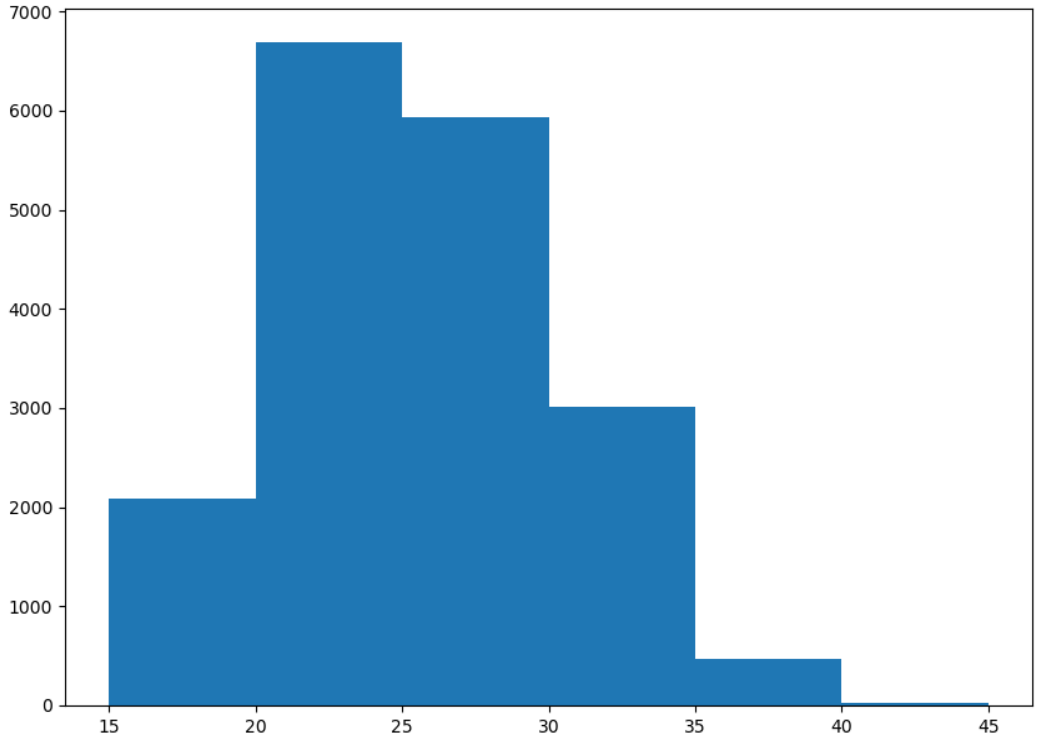
Data Analytics
Project - 1 Clustering
By:- Kartik Garg(2019101060) and Shreyash Rai(2019101096)

Q1.) Data Visualisation

a.)

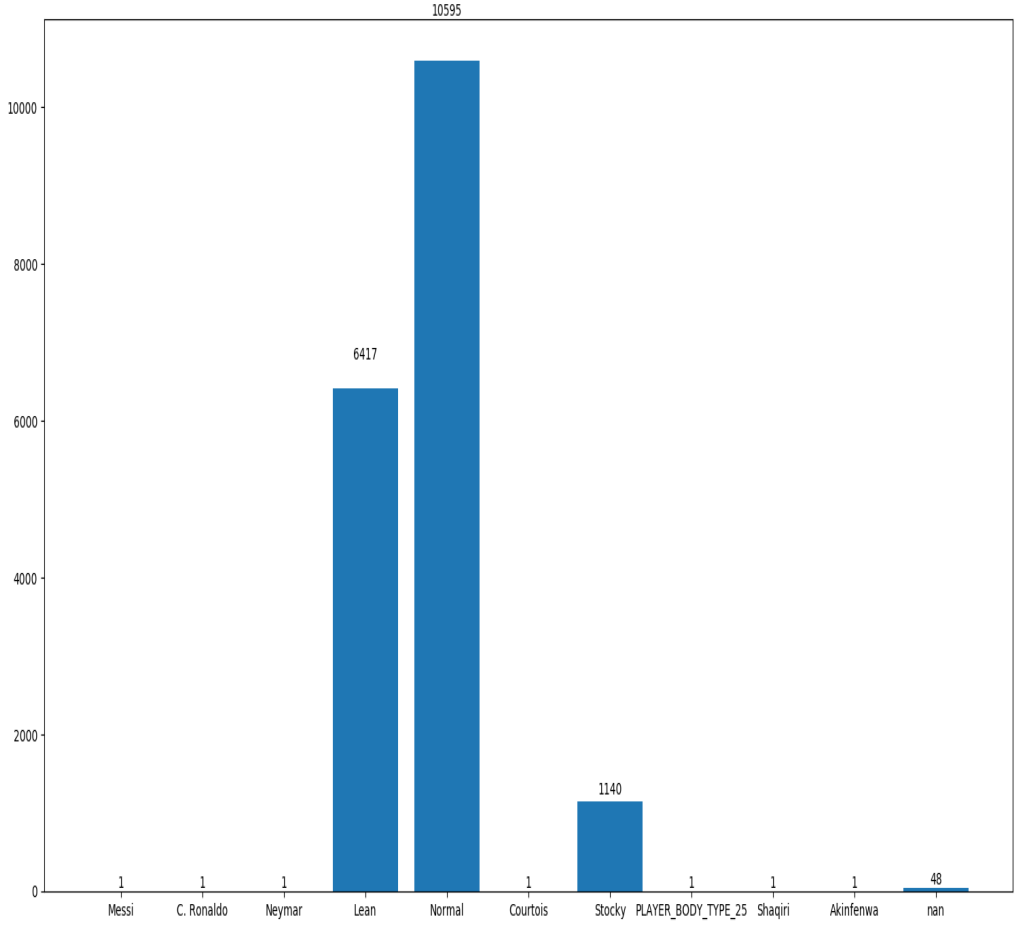


The above image is the age distribution among the players in the given data set.

The size of each interval considered is 5, and it is very easily observable that the **majority** of the **players** lie in the **mid-age** range that is **20-30.**

The png of the this graph named as ageDistro.png can also be found in the pyplots directory.

b.)

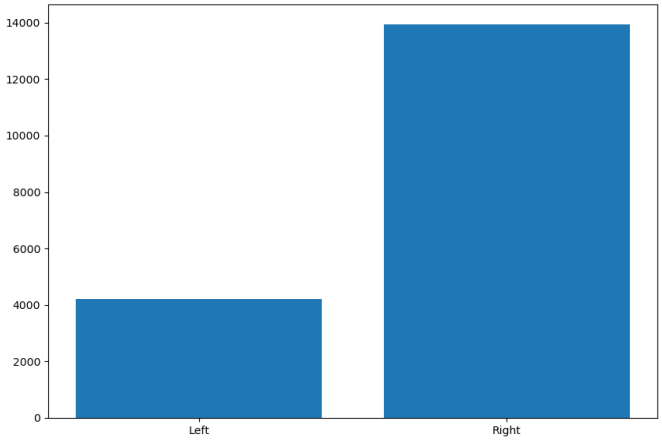


The above image is the histogram demonstrating the distribution of players in the different body types.

We can clearly observe that the **outliers** like Messi, Ronaldo and Neymar are themselves **given a different body type** named on their first names respectively.

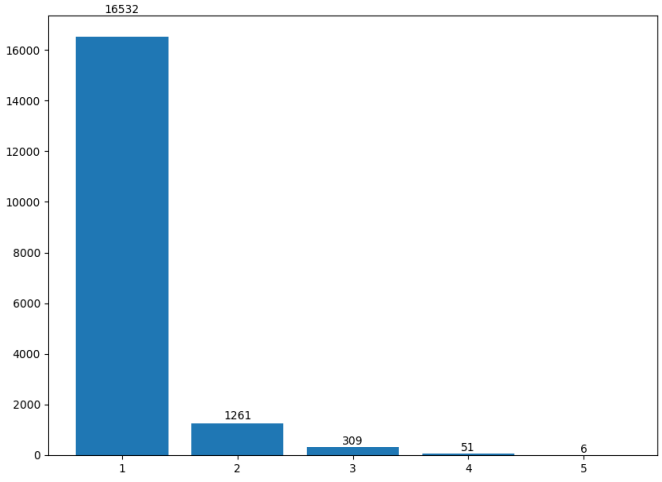
The png of the this graph named as bodyTypeDistro.png can also be found in the pyplots directory.

c.)



The above image is the distribution of the players on the basis of whether they are left footers or right footers.
We Observe that the data is in accordance with the data available online as they say around 15-30% of the world population is lefty. The same can be observed in this dataset.
The png of the this graph named as footDistro.png can also be found in the pyplots directory.

