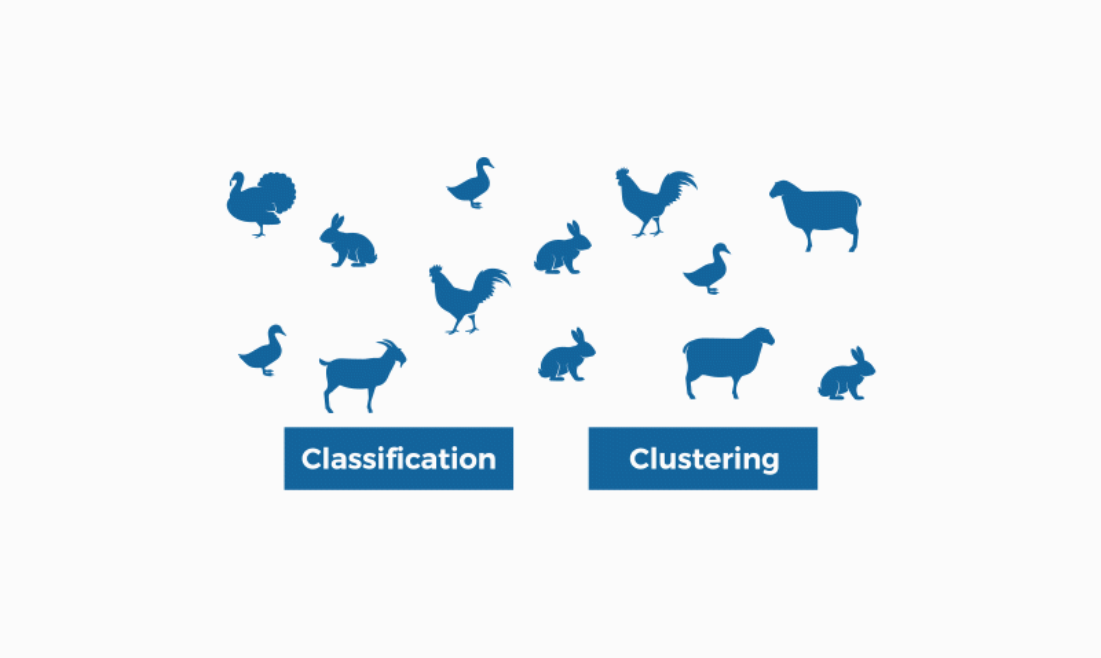
DSC510: Introduction to Data Science and Analytics
Lab 9: Clustering

University of Cyprus
Department of
Computer Science

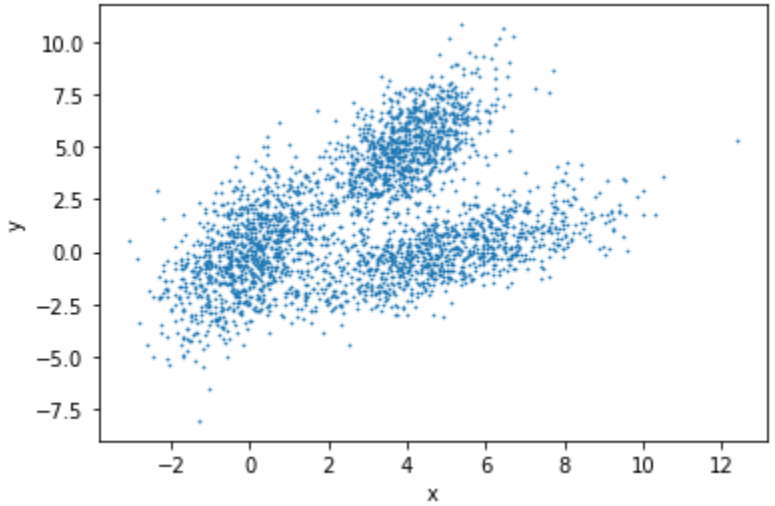
Pavlos Antoniou

Office: B109, FST01

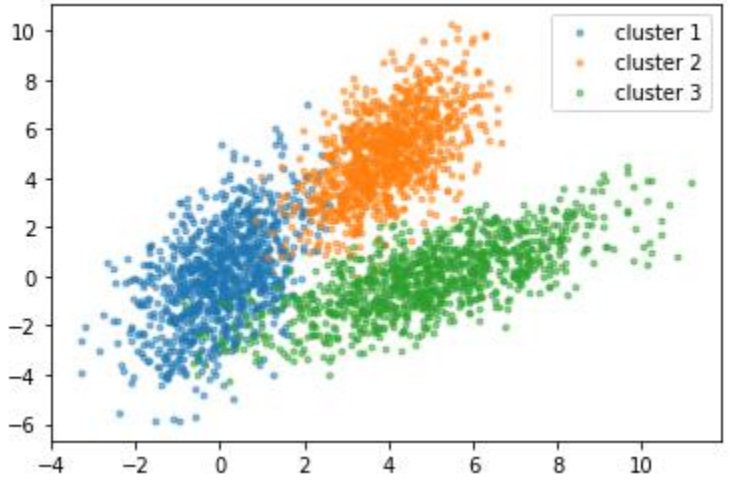


Clustering

- Unsupervised Machine Learning process of dividing the dataset into groups consisting of similar data points
- Each group is called a cluster and contains data points with high similarity and with low similarity with data points in other clusters



Samples in two-dimensional (2 features) space BEFORE clustering



Samples in two-dimensional (2 features) space AFTER clustering in three groups

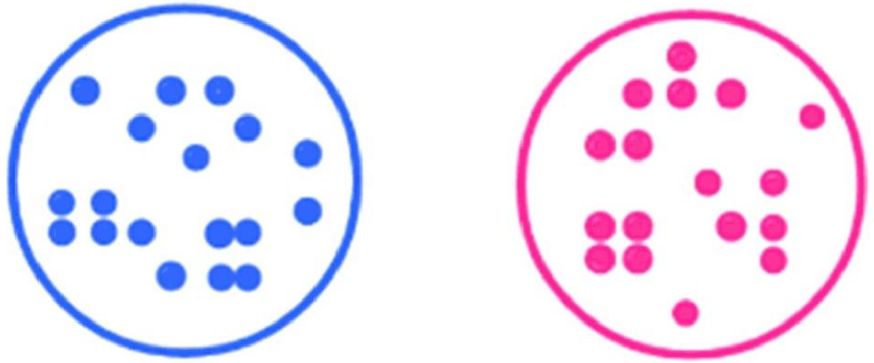
The number of clusters (k) is a hyper parameter of clustering models: needs to be defined prior performing clustering

