MET CS 688
Clustering I

Leila Ghaedi – Fall 2024

Creator: cemagraphics | Credit: Getty Images/iStockphoto

Time Series Feature Extraction

- A time series dataset is a sequence of data points or observations gathered at either regular or varying time intervals. Typically, it involves a consecutive set of data points captured at uniform time gaps, which could occur hourly, daily, weekly, monthly, quarterly, or annually.
- Conventional machine learning algorithms might not be the most appropriate choice for analyzing unprocessed time series data since they struggle to grasp the strong interdependence between successive time points. Treating individual time points as features might not yield the best results in such cases.

Time Series Feature Extraction

- For time series data, feature extraction can be done using different time series analysis and decomposition techniques.
- Time Series Feature Extraction Library (TSFEL) is a python package for feature extraction on time series data.

Pip install tsfel

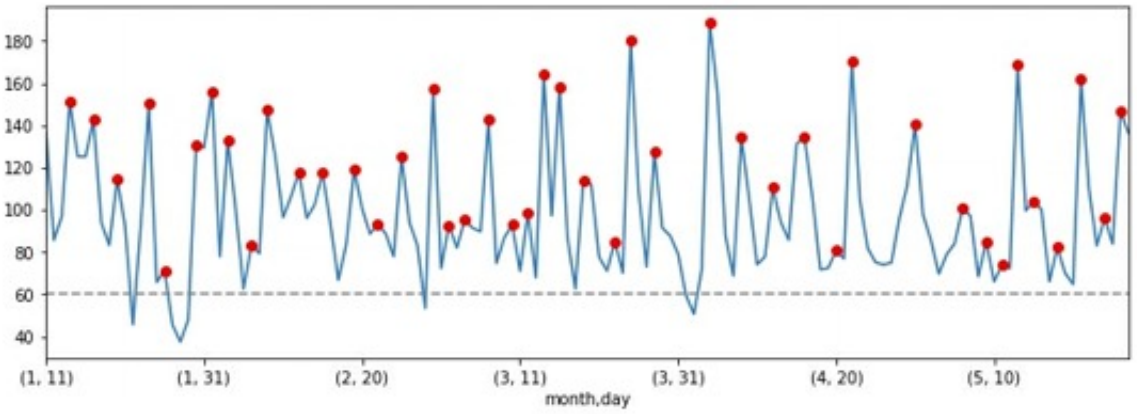
https://tsfel.readthedocs.io/en/latest/descriptions/get_started.html#overview

TSFEL List of Features

- https://tsfel.readthedocs.io/en/latest/descriptions/feature_list.html

Time Series Feature Extraction

1. Aggregation Methods such as maximum, minimum, average, ... That translates his historical data into one single data point.
2. Window Based Aggregation Methods: creates an aggregate measure over a window of time
3. Determining the number of local maxima and minima (Trend)



Time Series Feature Extraction

4- Fourier Transforms: Decomposition of the time series into its constituent frequencies. (Periodic, Seasonality)
5- Wavelet transforms: Analysis of time series data in both time and frequency domains.
6-Autoregression: Modelling the relationship between the time series and lagged versions of itself.
7- Seasonality decomposition: Separation of the time series into trend, seasonal, and residual components.

Cluster Analysis

- Clustering is the task of partitioning the dataset into groups, called clusters. The goal is to split up the data in such a way that points within a single cluster are very similar and points in different clusters are different. Similarly, to classification algorithms, clustering algorithms assign (or predict) a number to each data point, indicating which cluster a particular point belongs to.

K-Means Clustering

k-means clustering is one of the simplest and most commonly used clustering algorithms.

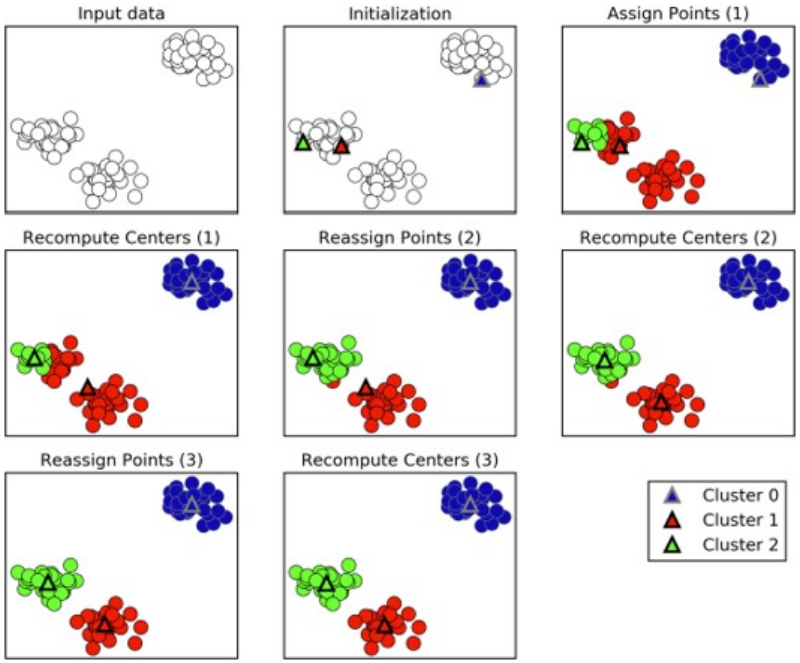
It tries to find cluster centers that are representative of certain regions of the data.

The algorithm alternates between two steps:

1. Assigning each data point to the closest cluster center
2. Setting each cluster center as the mean of the data points that are assigned to it.

The algorithm is finished when the assignment of instances to clusters no longer changes.

K-Means Clustering



K-Means Clustering

- Cluster centers are shown as triangles, while data points are shown as circles. Colors indicate cluster membership.
- We specified that we are looking for three clusters, so the algorithm was initialized by declaring three data points randomly as cluster centers (initialization)
- Then the iterative algorithm starts. First, each data point is assigned to the cluster center it is closest to (see “Assign Points (1)”). Next, the cluster centers are updated to be the mean of the assigned points (see “Recompute Centers (1)”). Then the process is repeated two more times. After the third iteration, the assignment of points to cluster centers remained unchanged, so the algorithm stops.

K-Means Steps

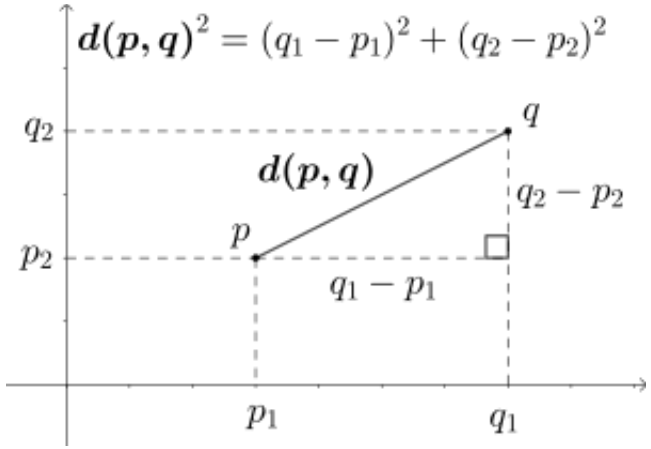
1. Initialization: Randomly choose K data points (centroids) as the initial cluster centers.
2. Assignment: Assign each data point to the nearest centroid, forming K clusters.
3. Update: Recalculate the centroids of the newly formed clusters.
4. Repeat: Repeat steps 2 and 3 until the centroids no longer change significantly or a maximum number of iterations is reached.

Distance Metrics in K-Means Clustering

- In K-means clustering, the distance between data points and cluster centroids is an important metric.
- The most commonly-used distance methods for K-means clustering are Euclidean distance and Manhattan distance,.

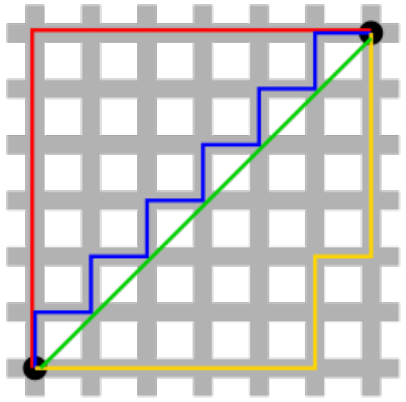
Euclidean Distance

- In mathematics, the Euclidean distance between two points in Euclidean space is the length of a line segment between the two points.