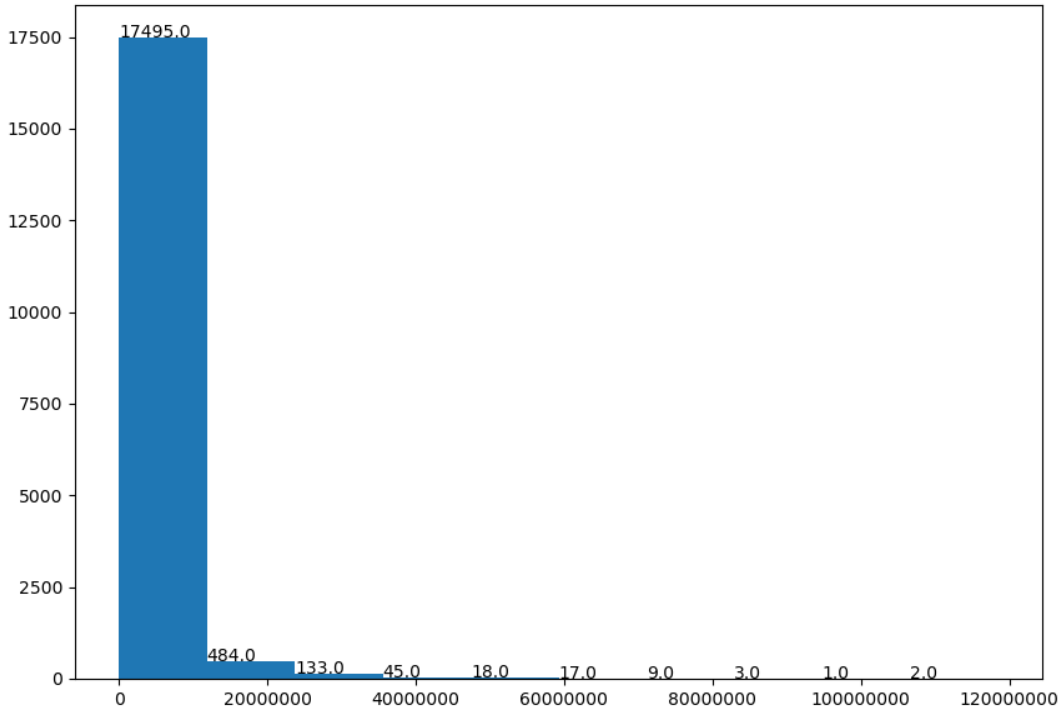
d.)



The image on the left is the distribution of players in the different reputation levels ranging between 0-5 in the given dataset.
We can observe that the outliers are also prominent here, for instance the players having reputation = 5 are only 6, out of which 3 are Messi, Ronaldo and Neymar.

The png of the this graph named as reputationDistro.png can also be found in the pyplots directory.

e.)

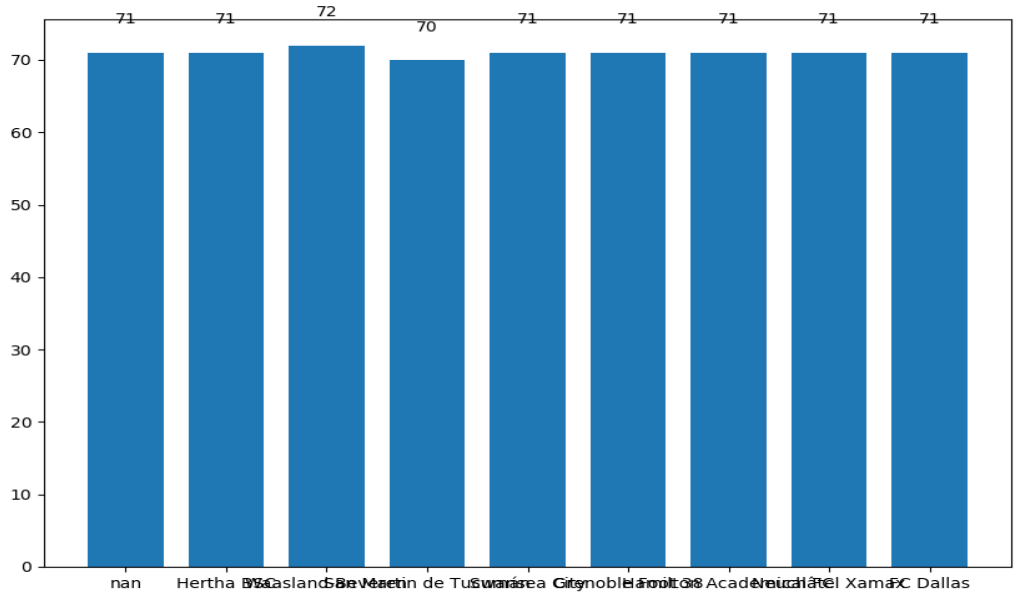


The above is the graph demonstrating the distribution of the players among the different value brackets. To draw this graph i took the highest value that was present in the dataset and made 10 bins out of it to plot histogram.

We can observe that 17495 players out of the 18207 players lie in the first bracket that is 10% of the maximum value that is earned by some players (i.e 2). The outliers can be very easily spotted, the last 7 bins more or less correspond to the outliers. Messi, Ronaldo and Neymar solely occupy the top 2 bins.

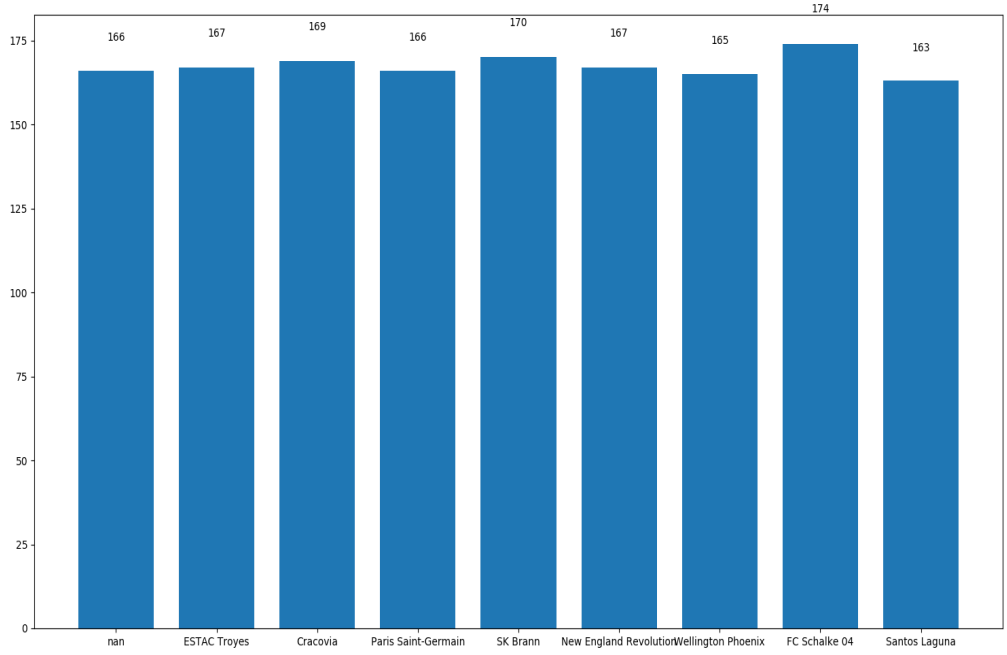
The png of the this graph named as valDistro.png can also be found in the pyplots directory.

f.)



The above graph shows the average height in inches for some randomly chosen clubs, here we analyse that over these 9 clubs that are listed above that the average height lies between 70-71 inches. So without loss of generality we can say that the average height lie around 5'10"
The png of the this graph named as heightvsClubDistro.png can also be found in the pyplots directory.

g.)



The above graph shows the average weight in pounds for some randomly chosen clubs, here we analyse that over these 9 clubs that are listed above that the average weight lies between 163-174 pounds. So without loss of generality we can say that the average weight lies around 169 pounds.

The png of the this graph named as avgWeightSomeClubs.png can also be found in the pyplots directory.

Q2.) K-Means Algorithm

a.) **K Means** algorithm is implemented from **scratch** and can be found in the kmeans.py file in the code directory.

The defined function takes an **argument** that is the **dataset** given.

Then firstly in this function I **preprocess** my data in order to have only numerical attributes as suggested in the requirements document. Preprocessing includes converting **height** to inches in order to treat it as integer, similarly for **weight** removing the lbs from each value was the task.

I also needed to remove the **"+2"/"+3" terms** (similar) from the attributes ranging from **[28,53]**

That was all about **preprocessing** .

Then in the implementation of the algorithm is there, which works on this preprocessed form. **To limit the computation time** the number of iterations are limited to **50**, that means if we encounter zero cluster range before 50th iteration then its good otherwise my algo automatically stops clustering after 50th iteration and it will go on to calculate the Mean squared error of the clusters formed and subsequently it will export the formed clusters as a json file.

