Name: Tina Gholami **Report Homework 2** **Course: Data Mining**
Student Number: 9531307

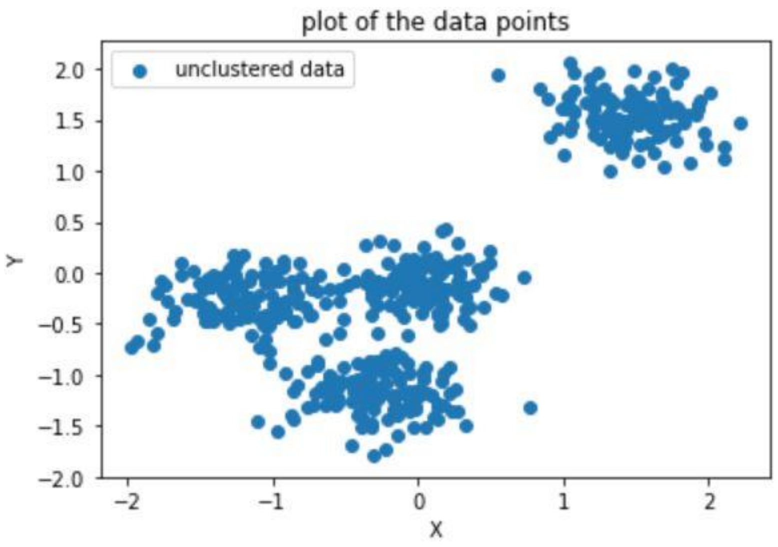
Theoretical Questions (1-5):

***** The solution to the theoretical questions (questions 1 to 5 in the first part of the assignment) are sent in the “report” folder and is named “9531307_TinaGholami_Solutions of theory questions2”.

Implementation 1: k-means Clustering Algorithm

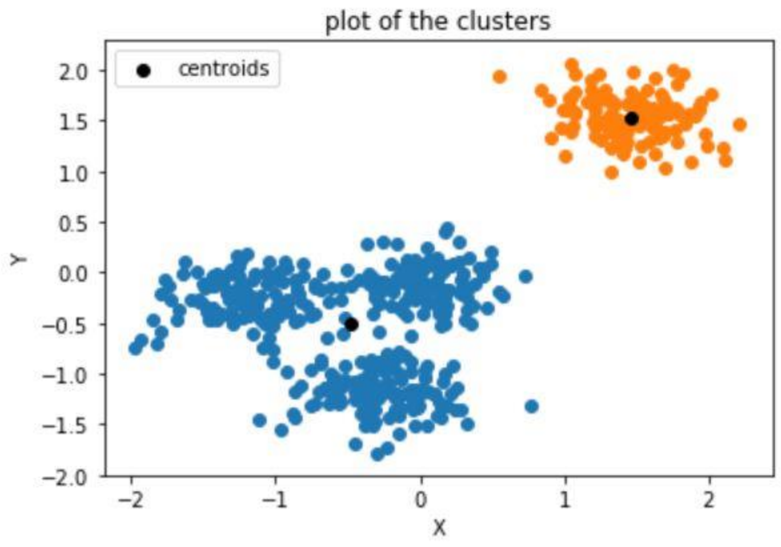
A) (The name of the file: A.ipynb)

The main data frame looks like:

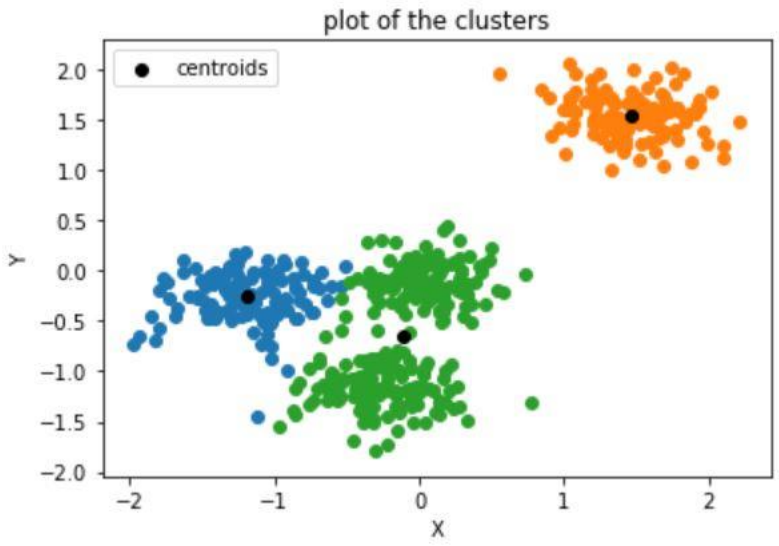


After running the k-means algorithm on 'Dataset1.csv', with 15 iterations, the output (clustered data frame) is:

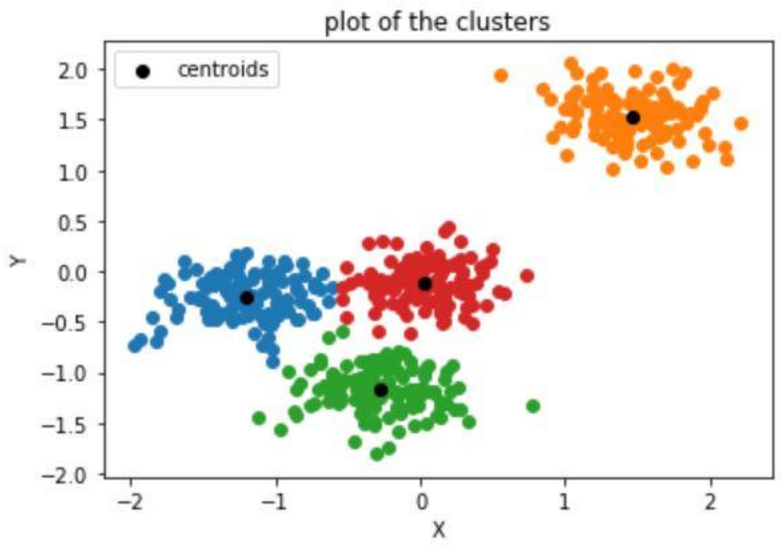
k = 2:



k = 3:



k = 4:



B) (the name of the file: B.ipynb)

We should add these bottom lines to the k-means algorithm code above, in order to calculate the 'cluster-error' for each cluster:

```
#calculating the 'cluster-error' for each of the clusters:
cluster_error_matrix = np.array([])
for i in range(k):
    sum_distances = 0
    for r in range(len(dic[i+1])):
        sum_distances += distance(dic[i+1][r][0], dic[i+1][r][1], centroids[i][0]
, centroids[i][1])
    sum_distances /= len(dic[i+1])
    cluster_error_matrix = np.append(cluster_error_matrix, sum_distances)

print(cluster_error_matrix)
```

and the output ('cluster-error' for each cluster) is: (where each element in the array corresponds to each of the clusters. For instance, for k = 4 we have:)

```
array([0.32383689, 0.31600882, 0.33612332, 0.28865974])
```

C) (The name of the file: C.ipynb)

We only need to add these lines, which is simply a mean, to the code in part (B):

```
#Calculating the 'clustering-error', which basically is the average 'cluster-
error':
clustering_error = cluster_error_matrix.mean()
print(clustering_error)
```

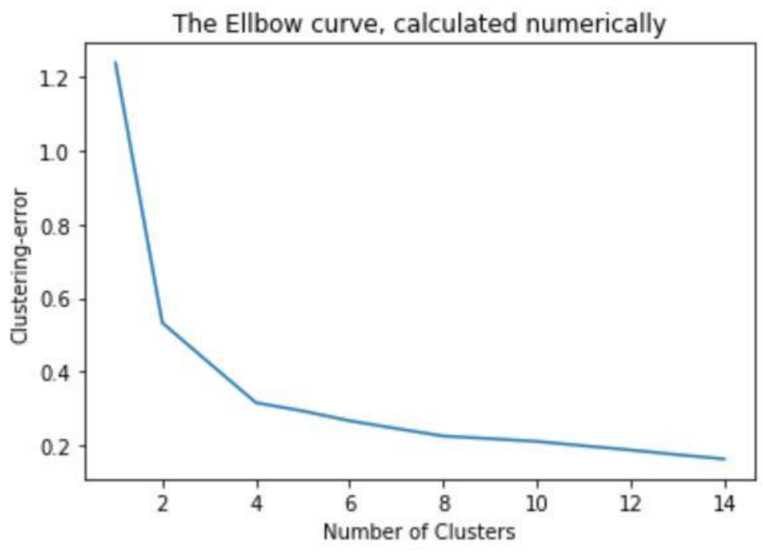
and the output (which is the total (average) 'clustering-error') is:

```
0.3161571935827624
```

D) (The name of the file: D.ipynb)

