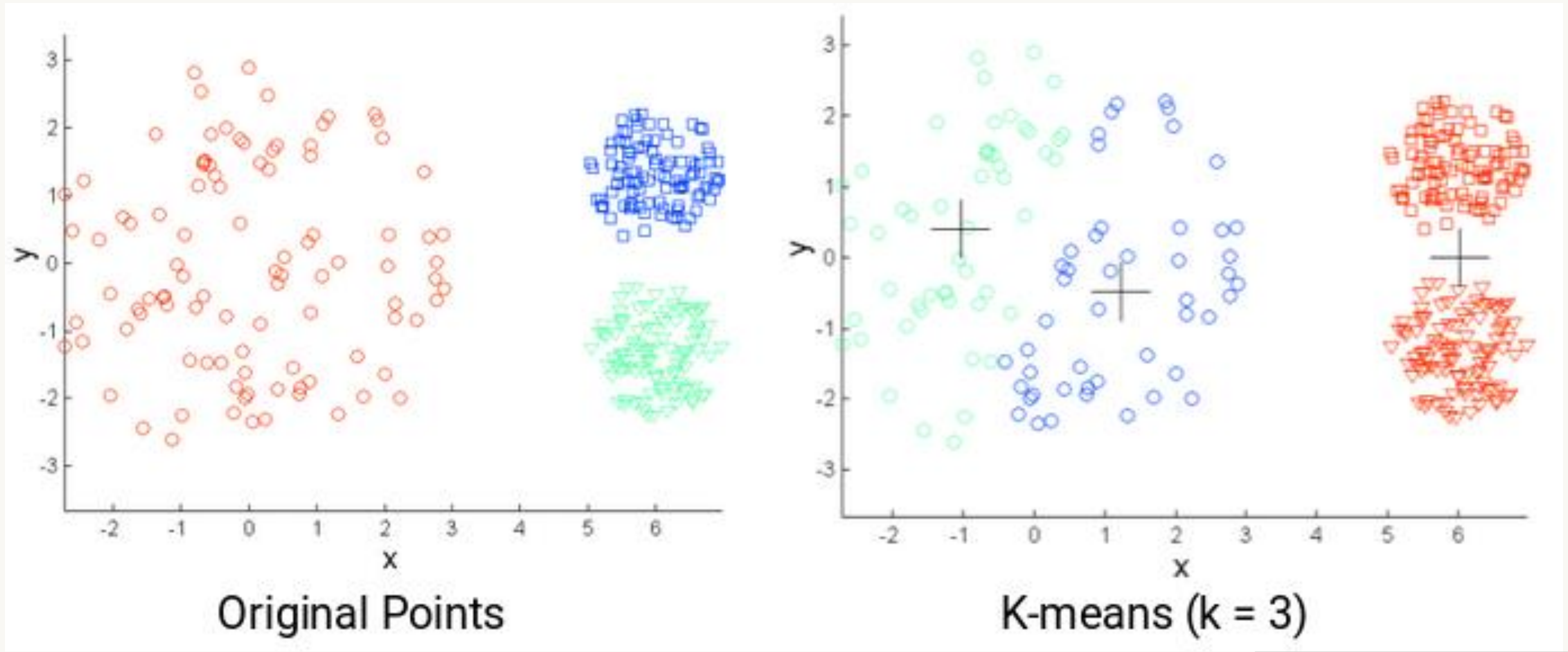


The left and the right most clusters are of smaller size compared to the central cluster.

Now, if we apply k-means clustering on these points, the results will be something like this!!

Challenges with K-Means Algorithm (Cont.)

- Another challenge with k-means is **when the densities of the original points are different.**



The points in the red cluster are spread out whereas the points in the remaining clusters are closely packed together.

Now, if we apply k-means on these points, we will get clusters like this!!

We can see that the compact points have been assigned to a single cluster. Whereas the points that are spread loosely but were in the same cluster, have been assigned to different clusters.

Not ideal so what can we do about this?

Challenges with K-Means Algorithm (Cont.)

- **One of the solutions is to use a higher number of clusters.** So, in all the previous scenarios, instead of using 3 clusters, we can have a bigger number. Perhaps setting k=10 might lead to more meaningful clusters.
- **Remember** how we randomly initialize the centroids in k-means clustering? Well, this is also potentially problematic because we might get different clusters every time.
- So, **to solve this problem of random initialization**, there is an algorithm called K-Means++ that can be used to choose the initial values, or the initial cluster centroids, for K-Means.

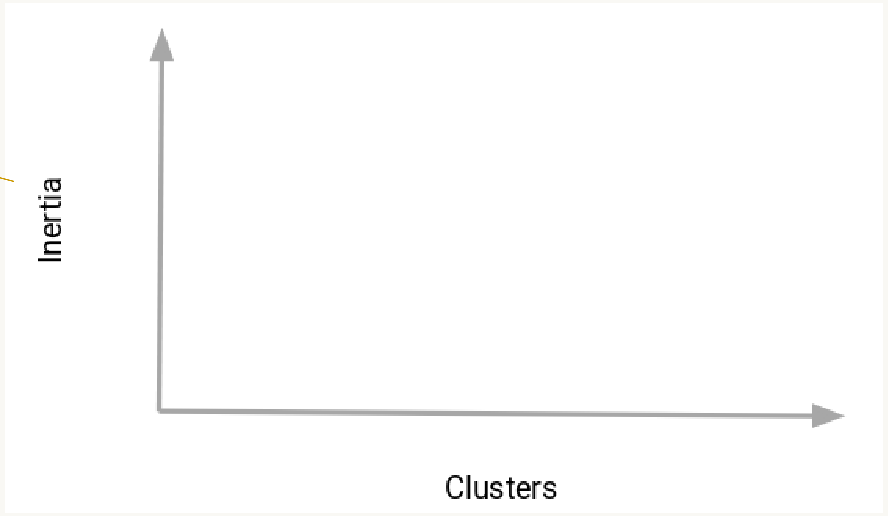
How to choose the Right Number of Clusters in K-Means?

