**ANALYSIS OF COVID-19 DATA**

**USING CLUSTERING**

by

HARSHA PRIYA 19BPS1011

RUPALI TRIPATHI 19BCE1508

A project report submitted to

**Dr. SALEENA B.**

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING**

in partial fulfillment of the requirements for the course of

**CSE2003 – DATA STRUCTURES AND ALGORITHMS**

in

**B.Tech. COMPUTER SCIENCE ENGINEERING**



**VIT University**

**NOVEMBER 2020**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S no.** |  | **Description** | **Page no.** |
| 1 |  | *Bonafide Certificate* | 3 |
| 2 |  | *Abstract* | 4 |
| 3 |  | *Acknowledgements* | 5 |
| 4 |  | *Introduction* | 6 |
|  |  | *What is clustering?* |  |
|  |  | *Objectives and Goals* | 8 |
| 5 |  | *Algorithm used* | 9 |
| 6 |  | *Dataset Used* | 13 |
| 7 |  | *Algorithm Implementation* | 14 |
|  | 7.1 | *K-means Clustering Algorithm* |  |
|  | 7.2 | *DBScan Clustering Algorithm* |  |
|  | 7.3 | *Hierarchical clustering Algorithm* |  |
|  | 7.4 | *Fuzzy c-means Algorithm* |  |
| 8 |  | *Output and result* | **25** |
|  | 8.1 | *Analysis of algorithms* |  |
|  | 8.2 | *Prediction using the Algorithms* |  |
| 9 |  | *Conclusion* | 38 |
| 10 |  | *Future Work* | 39 |
| 11 |  | *References* | 40 |
| 12 |  | Poster | 41 |
|  |  |  |  |

1. **BONAFIDE CERTIFICATE**

Certified that this project report entitled “**ANALYSIS OF COVID-19 DATA USING CLUSTERING”** is a bonafide work of RUPALI TRIPATHI (19BCE1508) and HARSHA PRIYA (19BPS1011**)** who carried out the Project work under my supervision and guidance.

**Dr. SALEENA B**

Associate Professor

School of Computing Science and Engineering (SCSE),

VIT University, Chennai

Chennai – 600127.

**2. ABSTRACT**

The word ‘clustering’ means grouping similar things together. The present study presents a novel analysis which results to clustering countries with respect to a recovered cases, deaths and confirmed cases per state in India and comparing each states’ condition w.r.t each other based on COVID time-series dataset provided by **covid-19-india dataset.**

The presented cluster results could be useful to a variety of different policy makers, such as physicians and managers of the health sector, economy/finance experts, politicians and even to sociologists.

The features of the data used in this clustering analysis can tell us about how every state is handling this covid-19 pandemic and will help us to follow the steps of those states which are doing good and avoid the mistakes of the states doing bad in this pandemic, hence improve the condition of India.

1. **ACKNOWLEDGEMENT**

We are extremely grateful to **Dr. Jagadeesh Kannan R,** Dean of the School of Computer Science Engineering, VIT Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our Programme Chair **Dr. Justus S** for his support throughout the course of this project.

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Saleena B.** , School of Computing Science and Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

HARSHA PRIYA RUPALI TRIPATHI

1. **INTRODUCTION**

**What is Clustering?**

**How it is used in this project?**

**Clustering** is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.It comes under the **unsupervised machine learning technique.**

We used clustering in our project using :

**1) K-means** Clustering algorithm

**2) DBScan** Clustering algorithm

**3) Fuzzy c-means** Clustering algorithm

**4) Hierarchical** Clustering algorithm

**1) K-means Clustering algorithm**

K-means clustering is one of the most widely used **unsupervised machine learning algorithms** that forms clusters of data based on the similarity between data instances.

K-means clustering is a good place to start exploring an unlabeled dataset.

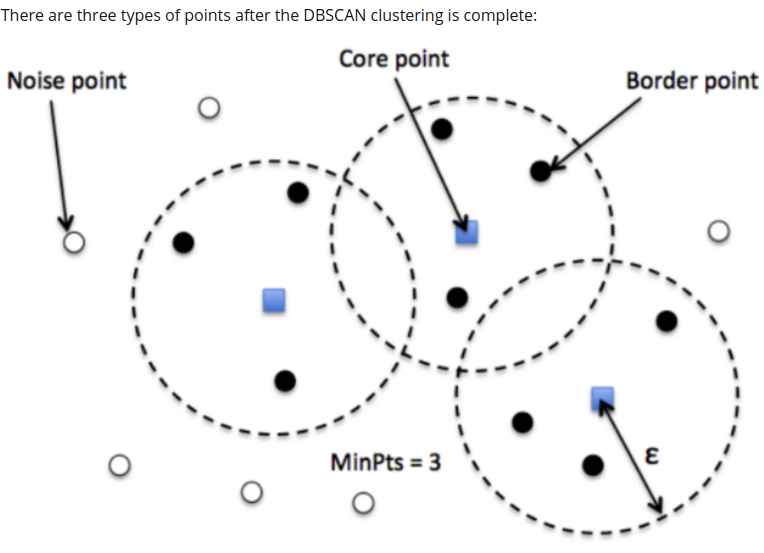
For this particular algorithm to work, the number of clusters has to be defined

beforehand. The K in the K-means refers to the number of clusters.

**2) DBScan Clustering algorithm**

Density-Based Clustering refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.



**3) Fuzzy c-means Clustering algorithm**

**Fuzzy clustering** (also referred to as **soft clustering** or **soft *k*-means**) is a form of clustering in which each data point can belong to more than one cluster.

Membership grades are assigned to each of the data points (tags). These membership grades indicate the degree to which data points belong to each cluster. Thus, points on the edge of a cluster, with lower membership grades, may be *in the cluster* to a lesser degree than points in the center of cluster.

One of the most widely used fuzzy clustering algorithms is the Fuzzy C-means clustering (FCM) Algorithm.

**4) Hierarchical Clustering algorithm**

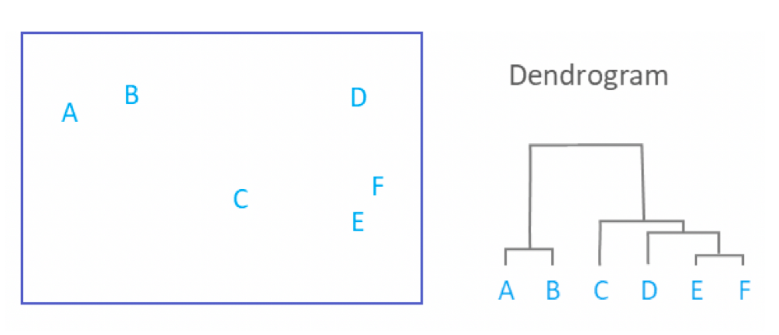
Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

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.

The main output of Hierarchical Clustering is a [dendrogram](https://www.displayr.com/what-is-dendrogram/), which shows the hierarchical relationship between the clusters:



**OBJECTIVES AND GOALS**

To implement the algorithm to predict the clusters and do its analysis .

1. To Learn and Summarize what K-means, DBscan, fuzzy-c means and Hierchical Clustering is.
2. To get a brief idea about unsupervised machine learning .
3. To explore the states’ dataset!
4. To apply all the 4 techniques on the states’ dataset, with the goal of finding a useful grouping. (Unsupervised learning)
5. The features of the data used in this clustering analysis can tell us about how every state is handling this covid-19 pandemic and will help us to follow the steps of those states which are doing good and avoid the mistakes of the states doing bad in this pandemic, hence improve the condition of India.
6. **ALGORITHMS USED**

**1) K-means Clustering algorithm**

Specify number of clusters K.

* Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
* Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn’t changing.
* Compute the sum of the squared distance between data points and all centroids.
* Assign each data point to the closest cluster (centroid).
* Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

Expectation-Maximization The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.

**2) DBSCAN- Clustering Algorithm**

**Density-Based Spatial Clustering of Applications with Noise (DBSCAN)** is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

**The DBSCAN algorithm uses two parameters:**

* minPts: The minimum number of points (a threshold) clustered together for a region to be considered dense.
* eps (ε): A distance measure that will be used to locate the points in the neighborhood of any point.

These parameters can be understood if we explore two concepts called Density Reachability and Density Connectivity.

Reachability in terms of density establishes a point to be reachable from another if it lies within a particular distance (eps) from it.

Connectivity, on the other hand, involves a transitivity based chaining-approach to determine whether points are located in a particular cluster. For example, p and q points could be connected if p->r->s->t->q, where a->b means b is in the neighborhood of a.

There are three types of points after the DBSCAN clustering is complete:

* Core — This is a point that has at least m points within distance n from itself.
* Border— This is a point that has at least one Core point at a distance n.
* Noise — This is a point that is neither a Core nor a Border. And it has less than m points within distance n from itself.

What’s nice about DBSCAN is that you don’t have to specify the number of clusters to use it. All you need is a function to calculate the distance between values and some guidance for what amount of distance is considered “close”. DBSCAN also produces more reasonable results than k-means across a variety of different distributions

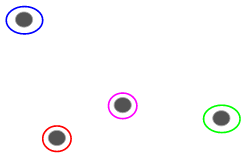
**3) Hierarchical Clustering algorithm**

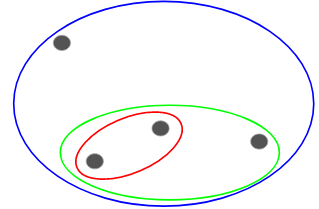
Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

Agglomerative: This is a "bottom up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.

Divisive: This is a "top down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

Agglomerative Hierarchical clustering Technique: In this technique, initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.





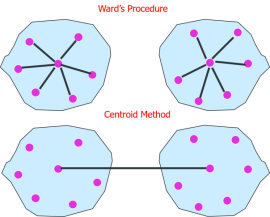
**The basic algorithm of Agglomerative is straight forward.**

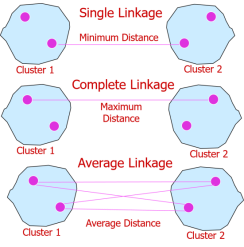
* **Compute the proximity matrix**
* **Let each data point be a cluster**
* **Repeat: Merge the two closest clusters and update the proximity matrix**
* **Until only a single cluster remains**

Key operation is the computation of the proximity of two clusters

There are different ways to find distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Following are some of the options to measure distance between two clusters:

1. Single- Measure the distance between the closes points of two clusters.
2. Complete- Measure the distance between the farthest points of two clusters.
3. Centroid- Measure the distance between the centroids of two clusters.
4. Average - Measure the distance between all possible combination of points between the two clusters and take the mean
5. ward's method - sum of squared Euclidean distance is minimized

****

****

A **dendrogram** is a tree-like diagram that records the sequences of merges or splits.Dendogram is used to figure out the number of clusters.

Advantages

1) No Apriori information about the number of clusters required.

2) Easy to implement and gives best result in some cases.

Disadvantages

1) Algorithm can never undo what was done previously.

2) Time complexity of at least O(*n2 log n*) is required, where *‘n’* is the number of data points.

3) Based on the type of distance matrix chosen for merging different algorithms can suffer with one or more of the following:

i) Sensitivity to noise and outliers

ii) Breaking large clusters

iii) Difficulty handling different sized clusters and convex shapes

4) Sometimes it is difficult to identify the correct number of clusters by the dendogram.

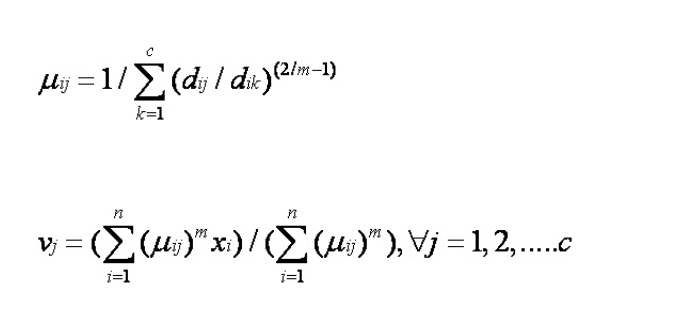
**4) Fuzzy c-means Clustering algorithm**

The unsupervised k-means clustering algorithm gives the values of any point lying in some particular cluster to be either as 0 or 1 i.e., either true or false. But the **fuzzy logic gives the fuzzy values of any particular data point to be lying in either of the clusters.** Here, in fuzzy c-means clustering, we find out the centroid of the data points and then calculate the distance of each data point from the given centroids until the clusters formed becomes constant.

This algorithm works by assigning membership to each data point corresponding to each cluster center on the basis of distance between the cluster center and the data point. More the data is near to the cluster center more is its membership towards the particular cluster center.

Clearly, summation of membership of each data point should be equal to one

After each iteration membership and cluster centers are updated according to the formula:



where,

*'n'* is the number of data points.

*'vj'* represents the *jth* cluster center.

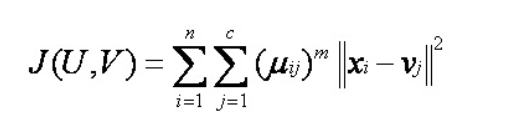
*'m'* is the fuzziness index m € [1, ∞].

*'c'* represents the number of cluster center.

*'µij'* represents the membership of *ith* data to *jth* cluster center.

*'dij'* represents the Euclidean distance between *ith* data and *jth* cluster center.

Main objective of fuzzy c-means algorithm is to minimize:



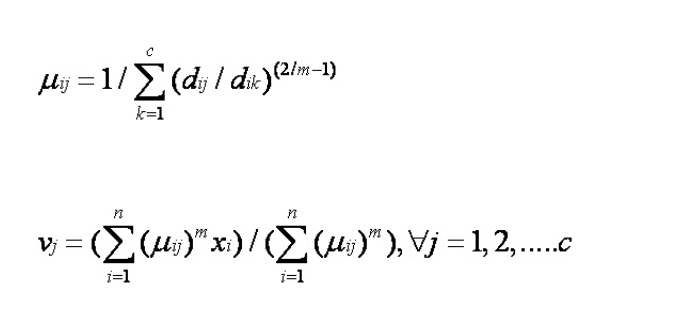
where,

*'||xi – vj||'* is the Euclidean distance between *ith* data and *jth* cluster center.

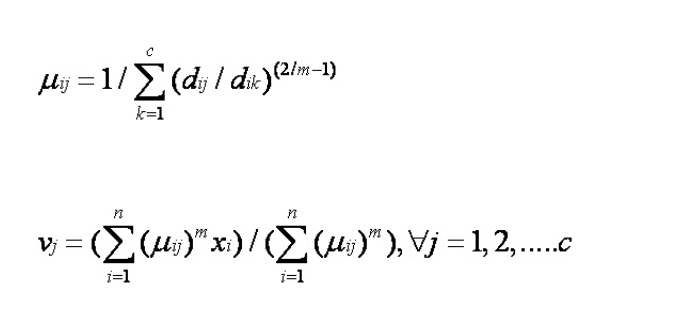
**Algorithmic steps for Fuzzy c-means clustering**

**Let X = {x1, x2, x3 ..., xn} be the set of data points and V = {v1, v2, v3 ..., vc} be the set of centers.**

1. **Randomly select *‘c’* cluster centers.**
2. **2) Calculate the fuzzy membership *'µij'* using:**



**3) Compute the fuzzy centers *'vj'* using:**



4) Repeat step 2) and 3) until the minimum *'J'* value is achieved or *||U(k+1) - U(k)|| < β.*

where, *‘k’* is the iteration step.

