Geo-Location Clustering with the K-Means Algorithm

Introduction and Motivation:

Clustering is the process of grouping a set of objects (or data points) into a set of k clusters of similar objects. Thus, objects that are similar should be in the same cluster and objects that are dissimilar should be in different clusters.

Clustering has many useful applications such as finding a group of consumers with common preferences, grouping documents based on the similarity of their contents, or finding spatial clusters of customers to improve logistics. More specific use cases are

- *Marketing*: given a large set of customer transactions, find customers with similar purchasing behaviors.
- *Document classification*: cluster web log data to discover groups of similar access patterns.
- *Logistics*: find the best locations for warehouses or shipping centers to minimize shipping times.

I approach the clustering problem by implementing the k-means algorithm. K-Means is a distance-based method that iteratively updates the location of k cluster centroids until convergence. The main user-defined ingredients of the k-means algorithm are the distance function (often Euclidean distance) and the number of clusters k.

The Data contains information collected from mobile devices on Loudacre's network, including device ID, current status, location and so on. Loudacre Mobile is a (fictional) fast- growing wireless carrier that provides mobile service to customers throughout western USA. K-means clustering can be used to determine the information like where a new cell tower can be installed for maximum coverage or where the maximum number of users are.

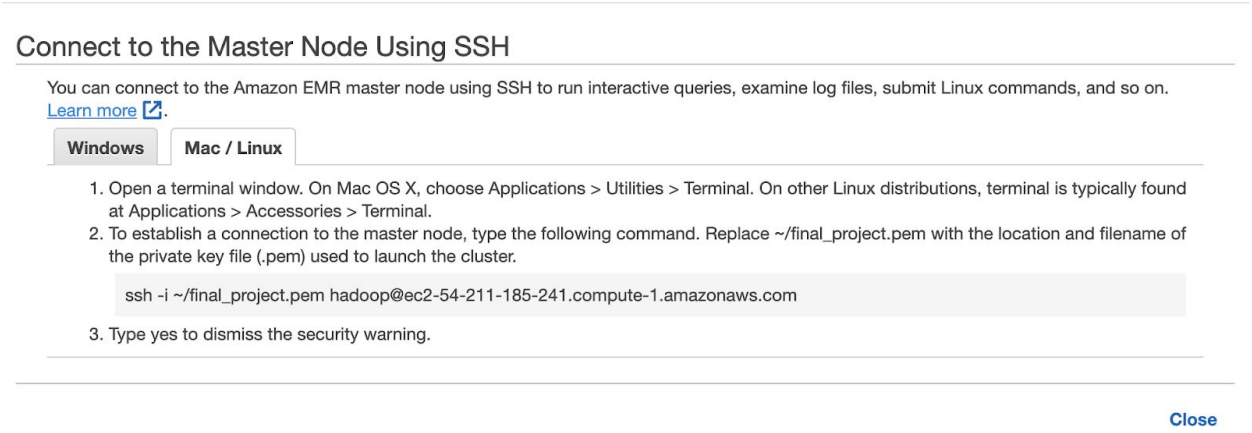
System Configuration:

First we start with AWS EMR configuration. Here are the steps to be followed for setting up EMR service.

- Go to the aws console and click on services where you will find emr, if not search it in the search bar.
- Click on the create cluster button and you will be redirected to a new tab which looks like the screenshot below.
- Choose your cluster name which should be unique. In the software configuration tab, choose spark as application.
- In an instance type choose m4.xlarge
- Create your ec2 key pair for this emr instance and click on create cluster.

After doing this you will see the cluster being started. It might take some time for service to be ready.

- Now open your terminal and use the following command to connect to your emr. We will be using ssh.



