# Clustering: Nearest-Neighbor Chain Algorithm

**Data Mining Project**

**Alexandra-Elena Mitulescu**

## 1. Importance and practical applications of the algorithm

Clustering is one of the most well-known Data Science approaches. It has a wide range of applications, from customer segmentation to outlier identification, and diverse algorithms that match different use cases.

Clustering is highly reliant on density and distance, but as dimensionality increases, these concepts become increasingly difficult to define. In several scientific papers, clustering algorithms, such as nearest-neighbor chain algorithm is used in applications that use high-dimensional data (actually, for data of any dimensionality). This algorithm is one of the fundamental algorithms in machine learning. A similarity measure based on the number of neighbors that two points share can be obsvered, and density can be defined as the sum of the similarities of a point's nearest neighbors. Then, based on these concepts a new clustering algorithm can be implemented.

This algorithm is of great importance because it removes noise (identified as low density points) and creates clusters by connecting non-noise points to representative or core points (identified as high density points).

Despite its simplicity, nearest-neighbor chain algorithm outperforms more powerful classifiers in a variety of applications, including:

- economic forecasting,
- data compression,
- genetics.

Traditional clustering approaches have occasionally been utilized for high-dimensional data.

Some interesting applications that I have discovered while doing my research for this project would be:

- **US Senator Clustering through Twitter**

Researchers have been looking at Twitter as a source of very valuable data since the disputed "Twitter mood predicts the stock market" article. In this example, Twitter is used to group US senators into political parties. The data is straightforward: it is examined which senators follow which other senators. This creates a network structure with senators as nodes and followers as edges.

The researcher utilized the Walktrap's method, which takes a random walk over the network and estimates senator similarity by the number of times you end up at a specific senator starting from a senator.

- **Charting Evolution through Phylogenetic Trees**

Scientists fought for decades to settle a seemingly easy question: how related are some animals to the others? We may now utilize DNA sequencing and hierarchical clustering to reconstruct the evolutionary tree of animal evolution:

- Make the DNA sequences

- Determine the edit distance between each sequence.
- Based on the edit distances, compute the DNA similarities
- Create a phylogenetic tree.

- **Tracking Viruses through Phylogenetic Trees**

Tracking viral epidemics and their causes is a significant public health concern. Tracing these epidemics to their origins can provide scientists with more information about why and how the outbreak started, perhaps saving lives.

Viruses, such as HIV, have a high mutation rate, which implies that the similarity of the DNA sequence of the same virus depends on how long it has been propagated. This may be used to track transmission pathways.

This technology was utilized as evidence in a court case, when it was discovered that the victim's strand of HIV was more similar to the accused patient's strand than a control group.

We can utilize agglomerative clustering, such as nearest neighbor to find the dendrogram once we have these commonalities.

## 2. The algorithm – general presentation

In that data are grouped, clustering is comparable to classification. The groupings, unlike categorization, are not predetermined. Instead, the grouping is achieved by discovering commonalities across data based on attributes discovered in the real data. Clusters are the names given to the groupings. According to some writers, clustering is a subset of classification. However, researchers tendto take a more traditional stance and argue that the two are distinct.

Many cluster definitions have been proposed:

- A collection of related components. Elements from several clusters are not identical.
- The distance between two points in a cluster is shorter than the distance between any two places outside the cluster.

Database segmentation is a phrase related to clustering in which like tuples (records) in a database are grouped together. This is done to partition or segment the database into components that provide the user with a more comprehensive view of the data.

