FINAL PROJECT Option #1:

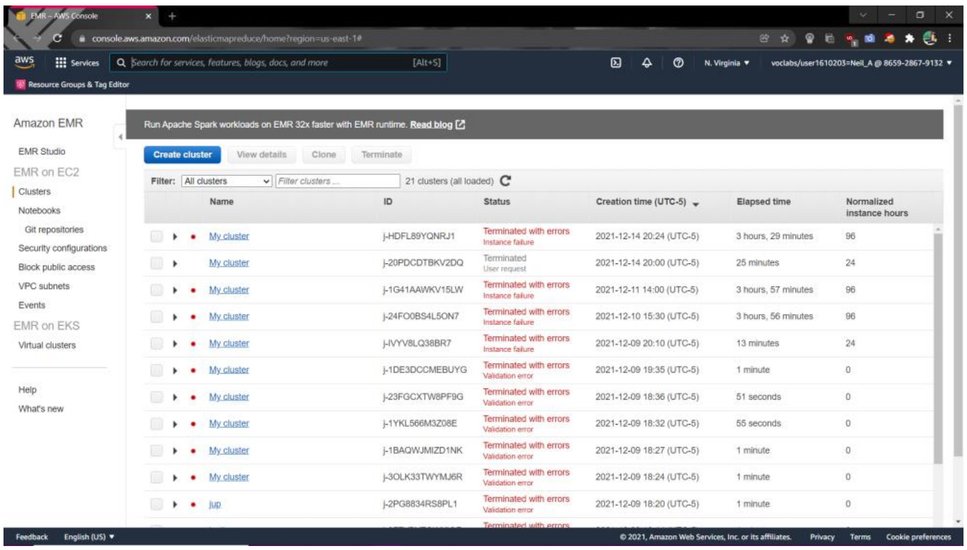
GEO-LOCATION CLUSTERING USING THE k-MEANS ALGORITHM

Introduction and Motivation:

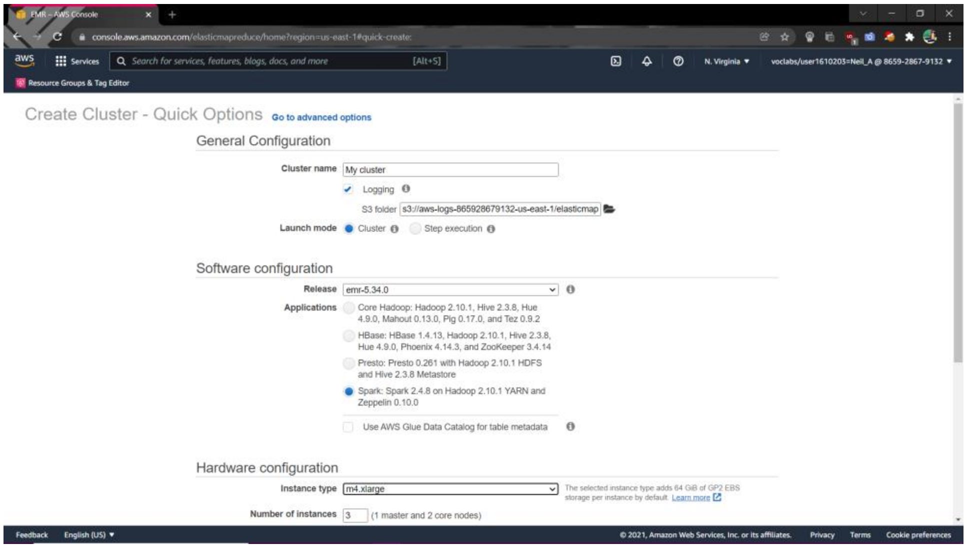
Clustering is a process where a similar kind of data is within a same cluster, but distinct kinds of data are within different clusters. In a regular basis, we can use clustering in business, marketing, logistics, data documentation etc. In this project we used k-means clustering in geo-location.