Manhattan Distance

- Manhattan distance is a distance measure that is calculated by taking the sum of distances between the x and y coordinates. The Manhattan distance is also known as Manhattan length. In other words, it is the distance between two points measured along axes at right angles.
- $d(A, B) = \sum |a_i - b_i|$



Distance Metrics

- Minkowski Distance
- Euclidean, Manhattan , and Minkowski distances are commonly used for computing the dissimilarity of objects described by numeric attributes.

Minkowski Distances

- L_p norm $d(A,B) = (\sum_{i=1}^{n} |a_i - b_i|^p)^{1/p}$
- L_1 norm $d(A,B) = \sum_{i=1}^{n} |a_i - b_i|^1$
- L_2 norm $d(A,B) = (\sum_{i=1}^{n} |a_i - b_i|^2)^{1/2}$

Minkowski Distances

Minkowski distance is a generalization of the Euclidean and Manhattan distances.

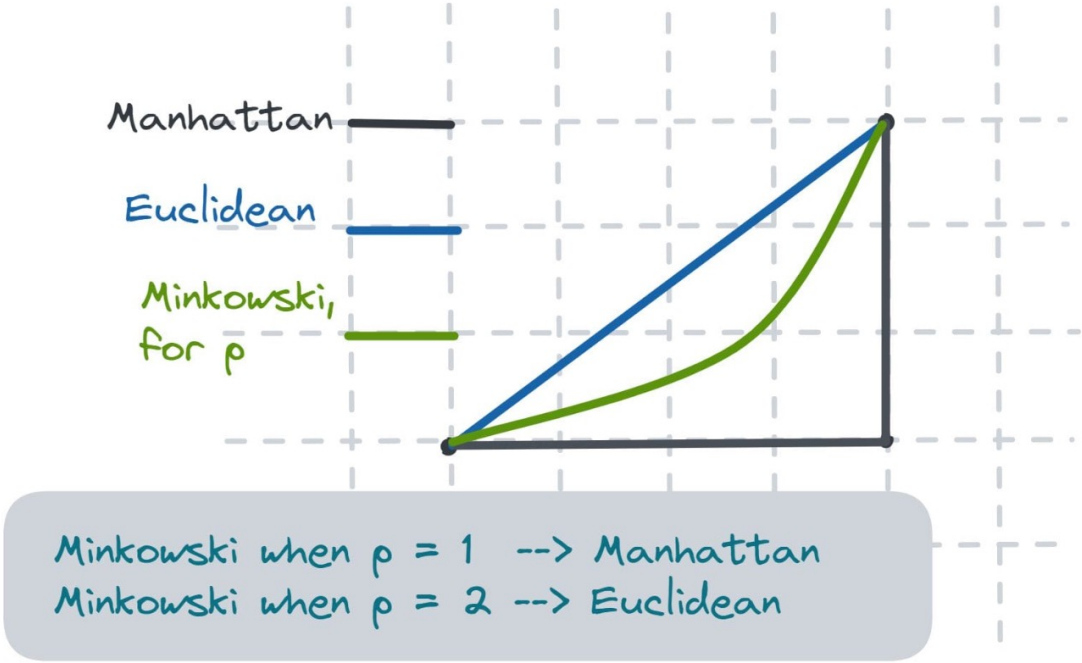
Euclidian

Lp2 norm, a type of Minkowski distance. Useful for sparse and high dimensional data.

Manhattan

Lp1 norm, a type of Minkowski distance. Useful for sparse and high dimensional data. Handles outlier better than Eculidean distance.

Minkowski Distances

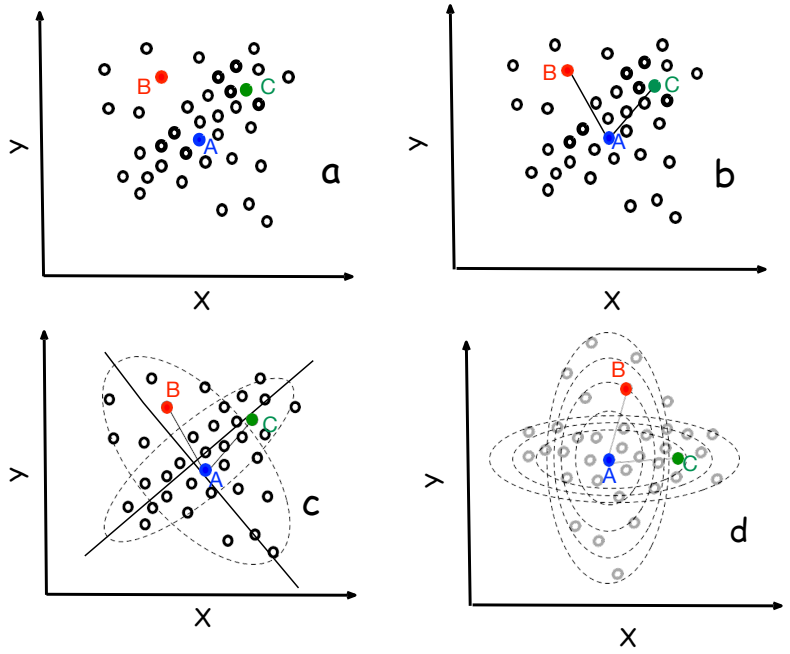


Mahalanobis Distance

- The Mahalanobis distance is a measure of the distance between a point and a distribution.
- It is a multi-dimensional generalization of the idea of measuring how many standard deviations away a point is from the mean of a distribution.
- Unlike the Euclidean distance, which assumes the data is isotopically distributed and all dimensions are equally important, the Mahalanobis distance considers the covariance between different dimensions, thus taking into account the correlations in the data.
- Mahalanobis distance is applied in clustering algorithms to measure the distance between data points and centroids. It helps in clustering data points more accurately by considering the data's covariance structure.

Mahalanobis Distance

- Sometimes distribution of the data affects the similarity measurement.



Mahalanobis Distance

$$d_M(X_1, X_2) = \sqrt{(X_1 - X_2)' M (X_1 - X_2)} = \sqrt{\sum_{i,j=1}^{p} (X_{1,i} - X_{2,i}) M(i, j) (X_{1,j} - X_{2,j})},$$

M is a p×p symmetric positive semidefinite matrix.
Compared with Euclidean distance, Mahalanobis distance:

1. Assigns different weights (through the diagonal elements of matrix M) to different dimensions.
2. Incorporates the interaction effect of different dimensions (through the off-diagonal elements of matrix M) in measuring the distance between the two input tuples.

Depending on the specific choice of the M matrix, the Mahalanobis distance between two data tuples will vary.

Mahalanobis distance is similar to Euclidean distance except that it normalizes the data on the basis of the inter- attribute correlations.

Hamming Distance

- The Hamming distance between two strings of equal length is the number of positions at which the corresponding characters are different.
- It is a measure of the minimum number of changes required to turn one string into another.

Hamming Distance

In [1]:
```
import nltk
# Compute the Hamming distance between two strings
hamming_distance = nltk.edit_distance('hello', 'world')

print( hamming_distance)
```

```
4
```

In [2]:
```
# Compute the Hamming distance between two strings
hamming_distance = nltk.edit_distance('001001001', '110001001')

print(hamming_distance)
```

```
3
```

Levenstein (Edit) Distance

- The Levenshtein distance, also known as the edit distance, is a metric used to quantify the difference between two sequences of strings.
- It measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into another.
- It is commonly used in various fields, including computational linguistics, bioinformatics, and data mining, for tasks such as spell checking, plagiarism detection, and similarity analysis.

Levenstein Distance

In [3]:
```
from Levenshtein import distance
# Compute the Levenshtein distance between two strings
str1 = "HONDA"
str2 = "HYUNDAI"
l_dist = distance(str1, str2)
print(l_dist)
```

```
3
```

Longest Common Subsequences Distance (LCS)

- The Longest Common Subsequence (LCS) distance between two strings/sequences x and y is the minimum cost of operations (insertions and deletions) required to transform x into y.
- The LCS similarity is more commonly used, which can be interpreted as the length of the longest subsequence common to x and y.

Example: String1=AGGTAB

String2=GXTXAYB

LCS=GTAB

Cosine Similarity

- Cosine similarity measures the similarity between two vectors of an inner product space.
- It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction.
- It is often used to measure document similarity in text analysis.
- Let x and y be two vectors

$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}$$

Cosine Similarity

- Let x and y be two vectors
- ||x|| is the Euclidean nor of the vector x
- x=(5,0,3,0,2,0,0,2,0,0) and y=(3,0,2,0,1,1,0,1,0,1)
- x.y=5*3+0*0+3*2+…=25
- ||x||= (5^2+0^2+3^2+…)^0.5= 6.48
- ||y||= (3^2+0^2+2^2+…)^0.5= 4.12
- sim(x,y)= 0.94
- x and y can be the frequency of a series of words in two strings.