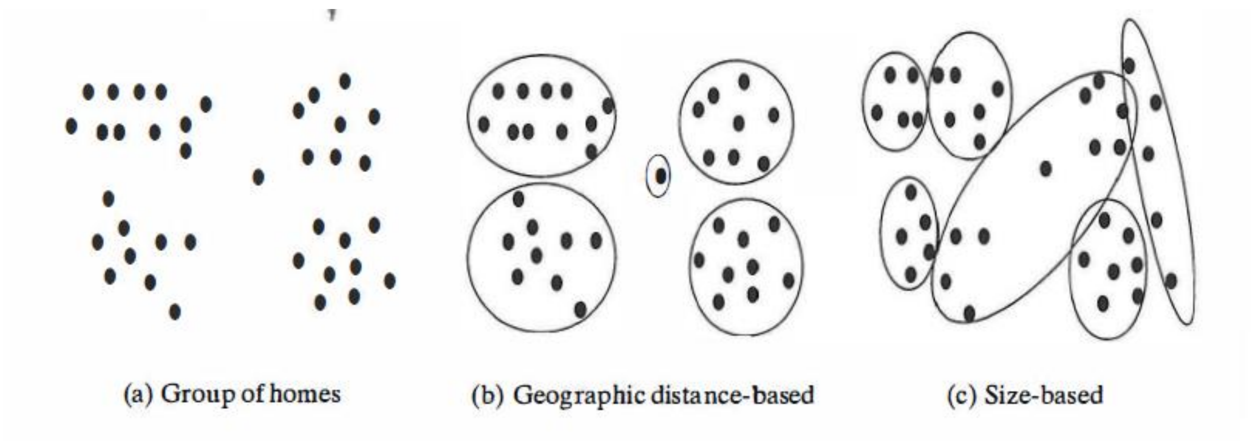


*Figure 1 Different clustering attributes*

A given collection of data can be grouped on different properties, as shown in Figure 1. A group of houses in a specific geographic location is depicted here. The first sort of clustering is dependent on the home's location. Homes in close proximity to one another are grouped together. Homes are placed in the second clustering depending on their size.

Many fascinating difficulties arise when clustering is applied to a real-world database:

- Handling outliers is challenging. The elements in this case do not naturally cluster.They can be thought of as lonely clusters. If a clustering algorithm tries to locate larger clusters, these outliers will be compelled to be placed in one of them. By joining two existing clusters and leaving the outlier in its own cluster, this technique may result in the formation of bad clusters.
- Because the database contains dynamic data, cluster membership may vary over time
- It may be challenging to interpret the semantic meaning of each cluster. The labeling of the classes is known ahead of time with classification. However, with clustering, this may not be the case.

We can summarize some basic features of clustering (as opposed to classification):

- The (best) number of clusters is not known.
- There may not be any a priori knowledge concerning the clusters.
- Cluster results are dynamic.

A classification of the different types of clustering algorithms is shown in Figure 2.

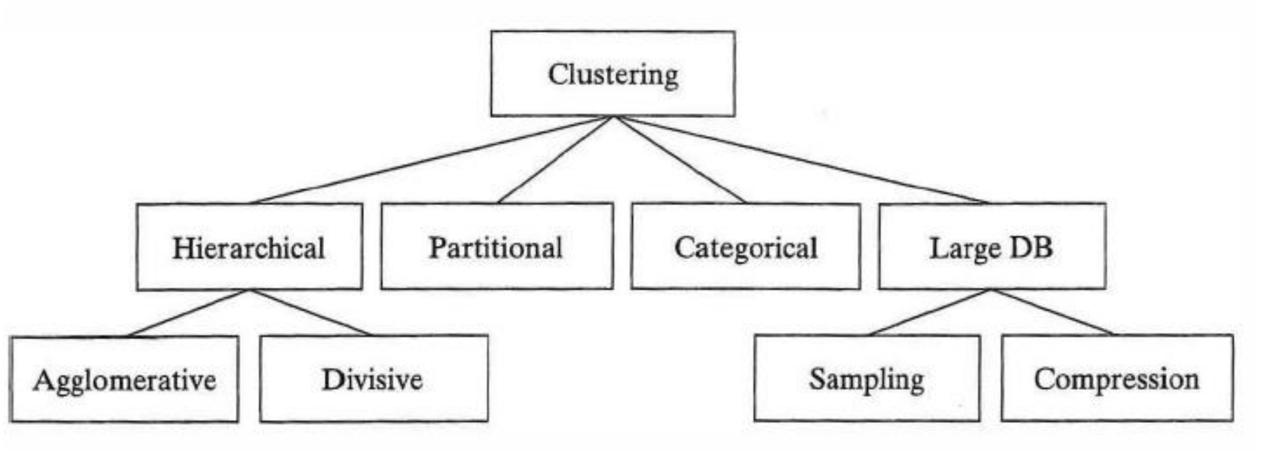


*Figure 2 Classification of clustering algorithms*

Every level of the hierarchy has its own set of clusters. At the most basic level, each item is in its own distinct cluster. At the most basic level, all things are members of the same cluster. The required number of groups is not specified with hierarchical clustering.

