We at The Data Monk hold the vision to make sure everyone in the IT industry has an equal stand to work in an open domain such as analytics. Analytics is one domain where there is no formal under-graduation degree and which is achievable to anyone and everyone in the World.

We are a team of 30+ mentors who have worked in various product-based companies in India and abroad, and we have come up with this idea to provide study materials directed to help you crack any analytics interview.

Every one of us has been interviewing for at least the last 6 to 8 years for different positions like Data Scientist, Data Analysts, Business Analysts, Product Analysts, Data Engineers, and other senior roles. We understand the gap between having good knowledge and converting an interview to a top product-based company.

Rest assured that if you follow our different mediums like our blog cum questions-answer portal www.TheDataMonk.com , our youtube channel - The Data Monk, and our e-books, then you will have a very strong candidature in whichever interview you participate in.

There are many blogs that provide free study materials or questions on different analytical tools and technologies, but we concentrate mostly on the questions which are asked in an interview. We have a set of 100+ books which are available both on Amazon and on The Data Monk e-shop page

We would recommend you to explore our website, youtube channel, and e-books to understand the type of questions covered in our articles. We went for the question-answer approach both on our website as well as our e-books just because we feel that the best way to go from beginner to advance level is by practicing a lot of questions on the topic.

We have launched a series of 50 e-books on our website on all the popular as well as niche topics. Our range of material ranges from SQL, Python, and Machine Learning algorithms to ANN, CNN, PCA, etc.

We are constantly working on our product and will keep on updating it. It is very necessary to go through all the questions present in this book.

Give a rating to the book on Amazon, do provide your feedback and if you want to help us grow then please subscribe to our Youtube channel.

Q1. What is an unsupervised learning approach and Why is it needed ?

A1. Unsupervised Learning is a technique in which models are not supervised using training dataset. Instead model itself finds the hidden pattern and relations, insights from the given dataset. It mainly deals with the unlabeled data.

Example:

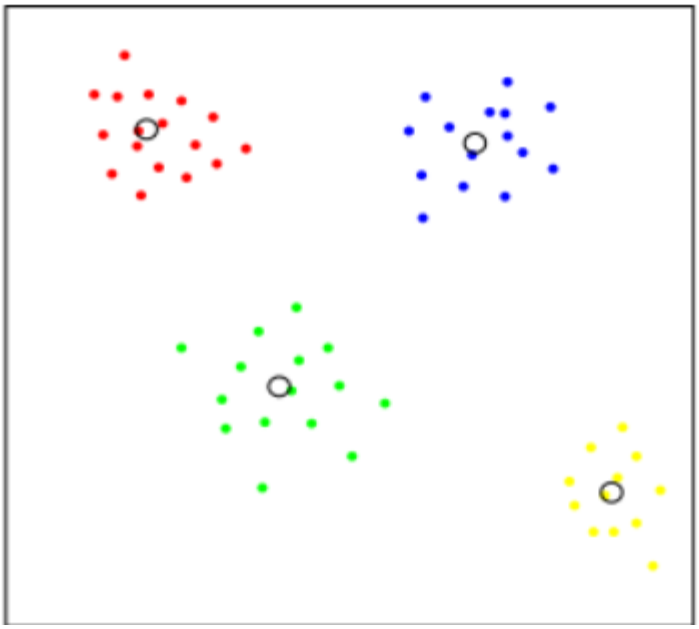
Let Suppose the unsupervised algorithm is given an input dataset containing images of of cats and dogs. The algorithm is never seen that data before. It is never trained upon the given dataset, which means it does not have any idea about the features (variables) of the dogs and cats.

Unsupervised learning algorithm automatically identifies the features(of the dataset. Unsupervised learning algorithm will perform this task with the help of clustering.

Q 2. What is Clustering? Explain with the help of proper Examples

A2. Clustering is an unsupervised approach which finds a structure/pattern in a collection of unlabeled data(in which do not have y value).

A cluster is a collection of data points which are “similar” amongst themselves and are “dissimilar” to the objects belonging to a different cluster. For example:



Let's suppose we give a child different objects to group. How does a child make a group?

The child may group over the colour, over the shape, over the hardness or softness of the objects etc. The basic idea here is that the child tries to find out similarities and dissimilarities between different objects and then tries to make a group of similar objects. This is called **clustering**, the method of identifying similar instances and keeping them together. In Other words, clustering identifies homogeneous subgroups among the observations.

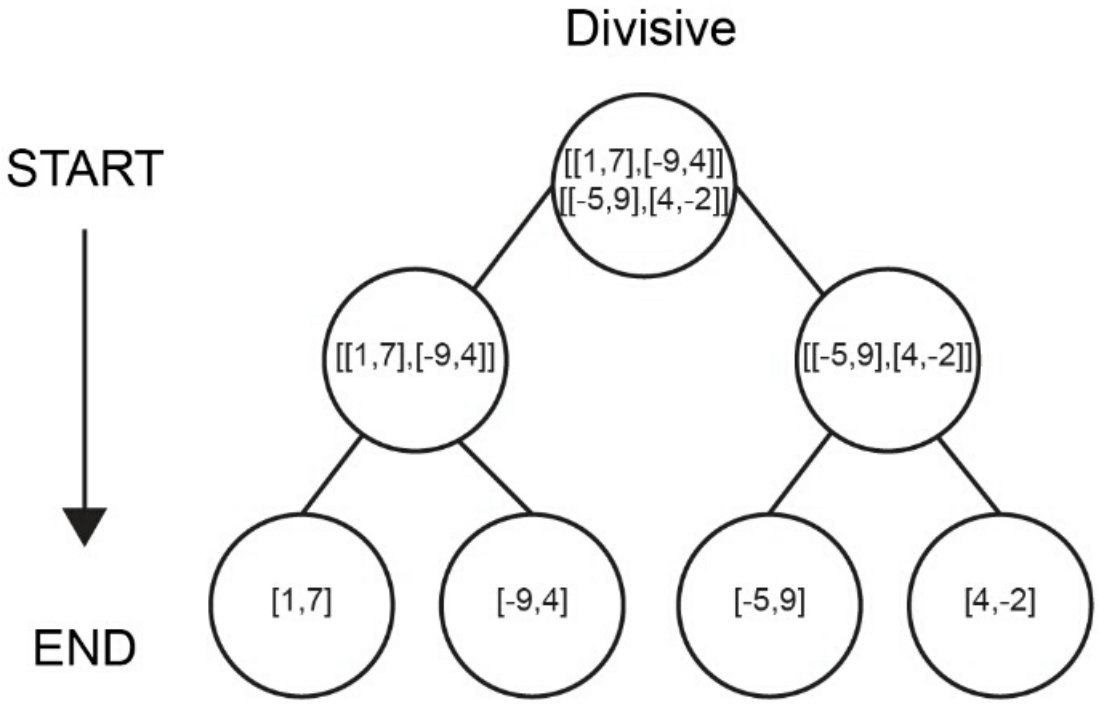
Q3. What are the different approaches of clustering?

A3. There are two Approaches of Clustering:

1) Divisive Approach
2) Agglomerative Approach

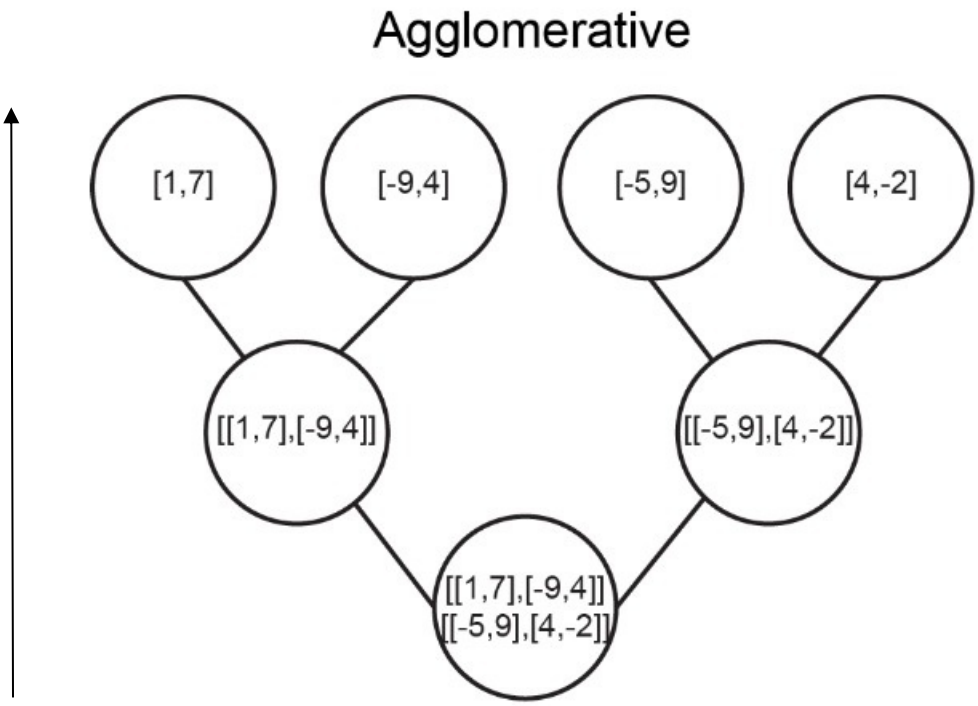
Q4. What is Divisive Approach? Explain with the help of proper Diagram.