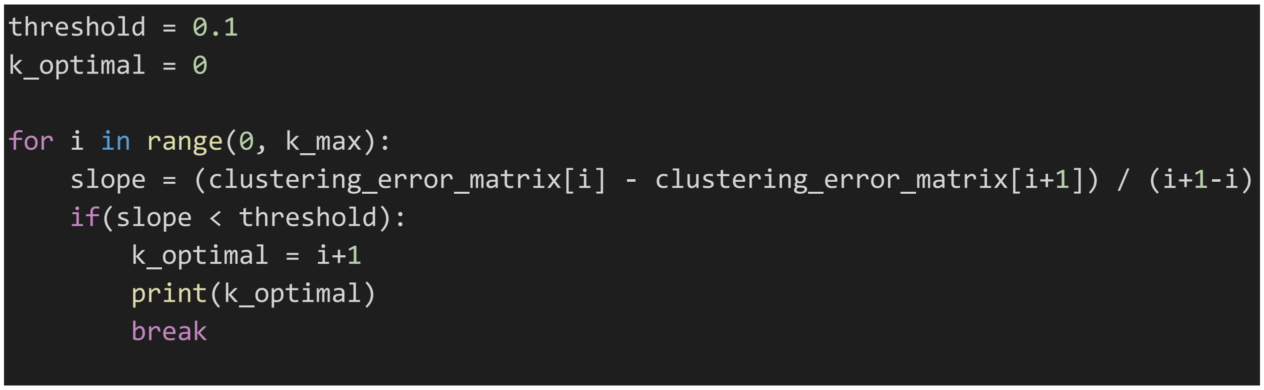
In this part, we should put all of the code in part (c) in a for-loop, with k ranging from 1 to k_max = 15. Then we should visualize all the calculated 'clustering-error's related to each k, in respect to k.

The output is a curve that we call 'The Elbow Curve', which the "elbow" part is where the curve's slope changes drastically and becomes more even. As we see bellow, in this case, it is k = 4!



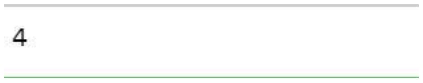
E) (The name of the file: E.ipynb)

In this part, we should use part (D)'s curve to see which k is optimal. So I wrote a code that calculates the slope of the curve for different k, and checks if a drastic change happens in the slope, and if so, then that corresponding k is optimal. The code is:



```
threshold = 0.1
k_optimal = 0

for i in range(0, k_max):
    slope = (clustering_error_matrix[i] - clustering_error_matrix[i+1]) / (i+1-i)
    if(slope < threshold):
        k_optimal = i+1
        print(k_optimal)
        break
```

and the output is:



```
4
```

just like what I comprehended based on the "Elbow curve" in part (D)!

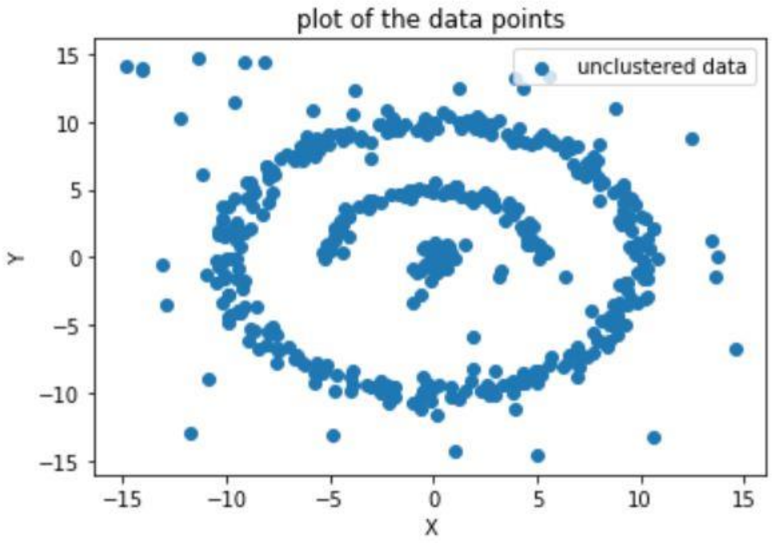
(we can also calculate the optimal value for k using Sklearn's KMeans and Insertia method, and I have commented it in my code as well. :))

F) (The name of the file: F.ipynb)

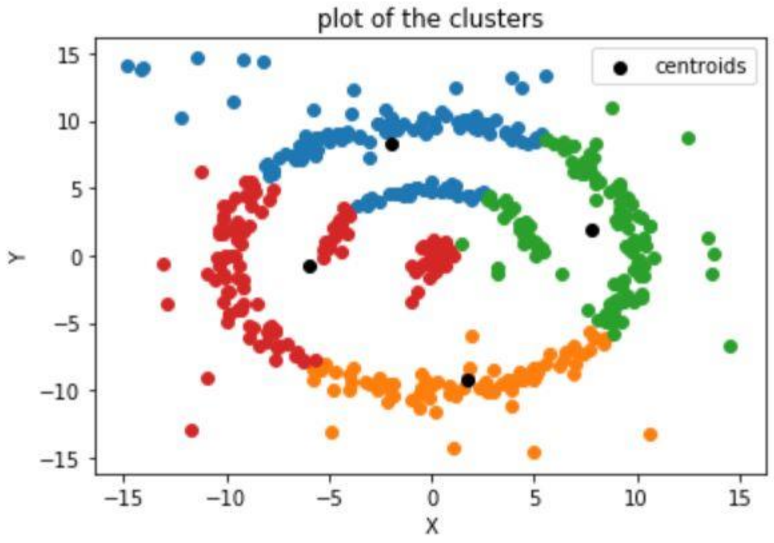
This part is like what we did in part (A), except that the dataset is 'Dataset2.csv'