Data Preparation

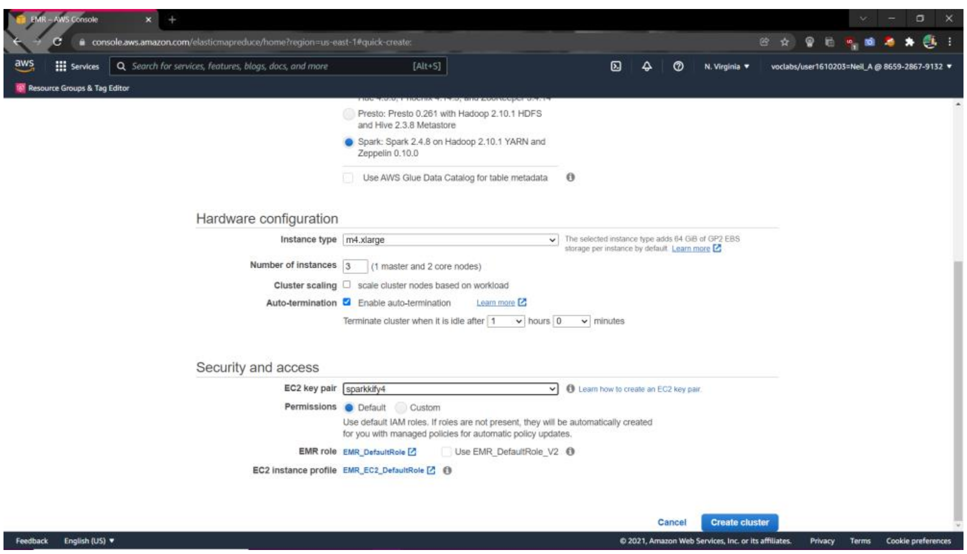
1. MAKE AN EMR CLUSTER



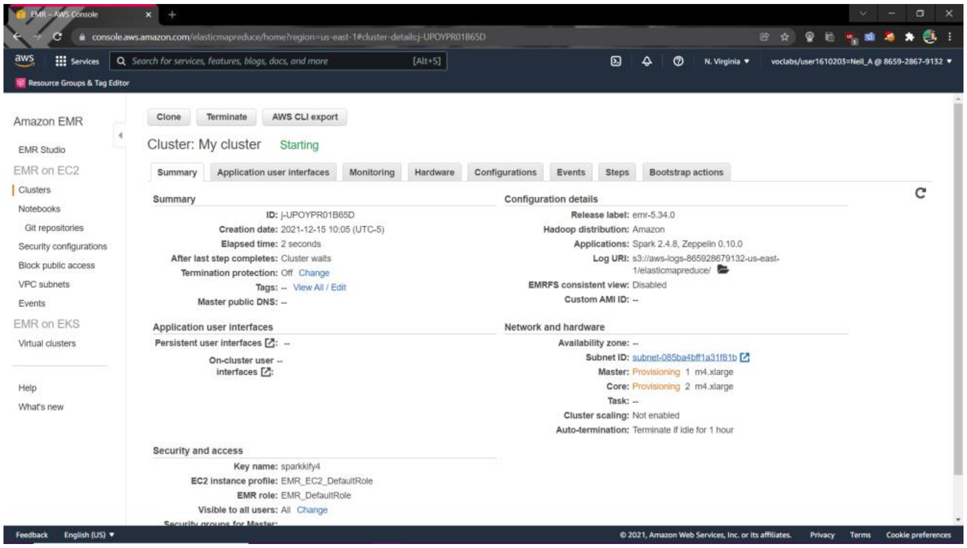
2. Name your EMR cluster along with hardware config as m4.xlarge and software config as spark



