

Geolocation Clustering

using the K-Means Algorithm

PROJECT GOAL:

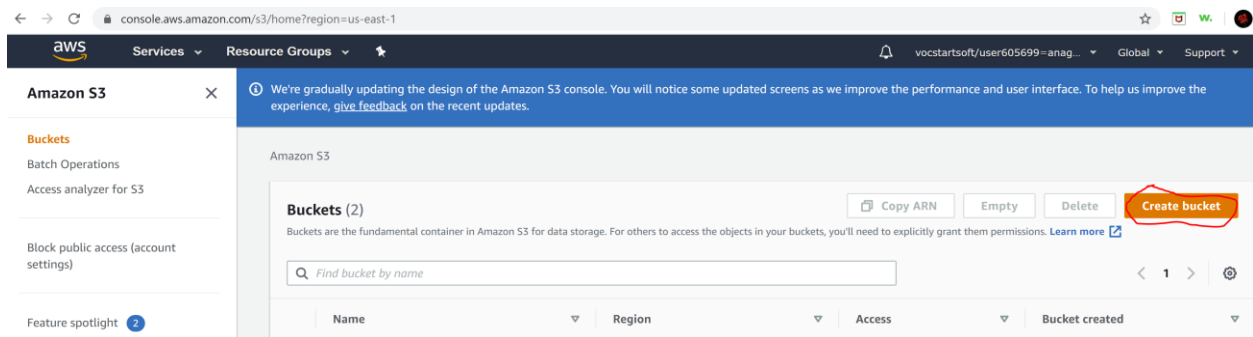
Implement k-means in SPARK and use it for geo-location clustering on various datasets of spatial locations.

PROCEDURE:

Configuration steps

1. Download **devicestatus.txt**, **sample_geo.txt**, **lat_lons.txt** dataset files on my PC from blackboard.
2. Log into AWS educate console → browse to S3 services → **create a new S3 bucket**

Click on “create bucket”



Specify name for your S3 bucket (it must be unique across AWS).

Uncheck “Block all public access” check box

General configuration

Bucket name

geo-project-data

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

Region

US East (N. Virginia) us-east-1

Bucket settings for Block Public Access

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)



Block all public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

Check the box to acknowledge changes → click on “create Bucket”



Turning off block all public access might result in this bucket and the objects within becoming public

AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.



acknowledge that the current settings might result in this bucket and the objects within becoming public.

► Advanced settings

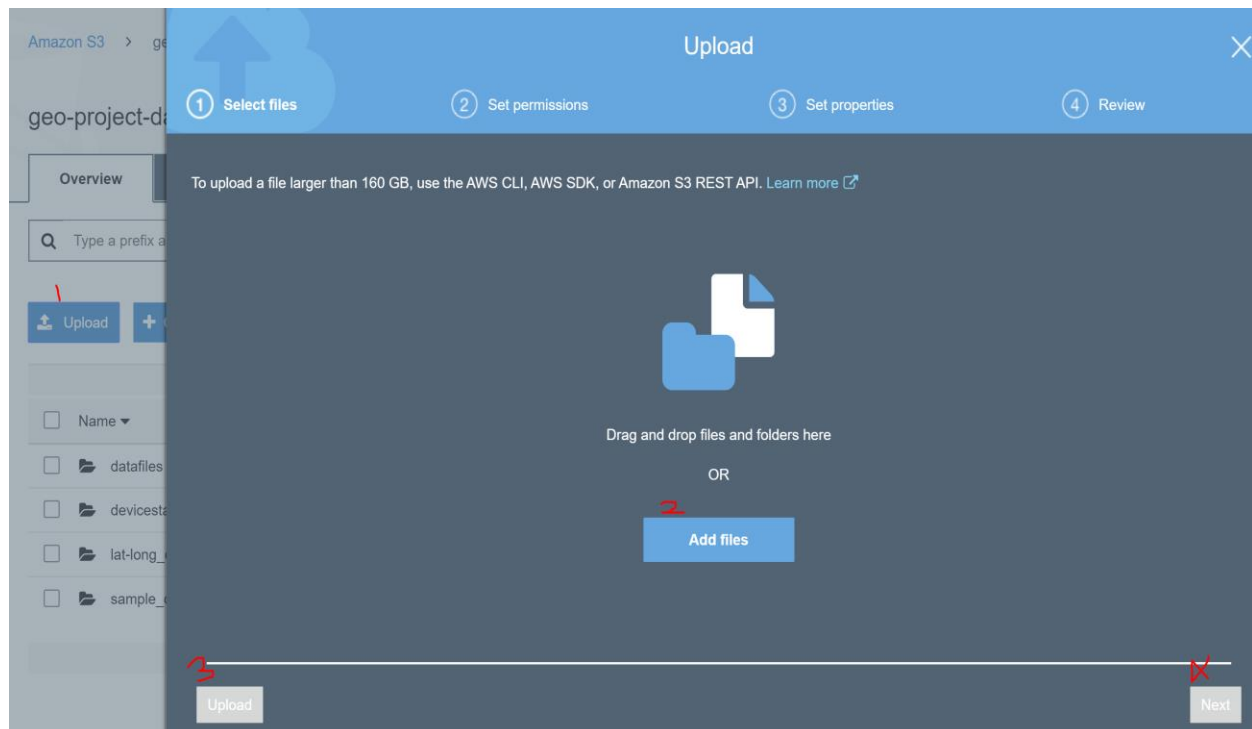
Cancel

Create bucket

3. To **upload devicestatus.txt dataset** to **S3** bucket

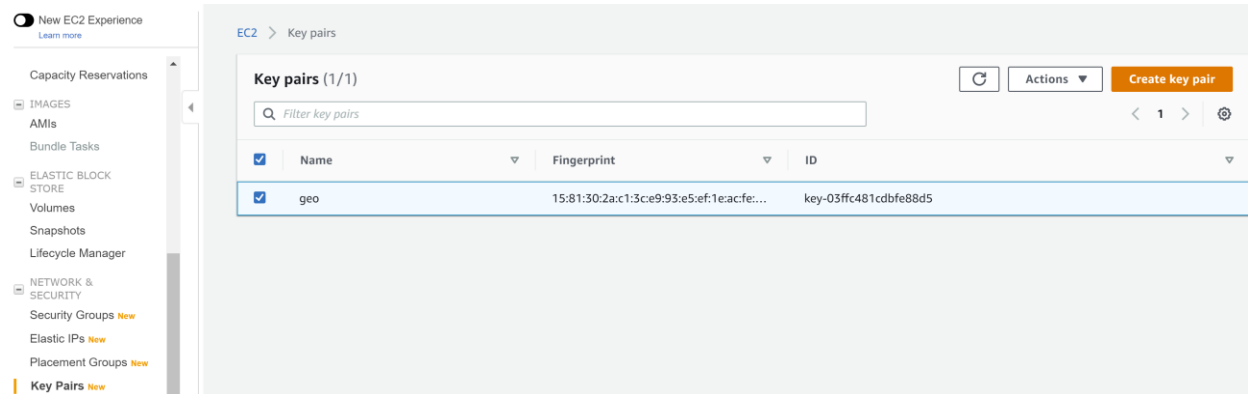
Click on bucket name -> upload -> Add files -> drop or browse files from pc -> upload ->Next

Set permissions (optional) -> set properties(optional) -> upload.



4. Key pair creation

Under EC2 service → “Network & Security” → “Key Pairs”



Select “Create key pair” → Specify key pair name & choose file format → Create key pair

EC2 > Key pairs > Create key pair

Create key pair

Key pair

A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.

Name

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

File format

☒ pem
For use with OpenSSH

☐ ppk
For use with PuTTY

CancelCreate key pair

One you click on “create key pair”, it gets downloaded on your system automatically.

5. Creating EMR cluster

Browse to EMR service under Analytics category,

Click on “Create cluster” -> “Go to advanced options” ->

The screenshot shows the AWS Management Console interface for creating an EMR cluster. The top navigation bar includes the AWS logo, 'Services', 'Resource Groups', and a user profile. The left sidebar shows 'Amazon EMR' with options for 'Clusters' and 'Security configurations'. The main content area is titled 'Create Cluster - Quick Options' and includes a 'Go to advanced options' link highlighted in yellow. Below this, the 'General Configuration' section is visible, showing a 'Cluster name' field with the value 'My cluster' and a 'Logging' checkbox that is checked. The 'S3 folder' is set to 's3://aws-logs-230483383016-us-east-1/elasticmapred'.

Select software packages required for project → Next

The screenshot shows the 'Software Configuration' page in the AWS EMR console. The 'Release' dropdown is set to 'emr-5.29.0'. Below this, there are three columns of software packages, each with a checkbox. The first column includes Hadoop 2.8.5, JupyterHub 1.0.0, Ganglia 3.7.2, Hive 2.3.6, MXNet 1.5.1, Hue 4.4.0, and Spark 2.4.4. The second column includes Zeppelin 0.8.2, Tez 0.9.2, HBase 1.4.10, Presto 0.227, Sqoop 1.4.7, Phoenix 4.14.3, and HCatalog 2.3.6. The third column includes Livy 0.6.0, Flink 1.9.1, Pig 0.17.0, ZooKeeper 3.4.14, Mahout 0.13.0, Oozie 5.1.0, and TensorFlow 1.14.0. The 'Hadoop 2.8.5' checkbox is checked.


Hardware configuration → for Master & Core edit Instance type to “m4.xlarge” → save → Next


The screenshot shows the 'Instance types' dialog box in the AWS EMR console. The dialog has a table with the following columns: Instance type, Number of instances, vCPUs, and Memory. The 'm4.xlarge' instance type is highlighted in yellow. The table lists various instance types including m2.xlarge, m2.2xlarge, m2.4xlarge, m3.xlarge, m3.2xlarge, m4.large, m4.xlarge, m4.2xlarge, m4.4xlarge, m4.10xlarge, and m4.16xlarge. The 'm4.xlarge' instance type is selected, and the 'Save' button is highlighted in yellow.


Specify cluster name below,


General Options

Cluster name

☒ Logging 


S3 folder 

☒ Debugging 

☒ Termination protection 

Under “Security options” → select EC2 key pair → create cluster

Security Options

EC2 key pair 

☒ Cluster visible to all IAM users in account 

Permissions

☒ Default ☐ Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.