So the output is:

The dataset:



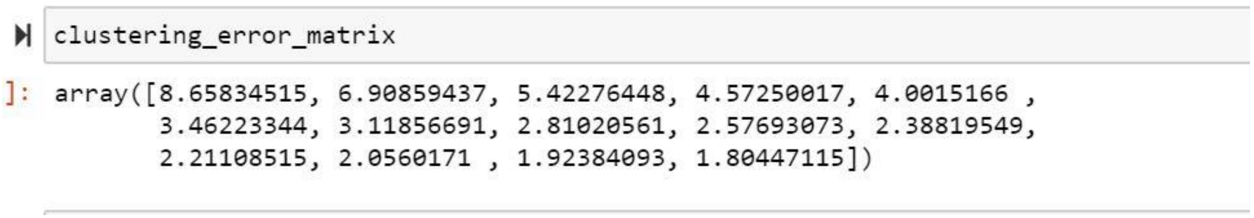
The clustered dataset:



So as we above, because we have arbitrary data shapes which are not linearly separable, then we can’t employ k-means algorithm to cluster ‘Dataset2.csv’. Instead, we should use Density-based clustering (such as DBSCAN algorithm) to cluster the data.

I have also checked the ‘clustering-error’ for k=1 to k=14 for ‘Dataset2.csv’, and as we see bellow, these are way higher than those of ‘Dataset1.csv’ which were around 0.3. This means the clustering method’s application for this dataset is not right; therefore, k-means clustering algorithm fails on ‘Dataset2.csv’.

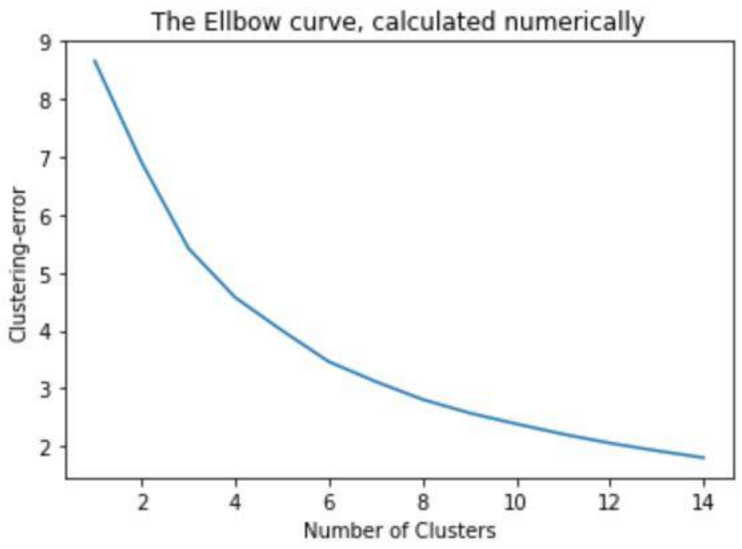
‘clustering-error’ for k=1 to k=14 for ‘Dataset2.csv’:



```
clustering_error_matrix
```

```
]: array([8.65834515, 6.90859437, 5.42276448, 4.57250017, 4.0015166 ,
          3.46223344, 3.11856691, 2.81020561, 2.57693073, 2.38819549,
          2.21108515, 2.0560171 , 1.92384093, 1.80447115])
```

And the ‘Elbow curve’ for ‘Dataset2.csv’ has less angles and is curvier, compared to that of the ‘Dataset1.csv’:



Implementation 2: DBSCAN Clustering Algorithm

A) (The name of the file: A.ipynb) (And the image files are: map.html, map_points.html)

We use 'Folium' library to load the map of a specific area in Jupyter Notebook, and then we can add our dataset's points (X, Y) on the map (for instance, in the form of circles).