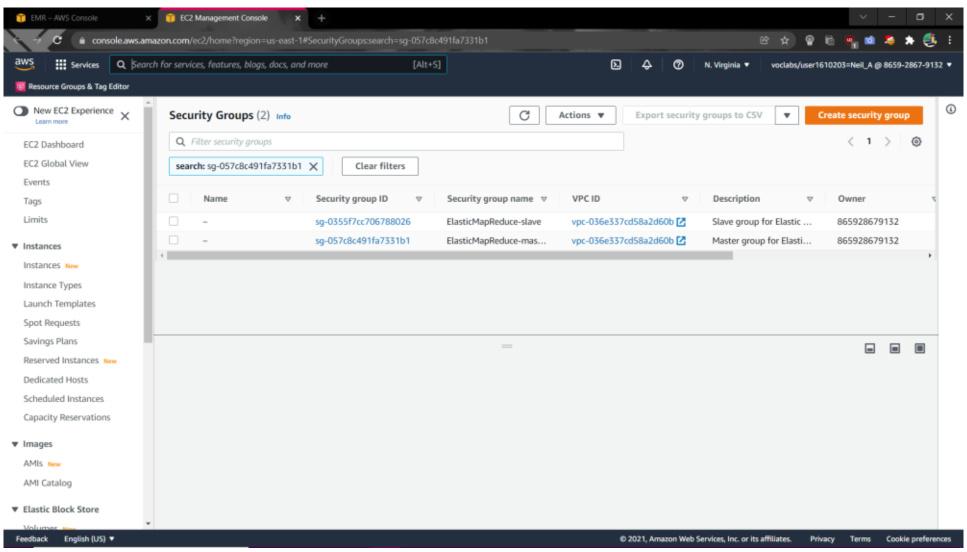
3. Choose your key pair



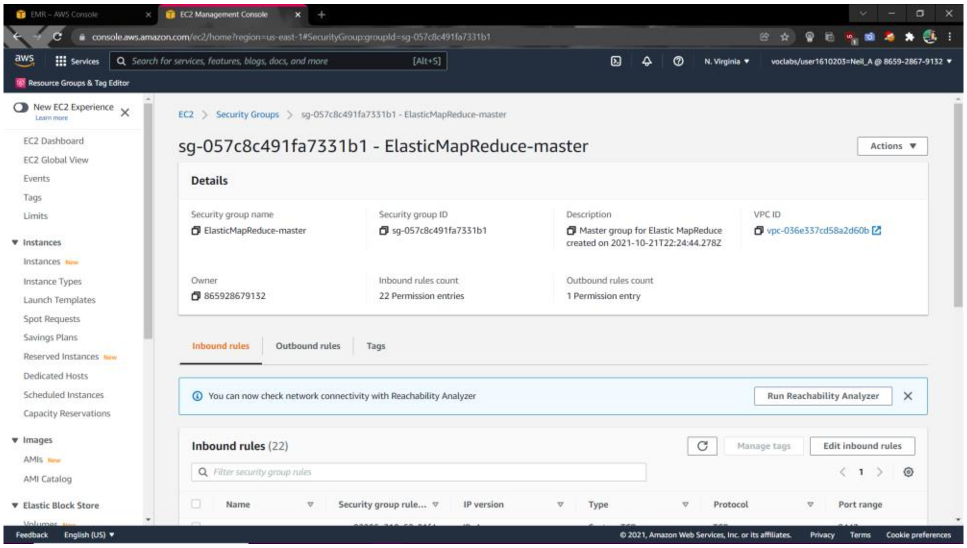
4. Click on create cluster



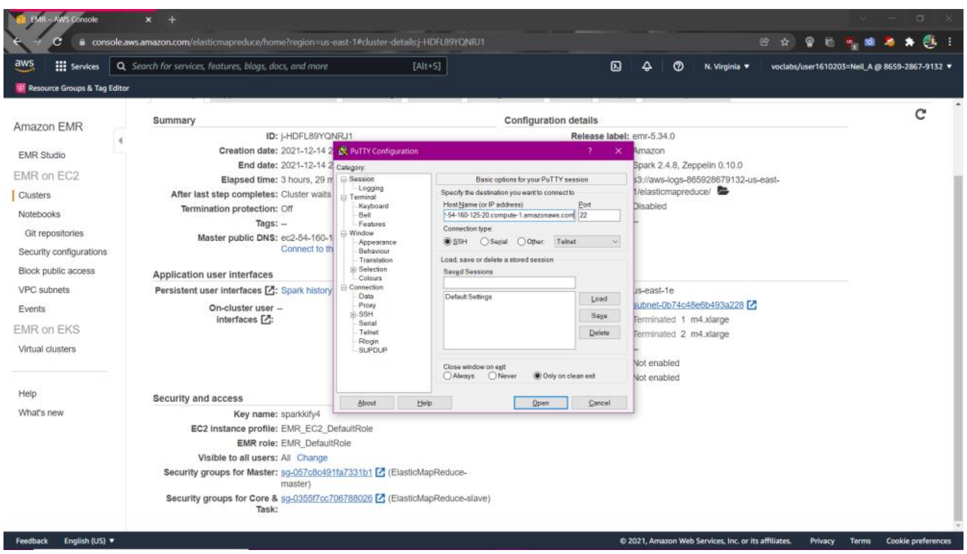
5. Goto security and access and click on master group



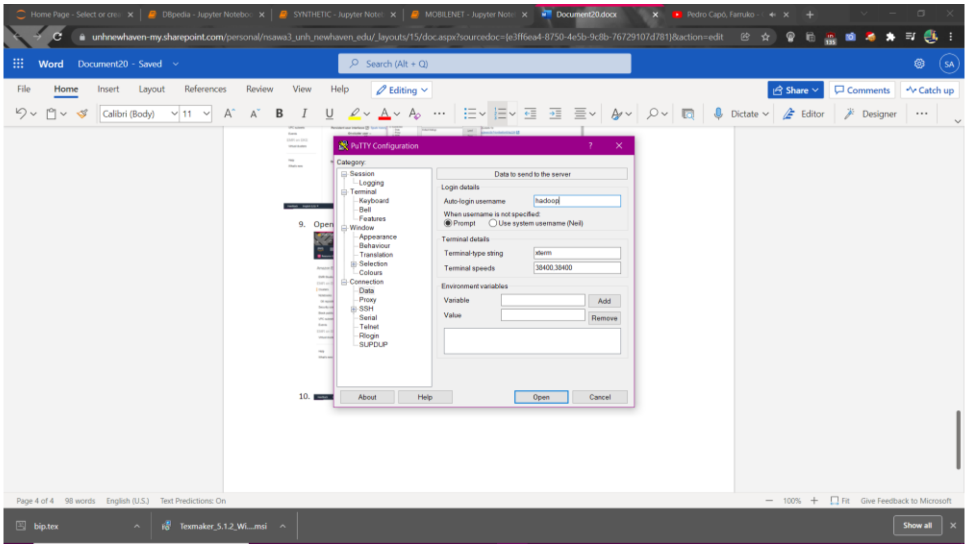
6. Open the master group and click on edit inbound rules



