**EXPERIMENT NO: 1**

**NAME OF EXPERIMENT**: Implementation of Apriori algorithm for finding frequent Item set using Python.

**INTRODUCTION:**

The Apriori algorithm is a data mining technique used to find frequent item sets and association rules in large datasets. It is based on the observation that if an item set is frequent, then all of its subsets must also be frequent. The algorithm works by first finding all frequent individual items, then generating larger item sets by joining smaller ones, and finally finding all frequent item sets.

Support (S): Support of association rule X =>Y is the percentage of transactions in dataset that contain both items (X & Y).

Confidence (C): Confidence of association rule X =>Y is the percentage of transactions containing X that also contains Y.

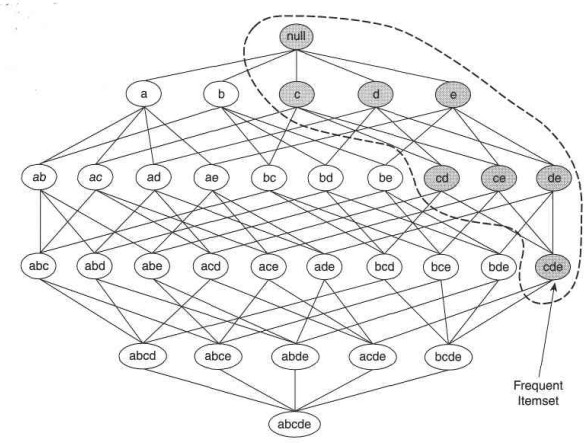
Lift: It measures how many times more often X and Y occur together then expected if they were statically independent.

Fig: Apriori algorithm

STEPS:

Step 1: Apriori employs an iterative approach known as level-wise search, where

frequent K item sets are used to explore frequent K + 1 item sets.

Step 2: First, the set of frequent 1-itemset that satisfy minimum support is found by scanning the database. The resulting set is denoted by L1.

Step 3: Next, L1 is used to find L2 and L3 and so on until no more frequent K item sets can be found.

Step 4: The finding of each Lk requires one full scan of the database. Output all frequent item sets of size k(Lk) and stop.

Use APRIORI algorithm to generate strong association rules from the following transaction database. Use min\_sup=50% and min\_confidence=75%.

|  |  |
| --- | --- |
| Transaction id | Item Set |
| 1 | Bread, Egg, Juice, Cheese |
| 2 | Bread, , Juice, Cheese |
| 3 | Bread, Milk, Yoghurt, Cheese |
| 4 | Cheese, Juice, Milk, |
| 5 | Bread, Cheese, Egg |

→ Solution,

Step1: Find out the frequency of each Item of the item set L1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item id | Item Set | Frequency | Support | % |
| 1 | Bread | 4 | 4/5 | 80% |
| 2 | Cheese | 5 | 5/5 | 100% |
| 3 | Egg | 2 | 2/5 | 40% |
| 4 | Juice | 3 | 3/5 | 60% |
| 5 | Milk | 2 | 2/5 | 40% |
| 6 | Yoghurt | 1 | 1/5 | 20% |

Here, the items egg, milk, yogurt didn’t cross the minimum criteria of support=50%. So, discard those items.

Step2: Take 2-item set L2 and calculate support count.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SN | Pair Item Set | Frequency | Support | % |
| 1 | (Bread, Cheese) | 4 | 4/5 | 80% |
| 2 | (Bread, Egg) | 2 | 2/5 | 40% ❌ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | (Bread, Juice) | 2 | 2/5 | 40% ❌ |
| 4 | (Bread, Milk) | 1 | 1/5 | 20% ❌ |
| 5 | (Bread, Yoghurt) | 1 | 1/5 | 20% ❌ |
| 6 | (Cheese, Egg) | 2 | 2/5 | 40% ❌ |
| 7 | (Cheese, Juice) | 3 | 3/5 | 60% |
| 8 | (Cheese, Milk) | 2 | 2/5 | 40% ❌ |
| 9 | (Cheese, Yoghurt) | 1 | 1/5 | 20% ❌ |
| 10 | (Egg, Juice) | 1 | 1/5 | 20% ❌ |
| 11 | (Juice, Milk) | 1 | 1/5 | 20% ❌ |
| 12 | (Milk, Yoghurt) | 1 | 1/5 | 20% ❌ |

Here, the pair items {Bread, Cheese} and {Cheese, Bread} only meet the minimum criteria of support=50% so add those item set and remove the others.

Step3: Take 3–itemset and calculate the Support count.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SN | 3 pair Item Set | Frequency | Support | % |
| 1 | (Bread, Cheese, Egg) | 2 | 2/5 | 40% ❌ |
| 2 | (Bread, Cheese, Juice) | 2 | 2/5 | 40% ❌ |
| 3 | (Cheese, Egg, Juice) | 1 | 1/5 | 20% ❌ |

None of the above item set crosses the minimum criteria of support. Step 3 is discarded and finding out the confidence according to the pair item set from step 2.

Step4: Finding out the confidence from the item sets of step2.

