Geo-Location Clustering Using K Means Algorithm

DISTRIBUTED AND SCALABLE DATA ENGR

FINAL PROJECT REPORT-MANASA RANGINENI

Submitted to
Dr. Vahid Behzadan
ASSISTANT PROFESSOR
UNIVERSITY OF NEW HAVEN
FALL 2021

Introduction

Clustering is a process of grouping a set of data points into clusters so that points that are put within the same cluster are similar to each other whereas points from different clusters are dissimilar.
Clustering has many useful applications for marketing, logistics, and document classification.
We will 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. This parameter needs to be set according to the application or problem domain. (There is no magic formula to set k.) In a nutshell, k-means groups the data by minimizing the sum of squared distances between the data points and their respective closest centroid.

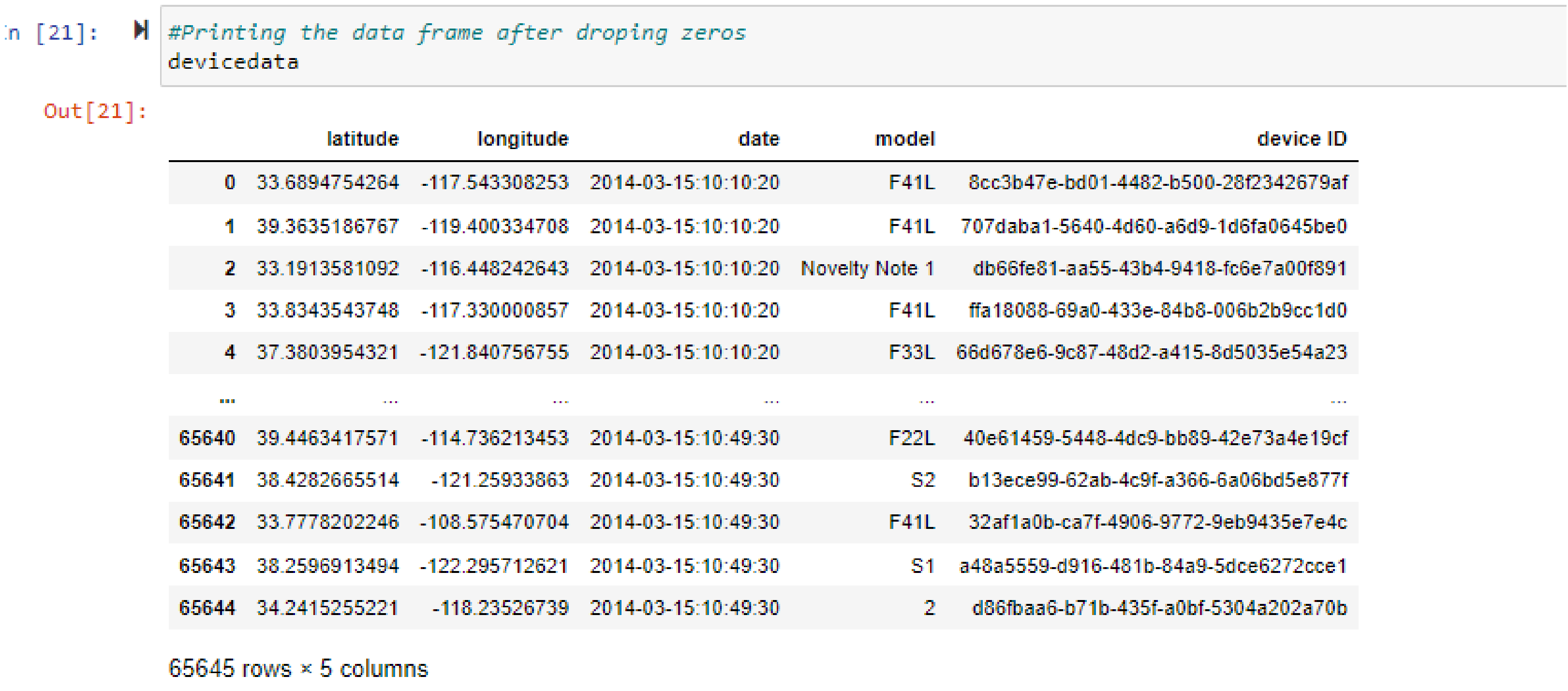
Data Preparation

Before implementing the actual algorithm, I went through pre-processing step in order to convert the data into a standardized format for later processing. The following describes the pre-processing process for device status data:

1. Load the dataset
2. Determine which delimiter to use
3. Filter out any records which do not parse correctly; each record should have exactly14 values
4. Extract the date, model, device ID, and latitude and longitude
   a. date: 1st field
   b. model: 2nd field
   c. device ID: 3rd field
   d. latitude: 13th field
   e. longitude: 14th field
5. Store latitude and longitude as the first two fields
6. Filter out locations that have a latitude and longitude of 0
7. Split the model field that contains the device manufacturer and model name byspaces
8. Save the extracted data as comma separated values file in S3 bucket.
9. Confirm the data in the file(s) was saved correctly

Data:

Mobilenet:



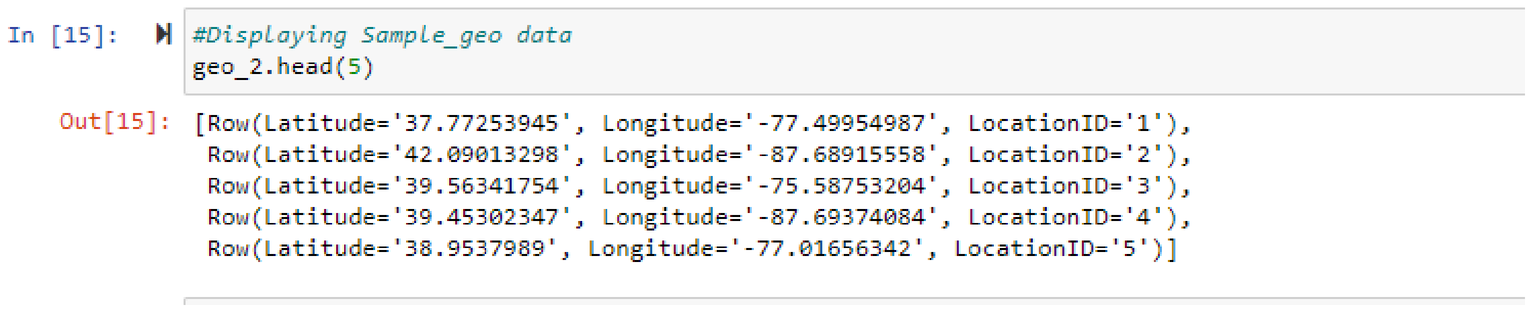
```
n [21]:  #Printing the data frame after droping zeros
         devicedata
```

Out[21]:

	latitude	longitude	date	model	device ID
0	33.6894754264	-117.543308253	2014-03-15:10:10:20	F41L	8cc3b47e-bd01-4482-b500-28f2342679af
1	39.3635186767	-119.400334708	2014-03-15:10:10:20	F41L	707daba1-5640-4d60-a6d9-1d6fa0645be0
2	33.1913581092	-116.448242643	2014-03-15:10:10:20	Novelty Note 1	db66fe81-aa55-43b4-9418-fc6e7a00f891
3	33.8343543748	-117.330000857	2014-03-15:10:10:20	F41L	ffa18088-69a0-433e-84b8-006b2b9cc1d0
4	37.3803954321	-121.840756755	2014-03-15:10:10:20	F33L	66d678e6-9c87-48d2-a415-8d5035e54a23
...	...	...	...	...	...
65640	39.4463417571	-114.736213453	2014-03-15:10:49:30	F22L	40e61459-5448-4dc9-bb89-42e73a4e19cf
65641	38.4282665514	-121.25933863	2014-03-15:10:49:30	S2	b13ece99-62ab-4c9f-a366-6a06bd5e877f
65642	33.7778202246	-108.575470704	2014-03-15:10:49:30	F41L	32af1a0b-ca7f-4906-9772-9eb9435e7e4c
65643	38.2596913494	-122.295712621	2014-03-15:10:49:30	S1	a48a5559-d916-481b-84a9-5dce6272cce1
65644	34.2415255221	-118.23526739	2014-03-15:10:49:30	2	d86fbaa6-b71b-435f-a0bf-5304a202a70b