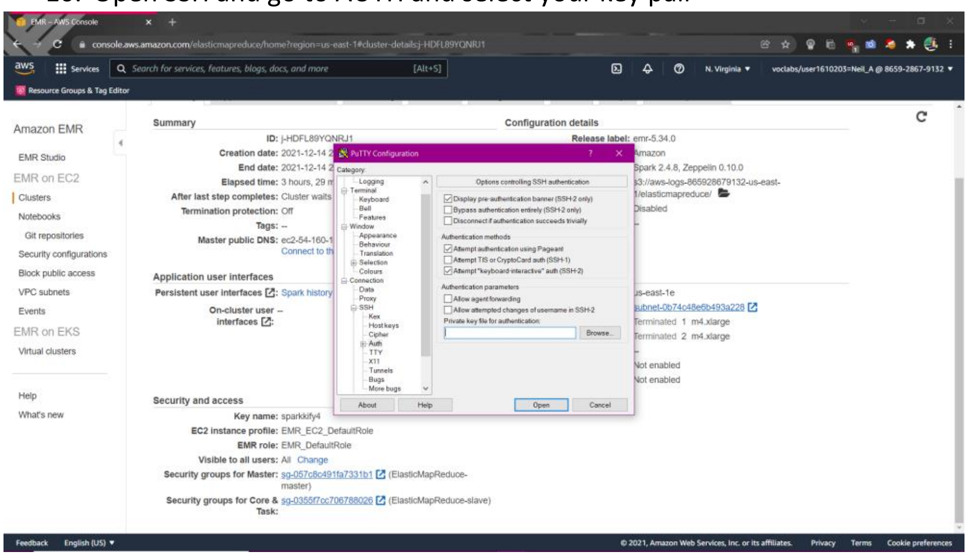
7. Create rules with ports 22 as SSH and 8888 as custom
8. Open putty for windows users and hostname as "hadoop@your Master public DNS address"