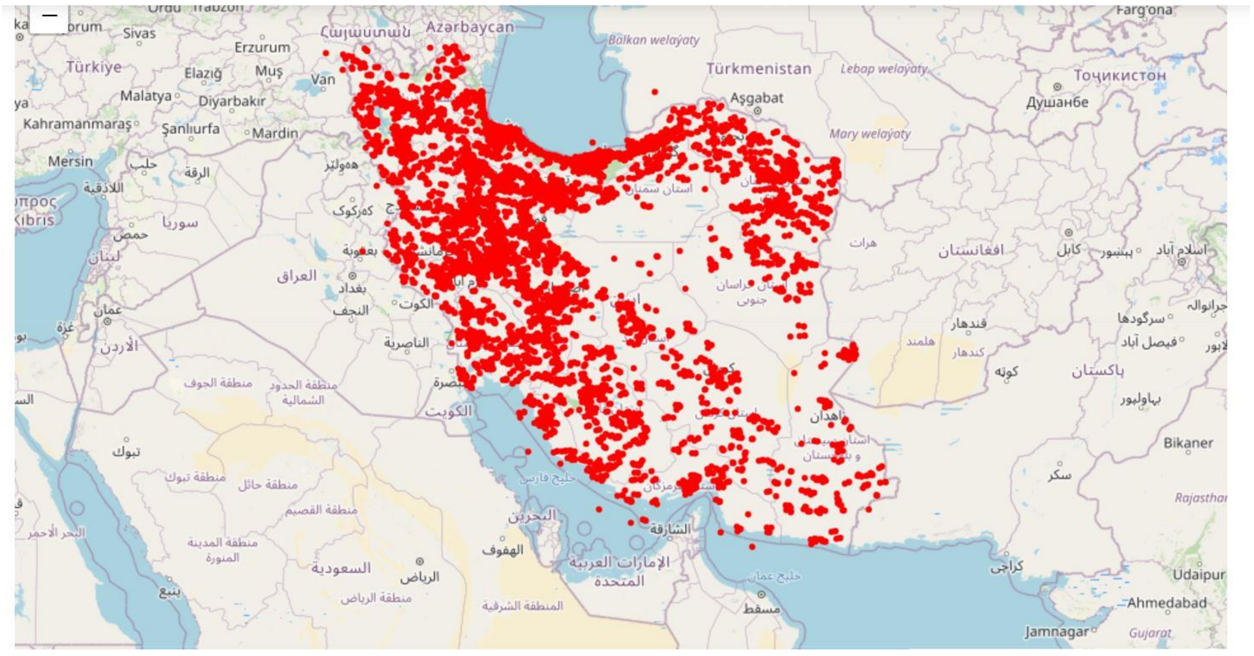
The map corresponding to location:

```
location = [32.427910, 53.688046]
```

which is the map of Iran:

Sivas
Erzurum
Azərbaycan
Türkiye
Elazığ
Muş
Van
Malatya
Diyarbakır
Kahramanmaraş
Şanlıurfa
Mardin
Mersin
Balkan welaýaty
Türkmenistan
Lebap welaýaty
Aşgabat
Mary welaýaty
Тоҷикистон
Душанбе
Farg'ona
ایران
Bikaner
Rajasthan
Udaipur
Ahmedabad
Jamnagar
Gujarat

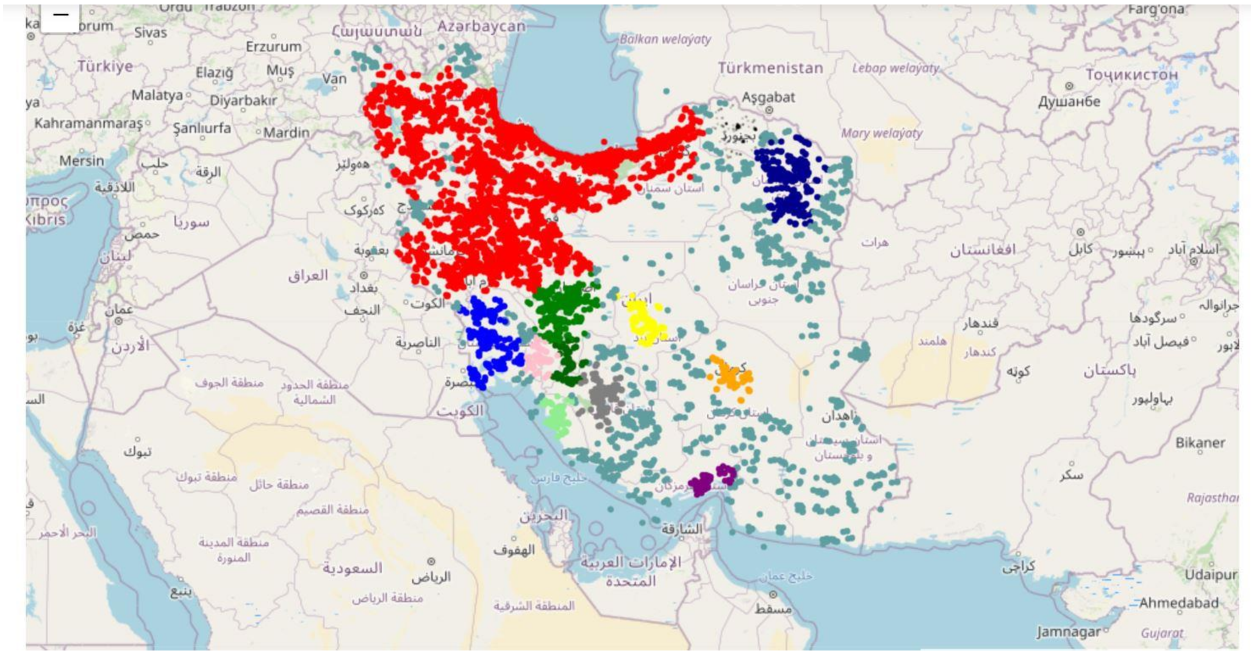
And if we add the dataset's points on the map, we have:



Which shows the density of the spread of COVID-19 in the country. Some places such as Tehran and the western part of Iran have higher number of patients than the eastern side.