For calculating **distances**, i am **not using the cartesian distance** instead i have defined my **own metric** to calculate similarity subsequently distance. For each of the 70 numerical **attribute**s I have **defined a range** over which two values can be **considered the same**. For **instance**, if we consider weight then in that case I have kept the range to be +-5, that is if the weight is 170 lbs then any weight between 165-175 will be considered the same as 170 which will in turn contribute 1 towards overall similarity.

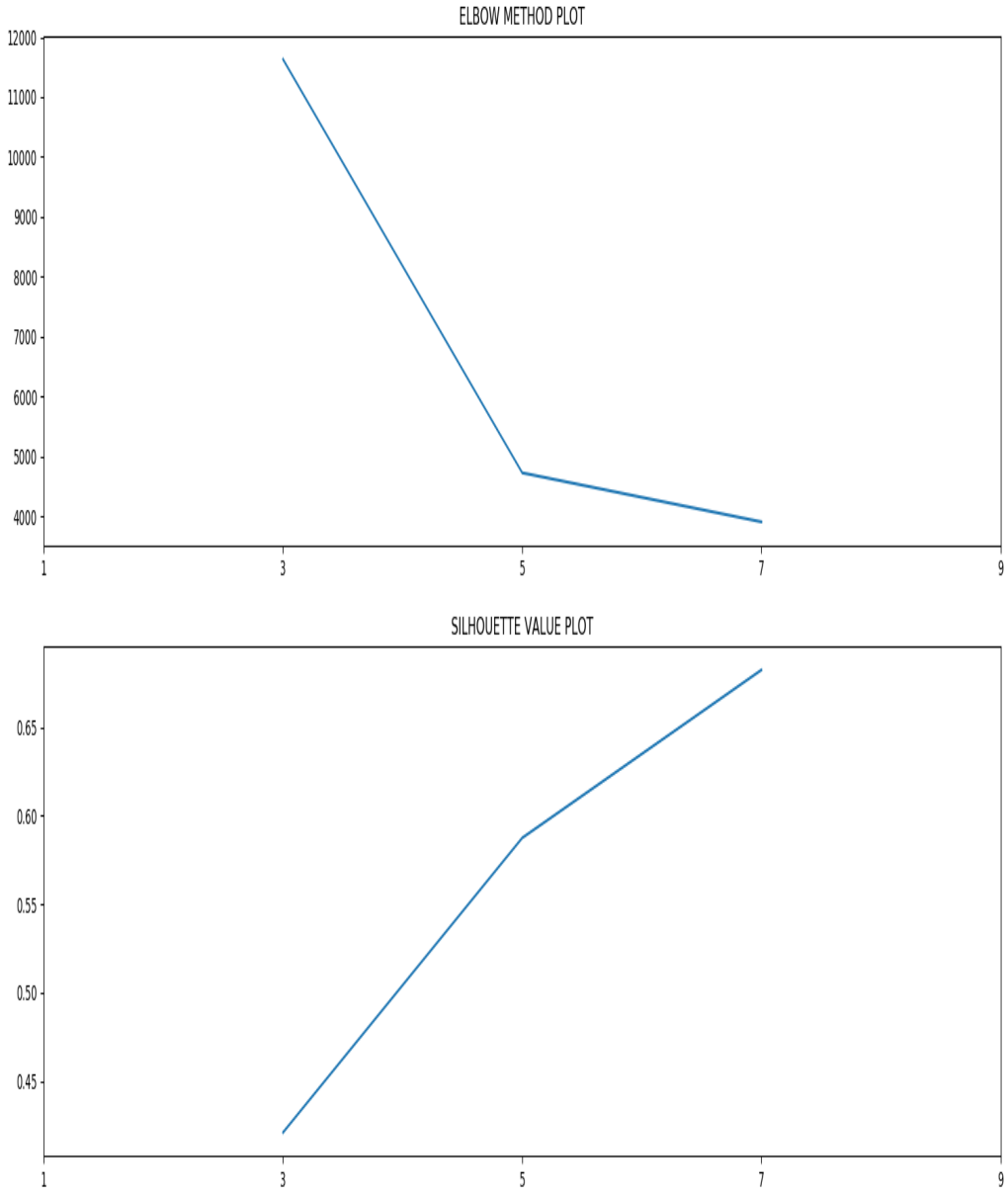
The same can be observed in the getSimilarity method of kmeans.py file, although I have defined a different metric for jersey number and it is also evident in the mentioned method.

For distance calculation I calculate **similarity ratio that is similarity/Total_Attributes**, then I **subtract it from 1** to calculate **distance** which will be used in calculating MSE after squaring it.

b.) For k = 3,5,7; i have made my code to pre run and the corresponding clusters formed are present as .json files in the code directory. The naming format is "CLUSTERS_K_T.json"
Where k is the value given as the argument to k means and T is the try number, at present I have json files 2 tries for each of the k values.

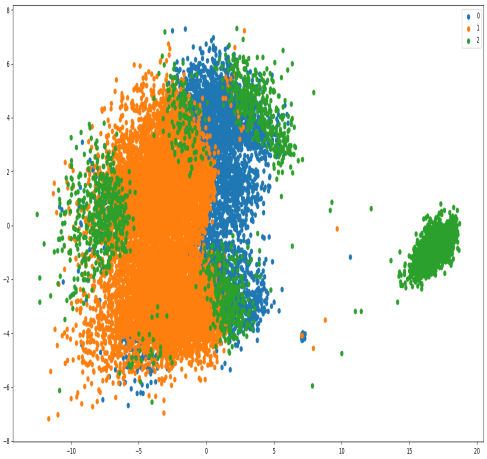
c.) Following from the previous part, the best k that i can guess from the **elbow method is k = 5** and from silhouette method it will be k = 7
I will go with the **silhouette method and say optimal k is 7** out of the 3 values that we were given.
Since for calculating **silhouette values we need O(n^2) complexity**, and since we cannot save the distance between each pair of data points due to the memory issues, which in turn means that we need to calculate the distances on the go again and again which is approx O(M), and M is around 70
Therefore total time complexity for a given set with 18000 data points will be of the order O(10^10), so in order to **reduce it** I have **reduced my data set by choosing some data points** instead of all of them. And without loss of generality we can say that it won't affect my overall calculation since in the end I only want the average of the individual silhouette values of each data point. So reducing the data set seemed fine to me in order to compute the value in reasonable time.

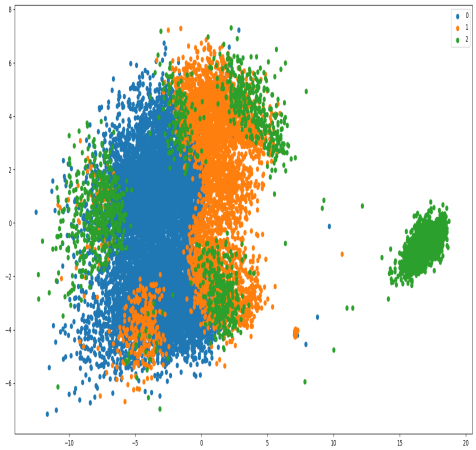
GRAPH:



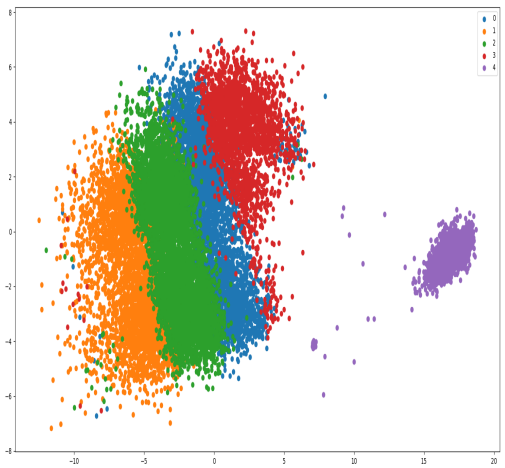
The png of the this graph named as elbow_silhouette_plot.png can also be found in the pyplots directory

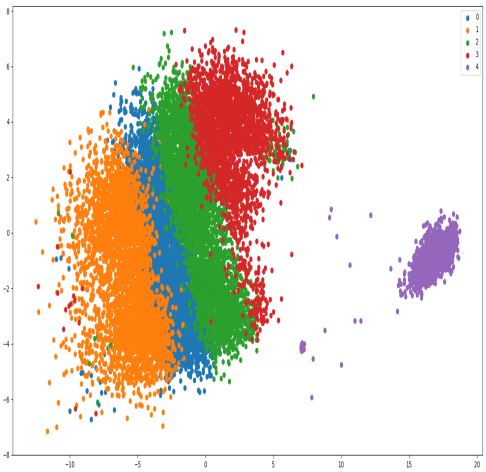
d.)



