There are several Machine Learning algorithms, one such important algorithm of machine learning is Clustering.

Clustering is an unsupervised learning method in machine learning. It means that it is a machine learning algorithm that can draw inferences from a given dataset on its own, without any kind of human intervention.

Types of clustering method

There are five types of clustering methods in machine learning, these are as follows:

1.Partitioning Clustering

2.Density-Based Clustering

3.Distribution Model-Based Clustering

4.Hierarchical Clustering

5.Fuzzy Clustering

About Hierarchical Clustering

Hierarchical clustering, also known as hierarchical cluster analysis or HCA, is another unsupervised machine learning approach for grouping unlabeled datasets into clusters.

The hierarchy of clusters is developed in the form of a tree in this technique, and this tree-shaped structure is known as the dendrogram.

Simply speaking, Separating data into groups based on some measure of similarity, finding a technique to quantify how they're alike and different, and limiting down the data is what hierarchical clustering is all about.

Hierarchical clustering method functions in two approaches-

1.Agglomerative

2.Divisive

Approaches of Hierarchical Clustering

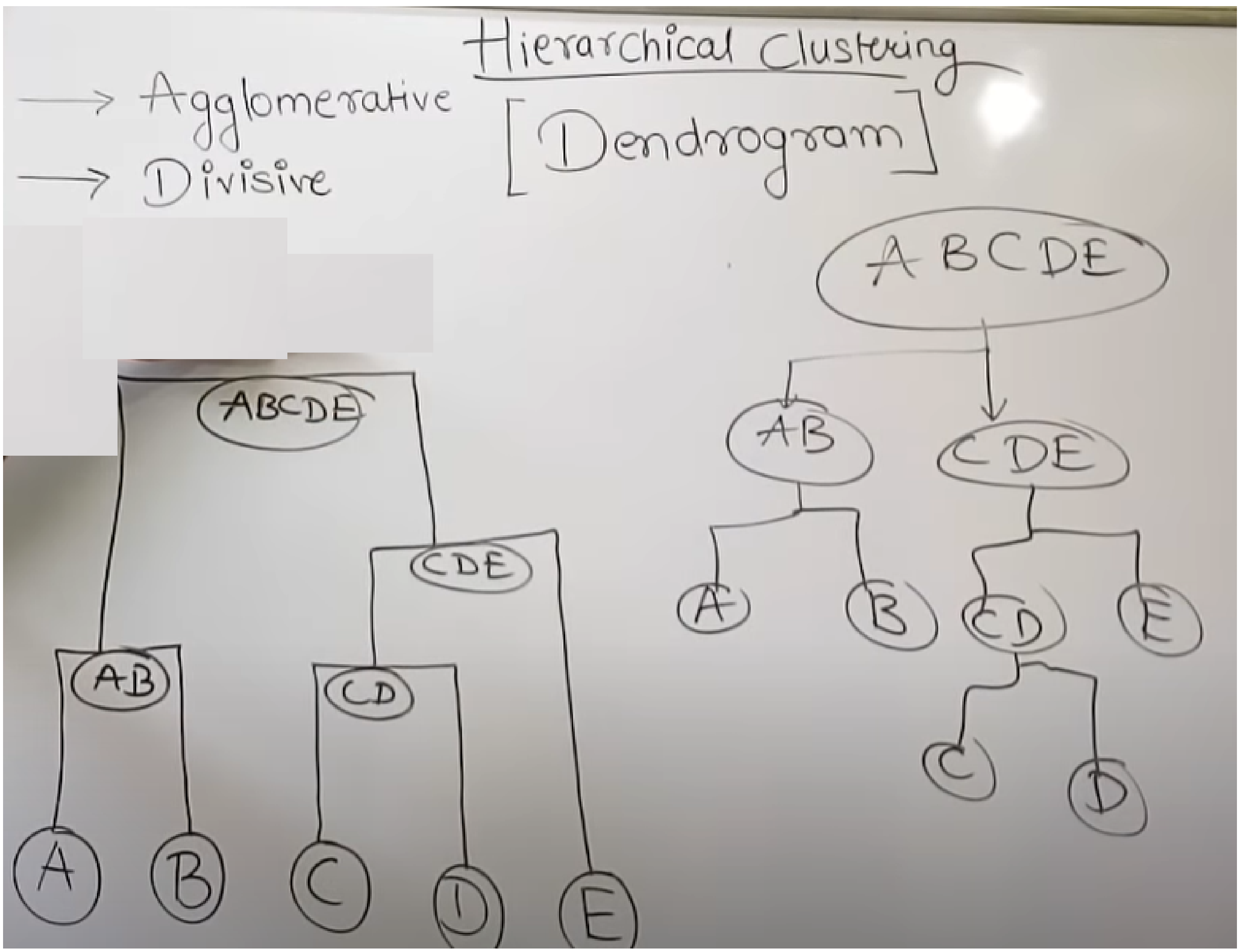
1.Agglomerative clustering:

Agglomerative Clustering is a bottom-up strategy in which each data point is originally a cluster of its own, and as one travels up the hierarchy, more pairs of clusters are combined. In it, two nearest clusters are taken and joined to form one single cluster.

2.Divisive clustering:

The divisive clustering algorithm is a top-down clustering strategy in which all points in the dataset are initially assigned to one cluster and then divided iteratively as one progresses down the hierarchy.

It partitions data points that are clustered together into one cluster based on the slightest difference. This process continues till the desired number of clusters is obtained.



Double-click (or enter) to edit

How hierarchical clustering works

Hierarchical clustering starts by treating each observation as a separate cluster. Then, it repeatedly executes the following two steps:

(1) identify the two clusters that are closest together,

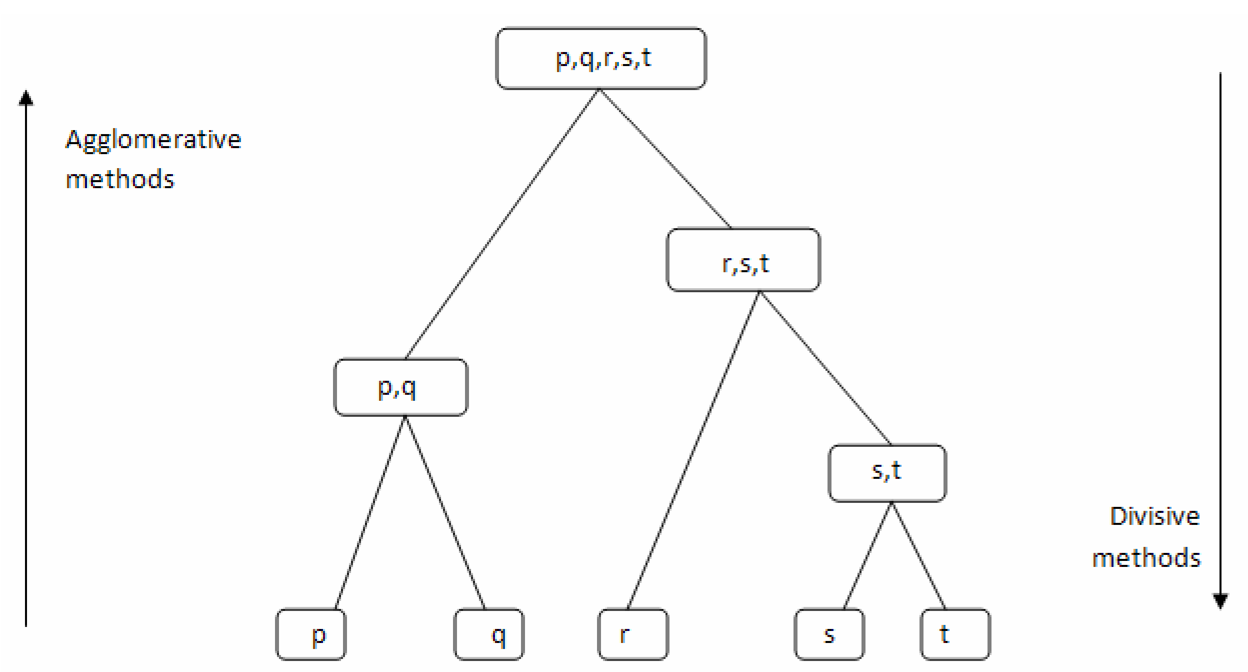
and (2) merge the two most similar clusters. This iterative process continues until all the clusters are merged together. This is illustrated in the diagrams below

Hierarchical clustering algorithms group similar objects into groups called

clusters. There are two types of hierarchical clustering algorithms:

Agglomerative — Bottom up approach. Start with many small clusters and merge them together to create bigger clusters.