EMR role [EMR_DefaultRole](#) 

EC2 instance profile [EMR_EC2_DefaultRole](#) 

Auto Scaling role [EMR_AutoScaling_DefaultRole](#) 

► Security Configuration

► EC2 security groups

 No EC2 key pair has been selected, so you will not be able to SSH to this cluster. [Learn how to create an EC2 Key Pair.](#)

[Cancel](#)

[Previous](#)

[Create cluster](#)

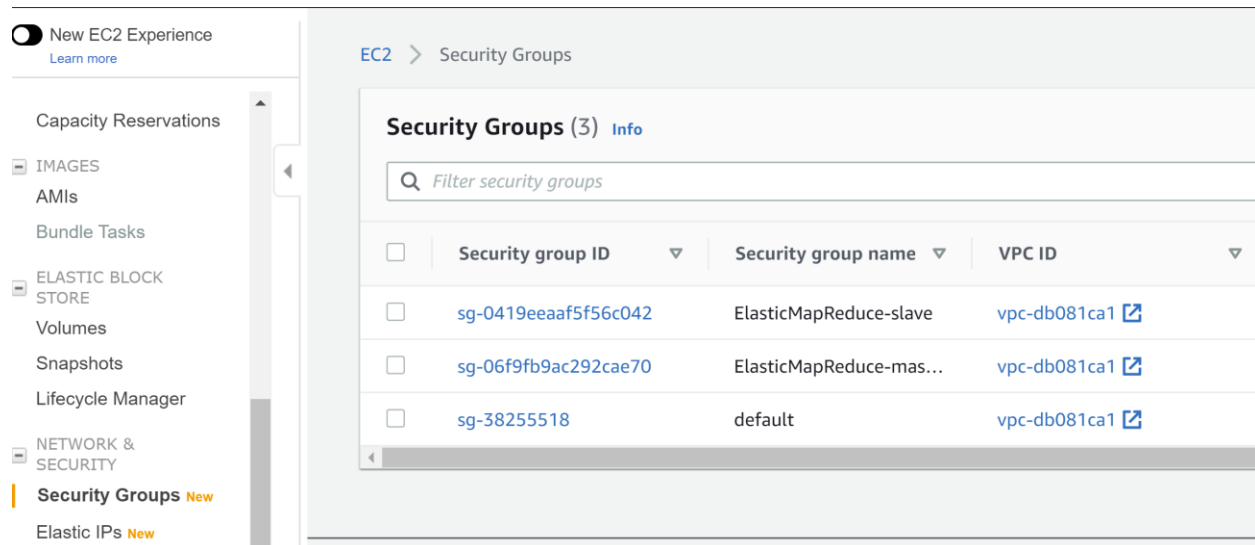
Remaining options can be left default

You can now find this cluster listed under “Clusters”

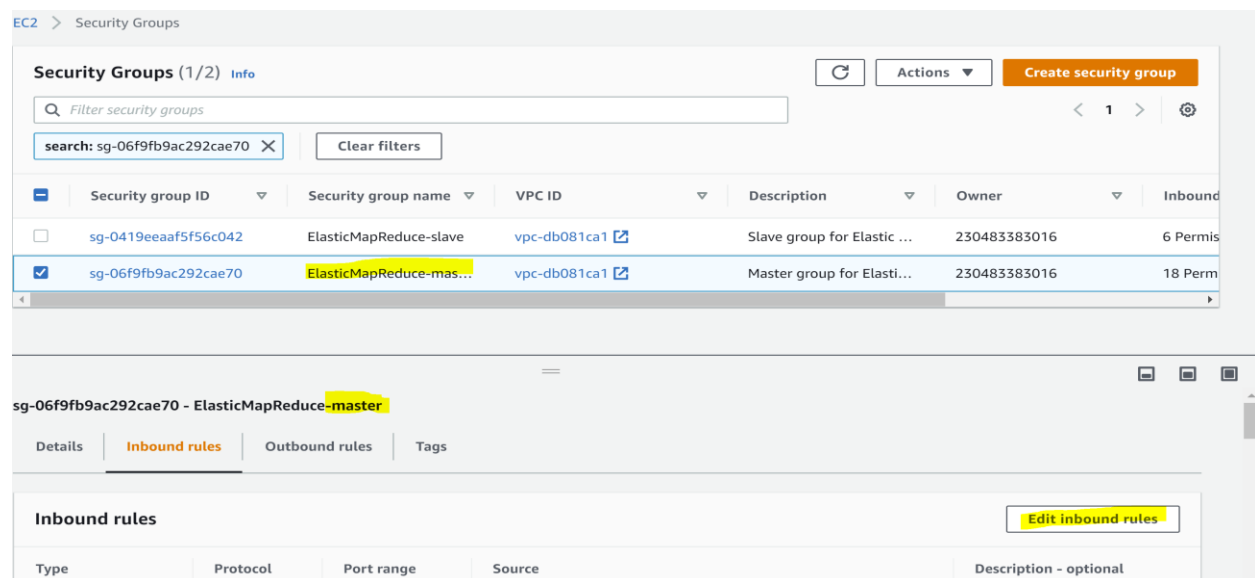
(Note: It takes about 2-3 minutes for the cluster to be up and running)

6. Security settings

Under EC2 service → Network & Security → Security Groups – find the security group configured for master node of above cluster.



Click on master node security group id → “Edit inbound rules” → “Add rules”



Under inbound rules:

Allow SSH port 22 – Anywhere

Allow custom TCP port 8888 – Anywhere (jupyter notebook port)

You should be able to see 3 new instances in EC2, these are 3 nodes of Hadoop cluster.

7. SSH to master node using command

`ssh -i C:/Users/ashan/Downloads/geo.pem hadoop@ec2-100-25-22-128.compute-1.amazonaws.com`

```
hadoop@ip-172-31-63-135:~
C:\Users\ashan>ssh -i C:/Users/ashan/Downloads/geo.pem hadoop@ec2-100-25-22-128.compute-1.amazonaws.com
The authenticity of host 'ec2-100-25-22-128.compute-1.amazonaws.com (100.25.22.128)' can't be established.
ECDSA key fingerprint is SHA256:5Fp1k2QRJoVWMgTxAwXjZ0EGQVsoi8nrS3A09v61Vac.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-100-25-22-128.compute-1.amazonaws.com,100.25.22.128' (ECDSA) to the list of known hosts.

 _ | _ | _ )
 _ | ( _ | /  Amazon Linux AMI
 _ | \ _ | _ |

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
21 package(s) needed for security, out of 40 available
Run "sudo yum update" to apply all updates.

 _ | _ | _ )
 _ | ( _ | /  Amazon Linux AMI
 _ | \ _ | _ |

https://aws.amazon.com/amazon-linux-ami/2018.03-release-notes/
21 package(s) needed for security, out of 40 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRRRRRRRRRRR
E:::EEEEEEEEEEEE::E M:::M M:::M R:::R
EE:::EEEEEEEEEEEE::E M:::M M:::M R:::R
E:::E EEEEE M:::M M:::M RR:::R R:::R
E:::E M:::M M:::M M:::M R:::R R:::R
E:::EEEEEEEEEEEE M:::M M:::M M:::M R:::R R:::R
E:::EEEEEEEEEEEE M:::M M:::M M:::M R:::R R:::R
E:::E EEEEE M:::M M:::M M:::M R:::R R:::R
EE:::EEEEEEEEEEEE::E M:::M M:::M R:::R R:::R
E:::EEEEEEEEEEEE::E M:::M M:::M RR:::R R:::R
EEEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRR RRRRRR

[hadoop@ip-172-31-63-135 ~]$
```

8. run `sudo pip install pyyaml ipython jupyter ipyparallel pandas boto -U`

9. open .bashrc file: `vi .bashrc`

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.

Add the below two lines to the end of the file.

```
export PYSPARK_DRIVER_PYTHON=/usr/local/bin/jupyter
export PYSPARK_DRIVER_PYTHON_OPTS="notebook --no-browser --ip=0.0.0.0 --port=8888"
```

Then run `source ~/.bashrc` followed by `pyspark`

You should see the standard output telling you that Jupyter Notebook is starting.

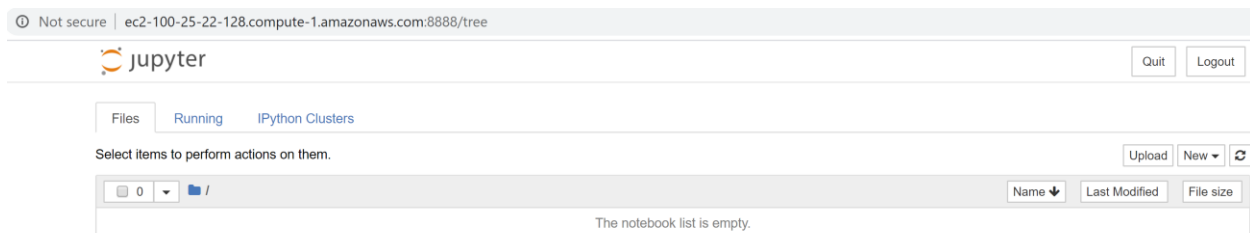
Run the url generated in the terminal on browser to access jupyter notebook

```

Running setup.py install for pandocfilters ... done
Running setup.py install for Send2Trash ... done
Found existing installation: numpy 1.14.5
Uninstalling numpy-1.14.5:
  Successfully uninstalled numpy-1.14.5
Found existing installation: boto 2.48.0
Uninstalling boto-2.48.0:
  Successfully uninstalled boto-2.48.0
Successfully installed MarkupSafe-1.1.1 Send2Trash-1.5.0 attrs-19.3.0 backports-abc-0.5 backports.shutil-get-terminal-si
ze-1.0.0 bleach-3.1.5 boto-2.49.0 configparser-4.0.2 contextlib2-0.6.0.post1 decorator-4.4.2 defusedxml-0.6.0 entrypoint
s-0.3 enum34-1.1.10 functools32-3.2.3.post2 futures-3.3.0 importlib-metadata-1.6.0 ipaddress-1.0.23 ipykernel-4.10.1 ipy
parallel-6.2.5 ipython-5.10.0 ipython-genutils-0.2.0 ipywidgets-7.5.1 jinja2-2.11.2 jsonschema-3.2.0 jupyter-1.0.0 jupyt
er-client-5.3.4 jupyter-console-5.2.0 jupyter-core-4.6.3 mistune-0.8.4 nbconvert-5.6.1 nbformat-4.4.0 notebook-5.7.9 num
py-1.14.5 packaging-20.3 pandas-0.24.2 pandocfilters-1.4.2 pathlib2-2.3.5 pexpect-4.8.0 pickleshare-0.7.5 prometheus-cli
ent-0.7.1 prompt-toolkit-1.0.18 ptyprocess-0.6.0 pygments-2.5.2 pyparsing-2.4.7 pyrsistent-0.16.0 python-dateutil-2.8.1
pytz-2020.1 pyyaml-5.3.1 pyzmq-19.0.1 qtconsole-4.7.3 qtpy-1.9.0 scandir-1.10.0 setuptools-44.1.0 simplegeneric-0.8.1 si
ngledispatch-3.4.0.3 six-1.14.0 terminado-0.8.3 testpath-0.4.4 tornado-5.1.1 traitlets-4.3.3 wcwidth-0.1.9 webencodings-
0.5.1 widgetsnbextension-3.5.1 zipp-1.2.0
You are using pip version 9.0.3, however version 20.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[hadoop@ip-172-31-63-135 ~]$ vi .bashrc
[hadoop@ip-172-31-63-135 ~]$ [hadoop@ip-172-31-63-135 ~]$ source ~/.bashrc
[hadoop@ip-172-31-63-135 ~]$ pyspark
[I 22:09:06.073 NotebookApp] Writing notebook server cookie secret to /home/hadoop/.local/share/jupyter/runtime/notebook
_cookie_secret
[I 22:09:06.242 NotebookApp] Loading IPython parallel extension
[I 22:09:06.247 NotebookApp] Serving notebooks from local directory: /home/hadoop
[I 22:09:06.247 NotebookApp] The Jupyter Notebook is running at:
[I 22:09:06.247 NotebookApp] http://(ip-172-31-63-135 or 127.0.0.1):8888/?token=da6e5fbb52cf91baa2296454a479917a4dc9d08e
3f0d5b61
[I 22:09:06.247 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 22:09:06.251 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/hadoop/.local/share/jupyter/runtime/nbserver-17586-open.html
Or copy and paste one of these URLs:
http://(ip-172-31-63-135 or 127.0.0.1):8888/?token=da6e5fbb52cf91baa2296454a479917a4dc9d08e3f0d5b61