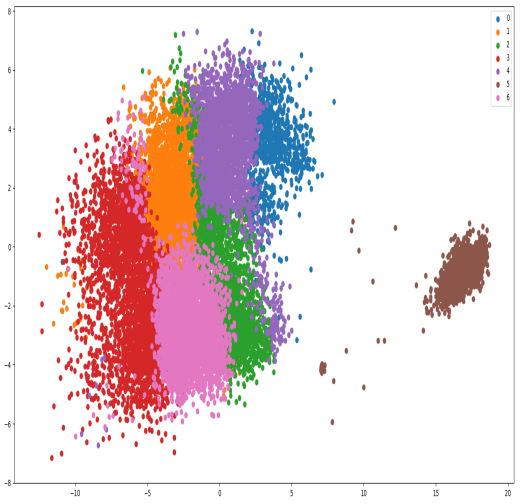


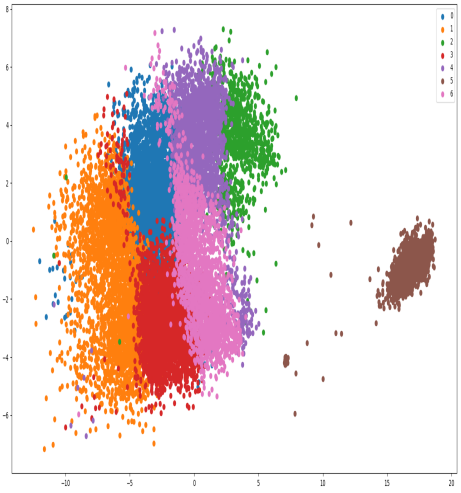
For k = 3, as we can see that three clusters in both the tries are formed and in both of them we can observe that in the larger collection we have a green border and the smaller collection is fully green. If we draw an analogy of this clustering with DBSCAN then we can relate these points colored green as outlier and in the DBSCAN section we can see in the graphs the envelope is marked as -1 in the index which is the scipy standard.





For k = 5, we can see a very clearer/prominent distinction when compared to k=3 visualisation. On first observation I thought k = 5 will be the best k for this dataset which is in fact in accordance with the elbow value plot but the silhouette plot deduces something different. One reason to justify this is that in k = 5 clustering we no doubt had clearer/better clusters but still intracluster differences were still high which led to the increment of the silhouette value when compared with k = 3 but it further increases on going to k = 7, precisely the increment of the b factor in silhouette formulae.





For k = 7, this is best cluster formation according to the silhouette and no doubt we can again clearly see marked clusters in the visualization.

Q.3)

a.) Clustering based on Agglomerative Method

The code file is bottomUpBottom.py

The different images attached in this section can be found in the pyplots directory with naming mechanism as follows

"agglomerative_bottom_up_K.png"

Where k is the intended number of clusters.

For this I used **sklearn's inbuilt agglomerative clustering library**. In order to **speed up** the process of clustering I required r**educing the dimensions** in the given data set. Since I am

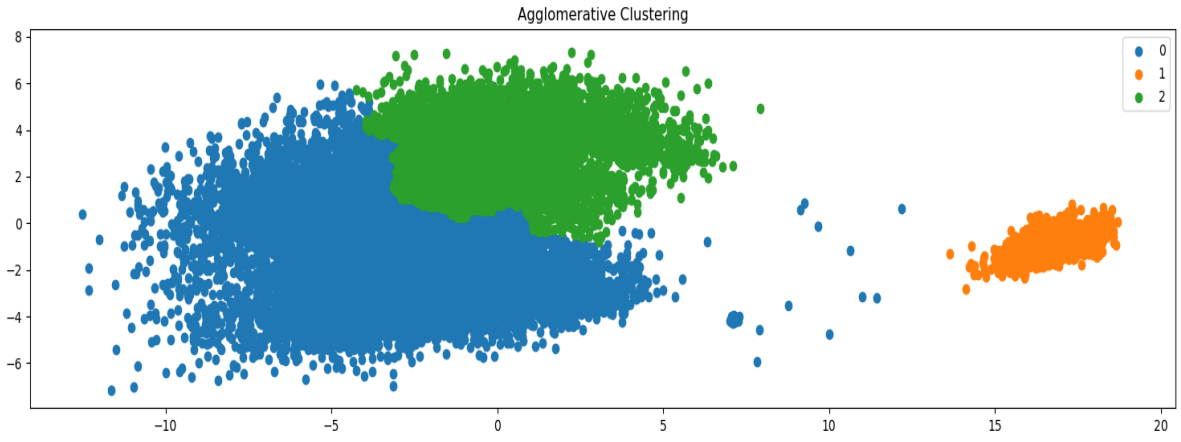
seeking **dimensionality reduction**, I went on with **using PCA** to reduce the number of dimensions such that I can **easily plot** the graphs too. As a matter of fact all the graphs representing cluster visualisations have **x coordinate** and **y coordinate** as the two most **significant components** as given by the PCA.

Now the agglomerative clustering offered by sklearn has this **default** thing that it will take **euclidean distance** so at first i decided to provide a **custom function** in order to cluster my whole original data set but it was **taking way too much computational time** or in layman terms the computational cost was too high. But when I applied **PCA** I realised that since now I have **two primary components** that I am considering so it seemed more obvious to me to use **euclidean distance** to cluster whereas originally as in K Means algorithm i had my own metric defined.

One Question might arise that **how** am i gonna **compare** the two clustering methods since the distance metric is different in both of them but then i thought **that's what i am analysing** how the clusters are formed and which one will give me the best results.

For report purposes I have used the final number of clusters to be formed as 3,5,7 as they were suggested in question 2 as well and one more supporting reason behind choosing it is that since ultimately I am gonna compare all these methods so a similar number of clusters will assist my comparison.

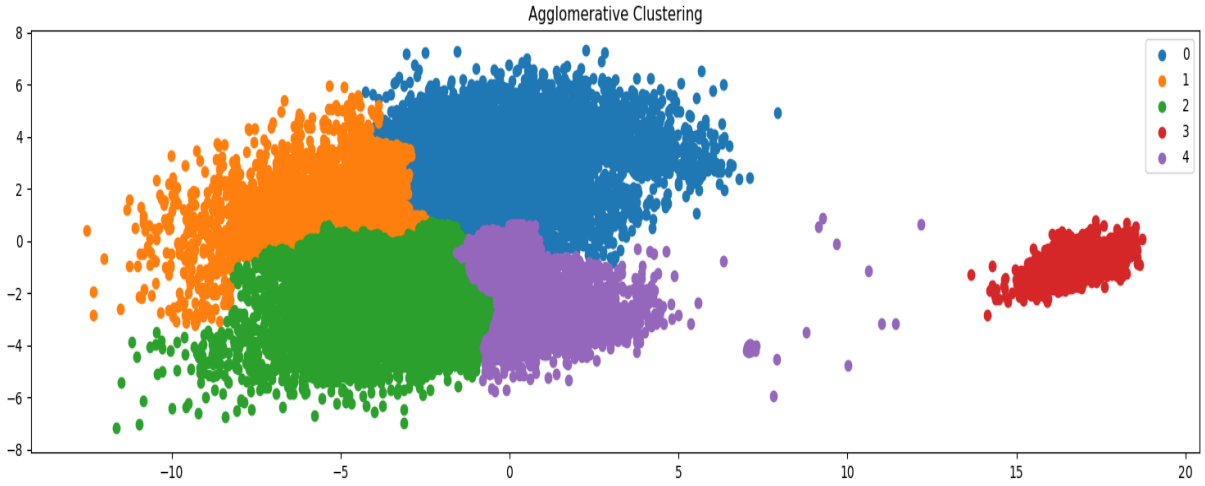
For k = 3, agglomerative clustering on data after dimensionality reduction gave the following graph [CLUSTER VISUALISATION]



On comparing this graph with the one given by K Means algorithm we observe that this is nothing but a more clearer version of the K Means one. Like in this we can clearly see the

cluster boundaries whereas in the K Means one the clusters were overlapping where they were joining that is diffusing into each other. So we see that this graph is in accordance with K Means.

For k = 5, agglomerative clustering on data after dimensionality reduction gave the following graph [CLUSTER VISUALISATION]



However for k = 5, on comparing this given graph with K means we find that now the clusters formed are pretty different in nature like in the graph for K means we see that the clusters are vertically oriented that is they span more on y axis when compared to x axis. But in agglomerative clustering we see that now the clusters are more x - axis oriented as compared to y axis . One major reason that we think behind it is the change of distance metric in two clustering mechanisms as the second one that is agglomerative is the one using euclidean distance so it make sense to have more globular structure whereas in the K means it is the similarity which can somehow be related to manhattan distance.