The maximum possible number of clusters will be equal to the number of observations in the dataset.

- **How can we decide the optimum number of clusters?**

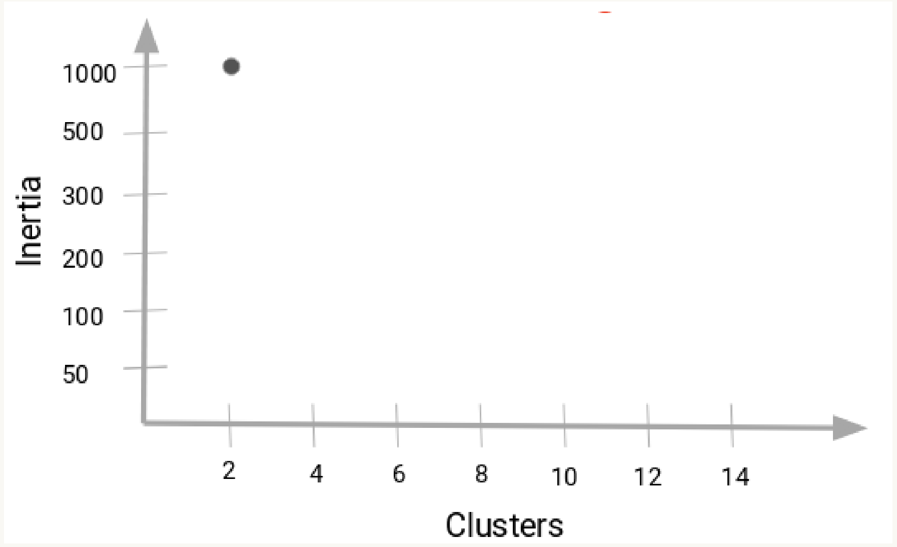
One thing we can do is **plot a graph, also known as an elbow curve,** where the x-axis will represent the number of clusters and the y-axis will be an evaluation metric. Let’s say inertia for now.

You can choose any other evaluation metric like the **Dunn index** as well

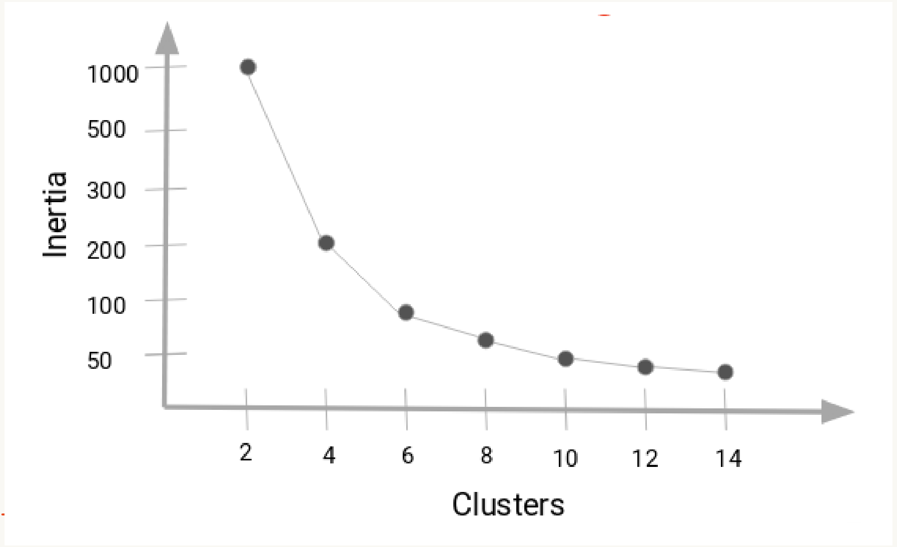


How to choose the Right Number of Clusters in K-Means?

- Next, we will <u>start with a small cluster value</u>, let's say 2. Train the model using 2 clusters, <u>calculate the inertia for that model</u>, and finally plot it in the graph. Let's say we got an inertia value of around 1000.



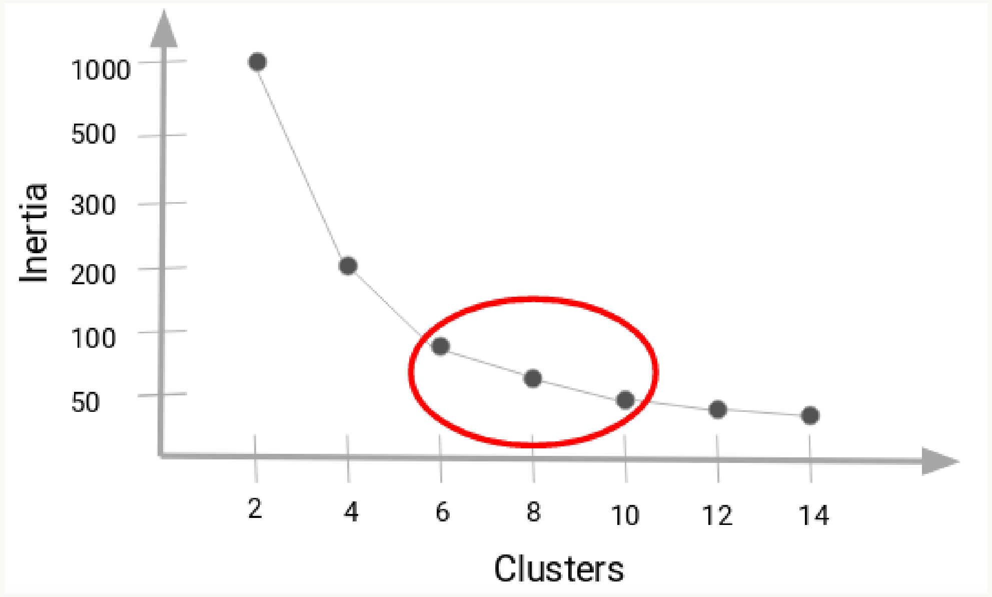
- Now, we will increase the number of clusters, train the model again, and plot the inertia value.



