CSE427S Final Project Write-Up

Project 3

Daniel Kwon, Jice Kunfeng Zheng, Rui Qiu

Introduction

Our final project aims to process the Uber pickup location data with the k-means algorithm to determine the ideal idle location for Uber drivers. We aim to generate clusters that yield recommendation of locations that the idle vehicles should be at. We've filtered our data according to different time brackets of the day as well as the difference between weekdays and the weekend, to enhance the accuracy of the recommendation. We expect the recommendations to be effective not only to human drivers but more so for the upcoming age of self-driving vehicles, assisting the location of idle vehicles for more cost-efficient operations of the self-driving FHV (For-Hire Vehicle) service.

Motivation

In the last decade, companies like Uber and Lyft have redefined the transportation industry. In many cities, people choose to call an Uber ride to go from doorstep to doorstep instead of having to engage in the convoluted public transport system, and many drivers commit their off-work time for some extra income. Such demand and supply coming together through smart-device technology has quickly replaced the long-lived taxi industry and has given birth to one of the largest and rapidly growing tech industries in the world.

In the recent few years, these large FHV (For-Hire Vehicle) companies seeking to move

on to the next generation with self-driving vehicles. Giants such as Uber and Lyft debuted their first self-driving cars in 2018, and while having some technological and ethical issues still ensuing, there is little doubt that the direction of technological advancement will change. The era of self-driving cars is quickly approaching, and the following are some new problems to solve.

Our final project is inspired by one of the problems in the upcoming generation of self-driving Uber rides. In the present day, when customers request an uber ride, the server-side algorithm detects available drivers and matches them to the customer. While variables such as locational proximity, driver's and customer's rating, and the traffic in the area are considered in the matching algorithm, being near-by the request often prioritizes the driver to receive the request. Naturally, human drivers are motivated to learn the repeating pattern of customer requests and locate themselves towards where a customer is more likely to be present. For example, during the semester that Wash U offered two free Uber rides every day, numerous drivers gathered around the campus and the Delmar Loop area to accommodate this vastly increased demand for rides, and when the policy was terminated, the number of drivers around the area decreased again.

We thought that such adaptation to repeated patterns and adjusting the cars' idle location was a critical feature that needs to be recreated in the self-driving vehicles. The behavior of human drivers to intuitively locate themselves back to a crowded area where they are more likely to be matched with a customer saves time and resource for both the service providers and customers compared to other scenarios where the cars far away need to be matched with a customer or where the idle vehicles are driving around randomly before they are matched with the next customer. We believed that by using the k-means algorithm with the right value of k, we can predict the location where it is mostly likely to be closest to the location of the upcoming request.

Approach – Selection and Preprocessing of Data

Our first step was to select an appropriate data comprehensive enough to provide relevant conclusions for each timeframe of the day. We discovered a dataset on Kaggle that recorded all Uber pickups in New York City from April 2014 to September 2014. The data comprises of 4.5 million data points with four columns: date/time, latitude, longitude, and the TLC base company code. We decided that we want to run the K-means model on eight 3-hour time brackets, since such division would more accurately capture the changing behaviors of people according to the different time of the day. We also felt the need to distinguish the weekdays from the weekend, since most people have different transportation patterns in their weekend compared to their weekday for reasons like they aren't going to work, they may go out in the evening, and etc. Our next step was cleaning up the original data based on our implementation choices. We preprocessed our data to contain which of the 8 time bracket that they belong to instead of the accurate timestamp. We also used Excel's innate function to identity which day of the week each data. After these values to filter the data by are generated through the preprocessing of the data, we move on to the k-means model.

The goal of K-means clustering is to partition n data points into k clusters such that the mean of the distances between each data point to its nearest centroid is minimized. We apply the K-means algorithm in the following steps:

1. Arbitrarily find k data points and set as the initial centroids.
2. Assign each data point to the nearest centroid. We used the geolocation distance formula, where latitudes and longitudes of two geolocation are the parameters, to calculate the distance.
3. Recalculate the location of each centroid by averaging the longitudes and latitudes