1. (Bread and Cheese)
   1. (Bread → Cheese)

= 𝑠𝑢𝑝𝑝𝑜𝑟𝑡( 𝐵𝑟𝑒𝑎𝑑 𝖴 𝐶ℎ𝑒𝑒𝑠𝑒)/ 𝑠𝑢𝑝𝑝𝑜𝑟𝑡 (𝐵𝑟𝑒𝑎𝑑)

= (⅘) / (⅘)

= 100%

* 1. (Cheese → Bread)

= 𝑠𝑢𝑝𝑝𝑜𝑟𝑡( 𝐶ℎ𝑒𝑒𝑠𝑒 𝖴 𝐵𝑟𝑒𝑎𝑑)/ 𝑠𝑢𝑝𝑝𝑜𝑟𝑡 (𝐶ℎ𝑒𝑒𝑠𝑒)

= (⅘) / (5/5)

= 80%

1. (Cheese and Juice)
   1. (Cheese → Juice)

= 𝑠𝑢𝑝𝑝𝑜𝑟𝑡( 𝐶ℎ𝑒𝑒𝑠𝑒 𝖴 𝐽𝑢𝑖𝑐𝑒)/ 𝑠𝑢𝑝𝑝𝑜𝑟𝑡 (𝐶ℎ𝑒𝑒𝑠𝑒)

= (⅗) /(5/5)

= 60%

* 1. (Juice → Cheese)

= 𝑠𝑢𝑝𝑝𝑜𝑟𝑡( 𝐽𝑢𝑖𝑐𝑒 𝖴 𝐶ℎ𝑒𝑒𝑠𝑒)/ 𝑠𝑢𝑝𝑝𝑜𝑟𝑡 (𝐽𝑢𝑖𝑐𝑒)

= (⅗) / (⅗)

= 100%

Here the associations rule of {Bread, Cheese} and {Juice, Cheese} is strong since they passes the minimum confidence threshold=75%.So output strong association rule is {Bread → Cheese} and {Juice → Cheese}.

ADVANTAGES OF APRIORI ALGORITHM:

1. Easy to understand.

2. Least memory consumption.

3. Easy implementation.

4. It uses Apriori property for pruning. Therefore, item-sets left for further support checking remains less.

LIMITATIONS OF APRIORI ALGORITHM:

1. It works slow compared to other algorithm.

2. It needs to generate a huge number of candidate’s sets if database is huge.

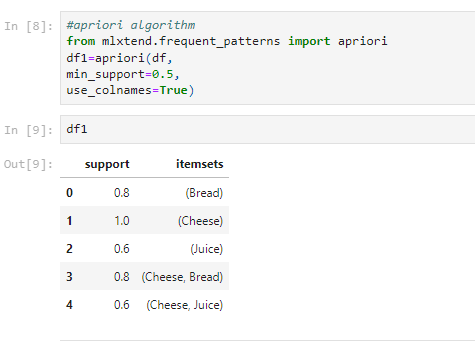
3. It needs to repeatedly scan the whole dataset and check a large set of candidates by pattern matching.

PROGRAM IMPLEMENTATION IN PYTHON:

Requirement: Anaconda Navigator

Source Code:





**CONCLUSION:** Hence, we have successfully implemented the Apriori algorithm in Python to find frequent item set in a given dataset.

**EXPERIMENT NO: 2**

**NAME OF EXPERIMENT**: Implementation FP-growth algorithm for finding frequent item set using Python.

**INTRODUCTION**: The FP-Growth algorithm is a data mining algorithm used for discovering frequent item sets in a transactional database. FP growth algorithm represents the database in the form of a tree called a frequent pattern tree or FP tree. The tree structure will maintain the association between the item sets.

Mining frequent patterns using FP-tree:

1. Start from each frequent length-1 pattern (suffix pattern).
2. Construct its conditional pattern base.
3. Construct its conditional FP-tree.
4. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

FP Tree:

An FP-tree, or frequent pattern tree, is a data structure used by the FP-Growth algorithm for efficient mining of frequent patterns in a transactional database. The FP-tree is constructed by recursively inserting each transaction in the database into the tree, creating a path from the root of the tree to a leaf node for each transaction. Each node in the tree represents an item in the transactional database, and the count of each item is stored in the corresponding node. Additionally, each node may also have a link to the next node in the tree with the same item, forming a linked list of nodes for each item.

**MANUAL SOLUTION:**

|  |  |
| --- | --- |
| Transaction ID | **Items** |
| 1 | E,K,M,N,O,Y |
| 2 | D,E,K,N,O,Y |
| 3 | A,E,K,M |
| 4 | C,K,M,U,Y |
| 5 | C,E,I,K,O,O |

Min support = 3

Step1: Find out frequency of each data set (1-itemset)

|  |  |
| --- | --- |
| Item | frequency |
| A | 1 |
| C | 2 |
| D | 1 |
| E | 4 |
| I | 1 |
| K | 5 |
| M | 3 |
| N | 2 |
| O | 3 |
| U | 1 |
| Y | 3 |

In above table item {K, E, M, O, Y} crosses the min support= 3

Now, Arrange items into descending order: {K: 5, E: 4, M: 3, O: 3 ,Y: 3}

Step2: For each transaction, the respective Ordered-itemset is built.

|  |  |  |
| --- | --- | --- |
| Transaction ID | Item sets | Ordered-item sets |
| 1 | E,K,M,N,O,Y | K,E,M,O,Y |
| 2 | D,E,K,N,O,Y | K,E,O,Y |
| 3 | A,E,K,M | K,E,M |
| 4 | C,K,M,U,Y | K,M,Y |
| 5 | C,E,I,K,O,O | K,E,O |

Step3: Building FP-tree by using Ordered-item set.