65645 rows × 5 columns

Synthetic:



```
In [15]:  #Displaying Sample_geo data
          geo_2.head(5)

Out[15]: [Row(Latitude='37.77253945', Longitude='-77.49954987', LocationID='1'),
          Row(Latitude='42.09013298', Longitude='-87.68915558', LocationID='2'),
          Row(Latitude='39.56341754', Longitude='-75.58753204', LocationID='3'),
          Row(Latitude='39.45302347', Longitude='-87.69374084', LocationID='4'),
          Row(Latitude='38.9537989', Longitude='-77.01656342', LocationID='5')]
```

Dbpedia:

```
In [19]:  #Showing pandas data frame
          pandas_df.head()
```

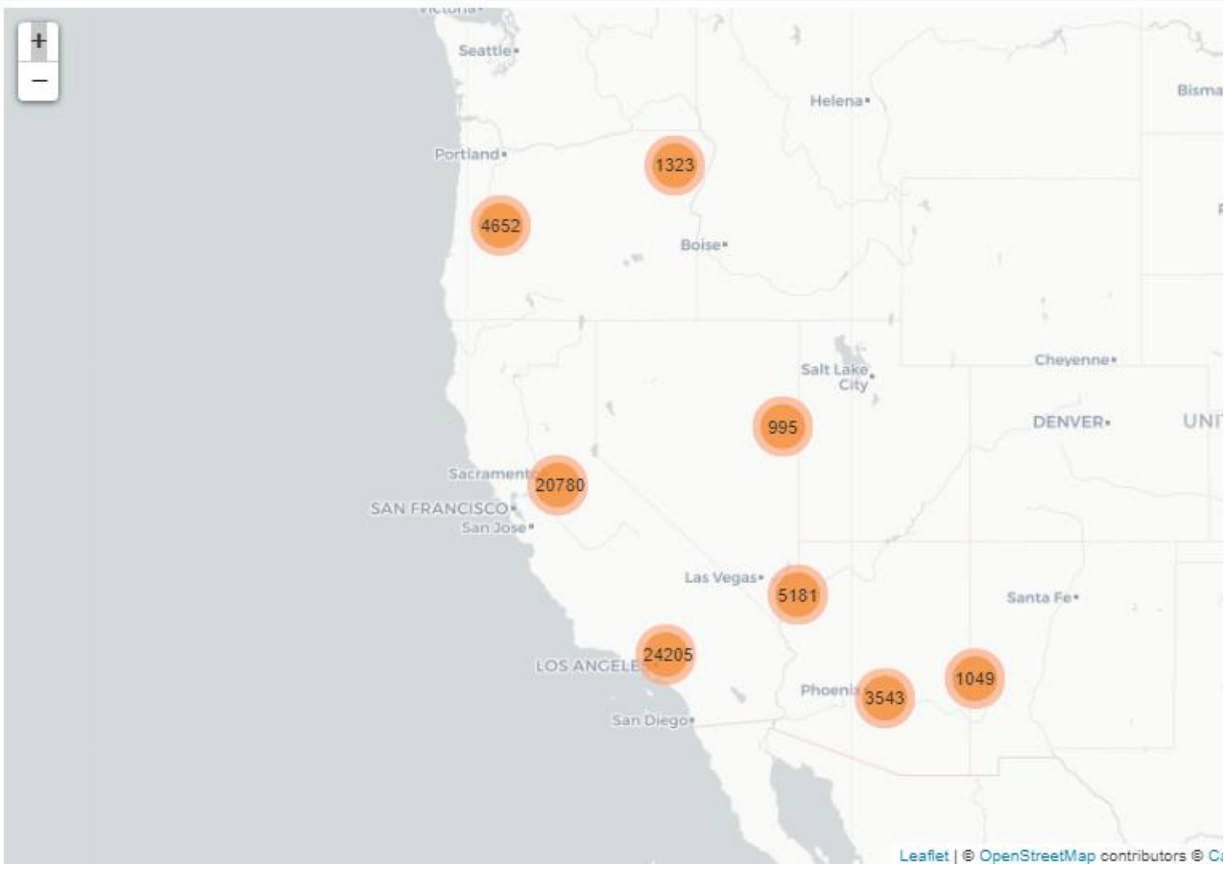
Out[19]:

	lat	long	name_of_page
0	36.7	3.216666666666667	<http://dbpedia.org/resource/Algeria>
1	42.5	1.5166666666666666	<http://dbpedia.org/resource/Andorra>
2	12.516666666666667	-70.03333333333333	<http://dbpedia.org/resource/Aruba>
3	-8.833333333333334	13.333333333333334	<http://dbpedia.org/resource/Angola>
4	41.333333333333336	19.8	<http://dbpedia.org/resource/Albania>