Clustering methods are divided further based on the imlementation methodology utilized. Hierarchical algorithms are classified as either agglomerative or divisive. The term "agglomerative" implies that clusters are formed from the bottom up, whereas divisive algorithms work from the top down. Although agglomerative can be used to describe both hierarchical and partitional algorithms, it is most commonly associated with hierarchical algorithms.

- **Hierarchical clustering** = creates sets of clusters
- **Agglomerative algorithms** = start with each individual item in its own cluster and iteratively merge clusters until all items belong in one cluster

Many clustering algorithms require that the distance between clusters (rather than elements) be determined. This is not an easy task given that there are many interpretations for distance between clusters. Given clusters K_i and K_J , there are several standard alternatives to calculate the distance between clusters. A representative list is :

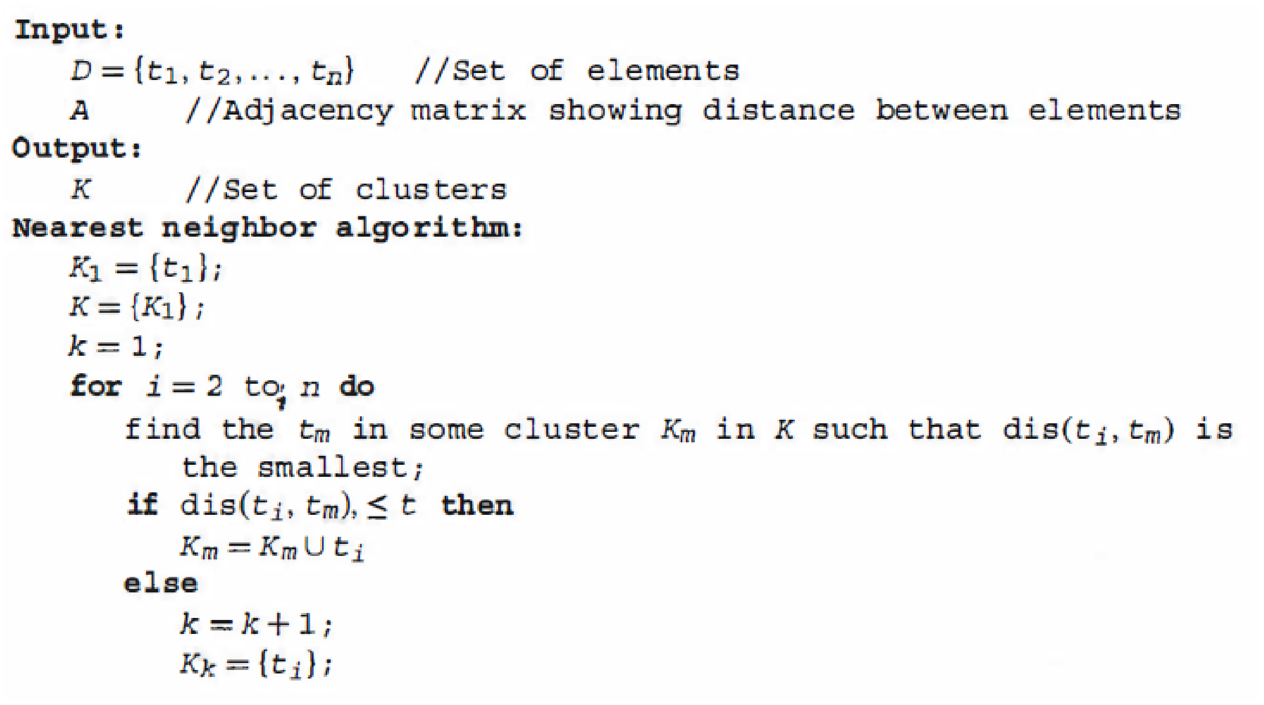
- **Single link** (or nearest neighbor chain algorithm): Smallest distance between an element in one cluster and an eiement in the other.
- **Complete link**: Largest distance between an element in one cluster and an element in the other.
- **Average**: Average distance between an element in one cluster and an element in the other.
- **Centroid**: If clusters have a representative centroid, then the centroid distance is defined as the distance between the centroids.
- **Medoid**: Using a medoid to represent each cluster, the distance between the clusters can be defined by the distance between the medoids