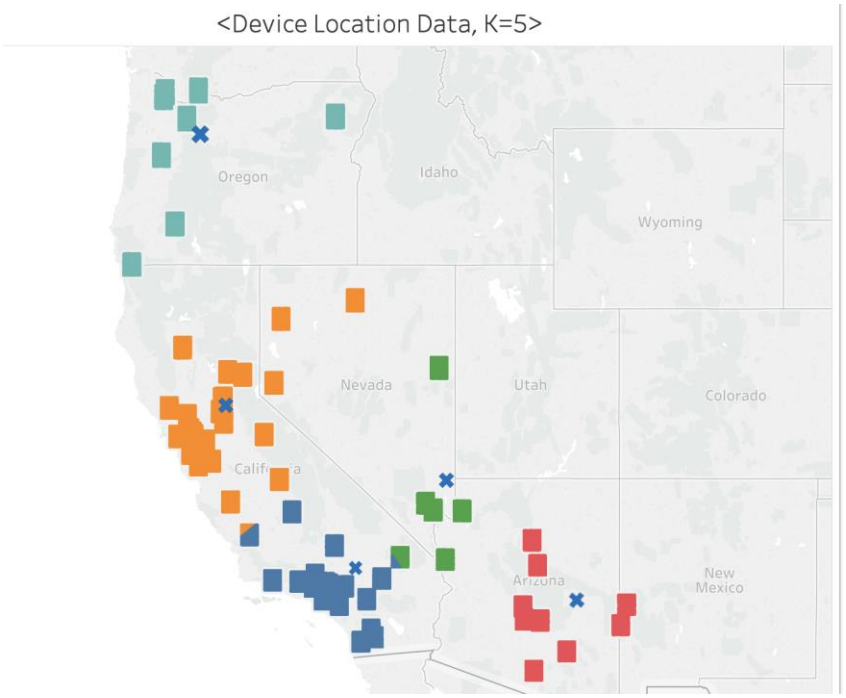
associated with it.

4. Repeat from step 2 until the location of every centroid converges.
5. The converging result is the final k clusters produced by the k-means algorithm.

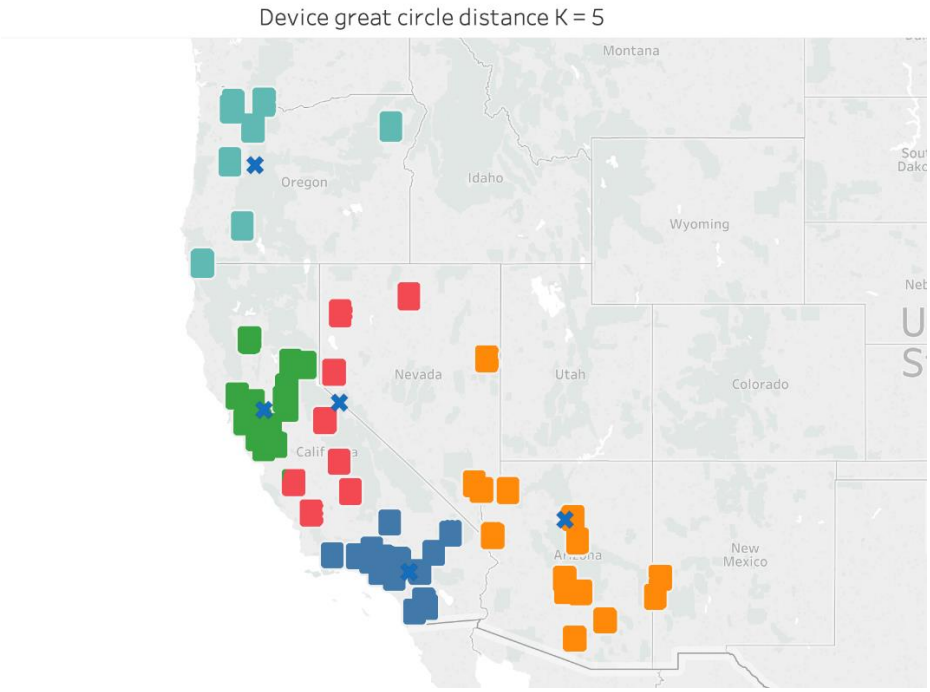
Results – Testing our K-means on Small Data / Pseudo-Cluster

The following graphs are the visualization of the product of our k-means implementation on the local. Each cluster is indicated with a different coloring, and we've marked the center of the k-means clusters with a blue X mark.