Types of Clustering

- Exclusive (non-overlapping) clustering
- Overlapping clustering
- Hierarchical clustering

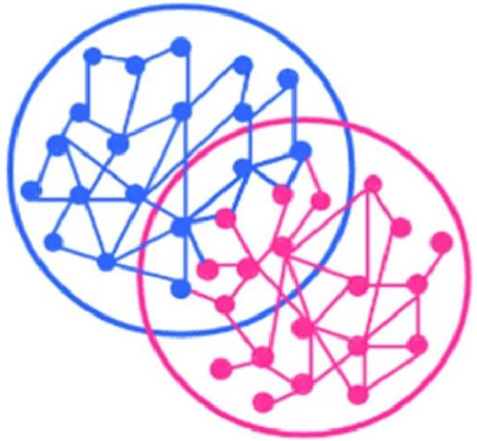
Exclusive (non-overlapping) clustering

- Hard clustering
- Data point belongs exclusively to one of the identified disjoint clusters
- Example algorithm: K-means Clustering



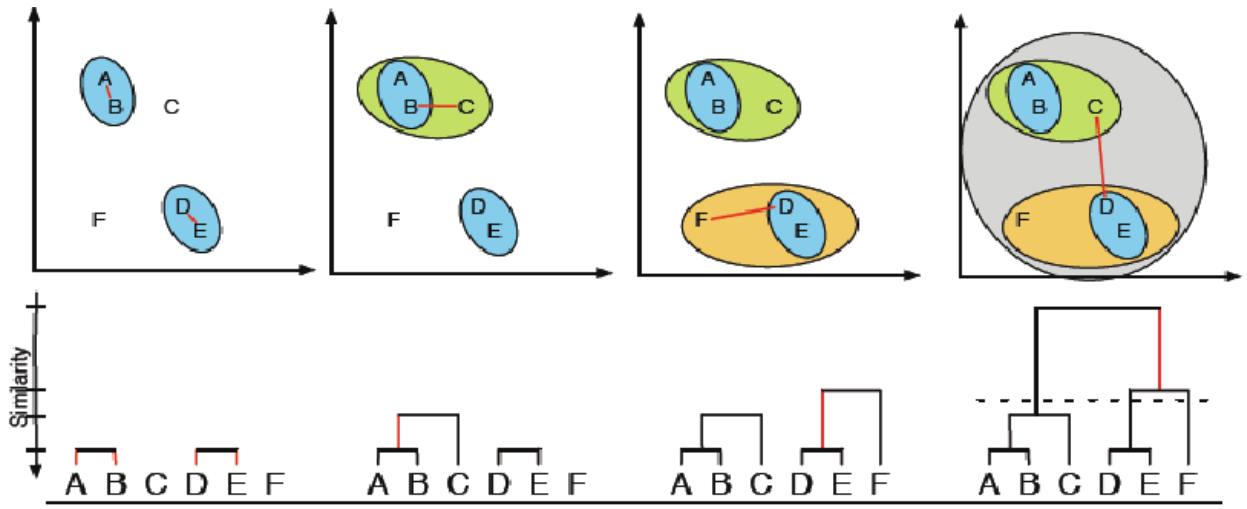
Overlapping clustering

- Soft clustering
- Data point belongs to more than one clusters
  - data point can belong to a cluster with some degree of membership (probability) between 0 and 1
- Example algorithm: Fuzzy / C-means Clustering



Hierarchical clustering

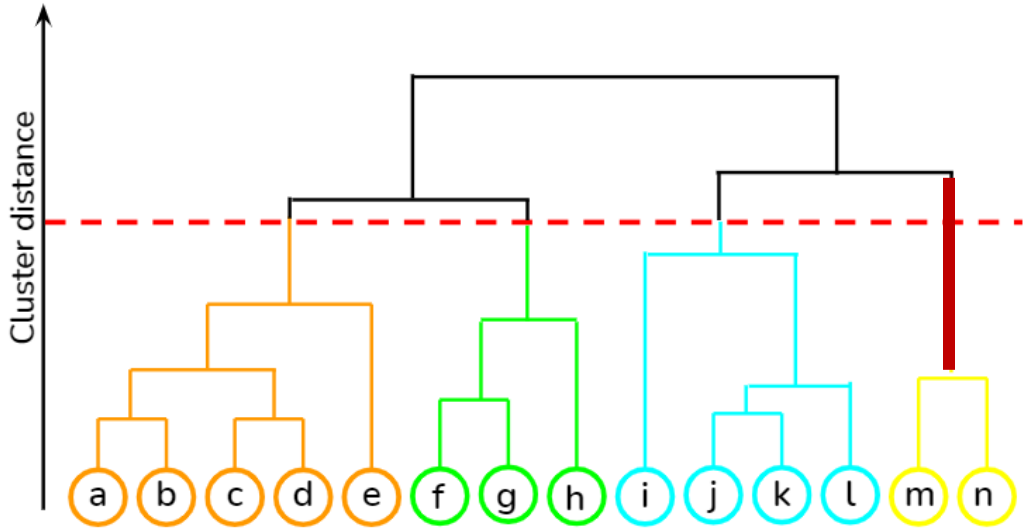
- Hierarchy of clusters is identified using either agglomerative (bottom-up) or divisive (top-down) approaches
- Agglomerative approach: we consider each data point as one cluster and we iteratively merge them according to a criterion e.g. distance
  - Create a matrix that contains all pairwise (linkage) distances among all clusters
  - The closest two clusters are merged
  - Main output of Agglomerative clustering is a dendrogram which shows the hierarchical relationship between clusters
  - The height of the vertical dendrogram lines reflects the distance (or dissimilarity) between the 2 merged clusters