The nearest-neighbor chain technique is a cluster analysis approach that can speed up numerous methods for agglomerative hierarchical clustering. These are algorithms that take a set of points as input and generate a hierarchy of point clusters by merging pairs of smaller clusters to produce larger clusters. Ward's technique, complete-linkage clustering, and single-linkage clustering are all clustering methods that can be used with the nearest-neighbor chain algorithm; they all function by repeatedly merging the closest two groups but utilize various definitions of the distance between clusters.

The cluster distances for which the nearest-neighbor chain algorithm works are known as reducible distances, and they are distinguished by a simple inequality between certain cluster distances.

The algorithm's core idea is to locate pairings of clusters to merge by following paths in the clusters' closest neighbor graphs. Every such path will eventually end at a pair of clusters that are nearest neighbors to one another, and the algorithm will select that pair of clusters to merge. The method employs a stack data structure to keep track of each path that it follows in order to save work by reusing as much of each path as possible. The nearest-neighbor chain algorithm merges its clusters in a different sequence than approaches that always discover and merge the closest pair of clusters by following these paths. Regardless of the change, it always generates the same hierarchy.

The nearest-neighbor chain technique produces a clustering in time proportional to the square of the number of clustered points. When the input is provided in the form of an explicit distance matrix, this is similarly proportional to the size of its input. When used for clustering methods such as Ward's method, which allow constant-time determination of the distance between clusters, the algorithm consumes memory proportionate to the number of points. However, for some other clustering methods, it requires more RAM in an auxiliary data structure that keeps track of the distances between clusters.



```
Input:
   $D = \{t_1, t_2, \ldots, t_n\}$   //Set of elements
   $A$      //Adjacency matrix showing distance between elements
Output:
   $K$      //Set of clusters
Nearest neighbor algorithm:
   $K_1 = \{t_1\}$;
   $K = \{K_1\}$;
   $k = 1$;
   for $i = 2$ to $n$ do
      find the $t_m$ in some cluster $K_m$ in $K$ such that dis($t_i, t_m$) is
         the smallest;
      if dis($t_i, t_m$), $\leq t$ then
         $K_m = K_m \cup t_i$
      else
         $k = k + 1$;
         $K_k = \{t_i\}$;
```

*Figure 3 Nearest-Neighbor Chain Algorithm*

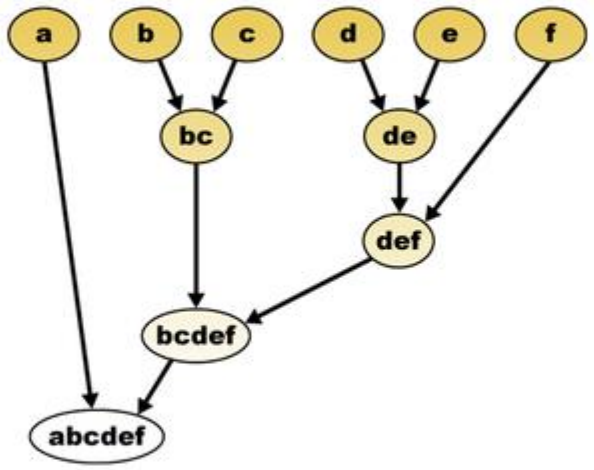


*Figure 4 Output of the nearest-neighbor algorithm*

In Figure 4, we can observe a hierarchical clustering of six points. The points to be clustered are at the top of the diagram, and the nodes below them represent clusters.