A4. It first considers all the points to be part of one big cluster and in the subsequent steps tries to find out the points/ clusters which are least similar to each other and then breaks the bigger cluster into smaller ones. This continues until there are as many clusters as there are data points. This is also called the top-down approach.



Q5. What is Agglomerative Approach? Explain with the help of Proper Diagram.

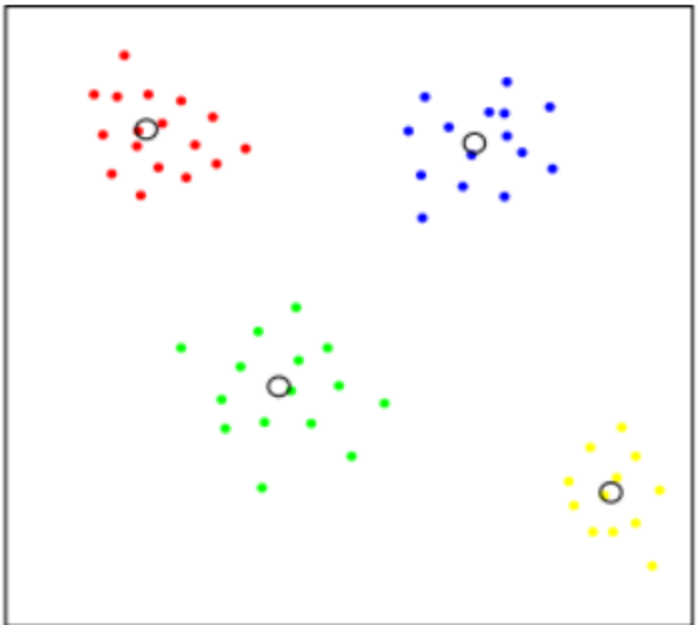
A5. This approach first considers all the points as individual clusters and then finds out the similarity between two points, puts them into a cluster. Then it goes on finding similar points and clusters until there is only one cluster left i.e., all points belong to a big cluster. This is also called the bottom-up approach.



Q6. What are the different types of Clustering Algorithms?

A6. The Clustering Algorithms are of many types. These are the some important clustering algorithms:

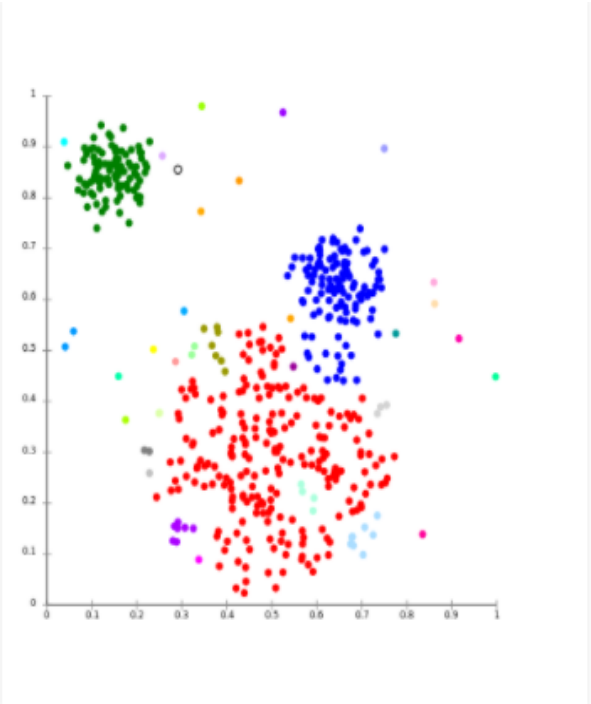
1) Centroid Based methods : This Algorithm is basically works on iterative Approach in which the clusters are formed by closeness of data points to Centroid of Cluster. Here Cluster is formed such that the distance of data point is minimum with the center.



Example : K Means Algorithm is one the popular example of Centroid based methods.

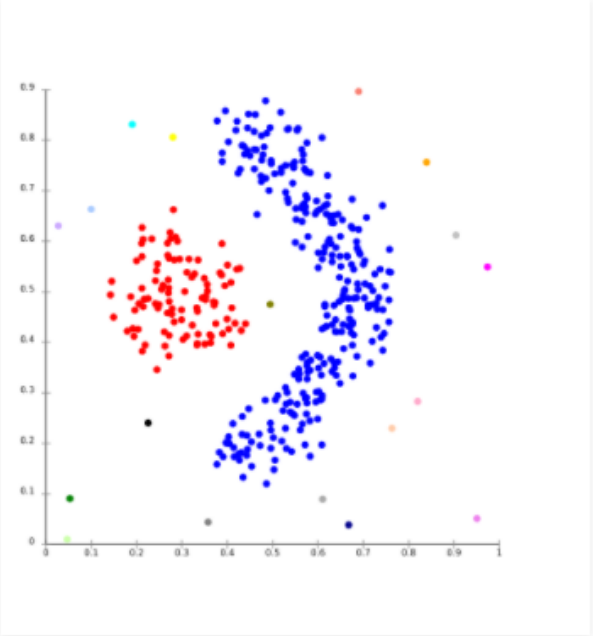
The main drawback of this algorithm is we have to specify the k value in advance.

2) Connectivity Based Methods : The main idea of Connectivity based methods is same as Centroid based methods which is basically defining the clusters on the basis of closeness of data points to Centroid. It provides hierarchy of clusters that merges each other at certain distances.



Example : Hierarchical Algorithms

3) Density Models : This type of density Models isolates various density regions based on different densities present in the data space .



For Example – DBSCAN

Let us understand all these Algorithms one by one.

Q7. What is K-Means Clustering?

A7. K- Means Clustering Algorithm is a Centroid based unsupervised learning algorithm which grouped unlabeled dataset into different clusters.

"It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties."

Q8. What is the significance of 'K' in K-Means?

A8. Here k defines the predefined number of clusters that needs to be created in the process. If k=3 there will be 3 numbers of clusters and if k=2 there will be 2 number of clusters and so on.

Q 9. What is the main goal of k means Algorithm?