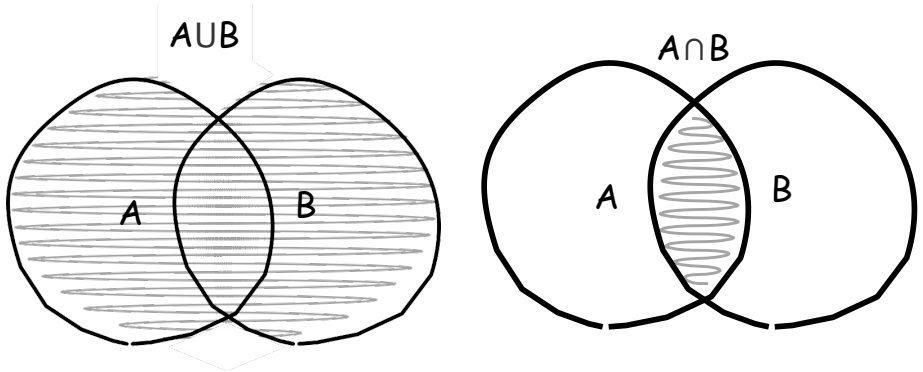
$$sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{||\boldsymbol{x}||||\boldsymbol{y}||}$$

A use case for Cosine Similarity

Document	Team	Coach	Hockey	Baseball	Soccer	Penalty	Score	Win	Loss	Season
Document1	5	0	3	0	2	0	0	2	0	0
Document2	3	0	2	0	1	1	0	1	0	1
Document3	0	7	0	2	1	0	0	3	0	0
Document4	0	1	0	0	1	2	2	0	3	0

Jaccard Similarity

$$d(\mathrm{A},\mathrm{B}) = \frac{A \cap B}{A \cup B}$$



Jaccard index is useful when we are dealing with **non-numerical** data or **binary** objects that their semantical similarity is important.

Jaccard Similarity

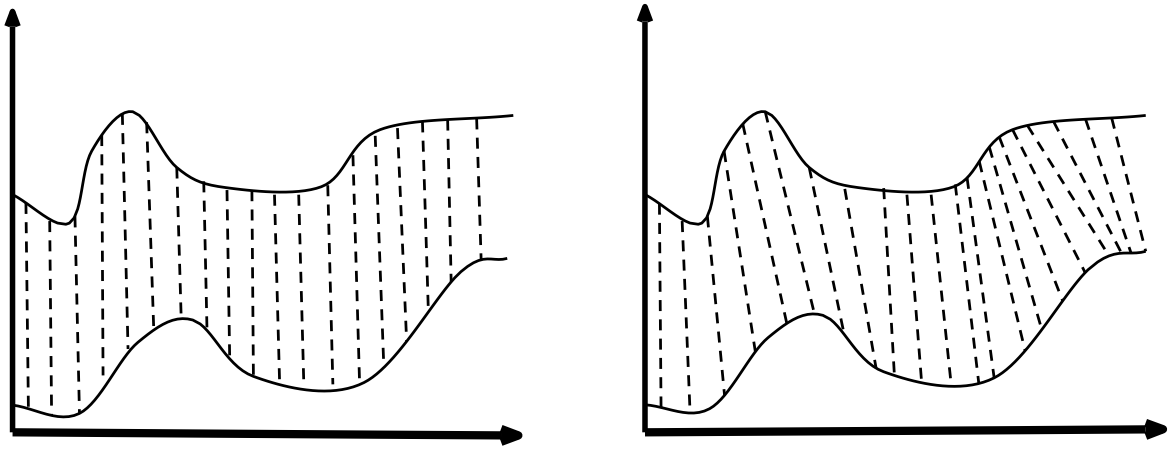
In [4]:
```
def jaccard_similarity(list1, list2):
    s1 = set(list1)
    s2 = set(list2)
    return float(len(s1.intersection(s2)) / len(s1.union(s2)))
list1 = ['dog', 'cat', 'cat', 'rat']
list2 = ['dog', 'cat', 'mouse']
jaccard_similarity(list1, list2)
```

Out[4]: 0.5

Dynamic Time Wrapping (DTW)

- Dynamic time warping (DTW) is a method used to measure the similarity between two temporal sequences, which may vary in speed. DTW allows for the comparison of sequences that may have variations in the timing or speed of their underlying processes.
- DTW is used in time series analysis and signal processing to align sequences and identify similarities between them.
- For example, similarities in walking could be detected using DTW, even if one person was walking faster than the other, or if they were changing the speed during an observation.
- DTW is one of the most effective metrics to measure distances among time series.

Why DTW?



Distance comparison between two time series. (left) Euclidean distance and (right) DTW distance.

DTW

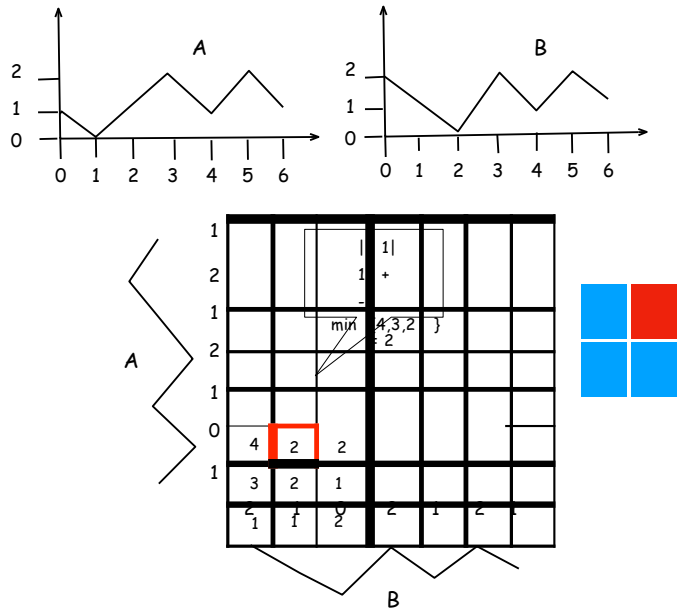
DTW is a method that calculates an optimal match between two given sequences:

- Every index from the first sequence must be matched with one or more indices from the other sequence, and vice versa
- The first index from the first sequence must be matched with the first index from the other sequence (but it does not have to be its only match)
- The last index from the first sequence must be matched with the last index from the other sequence (but it does not have to be its only match)
- The mapping of the indices from the first sequence to indices from the other sequence must be monotonically increasing, and vice versa
- The optimal match is the match that satisfies all the restrictions and the rules and that has the minimal cost, where the cost is computed as the sum of absolute differences, for each matched pair of indices, between their values.

Dynamic Time Warping Process

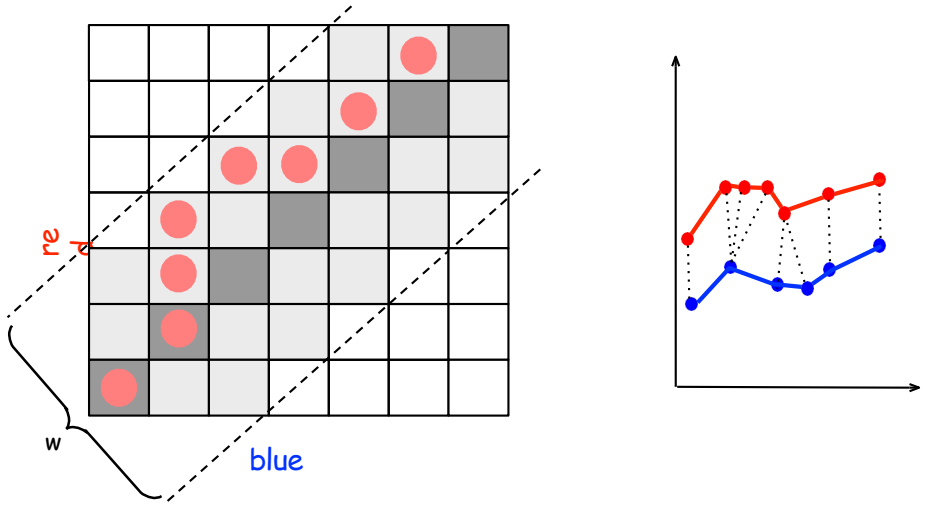
(1) Cumulative Matrix creation

$$DTW(\overline{A},\overline{B}) = \left|A_i - B_j\right| + min(DTW(i-1,j), DTW(i,j-1), DTW(i-1,j-1))$$



Dynamic Time Warping Process

(2)Identify the warping path from matrix diagonal



Warping path based on the warping constraint of w=2 for a two time series shown on the right. The main diagonal is shown in dark grey and the area that fits in w=2 is light grey.

w = 2

maximum amount of deviation

Distance Metrics Notes

- While converting interval, ordinal, or categorical variables into a number for a clustering algorithm, remember not to harm the relation between data objects and keep them after the conversion.
- Euclidian distance, Manhattan distance and in general L_p-norm distances is not good for sparse and high dimensional dataset, because usually those datasets are sparse and contain too many features, noise and outliers. Nevertheless, to handle outliers Manhattan distance is better than Euclidian.
- To measure similarities between ordinal data objects, usually we convert them into range of values. It is recommended to map the ordinal data between 0 and 1.