Divisive — Top down approach. Start with a single cluster than break it up into smaller clusters.



Double-click (or enter) to edit

Some pros and cons of Hierarchical Clustering

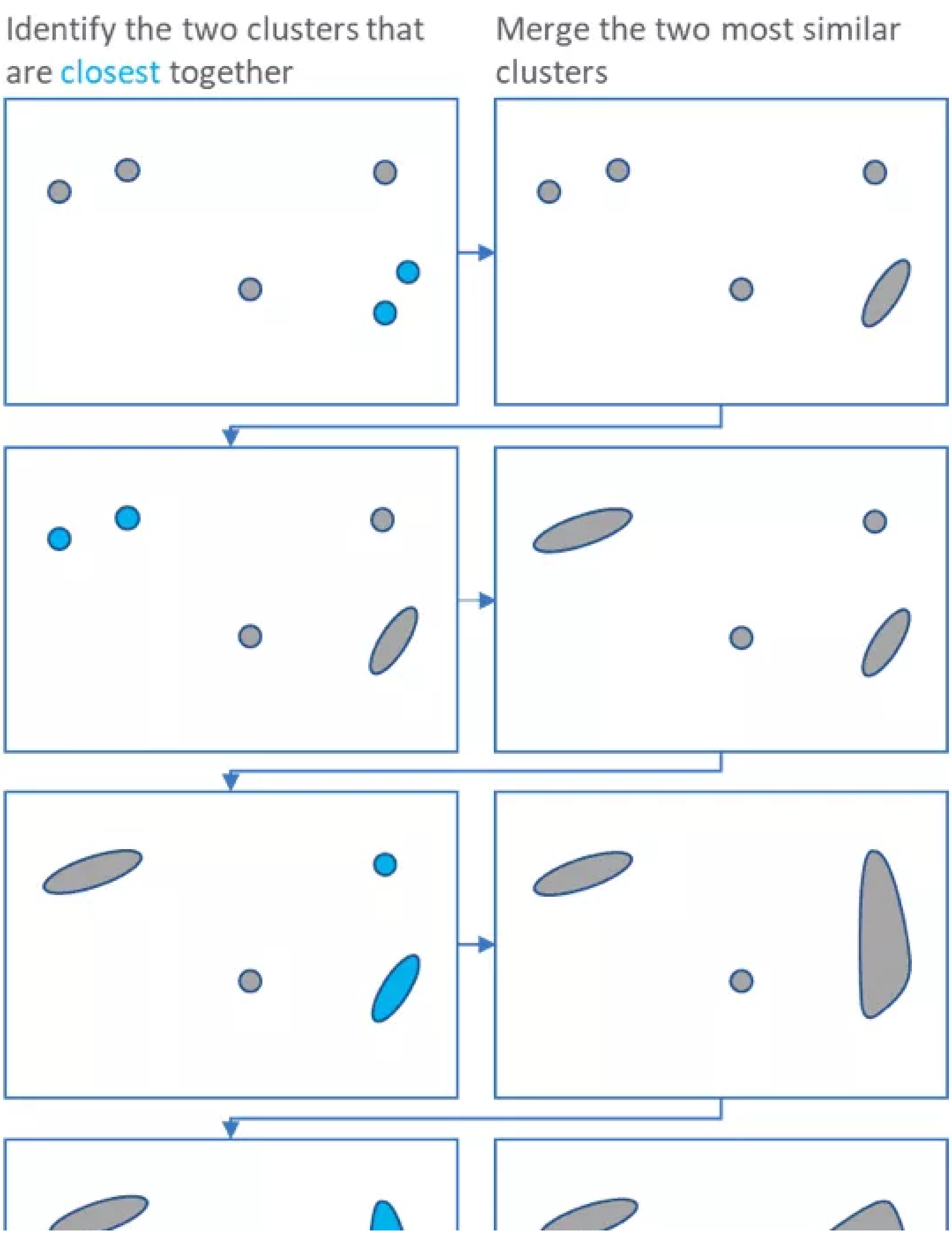
Pros

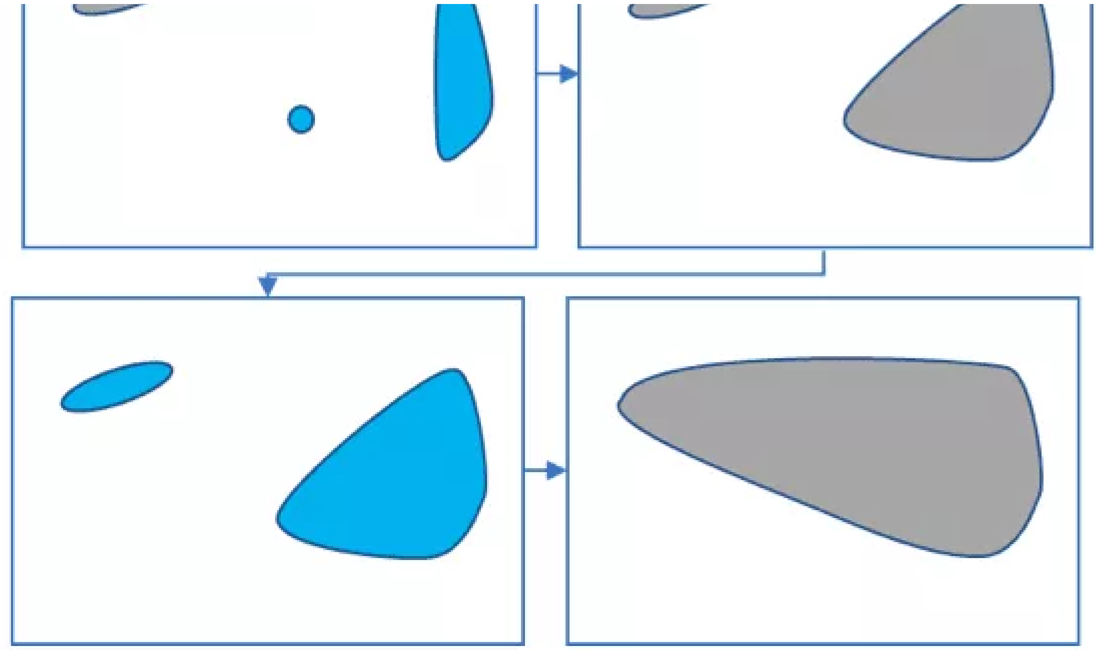