B) (The name of the file: B.ipynb) (And the name of the image file: clustered_map.html)

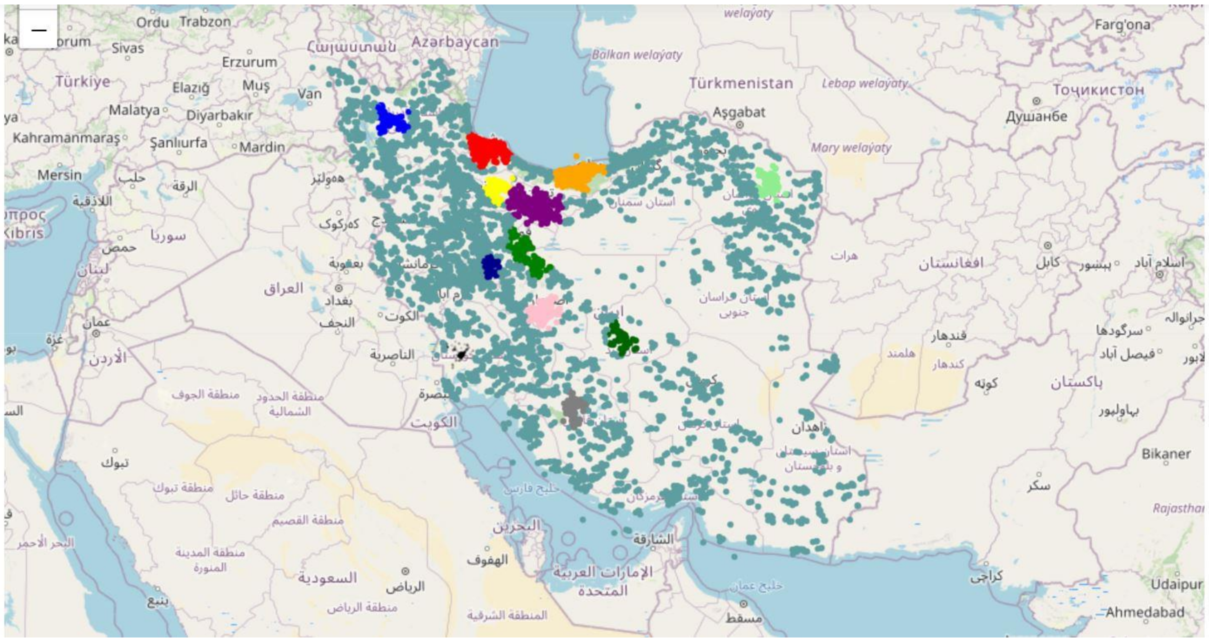
If we run DBSCAN algorithm for this dataset, with eps =0.5 and minPts=100, we have:



As we see, the data points are clustered and show by distinctive colors.

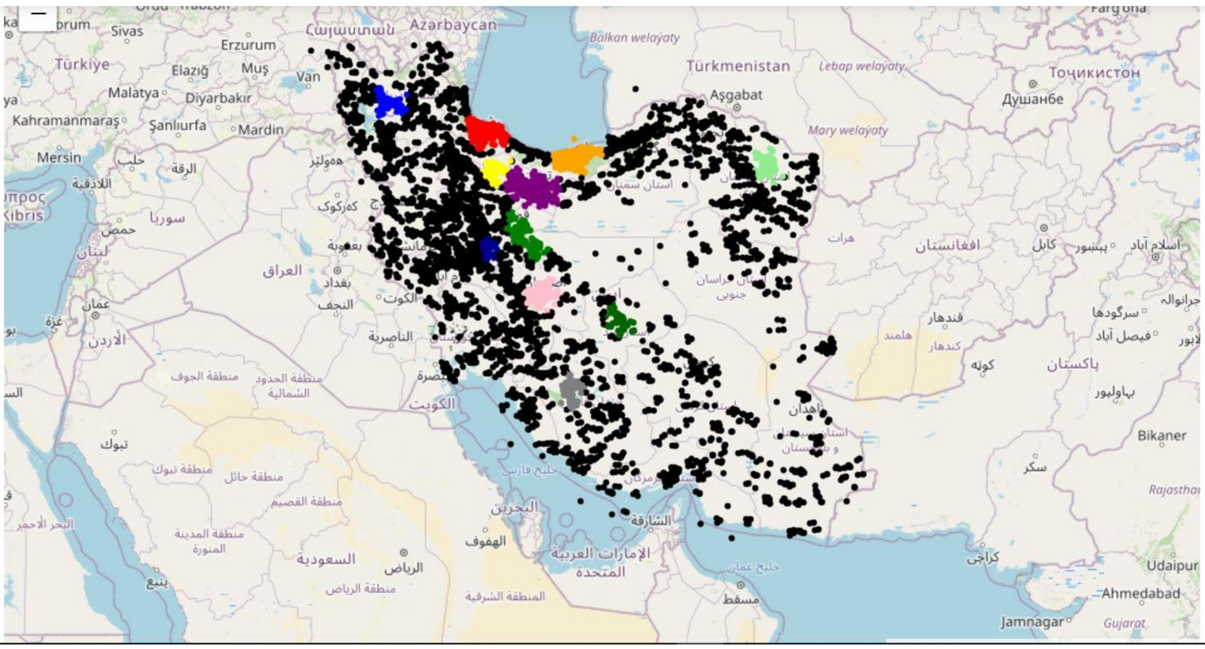
C) (The name of the file: C.ipynb) (And the image files are: best_clustered_map.html)