*‘β’* is the termination criterion between [0, 1].

*‘U’ = (µij)n\*c’* is the fuzzy membership matrix.

*‘J’* is the objective function.

**Advantages**

1) Gives best result for overlapped data set and comparatively better then k-means algorithm.

2) Unlike k-means where data point must exclusively belong to one cluster center here data point is assigned

membership to each cluster center as a result of which data point may belong to more then one cluster center.

**Disadvantages**

1) Apriori specification of the number of clusters.

2) With lower value of β we get the better result but at the expense of more number of iteration.

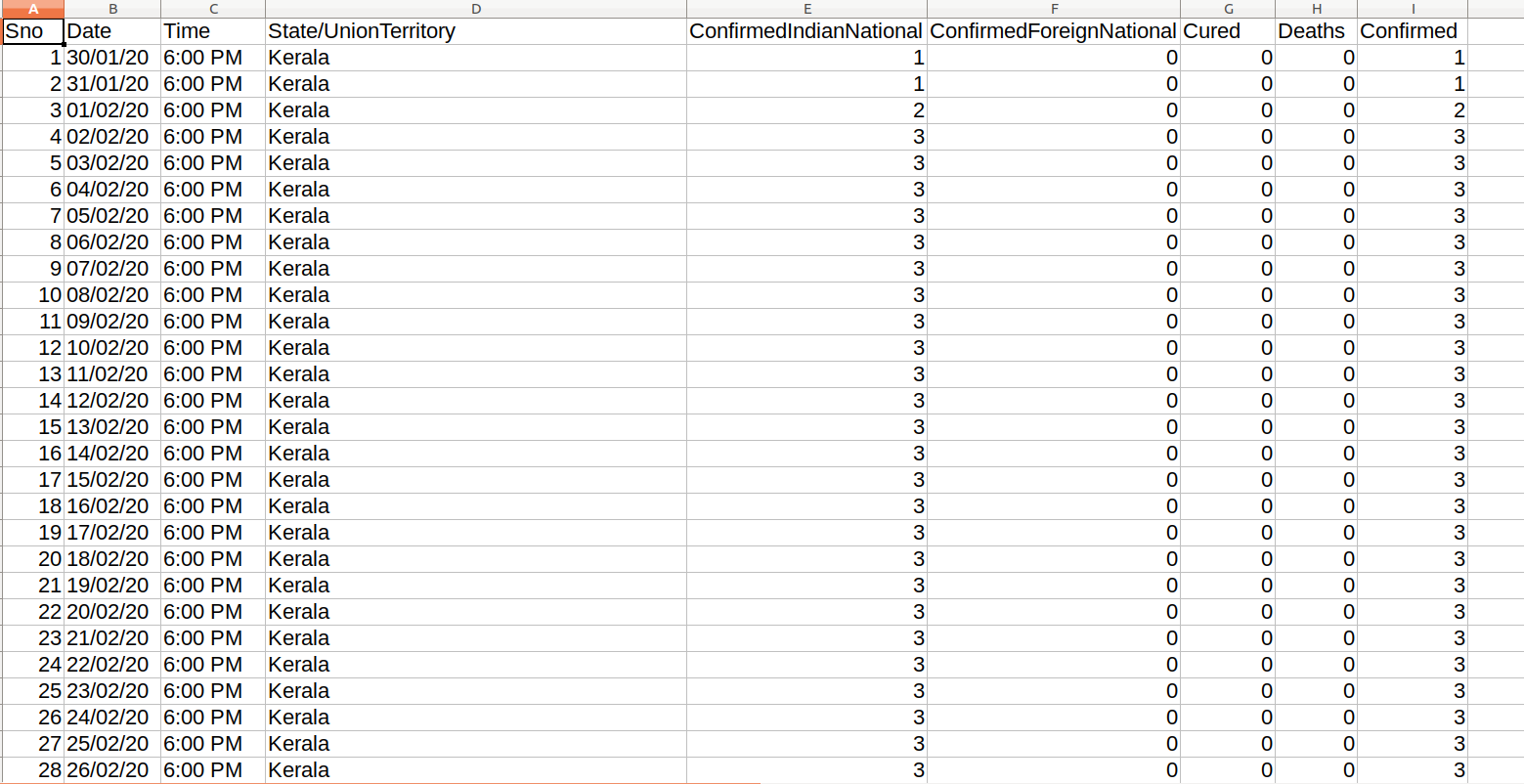
3) Euclidean distance measures can unequally weight underlying factors.

**DATASET**

**Dataset used covid\_19\_india.csv (**[**https://www.kaggle.com/sudalairajkumar/covid19-in-india?select=covid\_19\_india.csv**](https://www.kaggle.com/sudalairajkumar/covid19-in-india?select=covid_19_india.csv)**)**

**Dataset of all states from ( 20/1/2020 till 3/10/2020)**

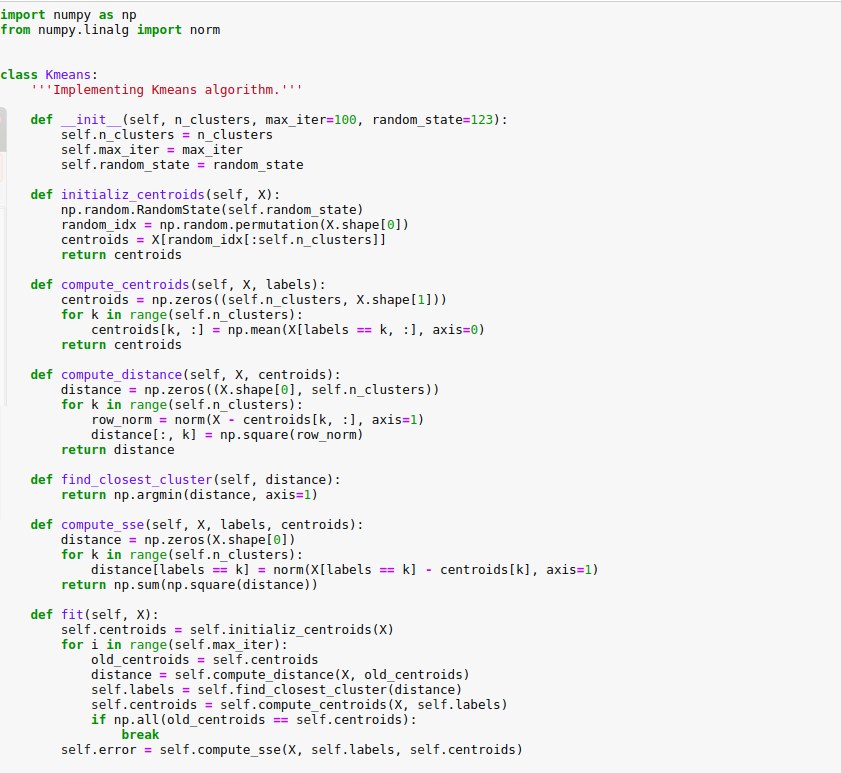
**Screenshots of the dataset**

****

1. **ALGORITHM IMPLEMENTATION**

**1)K-means Cluserting Algorithm**

K-means class used :



Modules Used/imported:

import matplotlib.pyplot as plt # to plot the results on a 2-D plot

from matplotlib.image import imread

import pandas as pd #for data manipulation and analysis

import seaborn as sns #for python data visualization library based on matplotlib

from sklearn.datasets.samples\_generator import (make\_blobs,make\_circles,make\_moons) #machine learning library for the python language

from sklearn.cluster import KMeans, SpectralClustering #importing k-means funtion of sklearn

from sklearn.preprocessing import StandardScaler #to standardize the dataset

from sklearn.metrics import silhouette\_samples, silhouette\_score #to find silhouette coefficeint and plot