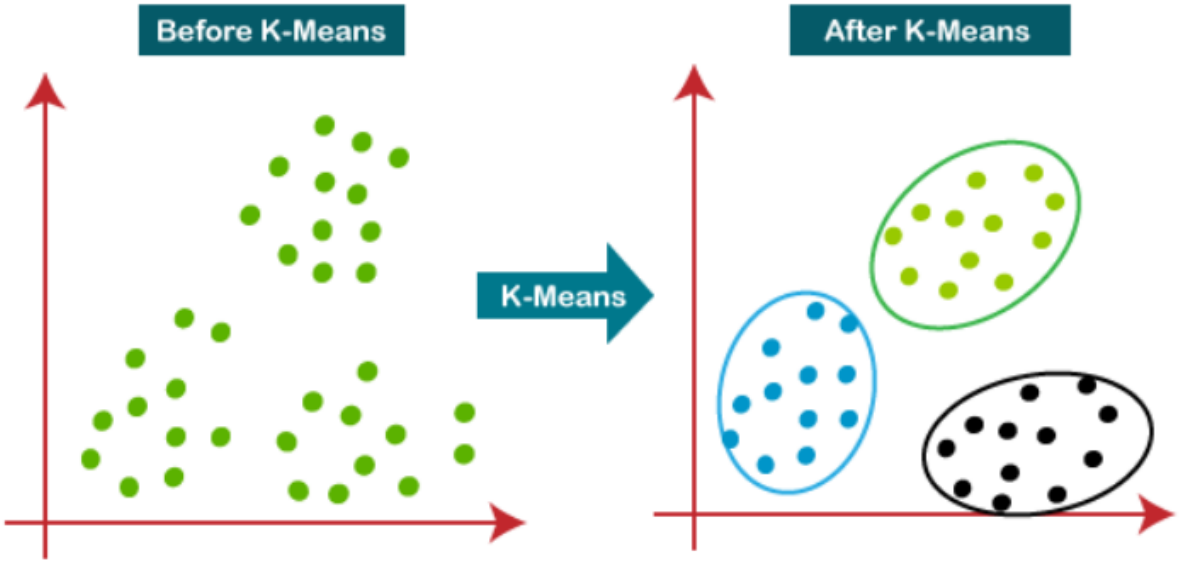
A9. K- means algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined(using WCSS) in this algorithm.

The k-means clustering algorithm mainly performs two tasks iteratively:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has data points with some similarity, and it is away from other clusters.

Given diagram explains the working of the K-means Clustering Algorithm:



Q10. How K-Means Algorithm does works?

A10. The working of k-means Algorithm is explained in the below steps:

Step 1: Our Very first step is to Select the value of k to decide the number of clusters.

Step2: Select Random K Centroids.

Step3: Measure the distance between Next point and Centroid.

Step4: Assign the point to nearest cluster.

Step5: Calculate the mean of each cluster as new centroid.

Step6 : Repeat 3-5 steps with the new centroid.

Step7: Adjusting the centroid for the newly formed cluster in step 4

Step8 : Repeating step 7 and 8 till all the data points are perfectly organized within a cluster space.

Q11. Explain the maths behind the K-Means Algorithm with the help of Simple Example.

A11. Let Suppose this is our Dataset and number of clusters=2

Height	Weight
185	72
170	56
168	60

179	68
182	72
188	77
180	71
183	70
180	84
180	88
177	67

Step1: Select the value of K to decide the number of clusters.
Which is K=2

Step2: Select Random K Centroids.
Let suppose here K1= (185, 72)
K2= (170, 56)

Step3 : Calculate the Euclidian distance between next data point (168,60) and clusters K1 and K2.

Distance for K1 (185, 72) and (168, 60)

$$ED = \sqrt{(168- 185)^2 + (60-72)^2}$$

$$ED = 20.80$$

Distance for K2 (170,56) and (168, 60)

$$ED = \sqrt{(168- 170)^2 + (60-56)^2}$$

$$ED = 4.48$$

Step4 : Assign the point to nearest cluster:

So, As we can see the distance between K2 Cluster and 3rd data point (168, 60) is minimum as compare to K1 Cluster. So, Our 3rd data point belongs to cluster 2 (K2).

Step5 : New Centroid Calculation (Calculate the mean of each cluster as new centroid) :

Just Because our 3rd data point belongs to K2. Our K2 will change and K1 remains same.

Our new Centroid (K2) will be :

$$\left(\frac{170+ 168}{2} , \frac{60+ 56}{2} \right)$$

K2 = (169 , 58)
K1 = (185, 72)

Step 6 : Again repeat Step 3– 5 with the new centroid K1 and K2.

After Doing all the these steps for all data points we will get to know which data point lies in which cluster.

Final Result :

K1 = { 1st , 4th ,5th ,6th ,7th, 8th ,9th ,11th ,12th ,10th } these data points lies in cluster 1.

K2= { 2nd, 4th } belongs to cluster 2.

Step 7 and 8 is based on WCSS.

Q12. What is WCSS ?

A12. WCSS stands for (Within-Cluster-Sum-of-Squares) . The number of clusters(K) that we Select for our model cannot be random. Each cluster is formed by calculating and comparing the distances of data points within a cluster to its centroid.

An ideal way to choose the right number of clusters would be to calculate the Within-Cluster-Sum-of-Squares (WCSS).
The theory discussed above can be mathematically expressed as:

- Let C1, C2, Ck be the K clusters
- Then we can write: $C1UC2UC3U...UCk$ ={1,2,3,...,n} i.e., each data point has been assigned to a cluster.
- Also,

$$C_k \cap C_{k'} = \emptyset \text{ for all } k \neq k'.$$

This means that the clusters are not overlapping.

- The idea behind the K-Means clustering approach is that the within-cluster variation amongst the point should be minimum. The within-cluster variance is denoted by: W(Ck).
- Hence, according to the above statement, we need to minimize this variance for all the clusters. Mathematically it can be written as:

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\}.$$

- The next step is to define the criterion for measuring the within-cluster variance. The criterion is the Euclidean distance between two data points.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

- The above formula says that we are calculating the distances between all the point in a cluster, then we are repeating it for all the K clusters (That's why there is two summation signs in the above formula) and then we are dividing it by the number of observation in the clusters (Ck is the number of observations in the Kth cluster) to calculate the average.

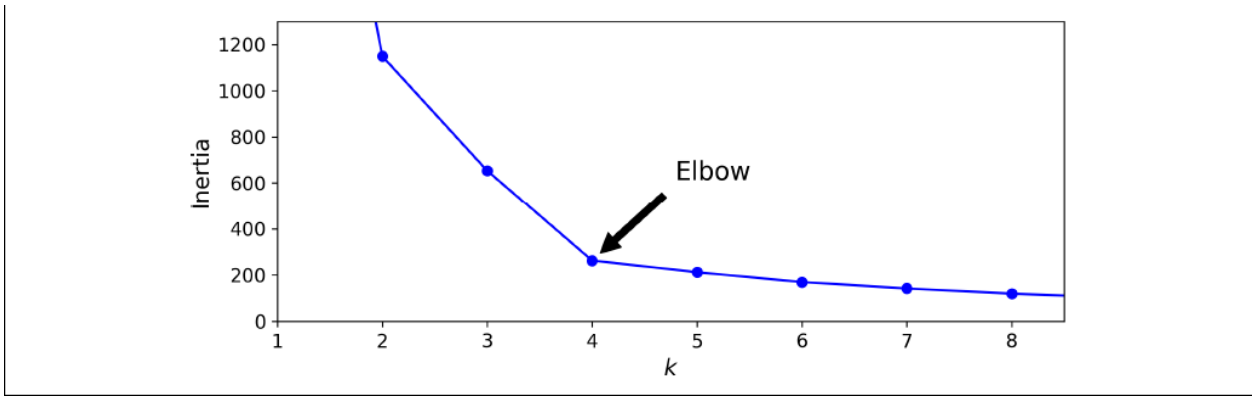
  So, ultimately our goal is to minimize:

$$\underset{C_1,\ldots,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}.$$

Q13. What is Elbow method and why do we use it ?

A13. This method is based on the relationship between the within-cluster sum of squared distances(WCSS) and the number of clusters. It is observed that first with an increase in the number of clusters WCSS decreases and then after a certain number of clusters the drop in WCSS is not that prominent.

The point after which the graph between WCSS and the number of clusters becomes comparatively smother is termed as the elbow(this is our number of clusters K value for our model) and the number of cluster at that point are the optimum number of clusters as even after increasing the clusters after that point the variation is not decreasing by much i.e., we have accounted for almost all the dissimilarity in the data. An elbow-curve looks like:



Q14. Example all these steps with python implementation.

A14. Step 1: Importing all necessary libraries and dataset

```
In [1]: # importing all necessary libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [4]: # Importing the Dataset
        dataset=pd.read_csv('Mall_Customers.csv')
```

```
In [5]: dataset.head()
```

Out[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

Step 2: Select Annual Income and spending score column for further calculation.