Continuing code of part (B), I tried different values for 'eps' and 'min_samples' (= minPts), and the best tuning was yield by 'eps = 0.32' and 'min_samples = 200':



D) (The name of the file: D.ipynb) (And the image files are: best_clustered_map_with_outliers.html)

I added 'black' to the color array, so that index '-1' which corresponds to outliers, is drawn in black on the map:



So as we see above, the outliers are related to places with fewer COVID-19 spread, and the colored clusters show denser hit areas, involving cities such as Tehran and Qom as highly dense areas with COVID-19 patients.

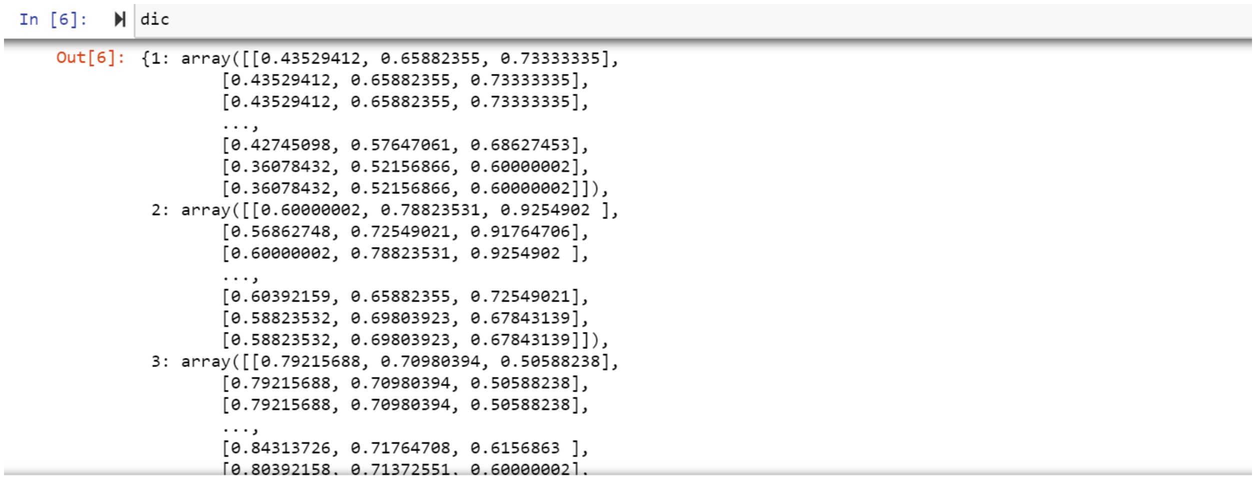
Implementation 3: Image Compression

***** I used “imageSmall.png” to run all the following parts.

A) (The name of the file: A.ipynb)

I used the k-means clustering algorithm I wrote in ‘Implementation 1/ part A’ for the pixels in the image, so that each pixel acts as a data point. Then, just like what I did in ‘Implementation 1/ part A’, I appended the clusters to a dictionary, which its values are numpy arrays of size 3, corresponding to the three colors RGB.