%matplotlib inline # styling of plots

sns.set\_context('notebook')

plt.style.use('fivethirtyeight')

from warnings import filterwarnings

filterwarnings('ignore')

# Import the dataset

df = pd.read\_csv("./Downloads/covid\_19\_india.csv")

#finding mean of all the data results of each day for every state to get mean value of each feature state wise

df2 = df.groupby(['State/UnionTerritory']).agg({'Cured': 'mean', 'Deaths': 'mean', 'Confirmed': 'mean'})

df2.size

# Plot the raw data

plt.figure(figsize=(6, 6))

plt.scatter(df2.iloc[:, 0], df2.iloc[:, 1])

df2.head(n=39) #print the the 39 columns obtained

Elbow method

# Run the Kmeans algorithm and get the index of data points clusters

sse = []

list\_k = list(range(1, 10))

for k in list\_k:

km = KMeans(n\_clusters=k)

km.fit(X\_std)

sse.append(km.inertia\_)

#Finding optimal k-value from the elbow method

# Plot sse against k

plt.figure(figsize=(6, 6))

plt.plot(list\_k, sse, '-o')

plt.xlabel(r'Number of clusters $k$')

plt.ylabel('Sum of squared distance');

**Found the elbow at k=3 , the most optimal value to use for k-means clustering of this dataset**

Finding the silhouette plot and coefficient

for i, k in enumerate([2, 3, 4]):

fig, (ax1, ax2) = plt.subplots(1, 2)

fig.set\_size\_inches(18, 7)

# Run the Kmeans algorithm

km = KMeans(n\_clusters=k)

labels = km.fit\_predict(X\_std)

centroids = km.cluster\_centers\_

# Get silhouette samples

silhouette\_vals = silhouette\_samples(X\_std, labels)

# Silhouette plot

y\_ticks = []

y\_lower, y\_upper = 0, 0

for i, cluster in enumerate(np.unique(labels)):

cluster\_silhouette\_vals = silhouette\_vals[labels == cluster]

cluster\_silhouette\_vals.sort()

y\_upper += len(cluster\_silhouette\_vals)

ax1.barh(range(y\_lower, y\_upper), cluster\_silhouette\_vals, edgecolor='none', height=1)

ax1.text(-0.03, (y\_lower + y\_upper) / 2, str(i + 1))

y\_lower += len(cluster\_silhouette\_vals)

# Get the average silhouette score and plot it

avg\_score = np.mean(silhouette\_vals)

ax1.axvline(avg\_score, linestyle='--', linewidth=2, color='green')

ax1.set\_yticks([])

ax1.set\_xlim([-0.1, 1])

ax1.set\_xlabel('Silhouette coefficient values')

ax1.set\_ylabel('Cluster labels')

ax1.set\_title('Silhouette plot for the various clusters', y=1.02);

# Scatter plot of data colored with labels

ax2.scatter(X\_std[:, 0], X\_std[:, 1], c=labels)

ax2.scatter(centroids[:, 0], centroids[:, 1], marker='\*', c='r', s=250)

ax2.set\_xlim([-2, 2])

ax2.set\_xlim([-2, 2])

#ax2.set\_xlabel('Eruption time in mins')

#ax2.set\_ylabel('Waiting time to next eruption')

ax2.set\_title('Visualization of clustered data', y=1.02)

ax2.set\_aspect('equal')

plt.tight\_layout()

plt.suptitle(f'Silhouette analysis using k = {k}',

fontsize=16, fontweight='semibold', y=1.05);

Using the inbuilt kmeans function to plot the clustered data

# Standardize the data so that no one feature with relatively higher mean values should not suppress the value provided by the other features to the results

X\_std = StandardScaler().fit\_transform(df2)

# Run local implementation of kmeans

km = Kmeans(n\_clusters=3, max\_iter=100)

km.fit(X\_std)

centroids = km.centroids # assign random centroids

# Plot the clustered data

fig, ax = plt.subplots(figsize=(6, 6))

plt.scatter(X\_std[km.labels == 0, 0], X\_std[km.labels == 0, 1],marker='o',s=200,

c='#9E2AD4', label='cluster 1')

plt.scatter(X\_std[km.labels == 1, 0], X\_std[km.labels == 1, 1],marker='o',s=200,

c='#2AA6D4', label='cluster 2')

plt.scatter(X\_std[km.labels == 2, 0], X\_std[km.labels == 2, 1],marker='o',s=200,

c='#D4682A', label='cluster 3')

plt.scatter(centroids[:, 0], centroids[:, 1], marker='\*', s=300,

c='red', label='centroid')

plt.legend()

plt.title('Visualization of clustered data', fontweight='bold')

ax.set\_aspect('equal');

**2) DBSCAN clustering Algorithm**

#importing the modules and libraries

import numpy as np

from sklearn.cluster import DBSCAN

from sklearn import metrics

from sklearn.datasets import make\_blobs

from sklearn.preprocessing import StandardScaler

#import the dataset

df = pd.read\_csv("./Downloads/covid\_19\_india.csv")

#finding mean of all the data results of each day for every state to get mean value of each feature state wise

df2 = df.groupby(['State/UnionTerritory']).agg({'Cured': 'mean', 'Deaths': 'mean', 'Confirmed': 'mean'})

#standardize the dataset

X = StandardScaler().fit\_transform(df2)

X, labels\_true = make\_blobs(n\_samples=126, centers=centers, cluster\_std=0.4,

random\_state=0)

df2.head(n=39) # print the standardised dataset

Compute DBSCAN

db = DBSCAN(eps=0.365555, min\_samples=6).fit(X)

core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = True

labels = db.labels\_

# Number of clusters in labels, ignoring noise if present.

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0) #predict number of clusters

n\_noise\_ = list(labels).count(-1) #if the label of datapoint is -1 then it is noise

for i in range(0,len((db.labels\_))): #print the states which are noise

if(labels[i]==-1):

print(i)

print((df2.index[i]))

#print('\n')

print('\n')

# print the estimated values

print('Estimated number of clusters: %d' % n\_clusters\_)

print('Estimated number of noise points: %d' % n\_noise\_)

print("Homogeneity: %0.3f" % metrics.homogeneity\_score(labels\_true, labels))

print("Completeness: %0.3f" % metrics.completeness\_score(labels\_true, labels))

print("V-measure: %0.3f" % metrics.v\_measure\_score(labels\_true, labels))

print("Adjusted Rand Index: %0.3f"

% metrics.adjusted\_rand\_score(labels\_true, labels))

print("Adjusted Mutual Information: %0.3f"

% metrics.adjusted\_mutual\_info\_score(labels\_true, labels))

print("Silhouette Coefficient: %0.3f"

% metrics.silhouette\_score(X, labels))

# Plot result

import matplotlib.pyplot as plt

%matplotlib inline

# Black removed and is used for noise instead.

unique\_labels = set(labels)

colors = [plt.cm.Spectral(each)

for each in np.linspace(0, 1, len(unique\_labels))]

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)

xy = X[class\_member\_mask & core\_samples\_mask]

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

markeredgecolor='k', markersize=14)

xy = X[class\_member\_mask & ~core\_samples\_mask]

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

markeredgecolor='k', markersize=8)

plt.title('Estimated number of clusters: %d' % n\_clusters\_)

plt.show()

**3)HIERARCHICAL CLUSTERING ALGORITHM**

#1 Importing the libraries

import pandas as pd

import numpy as np

#2 Importing the covid\_india dataset by pandas

df = pd.read\_csv(r'C:\Users\DELL\Desktop\dsa project\covid\_19\_india.csv')

dataset=df.groupby(['State/UnionTerritory']).agg({'Cured':'mean','Deaths':'mean','Confirmed':'mean'})

dataset.size

dataset

dataset.info()

#Using the dendrogram to find the optimal numbers of clusters.