After you have created the EMR Spark cluster, and connected it follow these steps:

1. Open the Security Groups for the Master node, and add 2 inbound rules for SSH (anywhere) and Custom TCP Port 8888 (anywhere)
2. run **sudo pip install pyyaml ipython jupyter ipyparallel pandas boto -U**
3. You will need to do two things: set PYSPARK_DRIVER_PYTHON to the location of jupyter and set PYSPARK_DRIVER_PYTHON_OPTS so that it tells the notebook to accept connections from all IP addresses and without opening a browser. To find out where jupyter is installed, you can use the which command (as in which jupyter). Your PYSPARK_DRIVER_PYTHON_OPTS will look familiar is they are the same options you used in the Distributed Systems lab, namely notebook --no-broweser --ip=0.0.0.0 ---port=8888. In the end, you will want to append the following two lines to your .bashrc file:

export PYSPARK_DRIVER_PYTHON=/usr/local/bin/jupyter

export PYSPARK_DRIVER_PYTHON_OPTS="notebook --no-browser --ip=0.0.0.0 --port=8888"

4. Then run source ~/.bashrc followed by pyspark (*not* jupyter notebook). You should see the familiar standard output telling you that Jupyter Notebook is starting.

Note: Make sure you are allowing connections on port 8888 (in Security Groups) and point your browser to the master node:8888 and don't forget the token. (*i.e.* your URL should look something like http://ec2-XXX-XXX-XXX-XXX.compute-1.amazonaws.com:8888/tree?token=0001112223334445556667778889999aaabbbcccdddeeefff#)

5. Copy the link and paste it in the browser, jupyter notebook will be started.

6. For visualization we will be using geopandas. Use the following commands to install geopandas from the terminal. First update pip and install geopandas and descartes.

```
# python -m pip install --user --upgrade pip
# pip3 install geopandas
# pip3 install descartes
```

Now we are ready to go.

Big Data Application:

Data Preparation:

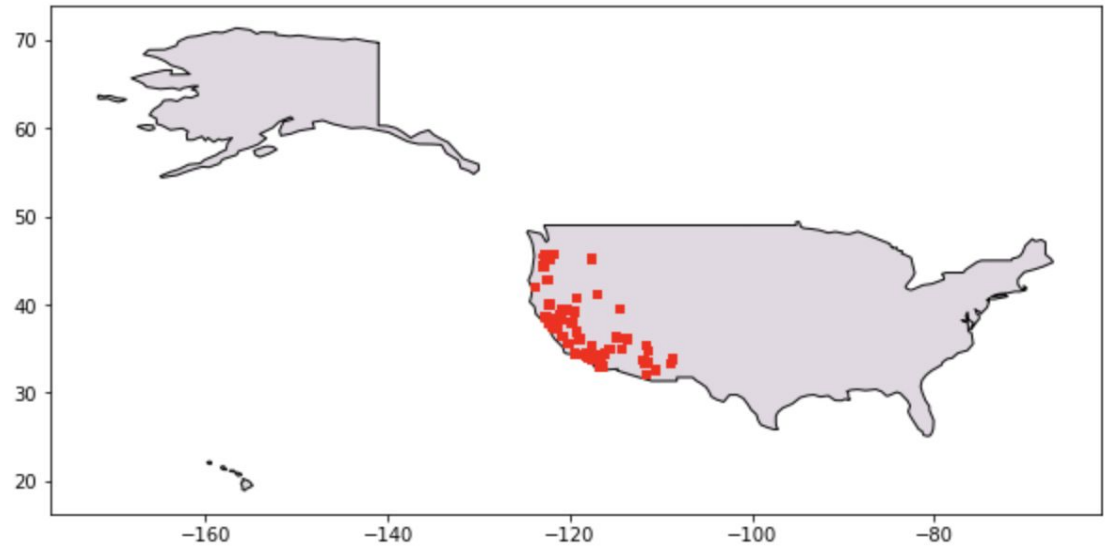
I use pyspark and spark machine learning library for this project. I preprocessed the data before implementing the actual algorithm to convert the dataset into standardized form for algorithm implementation. I did following preprocessing process for device status dataset:

- Load the dataset
- Determine which delimiter to use
- Filter out any records which do not parse correctly each record should have exactly 14 values
- Extract the date, model, deviceID, latitude and longitude.
    - Date: 1st field
    - Model: 2nd field
    - Device Id: 3rd field
    - Latitude: 13th field
    - Longitude: 14th field
- Store longitude and latitude as the first two field
- Filter out locations that have a latitude and longitude of 0.
- The model field contains the device manufacturer and model name (e.g. Ronin S2.) Split this field by spaces to separate the manufacturer from the model (e.g. manufac- turer Ronin, model S2.)
- Save the extracted data to comma delimited text files in S3.