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

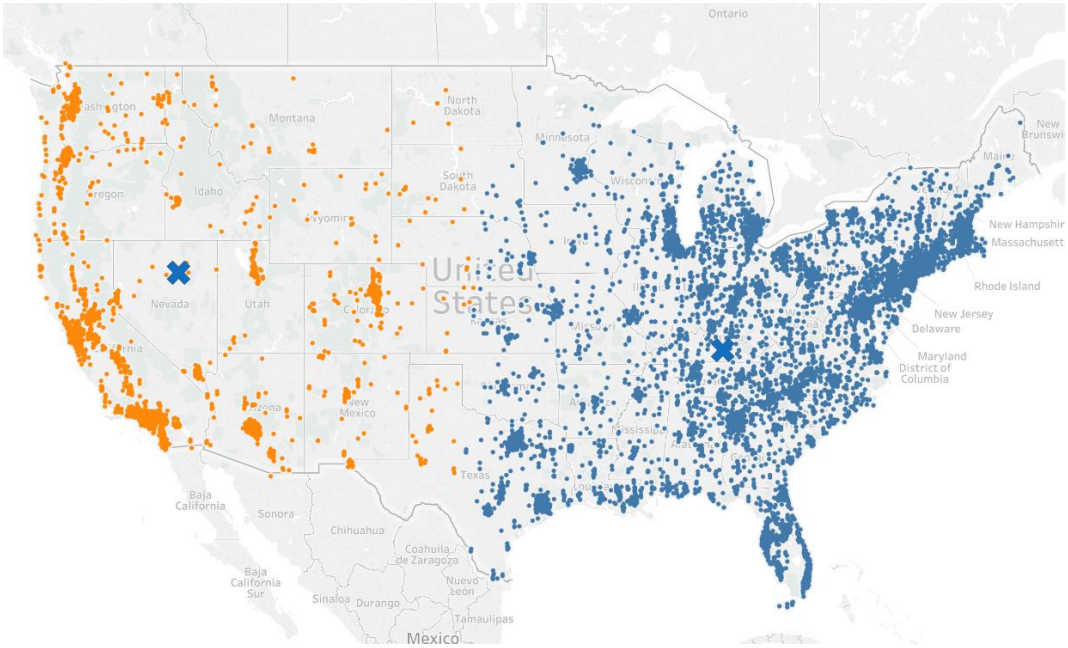


k = 2, Euclidean distance

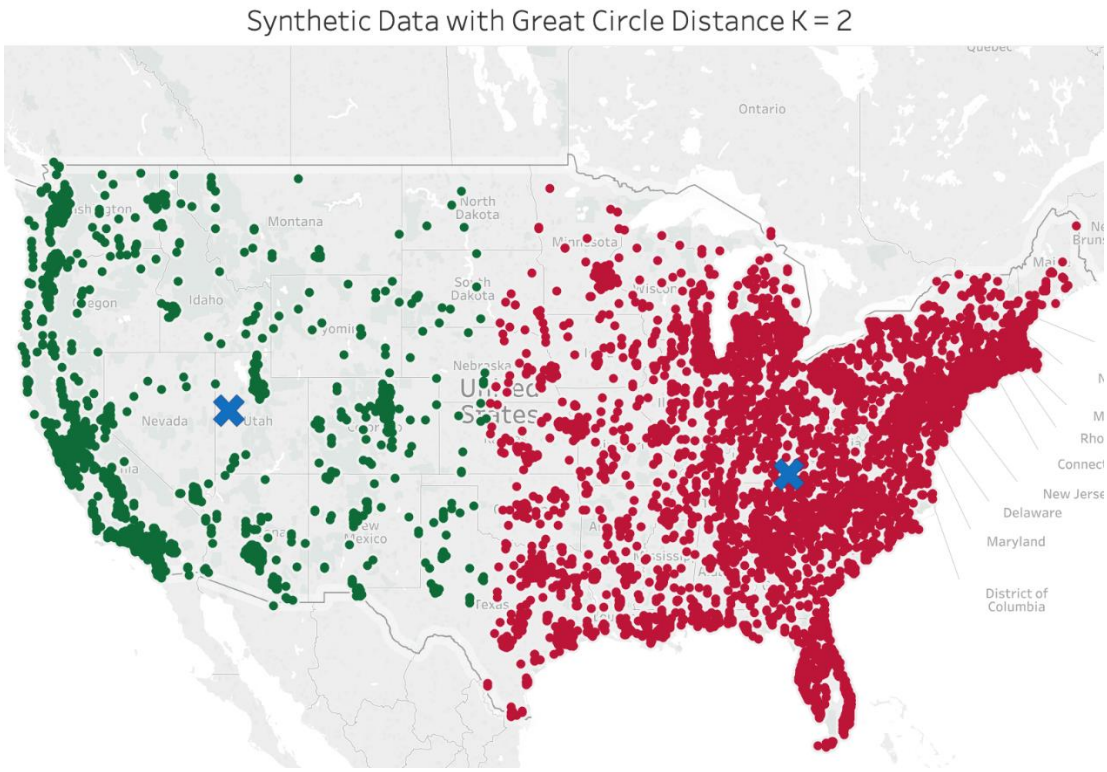