Inserting the set {K,E,M,O,Y}

NULL

K: 1 2 3 4 5

E: 1 2 3 4

M: 1

M: 1 2

O: 1 2

O: 1

Y: 1

Y: 1

Y: 1

Step4: Conditional Pattern

|  |  |  |
| --- | --- | --- |
| Item | Conditional Pattern | Conditional FP-tree |
| Y | {K,E,M,O: 1},{K,E,O: 1},{K,M: 1} | K=3 |
| O | {K,E,M: 1},{K,E: 2} | K,E=3 |
| M | {K,E: 2}, {K:1} | K=3 |
| E | {K: 4} | K=4 |
| K | --- | --- |

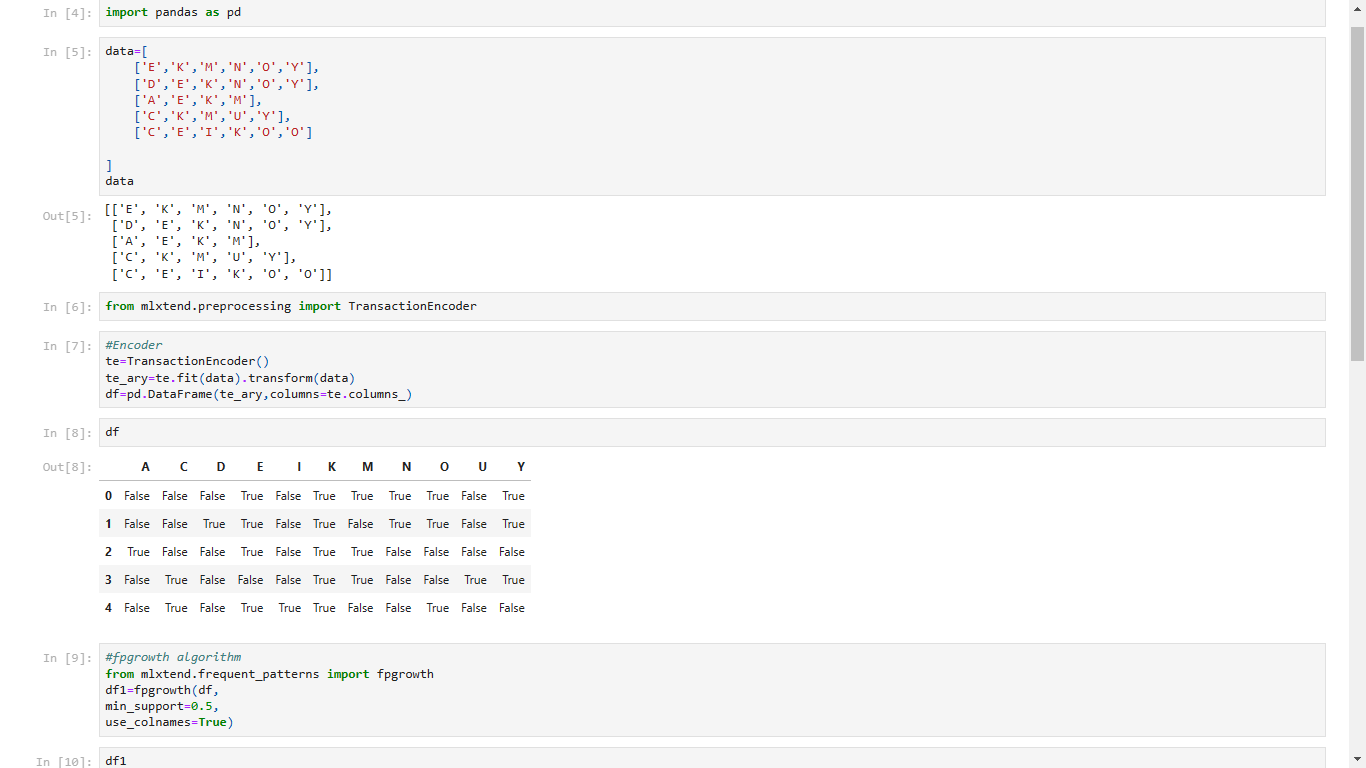
ADVANTAGES OF FP-GROWTH ALGORITHM: -

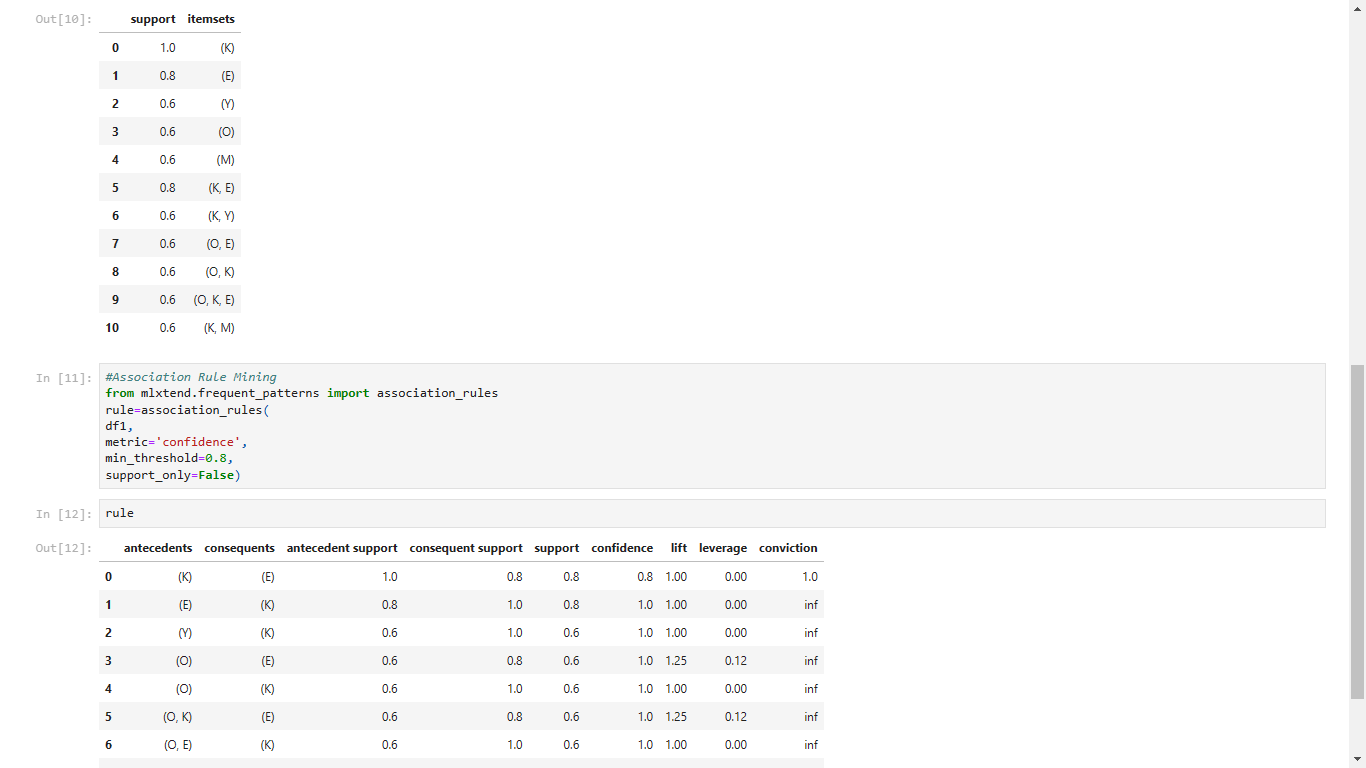
1. No candidate generation, no candidate test.
2. Use compact data structure called FP- Tree
3. It is faster than Apriori algorithm.

DISADVANTAGES OF FP-GROWTH ALGORITHM: -

1. FP-tree is difficult to build than Apriori.
2. It may be expensive.
3. FP-tree may not fit in the memory

**PROGRAM IMPLEMENTATION IN PYTHON**

Source Code: 



CONCLUSION: -

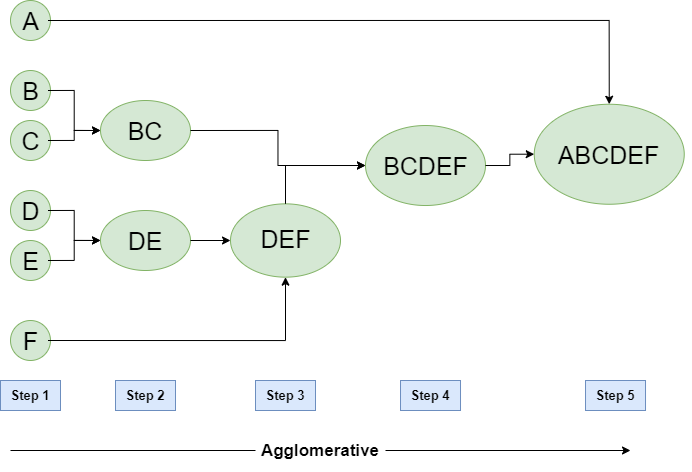
Hence, FP-growth algorithm using Python successfully implemented.

**EXPERIMENT NO: 3**

**NAME OF EXPERIMENT**: -Implementation of Agglomerative Hierarchical clustering algorithm using Python.

**INTRODUCTION:** -Hierarchical clustering is a clustering algorithm that group’s similar data points together based on their distance or similarity. It is a type of unsupervised machine learning technique that aims to find patterns or structures in the data without the need for predefined labels or categories.

Agglomerative clustering is a type of hierarchical clustering algorithm which starts with treating each data point as an individual cluster and then merges the clusters that are most similar to each other based on some distance metric. The process continues until all data points are in a single cluster, forming a dendrogram.

Agglomerative clustering is a bottom-up approach to clustering, and it has the advantage of being able to handle a large number of data points. However, it can be computationally expensive for large datasets and sensitive to the choice of distance metric and linkage method.

Algorithm: -

1. Compute the distance matrix between the input data points.
2. Let each data points to be a cluster.
3. Repeat
4. Merge the two clusters.
5. Update the distance matrix.
6. Until only k cluster remains.

MANUAL CALCULATION:

* Perform the agglomerative hierarchy clustering algorithm based on the following data points.

|  |  |  |
| --- | --- | --- |
| Sample No | X | Y |
| A1 | 0.40 | 0.53 |
| A2 | 0.22 | 0.38 |
| A3 | 0.35 | 0.32 |

* Solution,

Step1: Calculate Euclidean distance between each data and every point.