9. Open data in connection and write "hadoop" in auto login username



10. Open SSH and go to AUTH and select your key pair

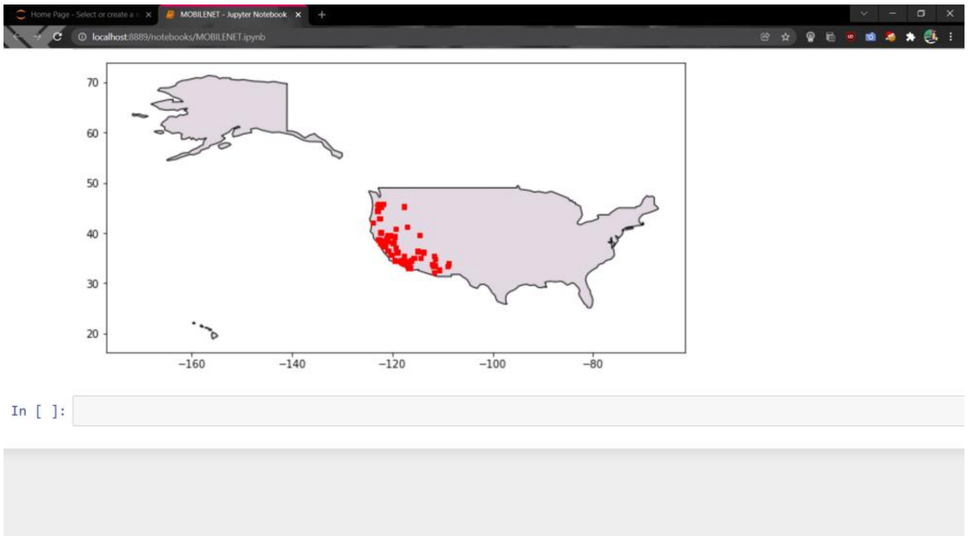


11. You will see your putty opened now type the below commands

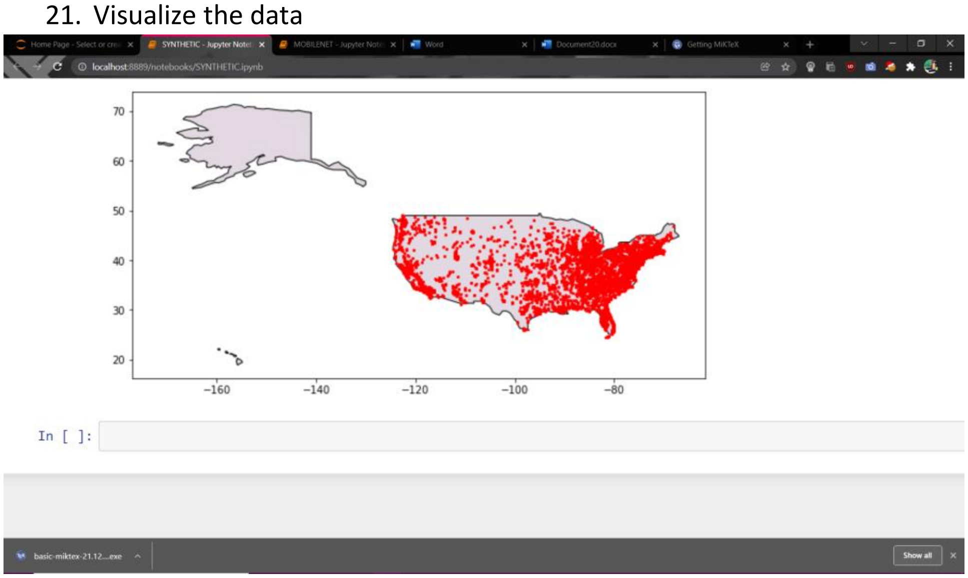
- sudo su
- Python3 –m pip install pyyaml ipython jupyter ipyparallel pandas boto –U
- Type exit
- `export PYSPARK_DRIVER_PYTHON=/usr/local/bin/jupyter`
- `export PYSPARK_DRIVER_PYTHON_OPTS="notebook --no-browser --ip=0.0.0.0 --port=8888"`
- `Source ~/.bashrc`
- `pyspark`

12. You will see your address to run jupyter notebook now copy paste the address in your browser

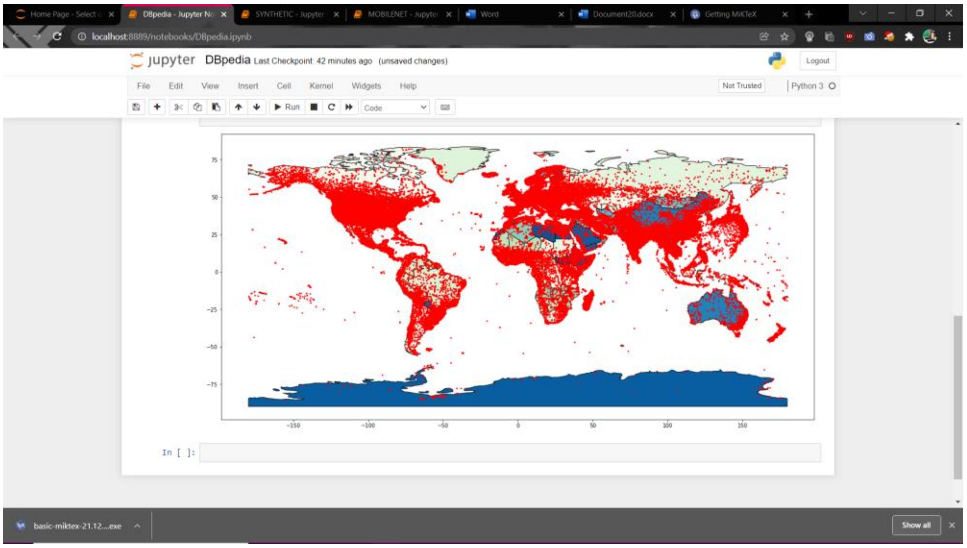
13. Now replace the part of your address that is before port number in my case ":8888" with your EMR cluster Public DNS
14. Now open a new jupyter notebook python file
15. We have 4 in total jupyter notebook files
16. Name the first file MOBILENET and use the data from device_status
17. We created a data frame to see the required coloumns
18. Visualize the data by plotting it



19. In a new jupyter notebook named Synthetic use data from file sample geo
20. We created a dataframe
21. Visualize the data



22. Again in a new jupyter notebook named DBpedia use data from Lat_longs file
23. We created new dataframe
24. Visualize it