Distance Metrics Notes

- A real-world dataset is usually multivariate and thus it includes different data with different types including numerical, binary, categorical (nominal), etc. We can focus on one attribute/feature/field for clustering. Nevertheless, it is not useful to resolve a problem only for one attribute. A better method is to convert all data objects into a single data types, such as numerical values between 0 and 1 and then apply the clustering. However, a conversion to another format is associated with smoothing the data, and smoothing can negatively affects the precision of the algorithm. Furthermore, we should be able to select specific number of columns from the dataset, which is called feature selection.
- More important than the selection of clustering algorithm is the selection of similarity measurement method.
- Cosine -> Text, Jaccard —> Non-numerical and Binary, DTW—> Time series
- DTW can not measure the distance between two time series with different length, you can see this by mistake in many textbooks. It can tolerate a small size differences but not a large ones.

Clustering Methods

- Partitioning Methods (K-Means, K-Medoids)
- Hierarchical methods
- Density-based and grid-based methods

K-Medoids Clustering

- Partitioning Around Medoids (PAM)
- The k-means algorithm is sensitive to outliers because such objects are far away from the majority of the data, and thus, when assigned to a cluster, they can dramatically distort the mean value of the cluster.
- Instead of taking the mean value of the objects in a cluster as a reference point, we can pick actual objects to represent the clusters, using one representative object per cluster. Each remaining object is assigned to the cluster of which the representative object is the most similar.

K-Medoids Clustering

- The partitioning method is then performed based on the principle of minimizing the sum of the dissimilarities between each object p and its corresponding representative object. That is, an absolute-error criterion is used, defined as:

$$E = \sum_{i=1}^{k} \sum_{\boldsymbol{p} \in C_i} dist(\boldsymbol{p}, \boldsymbol{o}_i),$$

K-Means vs K-Medoids

K-Mean	K-Medoids
K-means takes the mean of data points to create new points called centroids.	K-medoids uses points from the data to serve as points called medoids.
Centroids are new points previously not found in the data.	Medoids are existing points from the data.
Partitioning Method	Partitioning Method
Requires to have the number of clusters as input parameter.	Requires to have the number of clusters as input parameter.
Sensitive to noise and outliers.	More robust to noise and outliers.
To calculate the centroid from the cluster, sum up the position of all points of a single cluster, and divide by the number of points.	The medoid of a cluster is defined as the object in the cluster whose average dissimilarity to all the objects in the cluster is minimal, that is, it is a most centrally located point in the cluster.

K-Modes: Clustering nominal data

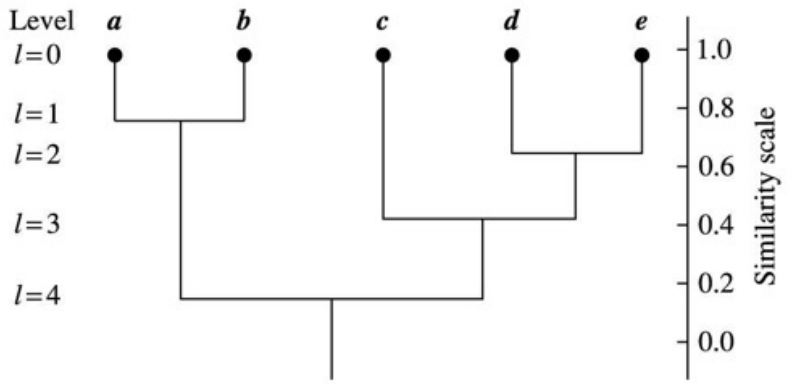
- One limitation of the k means method is that it can be applied only when the mean of a set of objects is defined.
- This may not be the case in some applications such as when data with nominal attributes is involved.
- The k modes method is a variant of k means, which extends the k means paradigm to cluster nominal data by replacing the means of clusters with modes.

Hierarchical Clustering Methods

- To divide our data into various tiers or levels of groups, a hierarchical clustering technique operates by organizing data elements into a hierarchical structure or a "tree" of clusters.
- Representing data objects in the form of a hierarchy is useful for data summarization and visualization.
- We may believe that the data bear an underlying hierarchical structure that we want to discover. For example, in the study of biological evolution, hierarchical clustering may group living creatures according to their biological features to uncover evolutionary paths, which are a hierarchy of species.

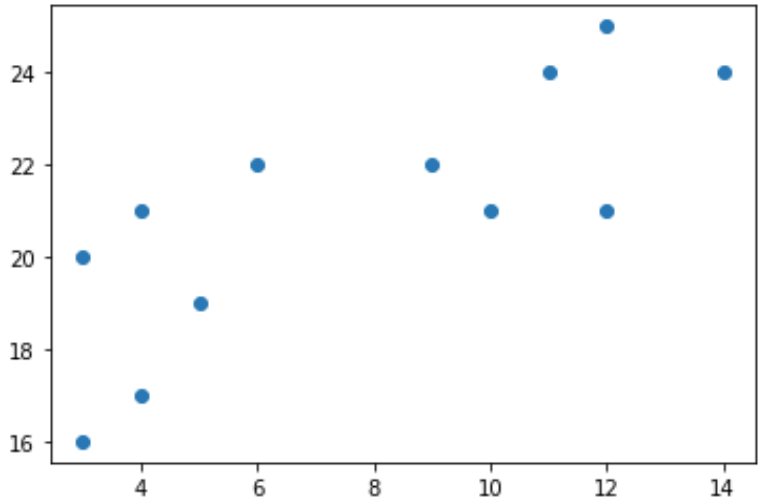
Dendrogram

- A tree structure called a dendrogram is commonly used to represent the process of hierarchical clustering. It shows how objects are grouped together (in an agglomerative method) or partitioned (in a divisive method) step-by-step.



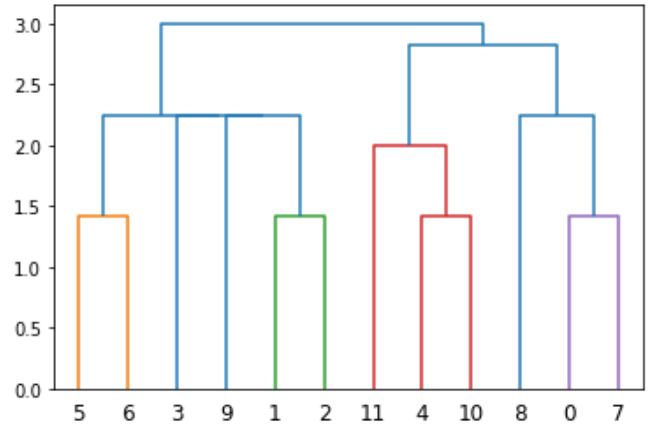
Dendrogram Example

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
x = [11, 3,  4,  5,  9,  4,  3, 12, 14 , 6, 10, 12]
y = [24, 20, 21, 19, 22, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()
```



Dendrogram Example

In [2]:
```
data = list(zip(x, y))
# Now we compute the single linkage using euclidean distance,
# and visualize it using a dendrogram:
linkage_data = linkage(data, method='single', metric='euclidean')
dendrogram(linkage_data)