No assumption of a particular number of clusters (i.e. k-means) May correspond to meaningful taxonomies

Cons

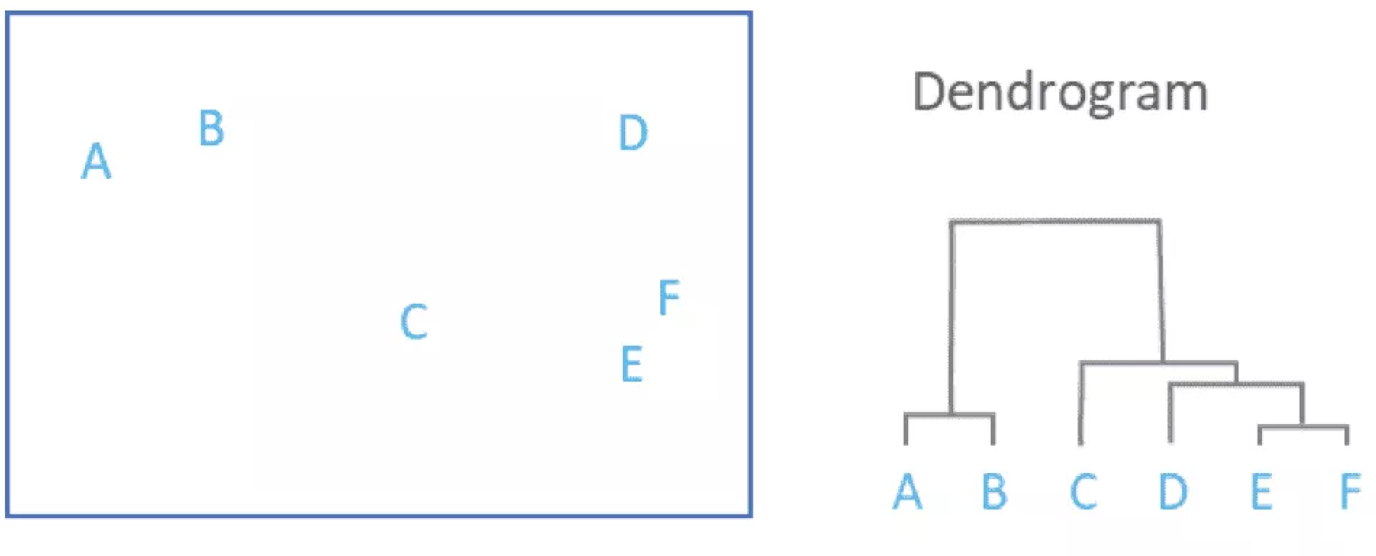
Once a decision is made to combine two clusters, it can’t be undone Too slow for large data sets, O(n2 log(n))