- **Note**: When we changed the cluster value from 2 to 4, the inertia value reduced very sharply. Then, it eventually becomes constant as we increase the number of clusters.

How to choose the Right Number of Clusters in K-Means?

- **The cluster value where this decrease in inertia value** becomes constant can be chosen as the right cluster value for our data.



- We **can choose any number of clusters between 6 and 10**. We can have 7, 8, or even 9 clusters.
- You **must also look at** the computation cost while deciding the number of clusters. If we increase the number of clusters, the computation cost will also increase.
  - For example, if you do not have high computational resources, my advice is to choose a lesser number of clusters.

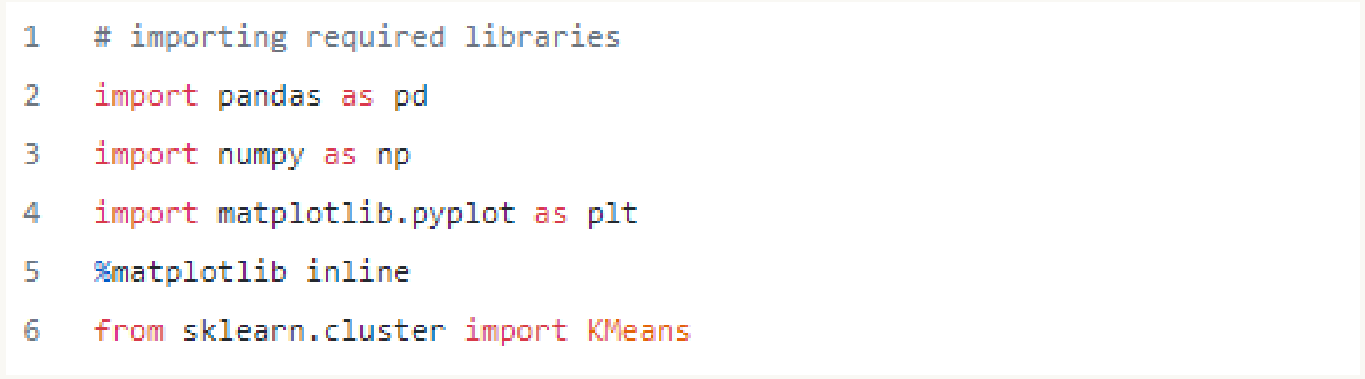
Implementing K-Means Clustering in Python

- We will be working on a **wholesale customer segmentation problem**. You can *download the dataset using the link below.*

The aim of this problem is to segment the clients of a wholesale distributor based on their annual spending on diverse product categories, like milk, grocery, etc.

So, let's start coding!

will first import the required libraries:

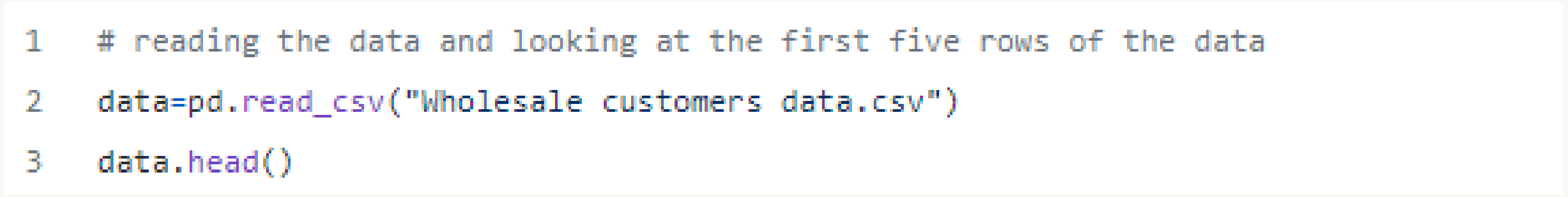


```python
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans
```

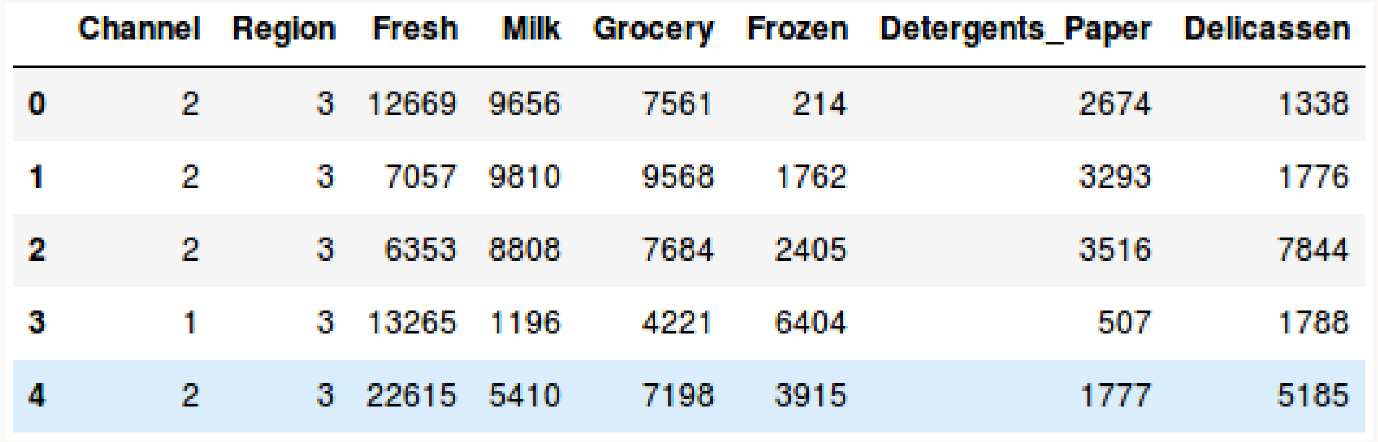
Download Dataset: *https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale%20customers%20data.csv*

Implementing K-Means Clustering in Python (Cont.)

