Final – Option 1 – Geoclustering

DSCI6007

Mohamed Moustafa

1. Motivation

Due to the data revolution in recent years, data is being generated at an unprecedented pace. With an increase in the volume and veracity of data, there is a need to effectively capture, store, process, and make use of this data. Complications, however, arise due to the variety and veracity of the data, making "data wrangling" one of the most important steps in a big data application.

This project is an application of the k-means algorithm for the clustering of large geospatial data. Clustering is the process of grouping together "similar" datapoints. This idea of clustering can be useful in areas such as marketing, analytics, medicine, natural language processing, and so on.

The k-means algorithm is an iterative approach to cluster datapoints together, normally performed in Euclidean distance, but you can modify the function if necessary. The "k" in k-means clustering refers to the number of clusters that's desired. This parameter should be carefully selected, and ideally selected to suit the desired model.

When working on a big data application, the volume of the dataset can present significant computational challenges. This is where we can benefit from parallel processing to improve efficiency and speed.

The motivation of this project is to apply the k means clustering algorithm to geospatial data using pyspark, with our data being stored on an S3 bucket on AWS.

2. System Configuration

An EMR cluster was created on AWS. I ssh'd into the master node of the respective ec2 instance. Since I was not focused on security, I allowed access to all traffic under security group settings. I then configured and opened jupyter notebook using the master node ec2 instance IP and token id.

3. Documentation of Approach

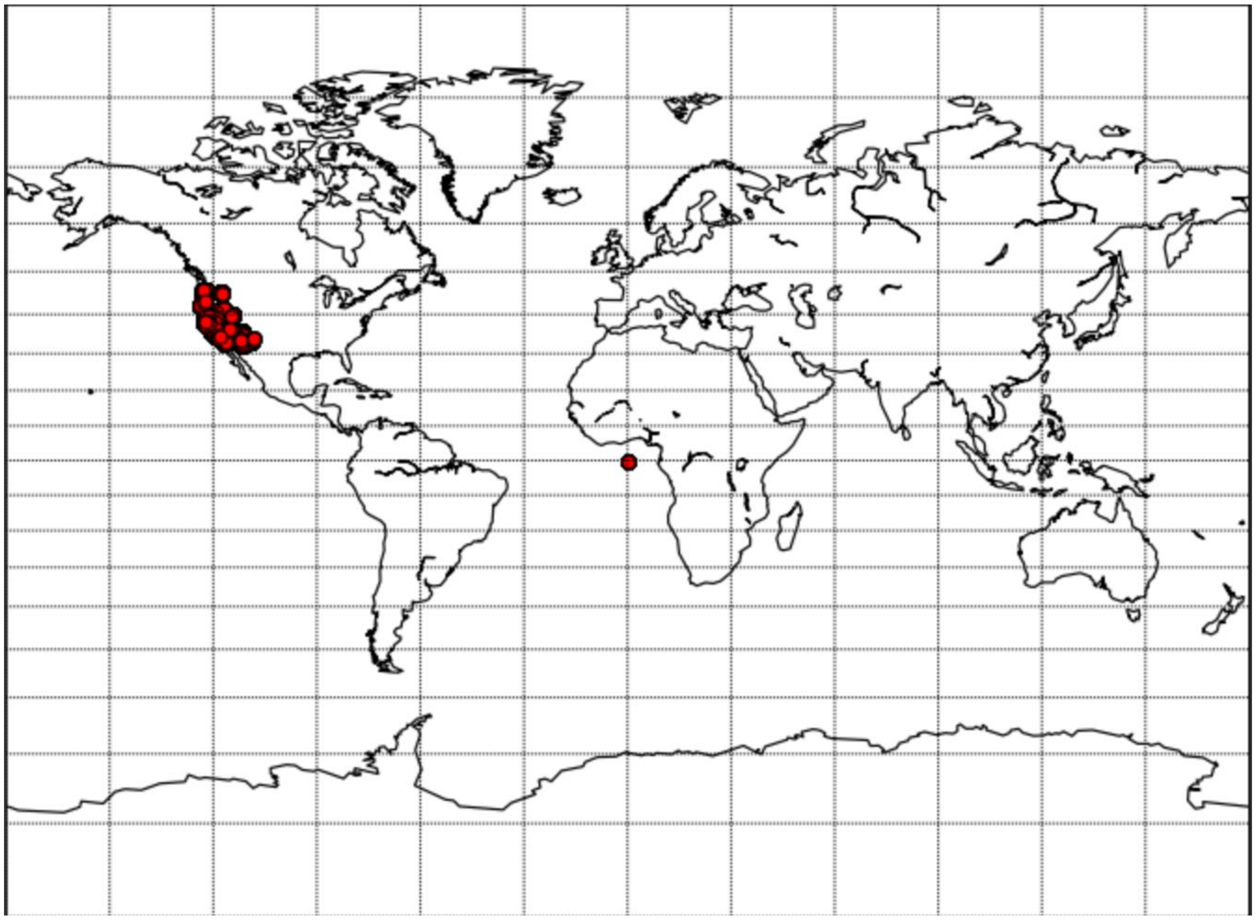
Pre-processing was the most important part of this project, so I had to focus on ensuring the data was usable. I downloaded each of the supplied zip files, extracted them, and then uploaded the files on my S3 bucket. I created 3 separate jupyter notebooks (as instructed) to process each of the files to be used, and their "cleaned" versions also stored on the same S3 bucket. I created a main jupyter notebook for the k means algorithm, plotted the results and documented the analytics.

4. Big Data Application/Dataset

4.a. Description

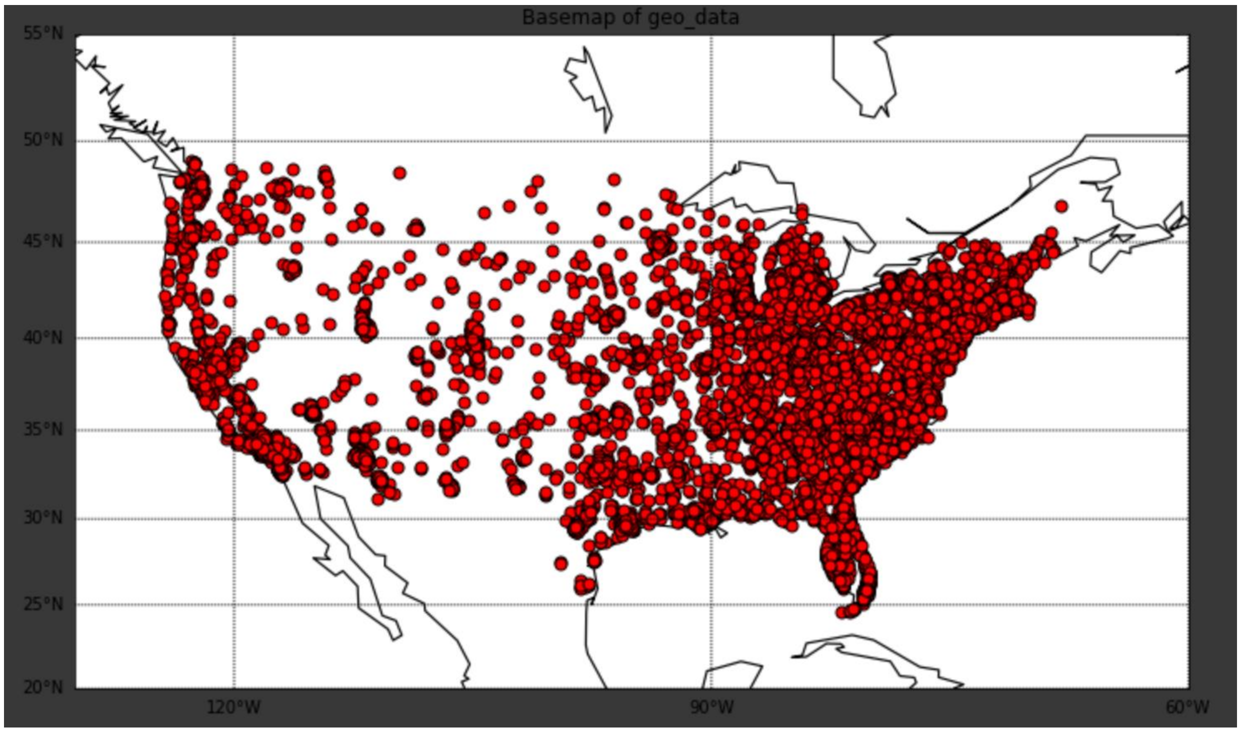
There were 3 datasets in this project:

- Devicestatus.txt contained about 100000 samples of positional data relating to device ID, date, model, gps location, etc.. We don't care about most of the features in this dataset, so they were ignored. Features were separated by the following delimiters: |/,

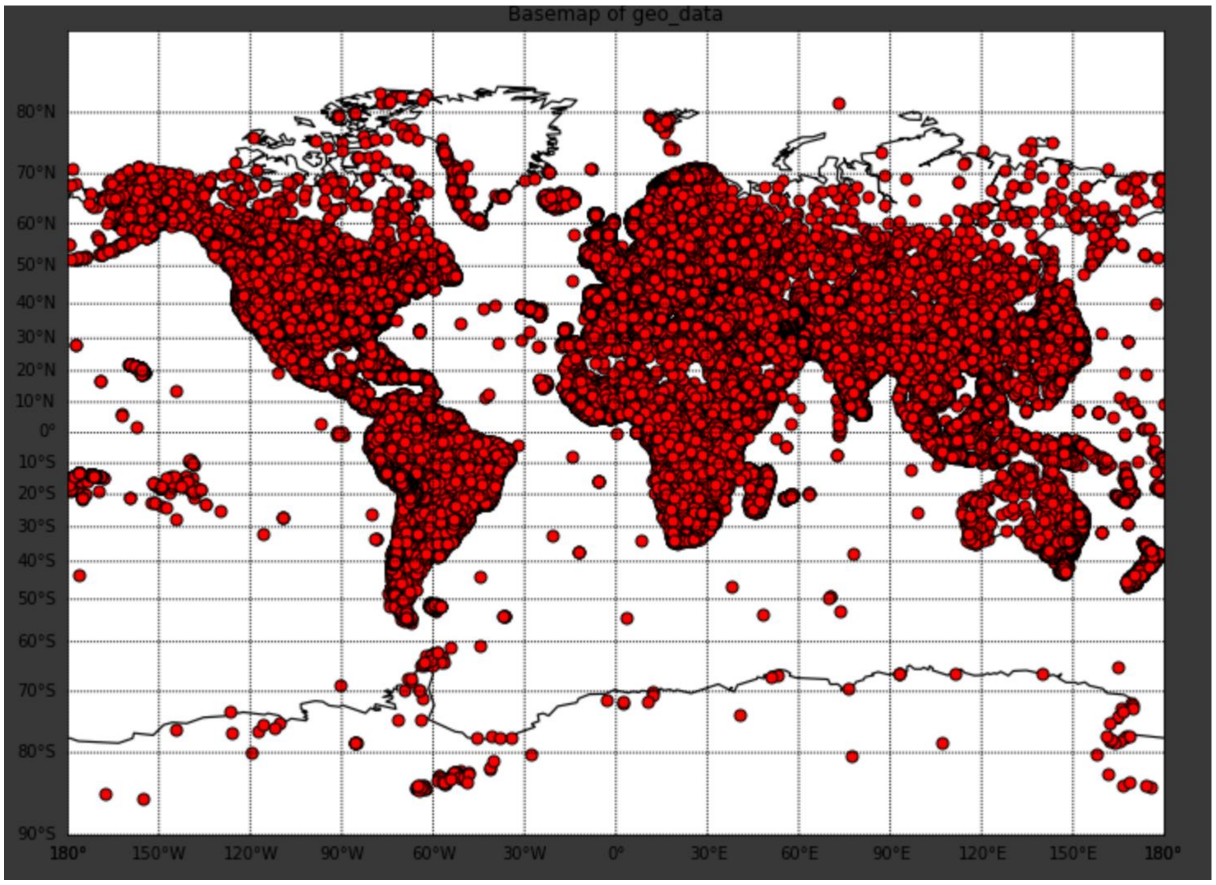


Note: Notice the small dot in the ocean near Africa. This is due to the fact that I did not remove the 0,0 coordinates when I was visualizing and exploring my data on google collab. Note, however, that it is clear that the data points are in California. Another visual without the erroneous 0,0 points will be displayed later.

- Sample_geo.txt had synthetic geo-location clustering data. This dataset had roughly 10000 samples, and only contained latitude, longitude, and id columns. This dataset was separated by tabs.



- Lat_longs.txt contained large-scale clustering data from DBPedia. This file contained 3 features: latitude, longitude, and website.



4.b Implementation:

I imported pyspark, numpy, sys, matplotlib, math, and pandas. I created my functions to be implemented for K-means clustering.

Closest_point takes a given point, data_point, and method and returns the index in the array of the center closest to the given datapoint.

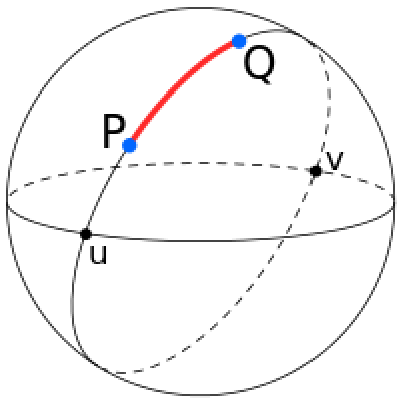
Add_points takes two points, and returns the component-wise sum. Note, we would return one point. We add the 2 x-components together, and the 2 y components together. Return sum_x, sum_y.

Euclidean_distance takes in two points, and returns the Euclidean distance. This is done by summing the components of the two points squared, and then taking the root of that. Much like the simple distance formula from trig. Returns the distance.

Great_circle distance takes in two points, and returns the orthodromic distance (otherwise known as the great circle distance). The formula for the great_circle_distance is given on Wikipedia, and is as follows:

$$\Delta\sigma = \arctan \frac{\sqrt{\left(\cos\phi_2 \sin(\Delta\lambda)\right)^2 + \left(\cos\phi_1 \sin\phi_2 - \sin\phi_1 \cos\phi_2 \cos(\Delta\lambda)\right)^2}}{\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos(\Delta\lambda)}.$$

Let λ_1, ϕ_1 and λ_2, ϕ_2 be the geographical longitude and latitude in radians of two points 1 and 2, and $\Delta\lambda, \Delta\phi$ be their absolute differences



Kmeans_cluster takes in data, desired converge_dist, method (Euclidean or orthodromic), and k. We take a sample, and find the closest points from each point. The following tutorial from Stanford helped explain it.

https://stanford.edu/~cpiech/cs221/handouts/kmeans.html

As discussed earlier, there were three separate jupyter notebooks that served to clean each .txt file independently. Data exploration was performed on separate notebooks (I used google colab because it was easier for me personally), and served to display the visuals using pandas as well as matplotlib.

I uploaded the .txt files on my S3 bucket. I ssh'd into the master node, and ran each independent jupyter notebook – verifying that the cleaned data was uploaded to my S3 bucket.

I created my main jupyter notebook, which contains my functions for implementing kmeans clustering. I loaded my data, and ran k means. I created a plotting function for convenience. I was able to plot the resulting k means for algorithms on mobileNET data with k=5; synthetic data with k = 2,4,6; and DBPedia with k = 6 and 10 – all both using Euclidean distance and isodromic distance. I felt that k of 6 was sufficient for the synthetic data, and k = 10 was sufficient for the DBPedia data. It was also visible on the plot when we changed between Euclidean and isodromic. Due to the way that the calculations were performed, the clustering was different.