k = 2, great circle distance

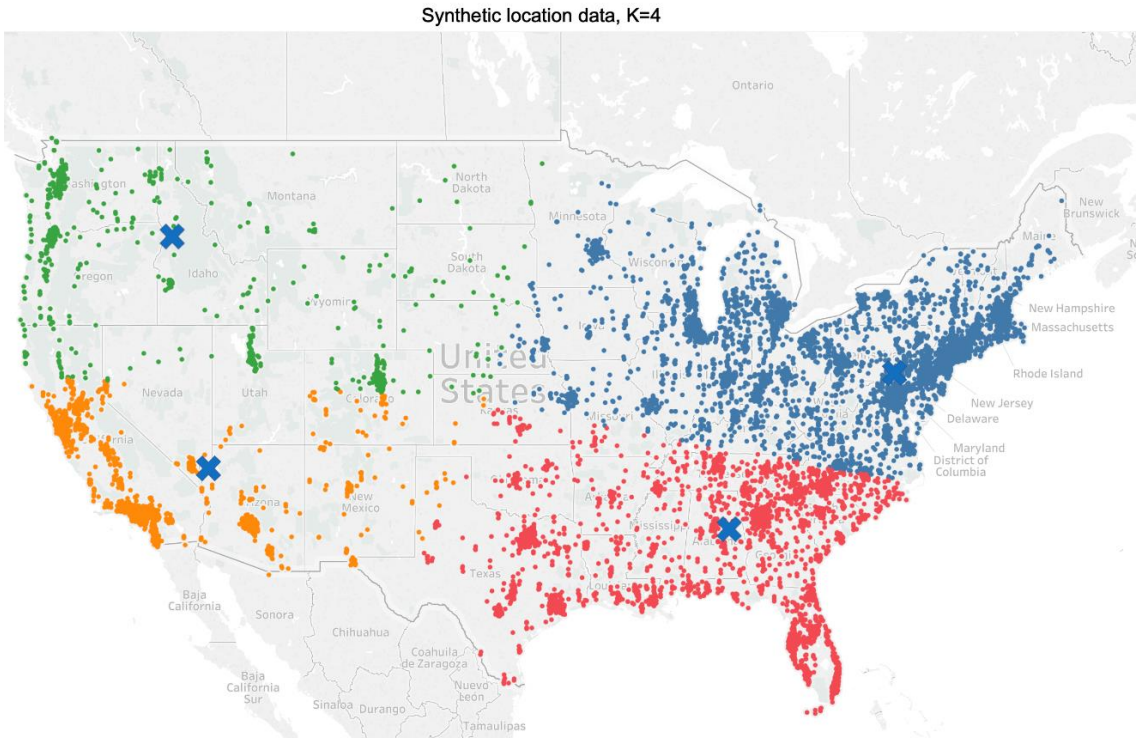
2. k-means clusters for the synthetic location data using k = 2 and k = 4.