```



Create new python notebook and start coding.

10. open new terminal to, ssh to master node and install below visualization packages on master node

run:

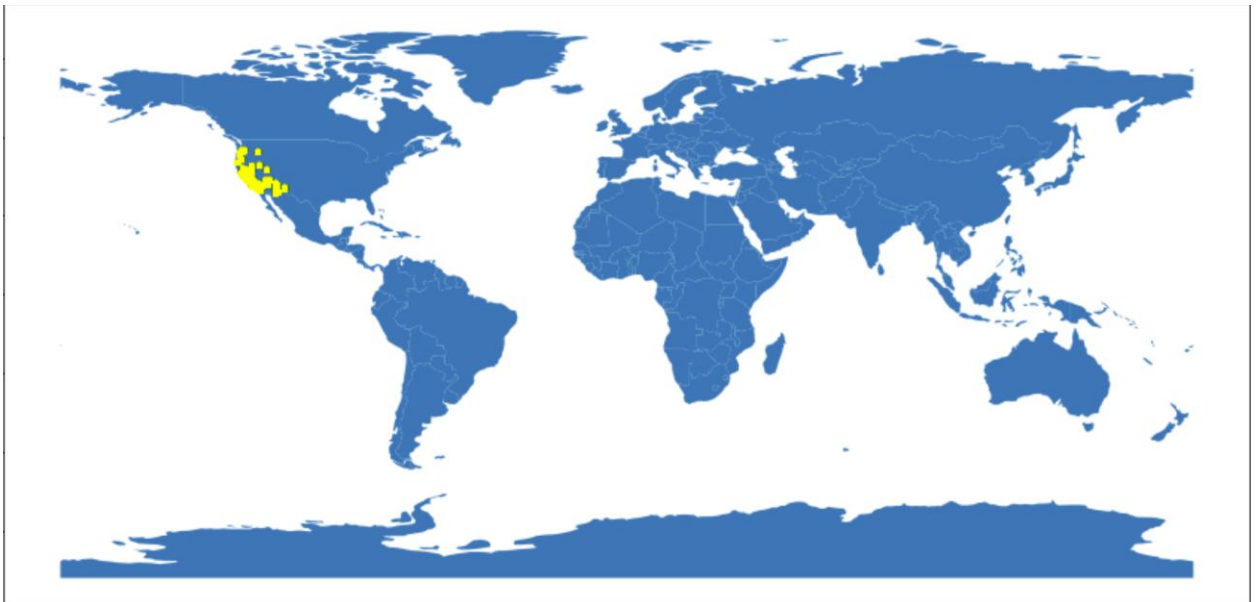
```
sudo pip install geopandas
```

```
sudo pip install matplotlib
```

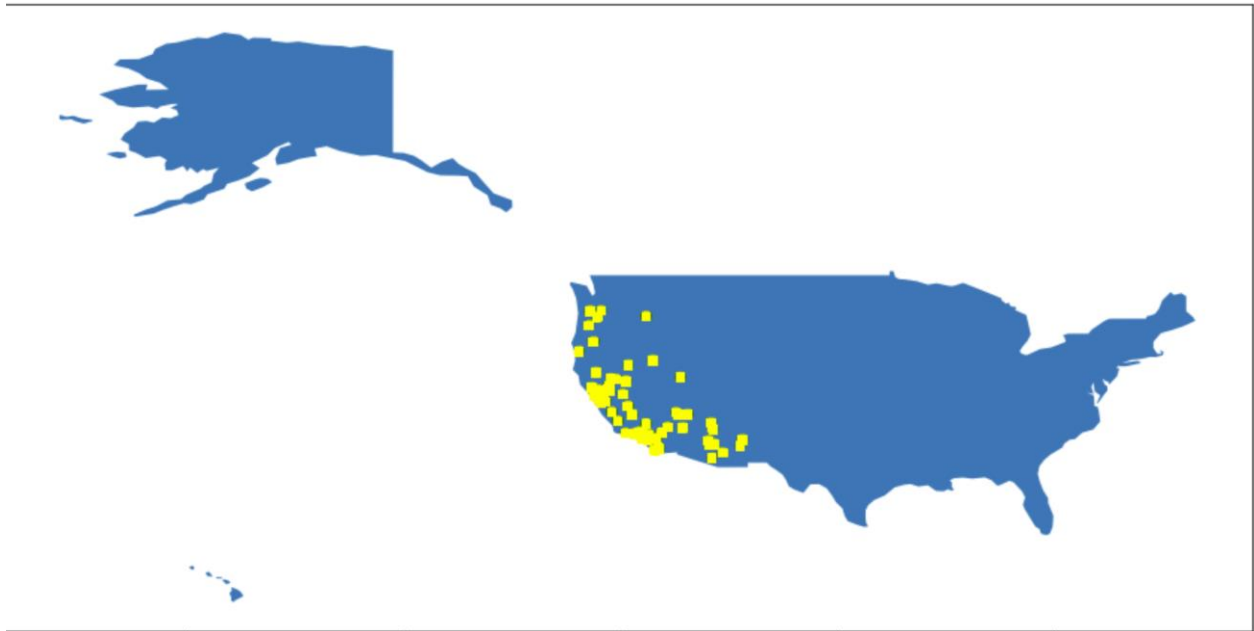
```
sudo pip install Descartes
```


Problem 2: Data Preparation

1. Create a new python notebook "Mobilenet"
2. import required libraries pandas, geopandas, matplotlib, Descartes.
3. Load devicestatus.txt from s3 bucket into a spark data frame 'df' , by giving file path and determining which delimiter to use.
4. Create a new data frame 'deviceDF' by extracting the df.latitude (13th filed), df.longitude(14th field), df.date (first field), df.model (second field) columns
5. Filter out locations that have a latitude and longitude of 0 by applying filter command on deviceDF.
6. Split this field by spaces to separate the manufacturer from the model using *pyspark.sql.functions.split()* and add manufacturer and model columns separately to deviceDF data frame.
7. write the obtained data frame to s3 bucket using *deviceDF.write.csv()* with header.
8. Visualization:
 - a) Convert the deviceDF spark dataframe into pandas dataframe 'df_pd'.
 - b) All the latitude and longitude values in current into float values by iterating through row in df_pd
 - c) Use geopandas to plot the coordinates on world/USA map.
 - d) Convert latitude longitude coordinates into geometry point using *points_from_xy()*
 - e) Create geopandas dataframe 'gdf' using the df_pd and above obtained geometry
 - f) Geopandas has built in dataset 'naturalearth_lowres'. Create a new geopandas 'world' data frame using this dataset.
 - g) Plot gdf and world dataframes



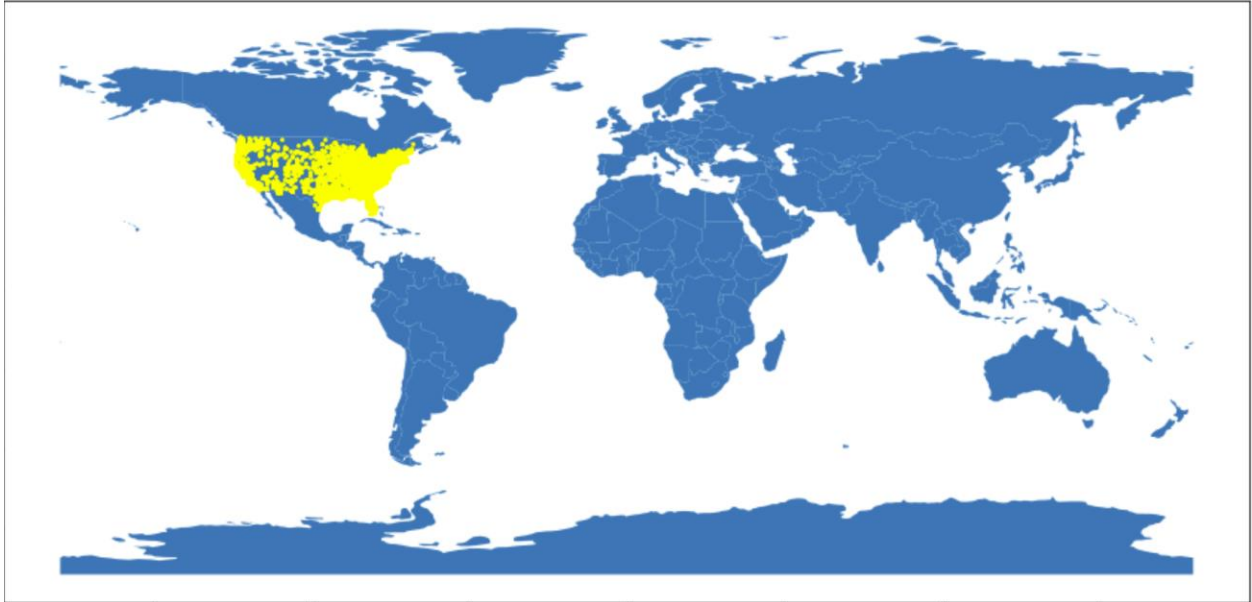
h) Filter world dataset with USA geometry and plot again