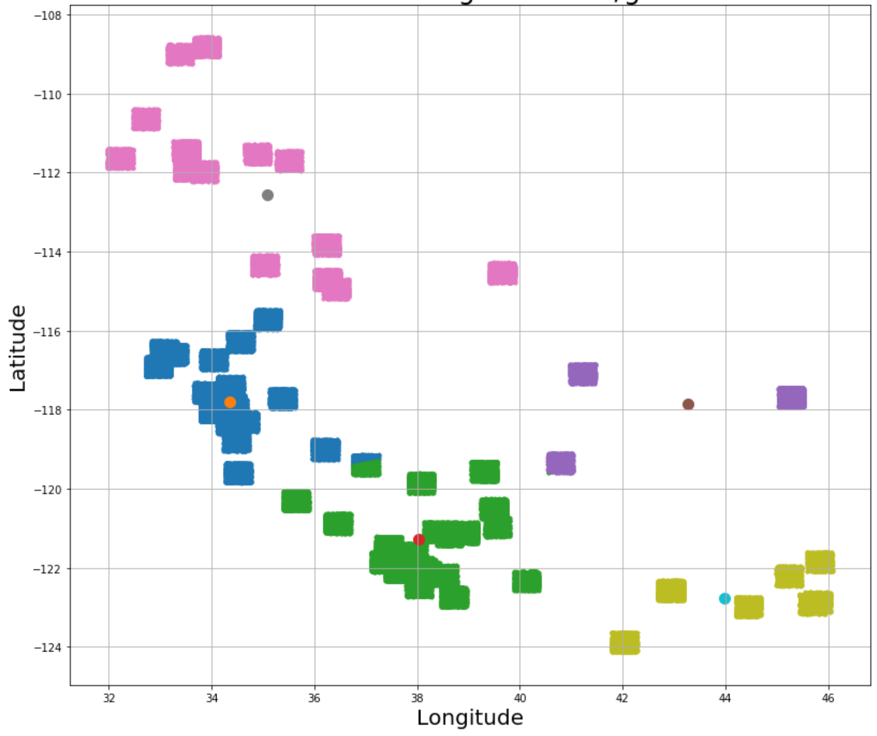
In general, the clusters represented a grouping of a "nearby" datapoints. It's apparent that landmasses, borders, and climate, etc impacted how the clustering was done, due to the clusters encompassing familiar regions, such as south America, or new England, etc..

Lastly, I performed runtime analytics to compare performance for each run. I then used multithreading on all three datasets simultaneously to highlight the benefits of parallel processing.

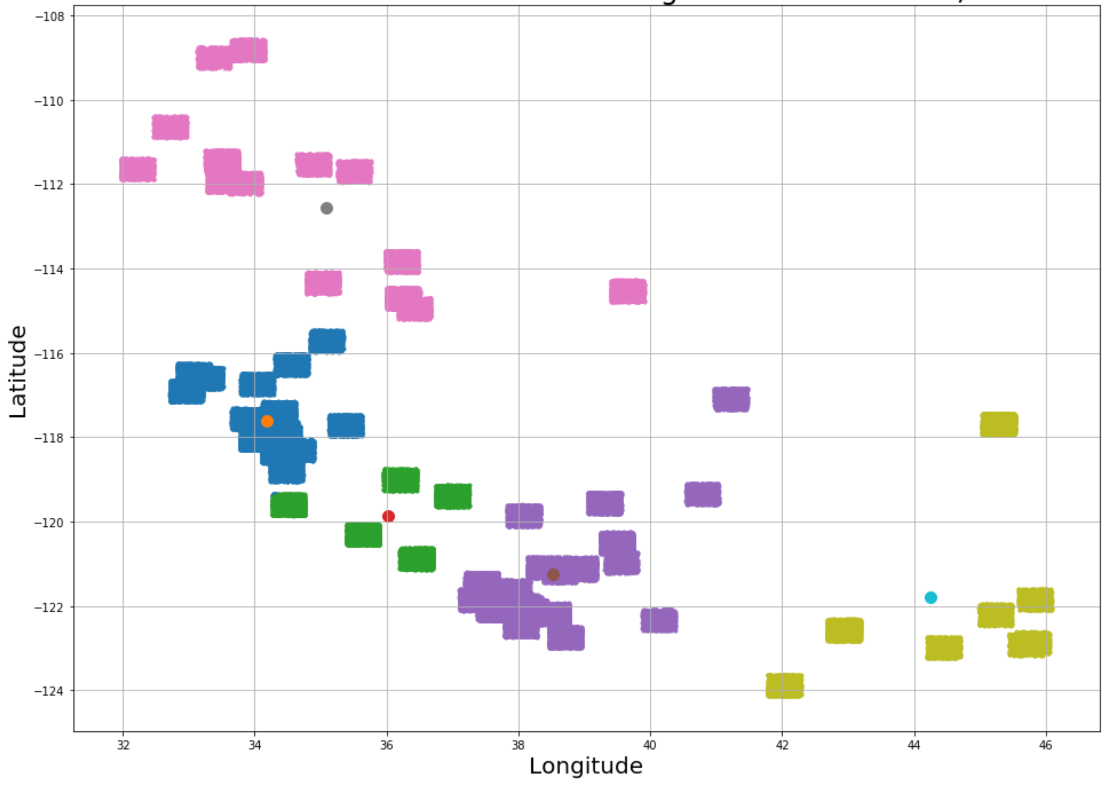
4.c Results and Discussion

The results of the kmeans cluster algorithm applied to the MobileNET data with k = 5 is shown below, with both Euclidean and isodromic distance. In this scenario, I felt like Euclidean was more appropriate since it appears that an ineffective cluster was formed on eastern edges of California. Euclidean may be more appropriate since it was a focused region that we were studying. Below are the plots:

devicestatus data from Loudacre Mobile using isodromic/great circle distance; k means = 5