So proposing an analogy to support the results obtained, consider 3 points

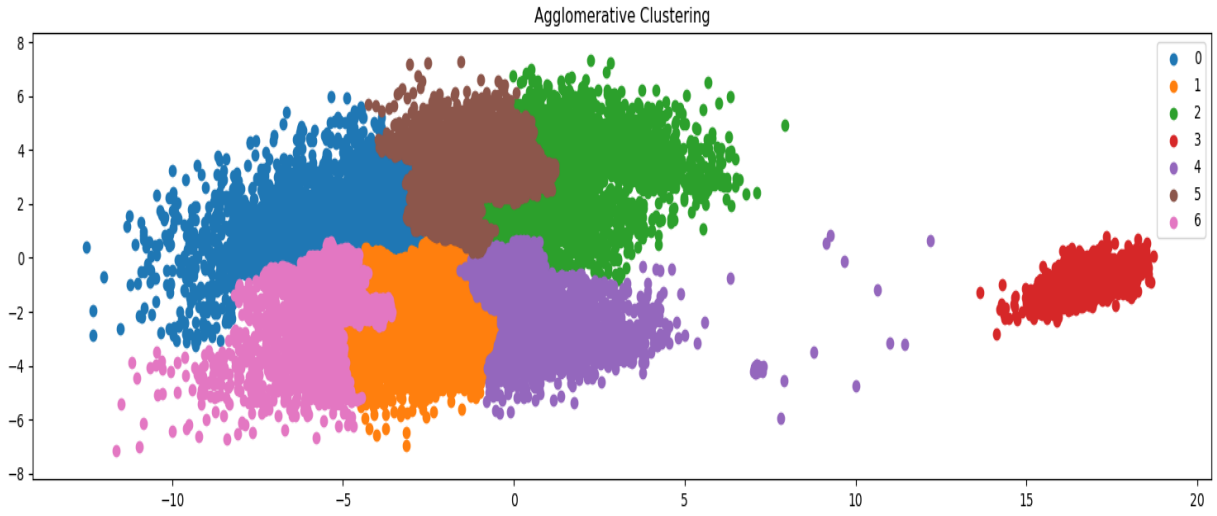
(0,0)

(3,4)

(0,5)

If we consider (0,0) as our cluster mean and we calculate distances to other two points then euclidean distance will be same that means both of them lie in the same cluster when taken (0,0) as reference whereas manhattan distance will be 7,5 respectively which means there are more chances of the third point to be in the same cluster.

So what i intend here to discuss is that when considered similarity over euclidean distances clusters are more likely to be vertically oriented as we noticed above that 0,5 is more closely similar to 0,0 when compared with 3,4 [Referring to manhattan distance]
For k = 7, agglomerative clustering on data after dimensionality reduction gave the following graph [CLUSTER VISUALISATION]



We can see the 7 clearly globule type clusters formed in the above picture. But here we see that this is different from the previous case, in accordance with the graph given by K mean algorithm for k = 7. This is because by increasing the required number of clusters the Manhattan distance gives approximately the same results as the euclidean metric.

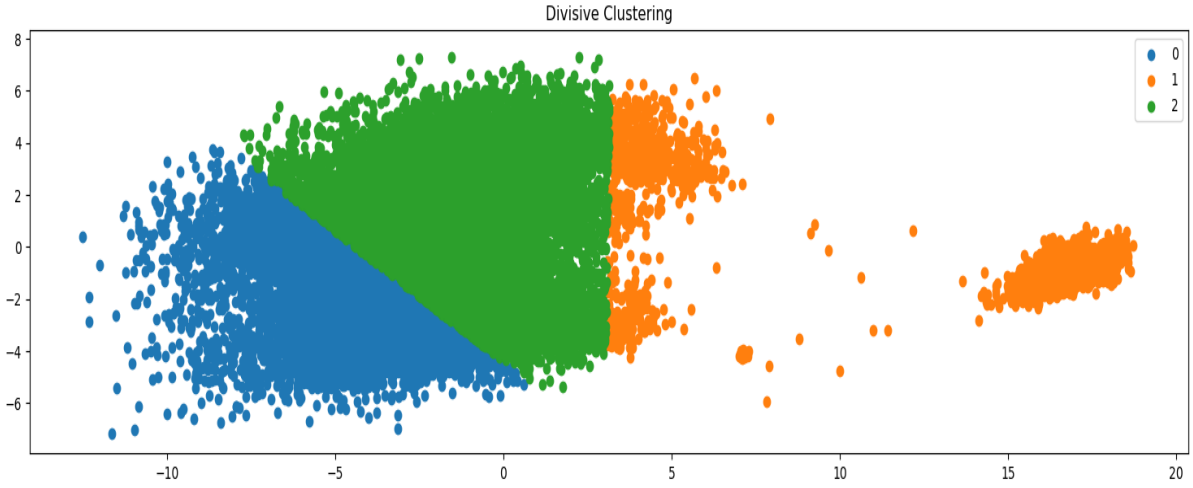
b.) Divisive Clustering

For this part I tried to stick to the method taught in class by sir rather than the method suggested by TAs in one of the mails over moodle to approach this part.
What I do is I take all the points initially in a single cluster and then in this cluster I look for the extremes and for these two extremes I recluster the data points in the cluster into two different clusters.
Now let's say I had N clusters, so first I get the maximum distance among any two points of each cluster and I pick the one with maximum distance and then I apply the above reclustering over this cluster. The mechanism described above yields K+1 clusters after an iteration if it had k clusters at the beginning of that particular iteration.

So yes, that's how i am approaching to this part of the problem
For comparison purposes I am considering the number of clusters to be formed as 3,5,7 so that I can compare the different results obtained although the number of clusters formed can be easily modified in the code by changing the value of a single variable.
The code file corresponding to this is bottomUpBottom.py
The different images attached in this section can be found in the pyplots directory with naming mechanism as follows

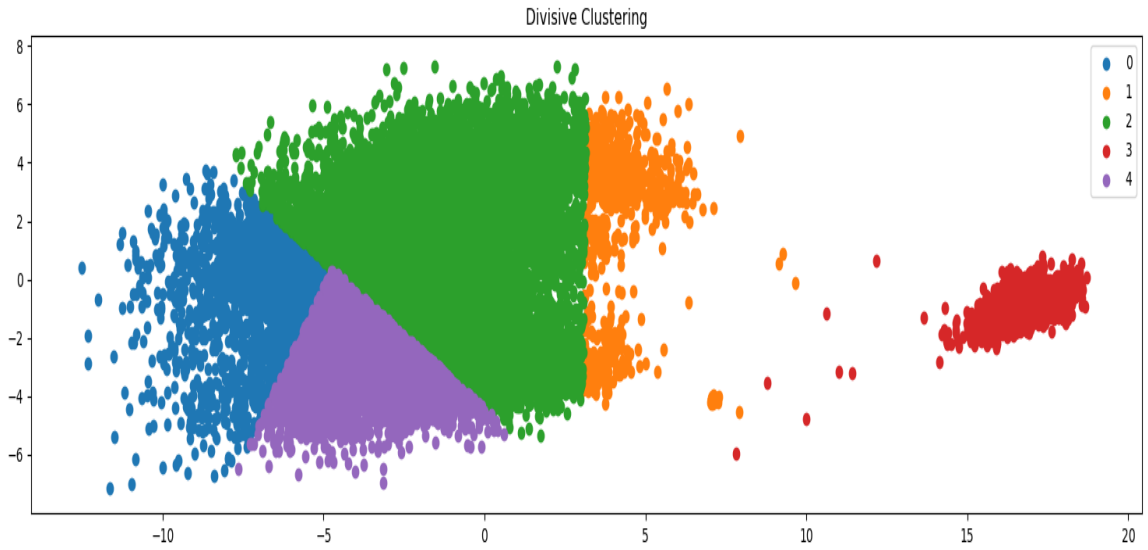
"divisive_up_bottom_K.png"

Where k is the intended number of clusters.
For k = 3, we got the following cluster visualization



In order to reduce computational costs this algorithm is also applied on the reduced dataset (dimension wise) given by PCA, so again the distance metric chosen is euclidean so we can expect similar results to agglomerative clustering and on observing the image we can clearly conclude that it is in fact the case. The clusters do have more of linear boundaries and are very well defined but the cluster formation is kind of similar to that of agglomerative and K-means.

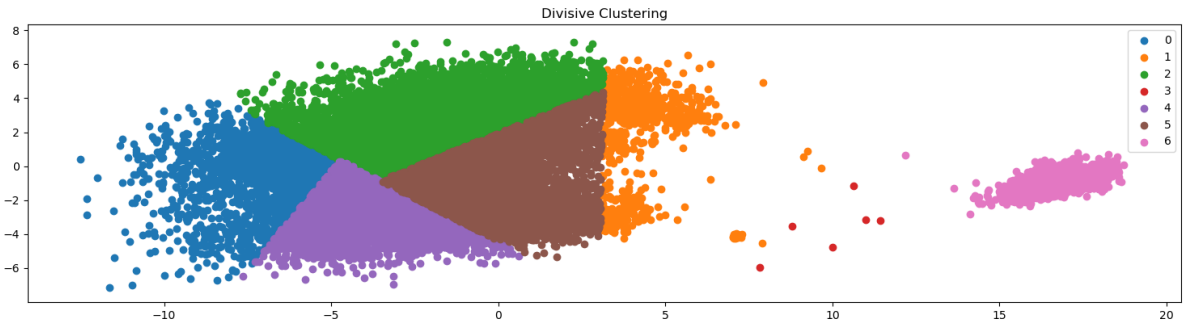
For k = 5, we got the following cluster visualisation



Since, the clustering is based on the maximum distance between the extremes of a cluster, so in accordance with this fact we can see that the cluster 0 in k = 3 had the maximum separation which subsequently underwent division and formed 2 clusters marked as 0 and 4 in the above image.