So the output cluster is:



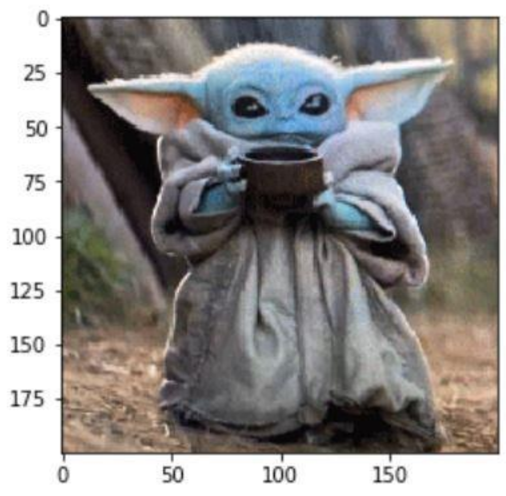
```
In [6]:    dic

   Out[6]: {1: array([[0.43529412, 0.65882355, 0.73333335],
                  [0.43529412, 0.65882355, 0.73333335],
                  [0.43529412, 0.65882355, 0.73333335],
                  ...,
                  [0.42745098, 0.57647061, 0.68627453],
                  [0.36078432, 0.52156866, 0.60000002],
                  [0.36078432, 0.52156866, 0.60000002]]),
            2: array([[0.60000002, 0.78823531, 0.9254902 ],
                  [0.56862748, 0.72549021, 0.91764706],
                  [0.60000002, 0.78823531, 0.9254902 ],
                  ...,
                  [0.60392159, 0.65882355, 0.72549021],
                  [0.58823532, 0.69803923, 0.67843139],
                  [0.58823532, 0.69803923, 0.67843139]]),
            3: array([[0.79215688, 0.70980394, 0.50588238],
                  [0.79215688, 0.70980394, 0.50588238],
                  [0.79215688, 0.70980394, 0.50588238],
                  ...,
                  [0.84313726, 0.71764708, 0.6156863 ],
                  [0.80392158, 0.71372551, 0.60000002],
```

B) (The name of the file: B.ipynb) (And the name of the final image file: img_compress.png')

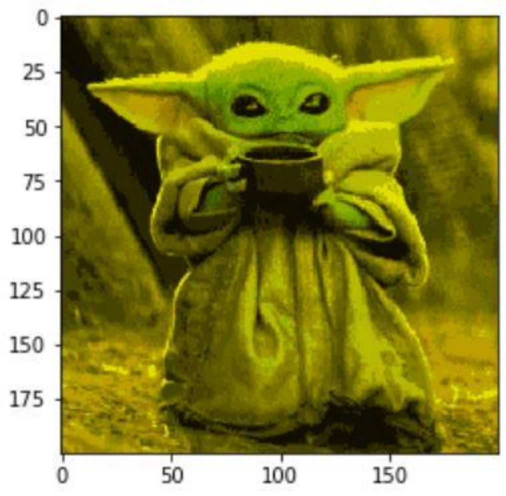
I used these lines to map each cluster's centroid to its corresponding pixels:

```
img_compress = centroids[index_matrix.astype(int), :]
img_compress = np.reshape(img_compress, (img.shape[0], img.shape[1], img.shape[2]
))
```

The original image is:



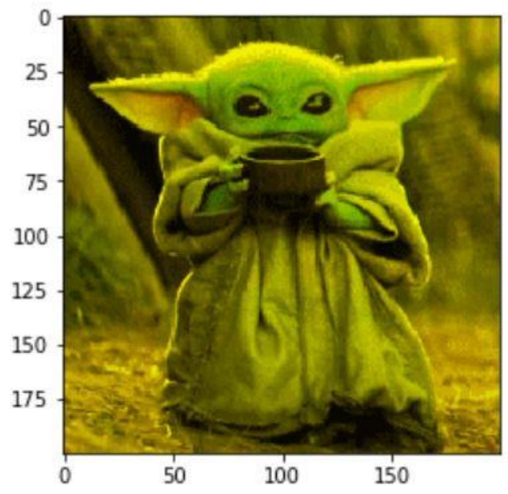
And the compressed image is:



C) (The name of the file: C.ipynb) (And the name of the final image file is img_k256_compress.png)

Here, we should set k=256 and run the algorithm again. Since k or the number of clusters increases, the centroids increases; thus, we have a wider range of colors to choose from for our pixels. Therefore, the final image should be more like the original image compared to that of part (B), since here we have more diverse colors.

The output image is:



So as we see in this part where k=256, the image is of much higher quality, compared to image in part (B), where k=16. Which is logical, since if the number of clusters increases, then the number of centroids and therefore the quality and color contrast increases. So we will see a better image than part (B).

D)

I have saved the images from each part in the ‘Implementation 3’ folder, and as it is written:

Size of “image.png”: 340 KB

Size of “imageSmall.png”: 32 KB

Size of “img_compress.png” from part (B): 2 KB

Size of “img_k256_compress.png” from part(C): 2 KB

As we expected, the size of “img_compress.png” from part (B) and (C) is indeed less than the size of the original image, since we have reduced its pixels’ colors by substituting each pixel color with its cluster centroid’s color.