In [6]:
```
#dataset
X=dataset.iloc[:,3:]
X
```

Out[6]:

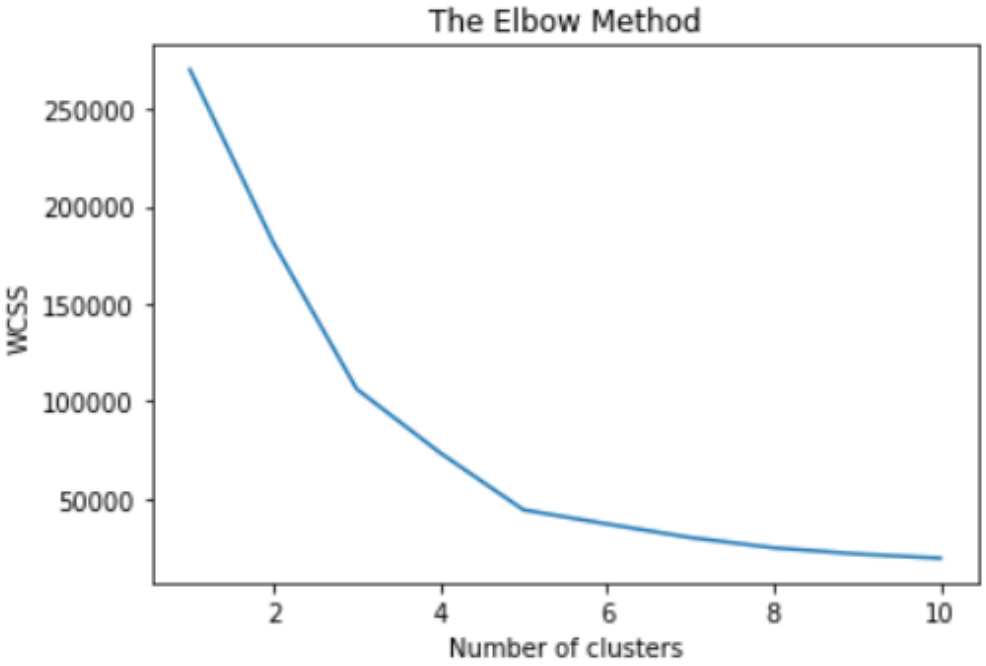
	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
...	...	...
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

200 rows × 2 columns

Step 3: Calculating right number of clusters by using elbow method

[7]:
```
#elbow method
from sklearn.cluster import KMeans
wcss=[]


for i in range (1,11):
    kmeans=KMeans(n_clusters=i,init='k-means++',random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)


plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

Elbow curve :



So, our k value is 5. Because after 5 the graph becomes smoother.

Step 4 : Fitting K-Means to the dataset

```
[8]: # Fitting K-Means to the dataset
     kmeans = KMeans(n_clusters = 5, init =
                     'k-means++', random_state = 42)
     y_kmeans = kmeans.fit_predict(X)
     print(y_kmeans)
```

```
[2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3
 3 2 3 2 3 2 0 2 3 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 0 4 1 4 0 4 1 4 1 4 0 4 1
 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4
 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4]
```

Step 5: It predicts the cluster number to which the datapoint belongs to

In [12]:
```
# It predicts the cluster number to which the datapoint belongs to
test=kmeans.predict(np.asarray([[3,3]]))
test[0]
```

Out[12]: 2

In [15]:
```
# Looking at the points which belong to Cluster0
X[y_kmeans==0]
```

Out[15]:

	Annual Income (k$)	Spending Score (1-100)
43	39	61
46	40	55
47	40	47
48	40	42
49	40	42

Step 6 : Visualizing the clusters

```
# Visualising the clusters
plt.scatter(X[y_kmeans == 0]['Annual Income (k$)'], X[y_kmeans == 0]
            ['Spending Score (1-100)'],s = 100,c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1]['Annual Income (k$)'], X[y_kmeans == 1]
            ['Spending Score (1-100)'],s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2]['Annual Income (k$)'], X[y_kmeans == 2]
            ['Spending Score (1-100)'],s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3]['Annual Income (k$)'], X[y_kmeans == 3]
            ['Spending Score (1-100)'],s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4]['Annual Income (k$)'], X[y_kmeans == 4]
            ['Spending Score (1-100)'],s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300,
            c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Output :