Intuitively, the nearest neighbor chain algorithm repeatedly follows a chain of clusters $A \rightarrow B \rightarrow C \rightarrow$ ... where each cluster is the nearest neighbor of the previous one, until reaching a pair of clusters that are mutual nearest neighbors.

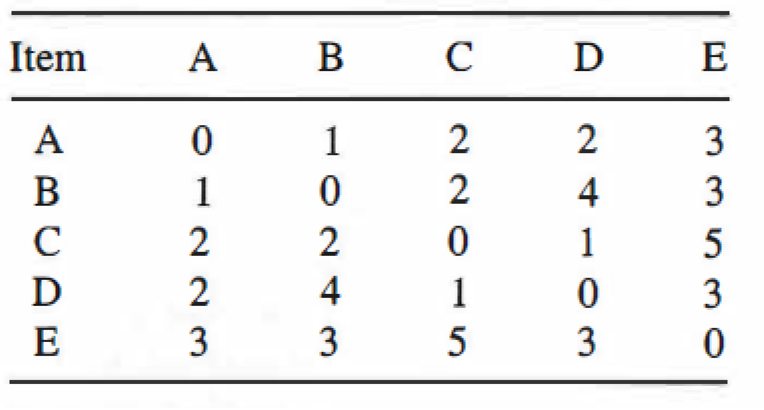
In more detail, the algorithm performs the following steps:

- Initialize the set of active clusters to consist of *n* one-point clusters, one for each input point.
- Let *S* be a stack data structure initially empty, the elements of which will be active clusters.

- While there is more than one cluster in the set of clusters:
    - If *S* is empty, choose an active cluster arbitrarily and push it onto *S*.
    - Let *C* be the active cluster on the top of *S*. Compute the distances from *C* to all other clusters, and let *D* be the nearest other cluster.
    - If *D* is already in *S*, it must be the immediate predecessor of *C*. Pop both clusters from *S* and merge them.
    - Otherwise, if *D* is not already in *S*, push it onto *S*.

When it is possible for one cluster to have multiple equal nearest neighbors, then the algorithm requires a consistent tie-breaking rule. For instance, one may assign arbitrary index numbers to all of the clusters, and then select (among the equal nearest neighbors) the one with the smallest index number. This rule prevents certain kinds of inconsistent behavior in the algorithm; for instance, without such a rule, the neighboring cluster *D* might occur earlier in the stack than as the predecessor of *C*.

***EXAMPLE***



| Item | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 2 | 2 | 3 |
| B | 1 | 0 | 2 | 4 | 3 |
| C | 2 | 2 | 0 | 1 | 5 |
| D | 2 | 4 | 1 | 0 | 3 |
| E | 3 | 3 | 5 | 3 | 0 |

*Figure 5 Sample data*

Initially, *A* is placed in a cluster by itself, so we have *K1* = {*A*}. We then look at B to decide if it should be added to *K1* or be placed in a new cluster. Since dis(*A*, *B*) = 1, which is less than the threshold of 2, we place *B* in *K*1 to get *K1* = {*A* , *B*}. When looking at C, we see that its distance to both A and B is 2, so we add it to the cluster to get *K1* = {*A* , *B, C* }. The dis(*D* , *C*) = 1 < 2, so we get *K1* = {*A, B, C* , *D*}. Finally, looking at E, we see that the closest item in K 1 has a distance of 3 , which is greater than 2, so we place it in its own cluster: *K2* = {*E*} .

The complexity of the nearest neighbor algorithm actually depends on the number of items. For each loop, each item must be compared to each item already in a cluster.

Obviously, this is n in the worst case. Thus, the time complexity is O(n2). Since we do need to examine the distances between items often, we assume that the space requirement is also O(n2).

## 3. Known results and issues

Cluster analysis has long been utilized in a broad range of domains, including psychology and other social sciences, biology, statistics, pattern recognition, information retrieval, machine learning, and data mining.

Cluster analysis is a difficult process with a number of well-known challenges, such as locating clusters in data with clusters of varying forms, sizes, and densities, or in data with a lot of noise and outliers. These concerns become increasingly pressing in the setting of high-dimensional data sets.