Following the same discussion the cluster 1 in k = 3 underwent division to form cluster 1 and 3 in the above image. On comparing it with the graphs of agglomerative and K Means we see that this graph is a combination of both of them which is due to the fact that I am considering the maximum separation in each cluster and taking the global maxima among all these to select the cluster which undergoes division now. To improve a bit I can try over other methods like taking the global maxima over the average distance in each cluster. That might change cluster formation to a greater extent.

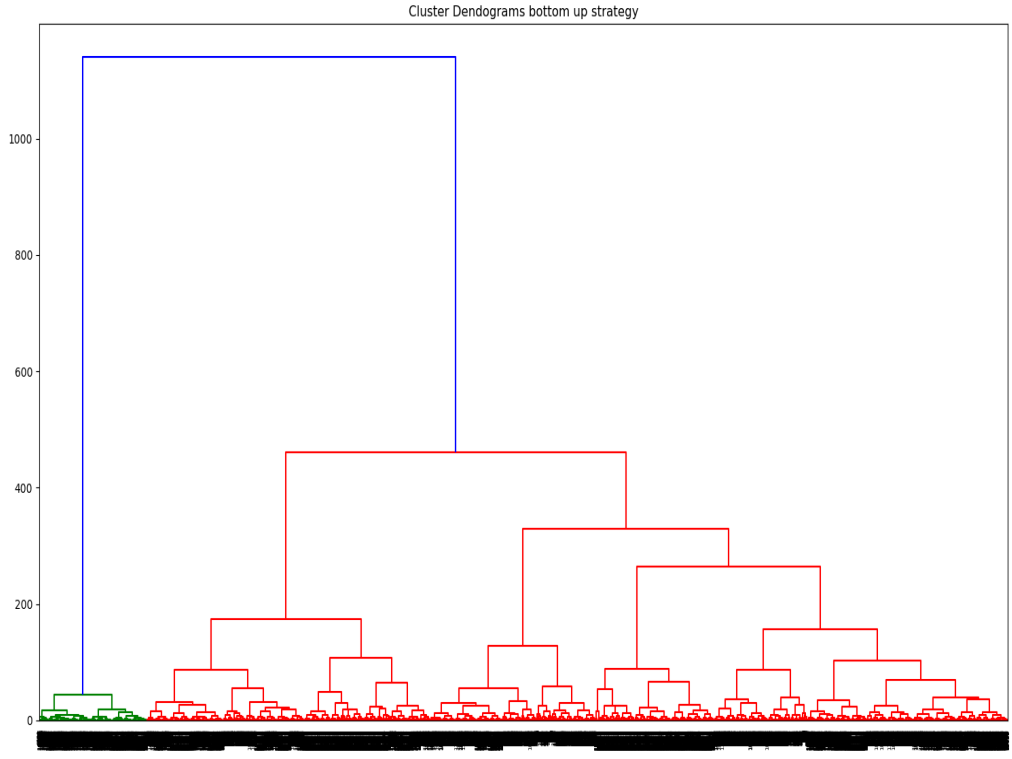
For k = 7, we have



The story here is similar to what we had in k = 5 case

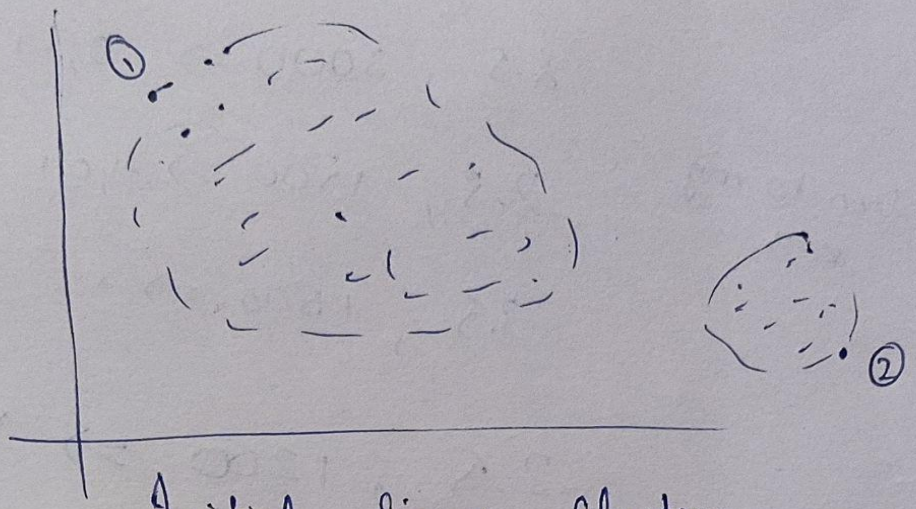
c.) Dendrograms and comparison, comparison is done on the go where each of the cluster visualisation is attached

Agglomerative Clustering Dendrogram



*given by the inbuilt python library

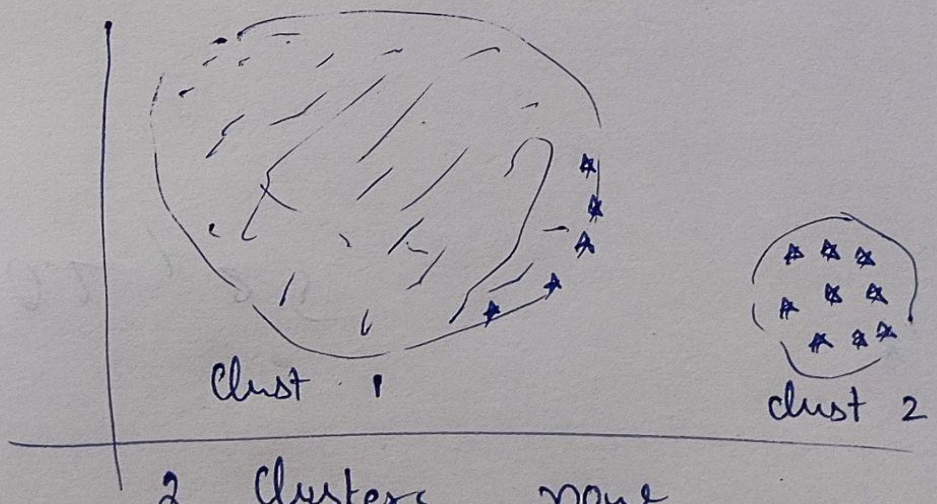
Divisive Clustering Discussion about dendogram formation