#First thing we're going to do is to import scipy library. scipy is

#an open source Python library that contains tools to do

#hierarchical clustering and building dendrograms. Only import the needed tool.

from scipy.cluster.hierarchy import linkage, dendrogram

import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = (10, 7)

plt.style.use('ggplot')

mergings = linkage(samples, method='ward')

dendrogram(mergings,

labels=names,

leaf\_rotation=90,

leaf\_font\_size=10,

)

plt.show()

dendrogram(mergings,

labels=names,

leaf\_rotation=90,

leaf\_font\_size=10,

)

plt.axhline(y=300000, color='r', linestyle='--')

#HIERARCHICAL CLUSTERING

# Fitting Hierarchical Clustering to the data, it tells which cluster each data belongs to

#ward's method - sum of squared Euclidean distance is minimized

from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n\_clusters = 3, affinity = 'euclidean', linkage = 'ward')

y\_hc = hc.fit\_predict(samples)

y\_hc

#put labels

dataset['labels']=y\_hc

dataset.head(n=39)

#visualize how it looks like

plt.scatter(samples[y\_hc== 0,0], samples[y\_hc==0,1],s=100,c='green')

plt.scatter(samples[y\_hc== 1,0], samples[y\_hc==1,1],s=100,c='orange')

plt.scatter(samples[y\_hc== 2,0], samples[y\_hc==2,1],s=100,c='red')

**4) FUZZY-C-MEANS ALGORITHM**

pip install fuzzy-c-means

#importing required libraries

import pandas as pd

import numpy as np

import random

import operator

import math

from fcmeans import FCM

from sklearn.datasets import make\_blobs

from matplotlib import pyplot as plt

from seaborn import scatterplot as scatter

# Importing the covid\_india dataset by pandas

df\_full = pd.read\_csv(r'C:\Users\DELL\Desktop\dsa project\covid\_19\_india.csv')

df=df\_full.groupby(['State/UnionTerritory']).agg({'Cured':'mean','Deaths':'mean','Confirmed':'mean'})

df.size

df.head()

df.info()

# Number of Attributes

num\_attr = len(df.columns) - 1

# Number of Clusters to make

k = 3

# Maximum number of iterations

MAX\_ITER = 100

# Number of data points

n = len(df)

# Fuzzy parameter

m = 2.00

#Fuzzy C-means Algorithm

#STEP 1: initializing the membership matrix with random values

def initializeMembershipMatrix():

membership\_mat = list()

for i in range(n):

random\_num\_list = [random.random() for i in range(k)]

summation = sum(random\_num\_list)

temp\_list = [x/summation for x in random\_num\_list]

membership\_mat.append(temp\_list)

return membership\_mat

#STEP 2: calculating the cluster center, is done in every iteration

def calculateClusterCenter(membership\_mat):

cluster\_mem\_val = list(zip(\*membership\_mat))

cluster\_centers = list()

for j in range(k):

x = list(cluster\_mem\_val[j])

xraised = [e \*\* m for e in x]

denominator = sum(xraised)

temp\_num = list()

for i in range(n):

data\_point = list(df.iloc[i])

prod = [xraised[i] \* val for val in data\_point]

temp\_num.append(prod)

numerator = map(sum, zip(\*temp\_num))

center = [z/denominator for z in numerator]

cluster\_centers.append(center)

return cluster\_centers

#STEP 3: updating the membership values using the cluster centers form step 2

def updateMembershipValue(membership\_mat, cluster\_centers):

p = float(2/(m-1))

for i in range(n):

x = list(df.iloc[i])

distances = [np.linalg.norm(list(map(operator.sub, x, cluster\_centers[j]))) for j in range(k)]

for j in range(k):

den = sum([math.pow(float(distances[j]/distances[c]), p) for c in range(k)])

membership\_mat[i][j] = float(1/den)

return membership\_mat

#Function defined which returns the Clusters from the Membership Matrix

def getClusters(membership\_mat):

cluster\_labels = list()

for i in range(n):

max\_val, idx = max((val, idx) for (idx, val) in enumerate(membership\_mat[i]))

cluster\_labels.append(idx)

return cluster\_labels

# the final fcm function,

#Calling fcm function which runs for MAX\_ITER number of times and returns the Result

#Repeat the steps until the constant values are obtained for the membership values or the difference is less than the tolerance value

def fuzzyCMeansClustering():

# Membership Matrix

membership\_mat = initializeMembershipMatrix()

curr = 0

while curr <= MAX\_ITER:

cluster\_centers = calculateClusterCenter(membership\_mat)

membership\_mat = updateMembershipValue(membership\_mat, cluster\_centers)

cluster\_labels = getClusters(membership\_mat)

curr += 1

print(membership\_mat)

return cluster\_labels, cluster\_centers

#Displaying the Results

#Outputting Cluster Labels and Cluster Centers

# calling the main function and storing the final results in labels, centers

print("printing the final membership matrix")

labels, centers = fuzzyCMeansClustering()

print("printing cluster centers")

print(centers)

n\_samples = 39

n\_bins = 3 # use 3 bins for calibration\_curve as we have 3 clusters here

centers = centers

X=df.iloc[:, [0,1, 2]]

X.head()

# fit the fuzzy-c-means

fcm = FCM(n\_clusters=3)

fcm.fit(X)

# outputs

fcm\_centers = fcm.centers

fcm\_labels = fcm.u.argmax(axis=1)

df['labels']=fcm\_labels

df.head(n=39)

fcm\_centers

pd.Series(fcm\_labels).value\_counts()

# plot result

%matplotlib inline

f, axes = plt.subplots(1, 2, figsize=(11,5))

scatter(X.iloc[:,0], X.iloc[:,1], ax=axes[0])

scatter(X.iloc[:,0], X.iloc[:,1], ax=axes[1], hue=fcm\_labels)

scatter(fcm\_centers.iloc[:,0], fcm\_centers.iloc[:,1], ax=axes[1],marker="s",s=100)

plt.show()

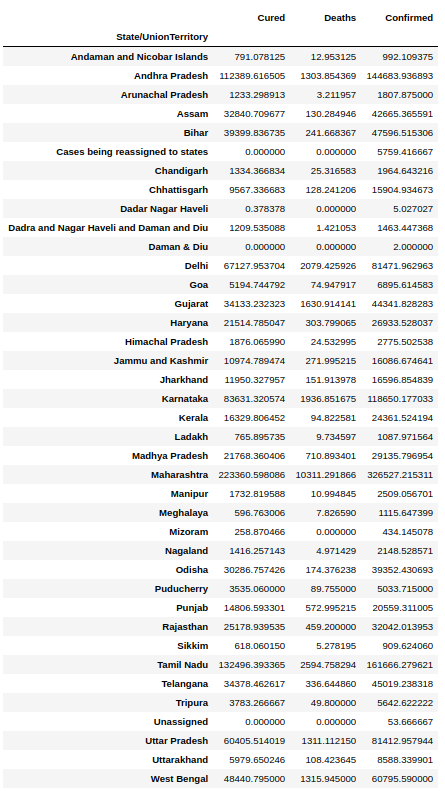
1. **OUTPUT AND RESULT**

**ANALYSIS OF ALGORITHMS**

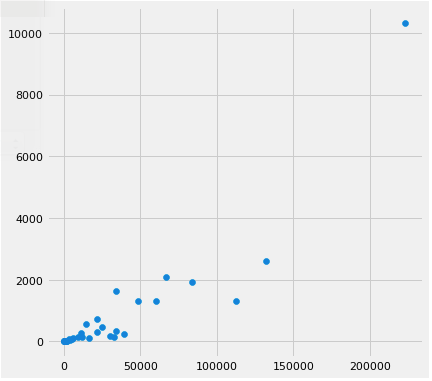
**1) K-means Algorithm**

****

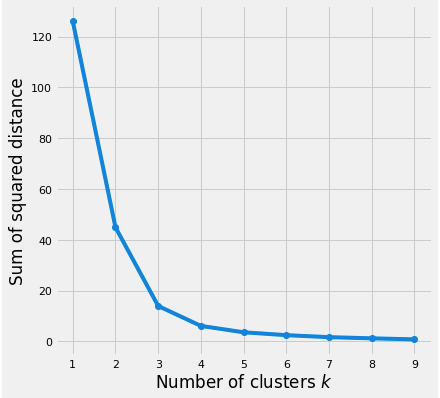
The dataset with mean values of each feature statewise:



**Visualization of raw data**

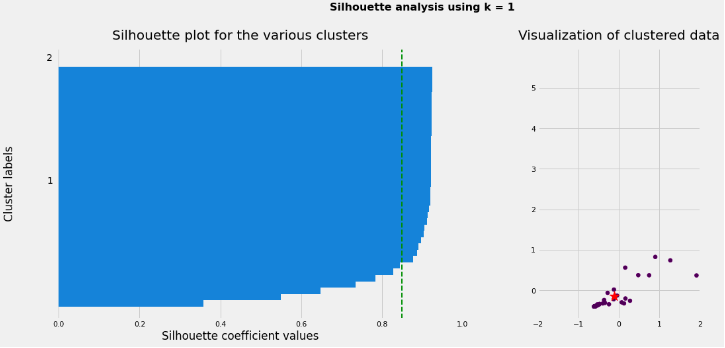
****

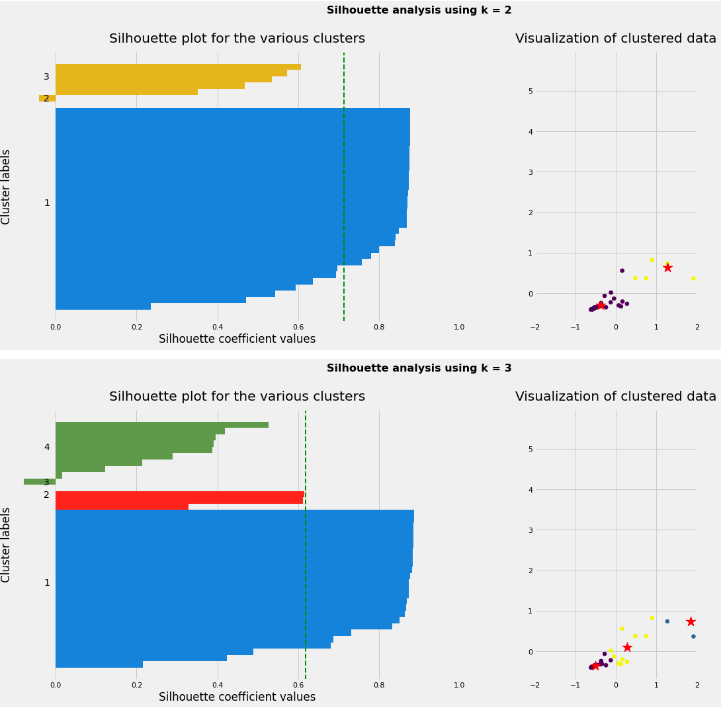
**Using the elbow method to find the optimal k value**

****

**Found the elbow at k=3 , the most optimal value to use for k-means clustering of this dataset**

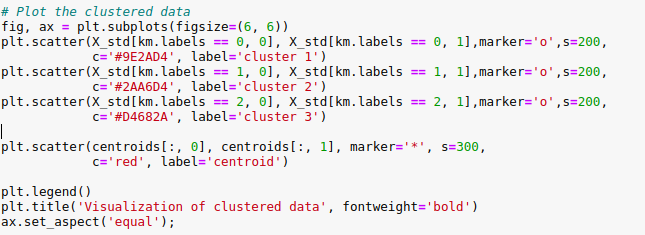
**Silhouette Analysis**

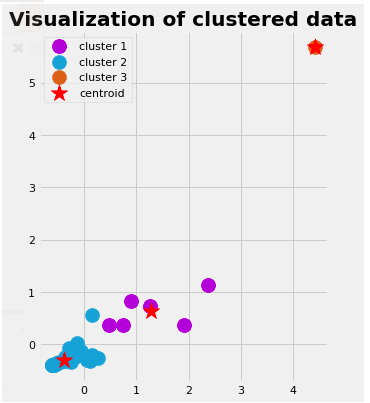
****

****

From The silhouette analysis we find that for k=1 and k=2 one of the clusters are occupying most of the space which doesn’t really give us any information of the types of clusters that can be formed while for k=3 , the red region shows that very less states are different from a big part of the collection that is the blue region , while the green region shows a medium density .

**Therefore we finally take k=3 for the k-means clustering.**

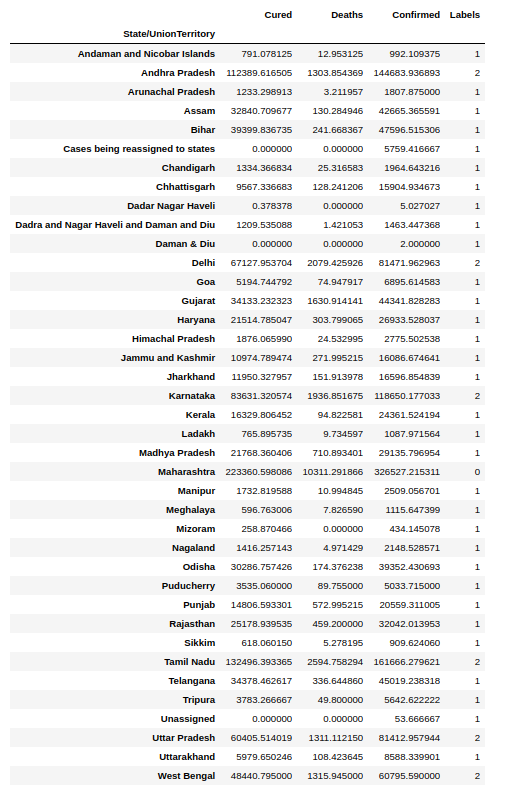


****

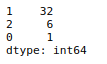
Only one state lies in the cluster 3, i.e it behaves extremely different from the other states in India

Labeling the states , to find which state lies in which cluster(0 or 1 or 2)