Traditional clustering approaches have occasionally been utilized for high-dimensional data. For grouping document data, for example, the K-means method and agglomerative hierarchical clustering approaches have been widely employed. While K-means is efficient and frequently delivers "acceptable" results, it retains all of its low dimensional constraints in large dimensions, i.e., it has problems with outliers and does not perform well when the clusters in the data are of diverse sizes, shapes, and densities. Problems exist with agglomerative hierarchical clustering systems, which are frequently assumed to be superior to K-means for low-dimensional data. For example, single link systems are extremely susceptible to noise and density changes. While group averages and full links are less susceptible to noise, they struggle with varying densities and, unlike single links, cannot accommodate clusters of varying forms and sizes.

Part of the issues with hierarchical clustering algorithms, such as nearest neighbor chain algorithm, stem from distance issues in high-dimensional space. It is commonly known that Euclidean distance does not perform well in high dimensions, and most clustering algorithms utilize distance or similarity measures that do, such as the cosine measure. However, using similarity metrics such as the cosine measure does not solve all similarity difficulties. Points in high-dimensional space, in particular, frequently have low similarities, and hence points in different clusters might be closer than points in the same cluster.

In [2], it has been discovered that 15-20% of a point's nearest neighbors were of a different class in many TREC datasets (with class labels) that were analyzed in the first place [3]. Our method to high-dimensional similarity starts with a k nearest neighbor list generated using the original similarity measure, but then constructs a new similarity measure based on the number of nearest neighbors shared by two points.

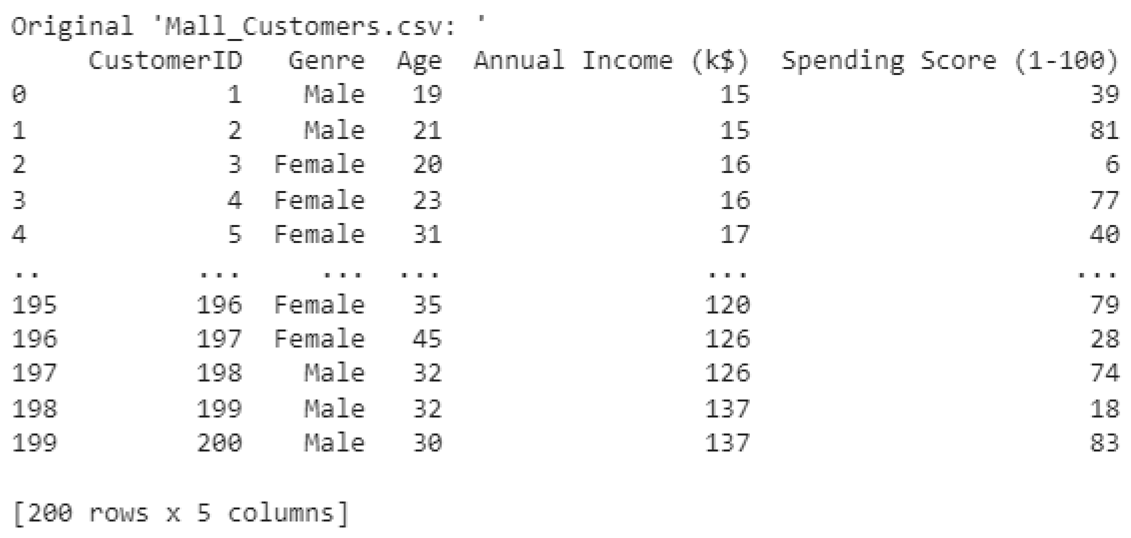
## 4. Dataset used

The dataset that I have used in this project can be found by accessing the following link:
https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python

This „mall_customers.csv" dataset was created for the learning purpose of the customer segmentation concepts, also known as market basket analysis.