  - Iterative process continues until all the clusters are merged together or the desired number of clusters is reached
  - Example algorithm: Agglomerative clustering

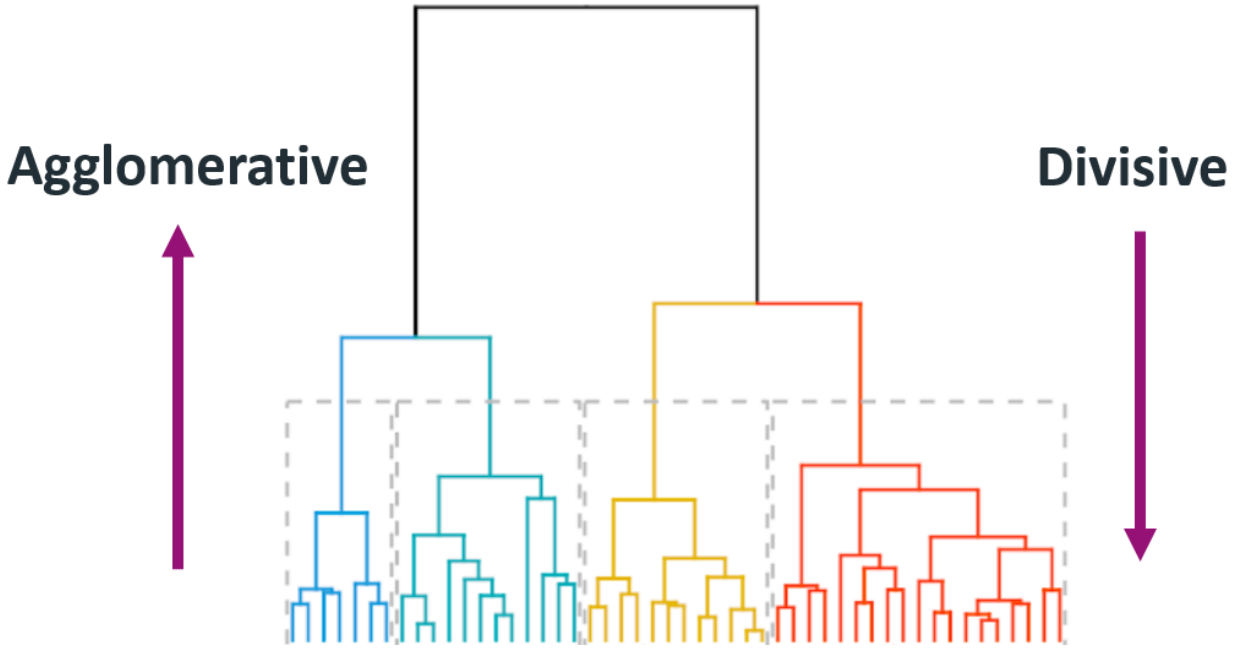
Which is the optimal num of clusters?

- To find the “optimal” number of clusters, you:
  - Look for the highest vertical line between successive merges (horizontal lines)
  - Draw a horizontal cut across the dendrogram at that gap.
  - The number of clusters = the number of vertical lines the cut passes through.
- Intuitively:
  - Cutting the dendrogram at a large jump in linkage distance separates groups that are significantly different
  - Cutting too low → too many small clusters
  - Cutting too high → merges dissimilar clusters



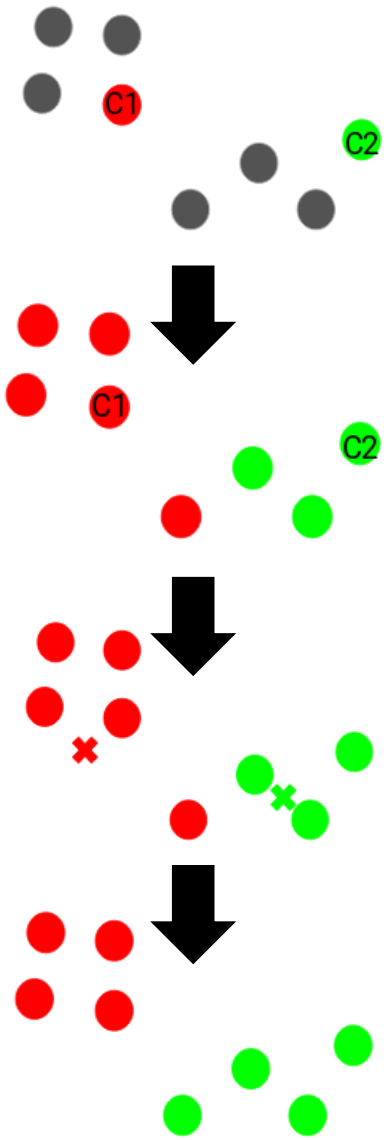
Hierarchical clustering

- Divisive approach: we consider the whole dataset as one cluster and iteratively split them into multiple clusters according to some evaluation criterion (e.g, distance-based criteria, statistical or variance-based criteria)
  - Example algorithm: Bisecting K-Means



K-means: exclusive clustering algorithm

- User provides the number of clusters K
- K-means iterative process involves the following steps:
  1. Selects K samples from data, or generates K points to be used as centroids
  2. Assigns all samples to the closest cluster centroid
  3. Recomputes the centroids of newly formed clusters
     - Centroid can be calculated as the mean of data points of the cluster
       - E.g. data points: (80, 56), (75, 53), (60, 50), (68,54) Centroid: (70.75, 53.25)
  4. Repeats steps 2 and 3
- Stopping criteria for K-means:
  - Centroids of newly formed clusters do not change
  - Samples remain in the same cluster
  - Maximum number of iterations is reached