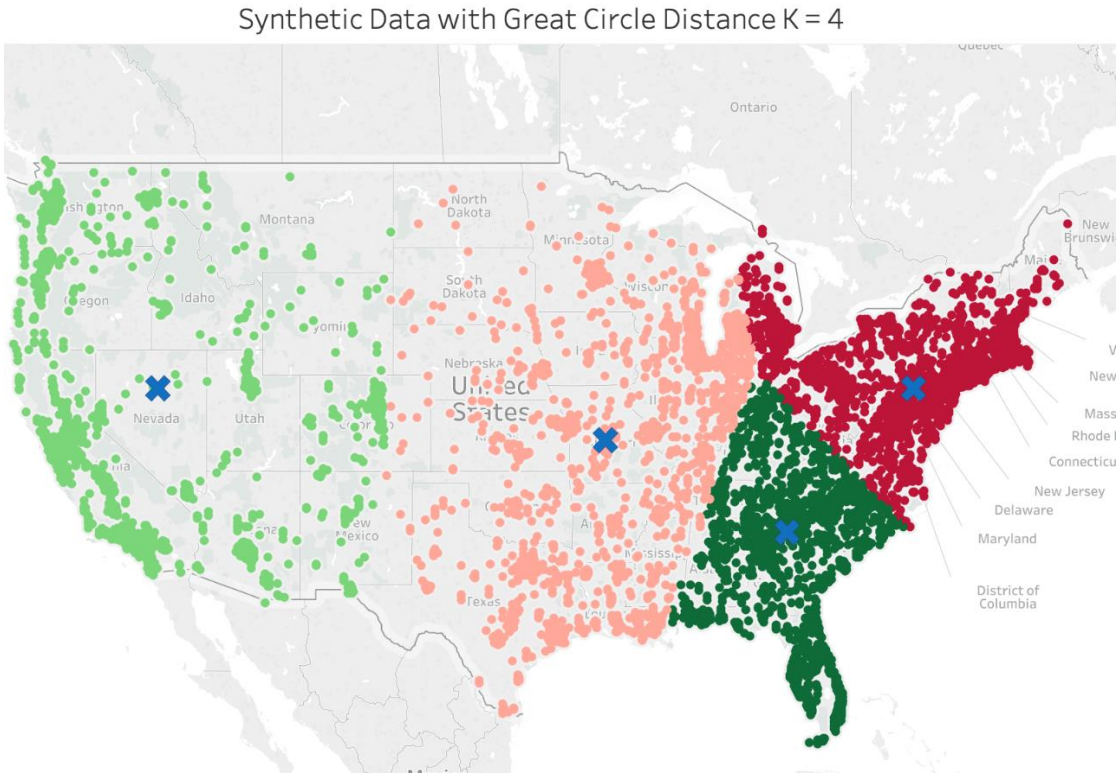
k = 2, Euclidean distance measure



k = 2, great circle distance measure

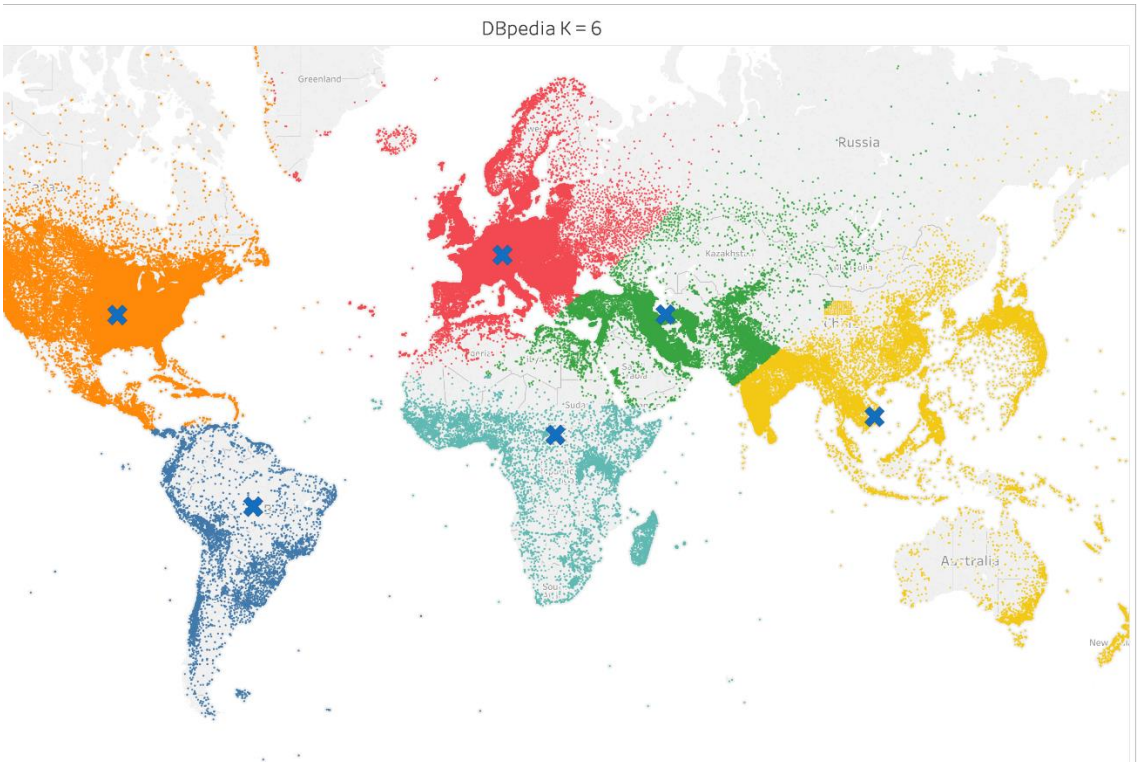


k = 4, Euclidean distance measure