Initialy Single Cluster

①, ② => most distant points i.e extremes

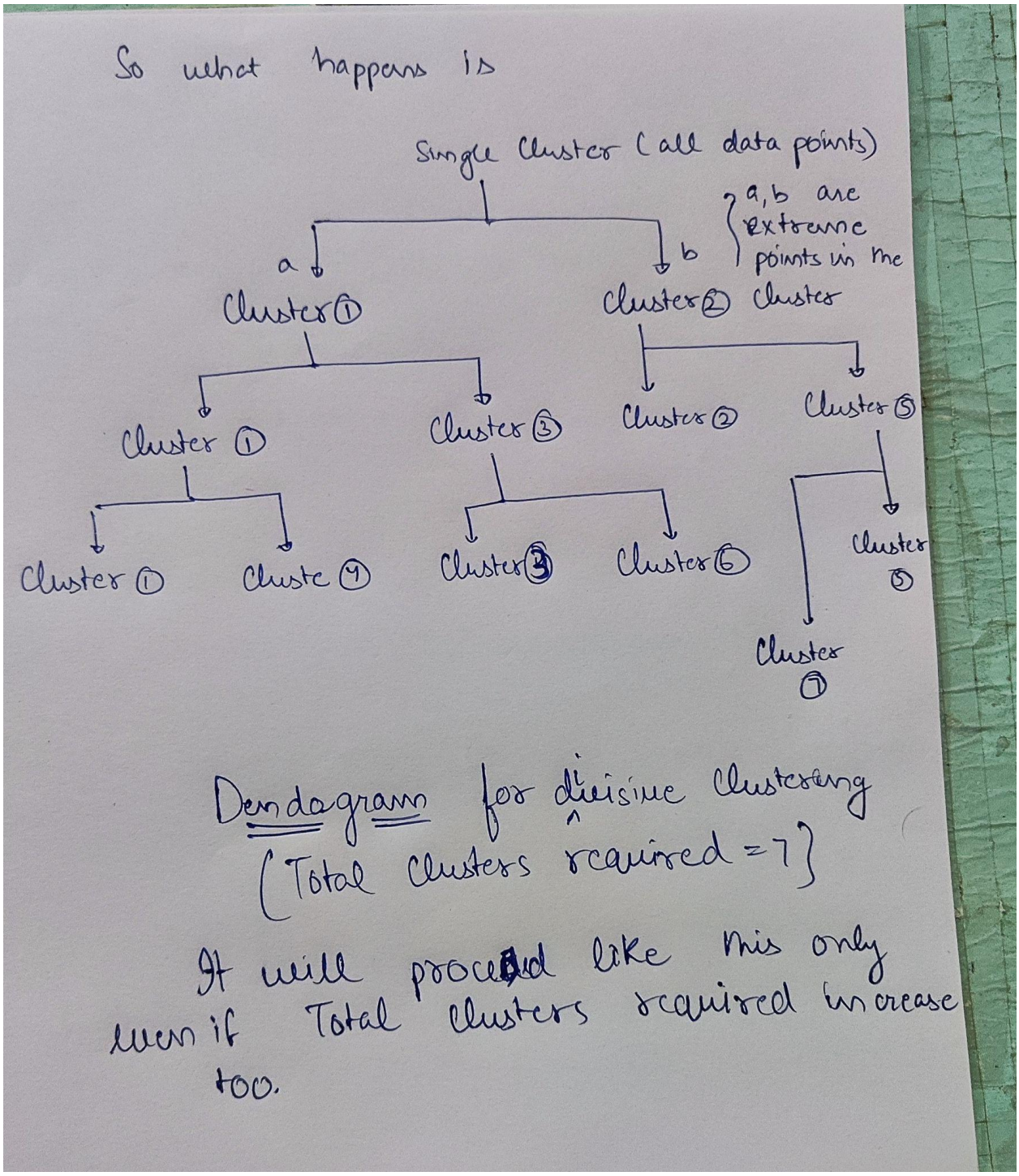
So recluster we get



2 Clusters now

it Further disintegrates (Cluster 1)

Dendogram for divisive Clustering:-



Q4.) DBSCAN

For dbscan algo, I have used the inbuilt python library.

On analysing the results for various values of epsilon and minPts for the algo i came to the conclusion that this algorithm will be worse to consider in clustering the given data set and after dimensionality reduction we see that the points are really dense in the center of the figure so the algorithm will definitely cluster them together rather than observing other similarities that might arise in the data set.

For this algorithm, I tried around 15 different value pairs of epsilon and minPts. Out of which i have the cluster visualisation for 9 of them in the pyplots directory.

Naming convention followed is

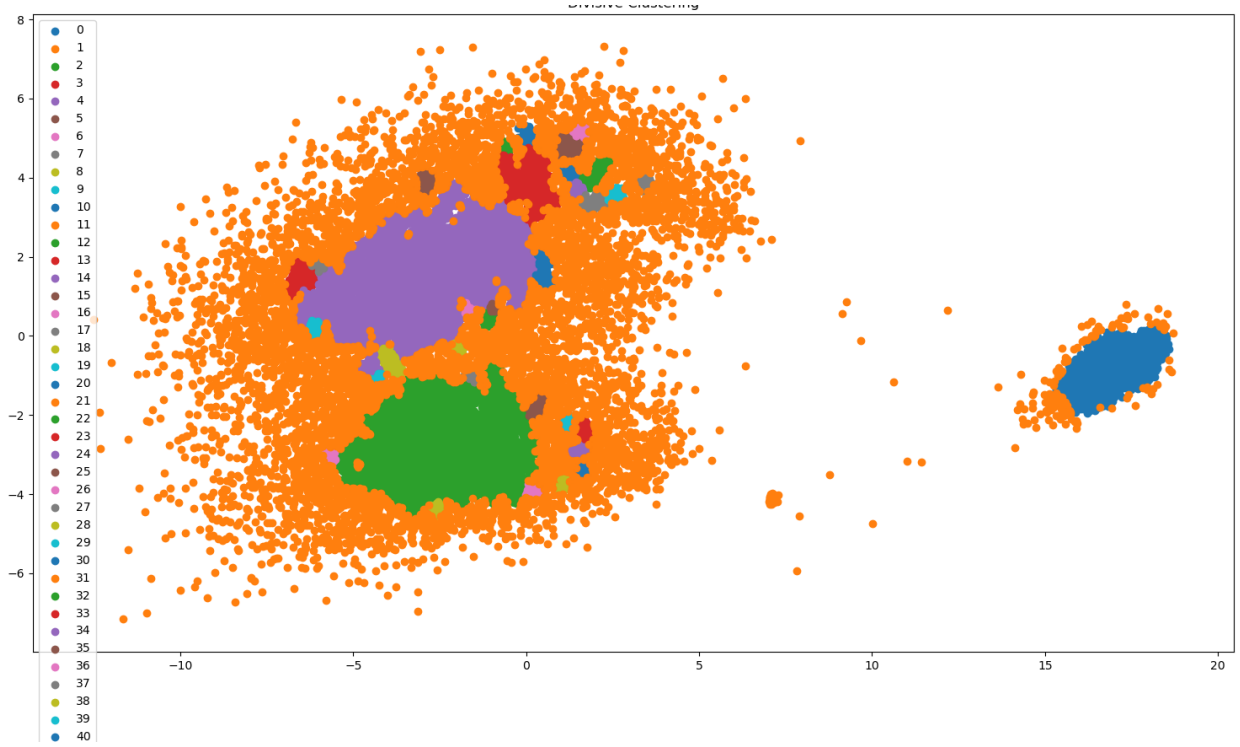
Let's say epsilon is x.y [that is if epsilon = 1.35 then x = 1 and y = 35]

And minPts is z

The corresponding plot for that is named as

"dbscan_x'y_z.png"

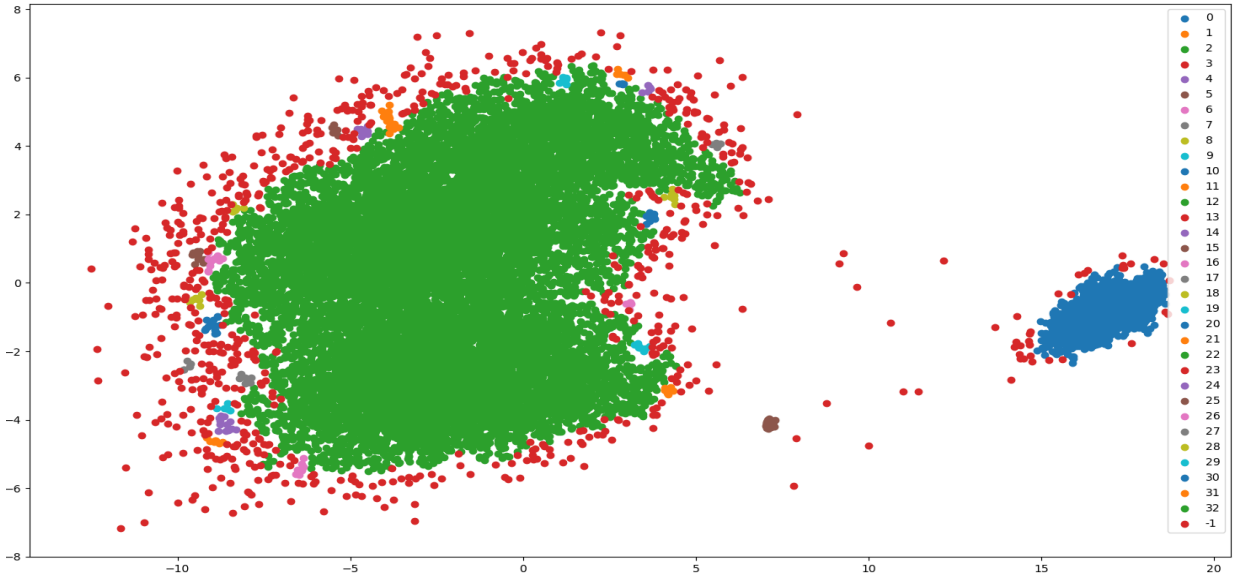
The best plot i think i have is for epsilon = 0.2 and minPts is 25



In this image we can see that we have around 40 clusters formed, some of them being very small in size. But now when i increase my minPts further that is in want my clusters to be

more dense then this graph starts shifting to a scenario where most of the border points of the bigger collection are part of outliers that "-1" cluster as the dbscan can not cluster them according to the given epsilon and minPts. That basically means that the number of core points have reduced.

Subsequently if we reduce the minPts value then the smaller clusters that are visible now start merging together to form a bigger cluster, like if we consider epsilon = 0.2 and minPts as 6 thent he visualisation obtained is



We see that reducing minPts have made the collection bigger as one single cluster.

Now on increasing epsilon and number of points we kind of shift towards the previous graph but with a lower number of clusters.