K-means Issues

- Works only with numerical data
  - Encoding techniques needed to convert categorical data to numerical
- Distance-based algorithm: Euclidean distance
  - Features should be of similar scale – data scaling needed e.g. Standard scaler, Robust scaler, etc.
- Depends on initial centroid selection
  - The more optimal the positioning of these initial centroids, the fewer iterations of the *k*-means algorithm will be required for convergence
    - Strategic consideration to the initialization of these initial centroids could prove useful
  - Available initialization strategies:
    - Random selection: prone to bad selection (e.g. very close to each other)
    - K-means++ is a smart centroid initialization technique which selects centroids being as far as possible from one another

K-means bottom line

- Easy to implement and use
- Needs to scale features if in different scales
- Good initial centroid selection method available: K-means++
- Need to set K prior running the algorithm

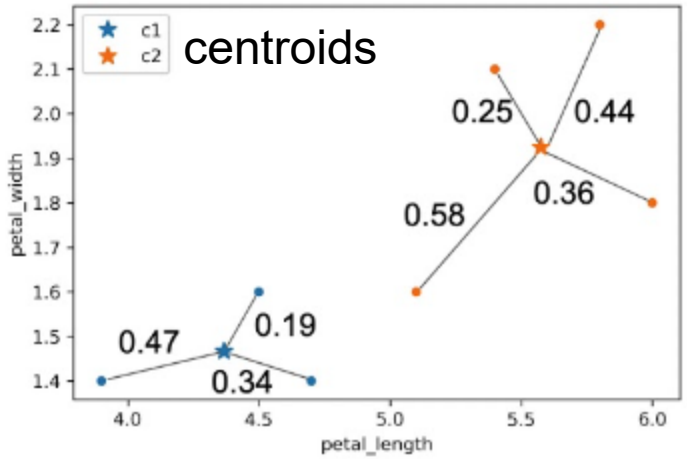
Choosing the best K (number of clusters)

- How can we determine the “best” value of *K?*
  - Is there an objective method?

- An estimation can be obtained using the following techniques:
  - Elbow method
  - Silhouette analysis

Elbow method parameters

- **Inertia**: The sum of squared distances* from each sample (data point or row) to its assigned cluster centroid
  - (*) Typically, the Euclidean distance metric is used
- **Distortion**: Weighted (by the cluster size) sum of squared distances from each sample (data point) to its assigned cluster centroid
- Example: use K-means, with K=2, to cluster 7 data points of a dataset having only 2 features in order to be able to visualize the distances in the 2-dimensional space and better understand the calculations below:

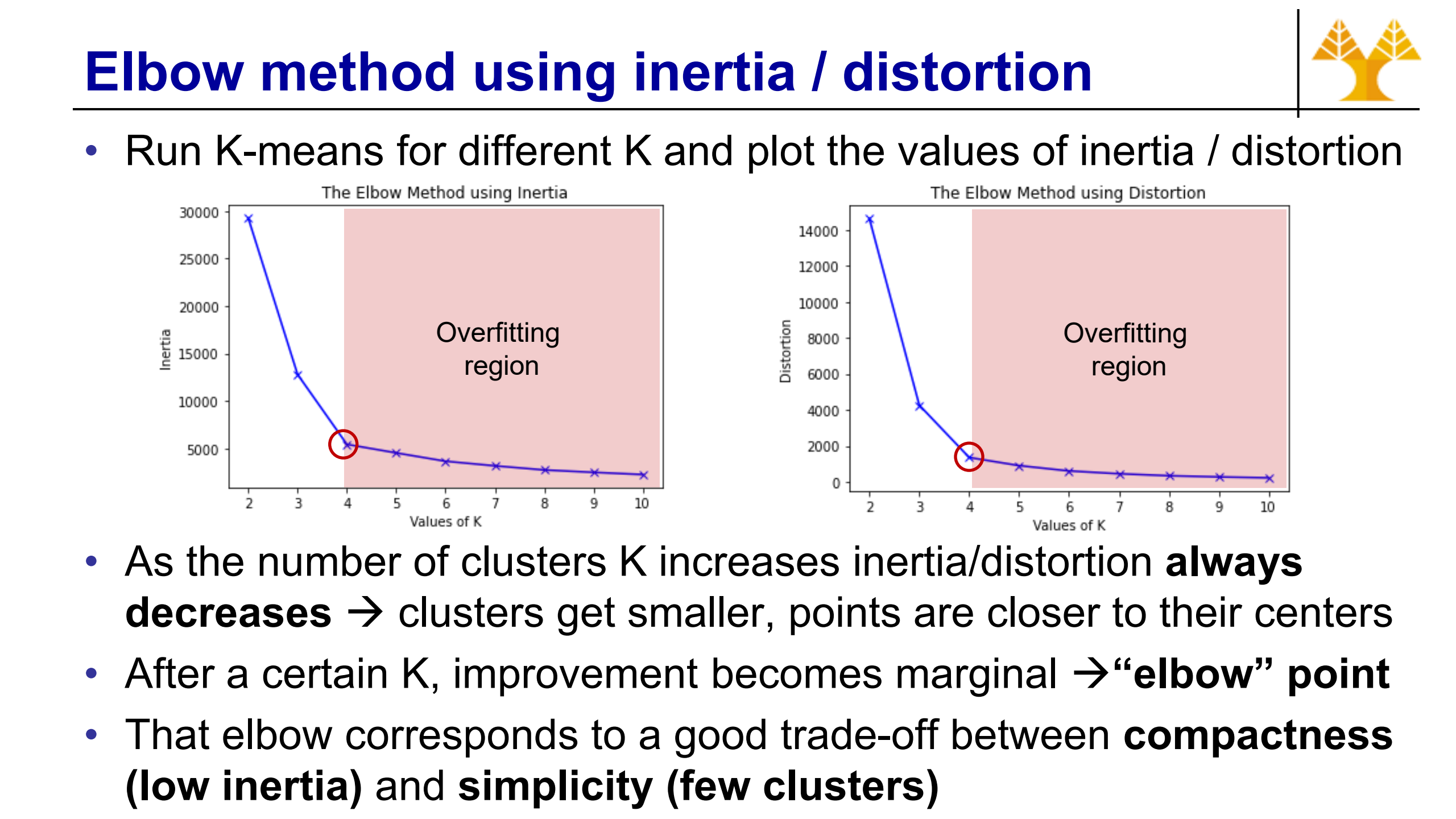


$$Inertia = 0.47^2 + 0.19^2 + 0.34^2 + 0.25^2 + 0.44^2 + 0.36^2 + 0.58^2$$

$$Distortion = \frac{(0.47^2 + 0.19^2 + 0.34^2)}{3} + \frac{(0.25^2 + 0.44^2 + 0.36^2 + 0.58^2)}{4}$$