Next, let’s **read the data** and **look at the first five rows**:



```python
# reading the data and looking at the first five rows of the data
data=pd.read_csv("Wholesale customers data.csv")
data.head()
```

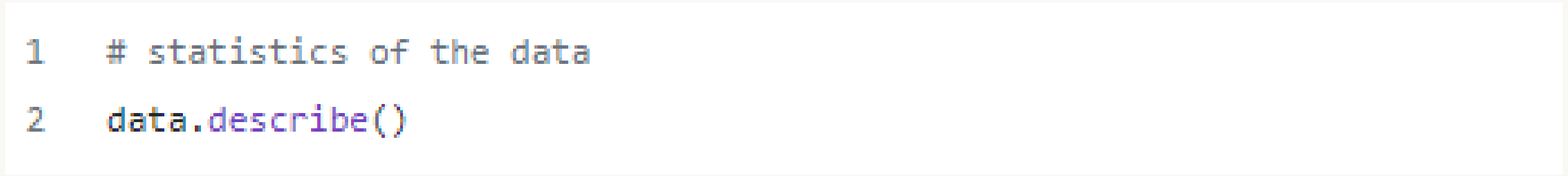


	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

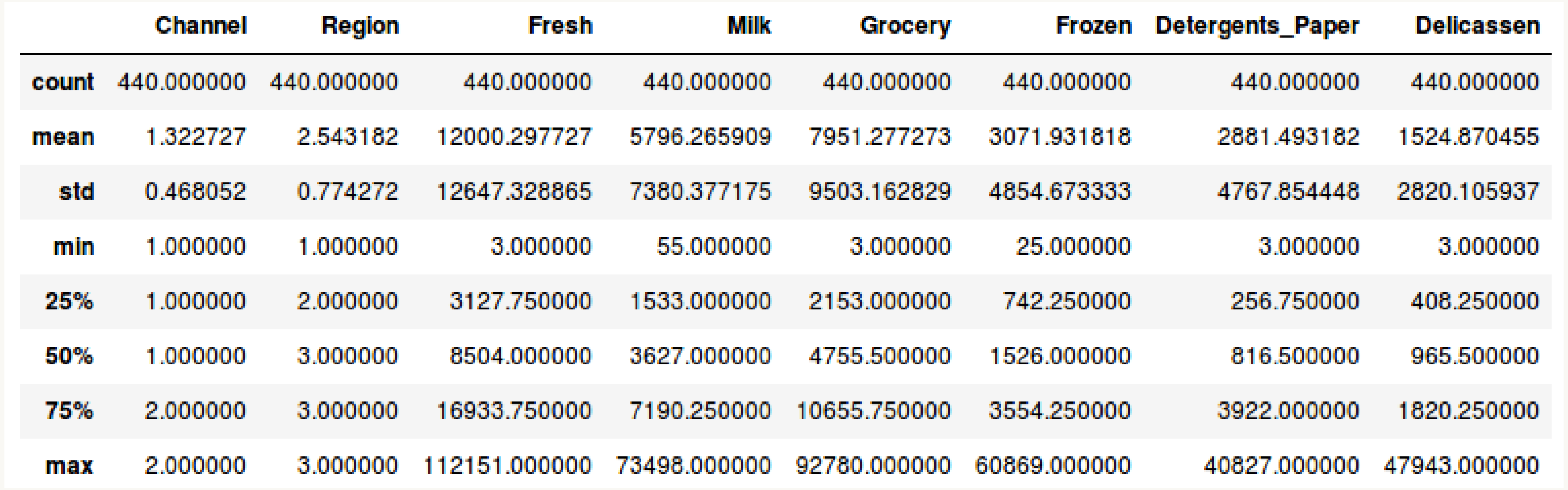
This shows the spending details of customers on different products like Milk, Grocery, Frozen, Detergents, etc.

Implementing K-Means Clustering in Python (Cont.)

Before start segmenting the customers based on the provided details, let's pull out some statistics related to the data:



```python
# statistics of the data
data.describe()
```

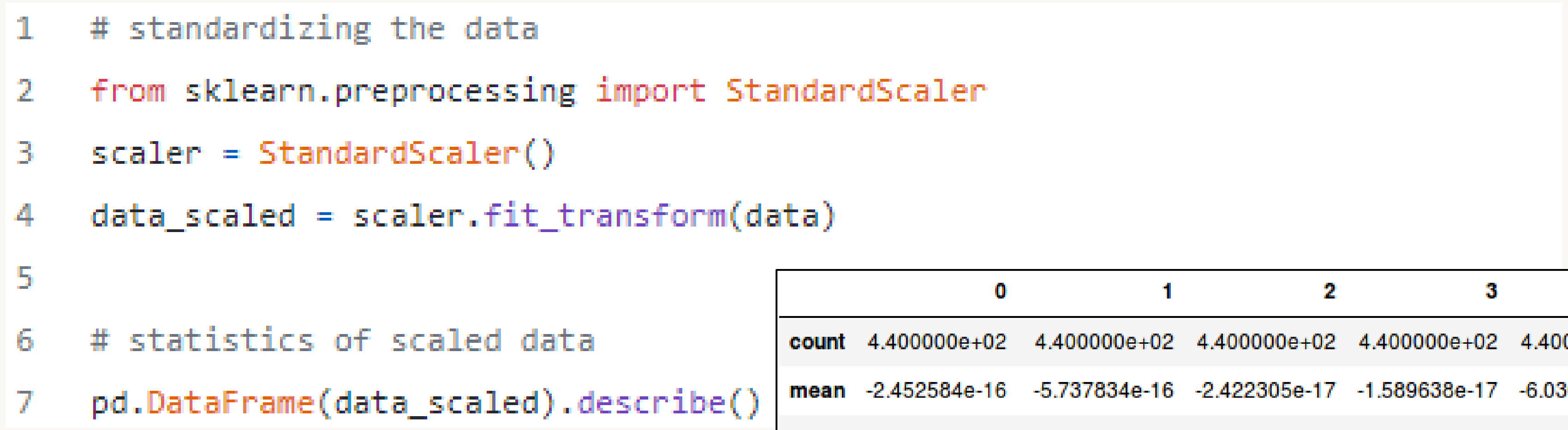


	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
count	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000	440.000000
mean	1.322727	2.543182	12000.297727	5796.265909	7951.277273	3071.931818	2881.493182	1524.870455
std	0.468052	0.774272	12647.328865	7380.377175	9503.162829	4854.673333	4767.854448	2820.105937
min	1.000000	1.000000	3.000000	55.000000	3.000000	25.000000	3.000000	3.000000
25%	1.000000	2.000000	3127.750000	1533.000000	2153.000000	742.250000	256.750000	408.250000
50%	1.000000	3.000000	8504.000000	3627.000000	4755.500000	1526.000000	816.500000	965.500000
75%	2.000000	3.000000	16933.750000	7190.250000	10655.750000	3554.250000	3922.000000	1820.250000
max	2.000000	3.000000	112151.000000	73498.000000	92780.000000	60869.000000	40827.000000	47943.000000