plt.show()
```



Hierarchical Clustering Methods

- Agglomerative: hierarchical decomposition is formed in a bottom-up (merging) fashion.
- Divisive: hierarchical decomposition is formed in a top-down (splitting) fashion.

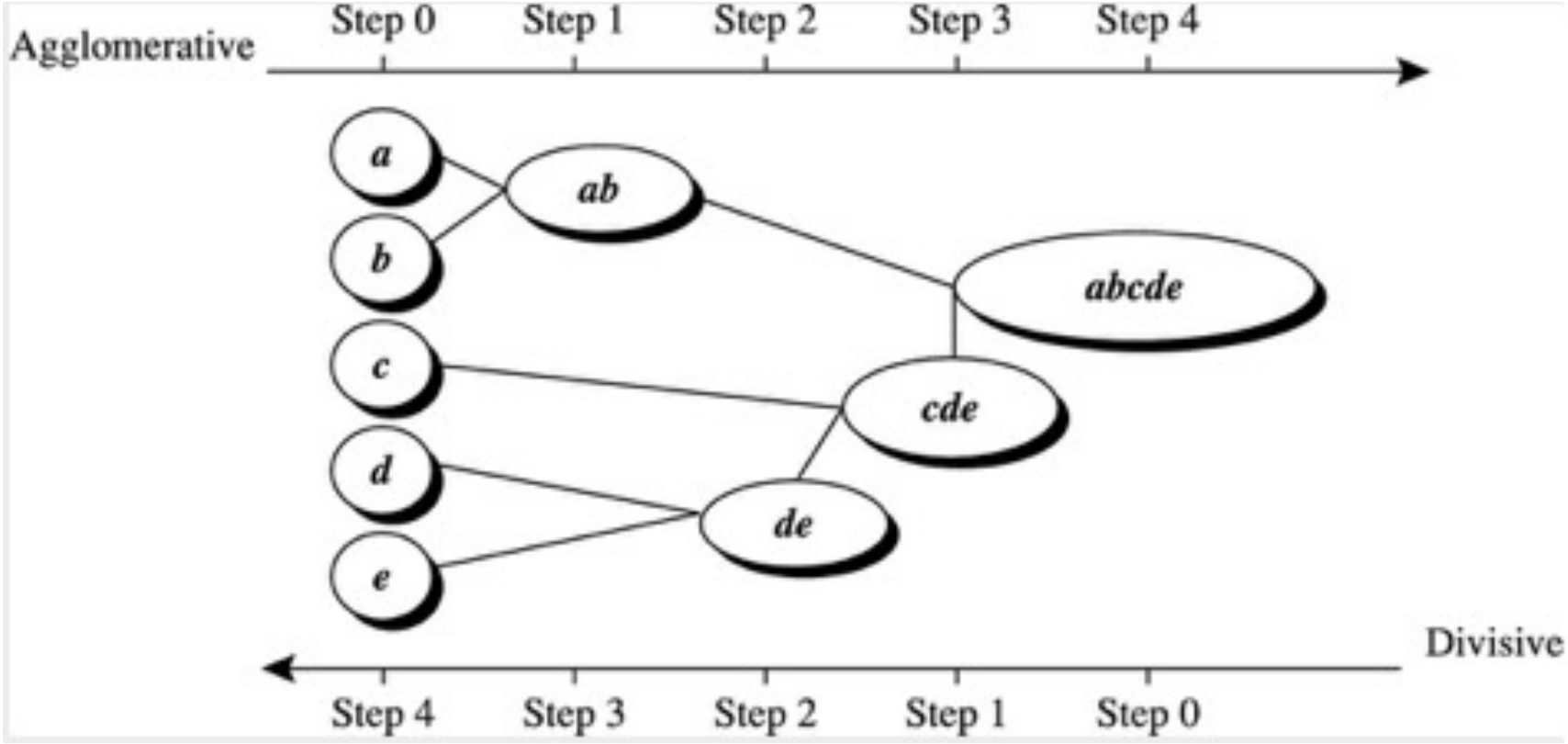
Agglomerative Hierarchical Clustering

- An agglomerative hierarchical clustering method uses a bottom-up strategy.
- At beginning, each object form its own cluster and iteratively merges clusters into larger and larger clusters, until all the objects are in a single cluster or certain termination conditions are satisfied.
- The single cluster becomes the hierarchy's root.
- For the merging step, it finds the two clusters that are closest to each other (according to some similarity measure) and combines the two to form one cluster. Because two clusters are merged per iteration, where each cluster contains at least one object, an agglomerative method requires at most n iterations.

Divisive Hierarchical Clustering

- A divisive hierarchical clustering method uses a top-down strategy.
- It starts by placing all objects in one cluster, which is the hierarchy's root.
- It then divides the root cluster into several smaller subclusters and recursively partitions those clusters into smaller ones.
- The partitioning process continues until each cluster at the lowest level is coherent enough—either containing only one object, or the objects within a cluster are sufficiently similar to each other.
- In either agglomerative or divisive hierarchical clustering, a user can specify the desired number of clusters as a termination condition.

Example: Clustering 5 objects



Similarity Measures in Hierarchical Clustering

- Four widely used measures for distance between clusters are as follows, where |p−p′| is the distance between two objects or points, p and p′ ; mi is the mean for cluster Ci ; and ni is the number of objects in Ci . They are also known as linkage measures.
- Minimum Distance $$dist_{min}(C_i, C_j) = \min_{p \in C_i, p' \in C_j} \{\|p - p'\|\}$$
- Maximum Distance $$dist_{max}(C_i, C_j) = \max_{p \in C_i, p' \in C_j} \{\|p - p'\|\}$$
- Mean Distance $$dist_{mean}(C_i, C_j) = \|m_i - m_j\|$$
- Average Distance $$dist_{avg}(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p \in C_i, p' \in C_j} \|p - p'\|$$

Single Linkage vs Complete Linkage

- Single linkage: computes the minimum distance between clusters before merging them.
- Complete linkage: computes the maximum distance between clusters before merging them.
- Single linkage and complete linkage are sensitive to noise and outlier.
- The use of mean or average distance is a compromise between the minimum and maximum distances and overcomes the outlier sensitivity problem.
- https://www.datacamp.com/tutorial/introduction-hierarchical-clustering-python

Density-based and grid-based methods

- Most of the partitioning and hierarchical methods are designed to find spherical-shaped clusters. They have difficulty finding clusters of arbitrary shape such as the “S” shape and oval clusters.
- Although some feature transformation methods, such as kernel k-means, may help, it is often tricky to choose appropriate kernel functions.

Density-based and grid-based methods

- To find clusters of arbitrary shape, we can model clusters as dense regions in the data space, separated by sparse regions.
- This is the main strategy behind density-based clustering methods , which can discover clusters of nonspherical shape.
- DBSCAN is one density-based clustering model that we review here.

DBSCAN
(Density-Based Spatial Clustering of Applications with Noise)

- How can we find dense regions in density-based clustering?
  - The density of an object o can be measured by the number of objects close to o .
- DBSCAN finds core objects , that is, objects that have dense neighborhoods.
- It connects core objects and their neighborhoods to form dense regions as clusters.

DBSCAN

- How does DBSCAN quantify the neighborhood of an object?
  - DBSCAN employs a user-specified parameter $\epsilon>0$ to specify the radius of a neighborhood that is considered for every object. The ϵ neighborhood of an object o is the space within a radius ϵ centered at o .
- Due to the fixed neighborhood size parameterized by ϵ , the density of a neighborhood can be measured simply by the number of objects in the neighborhood.
- To determine whether a neighborhood is dense or not, DBSCAN uses another user-specified parameter, MinPts , which specifies the density threshold of dense regions.
- An object is a core object if the ϵ neighborhood of the object contains at least MinPts objects, otherwise, it is noise . Core objects are the pillars of dense regions.

DBSACN Example

- https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
import warnings
warnings.filterwarnings("ignore", "is_categorical_dtype" )
```

In [2]:
```
df = pd.read_csv('Mall_Customers.csv')
df_n = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]
df.head()
```

Out[2]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

DBSACN Example

In [3]:
```
# We will fit df_n on the DBSCAN algorithm with
# eps 12.5 and min_sample 4  (8,7). Then, we will
# create a DBSCAN_dataset from df_n and the
# clusters for each data point
db = DBSCAN(eps=12.5, min_samples=4).fit(df_n)
DBSCAN_dataset = df_n.copy()
DBSCAN_dataset.loc[:,'cluster'] = db.labels_
DBSCAN_dataset.head()
```

Out[3]:

	Age	Annual Income (k$)	Spending Score (1-100)	cluster
0	19	15	39	0
1	21	15	81	0
2	20	16	6	-1
3	23	16	77	0
4	31	17	40	0