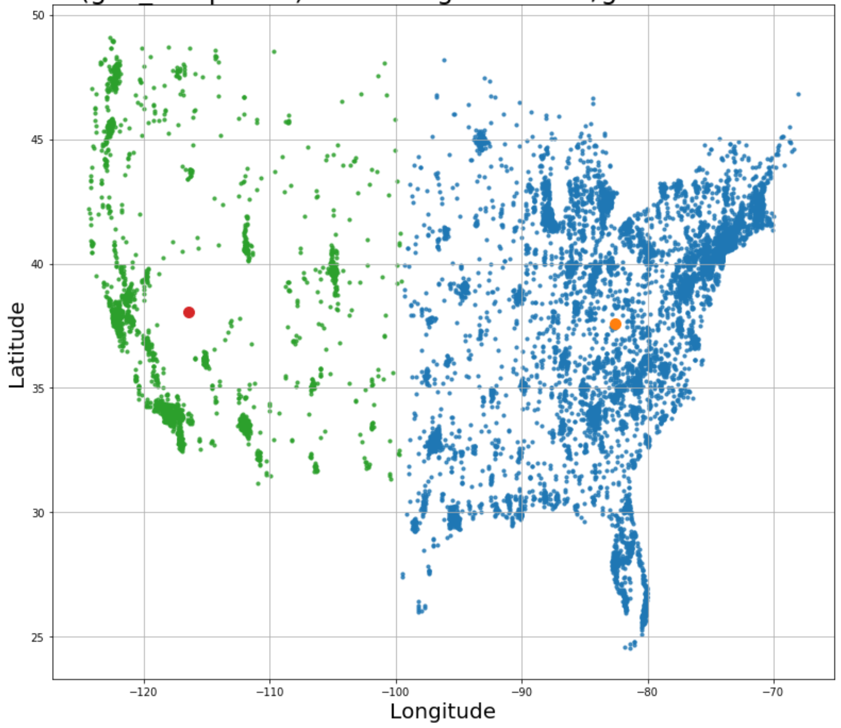


devicestatus data from Loudacre Mobile using euclidean distance; k means = 5

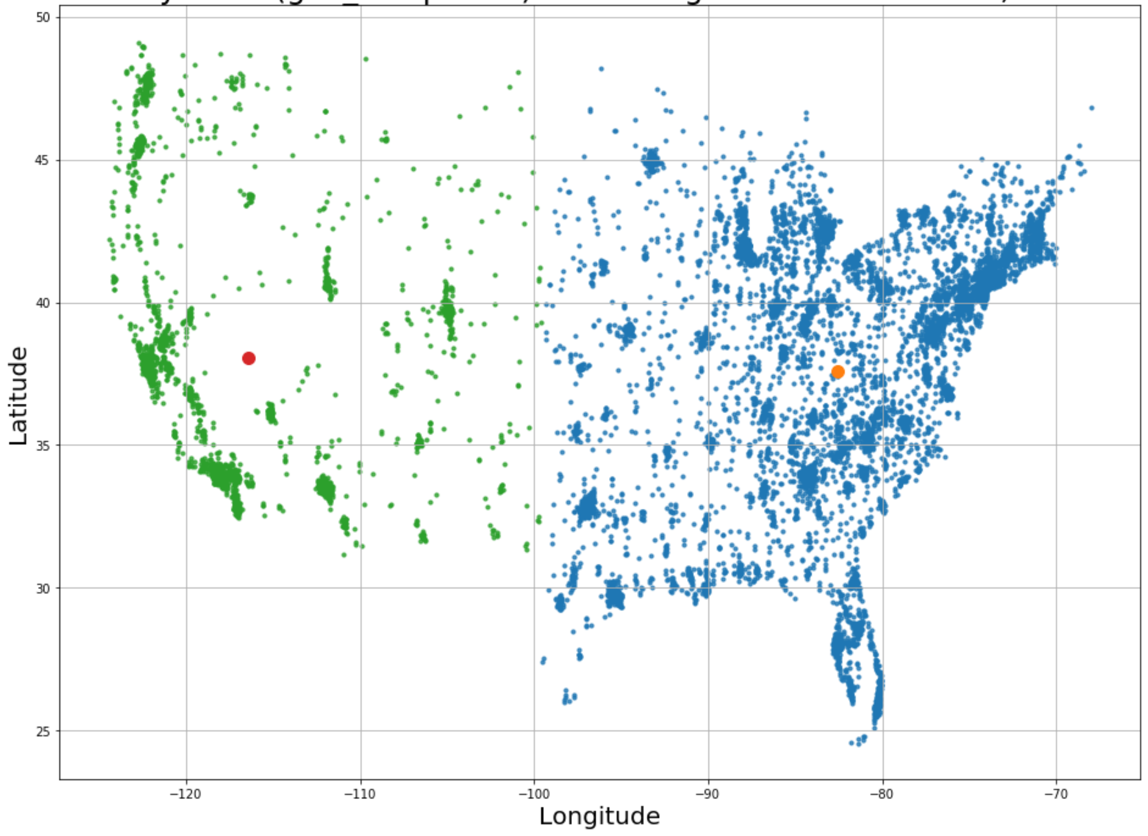


I then did the same, but applied kmeans to the synthetic data, with k = 2, 4, and 6:

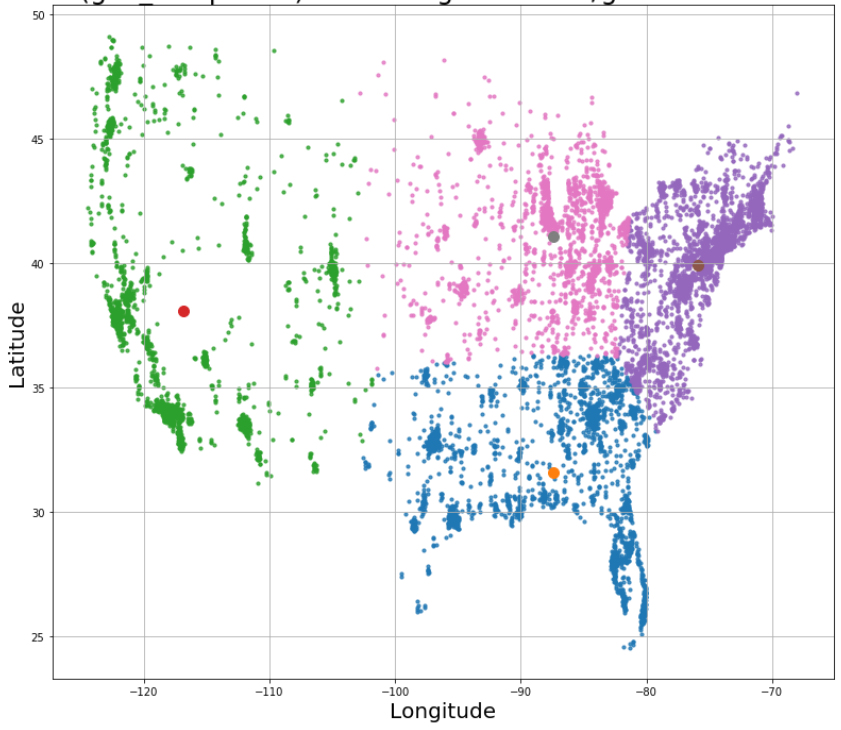
data from syntetic (geo_sample.txt) data using isodromic/great circle distance; k means = 2



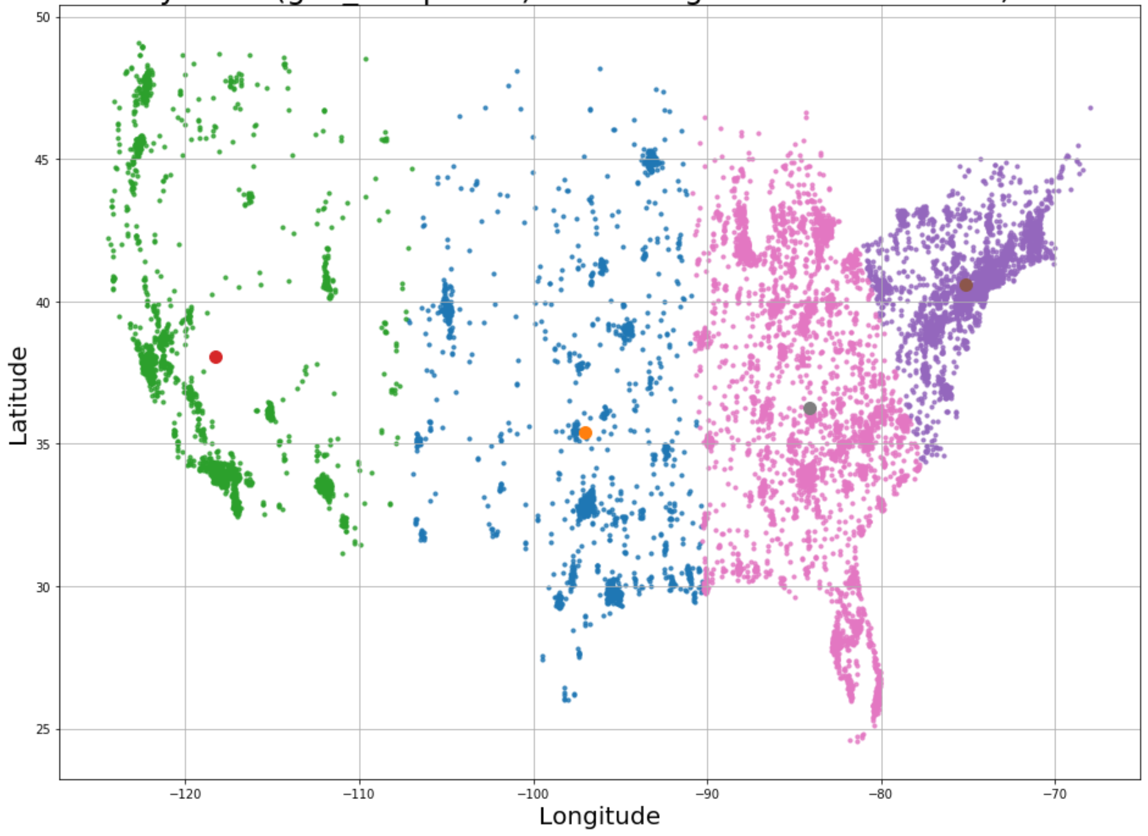
data from syntetic (geo_sample.txt) data using euclidean distance; k means = 2



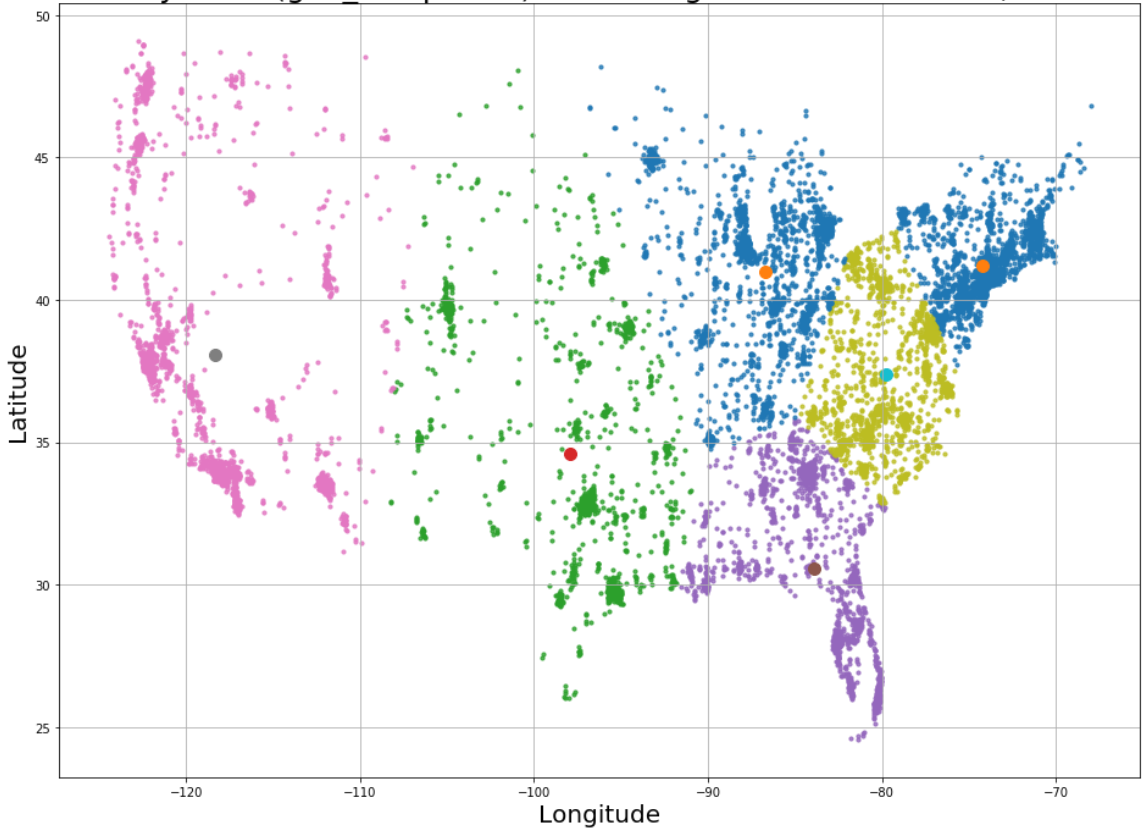
data from syntetic (geo_sample.txt) data using isodromic/great circle distance; k means = 4