****

****

**Now finding out the how my states lie in each cluster**

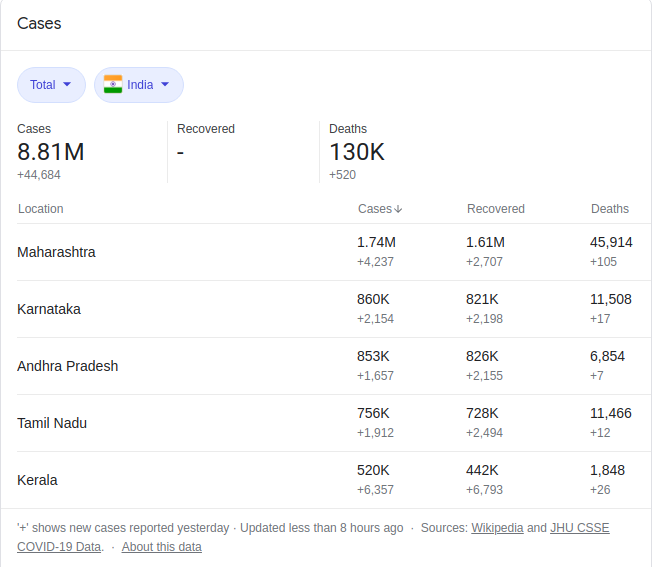
****

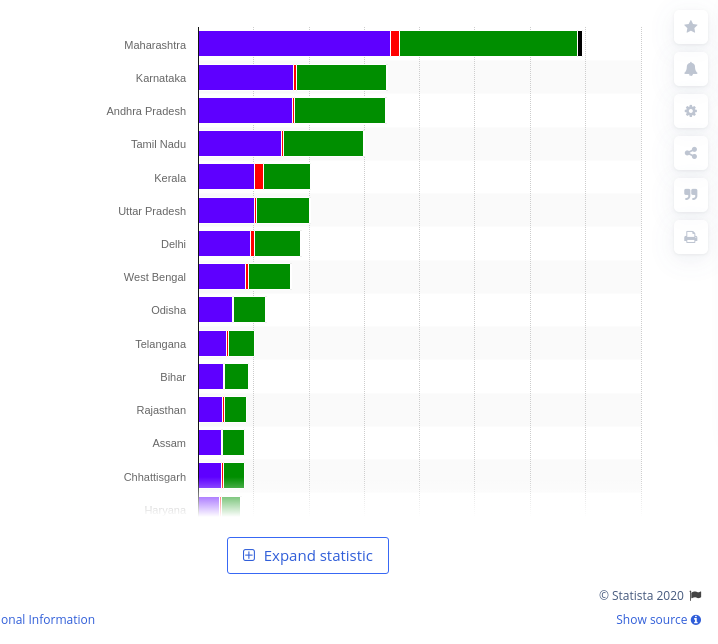
**From the above labeled dataset we find that the only state lying under the cluster 0 is**

**Maharashtra**

****

To prove the accuracy of our results we looked up the official statistics-

****



So Maharashtra does show exceptional behaviour with the most confirmed, recovered and deceased people , therefore at the top of all the charts released as of 15/11/2020 on official sites.

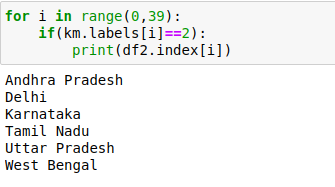
**Reasoning-**

One significant reason for infections not slowing down in the state may be high levels of urbanisation. While 62.8% of total infections in India come from urban centres, in Maharashtra this ratio is a higher 83.7%.

While Mumbai, Pune and Thane still account for 42% of total daily cases, over the last month their contribution has increased 1.7 times. Meanwhile, daily cases have grown 2.5 times for other urban centres. Infections in rural areas have risen 2.1 times but the increase in semi-urban districts has been the slowest at 1.8-times. Within urban districts too, it is the municipal corporations that are contributing more. On September 13, 60% of cases in Nagpur district were registered in the Nagpur municipal corporation as per the state government’s report.

While in Cluster 2 only 6 states lie

They are:



**Reasoning-**

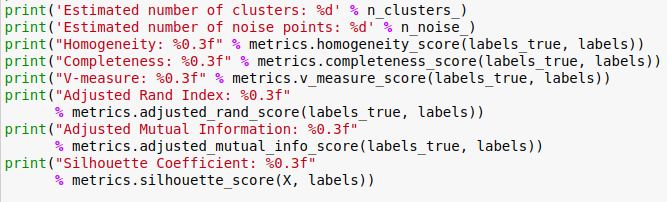
They are clustered together because they have all the 3 features i.e Cured, Deaths, Confirmed to be very much higher relative to other states’ features .

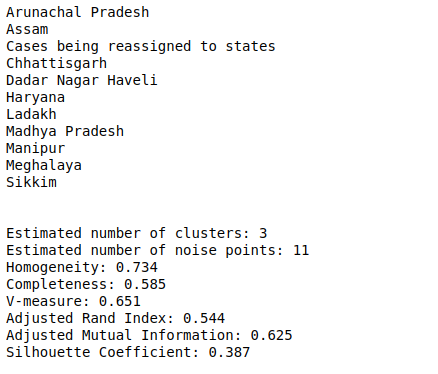
From the above graph these states do come up in the top 8 states .

While the other states in cluster 0 have one feature to be very high but others to be very low .

Hence the K-means clustering works perfectly with the official dataset taken and gives us very accurate results.

**2) DBSCAN Algorithm**



****

The 11 states mentioned are the noise points of this density based clustering, they do not belong to any of the clusters.

**Reasoning-**

These states are noise points according to DBSCAN algorithm because of the relatively very low death mean value of these states with respect to their other features.





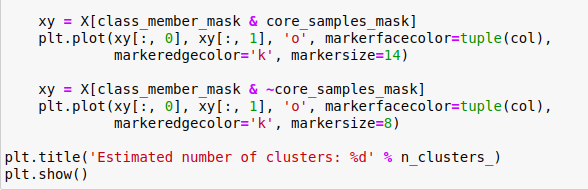


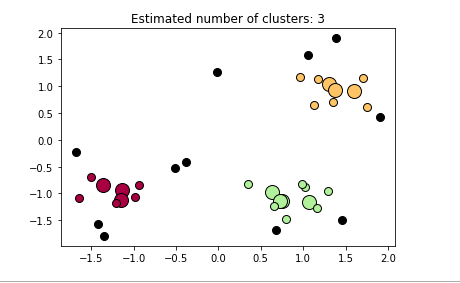




The middle row is the deaths column

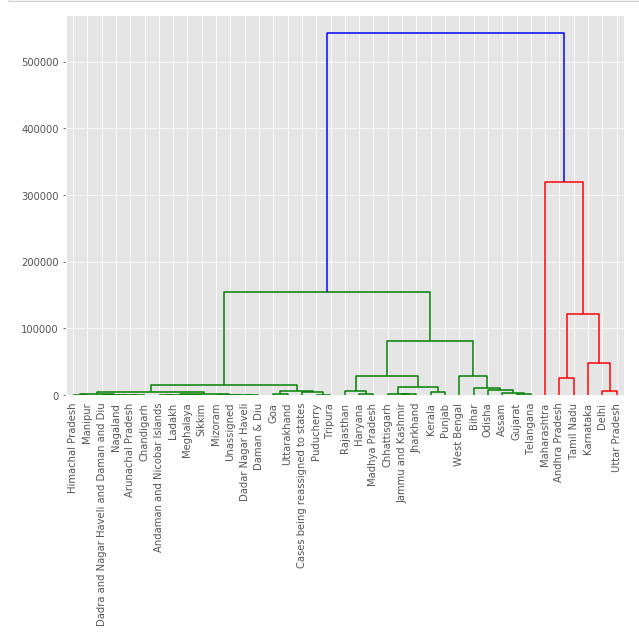
Plotting the clustered data



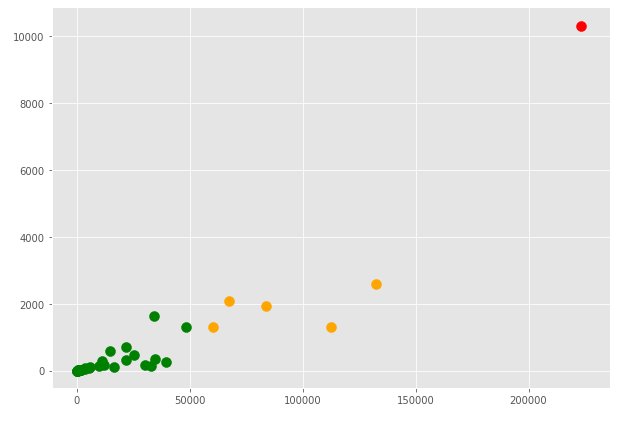
****

**The black points represent the noise points while each cluster is density based clustered , the bigger circles show more density while the smaller ones with low density.**