VISUALIZATION:

Mobilenet data:

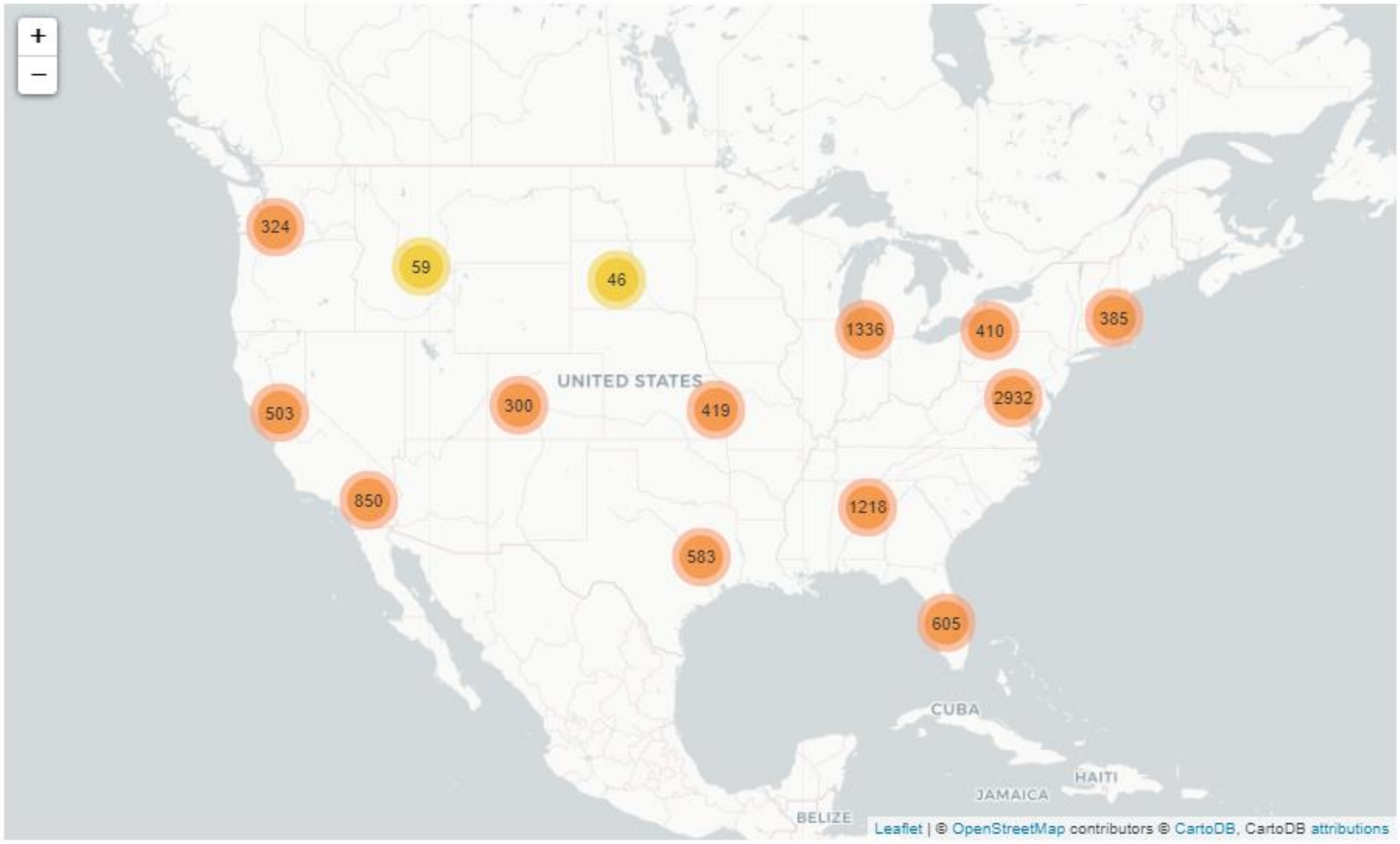
:[29]:



Synthetic Location Data

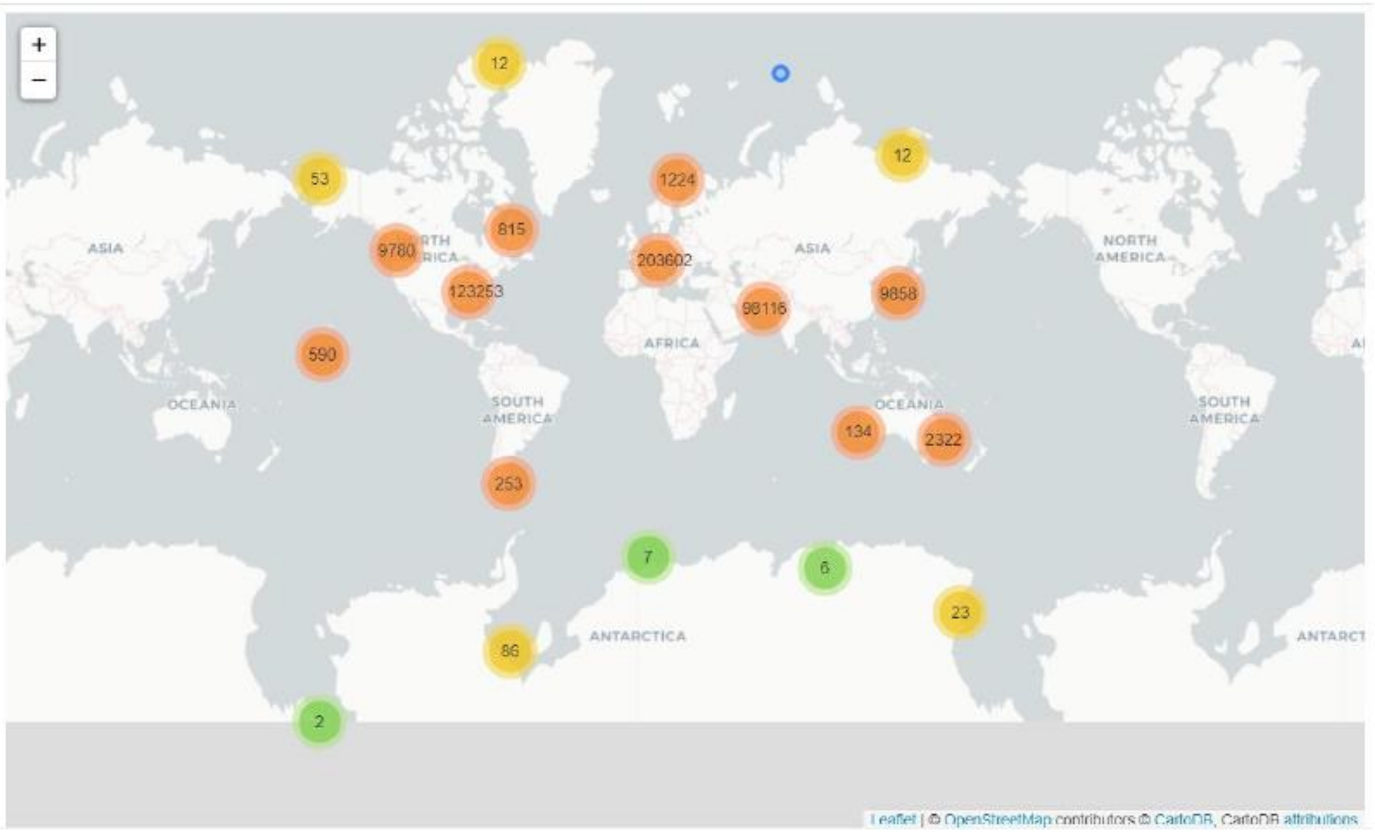
```
n [21]:  #Printing the plotted latitude and logitude information on map
         world_map
```

Out[21]:



DBPedia Location Data

```
Out[13]:
```



Clustering

The initial centroids are a *k*-sized random sample of all points in the dataset. On each iteration, the algorithm assigns each point to its nearest centroid, then calculates the new centroids by taking the mean of all points in that centroid's cluster. The distance between points and centroids is calculated using either Euclidean distance or Great Circle distance - this parameter is set by the user. The main difference between the two measures is that the former measures a straight-line distance between the points in 3D space, while the latter measures the distance across the spherical surface of the Earth.