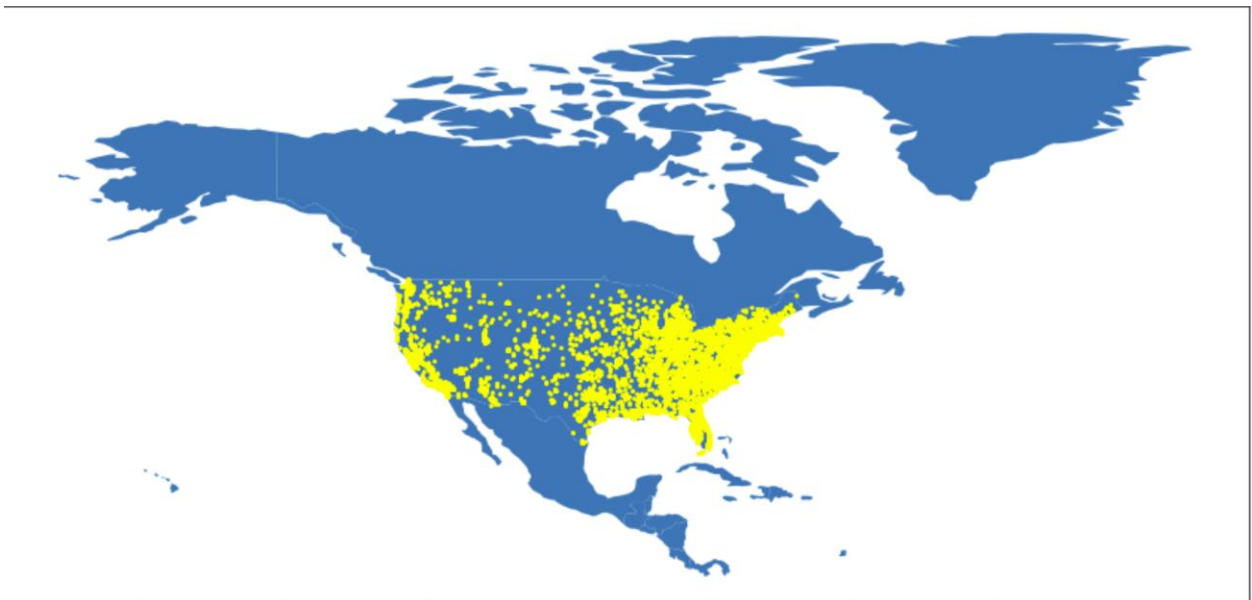
Submit your implementation by adding a Jupyter Notebook file (Mobilenet.ipynb) including your SPARK statements to the clustering/Step1 folder in your Github repository.

Get and Visualize synthetic location data

1. Create a new python notebook "Synthetic"
2. import required libraries pandas, geopandas, matplotlib, Descartes.
3. Load sample_geo.txt from s3 bucket into a spark data frame 'df', by giving file path and determining which delimiter to use.
4. write the obtained data frame to s3 bucket using *df.write.csv()* with header.
5. Visualization:
 - a) Convert the df spark dataframe into pandas dataframe 'df_pd'.
 - b) All the latitude and longitude values in current into float values by iterating through row in df_pd
 - c) Use geopandas to plot the coordinates on world/USA map.
 - d) Convert latitude longitude coordinates into geometry point using *points_from_xy()*
 - e) Create geopandas dataframe 'gdf' using the df_pd and above obtained geometry
 - f) Geopandas has built in dataset 'naturalearth_lowres'. Create a new geopandas 'world' data frame using this dataset.
 - g) Plot gdf and world dataframes



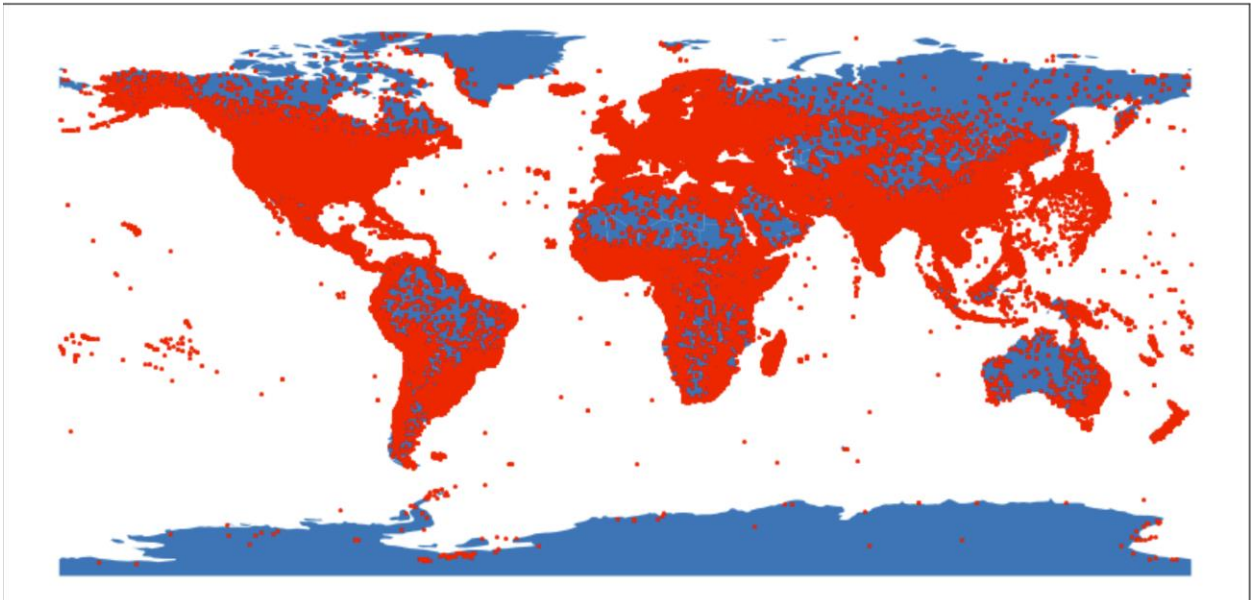
h) Filter world dataset with USA geometry and plot again



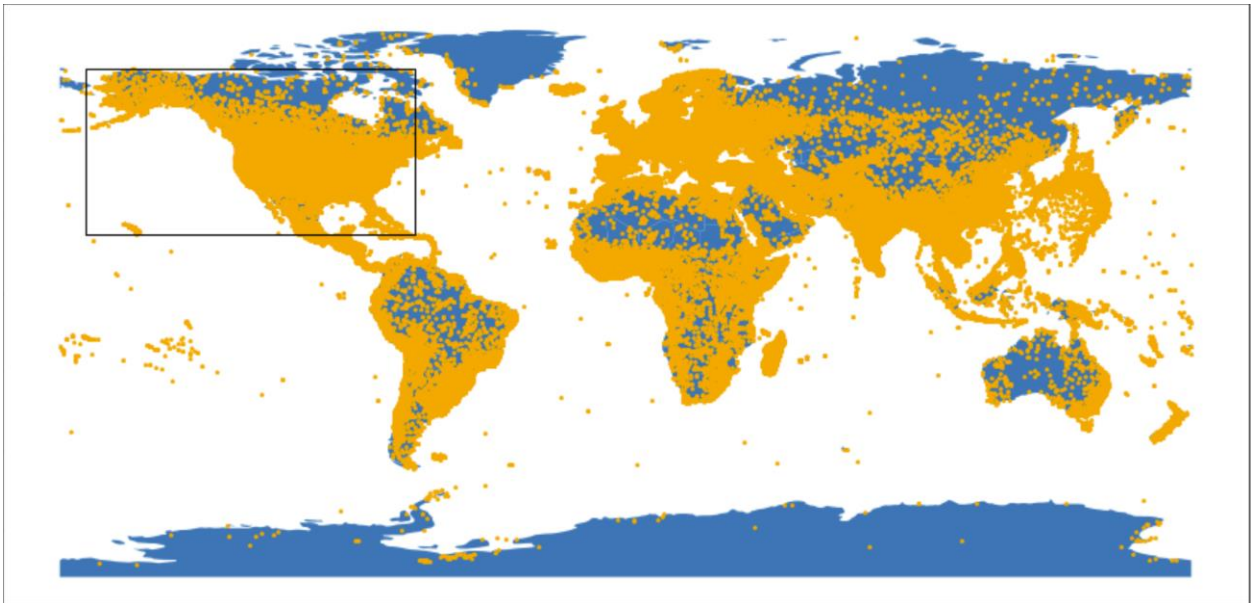
Get and Pre-process the DBpedia location data

1. Create a new python notebook "lat longs.txt"
2. import required libraries pandas, geopandas, matplotlib, Descartes.
3. Load devicestatus.txt from s3 bucket into a spark data frame 'df', by giving file path and determining which delimiter to use.
4. Rename df using *withColumnRenamed()* with column names Latitude, Longitude, name_of_page
5. Filter out locations that have a latitude and longitude with in USA boundary box coordinates: (-124.848974, 24.396308) - (-66.885444, 49.384358)
7. write the obtained data frame to s3 bucket using *USAdf.write.csv()* with header.
8. Visualization:
 - a) Convert the USAdf spark dataframe into pandas dataframe 'USApd'.
 - b) All the latitude and longitude values in current into float values by iterating through row in USApd.
 - c) Use geopandas to plot the coordinates on world/USA map.
 - d) Convert latitude longitude coordinates into geometry point using *points_from_xy()*
 - e) Create geopandas dataframe 'gdf' using the df_pd and above obtained geometry
 - f) Geopandas has built in dataset 'naturalearth_lowres'. Create a new geopandas 'world' data frame using this dataset.