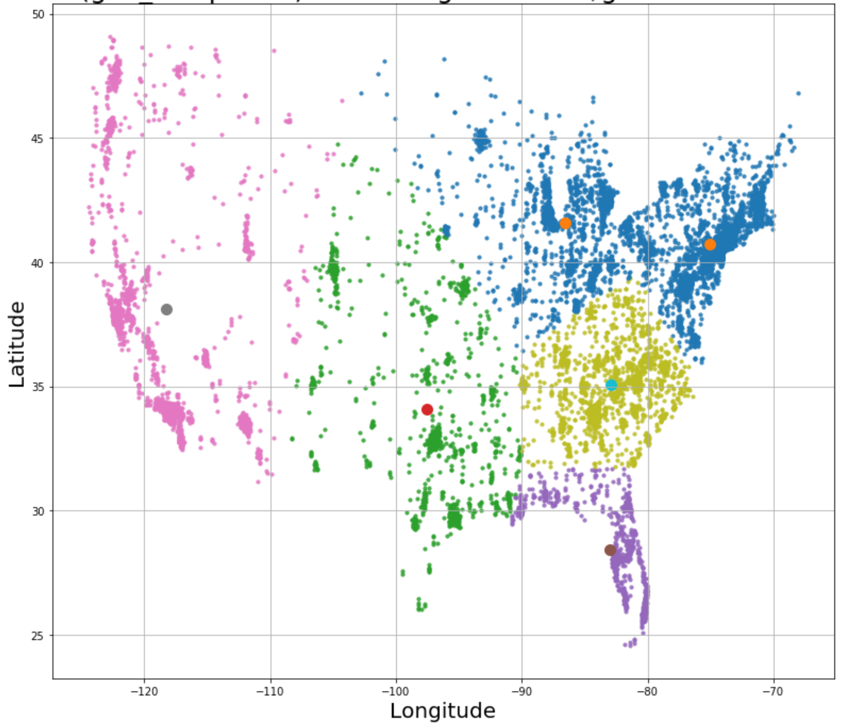
data from syntetic (geo_sample.txt) data using euclidean distance; k means = 4



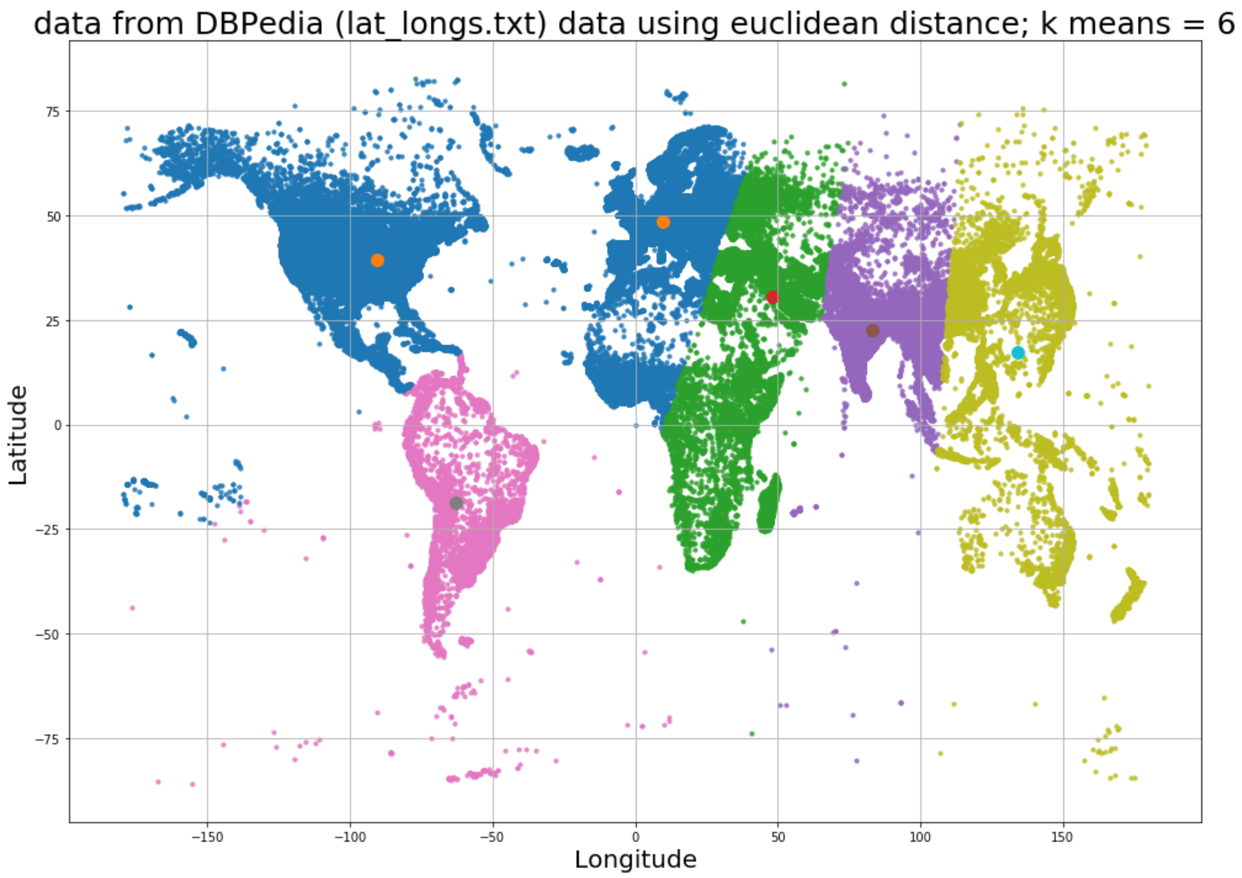
data from syntetic (geo_sample.txt) data using euclidean distance; k means = 6



data from syntetic (geo_sample.txt) data using isodromic/great circle distance; k means = 6