Though a "perfect" algorithm would iterate until the change in centroid locations converges to 0, this algorithm continues iterating until the sum of all changes in centroid locations
converges to α=0.1 km. That is, the algorithm calculates the distance between each centroid's new location and former location (using the distance measure specified by the user) on each iteration, and continues iterating until the sum of these distances for all centroids is less than 0.1 km. This requires larger values of *k* to converge more precisely than smaller values. Because data this algorithm runs on covers at least an entire continent, we determined that the alpha of 0.1 km or 100 m was a small enough value for this purpose. We found that this method gave us good results for the clusters without an extremely prohibitively long runtime.

Implementation

The implementation consists of kmeans.py file. kmeans.py is the Python script that should be submitted to Spark. The Spark job can be submitted as follows:

```
spark-submit --master local kmeans.py \
<input_path> <output_path> <distance_measure> <k>
```

where input_path is the path to the input data, output_path is the path where output data should be saved, distance_measure is either "Euclidean" or "GreatCircle" (case-insensitive), and k is an integer larger than 0.

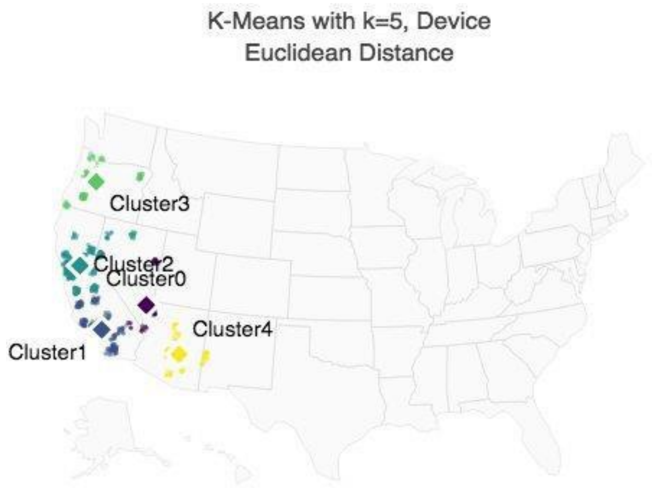
The implementation includes functions to calculate the distance between two points using the two distance measures: Euclidean and Great Circle. Calculating the Euclidean distance between points was done by first converting the latitude-longitude points to Cartesian coordinates, then calculating the distance, and finally converting the result back to a latitude-longitude point. The Great Circle distance formula works directly on latitude and longitude and does not involve conversion.

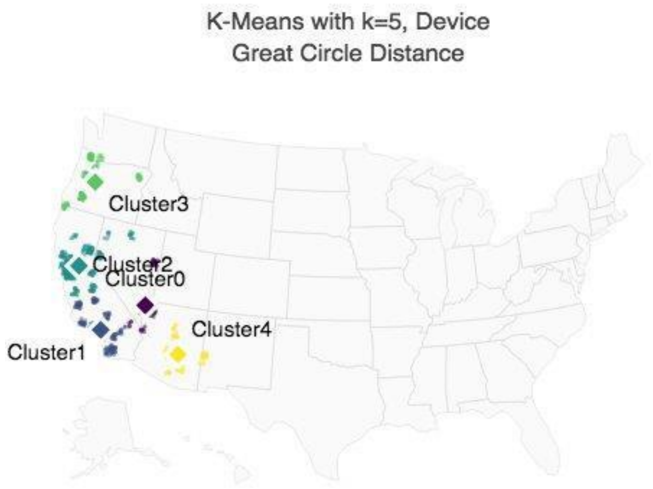
In addition to calculating the centroids and which points belong to which cluster/centroid, the implementation also calculates the mean distance between each point and its nearest centroid. This value is used as a metric for the quality of the model in order to determine appropriate values of *k* for the DBpedia and real-world datasets.