- Inertia measures:
  - Compactness (Cohesion) → how tightly grouped the data points are within each cluster.
  - Explained variance → lower inertia means the clustering explains the data better (less error).
- Inertia does *not* measure:
  - Separation between clusters (it ignores how far clusters are from each other).
  - Cluster shape or density (it assumes spherical clusters, as in k-means).
- Distortion measures:
  - The average compactness per data point.
  - How much “error” or “information loss” occurs when representing data by its cluster centers.

Elbow method using inertia / distortion



- Run K-means for different K and plot the values of inertia / distortion

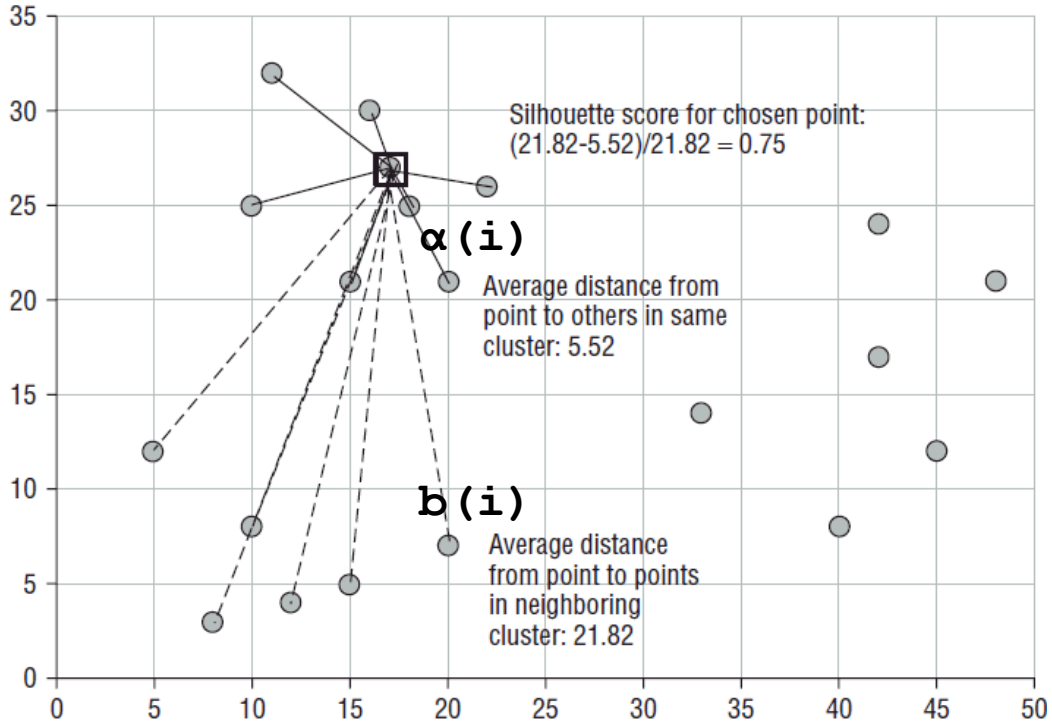
- As the number of clusters K increases inertia/distortion **always decreases** → clusters get smaller, points are closer to their centers
- After a certain K, improvement becomes marginal →**"elbow" point**
- That elbow corresponds to a good trade-off between **compactness (low inertia)** and **simplicity (few clusters)**

Silhouette score

- Measures how similar a data point is to its own cluster (compactness) compared to other clusters (separation)
  - Silhouette score for data point i :

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

  - ranges from -1 to 1
    - high value indicates that the data point is well matched to its own cluster and poorly matched to neighboring clusters
    - values near 0 indicate overlapping clusters
    - negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar
  - Find mean value of silhouette score of all data points => if most objects have a high value, then mean value is close to 1 and the clustering configuration is appropriate



Example: Drivers Dataset

- Includes 4000 drivers
- Each observation has 3 columns:
  - `Driver_ID`
  - `Distance_Feature`: mean distance covered per day
  - `Speeding_Feature`: mean percentage of time a driver was >5 mph over the speed limit

```
Driver_ID,Distance_Feature,Speeding_Feature
3423311935,71.24,28.0
3423313212,52.53,25.0
3423313724,64.54,27.0
```

- No target variable (no notion of groups, labels)
- Load dataset, drop Driver_ID column, scale features

```
dataset = pd.read_csv('fleet data.csv')
dataset = dataset.drop(columns=['Driver_ID'])
scaler = RobustScaler()
X = scaler.fit_transform(dataset)
```

- Source code for all clustering experimentations can be found here

K-Means

- Python implementation: `sklearn.cluster.Kmeans()` class
- Run algorithm to define groups (clusters)