Euclidean distance =

D (A1, A2) = = 0.23

D (A1, A3) = = 0.22

D (A1, A2) = = 0.14

|  |  |  |  |
| --- | --- | --- | --- |
|  | A1 | A2 | A3 |
| A1 | 0 |  |  |
| A2 | 0.23 | 0 |  |
| A3 | 0.22 | 0.14 | 0 |

Here, 0.14 is the smallest in above table. So, merge (A2, A3).

Step2: Calculate Euclidean distance of merged data.

|  |  |  |
| --- | --- | --- |
|  | A1 | A2, A3 |
| A1 | 0 |  |
| A2, A3 | 0.23 | 0 |

Hence, the new cluster group = A1, (A2, A3).

Step3: Construction of dendrogram.

A1 (A2, A3)

A1 A2 A3

ADVANTAGES OF AGGLOMERATIVE CLUSTERING ALGORITHM: -

1. Does not require the number of clusters to be specified in advance, allowing for more flexibility in the analysis
2. Produces a dendrogram that can be used to visually inspect the clustering structure and identify natural groups or outliers
3. Can be used with various distance metrics and linkage methods to fit different types of data.
4. Can handle a large number of data points

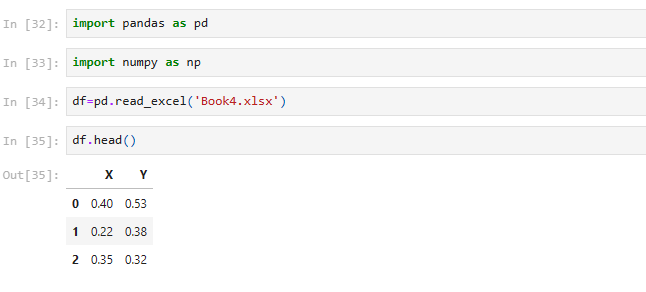
DISADVANTAGES OF AGGLOMERATIVE CLUSTERING ALGORITHM: -

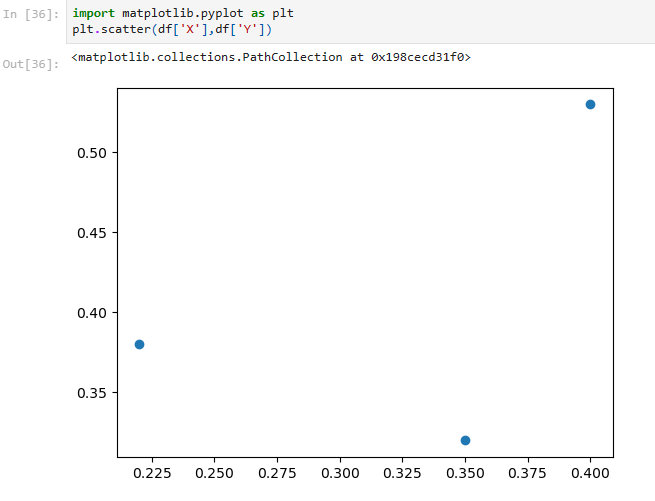
1. It is affected by outliers or noise which can cause suboptimal merging.
2. It has high time and space complexity, making it inefficient for large data sets.
3. Can be computationally expensive, especially for large datasets
4. Does not work well with non-Euclidean distance measures, such as categorical data or binary data.

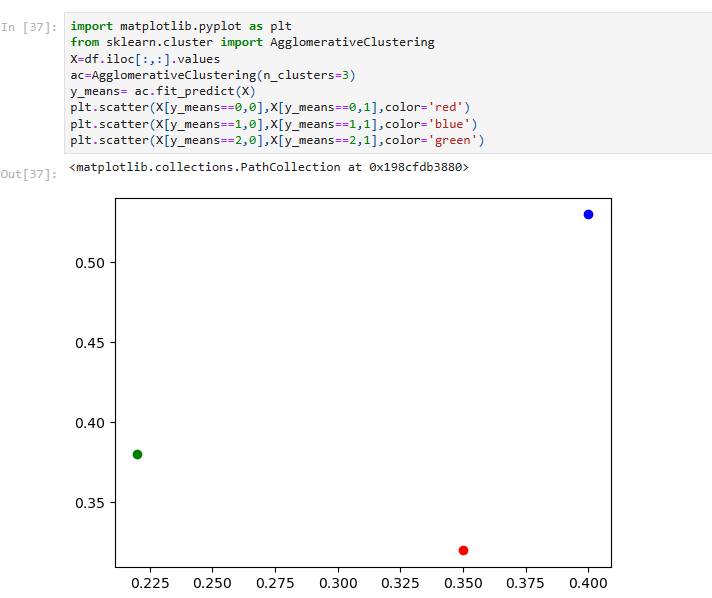
PROGRAM IMPLEMENTATION IN PYTHON: -

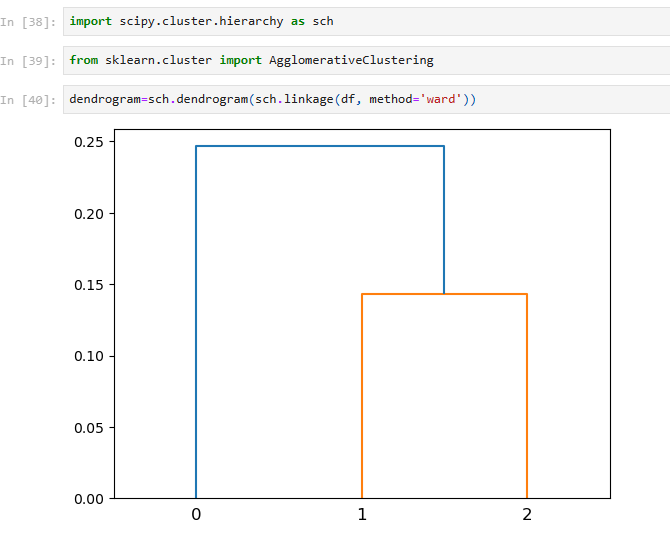
Requirement = Anaconda Navigator

Source Code:









CONCLUSION: -

Hence, Agglomerative clustering algorithm is successful implemented using Python.