Code :

```
# importing all necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Importing the Dataset
dataset=pd.read_csv('Mall_Customers.csv')
dataset.head()

#dataset
X=dataset.iloc[:,3:]

#elbow method
from sklearn.cluster import KMeans
wcss=[]
for i in range (1,11):
   kmeans=KMeans(n_clusters=i, init='k-means++',random_state=42)
   kmeans.fit(X)
   wcss.append(kmeans.inertia_)
   plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
```

```
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Fitting K-Means to the dataset
kmeans = KMeans(n_clusters = 5, init =
            'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
print(y_kmeans)

# It predicts the cluster number to which the datapoint belongs to
test=kmeans.predict(np.asarray([[3,3]]))
test[0]

# Looking at the points which belong to Cluster0
X[y_kmeans==0]

# Visualizing the clusters
plt.scatter(X[y_kmeans == 0]['Annual Income (k$)'], X[y_kmeans == 0]
        ['Spending Score (1-100)'],s = 100,c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1]['Annual Income (k$)'], X[y_kmeans == 1]
        ['Spending Score (1-100)'],s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2]['Annual Income (k$)'], X[y_kmeans == 2]
        ['Spending Score (1-100)'],s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3]['Annual Income (k$)'], X[y_kmeans == 3]
        ['Spending Score (1-100)'],s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_kmeans == 4]['Annual Income (k$)'], X[y_kmeans == 4]
        ['Spending Score (1-100)'],s = 100, c = 'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
300, c = 'yellow', label = 'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

Q15. What are the Challenges of K-Means Algorithm?

A15. These are the basic challenges we can face, while working with K-Means Algorithm.

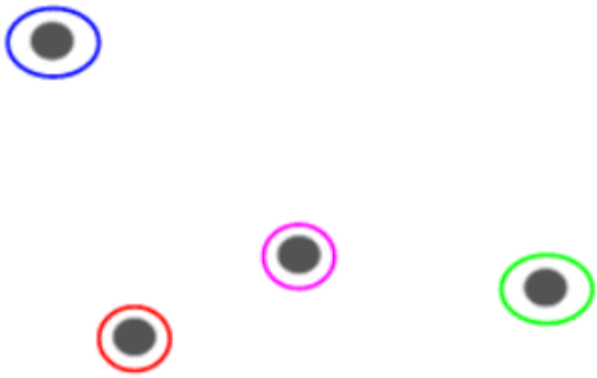
- We need to specify the number of clusters beforehand.
- It is required to run the algorithm multiple times to avoid a sub-optimal solution
- K-Means does not behave very well when the clusters have varying sizes, different densities, or non-spherical shapes.

Hierarchical Clustering Algorithm

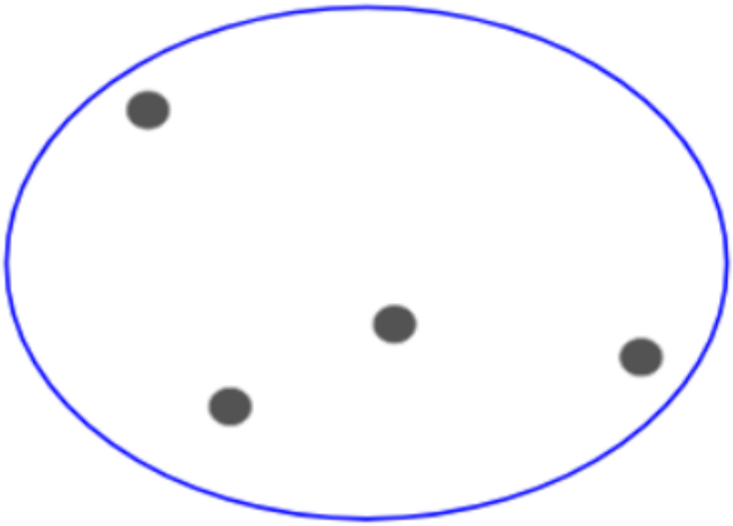
Q16. What is Hierarchical Clustering ?

A16. Hierarchical Clustering is the most popular and widely used clustering Algorithm. A Hierarchical Clustering works via grouping a data into tree of clusters.
Let 's understand Hierarchical clustering with an example:
Let's say we have a different – different data points we have to cluster them :

Now we will assign each data point as a separate cluster :



Now, Based on Similarity(minimum distance) of these clusters we will combine the most similar clusters together and repeat this process until only a single cluster is left:

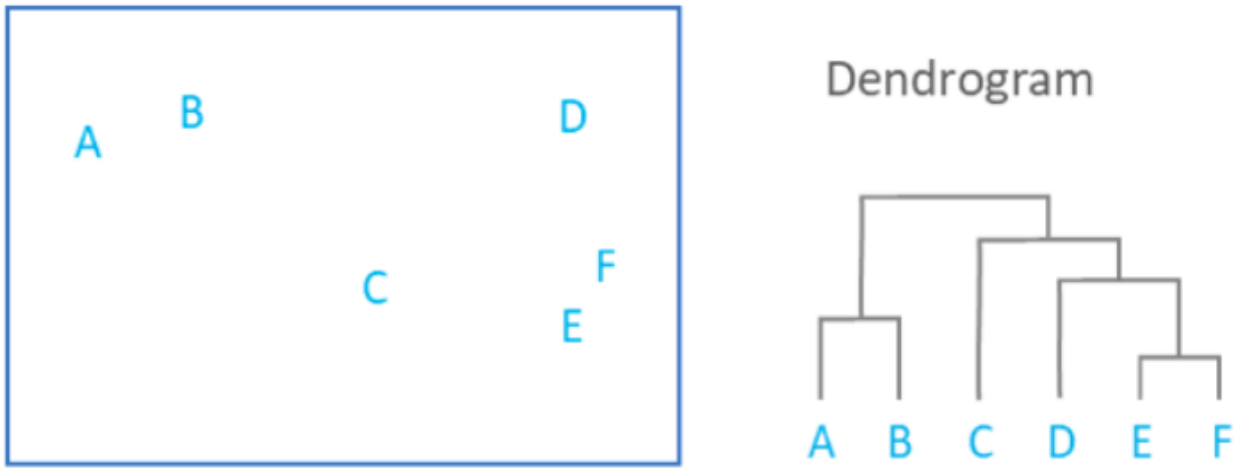


Here , we will build the hierarchy of the clusters That's why this Algorithm is called Hierarchical Clustering Algorithm.

Q17. What is Dendrogram and why do we use it?

A17. Dendrogram is a Diagram that shows the hierarchical relationship between the objects or data points. The main use of Dendrogram is to allocate the objects to clusters. The Dendrogram is mainly created as an output from hierarchical clustering.

We can understand the Dendrogram by using the below diagram:



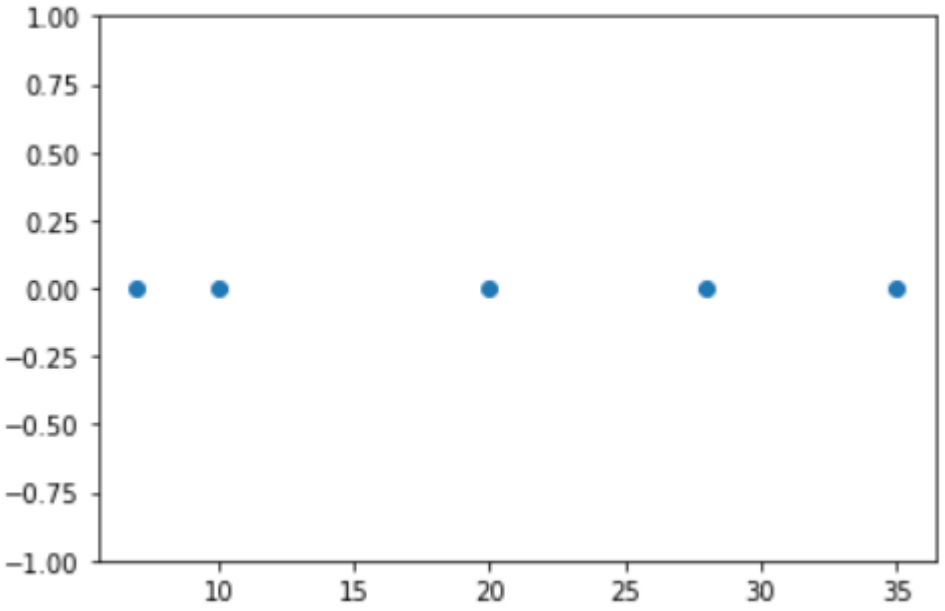
As we can see in the above diagram E and F are most similar(because the distance between E and F is minimum) as the height of the link that joins them together is the smallest. The next two most similar objects are A and B.
" Dendrogram cannot tell you that how many clusters you should have"
What mistake do we make while reading the Dendrogram that we think that by reading the Dendrogram, we will know the number of clusters. Which is not correct in every case.
In the example above, the incorrect interpretation is that the dendrogram shows that there are two clusters, as the distance between the clusters are highest between two and three clusters. Which is not true.

Q18. Explain Different types of Linkage methods with the help of proper Example .

A18 . The linkage methods work by calculating the distance (similarities) between all data points. Then the closest pair of clusters are combined into a single cluster.
We can use different types of linkage methods for finding the similarity between the data points:

Let us understand these linkage methods with the help of Example :