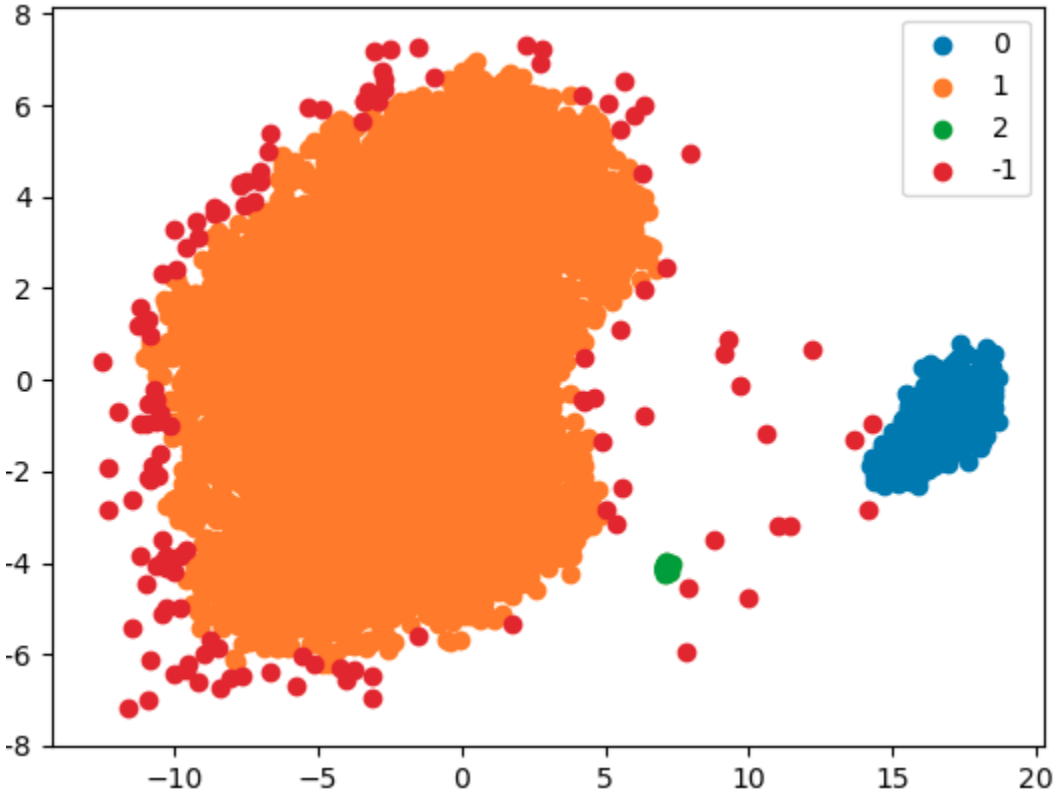
Basically the issue is the points near the center of the dataset are very dense and the data points near the border of the collection are very sparse. So what happens is if we increase minPts then either the border points form very small clusters or they become part of the "-1" cluster.

And on decreasing minPts, they get merged with the center as the number of core points is way too high near center.

Now if we consider the central part, increasing minPts reduces the cluster size. Basically it demands a more dense structure which results in cluster compression. But on the other hand on decreasing minPts value the central part becomes a single cluster in itself as the central part do have a very dense structure overall so instead of increasing the number of clusters on

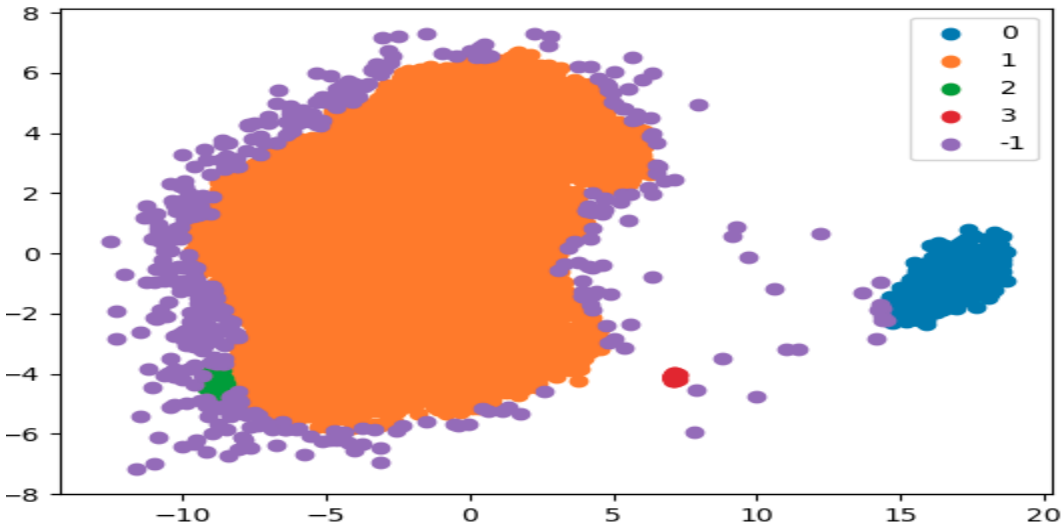
decreasing minPts they start lying in each others epsilon distance and every other point becomes a core point as well which results in a bigger cluster formation like shown in above picture.

Some more plots,

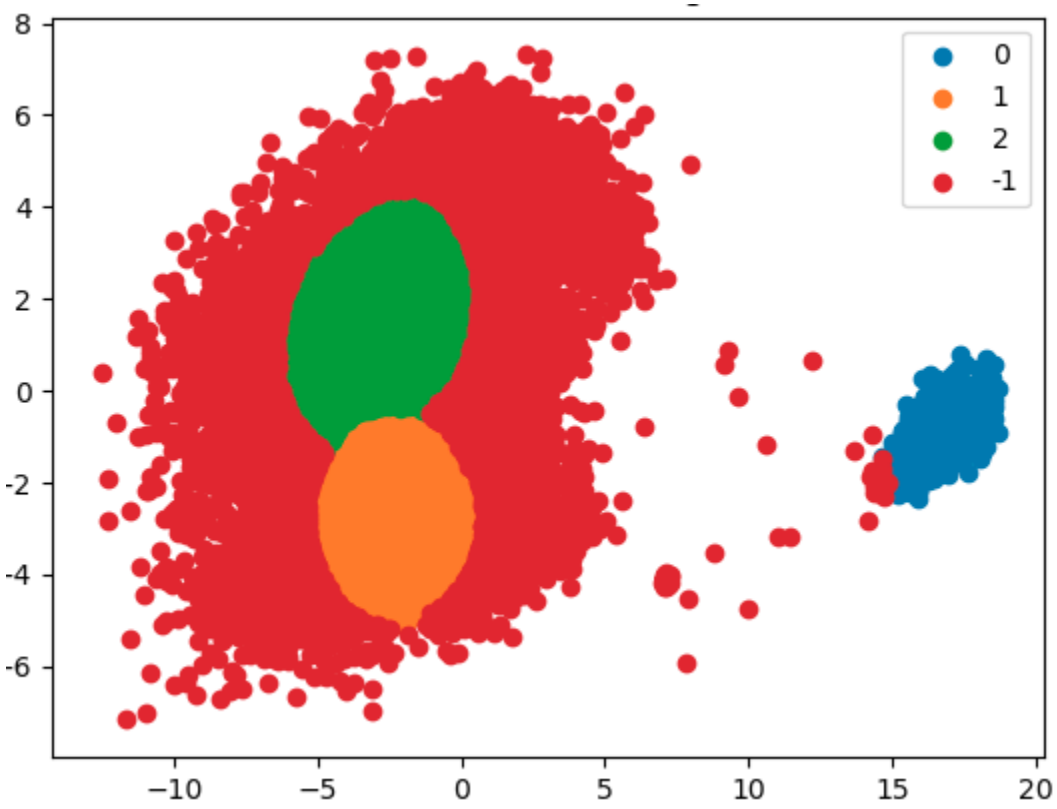


Epsilon = 0.5, minPts = 10

We observed that increasing the delta distance and keeping a low required density resulted in formation of a large orange cluster.



In this we increased the minPts value from 10 to 25 and kept the epsilon value as it is, it resulted in shortening of the orange cluster and most of the points near the border and not dense enough for both either to be a part of the orange one or to form new clusters.



There was this graph too with epsilon = 1.5 and minPts value as 1500

Most of the border points cant be clustered due to less density, and near the center where the dataset is pretty dense resulted in formation of two clusters.

Basically what we found is that dbscan only results in formation of either a single large cluster in the given data set or very small clusters which are not of any practical use to us.
So we deduce that due to the large variations in the density near the center and near the border dbscan is not very suitable for this dataset and we should stick to the other forms of clustering.
Dbscan only results in those clusters which are very closely related in case when clusters formed are high in number whereas when clusters formed are smaller then all the points qualify to lie in the same cluster.