DBSACN Example

In [4]:
```
DBSCAN_dataset.cluster.value_counts().to_frame()
```

Out[4]:

cluster	count
0	112
2	34
3	24
-1	18
1	8
4	4

In [5]:
```
no_clusters = len(np.unique(db.labels_) )
no_clusters
```

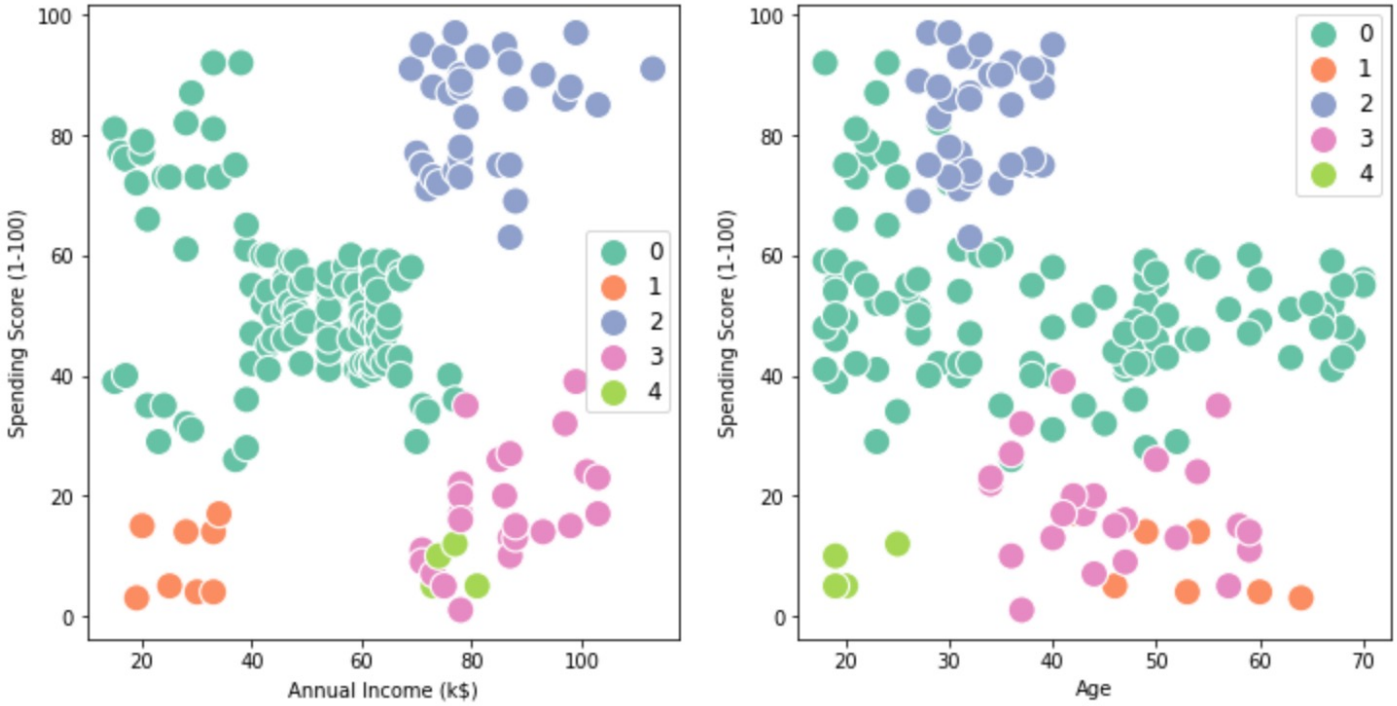
Out[5]: 6

DBSACN Example

In [6]:

```
fig2, (axes) = plt.subplots(1,2,figsize=(12,6))

sns.scatterplot(data=DBSCAN_dataset[DBSCAN_dataset['cluster']!=-1],
                x='Annual Income (k$)', y='Spending Score (1-100)',
                hue='cluster', ax=axes[0], palette='Set2', legend='full', s=200)

sns.scatterplot(data=DBSCAN_dataset[DBSCAN_dataset['cluster']!=-1],x='Age',
                y='Spending Score (1-100)',
                hue='cluster', palette='Set2', ax=axes[1], legend='full', s=200)

#axes[0].scatter(outliers['Annual Income (k$)'],
#outliers['Spending Score (1-100)'], s=10, label='outliers', c="k")

#axes[1].scatter(outliers['Age'], outliers['Spending Score (1-100)'],
#s=10, label='outliers', c="k")
axes[0].legend()
axes[1].legend()

plt.setp(axes[0].get_legend().get_texts(), fontsize='12')
plt.setp(axes[1].get_legend().get_texts(), fontsize='12')

plt.show()
```

DBSACN Example



Clustering Evaluation Methods

Clustering Evaluation Methods

- What settings would yield the best results for cluster parameters? Which clustering algorithm demonstrates the highest accuracy when applied to our dataset? Additionally, which algorithm proves to be more efficient in terms of resources?
- Various quantitative (objective, algorithm-generated) and qualitative (subjective, involving human annotations) methods are employed to evaluate clustering results. However, there is no optimum method, and each of these evaluation approaches has its own limitations.
- Methods for evaluation include:
    - Intrinsic Evaluation
    - Extrinsic Evaluation

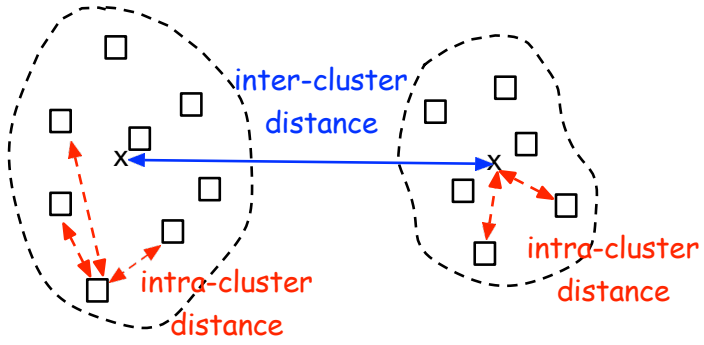
Intrinsic vs Extrinsic Evaluation Methods

- If ground truth is available, it can be used by the extrinsic methods , which compare the clustering against the ground truth and measure.
- If the ground truth is unavailable, we can use the intrinsic methods , which evaluate the goodness of a clustering by considering how well the clusters are separated.
- Ground truth can be considered as supervision in the form of “cluster labels.” Hence, extrinsic methods are also known as supervised methods , whereas intrinsic methods are unsupervised methods .

Intrinsic Methods

- The intrinsic methods are based on the fundamental intuition in clustering analysis, that is, examining how compact clusters are and how well clusters are separated.
- Many intrinsic methods take the advantage of a similarity or distance measure between objects in the data set.

  - Elbow Method
  - Silhouette Index
  - Dunn Index
  - Davies-Bouldin Index

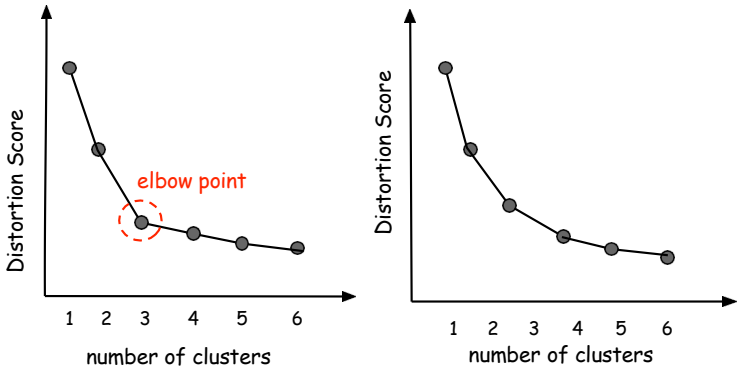


Elbow Method

- The elbow method is based on the observation that increasing the number of clusters can help to reduce the sum of within-cluster variance of each cluster.
- This is because having more clusters allows one to capture finer groups of data objects that are more similar to each other.
- However, the marginal effect of reducing the sum of within-cluster variances may drop if too many clusters are formed, because splitting a cohesive cluster into two gives only a small reduction.
- Consequently, a heuristic for selecting the right number of clusters is to use the turning point in the curve of the sum of within-cluster variances with respect to the number of clusters.

Elbow Method

The elbow method is used to identify optimal number of clusters.
It is mostly used for partitioning methods, e.g. *k*-means. Elbow method presents the relation between a distortion score and the number of clusters.
The point that the elbow is constructed identifies the optimal number of clusters.
Distortion score is computed, the sum of square distances from each point to its assigned center.



Elbow method example, (Left) there is an elbow and can conclude three is the number of optimal clusters. (Right) There is no elbow and thus we can not use elbow method to determine the optimal number of clusters.