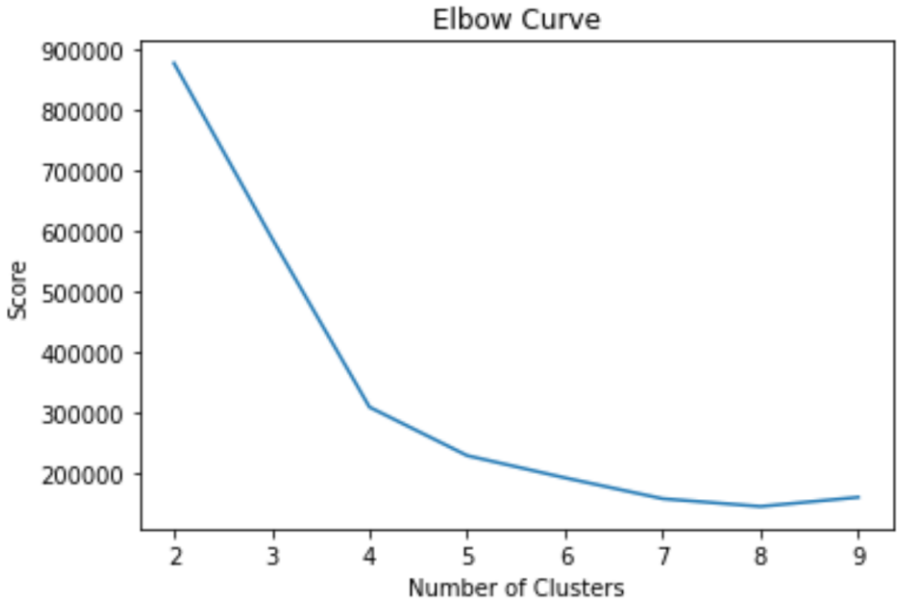
Clustering Big Data – k-means in Spark

k-means

1. K-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes.
2. To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids
    - It halts creating and optimizing clusters when either:
    - The centroids have stabilized — there is no change in their values because the clustering has been successful.
    - The defined number of iterations has been achieved

Implementation

Since the data is big, so in order to execuation of the model faster, we used AWS. Create an S3 bucket and upload all three folders. Then create an EMR cluster and run spark in browser using putty. Convert latitude and longitude coordinates to vector using vector assembler from pyspark. Combines all columns with features together. Done before the model cause the data has to be in vector form. To find the best K possible use elbow curve. Plot the curve against number of clusters and score. Now we find K means

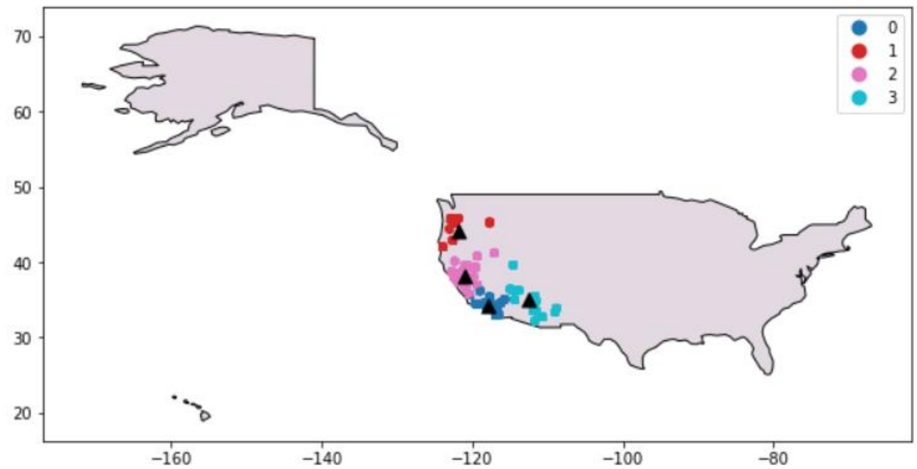


Results

1.Device location data:
Cluster and k=4 and seed = 1

```
%time 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]
CPU times: user 37.3 ms, sys: 4.27 ms, total: 41.6 ms
Wall time: 2.86 s
```




```
CPU times: user 16.9 s, sys: 163 ms, total: 17.1 s
Wall time: 17.2 s
```