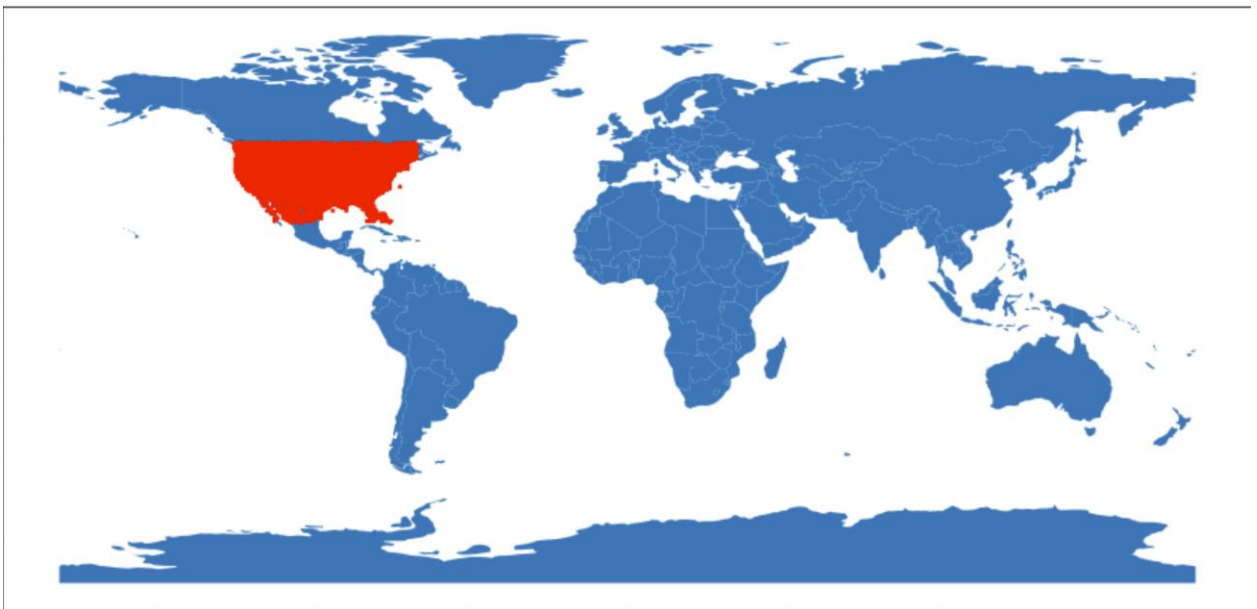
Map before filtering USA

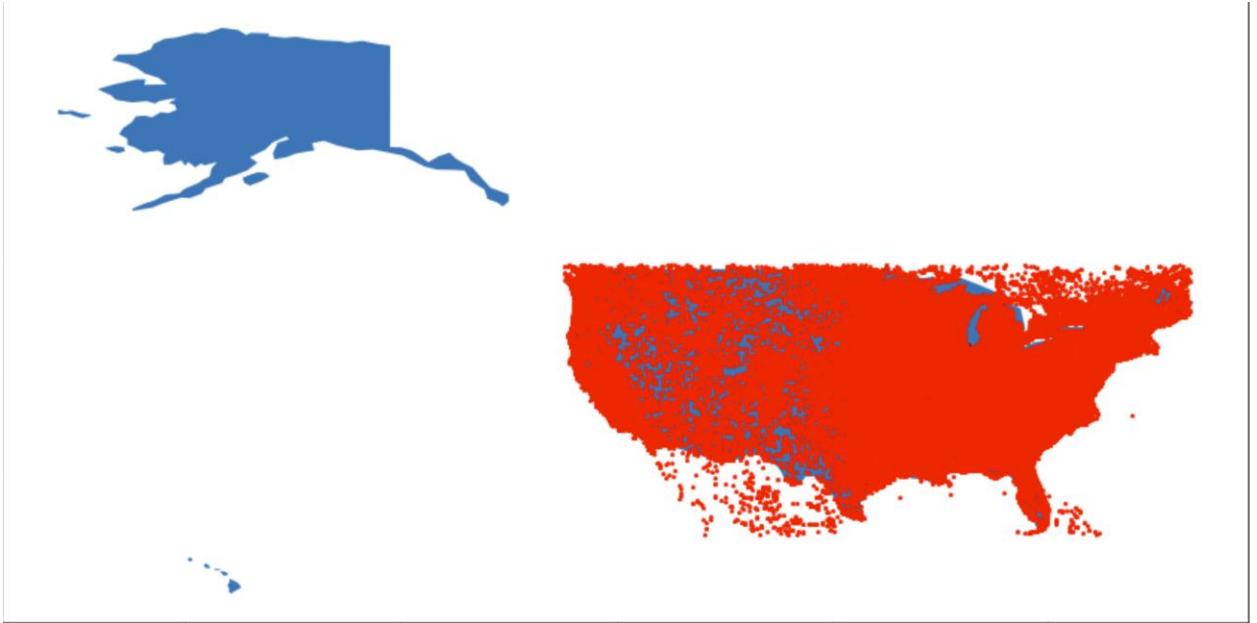


g) Map with USA boundary box



f) Plot gdf and filtered dataframe.





Implementing k-means

1. create Jupiter notebook “KMean”
2. import libraries – pandas, math, datetime, geopandas, matplotlib, Descartes.
3. create following functions required for kmean algorithm
 - **closestPoint:** given a (latitude/longitude) point and an array of current center points, returns the index in the array of the center closest to the given point
 - convert latitude and longitude values from dataset into float
 - iterate through the centroids list
 - by calling euclideanDistance/ greatCircleDistance for calculate distance between point and a centroid within centroid loop. Append all distances from a point and 3 centers to a list
 - find minimum values from list and return its index by incrementing it with 1(as python indexing starts with 0)
 - **convergence:** given 2 equal length arrays of points and returns the convergence distance
 - iterate through the range k
 - by calling euclideanDistance/ greatCircleDistance for calculate distance between same index points from old and new centroid lists
 - append the distances to a list
 - find sum of all values in distance list. Return sum
 - **euclideanDistance:** given two points, returns the Euclidean distance of the two.
 - Calculate distance using formula $\text{sqrt}((x_2-x_1)**2 + (y_2-y_1)**2)$

- **greatCircleDistance:** given two points, returns the great circle distance of the two
 - Calculate distance using formula

$$\text{temp} = \sin(\text{lat_diff}/2) * \sin(\text{lat_diff}/2) + \sin(\text{lon_diff}/2) * \sin(\text{lon_diff}/2) * \cos(X1) * \cos(X2)$$

$$\text{distance} = 6371 * 2 * \text{atan2}(\text{sqrt}(\text{temp}), \text{sqrt}(1-\text{temp}))$$

4. Kmean algorithm

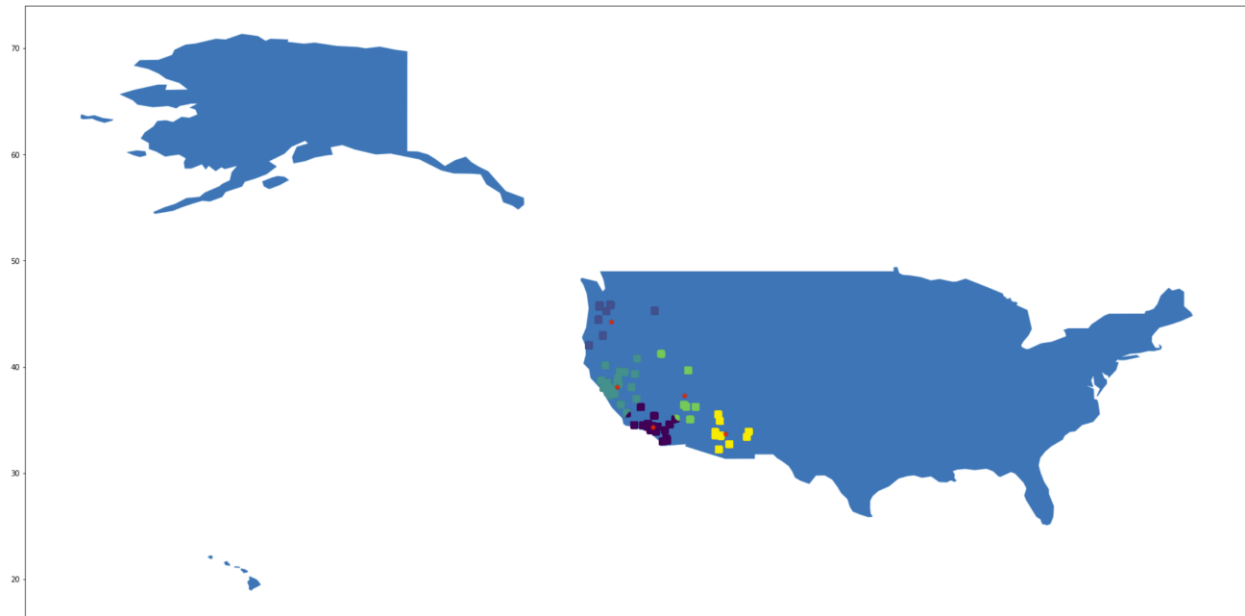
- Take user distance method (Euclidean/ Great Circle), number of clusters(k), data file path
- Assign a value to convergeDist say 0.1
- Load data into spark data frame 'df' using *spark.read.csv()* with header
- convert spark data frame to RDD 'df1' with just latitude, longitude columns
- persist df1 using *.cache()*
- randomly selecting K data points for the centroids without replacement using *.takeSample(False, k)*
- convert the centroids values into float.
- Initiate while loop, to exit the loop new convergence distance has to be less than assumed convergence distance.
 - Iterate through the df1 rdd, for each row call the closestPoint function to find label of nearest cluster
 - Create a list with latitude, longitude points and above obtained label
 - Create a dictionary from above list with each cluster label as key and list of all points with that label as value.
 - Using this dictionary calculate the new centroid points by finding mean of all latitudes and longitudes in each cluster.
 - Now, calculate new convergence distance by calling convergence function stated above giving new centroid points and old once.
 - Update the old centroid with new centroid list.
- This loop ends when new convergence distance < convergence distance.
- Now, create a final list of latitude, longitude points and label obtained by calling closestPoint function with final centroids as input.

Compute and Visualize Clusters

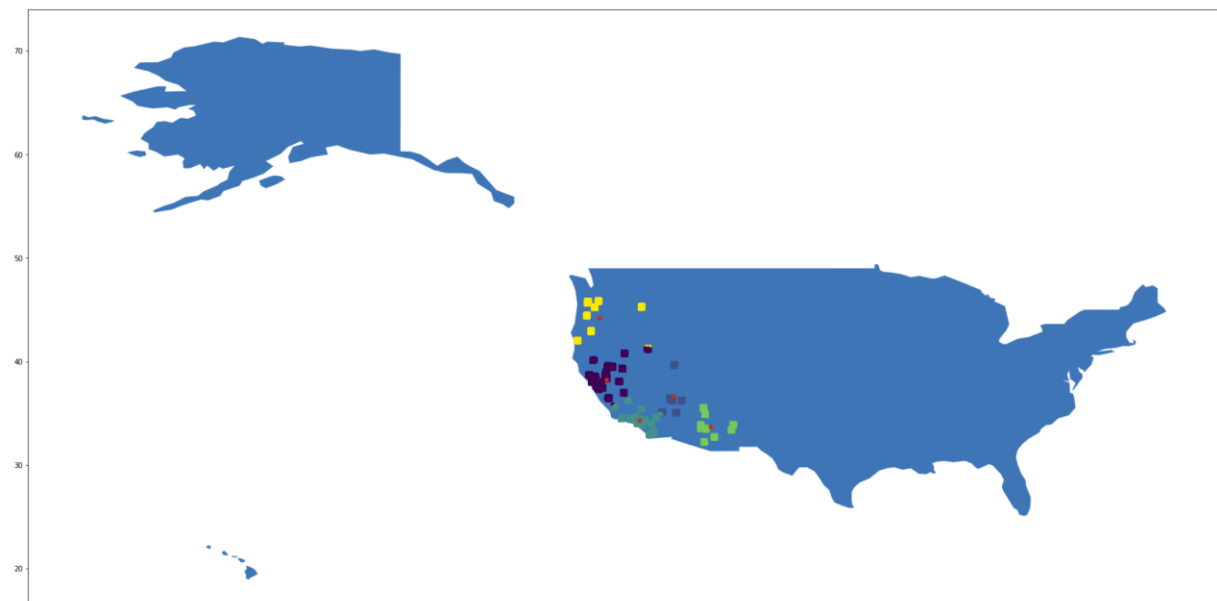
Run the kmean algorithm for below inputs and visualize the map.