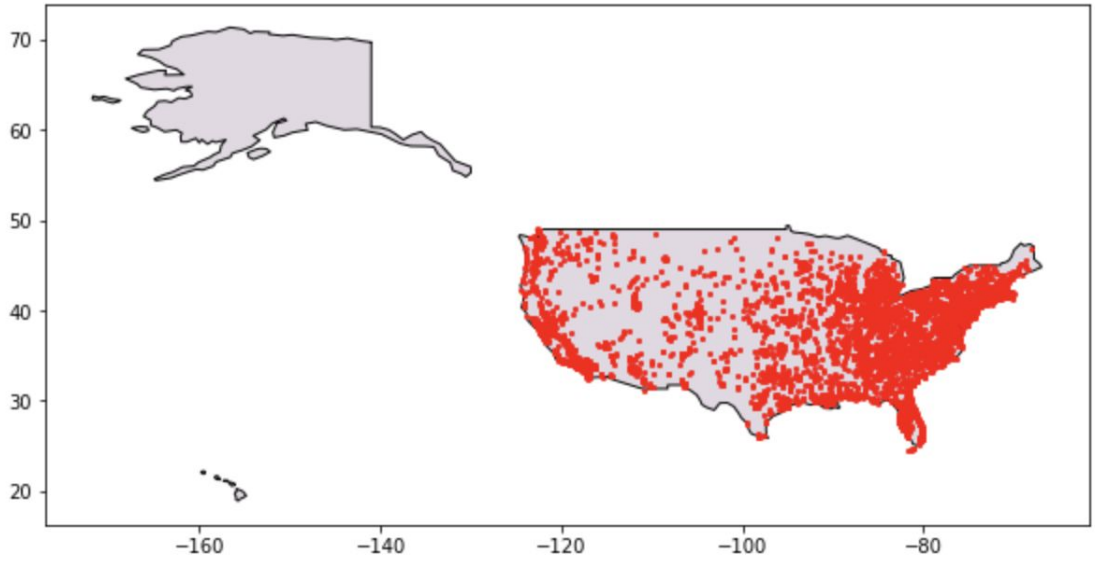
Visualization:

I use the geopanda, descartes python library for data visualization. These are the visualization of data before clustering.

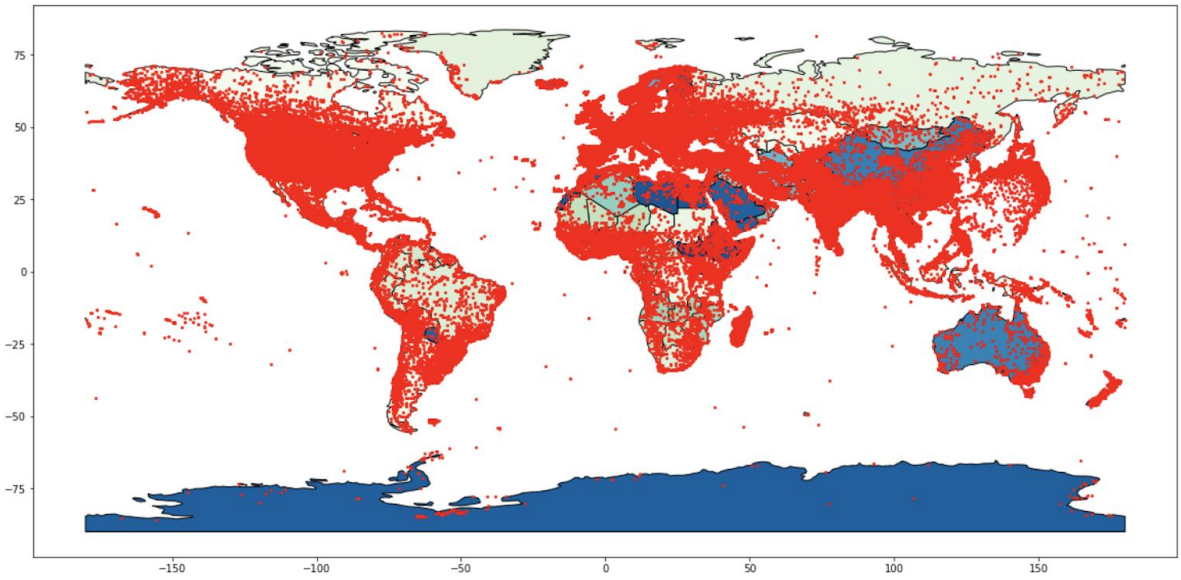
Device Status Data:



Synthetic Location Data:



DBPedia Location Data:



KMeans Clustering Approach:

Kmeans clustering algorithm is an iterative algorithm that partitions the dataset into predefined K subgroups/ clusters where each data point belongs to only one group. It tries to keep similar data together while also keeping the different clusters as far as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid is at the minimum.

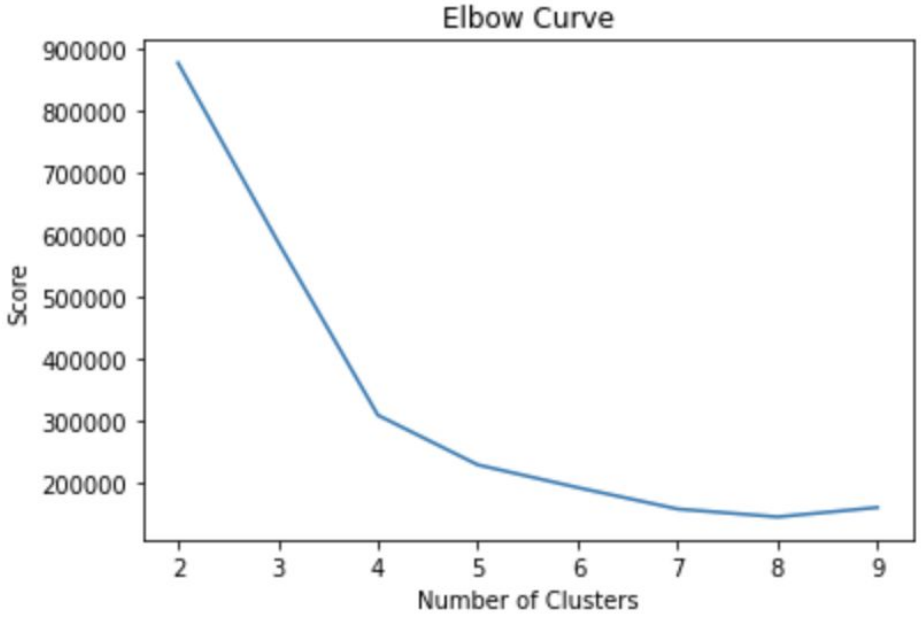
The way Kmeans algorithm works is as follows:

- 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.

Implementation:

Since the data is very large and processing of such is not possible in our local machine, I use AWS EMR as a ApacheSpark service to run all my processes. Datas were stored in an AWS S3 bucket. Before computing Kmeans, I converted the longitude and latitude data into vectors using the VectorAssembler library from pyspark. VectorAssembler combines all of the columns containing features into a single column. This has to be done before modeling can take place because every Spark modeling routine expects the data to be in this form. Then after that in order to find the right number of clusters for device status data, I use elbow curve to find the best K possible, providing the range of K's to the model and finding the best values of K for clustering. I calculated cost and plot to find the right cluster. From the curve we can clearly see that 4,5,6 could be the perfect cluster for this data.

Below is the figure showing elbow curve for device status data:



Now, the main part is to compute Kmeans. I created a general function to calculate kmean, transform dataframe, euclidean distance, sum of sq errors and center.