**3) HIERARCHICAL CLUSTERING)**



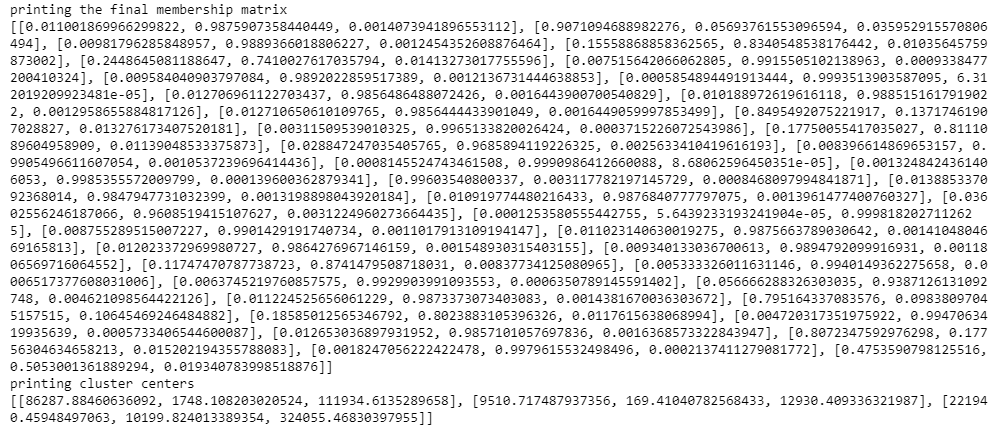
We set the treshold as 300000 to get three clusters.



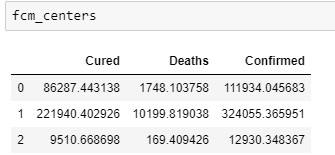
We get 1 state (Maharastra) in cluster 2 ,5 states in cluster 1 and the rest of the states/UT in cluster 0. By looking at the cluster we can say about the low, medium and high risk zones in india.

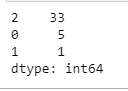
**(FOR FUZZY-C MEANS CLUSTERING)**

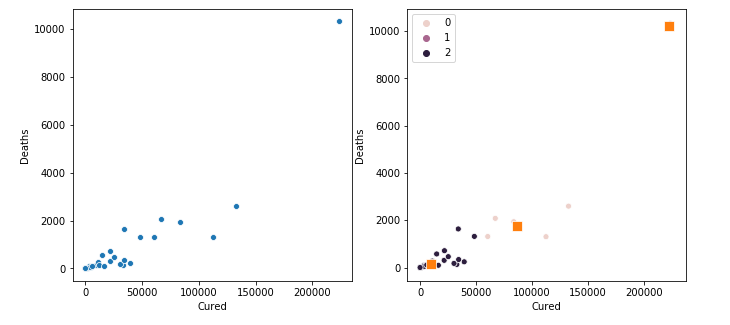


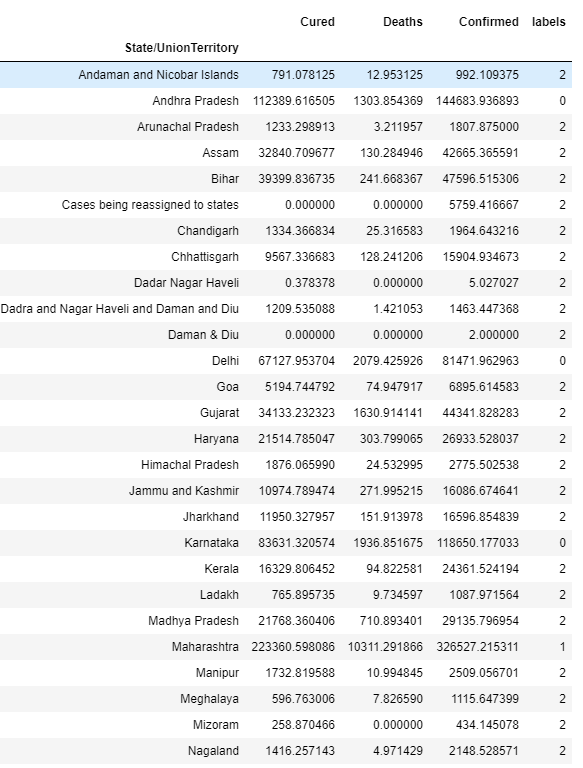
**These are the membership values of each point in the three clusters . We calculate the centers as below.(orange squares in the plot represent the centres of clusters generated).**



**From the clustering we find that 33 states/UT belong to cluster 2 ,5 states belong to cluster 1 and 1 state belong to cluster 1.**





1. **CONCLUSIONS**

We used the K-means, DBScan, Fuzzy c-means and Hierarchical Clustering algorithm, and learned its challenges and benefits. All models gave me the same results - three clusters, with states in least, mid, and highest development.

* Atlast we can say that urbanisation is a major cause of the increase number of confirmed cases of Covid-19 , also the less number of confirmed cases also mean that the tests conducted are less in these states and hence the deceased numbers are also less.
* The cured cases are found to be in sync with the confirmed cases, i.e. more the confirmed cases more is the recovery rate too.
* States like Maharashtra has all the values of mean Recovered, Deceased and Confirmed to very high wrt to other states all because of the higher rate of urbanisation in this state .
* More than 32 states/union territories are perfroming mediocre while the remaining 7 states/union territories are performing very poor.
* The worst hit states are  Maharashtra, Karnataka, Kerala, Andhra Pradesh, and Tamil Nadu.
* The smaller states like Goa, Manipur and Arunachal Pradesh have higher recovery rates.
* So inorder improve these results :
* More tests need to be conducted to increase the recovery rate (like Maharashtra, Kerela)
* Social distancing should be maintained
* Interstate travel should be reduce

1. **FUTURE WORK**

**Unsupervised learning** is a type of [machine learning](https://en.wikipedia.org/wiki/Machine_learning) that looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision. In contrast to supervised learning that usually makes use of human-labeled data, unsupervised learning, also known as [self-organization](https://en.wikipedia.org/wiki/Self-organization) allows for modeling of [probability densities](https://en.wikipedia.org/wiki/Probability_density_function) over inputs.

Two of the main methods used in unsupervised learning are [principal component](https://en.wikipedia.org/wiki/Principal_component_analysis) and [cluster analysis](https://en.wikipedia.org/wiki/Cluster_analysis).

For the cluster analysis that we worked upon can be extended with other datasets. It can WHO’s world data set and hence can cluster countries and extend our study . It can give us a very useful information about how different countries are handling covid-19 pandemic and can thus can use the policies they are using to better our country’s situtations,

Datasets with many other feautres could be used to enhance the accuracy and analysis of our clustering .

1. **REFERENCES**

We have taken the help of following websites during the course of our project which has helped us in making our project successful. We have learned how to do clustering in python.

* <https://www.geeksforgeeks.org/clustering-in-machine-learning/>
* <https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/>
* <https://www.geeksforgeeks.org/ml-fuzzy-clustering/>
* <https://pythonhosted.org/scikit-fuzzy/auto_examples/plot_cmeans.html>
* <https://pypi.org/project/fuzzy-c-means/>
* <https://towardsdatascience.com/>
* <https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html>
* wikipedia.com
* <https://scikit-learn.org/stable/>
* https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.divide.html
* <https://www.kaggle.com/sudalairajkumar/covid19-in-india>
* https://pandas.pydata.org/