1. device dataset

```
Enter distance measure(Euclidean/great circle): "Euclidean"  
Enter number of clusters: 5  
Enter input filepath: "s3a://geo-project-data/datafiles/mobilenet/"
```

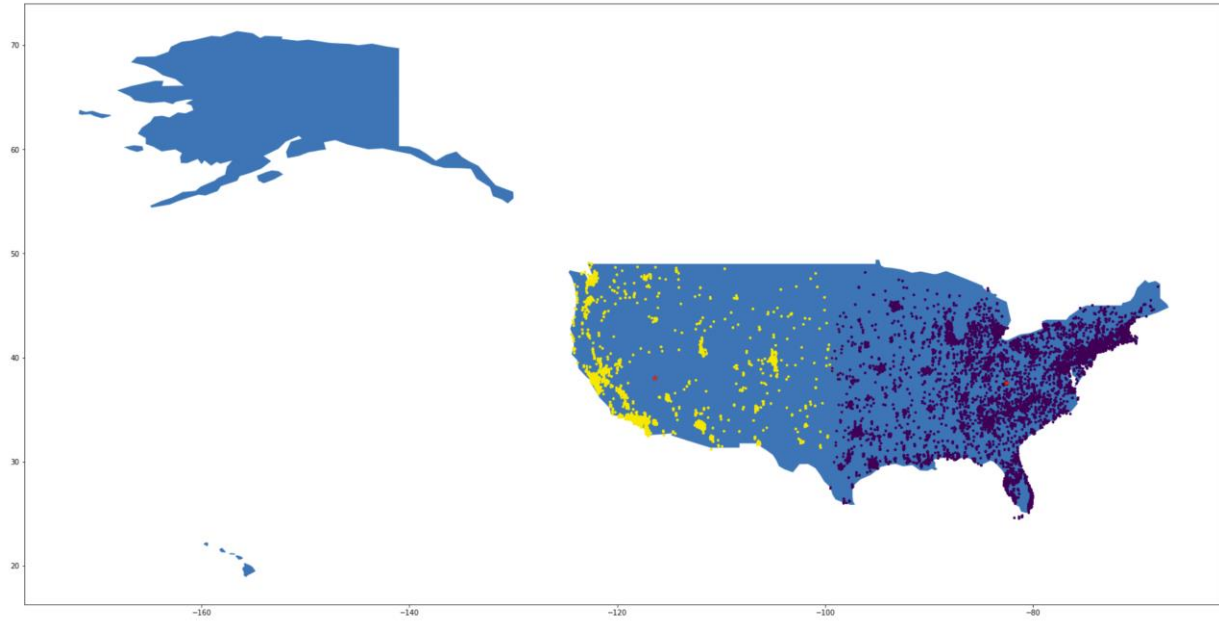


```
Enter distance measure(Euclidean/great circle): "great circle"  
Enter number of clusters: 5  
Enter input filepath: "s3a://geo-project-data/datafiles/mobilenet/"
```

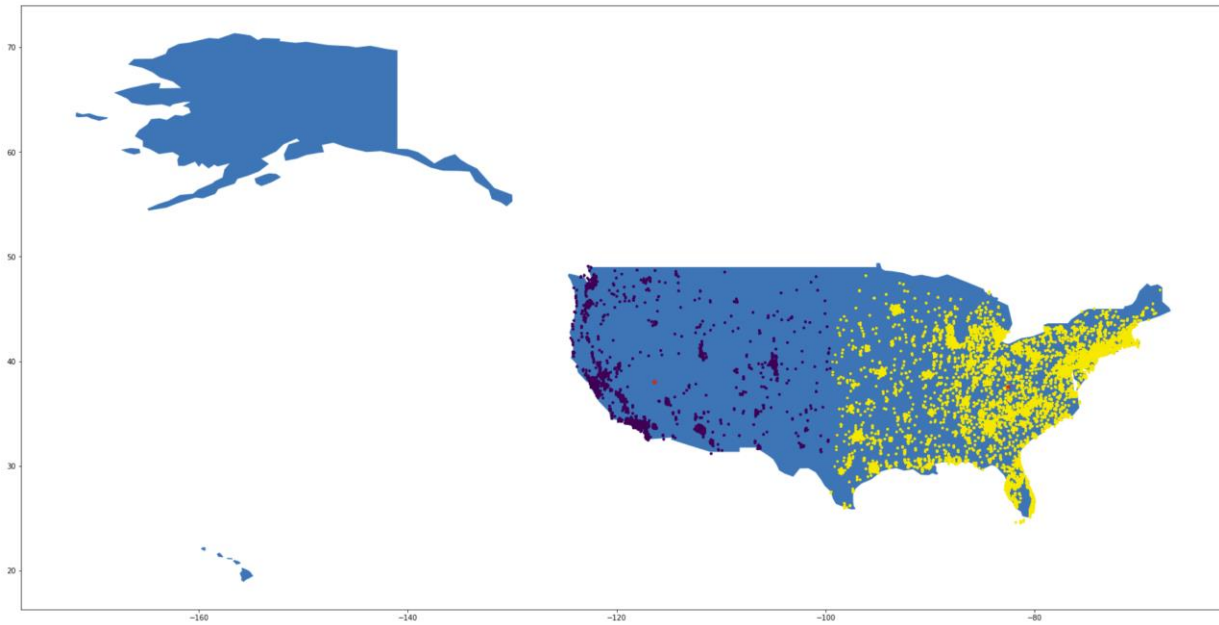


2. Synthetic dataset

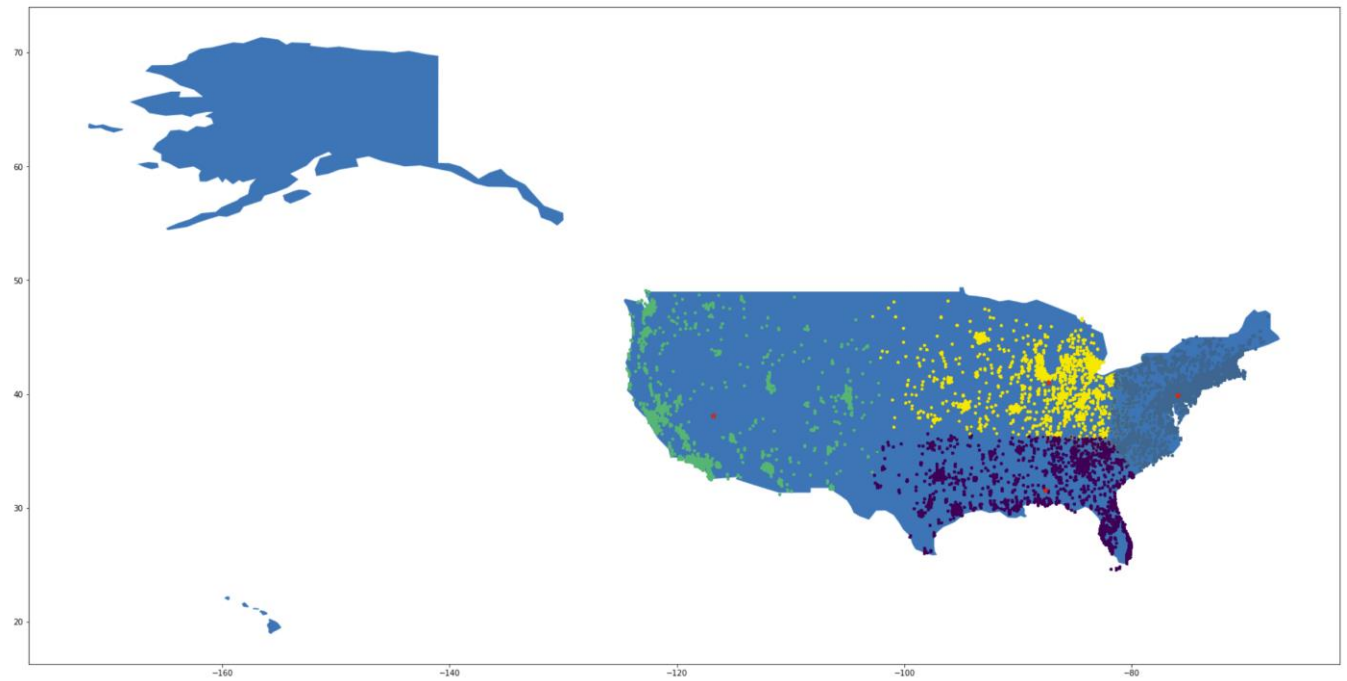
Enter distance measure(Euclidean/great circle): "Euclidean"
Enter number of clusters: 2
Enter input filepath: "s3a://geo-project-data/datafiles/synthetic/"