Results

Device Location Data

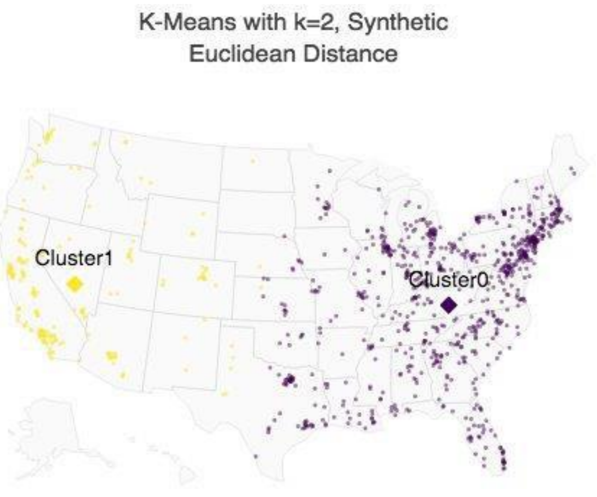
When generating a sample of the data points to determine the initial centroids, I used a fixed seed. This meant that the initial centroids were always the same for the same dataset and the same value of *k*. This allowed me to more directly compare the Euclidean and Great Circle distance measures without the confounding variable of having different initial centroids. Through this method, we found that the final centroids were consistently the same across the two distance measures. The difference between the two distance measures was not dramatic enough to result in points being assigned to different final clusters, thus the final centroids were the same whether I was using Euclidean or Great Circle.

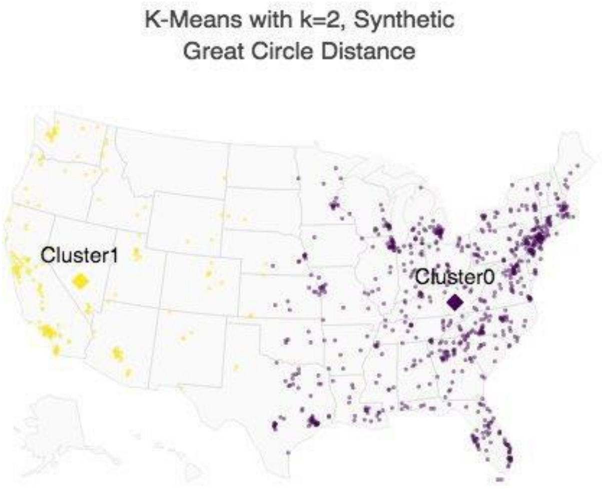


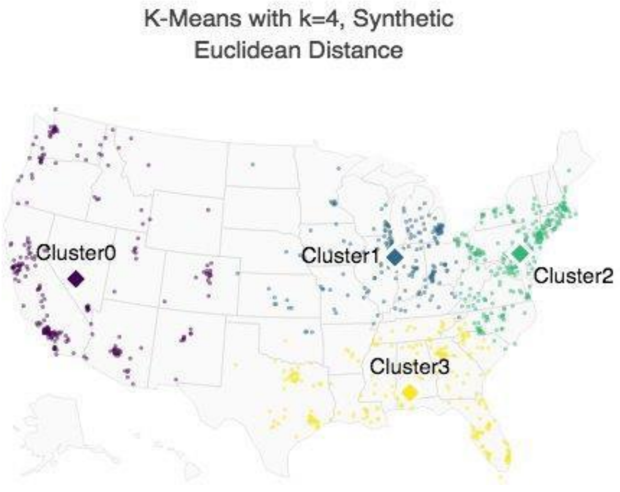


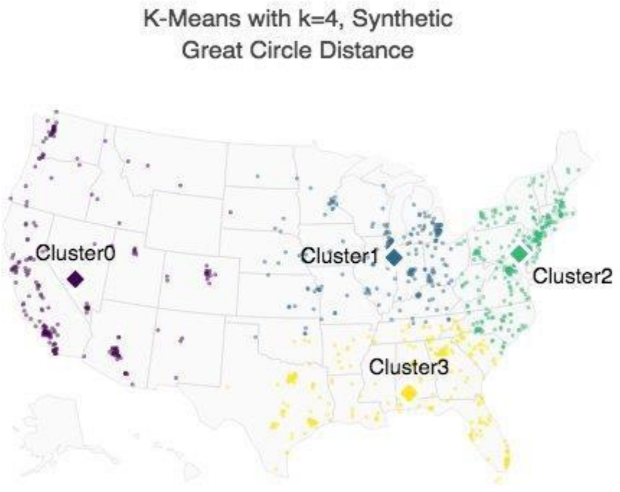
Synthetic Location Data

With k=2, the synthetic location data was grouped roughly into east and west:



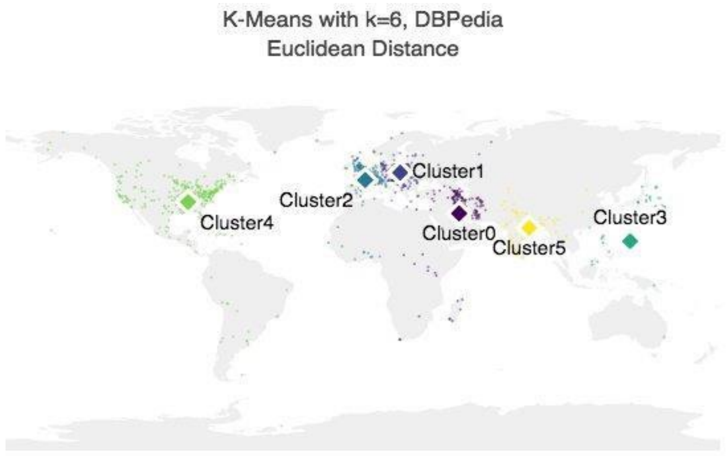


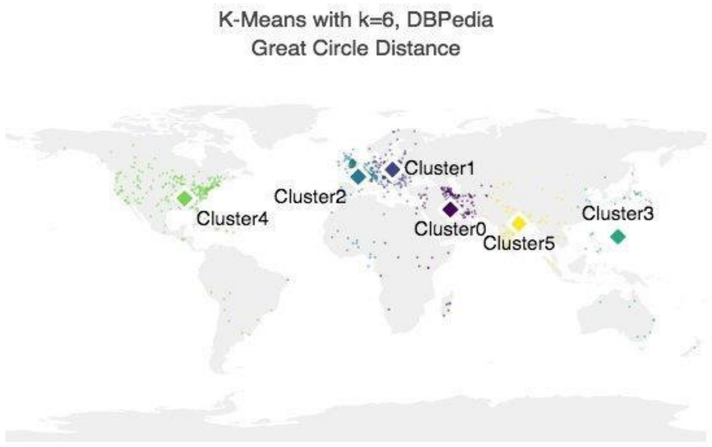




Dbpedia Data

As seen in the graphs, the sharpest improvement in quality is from k=5 to k=6. After k=6, there is not much improvement with increasing values of k. In addition, visual inspection of the plotted clusters shows that k=6 is a sensible value.
This is true for both Euclidean and Great Circle measures, which had very similar results. Using k=6 also makes sense because it corresponds to the number of continents minus Antarctica, which has significantly fewer DBpedia locations than the other continents.





Runtime Analysis

Dataset	K	Persist/Cache	Measure	Runtime (Sec)
DBpedia	6	Y	Euclidean	973.58963
DBpedia	6	N	Euclidean	1292.72722
synthetic	2	Y	Euclidean	13.55297
synthetic	2	N	Euclidean	14.45616
synthetic	4	Y	Euclidean	24.6351
synthetic	4	N	Euclidean	25.5668
device	5	Y	Euclidean	1198.696
device	5	N	Euclidean	1321.82274

CONCLUSION

When the *k* was found for each application, it seemed plausible for the corresponding application. *k* was a metric that not only divided the data into clusters but also gave new insight for the data set.

Visualization was also a significant portion of the project. Though the computer is able toprocess large number of data, in the end, it was necessary for humans to interpret the meaning of the data and final clusters yielded by the algorithm.

This project experiments with different methods to alter run time: RDD persistence. As *k* increased, the runtime of the algorithm increased. However, RDDpersistence somewhat decreased the runtime of the algorithm on the same arguments, andcloud execution drastically improved the runtime