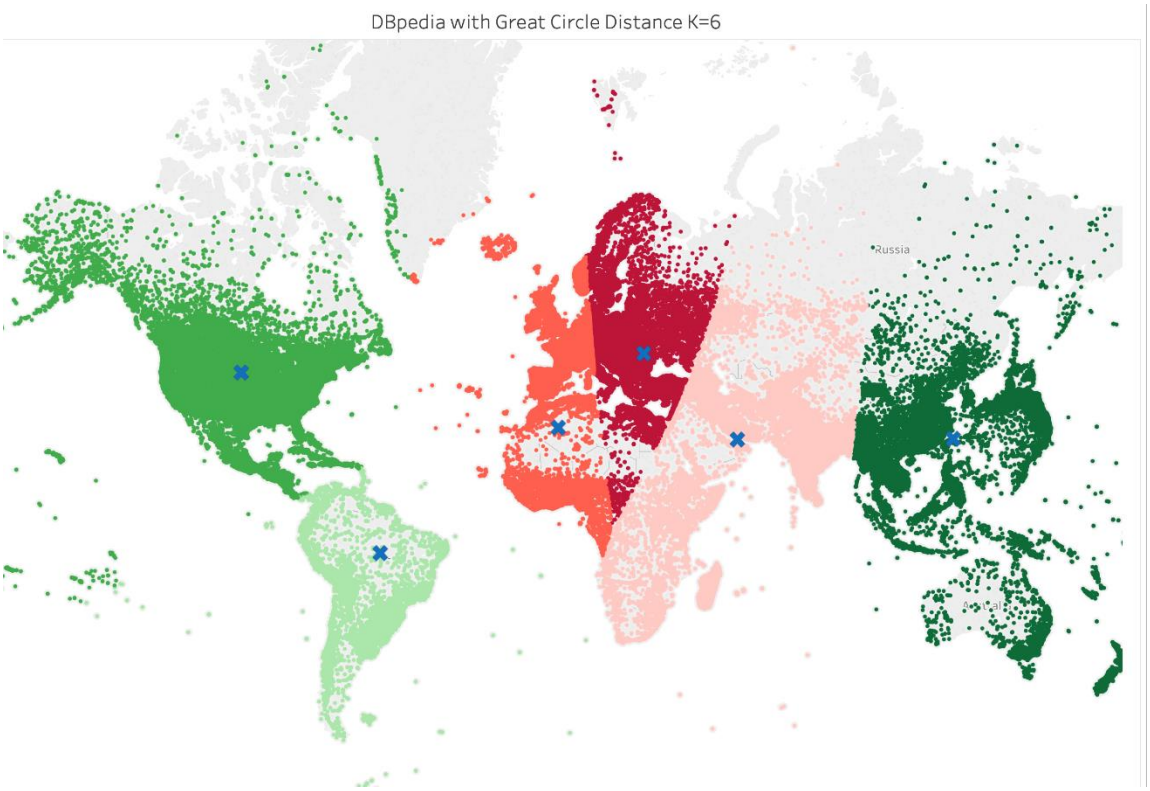


k = 4, great circle distance measure

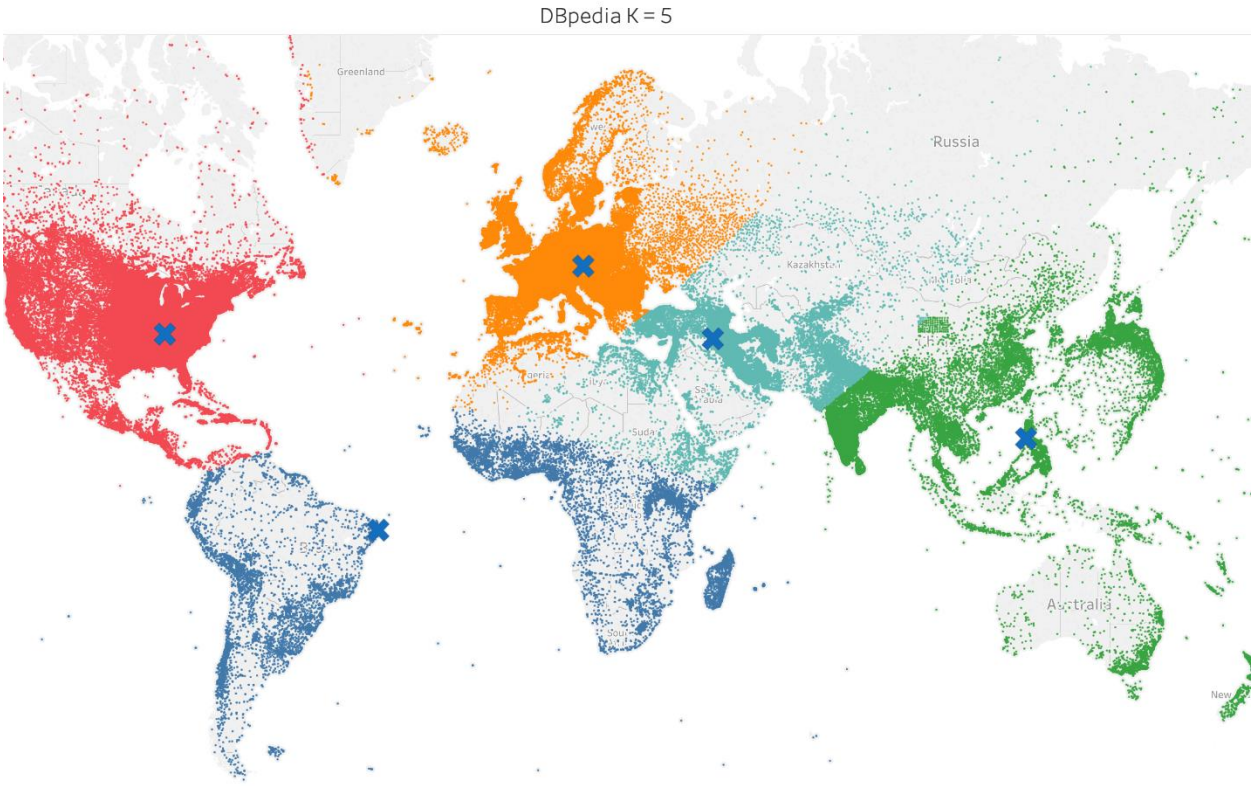
3. k-means clusters for the large-scale DBpedia location data. (k = 6)