Example : For the given data points { 7, 10, 20, 28, 35 } perform hierarchical clustering and also plot the dendrogram.
Let's first visualize the data points :



From the above diagram we can conclude that :

- The first two data points 7 and 10 are closer to each other and they should be in same cluster.
- Then 25 and 35 are closer to each other and should be in same cluster.
- But the Cluster of data point 20 is not easy to conclude as it lies in center.

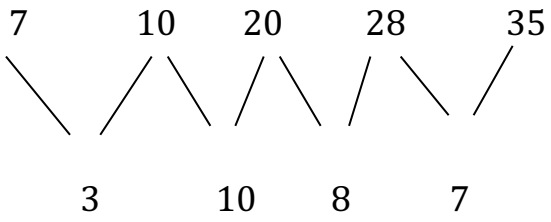
Lets solve the problem using different types of Linkage methods:

1) Single Linkage : In Single Linkage Hierarchical Clustering we will merge 2 clusters (whose two data points have the smallest distance) in each step.

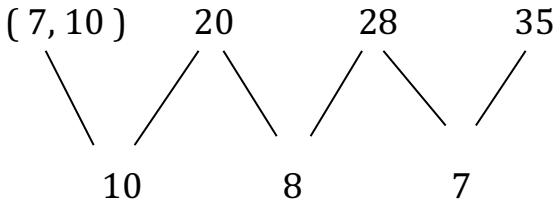
- cluster distance = smallest pairwise distance
- highly sensitive to outliers when forming flat clusters

- works well for low-noise data with an unusual structure

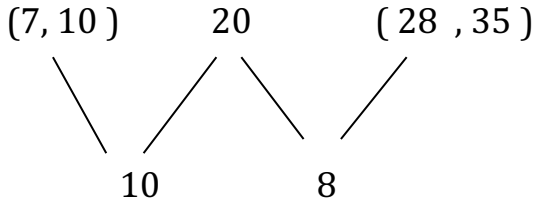
Step 1:



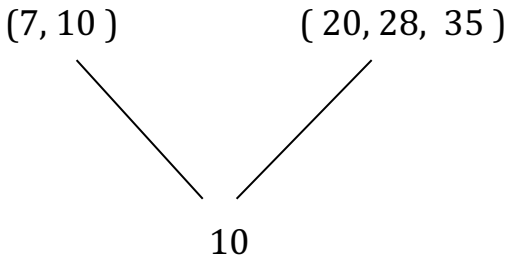
Step 2:



Step 3:



Step 4 :



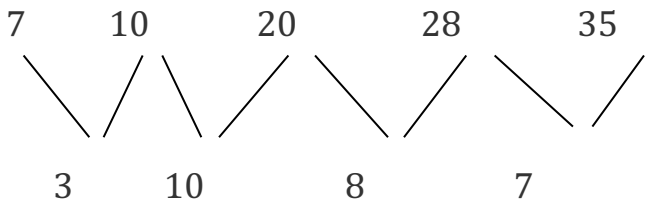
Using Single Linkage 2 Clusters are Formed :
Cluster 1 : (7, 10)
Cluster 2 : (20, 28 , 35)

2) Complete Linkage : In Complete Linkage Hierarchical clustering will merge in the data points of the clusters in each step, which provide the smallest maximum pairwise distance.

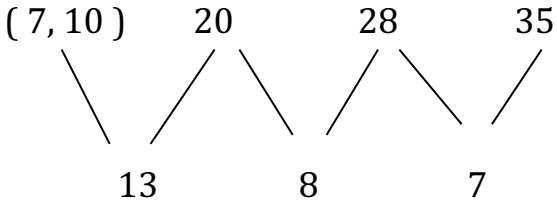
- cluster distance is the largest distance between any point in cluster 1 and any point in cluster 2
- less sensitive to outliers than single linkage
- cluster distance = largest pairwise distance

Let's understand the above example with the help of Complete Linkage Hierarchical Clustering:

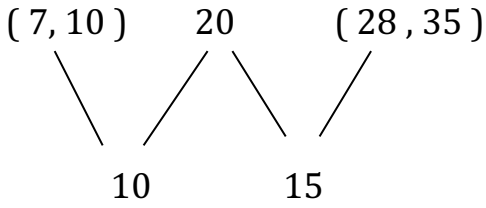
Step 1:



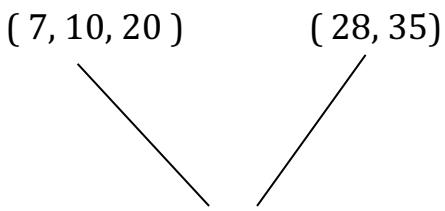
Step 2:



Step 3:



Step 4:



Using Complete Hierarchical Clustering 2 clusters are formed :

Cluster 1: (7, 10, 20)

Cluster 2: (28, 35)

Q 19. What is the Algorithm of Hierarchical Clustering ?

A19. **The Algorithm:**

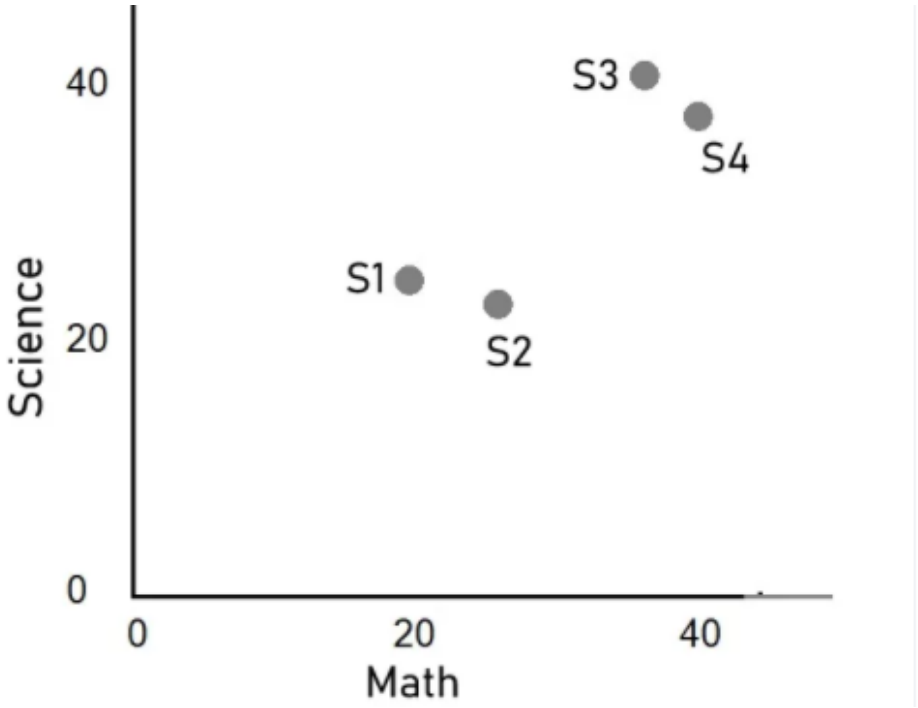
1. Begin with n observations and a measure (such as Euclidean distance) of all the n(n−1)/2 pairwise dissimilarities(or the Euclidean distances generally). Treat each observation as its own cluster. Initially, we have n clusters.
2. Compare all the distances and put the two closest points/clusters in the same cluster. The dissimilarity(or the Euclidean distances) between these two clusters indicates the height in the dendrogram at which the fusion line should be placed.
3. Compute the new pairwise inter-cluster dissimilarities(or the Euclidean distances) among the remaining clusters.
4. Repeat steps 2 and 3 till we have only one cluster left.

Q 20. Explain the maths behind the hierarchical clustering with the help of proper example .

A20. Lets visualize how hierarchical clustering works:
Let suppose we have a data of marks obtained by 4 students in Maths and Science and we have to create the clusters of the data to draw insights.

No.	Maths	Science
S1	20	25
S2	25	22
S3	35	40

S4	40	35



Step 1: Our first step is to calculate the distance between each data points and draw the distance matrix.

We will use Euclidean distance for this example :

$$d_{L2}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Euclidean Distance

Distance Matrix :

	S1	S2	S3	S4
S1	0	5.83	21.21	22.36
S2	5.83	0	20.59	19.84
S3	21.21	20.59	0	7.07
S4	22.36	19.84	7.07	0

Forming the cluster between S1 and S2 . Because the distance between S1 and S2 is minimum.

Step 2: Now a question arises , what does our data look like now ?

So, Now we took the average of the marks of Students S1 and S2 and the value we get will represent the marks f this cluster.

Data after first Cluster

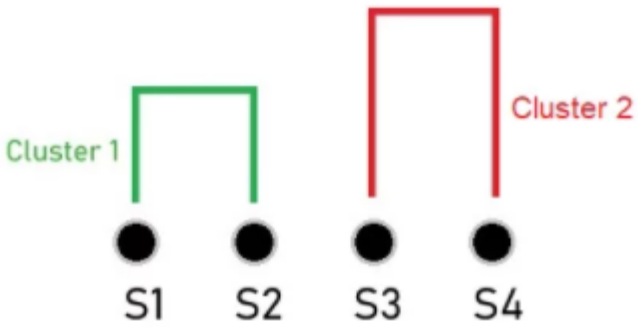
	Maths	Science
[S1, S2]	22.5	23.5
S3	35	40
S4	40	35

Distance Matrix

	[S1,S2]	S3	S4
[S1,S2]	0	20.7	20.9
S3	20.7	0	7.07
S4	20.9	7.07	0

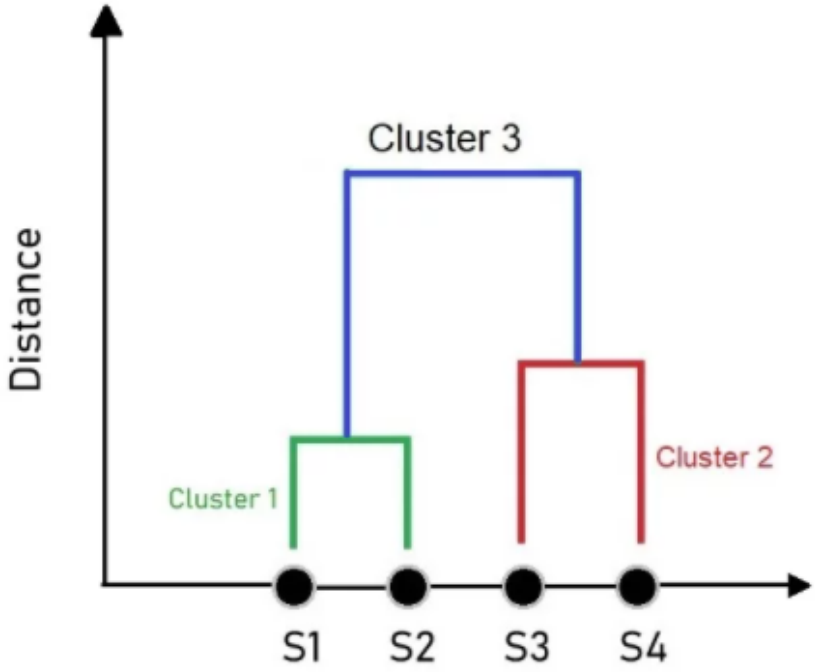
Proximity Matrix

Again We can in the above cluster the closest points are S3 and S4 and Create new cluster.



Clustering between S3 and S4

Step 3: If we repeat the steps above and keep on clustering until we are left with just one cluster containing all clusters.



Our final Dendrogram.

Q 21. Explain the Hierarchical clustering with the help of its python Implementation.

A 21.

Step 1: Importing the necessary libraries
scipy is use for calculating the distance between clusters