**EXPERIMENT NO: 4**

**NAME OF EXPERIMENT**: -Implementation DBSCAN algorithm for clustering analysis using Python.

**INTRODUCTION**: DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups together data points that are closely packed together in high-density regions, while also identifying and removing points that are not part of any cluster (noise). The DBSCAN algorithm has two important parameters: epsilon and minPts. Epsilon determines the radius around each data point, and minPts determines the minimum number of points required to form a dense region or cluster. These parameters can be difficult to set, and tuning them correctly is important for getting good clustering results.

**Epsilon (ε):** The distance that specifies the neighborhoods. Two points are considered to be neighbors if the distance between them is less than or equal to ɛ.

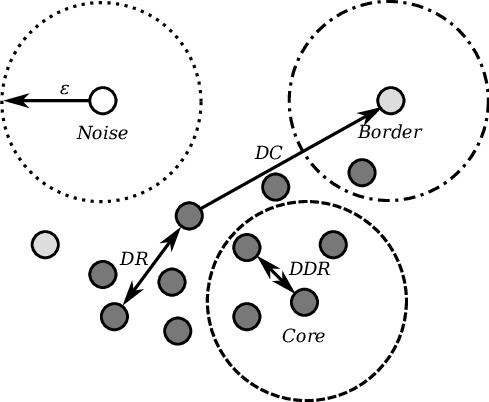
**MinPts:** The minimum number of points (a threshold) clustered together for a region to be considered dense.

Based on these two parameters, points are classified as core point, border point, or outlier:

**Core Point:** Data point that has at least minPts number of points within epsilon (ε) distance.

**Border Point:** Data point that has at least one core point within epsilon (ε) distance and lower than minPts number of points within epsilon (ε) distance from it.

**Noise or Outlier Point:** Data point that has no core points within epsilon (ε) distance.



DBSCAN Algorithm: -   
Input:

D: a data set containing n objects

ε: The radius parameter, and

Minpts: the neighborhood density threshold

Output: A set of density-based clusters  
Method:

1. Mark all objects as unvisited.
2. Do until no object is unvisited.
3. {
4. Randomly select an unvisited object p.
5. Mark p as visited.
6. If the ε-neighborhood of p has at least Minpts objects
7. Create a new cluster C, and add p to C.
8. Let N be the set of objects in the ε-neighborhood of p.
9. For each point p’ in N

{

* If p’ is unvisited

-mark p' as visited.

-If the ε-neighborhood of p' has at least Minpts points, Add those points to N.

* If p' is not yet a member of any cluster, add p' to C.

}

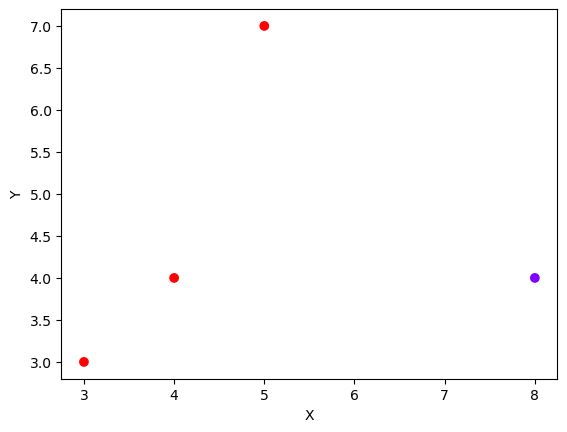
1. Output C.
2. Else mark p as noise
3. }

MANUAL CALCULATION: -

If Epsilon (ε)=3.5 and Minpts=2, what are the clusters that DBSCAN would discover with the following 4 data points shown in table:

|  |  |  |
| --- | --- | --- |
| Data Point | X | Y |
| S1 | 5 | 7 |
| S2 | 8 | 4 |
| S3 | 3 | 3 |
| S4 | 4 | 4 |

* Solution,



**S1**

**S2**

**S3**

**S4**

Step1: Obtain Euclidean distance among all the points.

Identify the neighbors of each point.

Euclidean distance =

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | S1 | S2 | S3 | S4 |
| S1 | 0 | 4.24 | 4.47 | 3.16 |
| S2 | 4.24 | 0 | 5.09 | 4 |
| S3 | 4.47 | 5.09 | 0 | 1.41 |
| S4 | 3.16 | 4 | 1.41 | 0 |

Point out the neighbors within the boundary of radius ε = 3.5

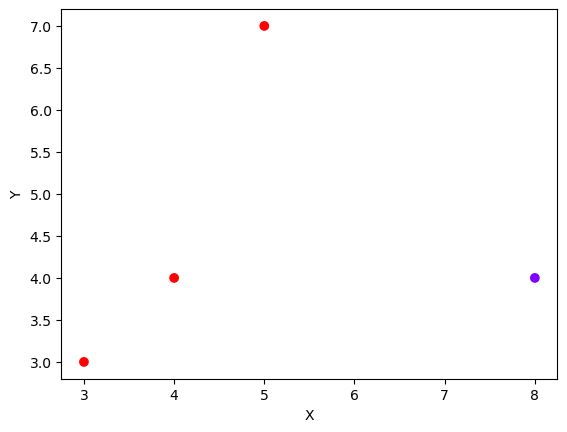
S1: S4,

S2: null,

S3: S4,

S4: S1, S3 {i.e., S4 <ε and S1, S3 <ε}

Step2: Assertion whether the points is the core point for the mean point 2 or not.