Here, we see that there is **a lot of variation in the magnitude of the data**. Variables like Channel and Region have *low magnitude* whereas variables like Fresh, Milk, Grocery, etc. have a *higher magnitude*.

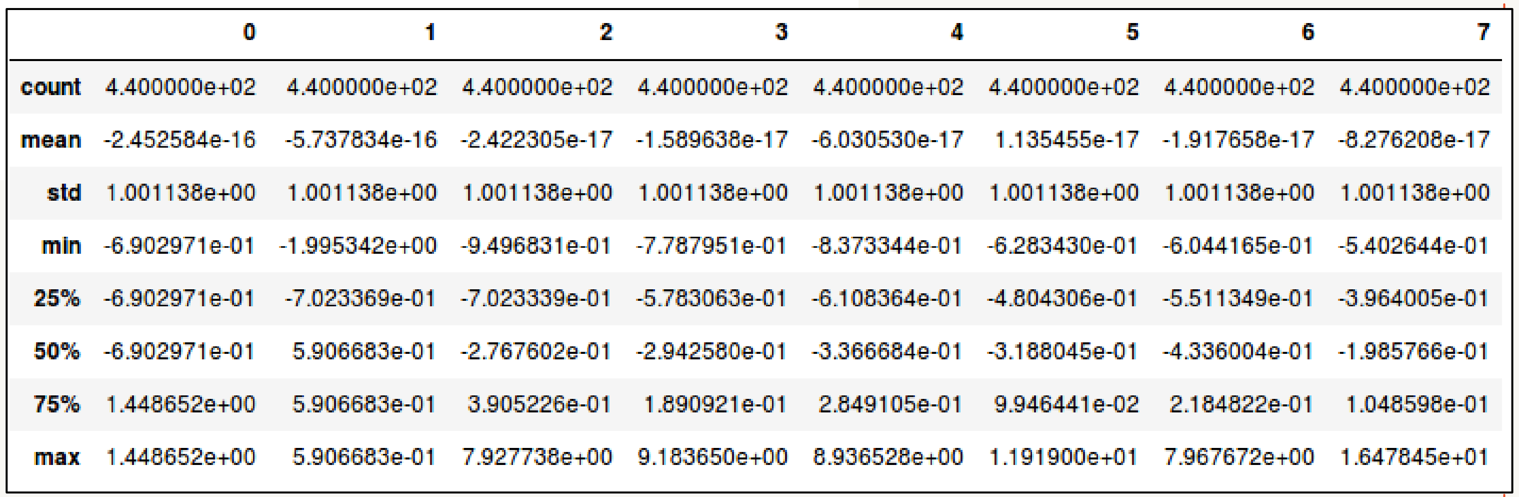
Implementing K-Means Clustering in Python (Cont.)

Since K-Means is a distance-based algorithm, this difference of magnitude can create a problem. **So, let’s first bring all the variables to the same magnitude**:



```python
# standardizing the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# statistics of scaled data
pd.DataFrame(data_scaled).describe()
```



	0	1	2	3	4	5	6	7
count	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02	4.400000e+02
mean	-2.452584e-16	-5.737834e-16	-2.422305e-17	-1.589638e-17	-6.030530e-17	1.135455e-17	-1.917658e-17	-8.276208e-17
std	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00	1.001138e+00
min	-6.902971e-01	-1.995342e+00	-9.496831e-01	-7.787951e-01	-8.373344e-01	-6.283430e-01	-6.044165e-01	-5.402644e-01
25%	-6.902971e-01	-7.023369e-01	-7.023339e-01	-5.783063e-01	-6.108364e-01	-4.804306e-01	-5.511349e-01	-3.964005e-01
50%	-6.902971e-01	5.906683e-01	-2.767602e-01	-2.942580e-01	-3.366684e-01	-3.188045e-01	-4.336004e-01	-1.985766e-01
75%	1.448652e+00	5.906683e-01	3.905226e-01	1.890921e-01	2.849105e-01	9.946441e-02	2.184822e-01	1.048598e-01
max	1.448652e+00	5.906683e-01	7.927738e+00	9.183650e+00	8.936528e+00	1.191900e+01	7.967672e+00	1.647845e+01

The magnitude looks similar now!!

Implementing K-Means Clustering in Python (Cont.)

- Next, let’s create a ***kmeans*** function and fit it on the data:

```python
# defining the kmeans function with initialization as k-means++
kmeans = KMeans(n_clusters=2, init='k-means++')

# fitting the k means algorithm on scaled data
kmeans.fit(data_scaled)
```

We have initialized **two clusters** and pay attention – the initialization is not random here. We have used the k-means++ initialization which generally produces better results.