Elbow Method Example

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets
```

In [2]:
```
iris = datasets.load_iris()
#we are usingh
df=pd.DataFrame(iris['data'])
df.head()
```

Out[2]:

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Elbow Method Example

In [3]:
```
mean = df.mean(axis=0)
std = df.std(axis=0)
# Calculate the z-score for each column in the dataframe
df_zscore = (df - mean) / std
print(df_zscore)
```

```
            0         1         2         3
0   -0.897674  1.015602 -1.335752 -1.311052
1   -1.139200 -0.131539 -1.335752 -1.311052
2   -1.380727  0.327318 -1.392399 -1.311052
3   -1.501490  0.097889 -1.279104 -1.311052
4   -1.018437  1.245030 -1.335752 -1.311052
..        ...       ...       ...       ...
145  1.034539 -0.131539  0.816859  1.443994
146  0.551486 -1.278680  0.703564  0.919223
147  0.793012 -0.131539  0.816859  1.050416
148  0.430722  0.786174  0.930154  1.443994
149  0.068433 -0.131539  0.760211  0.788031

[150 rows x 4 columns]
```

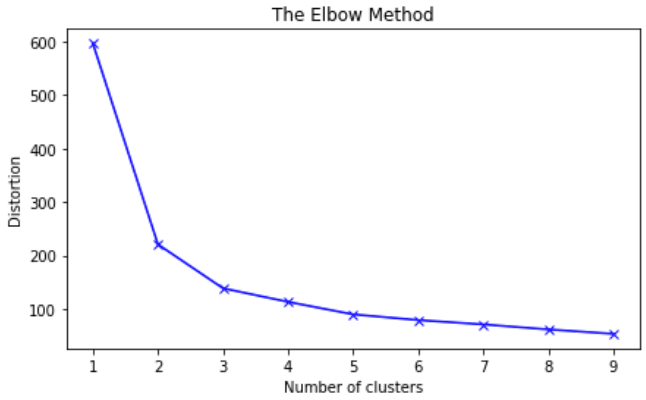
In [4]:
```
iris['target']
```

Out[4]:
```
array([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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Elbow Method Example

In [5]:
```
distortions = []
# number of clusters 1 to 10
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k ,n_init=10)
    kmeanModel.fit(df_zscore)
    distortions.append(kmeanModel.inertia_)
```

In [6]:
```
plt.figure(figsize=(7,4))
plt.plot(K, distortions, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.title('The Elbow Method')
plt.show()
```



Elbow Method Example

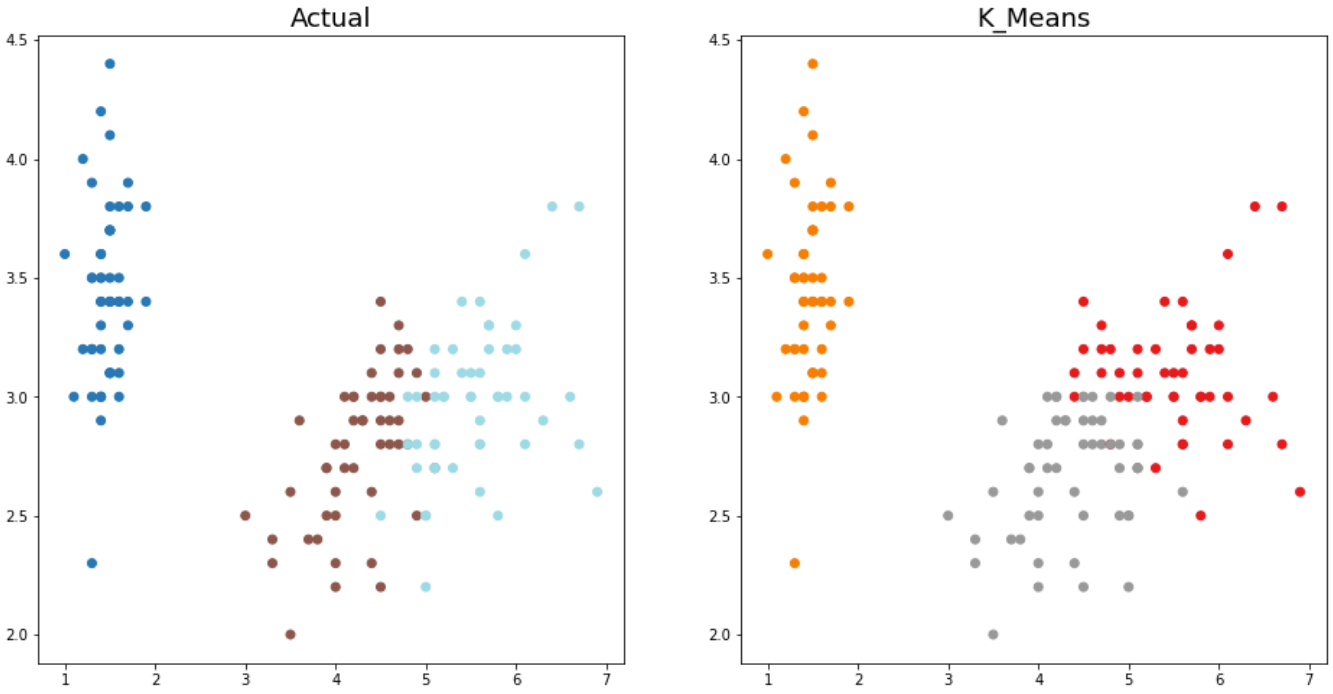
In [7]:
```
kmeanModel = KMeans(n_clusters=3 ,n_init=10)
kmeanModel.fit(df_zscore)
```

Out[7]:
```
KMeans
KMeans(n_clusters=3, n_init=10)
```

In [8]:
```
df['k_means']=kmeanModel.predict(df_zscore)
df['target']=iris['target']
fig, axes = plt.subplots(1, 2, figsize=(16,8))

axes[0].scatter(df[2], df[1], c=df['target'], cmap=plt.cm.tab20)
axes[1].scatter(df[2], df[1], c=df['k_means'], cmap=plt.cm.Set1)
axes[0].set_title('Actual', fontsize=18)
axes[1].set_title('K_Means', fontsize=18)
```

Out[8]:
```
Text(0.5, 1.0, 'K_Means')
```

Elbow Method Example



Silhouette Index

Silhouette coefficient is a score range from -1 to +1. It measures how similar is a member of a cluster is to its own cluster, i.e. cohesion, compared to other clusters, i.e. separation.

Silhouette coefficient for data object *i* is calculated based on parameters concepts, *a(i)* and *b(i).*
a(i) is the average distance between member *i* and other members of the cluster (the cluster which *i* belongs to it).
b(i) is the minimum average distance of *i* to all points in any other clusters (here we mean all clusters that *i* is not their member).

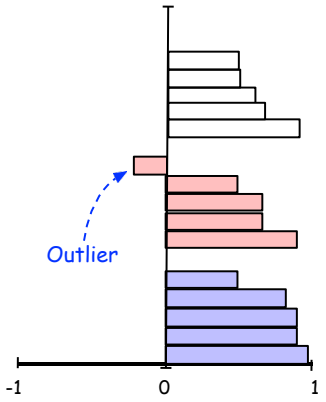
Silhouette coefficient is presented as: $$s(i) = \frac{b(i) - a(i)}{max(b(i), a(i))}$$

Silhouette Index

Some tools visualize Silhouette index, each cluster is presented with one color and you can see the red cluster has one outlier because its silhouette coefficient is negative.

After identifying Silhouette coefficient, assuming *k* is the number of clusters, we can easily calculate Silhouette index as, follows:

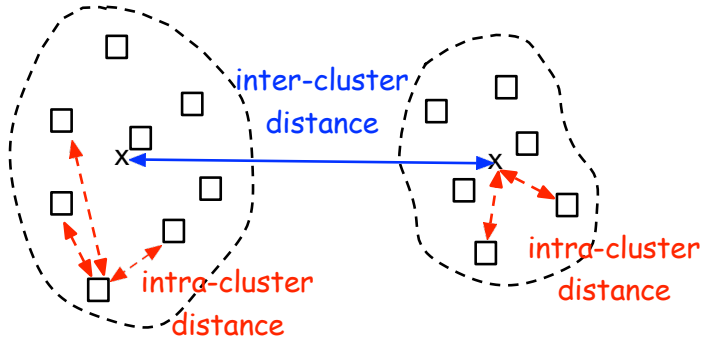
$$S_n = \frac{S_1 + S_2 + \ldots + S_k}{k}$$



Dunn Index

- Dunn Index reports the intra- class distance between two clusters divided by the maximum value of inter-class distance from one of those clusters.
- Assuming c_1, c_2, … are clusters, the DI can be written as follows:

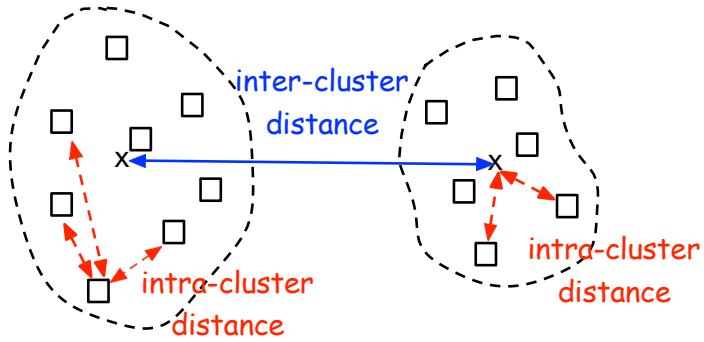
$$DI = \frac{\min \quad (\text{inter\_cluster\_disnace}(c_1, c_2))}{\max \quad (\text{intra\_cluster\_distnace}(c_1 \text{ or } c_2))}$$



- A higher Dunn index reveals that clusters' members are compact and clusters are well separated from each other.