```
# Importing all necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
```

Step 2: Loading the dataset and selecting certain features based on which we Will perform clustering.

```
# getting the data ready
data= load_iris()
df= data.data

# Selecting certain features
#based on which clustering is done
df = df[:,1:3]
df
```

Output :

```
array([[3.5, 1.4],
       [3. , 1.4],
       [3.2, 1.3],
       [3.1, 1.5],
       [3.6, 1.4],
       [3.9, 1.7],
       [3.4, 1.4],
       [3.4, 1.5],
       [2.9, 1.4],
       [3.1, 1.5],
       [3.7, 1.5],
       [3.4, 1.6],
       [3. , 1.4],
       [3. , 1.1],
       [4. , 1.2],
       [4.4, 1.5],
       [3.9, 1.3],
       [3.5, 1.4],
       [3.8, 1.7],
```

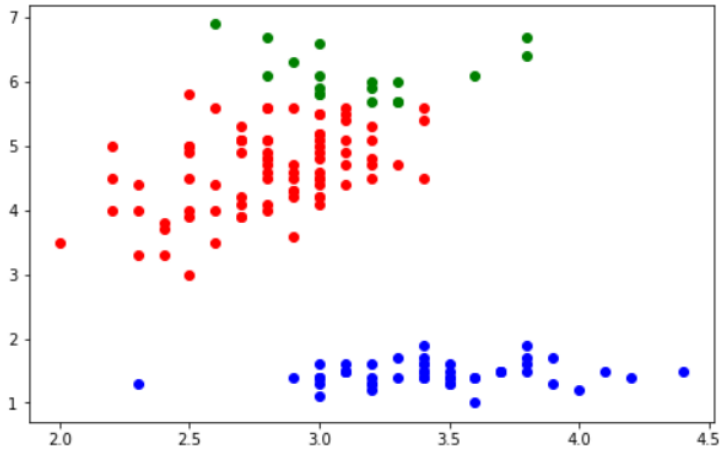
Step 3: Now, we will create the model and there are 4 number of clusters.

```
#Creating the model

agg_clustering = AgglomerativeClustering(n_clusters = 3,
                  affinity = 'euclidean', linkage = 'complete')

#predicting the labels

labels = agg_clustering.fit_predict(df)

# Number of classes

labels
```

```
array([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, 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, 2, 0, 2, 0, 2, 2, 0, 2, 0, 2,
       0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 0, 2, 2, 2,
       0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0, 0], dtype=int64)
```

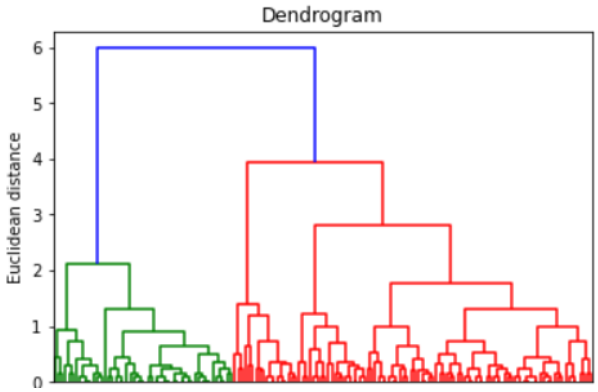
Step4: plotting the results

In [40]:
```
#Plotting the results

plt.figure(figsize = (8,5))
plt.scatter(df[labels == 0 , 0] , df[labels == 0 , 1] , c = 'red')
plt.scatter(df[labels == 1 , 0] , df[labels == 1 , 1] , c = 'blue')
plt.scatter(df[labels == 2 , 0] , df[labels == 2 , 1] , c = 'green')
plt.show()
```



Step 5: Dendrogram of the clusters

In [43]:
```
from scipy.cluster.hierarchy import dendrogram , linkage

#Linkage Matrix
Z = linkage(df, method = 'complete')

#plotting dendrogram
dendro = dendrogram(Z)
plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()
```



Code:

```
# Importing all necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering

# getting the data ready
data= load_iris()
df= data.data

# Selecting certain features
#based on which clustering is done
df = df[:,1:3]
df

#Creating the model
 agg_clustering = AgglomerativeClustering(n_clusters = 3,
                affinity = 'euclidean', linkage = 'complete')

#predicting the labels
 labels = agg_clustering.fit_predict(df)

# Number of clusters
labels

#Plotting the results
 plt.figure(figsize = (8,5))
plt.scatter(df[labels == 0 , 0] , df[labels == 0 , 1] , c = 'red')
plt.scatter(df[labels == 1 , 0] , df[labels == 1 , 1] , c = 'blue')
plt.scatter(df[labels == 2 , 0] , df[labels == 2 , 1] , c = 'green')
plt.show()

#  plotting the hierarchy of the clusters
```

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

#Linkage Matrix
Z = linkage(df, method = 'complete')

#plotting dendrogram
dendro = dendrogram(Z)
plt.title('Dendrogram')
plt.ylabel('Euclidean distance')
plt.show()
```

Q22. What are the pros and cons of Hierarchical Clustering?

A22. These are the pros and cons of Hierarchical Clustering:

Pros :

- Easy to implement and gives best result in some cases.
- No a prior information about the number of clusters required.

Cons :

- Algorithm can never undo what has done previously.
- Based on the type of distance matrix chosen for merging different algorithms can suffer with one or more of the following :
    - Sensitivity to noise and outliers
    - Breaking large clusters
    - Difficulty on handling different sized clusters and convex shapes
- Sometimes it is difficult to identify the correct number of clusters by the dendrogram.

DBSCAN Algorithm

Q23. What is DBSCAN Algorithm?

A23. DBSCAN stands for density based Spatial Clustering of Applications with noise. DBSCAN is a Clustering Algorithm which is based on the density. Here density shows that how many number of data points which are located in a given area.

Q24. What are the different parameters of DBSCAN Algorithm?

A24. These are the basic parameters of DBSCAN Algorithm :

Epsilon (eps) : Epsilon is radius of the circle. Basically epsilon specifies how close the data points should be to each other to be considered a part of cluster. If the distance between two data points is less than or equal to epsilon value the data points are considered as a neighbors.

Min_points(n) : The minimum number of data points to form a dense region.

Core points : Data points that has atleast Min_points (n) within epsilon distance. If the number of points inside the *eps radius* of a point is greater than or equal to the Min_points then it's called a core point.

Border Points: If the number of points inside the *eps radius* of a point is less than the Min_points and it lies within the *eps radius* region of a core point, it's called a border point.

Noise: A point which is neither a core nor a border point is a noise point.

Q 25. Explain the steps of DBSCAN Clustering Algorithm.

A25. Algorithm Steps:

1. The algorithm starts with a random point in the dataset which has not been visited yet and its neighboring points are identified based on the eps value.
2. If the point contains greater than or equal points than the min_pts, then the cluster formation starts and this point becomes a core point, else it's considered as noise. The thing to note here is that a point initially classified as noise can later become a border point if it's in the eps radius of a core point.

3. If the point is a core point, then all its neighbours become a part of the cluster. If the points in the neighbourhood turn out to be core points then their neighbours are also part of the cluster.
4. Repeat the steps above until all points are classified into different clusters or noises.

This algorithm works well if all the clusters are dense enough, and they are well separated by low-density regions.

Q 26. Write the python code of DBSCAN Algorithm.