- Let’s evaluate how well the formed clusters are. To do that, we will calculate the **inertia** of the clusters:

```python
# inertia on the fitted data
kmeans.inertia_
```

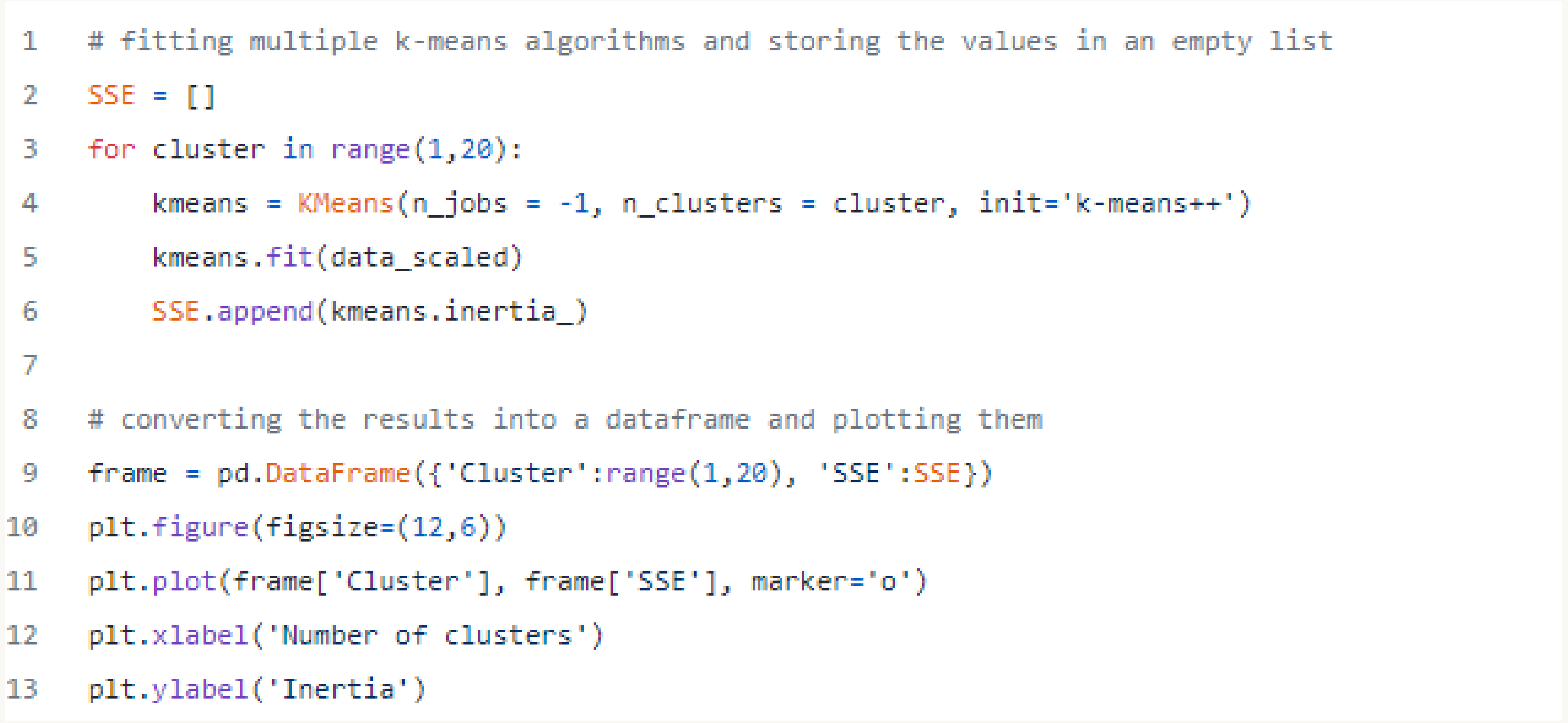
Output: 2599.38555935614

We got an inertia value of almost 2600.!!

Implementing K-Means Clustering in Python (Cont.)

- Now, let’s see **how we can use the elbow curve** to determine the optimum number of clusters in Python.

We will first fit multiple k-means models and in each successive model, we will increase the number of clusters. We will store the inertia value of each model and then plot it to visualize the result:



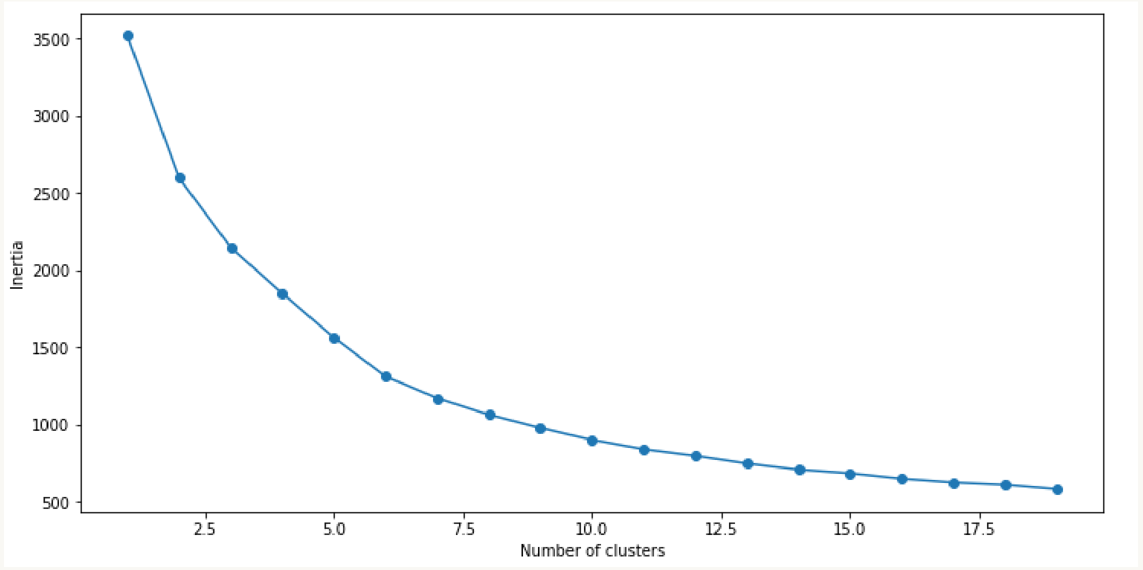
```python
# fitting multiple k-means algorithms and storing the values in an empty list
SSE = []
for cluster in range(1,20):
    kmeans = KMeans(n_jobs = -1, n_clusters = cluster, init='k-means++')
    kmeans.fit(data_scaled)
    SSE.append(kmeans.inertia_)

# converting the results into a dataframe and plotting them
frame = pd.DataFrame({'Cluster':range(1,20), 'SSE':SSE})
plt.figure(figsize=(12,6))
plt.plot(frame['Cluster'], frame['SSE'], marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

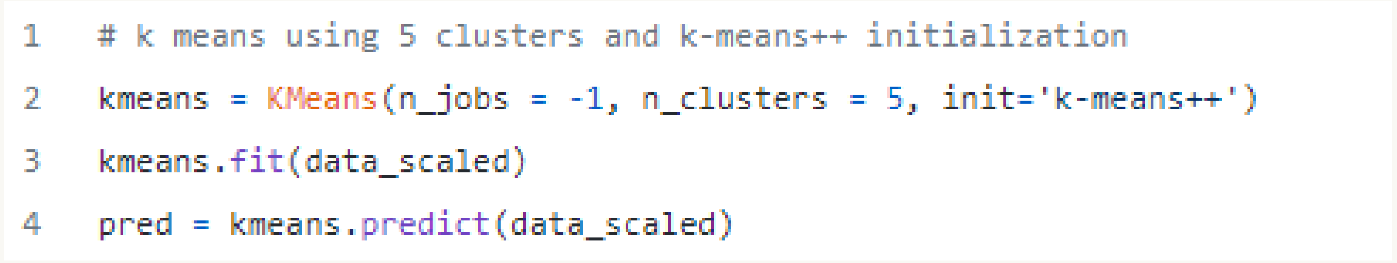
Implementing K-Means Clustering in Python (Cont.)

- **Can you tell the optimum cluster value from this plot?**

  Looking at the following elbow curve, **we can choose any number of clusters between 5 to 8**.



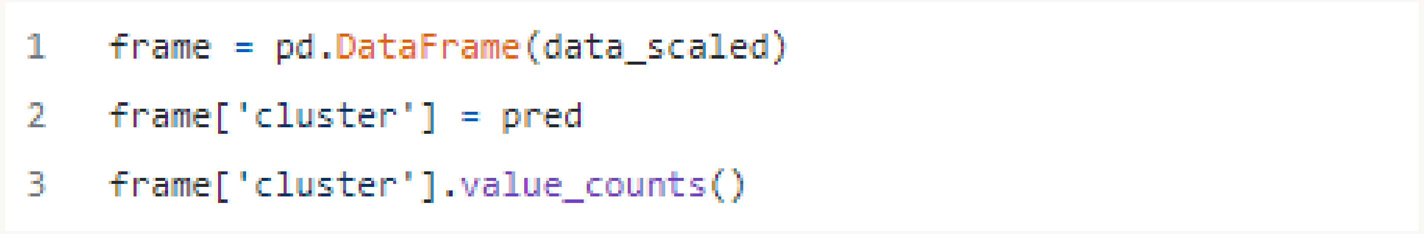
Let’s set the number of clusters as 5 and fit the model:



```
# k means using 5 clusters and k-means++ initialization
kmeans = KMeans(n_jobs = -1, n_clusters = 5, init='k-means++')
kmeans.fit(data_scaled)
pred = kmeans.predict(data_scaled)
```

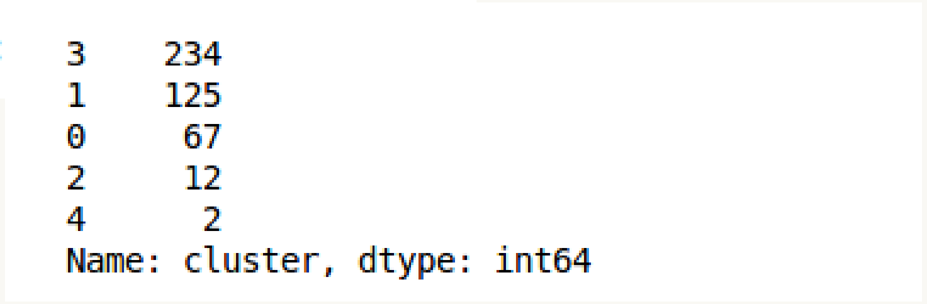
Implementing K-Means Clustering in Python (Cont.)

- **Finally, let’s look at the value count of points in each of the formed clusters:**



```python
frame = pd.DataFrame(data_scaled)
frame['cluster'] = pred
frame['cluster'].value_counts()
```

Output:

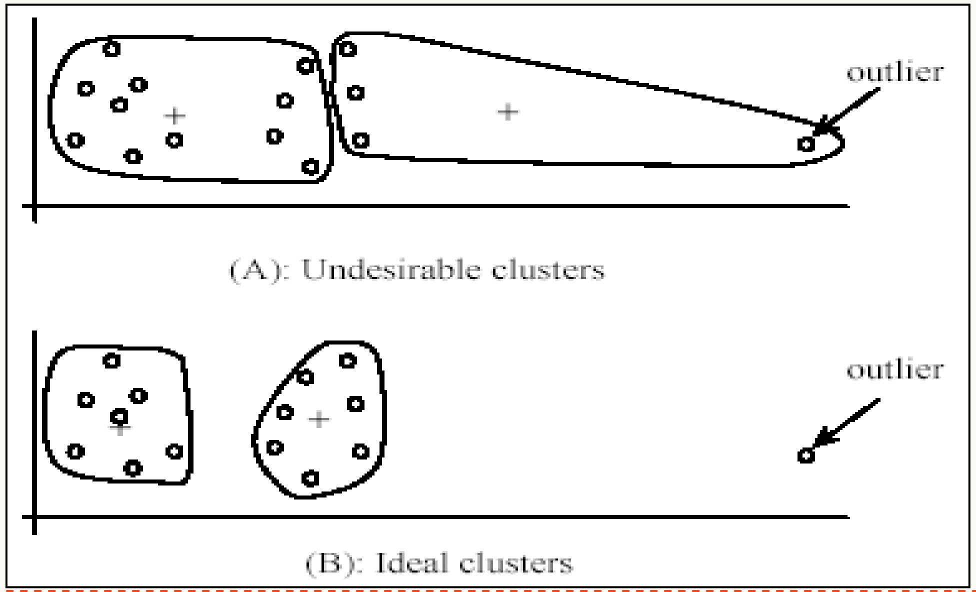