Results:

Device Location Data:

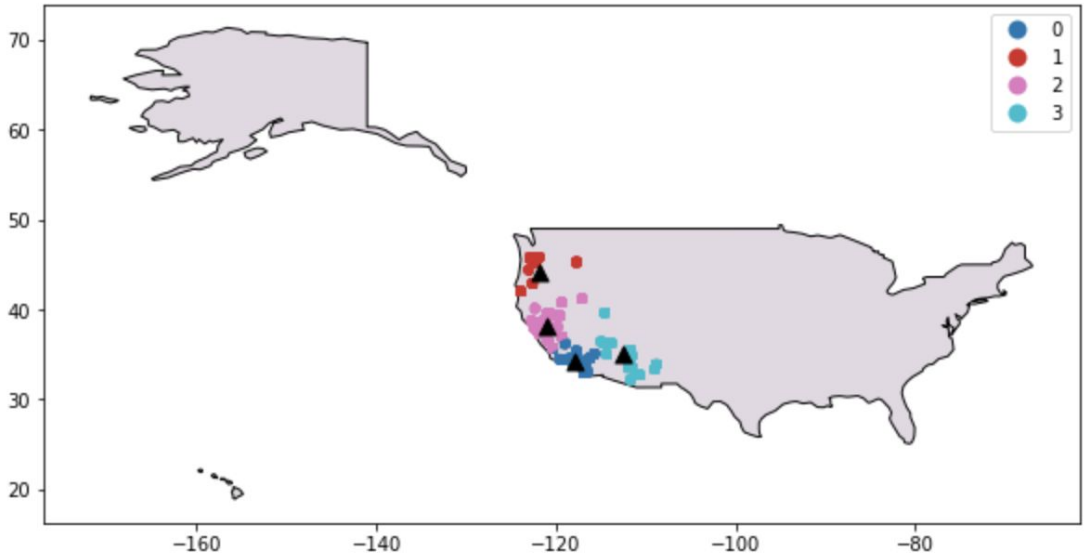
Cluster and visualize the device location data:

1. **K=4 DistanceMethod = Euclidean Distance**

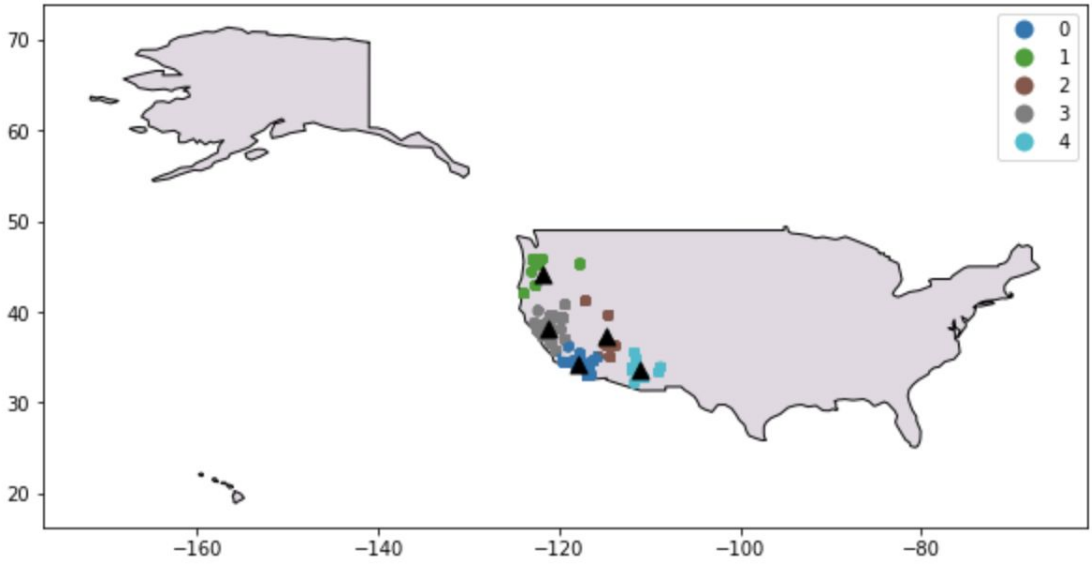
```
# Calculate for DEVICE STATUS DATAFRAMEs

prediction_df, centers = kmeans_model(df=device_status_df, k=4, seed=1)
```

```
Silhouette with squared euclidean distance = 0.7540828544162818
Within Set Sum of Squared Errors = 309322.4486914122
Cluster Centers:
[  34.30404014 -117.8032879 ]
[  44.23926087 -121.79580631]
[  38.19538776 -121.08051278]
[  35.08461054 -112.57140921]
```