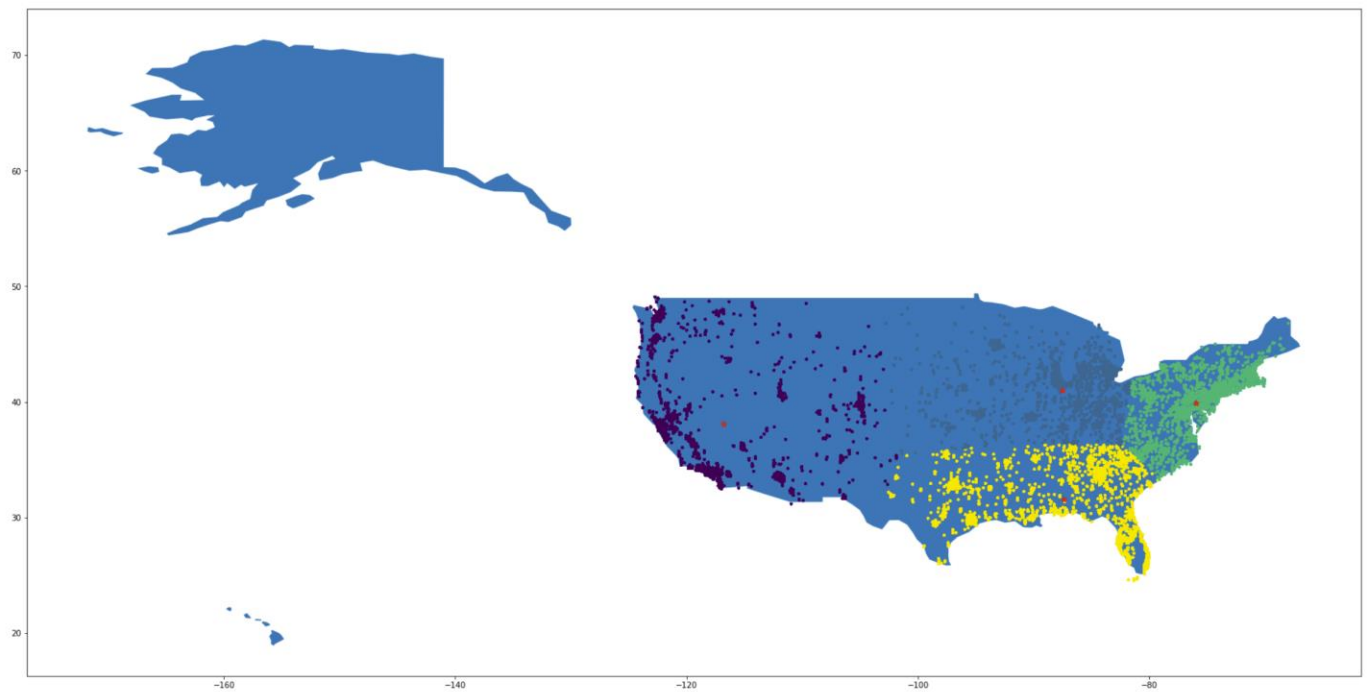
Enter distance measure(Euclidean/great circle): "Great circle"
Enter number of clusters: 2
Enter input filepath: "s3a://geo-project-data/datafiles/synthetic/"



Enter distance measure(Euclidean/great circle): "Euclidean"
Enter number of clusters: 4
Enter input filepath: "s3a://geo-project-data/datafiles/synthetic/"

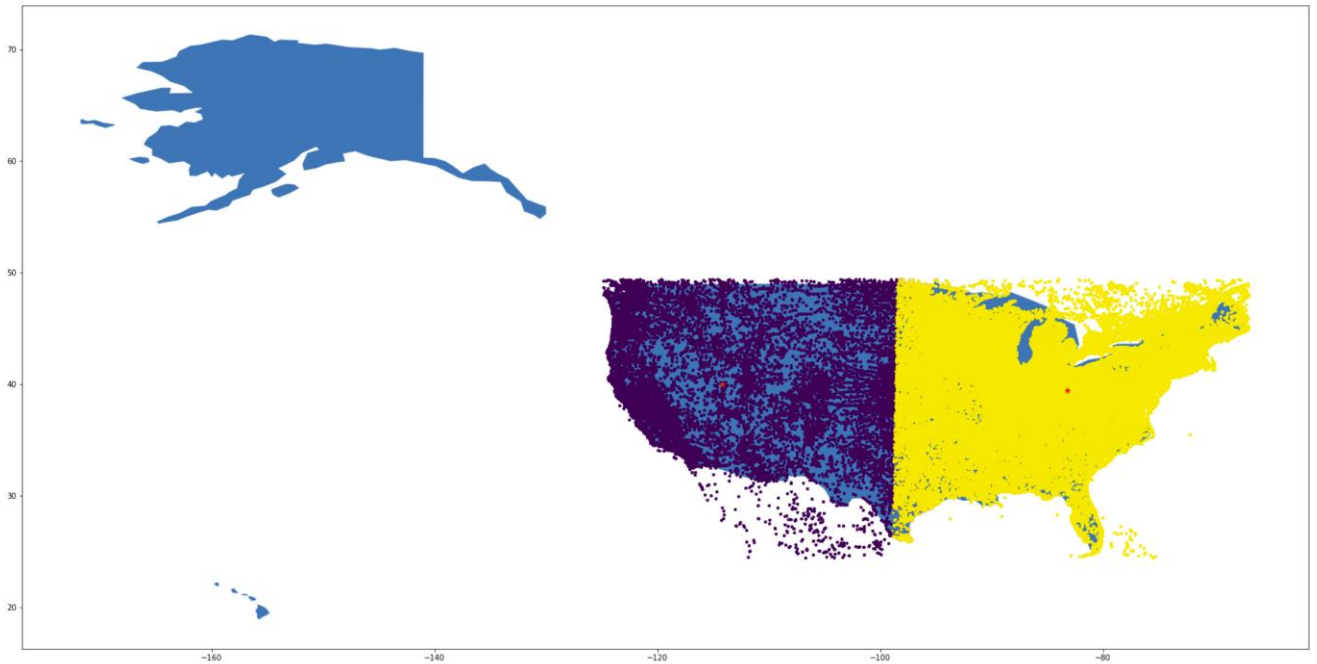


Enter distance measure(Euclidean/great circle): "great circle"
Enter number of clusters: 4
Enter input filepath: "s3a://geo-project-data/datafiles/synthetic/"

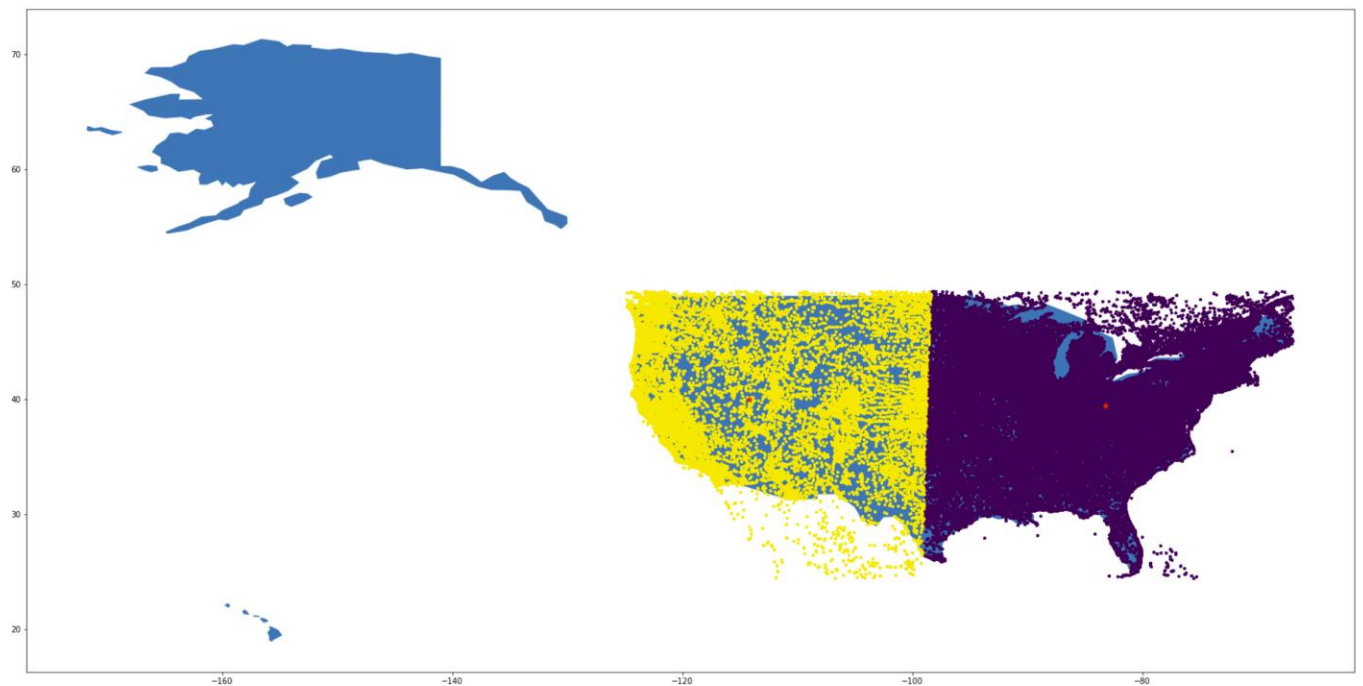


3. DBpedia USA data

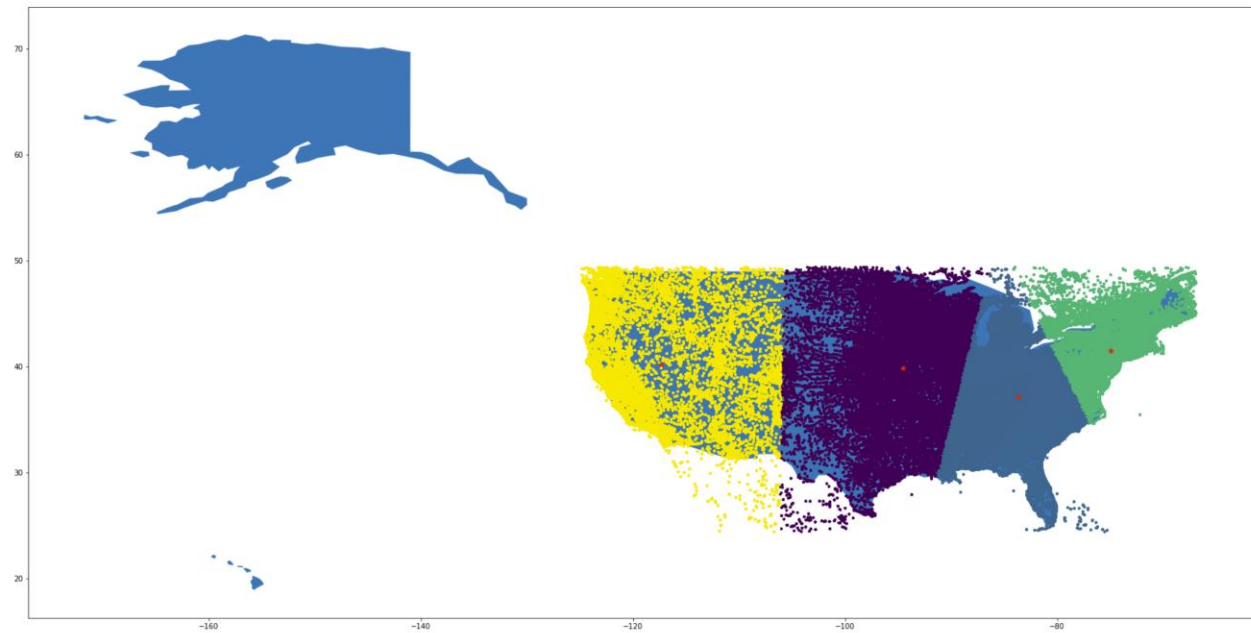
```
Enter distance measure(Euclidean/great circle): "Euclidean"  
Enter number of clusters: 2  
Enter input filepath: "s3a://geo-project-data/datafiles/dbpedia_USA/"
```