A26. Step 1: Importing Necessary Libraries

```
# Necessary Imports

import numpy as np

from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
```

```

```

Step 2: Data Creation

```
# Data creation
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750,
                            centers=centers,
                            cluster_std=0.4,
                            random_state=0)
# generate sample blobs

X = StandardScaler().fit_transform(X)
```

Step 3: Lets Apply algorithm to the dataset and also check the number of clusters.

```
# DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)

# we are calculating these for showcasing in diagram
# creating an array of true and false as the same size as db.labels
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)

# setting the indices of the core regions to True
core_samples_mask[db.core_sample_indices_] = True


# similar to the model.fit() method, it gives the labels of the clustered data
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
# the label -1 is considered as noise by the DBSCAN algorithm
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# calculating the number of clusters
n_noise_ = list(labels).count(-1)
```

Step 4 : Now we will check that how many noise points , and how many clusters our dataset contains.

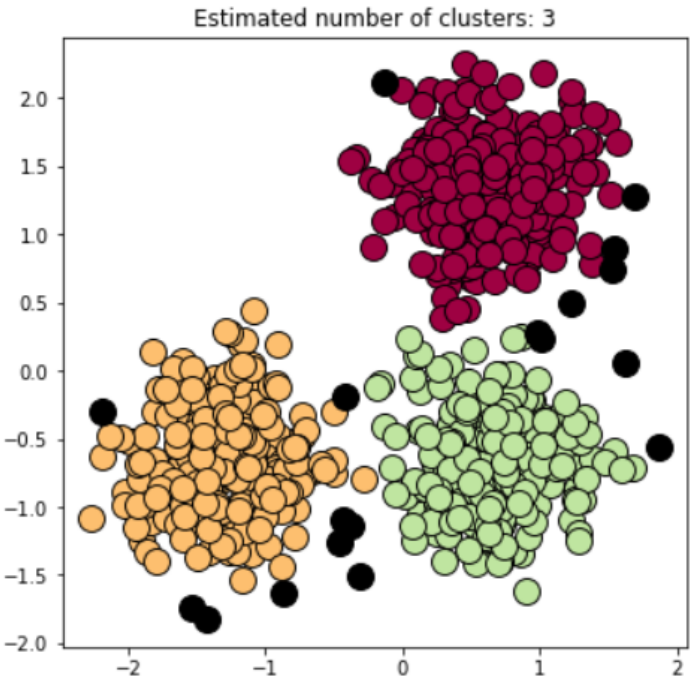
```
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)

"""Homogeneity metric of a cluster labeling given a ground truth.

A clustering result satisfies homogeneity if all of its clusters
contain only data points which are members of a single class."""

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
```

```
Estimated number of clusters: 3
Estimated number of noise points: 18
Homogeneity: 0.953
```

Step 5: Lets Visualize our result

```
# Plot result
import matplotlib.pyplot as plt

# Black  is used for noise.
unique_labels = set(labels)  # identifying all the unique labels/clusters
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]  # creating the list of colours, generating the colourmap



for k, col in zip(unique_labels, colors):


    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)   # assigning class members for each class


    xy = X[class_member_mask & core_samples_mask] # creating the list of points for each class

    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)


    xy = X[class_member_mask & ~core_samples_mask] # creating the list of noise points

    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)



plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

Output :



In short, DBSCAN is a very simple yet powerful algorithm, capable of identifying any number of clusters, of any shape, it is robust to outliers, and it has just two hyper parameters(eps and min_samples). However, if the density varies significantly across the clusters, it can be impossible for it to capture all the clusters properly.

Code :

```
# Necessary Imports

import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

# Data creation
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                            random_state=0)
```

```
# generate sample blobs
X = StandardScaler().fit_transform(X)

# DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)

# we are calculating these for showcasing in diagram
# creating an array of true and false as the same size as db.labels
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)

# setting the indices of the core regions to True
core_samples_mask[db.core_sample_indices_] = True


# similar to the model.fit() method, it gives the labels of the clustered data
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
# the label -1 is considered as noise by the DBSCAN algorithm
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# calculating the number of clusters
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)

"""Homogeneity metric of a cluster labeling given a ground truth.
A clustering result satisfies homogeneity if all of its clusters
contain only data points which are members of a single class."""

print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true,
labels))
```

Q 27. What are the main requirements that should be met by a clustering Algorithm ?

A27.

The primary requirements that should be met by a clustering algorithm are:

- It should be scalable
- It should be able to deal with attributes of different types;
- It should be able to discover arbitrary shape clusters;
- It should have an inbuilt ability to deal with noise and outliers;
- The clusters should not vary with the order of input records;
- It should be able to handle data of high dimensions.
- It should be easy to interpret and use.

Q 28. What are the various Applications of Clustering in real world?

A28. These are the various applications of clustering :

- **For customer segmentation:** You can cluster your customers based on their purchases,their activity on your website, and so on. This is useful to understand who your customers are and what they need, so you can adapt your products and marketing campaigns to each segment. For example, this can be useful in recommender systems to suggest content that other users in the same cluster enjoyed.
- **For data analysis:** When analyzing a new dataset, it is often useful to first discover clusters of similar instances, as it is often easier to analyze clusters separately.
- **As a dimensionality reduction technique:** Once a dataset has been clustered, it is usually possible to measure each instance's affinity with each cluster (affinity is any measure of how well an instance fits into a cluster). Each instance's feature vector x can then be replaced with the vector of its cluster affinities. If there are k clusters, then this vector is k dimensional. This is typically much lower dimensional than the original feature vector, but it can preserve enough information for further processing.
- **For anomaly detection (also called outlier detection):** Any instance that has a low affinity to all the clusters is likely to be an anomaly. For example, if you have clustered the users of your website based on their behavior, you can detect users with unusual behavior, such as an unusual number of requests per second, and so on. Anomaly detection is particularly useful in detecting defects in manufacturing, or for fraud detection.

- **For semi-supervised learning:** If you only have a few labels, you could perform clustering and propagate the labels to all the instances in the same cluster. This can greatly increase the amount of labels available for a subsequent supervised learning algorithm, and thus improve its performance. .
- **For search engines:** For example, some search engines let you search for images that are similar to a reference image. To build such a system, you would first apply a clustering algorithm to all the images in your database: similar images would end up in the same cluster. Then when a user provides a reference image, all you need to do is to find this image's cluster using the trained clustering model, and you can then simply return all the images from this cluster.
- **To segment an image:** By clustering pixels according to their color, then replacing each pixel's color with the mean color of its cluster, it is possible to reduce the number of different colors in the image considerably. This technique is used in many object detection and tracking systems, as it makes it easier to detect the contour of each object.

Q 29. How to Evaluate the performance of Clustering Algorithms ?

A29. There are lot of ways to evaluate the performance of Clustering Algorithms. Here we will see rand Index and jaccard Coefficient.

A good cluster will have:

- High inter-class similarity, and
- Low intraclass similarity

Based on these points we will evaluating the performance of Clustering Algorithms.

N: Number of objects in the data
P: $P1,2,\ldots,Pm$ the set of ground truth
clusters C: $C1,C2,\ldots Cn$ the set of clusters formed by the algorithm

The Incidence Matrix: N* N matrix
1) Pij=1 if the two points Oi and Oj belong to the same cluster in the ground truth else Pij=0.

2) Cij=1 if the two points Oi and Oj belong to the same cluster in the ground truth else Cij=0

Now there can be the following scenarios:

1. Cij=Pij=1--> both the points belong to the same cluster for both our algorithm and ground truth(Agree)--- **SS**
2. Cij=Pij=0--> both the points don't belong to the same cluster for both our algorithm and ground truth(Agree)--- **DD**
3. Cij=1 $butPij$=0--> The points belong in the same cluster for our algorithm but in different clusters for the ground truth (Disagree)---- **SD**
4. Cij=0 $but\ Pij$= 1 --> The points don't belong in the same cluster for our algorithm but in same clusters for the ground truth (Disagree)----**DS**

Rand Index= $\frac{Total\,Agree}{Total\,Disagree} = \frac{(SS+DD)}{(SS+DD+DS+SD)}$

The disadvantage of this is that it could be dominated by DD.

Jaccard Coeeficient= $\frac{SS}{(SS+SD+DS)}$

```
print("Rand Index: %0.3f"
% metrics.adjusted_rand_score(labels_true, labels))
```

```
Rand Index: 0.952
```

```
metrics.jaccard_similarity_score(labels_true, labels)
```

```
0.972
```

A higher value of Rand Index and Jaccard's coefficient mean that the clusters generated by our algorithm mostly agree to the ground truth.