The content of the csv is represented by some basic data about customers that come to a supermarket mall. The essential data is formed from:

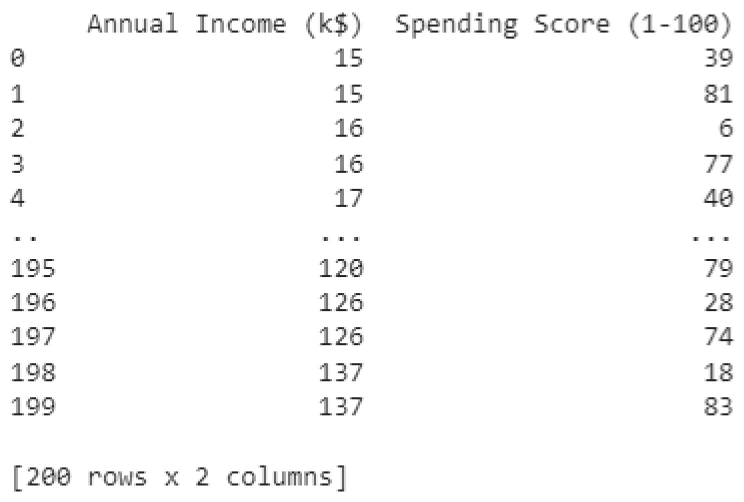
- CustomerID
- Age
- Gender
- Annual Income
- Spending Score



```
Original 'Mall_Customers.csv: '
     CustomerID   Genre  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
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
196         197  Female   45                 126                      28
197         198    Male   32                 126                      74
198         199    Male   32                 137                      18
199         200    Male   30                 137                      83

[200 rows x 5 columns]
```

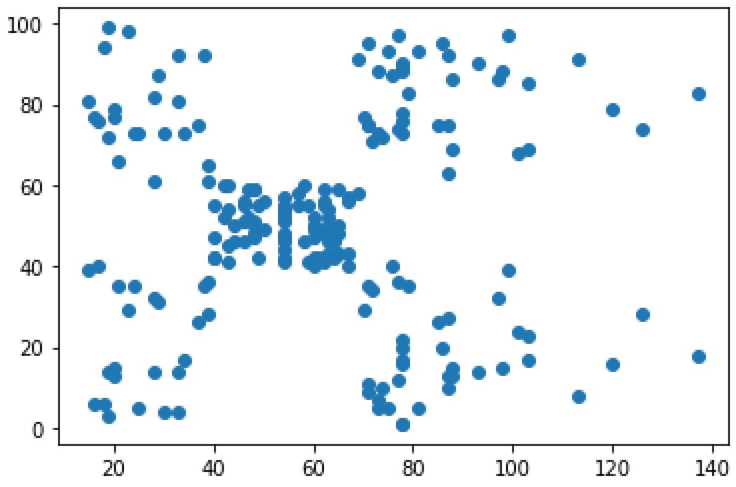
I have removed unecessary information regarding our customers, keeping just the „Annual Income” and „Spending Score” values.



```
     Annual Income (k$)  Spending Score (1-100)
0                    15                      39
1                    15                      81
2                    16                       6
3                    16                      77
4                    17                      40
..                  ...                     ...
195                 120                      79
196                 126                      28
197                 126                      74
198                 137                      18
199                 137                      83

[200 rows x 2 columns]
```

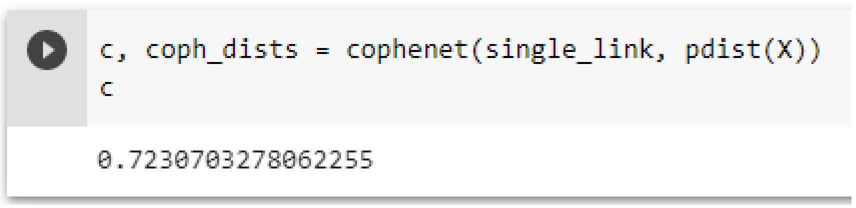
To observe the relationship between the remaining variables, I used a scatter plot and to see how change in one affects the other.



## 5. Results (including samples and evaluation)

I have used the routine ***cophenet(Z,[,Y])*** from spicy.cluster.hierarchy to compute a statistic on our hierarchy. Which calculates the cophenetic distances between each observation in the hierarchical clustering defined by the linkage Z.