```
3    234
1    125
0     67
2     12
4      2
Name: cluster, dtype: int64
```

So, there are **234** data points belonging to cluster 4 (index 3), then **125** points in cluster 2 (index 1), and so on.

Pros and Cons associated with K-Means

- **Pros**:
  - Simple: easy to understand and to implement.
  - Efficient: Time complexity: O(***tkn***), where:
    - ***n*** is the number of data points,
    - ***k*** is the number of clusters, and
    - ***t*** is the number of iterations
  - Since both k and t are small. k-means is considered a linear algorithm.
  - It terminates at a local optimum if SSE is used. The global optimum is hard to find due to complexity.
- **Cons**:
  - The algorithm is only applicable if the mean is defined.
  - The user may need to specify ***k***.
  - The algorithm is sensitive to outliers:
    - They 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.



Dealing with Outliers

In Clustering Context:

- **One method is** to remove some data points in the clustering process that are much further away from the centroids than other data points.
  - To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them
- **Another method is** to perform random sampling. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small.
  - Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

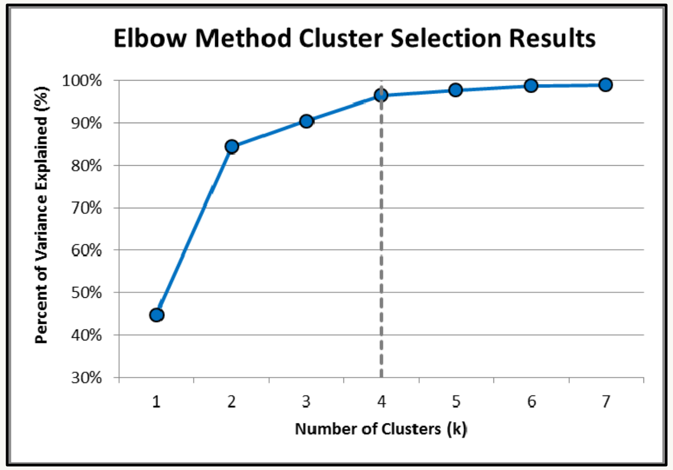
Generally:

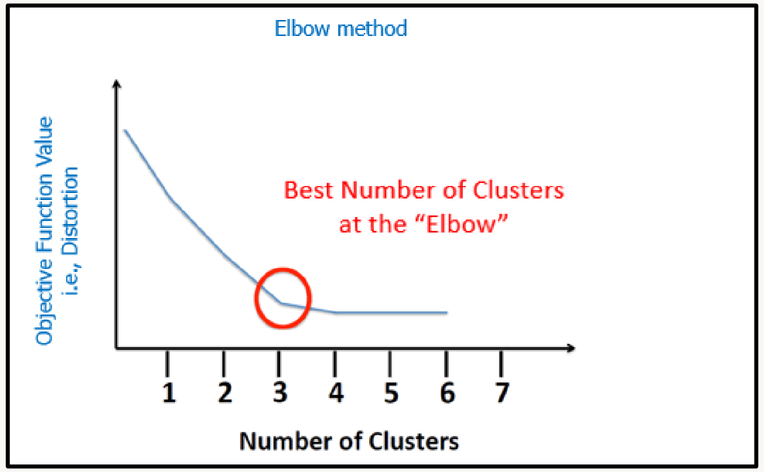
- Doing nothing.
- Deleting / Trimming
- Winsorizing.
- Transformation.
- Binning
- Use Robust Estimators
- Imputing

Summary

- Clustering has along history and still active
  - There are a huge number of clustering algorithms
  - More are still coming every year.
- We only introduced several main algorithms. There are many others, e.g.,
  - density based algorithm, sub-space clustering, scale-up methods, neural networks-based methods, fuzzy clustering, co-clustering, etc.
- Clustering is hard to evaluate, but very useful in practice. This partially explains why there are still a large number of clustering algorithms being devised every year.
- Clustering is highly application dependent and to some extent subjective.

- The **Elbow curve** is one way to solve the problem of the best initialization of **K** value.



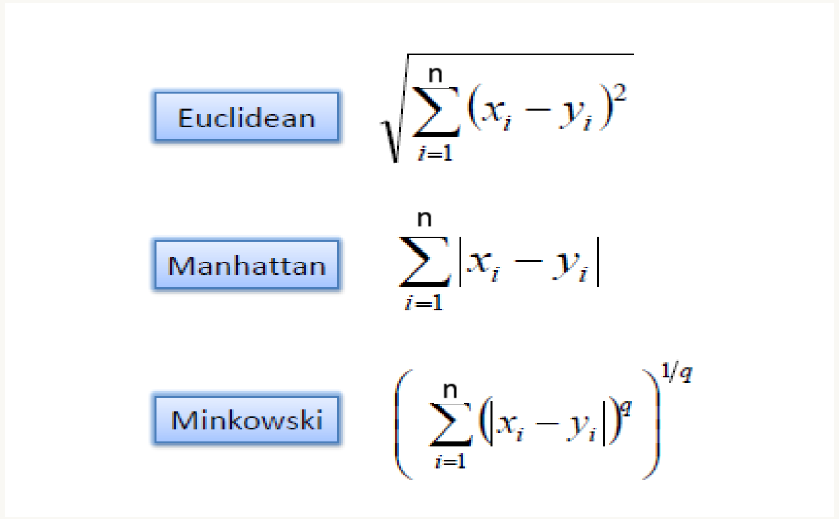


Appendix

- K-Means Visualization

  https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

- Some of the Distance measures



Euclidean	$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
Manhattan	$\sum_{i=1}^{n}\lvert x_i - y_i\rvert$
Minkowski	$\left(\sum_{i=1}^{n}(\lvert x_i - y_i\rvert)^q\right)^{1/q}$

End of Unsupervised Learning!!

Any questions?