How its Work





The main output of Hierarchical Clustering is a dendrogram, which shows the hierarchical relationship between the clusters:



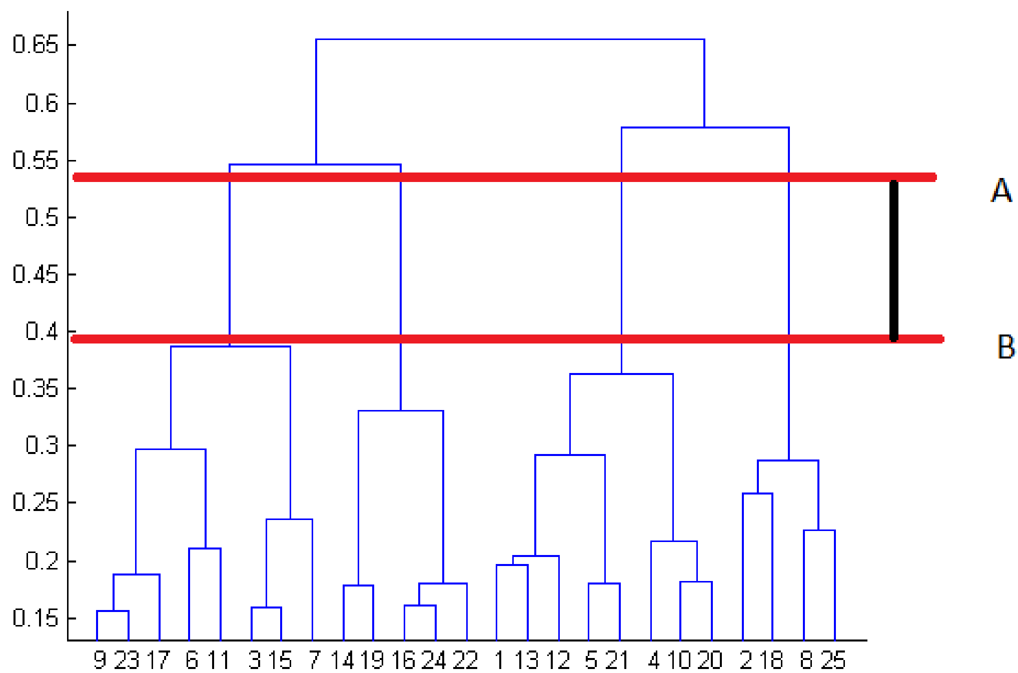
Dendrograms

We can use a dendrogram to visualize the history of groupings and figure out the optimal number of clusters.

1. Determine the largest vertical distance that doesn't intersect any of the other clusters

2.Draw a horizontal line at both extremities

3.The optimal number of clusters is equal to the number of vertical lines going through the horizontal line For eg., in the below case, best choice for no. of clusters will be 4.



Linkage Criteria

Similar to gradient descent, you can tweak certain parameters to get drastically different results.