**S1**

**S2**

**S3**

**S4**

**Noisy data**

ADVANTAGES OF DBSCAN ALGORITHM: -

1. Does not require the number of clusters to be specified in advance, which makes it useful for datasets with unknown or variable cluster structures.
2. Can handle clusters of different shapes and sizes, and is robust to outliers and noise in the data.
3. Can detect clusters of arbitrary shapes and sizes, as long as they are separated by areas of low density.
4. Can be faster and more efficient than other clustering algorithms for large datasets.

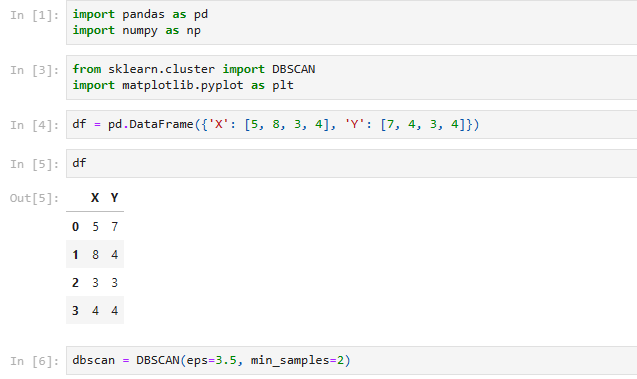
DISADVANTAGES OF DBSCAN CLUSTERING ALGORITHM: -

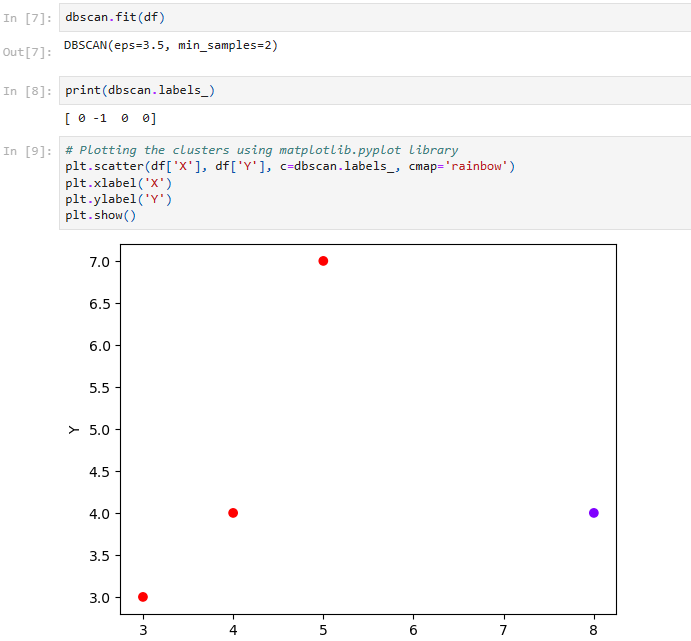
* Choosing appropriate values for hyperparameters (epsilon and minPts) can be difficult and time-consuming.
* Can produce different results depending on the choice of hyperparameters and initialization conditions.
* Does not work well with datasets that have clusters with vastly different densities.
* Can struggle with high-dimensional data due to the curse of dimensionality.

PROGRAM IMPLEMENTATION IN PYTHON: -

Requirement = Anaconda Navigator

Source Code: -





**CONCLUSION:** -

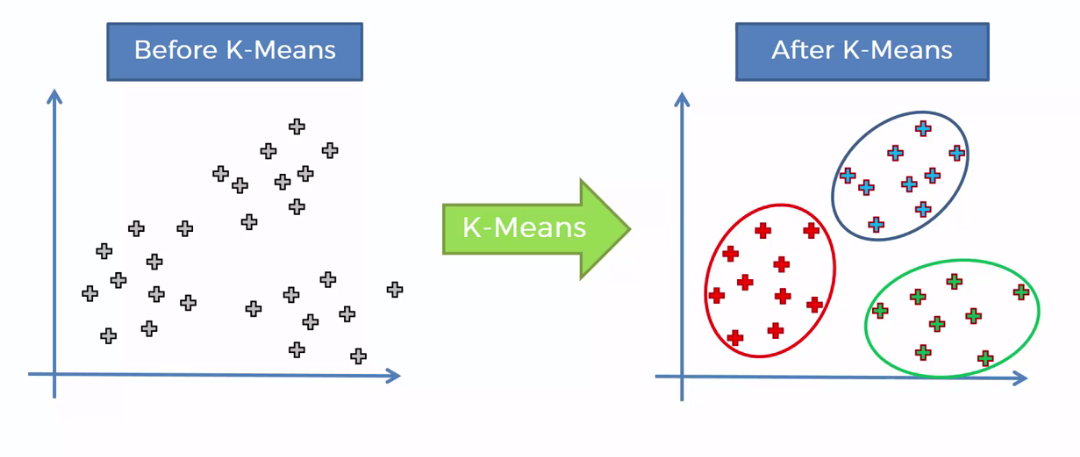
Hence, the DBSCAN algorithm using Python was successful implemented for cluster analysis .