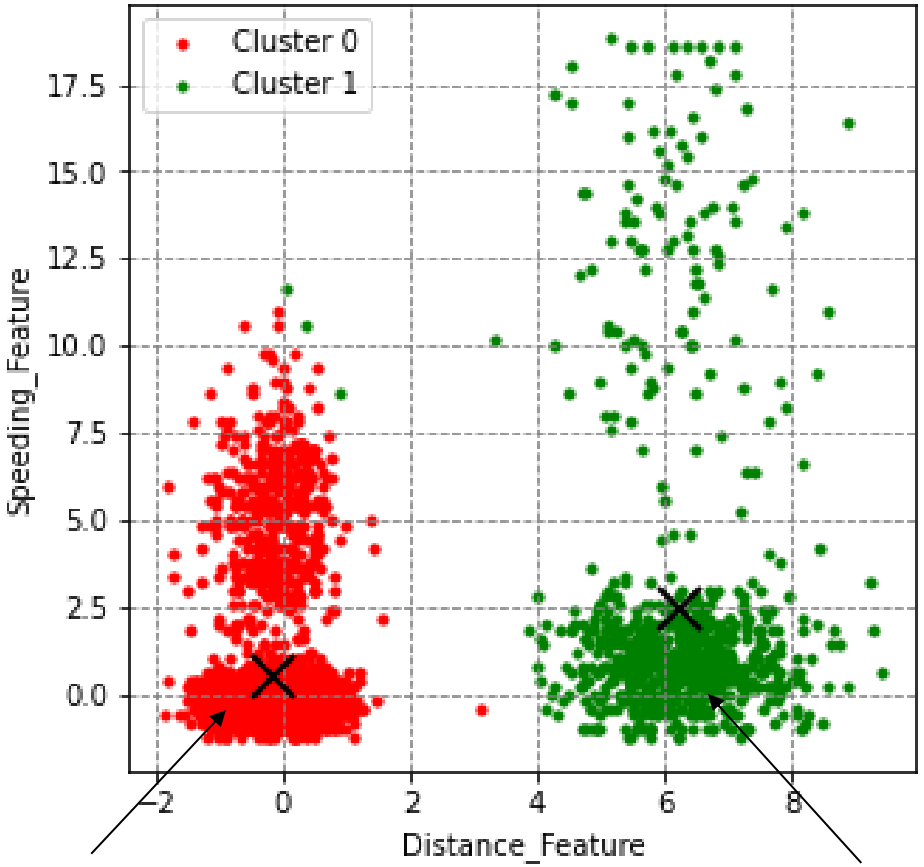
```
from sklearn.cluster import Kmeans
# create K-means object and run clustering on the input values (X)
# default k (n_clusters param) → 8
# default centroid initialization method (init param) → k-means++
kmeans = KMeans(n_clusters=2).fit(X)
print(kmeans.labels_) # labels of each sample
print(kmeans.centroids_)
```

- Assign new data samples to the most related cluster (closest centroid)

```
new_data = …
y_pred = kmeans.predict(new_data)
```

Visualizing results

- Run the *K*-means clustering algorithm for a range of *K* values
- Review the results