3. K=5 and distance method is euclidean method

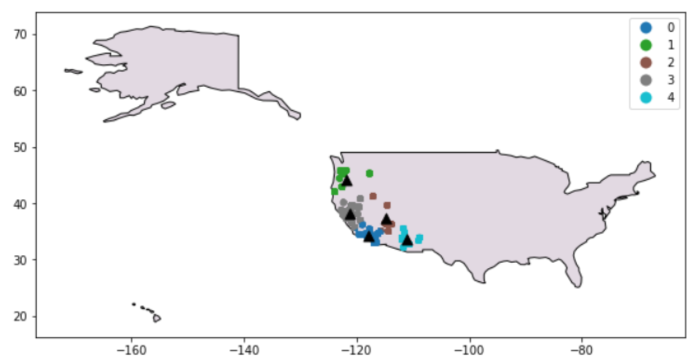
```
In [23]: # Step 3: Compute and Visualize Clusters

         # Calculate the k-means clusters for the device location data using k = 5.

         %time prediction_df, centers = kmeans_model(df=device_status_df, k=5, seed=1)

         %time plot_data(prediction_df, a=0, after_prediction=True)
```

```
Silhouette with squared euclidean distance = 0.7911724452946861
Within Set Sum of Squared Errors = 229329.08140511927
Cluster Centers:
[  34.29368632 -117.81213281]
[  44.23926087 -121.79580631]
[  37.33148137 -114.84963868]
[  38.0957093  -121.19758985]
[  33.68736796 -111.04079699]
CPU times: user 53.3 ms, sys: 451 µs, total: 53.7 ms
Wall time: 2.83 s
```




```
CPU times: user 15.6 s, sys: 156 ms, total: 15.7 s
Wall time: 16 s
```

Synthetic location data

1. When k=2 distance is euclidean

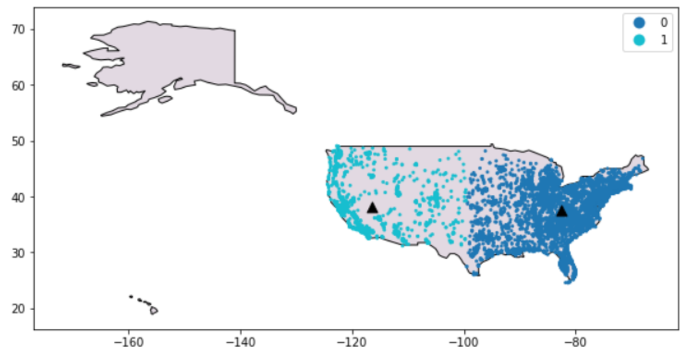
```
In [24]: # Calculate the k-means clusters for the synthetic location data using k = 2 and k = 4.

print("Calculating the k-means clusters for the synthetic location data using k = 2")

# calculate for sample geo data
%time prediction_df, centers = kmeans_model(df=sample_geo_log_df, k=2, seed=1)

%time plot_data(prediction_df, a=0, after_prediction=True)
```

```
Calculating the k-means clusters for the synthetic location data using k = 2
Silhouette with squared euclidean distance = 0.8525189358156998
Within Set Sum of Squared Errors = 704244.7373775935
Cluster Centers:
[ 37.5647472  -82.55711082]
[  38.07161548 -116.43342043]
CPU times: user 41.8 ms, sys: 197 µs, total: 42 ms
Wall time: 1.15 s
```




```
CPU times: user 1.87 s, sys: 140 ms, total: 2.01 s
Wall time: 1.91 s
```

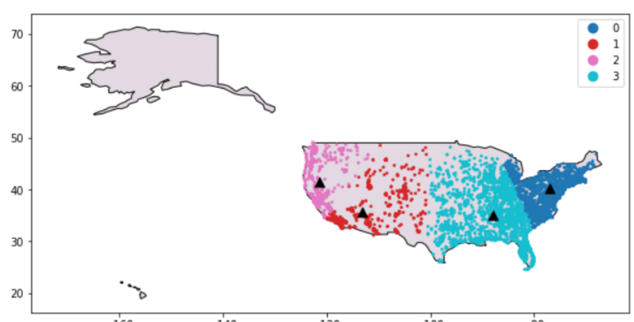
2. When k=4 and distance is euclidean

```
In [25]: print("Calculating the k-means clusters for the synthetic location data using k = 4")

%time prediction_df, centers = kmeans_model(df=sample_geo_log_df, k=4, seed=1)

%time plot_data(prediction_df, a=0, after_prediction=True)
```

```
Calculating the k-means clusters for the synthetic location data using k = 4
Silhouette with squared euclidean distance = 0.5548689033882502
Within Set Sum of Squared Errors = 365558.3934312812
Cluster Centers:
[ 40.14836238 -76.96598964]
[  35.57495009 -113.07189577]
[  41.49405835 -121.33793416]
[ 35.11449774 -87.93102449]
CPU times: user 41.1 ms, sys: 375 µs, total: 41.5 ms
Wall time: 1.18 s
```




```
CPU times: user 2.48 s, sys: 120 ms, total: 2.6 s
Wall time: 2.41 s
```