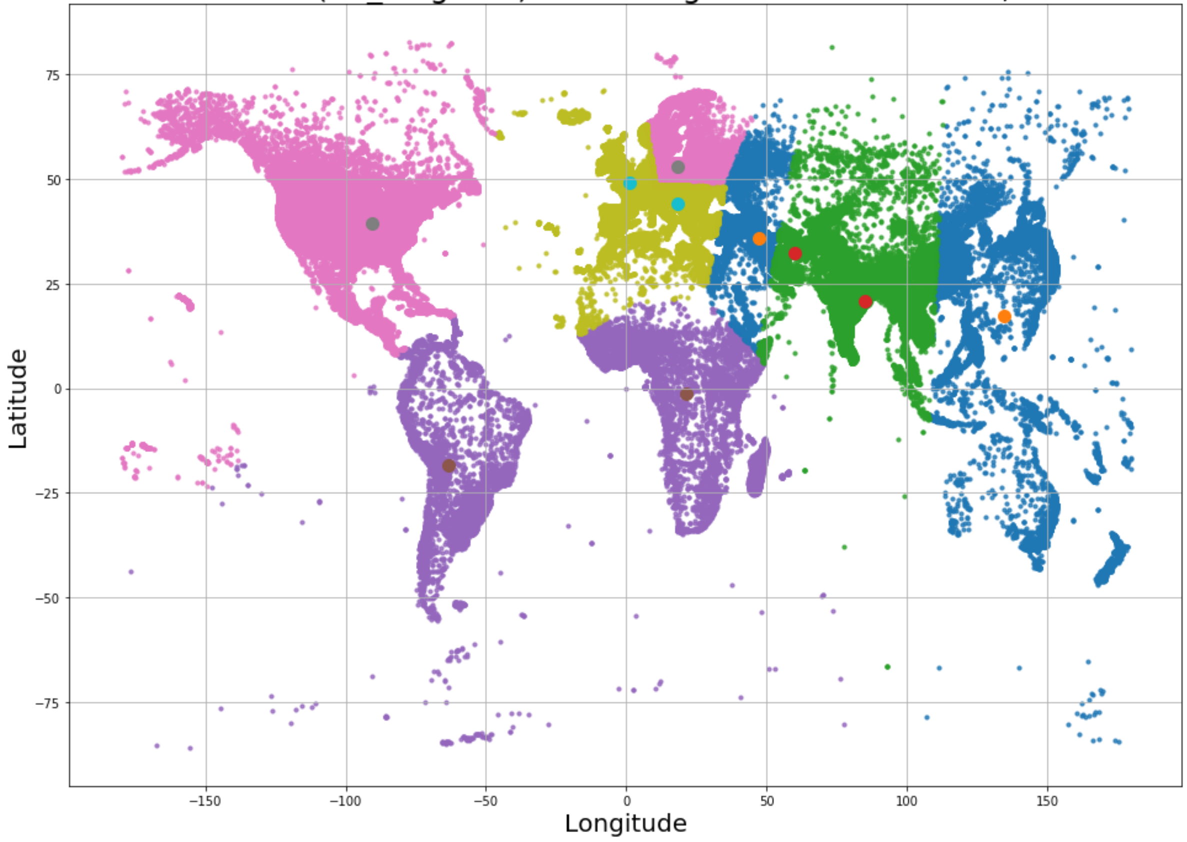


I also did the same thing with the DBPedia data, with k = 6 and 10.

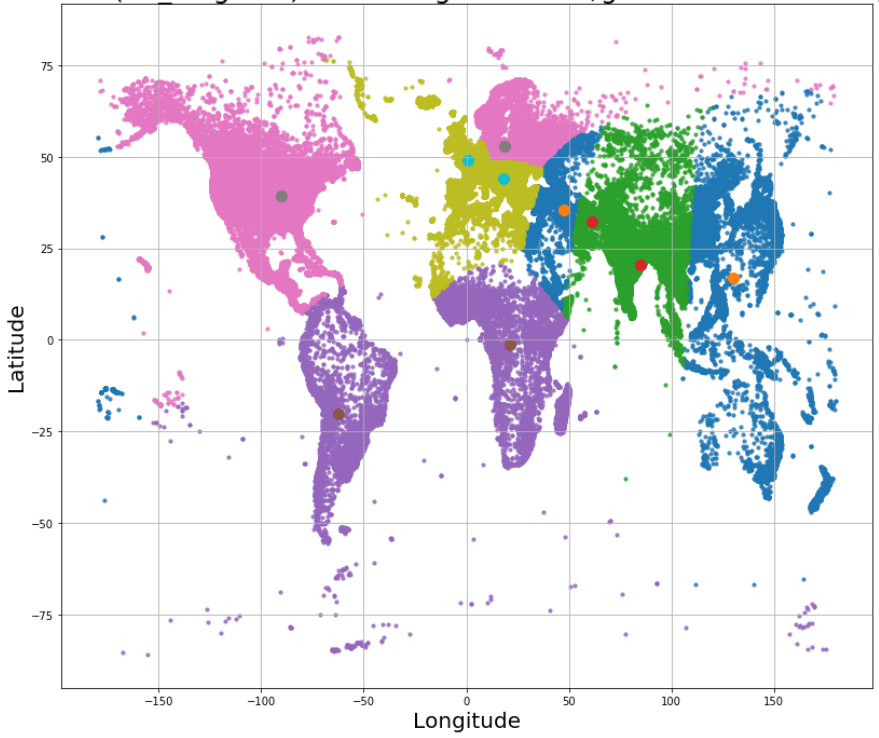


data from DBPedia (lat_longs.txt) data using euclidean distance; k means = 6
Latitude
Longitude
75
50
25
0
−25
−50
−75
−150
−100
−50
0
50
100
150

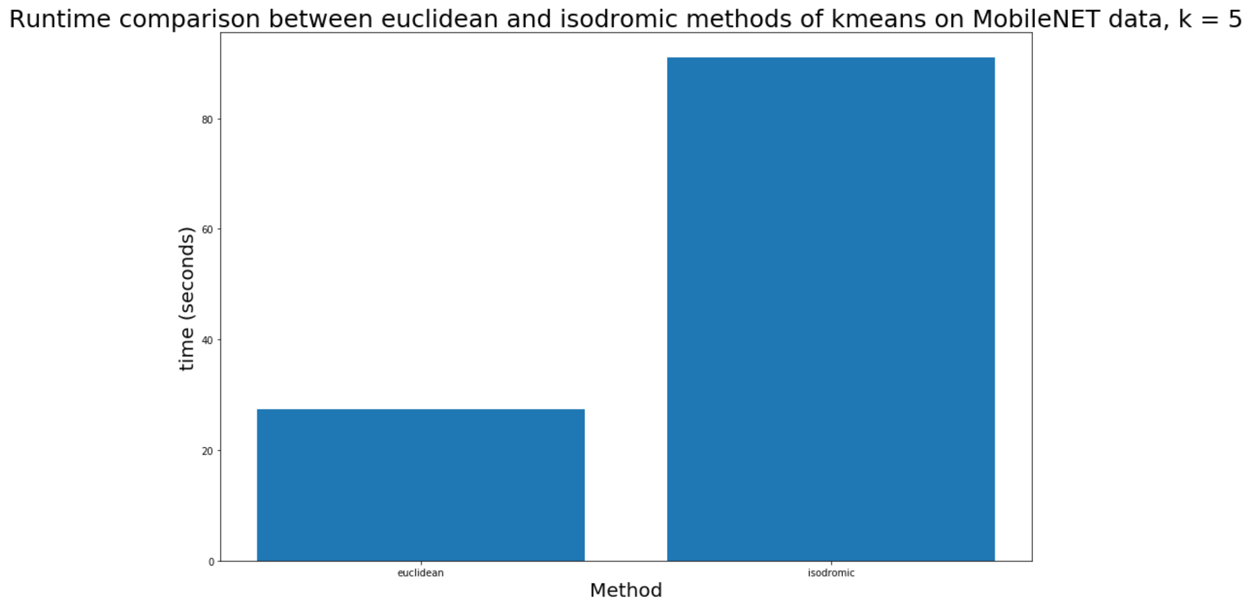
data from DBPedia (lat_longs.txt) data using euclidean distance; k means = 10