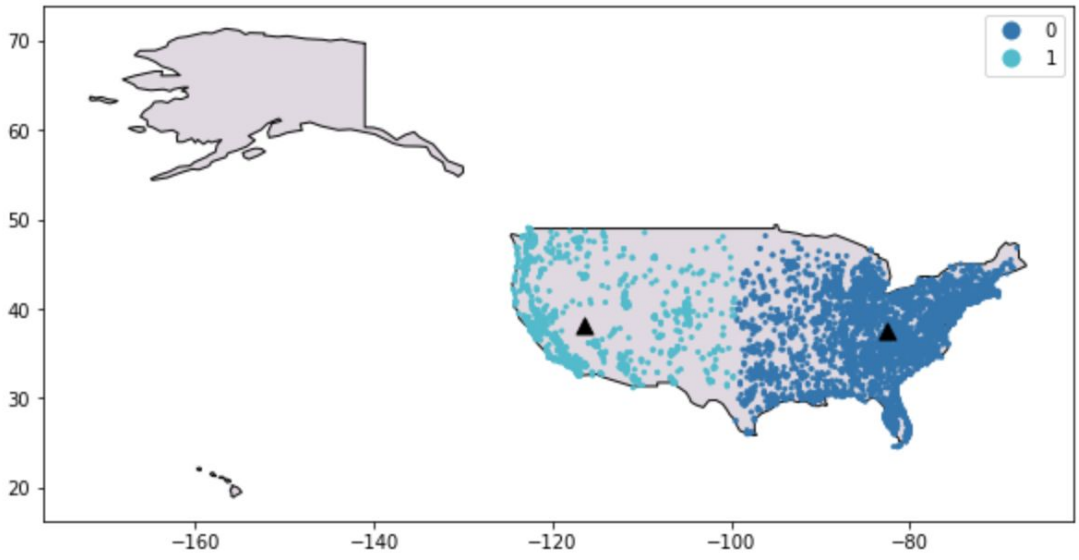
2. **K=5 DistanceMethod = Euclidean Distance**



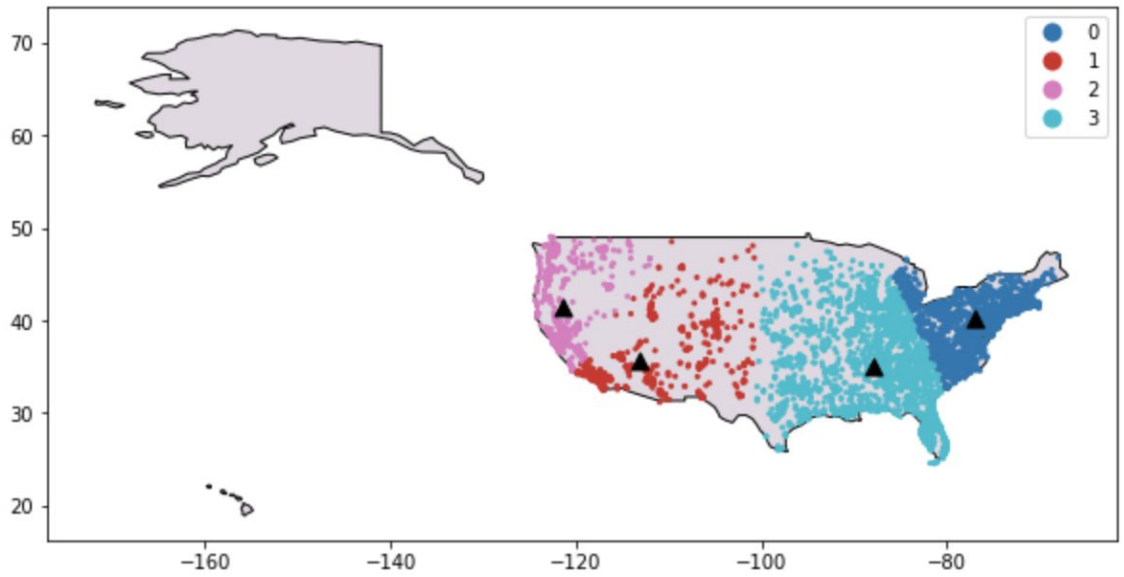
Synthetic location data:

Cluster and visualize the device location data:

1. **K=2, Euclidean distance**

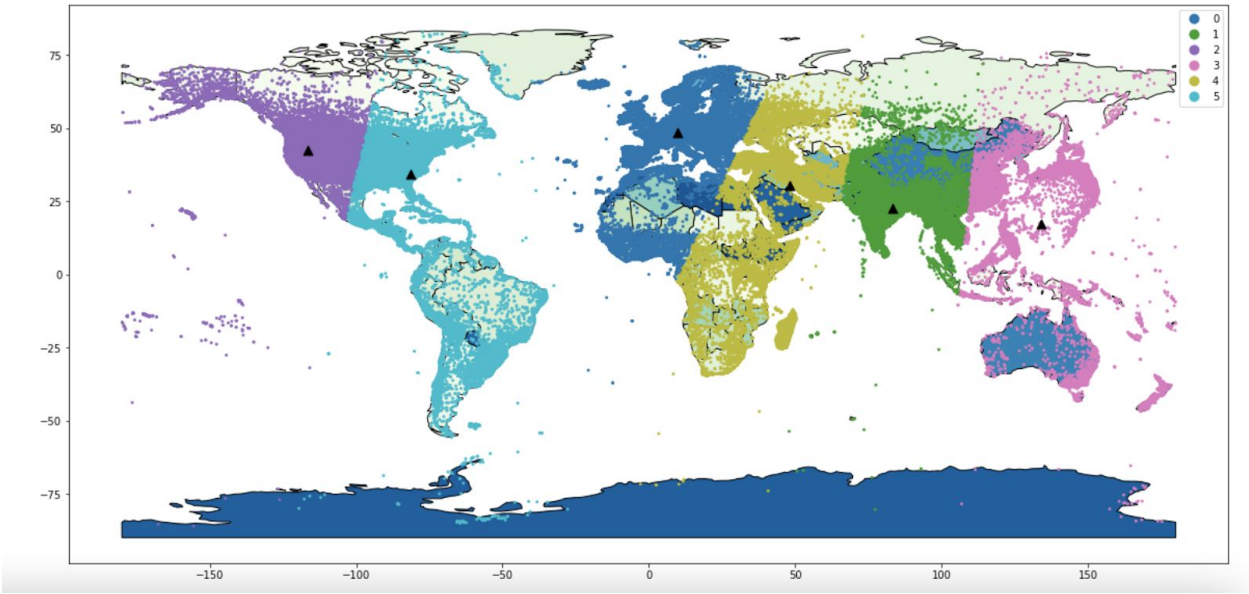


2. **K=4, Euclidean distance**



DBpedia location data:

1. **K=6, Euclidean distance**



Runtime Analysis:

Dataset	K	Persist/Cache	Measure	Run Time(ms)
Device location	**5**	**Y**	**Euclidean distance**	**44.8 ms**
Device location	**5**	**N**	**Euclidean distance**	**112.3 ms**
Synthetic location	**4**	**Y**	**Euclidean distance**	**47.4 ms**
Synthetic location	**4**	**N**	**Euclidean distance**	**120.8 ms**
DBpedia location	**6**	**Y**	**Euclidean distance**	**82.6 ms**
DBpedia location	**6**	**N**	**Euclidean distance**	**350.9 ms**

Since datasets were cached and runned in Aws EMR it didn't take much time to run the model. However, when not cached it took a bit longer.

Conclusion:

This project was built using Aws EMR, Aws S3 bucket, PySpark, geopandas for visualization jupyter notebook to implement K-means Clustering on all three data sets: device location, synthetic location and DBpedia location. I was successfully able to cluster these three datasets first by preprocessing it, extracting data from the log file, splitting it and storing it in a pyspark dataframe. Then I did some transformations to the data so it can be used in a clustering model. Finally I applied KMeans clustering to find the centroid and cluster the data as per K values. In order to find the meaningful information from each dataset, various k values were experimented. Using geopandas for visualization, I was able to see the exact output we wanted ie: clusters and their centroid.