Single Linkage The distance between two clusters is the shortest distance between two points in each cluster

Complete Linkage The distance between two clusters is the longest distance between two points in each cluster

Average Linkage The distance between clusters is the average distance between each point in one cluster to every point in other cluster

Ward Linkage The distance between clusters is the sum of squared differences within all clusters

Double-click (or enter) to edit

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

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Reading the data file into a DATAFRAME and checking the shape
dataset=pd.read_csv("/content/drive/My Drive/Colab Notebooks/Mall_Customers.csv")
print(dataset.shape)
dataset
```

```
(200, 5)
```

	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 × 5 columns

Python loc() function

The loc() function is label based data selecting method which means that we have to pass the name of the row or column which we want to select. This method includes the last element of the range passed in it, unlike iloc(). loc() can accept the boolean data unlike iloc(). Many operations can be performed using the loc() method like

Python iloc() function

The iloc() function is an indexed-based selecting method which means that we have to pass an integer index in the method to select a specific row/column. This method does not include the last element of the range passed in it unlike loc(). iloc() does not accept the boolean data unlike loc(). Operations performed using iloc() are:

Example 1:

Selecting rows using integer indices

Double-click (or enter) to edit

```
# selecting 0th, 2th, 4th, and 7th index rows
display(dataset.iloc[[0, 2, 4, 7]])
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
2	3	Female	20	16	6
4	5	Female	31	17	40
7	8	Female	23	18	94

```
x=dataset.iloc[:2,-1:]
x
```

	Spending Score (1-100)
0	39
1	81

```
X = dataset.iloc[:, [3, 4]].values
X
```

```
       [ 75,  93],
       [ 76,  40],
       [ 76,  87],
       [ 77,  12],
       [ 77,  97],
       [ 77,  36],
       [ 77,  74],
       [ 78,  22],
       [ 78,  90],
       [ 78,  17],
       [ 78,  88],
       [ 78,  20]
```

```
       [ 78,  20],
       [ 78,  76],
       [ 78,  16],
       [ 78,  89],
       [ 78,   1],
       [ 78,  78],
       [ 78,   1],
       [ 78,  73],
       [ 79,  35],
       [ 79,  83],
       [ 81,   5],
       [ 81,  93],
       [ 85,  26],
       [ 85,  75],
       [ 86,  20],
       [ 86,  95],
       [ 87,  27],
       [ 87,  63],
       [ 87,  13],
       [ 87,  75],
       [ 87,  10],
       [ 87,  92],
       [ 88,  13],
       [ 88,  86],
       [ 88,  15],
       [ 88,  69],
       [ 93,  14],
       [ 93,  90],
       [ 97,  32],
       [ 97,  86],
       [ 98,  15],
       [ 98,  88],
       [ 99,  39],
       [ 99,  97],
       [101,  24],
       [101,  68],
       [103,  17],
       [103,  85],
       [103,  23],
       [103,  69],
       [113,   8],
       [113,  91],
       [120,  16],
       [120,  79],
       [126,  28],
       [126,  74],
       [137,  18],
```

Training the Hierarchical Clustering model on the dataset

```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
```

```
y_hc
```

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 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, 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, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```

n_clusters int

The number of clusters to find. It must be None if distance_threshold is not None.

affinity

Metric used to compute the linkage.

linkage{‘ward’, ‘complete’, ‘average’, ‘single’}, default=’ward’

Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

‘ward’ minimizes the variance of the clusters being merged.

‘average’ uses the average of the distances of each observation of the two sets.

‘complete’ or ‘maximum’ linkage uses the maximum distances between all observations of the two sets.

‘single’ uses the minimum of the distances between all observations of the two sets.