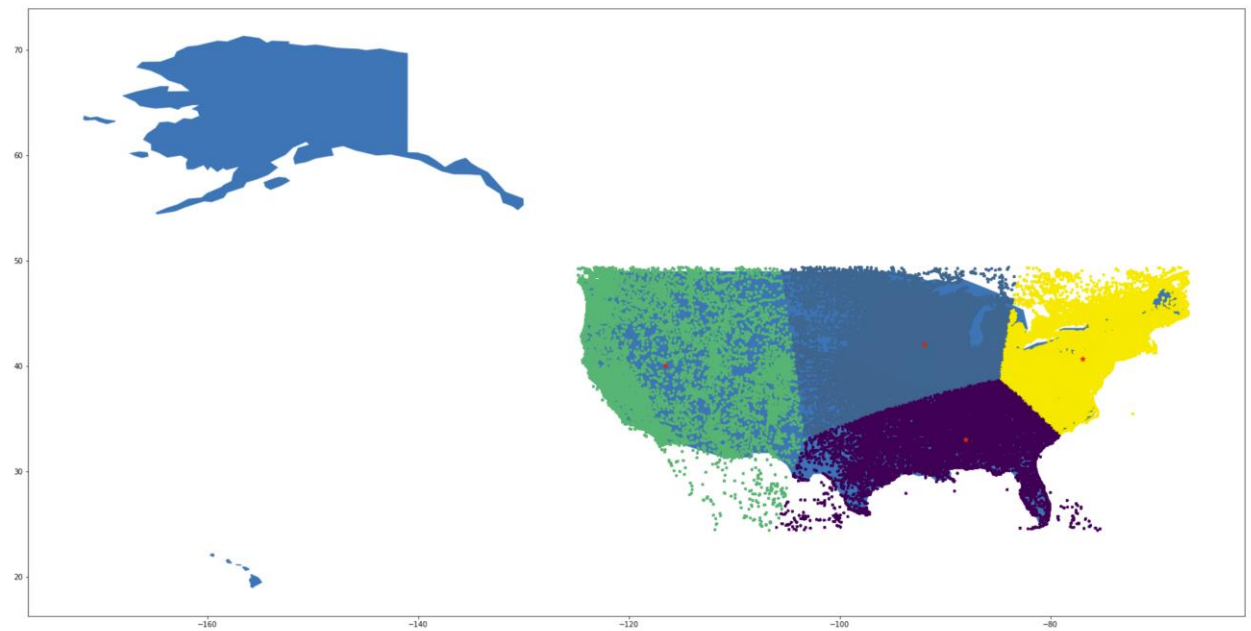
```
Enter distance measure(Euclidean/great circle): "great circle"  
Enter number of clusters: 2  
Enter input filepath: "s3a://geo-project-data/datafiles/dbpedia_USA/"
```



```
Enter distance measure(Euclidean/great circle): "Euclidean"  
Enter number of clusters: 4  
Enter input filepath: "s3a://geo-project-data/datafiles/dbpedia_USA/"
```



```
Enter distance measure(Euclidean/great circle): "Great circle"  
Enter number of clusters: 4  
Enter input filepath: "s3a://geo-project-data/datafiles/dbpedia_USA/"
```



observation:

1. device dataset has less number of values so the cluster partitions are not too specific
2. For synthetic and dbpedia datasets the cluster regions are more clearly distinguished.
3. With the increase in K value the deviation of points from center is reduced.
4. Euclidean and great circle displayed different cluster partition patterns

Runtime Analysis

1. Import datetime module
2. For all the above inputs combinations determine the execution time using *datetime.start()* and *datetime.end()*
3. Store the times in a list and create a pandas data frame for visualization.
4. Run the analysis for couple of time and take average of times.
5. Analyze the avg time for better results.

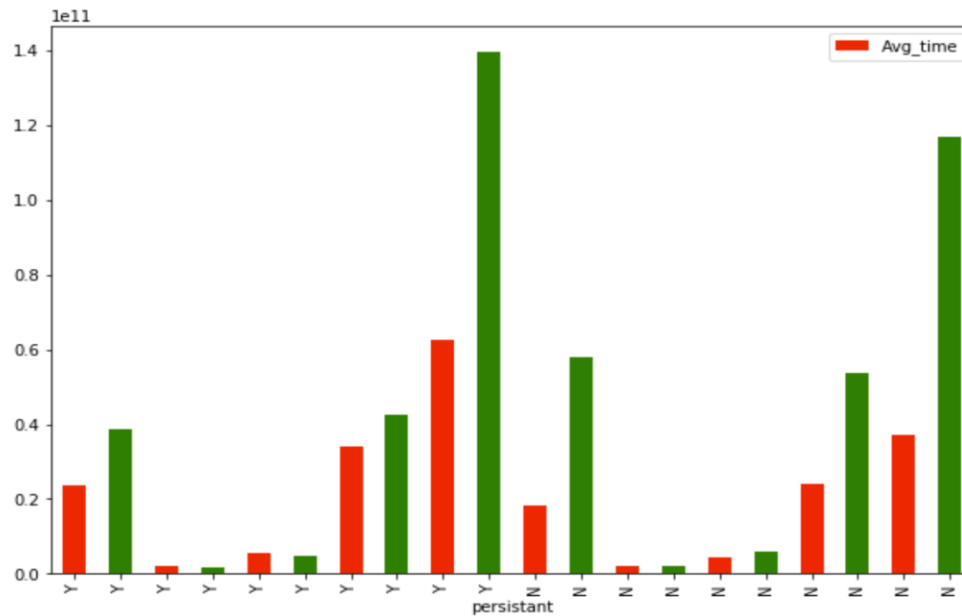
	data	k	persistant	method	time1	time2	Avg_time
0	Device	5	Y	euclidean	00:00:31.416589	00:00:16.134803	00:00:23.775696
1	Device	5	Y	great circle	00:00:58.477266	00:00:19.099695	00:00:38.788480
2	Synthetic	2	Y	euclidean	00:00:01.876999	00:00:02.260561	00:00:02.068780
3	Synthetic	2	Y	great circle	00:00:02.255648	00:00:01.255958	00:00:01.755803
4	Synthetic	4	Y	euclidean	00:00:06.317864	00:00:04.671790	00:00:05.494827
5	Synthetic	4	Y	great circle	00:00:04.514388	00:00:05.080228	00:00:04.797308
6	DBPedia	2	Y	euclidean	00:00:35.029559	00:00:33.150899	00:00:34.090229
7	DBPedia	2	Y	great circle	00:00:49.685142	00:00:35.232399	00:00:42.458770
8	DBPedia	4	Y	euclidean	00:01:21.595695	00:00:43.196841	00:01:02.396268
9	DBPedia	4	Y	great circle	00:02:38.287585	00:02:01.051618	00:02:19.669601
10	Device	5	N	euclidean	00:00:16.645306	00:00:20.116122	00:00:18.380714
11	Device	5	N	great circle	00:00:47.380484	00:01:08.323477	00:00:57.851980
12	Synthetic	2	N	euclidean	00:00:02.010163	00:00:02.148173	00:00:02.079168
13	Synthetic	2	N	great circle	00:00:01.476381	00:00:02.476700	00:00:01.976540
14	Synthetic	4	N	euclidean	00:00:05.864653	00:00:03.173622	00:00:04.519137
15	Synthetic	4	N	great circle	00:00:06.866152	00:00:05.114449	00:00:05.990300
16	DBPedia	2	N	euclidean	00:00:30.268860	00:00:18.029080	00:00:24.148970
17	DBPedia	2	N	great circle	00:00:52.298998	00:00:54.982206	00:00:53.640602
18	DBPedia	4	N	euclidean	00:00:32.417253	00:00:41.505202	00:00:36.961227
19	DBPedia	4	N	great circle	00:02:45.102679	00:01:08.874380	00:01:56.988529

Observations:

1. implementing kmean algorithm with Great circle method takes more time compared to Euclidean distance method

```
timedf.plot(x='persistent',y='Avg_time',kind="bar",color=['red', 'green'], figsize=(10,7))
```

8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7731685790>

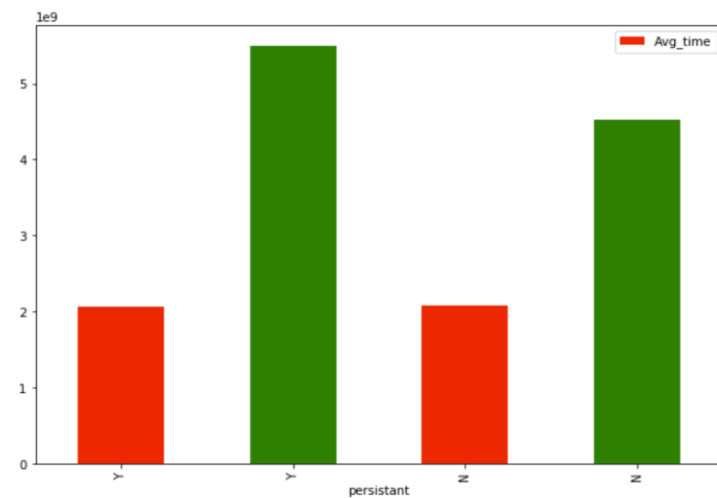


red bars represent euclidean method and green bars represent great circle.

2. Computation time increases with the number of clusters.

```
timedf[(timedf.method == "euclidean") & (timedf.data == "Synthetic")].plot(x='persistent',y='Avg_time',kind="bar",color=['red', 'green'],
```

45]: <matplotlib.axes._subplots.AxesSubplot at 0x7f773146ead0>



red bars represent k=2 and green ones k=4.

3. Time takes for 3 datasets in order: DBPedia > Synthetic > Mobilenet
4. for majority of input condition the run time is reduced due to parsing/cache

Finally, read off the runtimes and other statistics from the Spark History Server with command *hdfs dfs -rm -R /spark2-history/** in master node terminal.