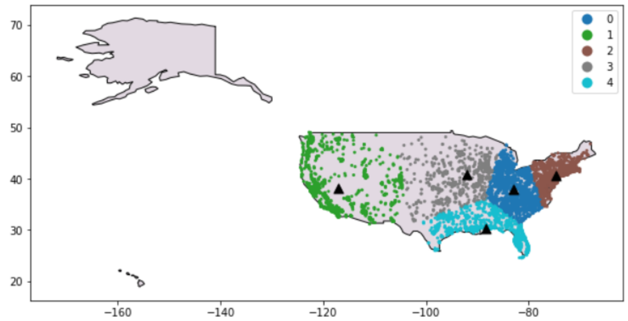
3. K=5 and distance is euclidean

```
In [26]: print("Calculating the k-means clusters for the synthetic location data using k = 5")

         %time prediction_df, centers = kmeans_model(df=sample_geo_log_df, k=5, seed=1)

         %time plot_data(prediction_df, a=0, after_prediction=True)
```

```
Calculating the k-means clusters for the synthetic location data using k = 5
Silhouette with squared euclidean distance = 0.579380126122293
Within Set Sum of Squared Errors = 276292.5761643455
Cluster Centers:
[ 38.03380474 -82.91911996]
[  38.08274514 -116.98428691]
[ 40.689765   -74.77185746]
[ 40.76409505 -91.99226496]
[ 30.22653902 -88.43888705]
CPU times: user 36.5 ms, sys: 8.14 ms, total: 44.6 ms
Wall time: 1.59 s
```




```
CPU times: user 1.94 s, sys: 104 ms, total: 2.04 s
Wall time: 1.9 s
```

DBpedia Location data

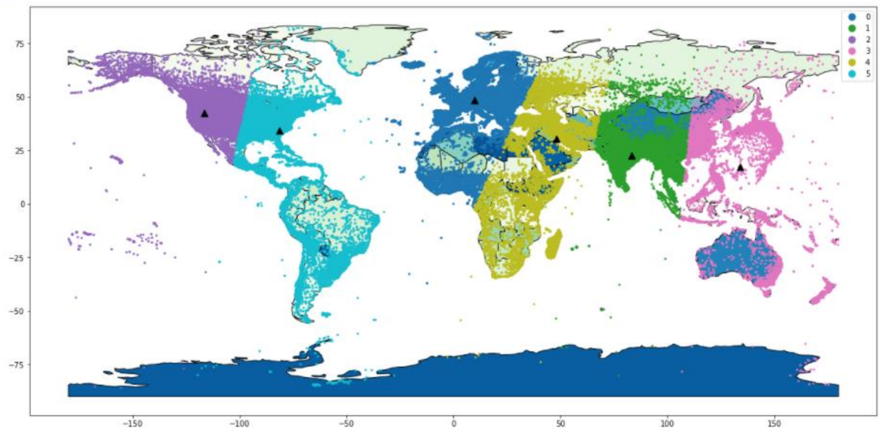
1. **When k=6 and euclidean distance**

```
print("Calculating the k-means clusters for the synthetic location data using k = 6")
# calculate for long lat data
%time prediction_df, centers = kmeans_model(df=long_lats_log_df, k=6, seed=1)

%time plot_data(prediction_df, a=1, after_prediction=True)
```

```
Calculating the k-means clusters for the synthetic location data using k = 6

Silhouette with squared euclidean distance = 0.7199107240404778
Within Set Sum of Squared Errors = 121593903.3697077
Cluster Centers:
[48.59554363  9.7731195 ]
[22.591739   83.18051205]
[  42.38381357 -116.64759113]
[ 17.2112931  134.06313631]
[30.47517672 48.12423989]
[ 34.19284121 -81.42496763]
CPU times: user 43.7 ms, sys: 4.31 ms, total: 48 ms
Wall time: 4.89 s
```




```
CPU times: user 1min 14s, sys: 453 ms, total: 1min 15s
Wall time: 1min 16s
```

Runtime Analysis:

Dataset	K	Measure	Runtime
Device location	5	Euclidean distance	53.7 ms
Synthetic location	2	Euclidean distance	42 ms
Synthetic location	4	Euclidean distance	41.5 ms
Synthetic location	5	Euclidean distance	44.6 ms
DBpedia location	6	Euclidean distance	48 ms

Conclusion:

The project was built in AWS EMR, S3 bucket, pyspark and jupyter notebook. I uploaded all the dataset to S3 bucket and then the data was extracted to Jupyter notebook from the log file. I used geopandas for visualization in jupyter notebook. K means clustering was applied to the dataset. Then cluster all the three datasets by preprocessing, extracting and then splitting and then storing it in pyspark dataframe. Apply k means clustering to find centroid and cluster the data as per k values. Use different k values for better output results. Geopandas visualized the perfect output of clusters and centroids.