**EXPERIMENT NO: 5**

**NAME OF EXPERIMENT**: Implementation of K-means algorithm for clustering in Python.

**INTRODUCTION**: The k-means algorithm is a popular unsupervised machine learning technique used for clustering data points into groups. The "k" in k-means refers to the number of clusters that the algorithm is instructed to form. It is a simple and most popular for cluster analysis. It aims to partition ‘n’ observations into ‘k’ clusters. Each cluster in the k-means clustering algorithm id represented by a centroid point. Centroid point is the average of all the points in the set.

The main idea of the k-means algorithm is to find k-centroid points and every point in the dataset will belong either of k-sets having minimum Euclidean distance.



K-Means Algorithm:

Input:

k = the number of clusters,

D = dataset containing n objects

Output:

A set of k clusters

Method:

1. Randomly select ‘k’ from D as the initial cluster center.
2. Calculate the distance between each datapoint and cluster centers.

Euclidean distance =

1. Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.
2. Update the cluster means (i.e., calculate the mean value of the objects for each cluster)

MANUAL CALCULATION:

Cluster the following instance of given data with the help of k-means algorithm.

|  |  |  |
| --- | --- | --- |
| S.N. | X | Y |
| 1 | 185 | 72 |
| 2 | 170 | 56 |
| 3 | 168 | 60 |
| 4 | 179 | 68 |

Take K = 2

🡪 Solution,

Step1: Assume two points (k1, k2) randomly as cluster center.

K1 = (185, 72) and K2 = (170, 56)

Euclidean distance for Row 3 (K1, 3)

K1= = = 20.8

Euclidean distance for row 3 (K2, 3) =

= = 4.47

Here, K2 < K1

So the data point 3 belongs to K2.

The resulting cluster is;

K1 = {Row 1}

K2 = {Row2, Row3}

Calculate new dataset for new centroid,

K1 = (185, 72) and K2 = (170+168/2, 56+60/2) = (169, 58)

Step2:

Euclidean distance for Row 4

K1= = = 7.21

K2= = = 14.14

Here, K1< K2

So data point 4 belongs to K1.

The resulting Cluster is;

K1 = {Row 1, Row 4}, K2 = {Row2, Row3}

Calculate new dataset for new centroid,

K1 =(185+179/2, 72+68/2) =(182, 70) and K2 = (169, 58)

The cluster of data points obtained in second iteration is same as the third .So terminate the third one.

Therefore, final cluster = K1 {(185, 72), (179, 68)} and K2 {(170, 56), (168, 60)}.

ADVANTAGES OF K-MEANS CLUSTERING ALGORITHM:

1. It is very simple to implement.
2. It is scalable to a huge dataset and faster to large datasets.
3. It adapts the new examples very frequents.

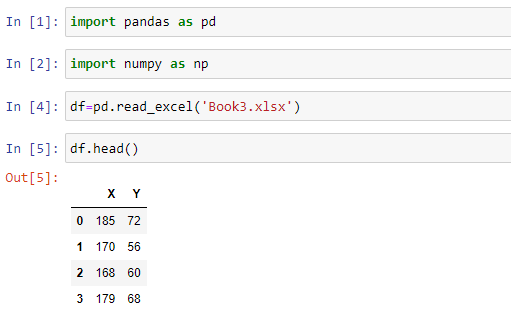
DISADVANTAGES OF K-MEANS CLUSTERINGALGORITHM:

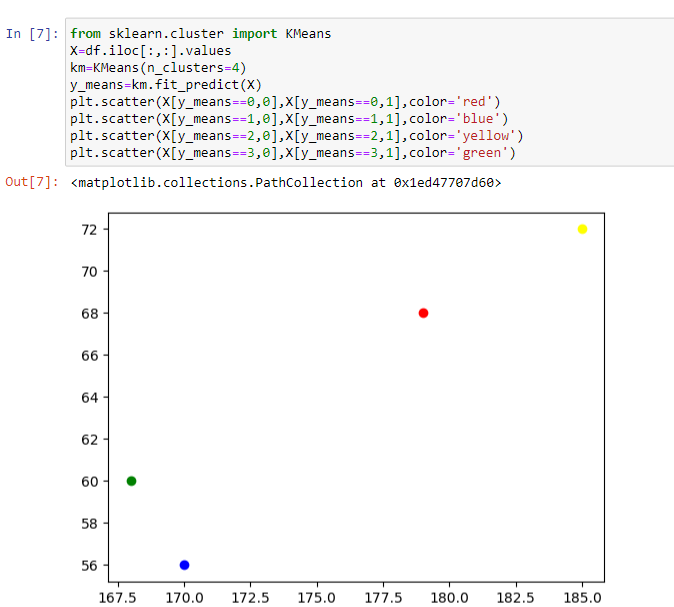
1. Dependent on initial values.
2. It is sensitive to the outliers.
3. Choosing the k values manually is tough job.
4. As the number of dimensions increases its scalability decreases.

PROGRAM IMPLEMENTATION IN PYTHON:

Requirement = Anaconda Navigator

Source Code:





**CONCLUSION:** Hence, Successful Implementation of k-means clustering algorithm using Python.