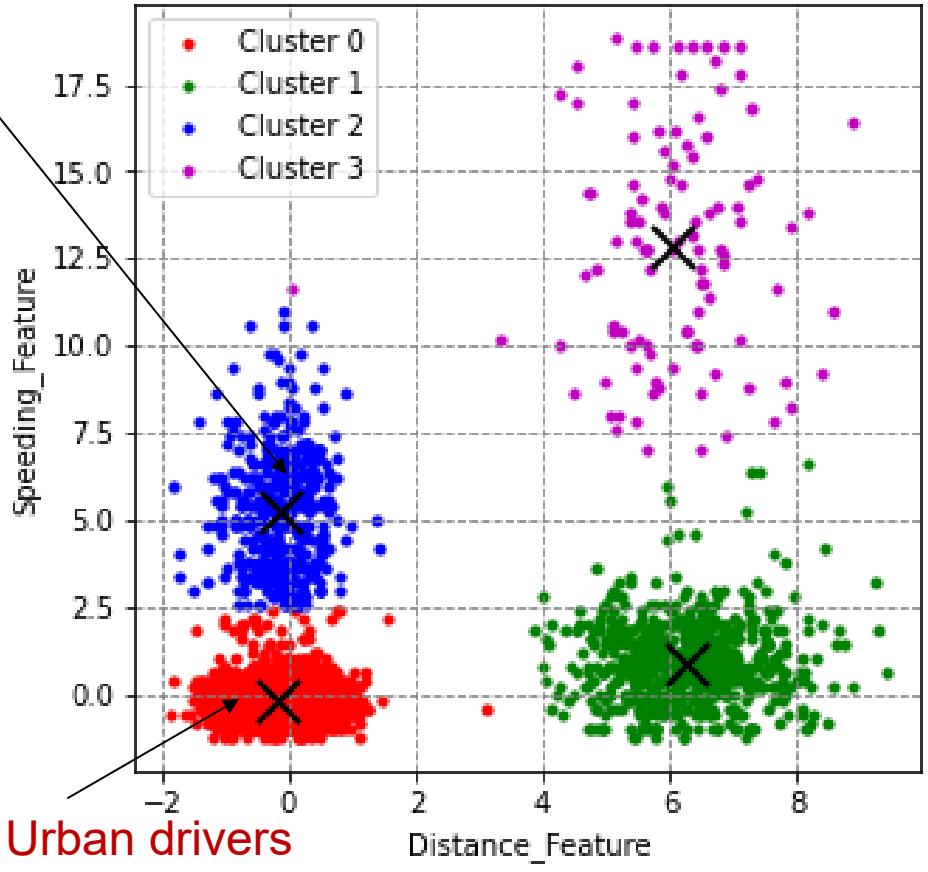


Urban drivers

Rural drivers

K=2

Urban drivers that are speeding frequently

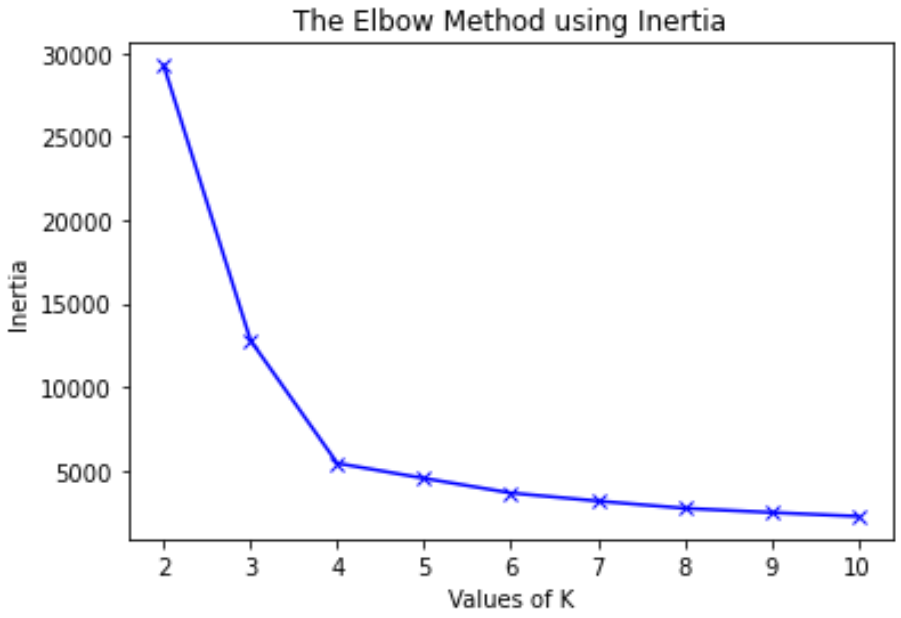


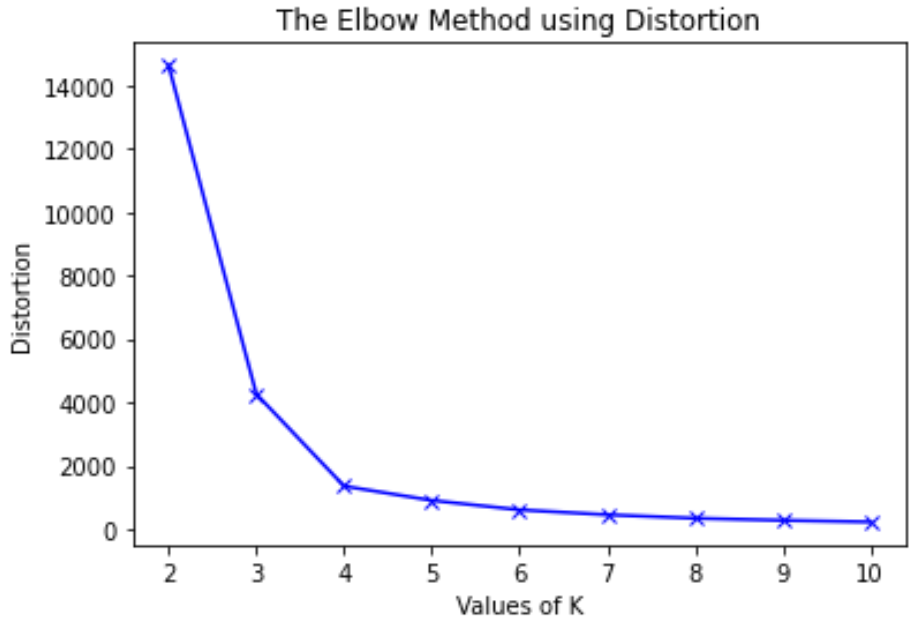
Urban drivers that follow speed limits

K=4

Elbow method

- We run the K-means algorithm for the values of k from 2 to 10 and plot the values of inertia and distortion for each iteration

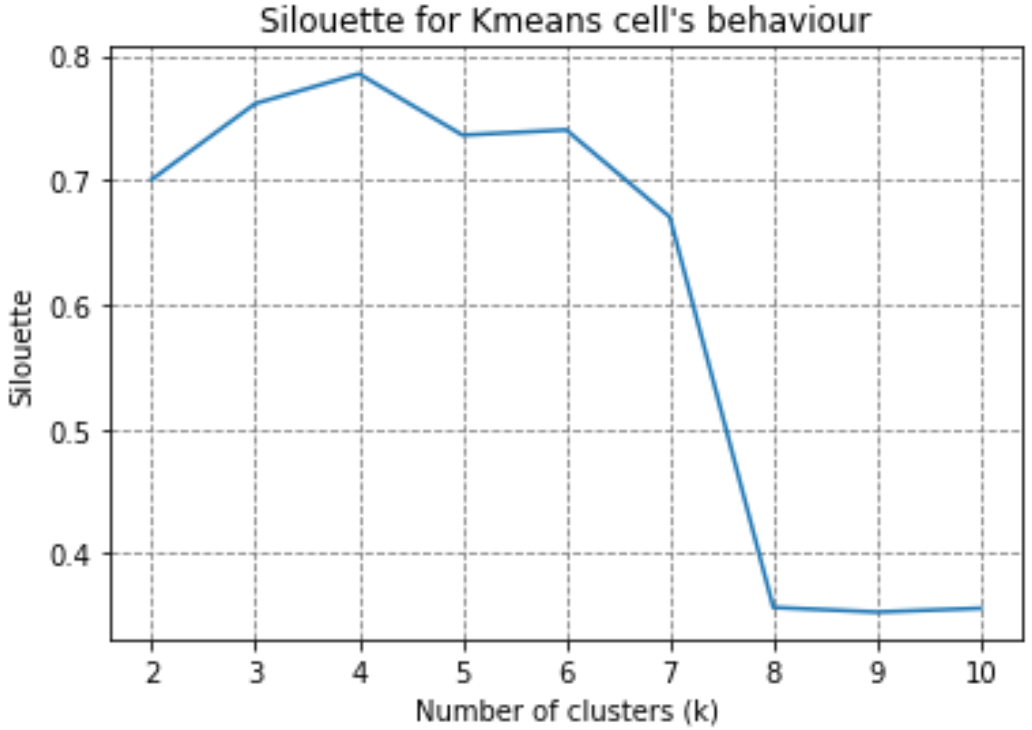




- It seems that the best number of clusters for grouping drivers is 4

Silhouette score

- We run the K-means algorithm for the values of k from 2 to 10 and plot the mean Silhouette score for each iteration