Davies-Bouldin Index

$$DBI = mean \quad (max \quad \{ \frac{inter\_distance(c1) + inter\_distance(c2)}{intra\_distance(C1,C2)} \})$$



All intrinsic methods are all operated based on the fact that a good clustering method is having a high intra-cluster similarity (low distance) and low inter-cluster similarity (high distance).

Extrinsic Methods

- Extrinsic evaluation will be done when a ground truth dataset is available. We use unsupervised learning, when we do not have a labeled data. However, this style of evaluation requires a small labeled dataset (ground truth dataset).
  - The matching-based methods
  - The information theory-based methods
  - The pairwise comparison-based methods

The matching-based methods

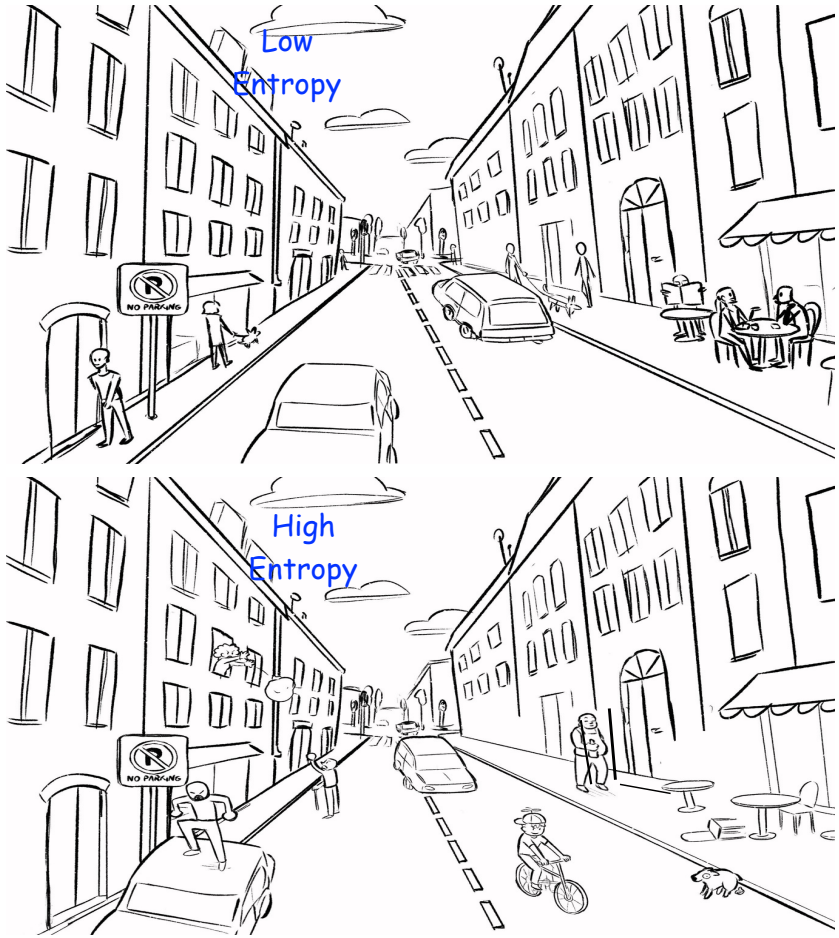
- Examines how well clustering results match the ground truth in partitioning the objects in the data set.
- For example, the purity methods assess how a cluster matches only those objects in one group in the ground truth.

The information theory-based methods

- Compare the distribution of the clustering results and that of the ground truth.
- Entropy or other measures in information theory are often employed to quantify the comparison.
- For example, we can measure the conditional entropy between the clustering results and the ground truth to measure whether there exists dependency between the information of the clustering results and the ground truth. The higher the dependency, the better the clustering results.

Entropy

- Entropy is a measure of the randomness or uncertainty in a system.
- In information theory, it quantifies the average amount of information contained in each message received.
- In statistical mechanics, it relates to the number of ways in which a system may be arranged, commonly associated with the second law of thermodynamics, which states that the entropy of an isolated system never decreases over time.



The pairwise comparison-based methods

- Treat each group in the ground truth as a class and then check the pairwise consistency of the objects in the clustering results.
- The clustering results are good if more pairs of objects of the same class are put into the same cluster, less pairs of objects of different classes are put into the same cluster, and less pairs of objects of the same class are put into different clusters.

Purity

- The higher the purity, the purer are the clusters, that is, the more objects in each cluster belong to the same group in the ground truth.

- Ci: clusters
- Gi: Ground truth groups

$$purity = \sum_{i=1}^{m} \frac{|C_i|}{n} \max_{j=1}^{l} \{ \frac{|C_i \cap G_j|}{|C_i|} \} = \frac{1}{n} \sum_{i=1}^{m} \max_{j=1}^{l} \{ |C_i \cap G_j| \}$$

Rand Index

RI is also the result of comparing our algorithm result with the ground truth dataset.

$$RI = \frac{TP + TN}{TP + FP + FN + TN}$$

Rand Index

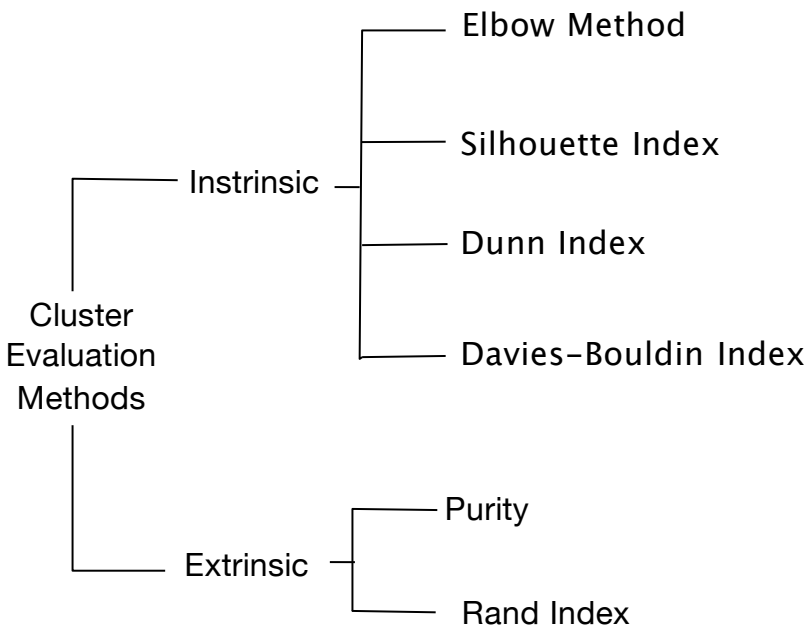
Consider C as clustering results of algorithms and G are clustering results of ground truth.

a. the number of data objects that are in the **same** clusters of C in and in the **same** clusters of G
b. the number of data objects that are in **different** clusters of C and in **different** clusters of G
c. the number of data objects that are in the **same** clusters of C and in **different** clusters of G
d. the data objects that are in the in **different** clusters of C and in the **same** clusters of G

a+ b = number of agreements, c+d = number of disagreements.

$$RI = \frac{a + b}{a + b + c + d}$$

Intrinsic and extrinsic cluster evaluation methods



References

- Galli, Soledad. Python Feature Engineering Cookbook: Over 70 recipes for creating, engineering, and transforming features to build machine learning models. Packt Publishing. Kindle Edition.
- Dey, Sandipan. Image Processing Masterclass with Python : 50+ Solutions and Techniques Solving Complex Digital Image Processing Challenges Using Numpy, Scipy, Pytorch and Keras (English Edition). BPB Publications. Kindle Edition.