Double-click (or enter) to edit

```
print(hc)
```

```
AgglomerativeClustering(n_clusters=5)
```

Visualising the clusters

Double-click (or enter) to edit

```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

How does it work?

We will use Agglomerative Clustering, a type of hierarchical clustering that follows a bottom up approach.

We begin by treating each data point as its own cluster. Then, we join clusters together that have the shortest distance between them to create larger clusters.

This step is repeated until one large cluster is formed containing all of the data points.

Hierarchical clustering requires us to decide on both a distance and linkage method.

We will use euclidean distance and the Ward linkage method, which attempts to minimize the variance between clusters.

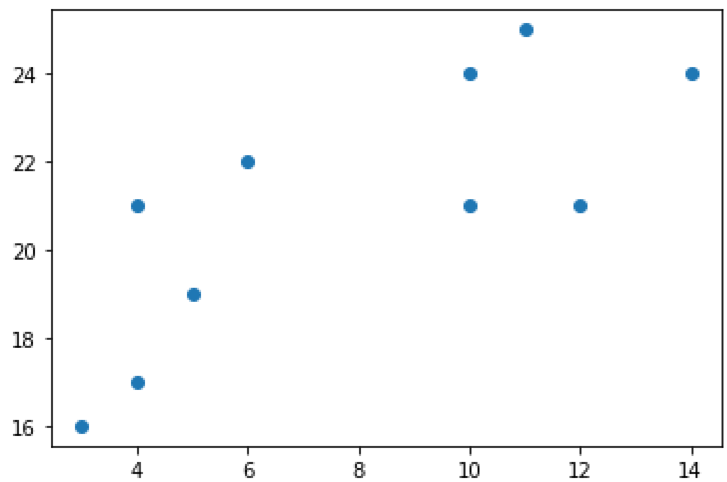
Example

Start by visualizing some data points:

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

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

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



Now we compute the ward linkage using euclidean distance, and visualize it using a dendrogram:

Example

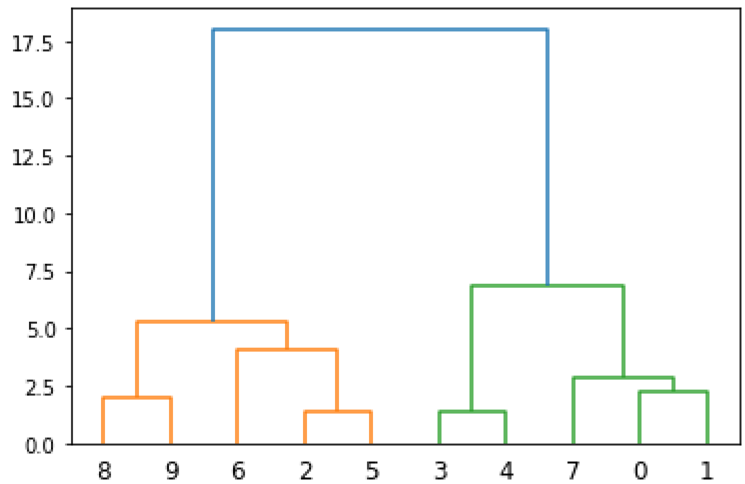
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

linkage_data = linkage(data, method='ward', metric='euclidean')
dendrogram(linkage_data)

plt.show()
```



Here, we do the same thing with Python's scikit-learn library. Then, visualize on a 2-dimensional plot:

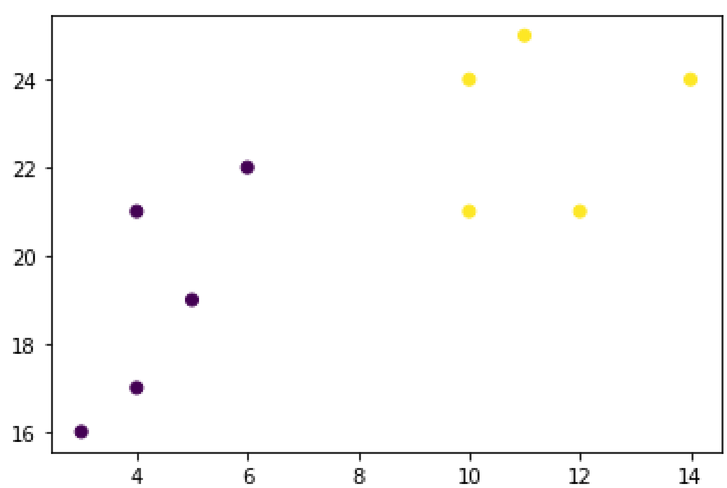
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='w
labels = hierarchical_cluster.fit_predict(data)

plt.scatter(x, y, c=labels)
plt.show()
```



Example Explained

Import the modules you need.

1.import numpy as np

2.import matplotlib.pyplot as plt

3.from scipy.cluster.hierarchy import dendrogram, linkage

4.from sklearn.cluster import AgglomerativeClustering

Create arrays that resemble two variables in a dataset.

Note that while we only two variables here, this method will work with any number of variables:

```
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
```

Turn the data into a set of points:

```
data = list(zip(x, y))
print(data)
```

```
[(4, 21), (5, 19), (10, 24), (4, 17), (3, 16), (11, 25), (14, 24), (6, 22), (10, 21), (1
```

Compute the linkage between all of the different points.

Here we use a simple euclidean distance measure and Ward's linkage,

which seeks to minimize the variance between clusters.

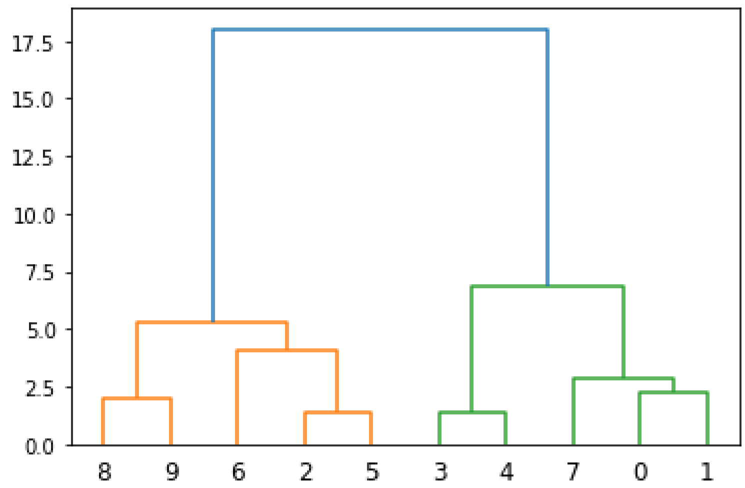
linkage_data = linkage(data, method='ward', metric='euclidean')

Finally, plot the results in a dendrogram.

This plot will show us the hierarchy of clusters from the bottom (individual points) to the top (a single cluster consisting of all data points).

plt.show() lets us visualize the dendrogram instead of just the raw linkage data.

```
dendrogram(linkage_data)
plt.show()
```



The scikit-learn library allows us to use hierarchichal clustering in a different manner.

First, we initialize the AgglomerativeClustering class with 2 clusters, using the same euclidean distance and Ward linkage.

hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')

The **.fit_predict** method can be called on our data to compute the clusters using the defined parameters across our chosen number of clusters.

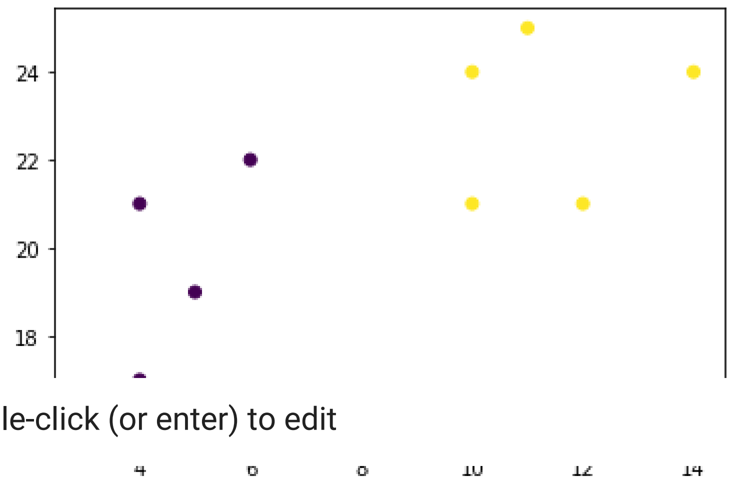
```
labels = hierarchical_cluster.fit_predict(data)
```

```
print(labels)
```

```
[0 0 1 0 0 1 1 0 1 1]
```

Finally, if we plot the same data and color the points using the labels assigned to each index by the hierarchical clustering method, we can see the cluster each point was assigned to:

```
plt.scatter(x, y, c=labels)
plt.show()
```



Double-click (or enter) to edit