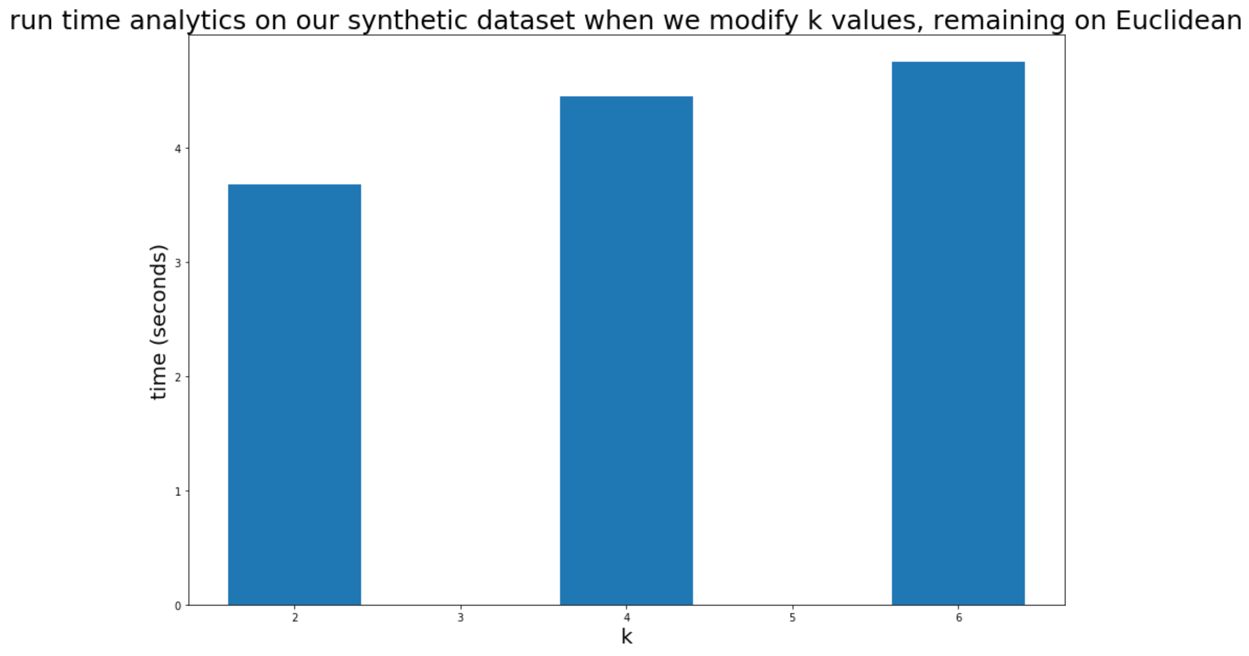


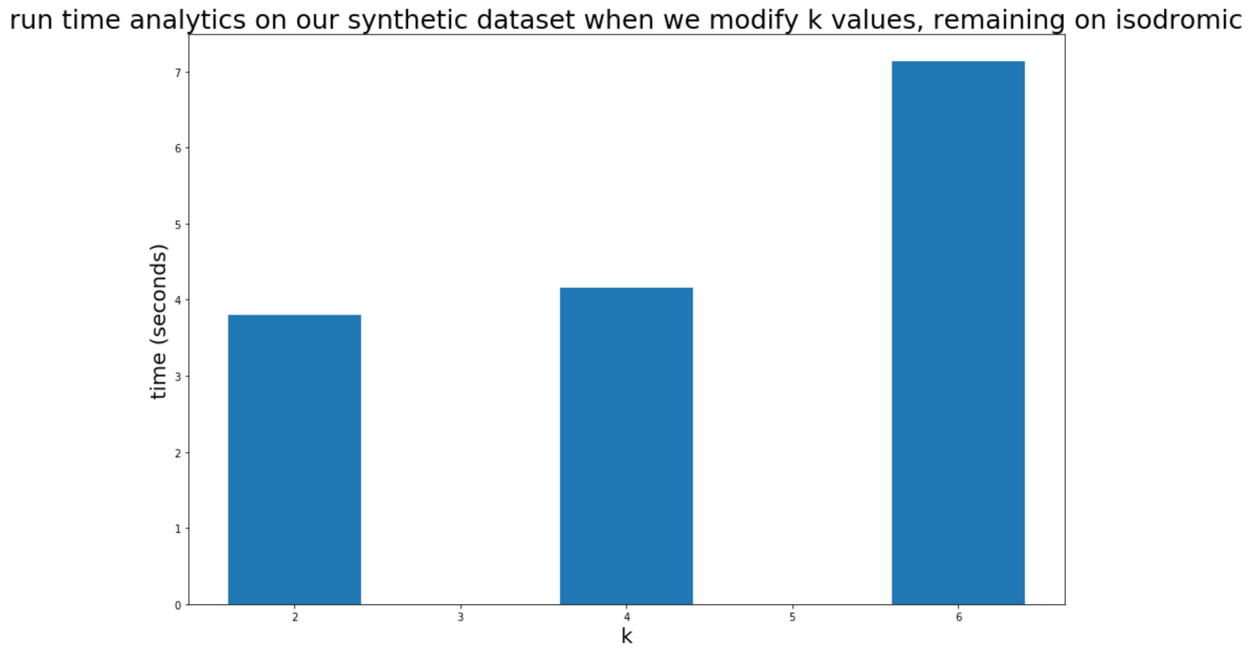
data from DBPedia (lat_longs.txt) data using isodromic/great circle distance; k means = 10

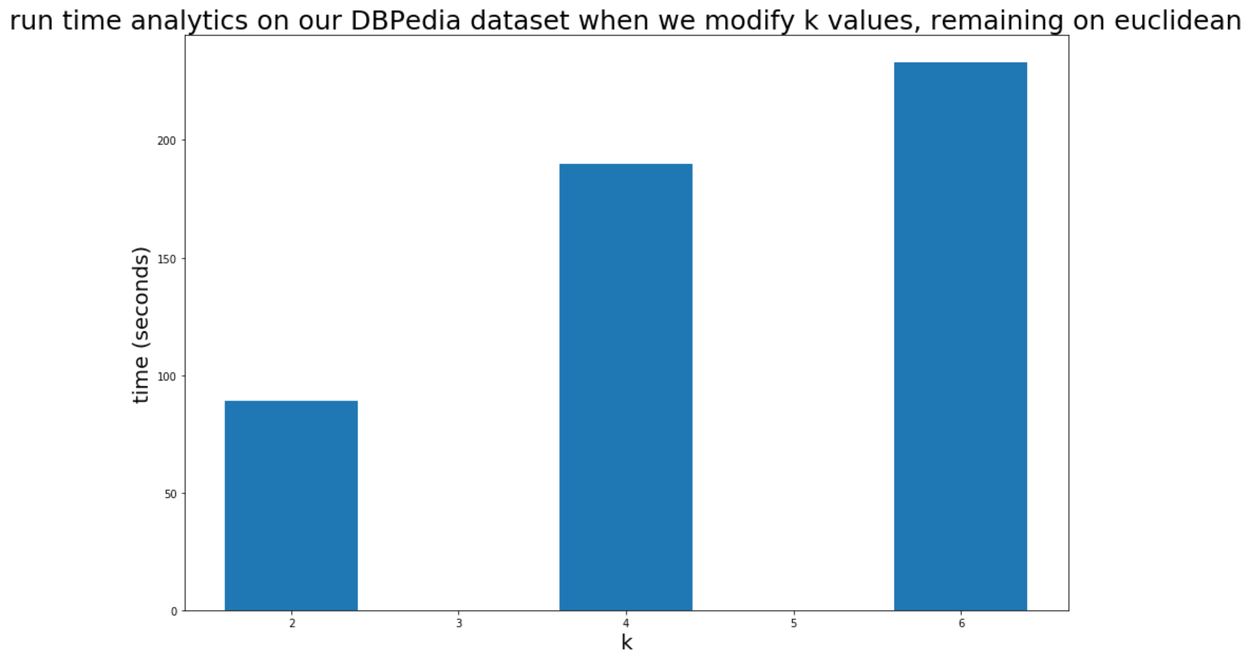


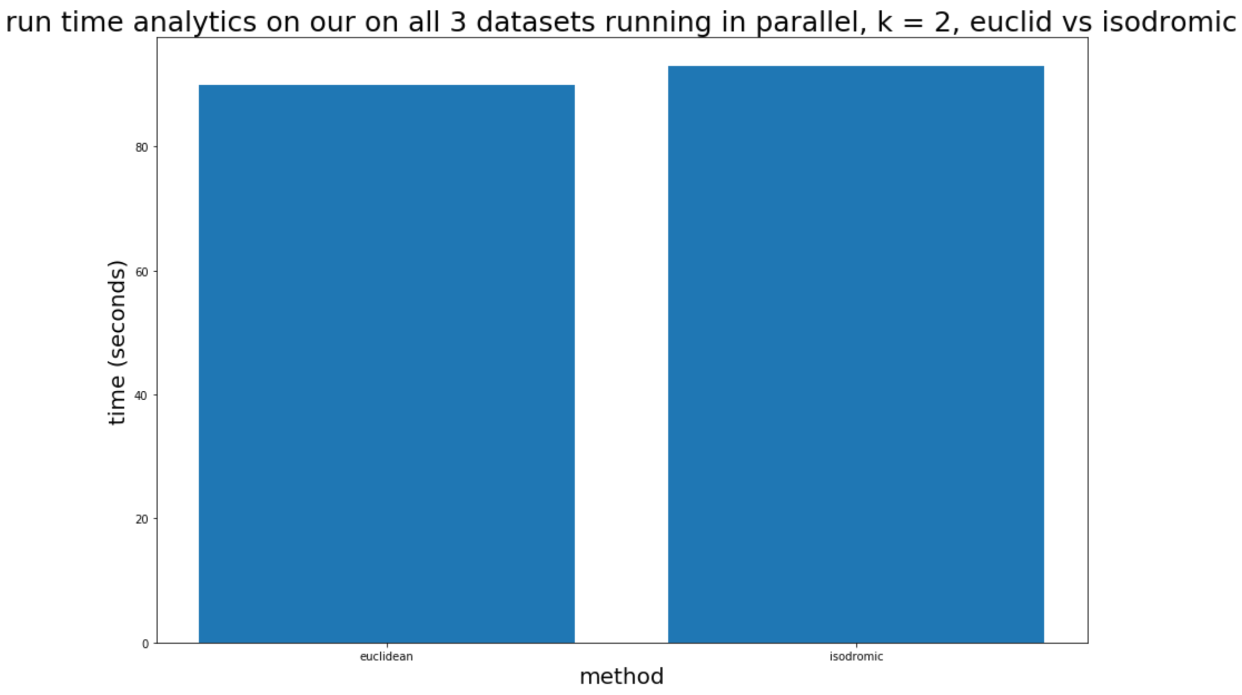
Some runtime analytics are shown below:

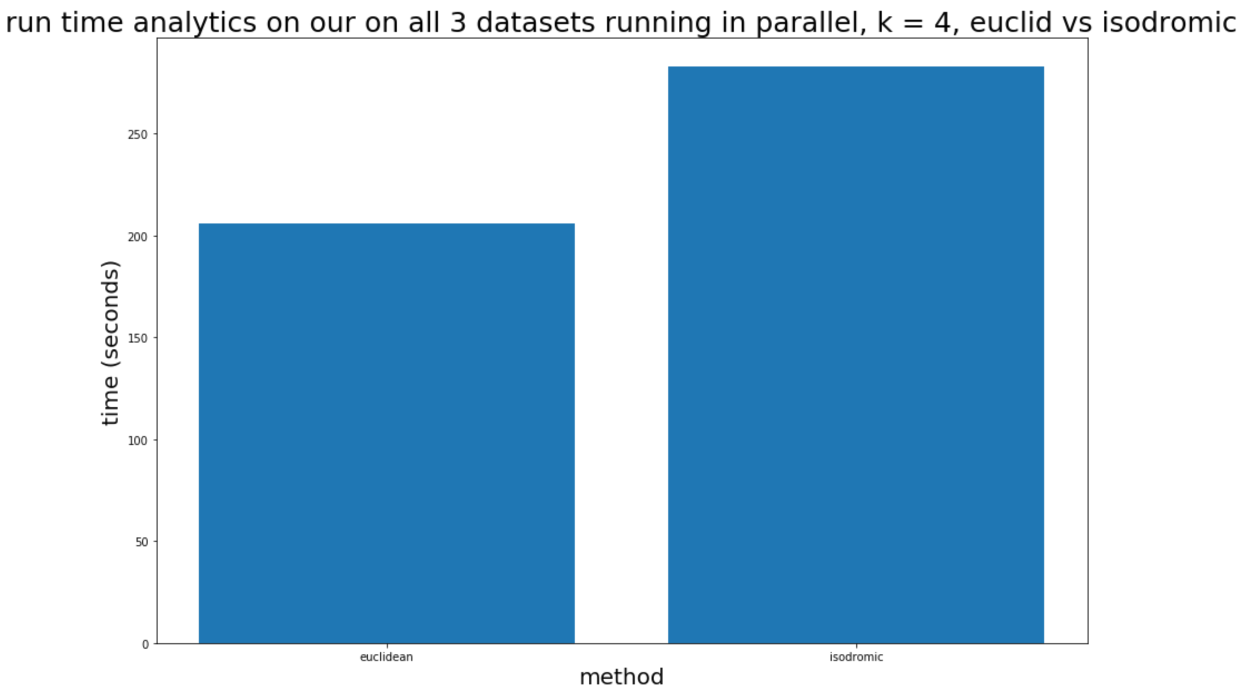


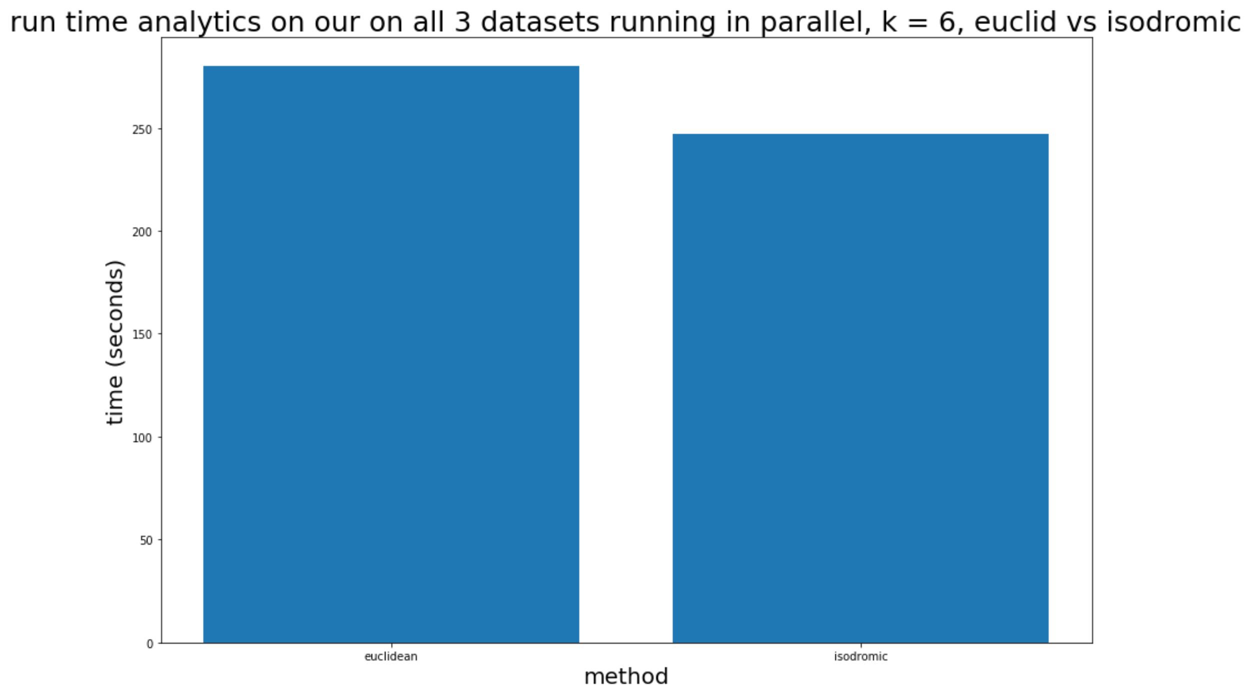

run time analytics on our synthetic dataset when we modify k values, remaining on Euclidean
time (seconds)
0
1
2
3
4
2
3
4
5
6
k


run time analytics on our synthetic dataset when we modify k values, remaining on isodromic
time (seconds)
0
1
2
3
4
5
6
7
2
3
4
5
6
k


run time analytics on our DBPedia dataset when we modify k values, remaining on euclidean
time (seconds)
0
50
100
150
200
k
2
3
4
5
6


run time analytics on our on all 3 datasets running in parallel, k = 2, euclid vs isodromic
time (seconds)
0
20
40
60
80
euclidean
isodromic
method


run time analytics on our on all 3 datasets running in parallel, k = 4, euclid vs isodromic
time (seconds)
0
50
100
150
200
250
euclidean
isodromic
method



Note: I think this last one looks odd due to the fact that I didn't average or get the best run time out of multiple runs. There may have been other underlying processes that impacted this.

5. Conclusion:

I successfully implemented the k means clustering algorithm, and applied it to all three datasets that were supplied in this project. As expected, runtime increased as k was increased, and when using isodromic. Multithreading allowed us to be more efficient, as is evident by the graphs above. I would have improved my work by using the pysparks kmeans function – but it didn't really make sense to me to use that if the problem asked us to find the other 4 functions. I would also improve how I handled the run time analytics by averaging or getting the best run times – to avoid faulty high run times due to other underlying processes.

Also, I'm not sure what the proper k value should be at all times – I genuinely don't think I have enough information to make a valid conclusion on desired k value. If I don't know what the nature of the synthetic dataset is, and so I'm not sure what a desired clustering k would be. MobileNET can be clustered into different areas of California – but why? Also, what is the motive behind clustering DBPedia data? Am I trying to see the articles according to countries, continents, regions – or what? I believe there needs to be more information about the problem at hand to more effectively select the k values.