- Silhouette score confirms that the best number of clusters for grouping drivers is 4

Agglomerative clustering

- Perform agglomerative clustering and print labels

```
from sklearn.cluster import AgglomerativeClustering
# create Agglomerative object and run clustering on the input
values (X)
# default k (n_clusters param) → 2
agglomerative = AgglomerativeClustering (n_clusters=4).fit(X)
print(agglomerative.labels_) # labels of each sample
```

Clustering for handling missing values

- Clustering (and sometimes classification) can be used to impute missing values in both categorical and numerical features.
- This approach leverages the similarity between data points rather than blindly filling missing values with global statistics (mean, median, or mode).
- **Step 1: Cluster the Data**
  - Temporarily remove the column(s) with missing values that you want to impute.
  - Apply a clustering algorithm (e.g., K-Means) to the remaining data.
  - Use methods like the **Elbow Method** or **Silhouette Score** to determine the optimal number of clusters K.

Clustering for handling missing values

- **Step 2: Assign Missing Values Based on Cluster Membership**
  - Once clusters are formed, each row is assigned to a cluster.
  - For rows with missing values in **categorical columns**, impute using the **most frequent value** of that column among other rows in the same cluster.
  - For rows with missing values in **numerical columns**, impute using the **mean or median** of that column among other rows in the same cluster.
- **Benefits:**
  - Often yields more accurate imputations than simply using global statistics (mean, median, or mode on all values of the column).

Assignment

- Download Lab09 - Clustering Assignment-students-file-with-figures.ipynb from Piazza
- Read comments carefully and implement all requested tasks. You need to replace the None commands with the appropriate command(s).
- Run the file and save results within the same file
- Submit .ipynb file to Moodle by Tuesday 18th of November @ 23.59
  - Login to Moodle: https://moodle.cs.ucy.ac.cy/course/view.php?id=312 using your UCY credentials
  - Follow “Lab9 Submission” link
  - Upload your .ipynb file

APPENDIX

Linkage methods in Hierarchical clustering

- When we have more than one data point in a cluster, how do we calculate distance between these clusters? To calculate distance we can use any of following linkage methods:
  - Single linkage: considers the minimum distance between two points belonging to different clusters
  - Complete linkage: considers the maximum distance between two points belonging to different clusters
  - Average linkage: considers the average pairwise distance between all points belonging to different clusters
  - Centroid linkage: considers the distance between centroids of clusters
    - Centroid can be calculated as the mean of data points of the cluster
      - E.g. data points: (80, 56), (75, 53), (60, 50), (68,54) centroid: (70.75, 53.25)

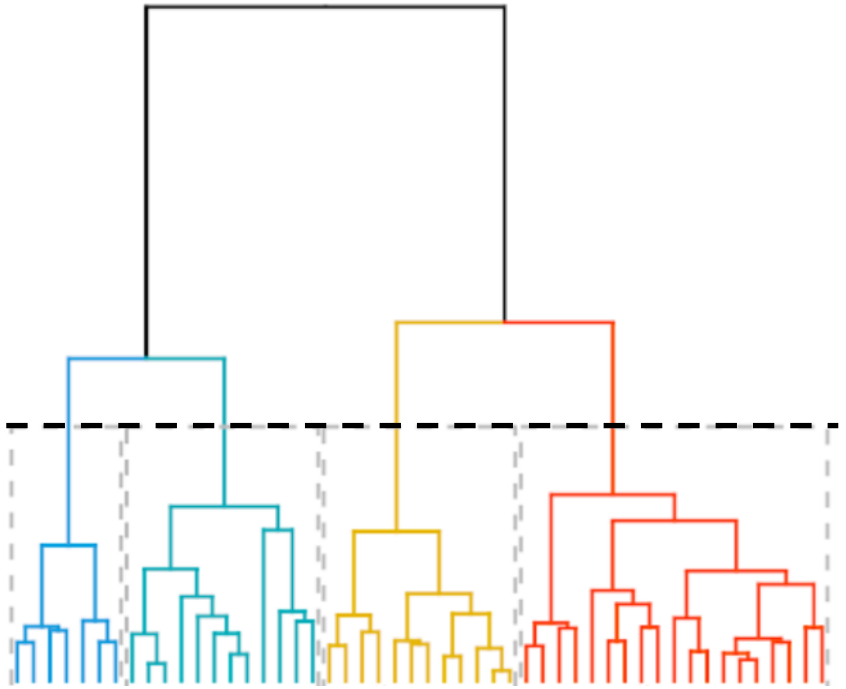
Bisecting K-Means (B-K-Means) vs K-Means

- Bisecting K-Means is a hybrid approach between Divisive Hierarchical Clustering (top down clustering) and K-means Clustering
  - Instead of partitioning the data set into K clusters in each iteration, Bisecting K-Means algorithm splits one cluster into two sub clusters at each bisecting step (by using K-Means) until K clusters are obtained.
- Bisecting K-Means is more efficient when K is large.
  - For the K-Means algorithm, the computation involves every data point of the data set and K centroids. On the other hand, in each Bisecting step of Bisecting K-Means, only the data points of one cluster and two centroids are involved in the computation. Thus, the computation time is reduced.
- Bisecting K-Means produce clusters of similar sizes, while K-Means is known to produce clusters of widely different sizes.

Bisecting K-Means

Dendrograms for finding best cluster number

- Dendrograms **cannot** always tell you how many clusters you should have
- However, if there is an obviously "correct" number of clusters this will often be evident in a dendrogram
- We can plot a dendrogram using this (SciPy) way or that (sklearn) way
  - Since vertical lines represent the distances between merged clusters, we can stop merging clusters when distance among them is long
  - For example, this horizontal cut seems to separate 4 disjoint clusters
- See more details here



Dendrograms for finding best cluster number

- Here is the dendrogram for the drivers dataset which reveals 3 or 4 disjoint clusters