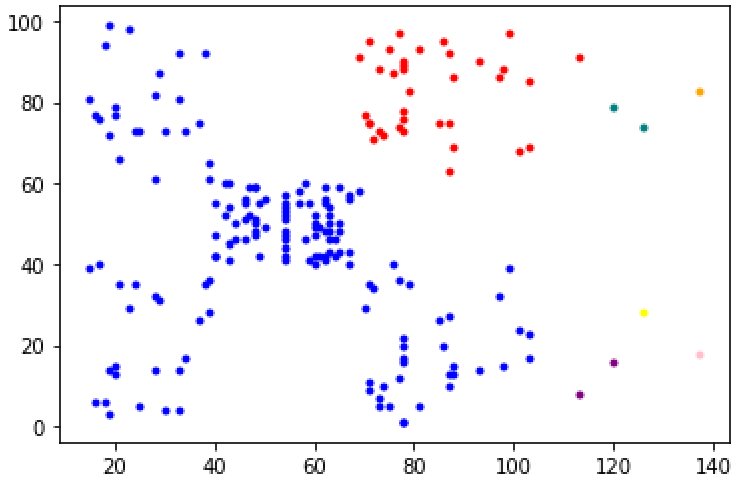
The closer the value is to 1, the better the clustering preserves the original distances, which in our case is reasonably close.



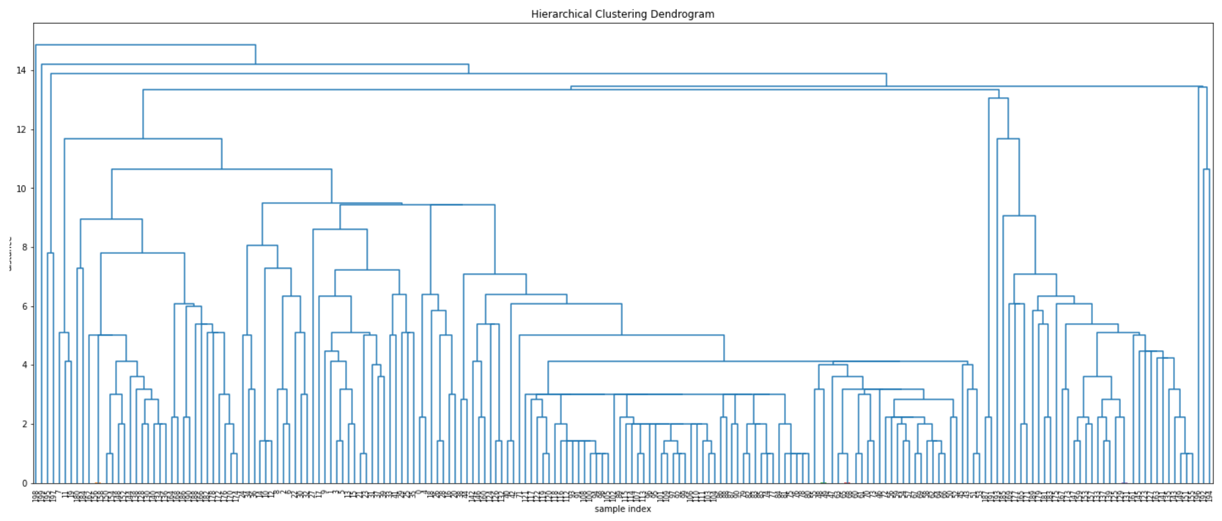
```
c, coph_dists = cophenet(single_link, pdist(X))
c
```

```
0.7230703278062255
```

This is the distribution of the formed clusters when the given threshold is equal to 7.



Also, for visualization I have used a dendrogram in form of a tree showing the order and distances of merges for each customer by their Annual Income(k$) and Spending Score(1-100).



## 6. References

[1] https://en.wikipedia.org/wiki/Nearest-neighbor_chain_algorithm

[2] Levent Ertöz, Michael Steinbach, Vipin Kumar - "A new shared nearest neighbor clustering algorithm and its applications"

[3] Michael Steinbach, George Karypis, and Vipin Kumar, "A Comparison of Document Clustering Algorithms," KDD-2000 Text Mining Workshop, 2000.

[4] https://towardsdatascience.com/hierarchical-clustering-and-its-applications-41c1ad4441a6