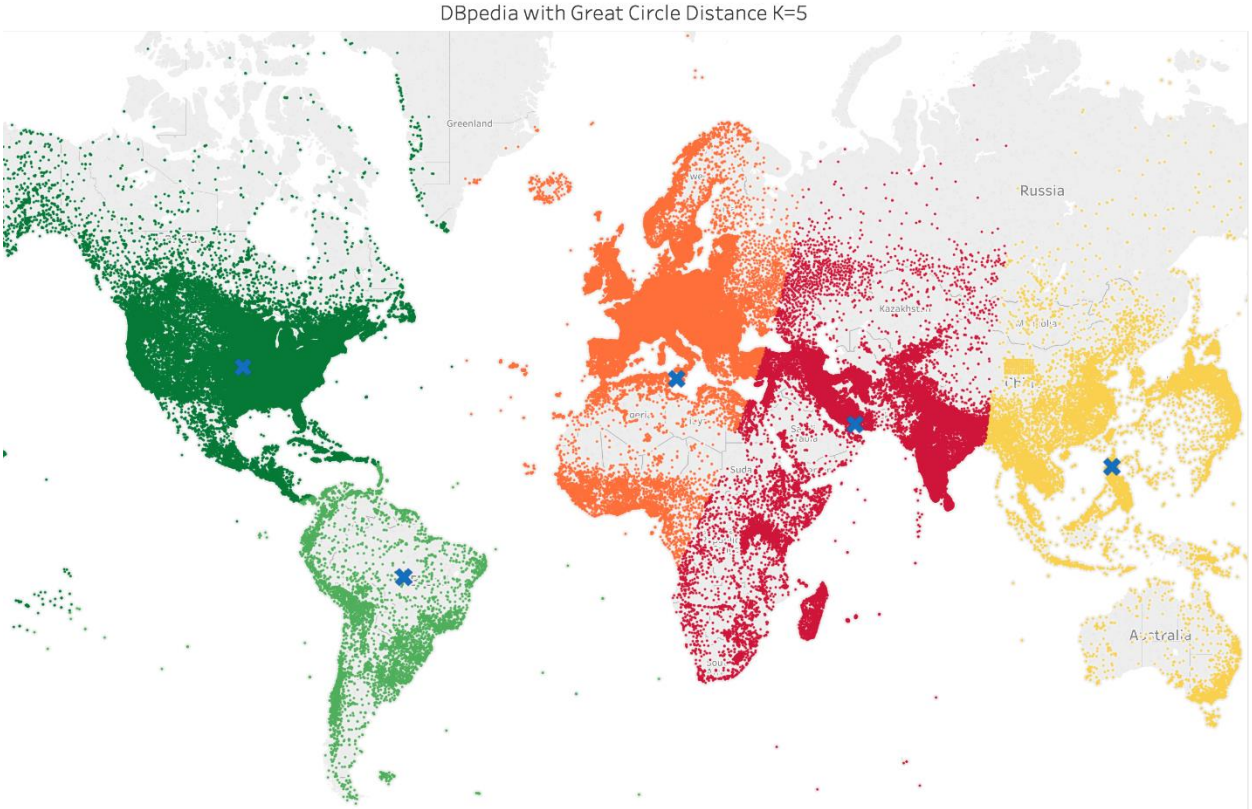
k = 6, Euclidean distance measure



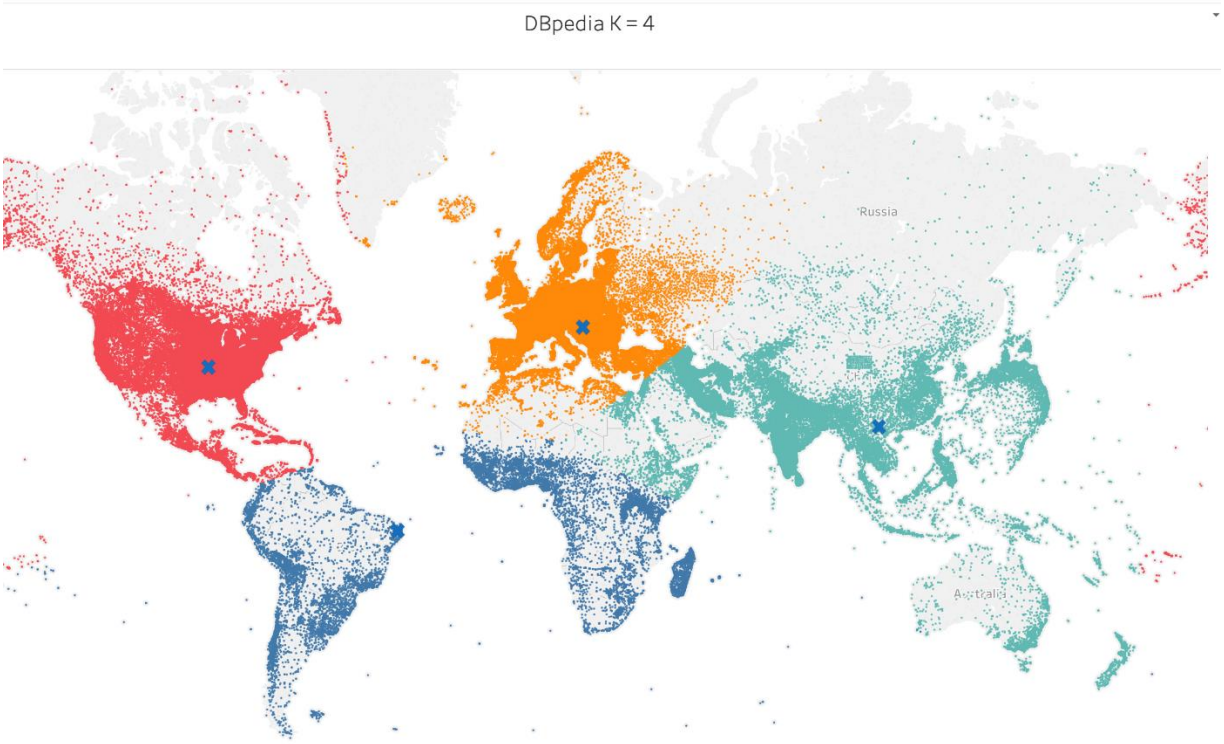
k = 6, great circle distance measure



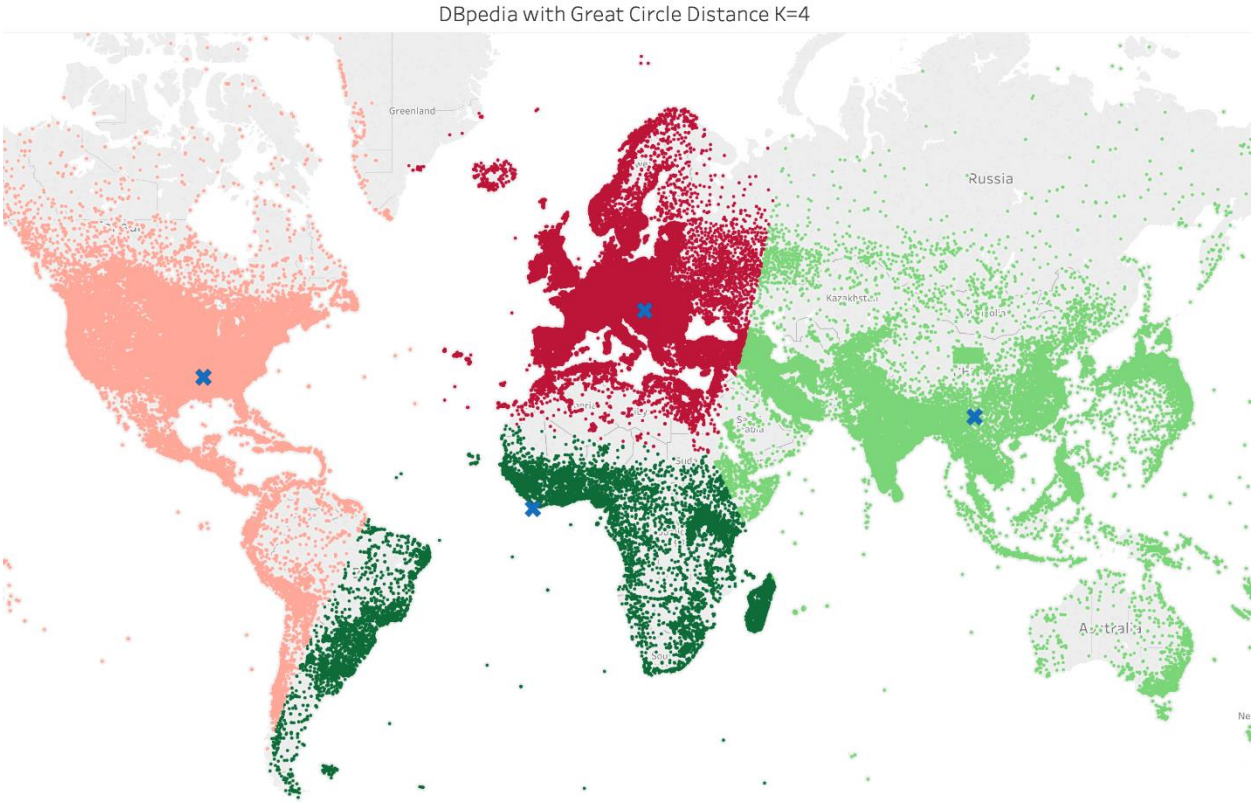
k = 5, Euclidean distance measure



k = 5, great circle distance measure



k = 4, Euclidean distance measure



k = 4, great distance measure

In the context of the DBpedia problem, the k-means location means the ideal location for the server that stores the data of the written Wikipedia pages. Thus, each increase of k means another database center built, which costs a huge amount of money. After running k-means at k = 6, 5, and 4, we concluded that k = 6 with Euclidean distance measure would be the best option, with one database in the each region without bearing too much body of water in between a massive population of users and the server.

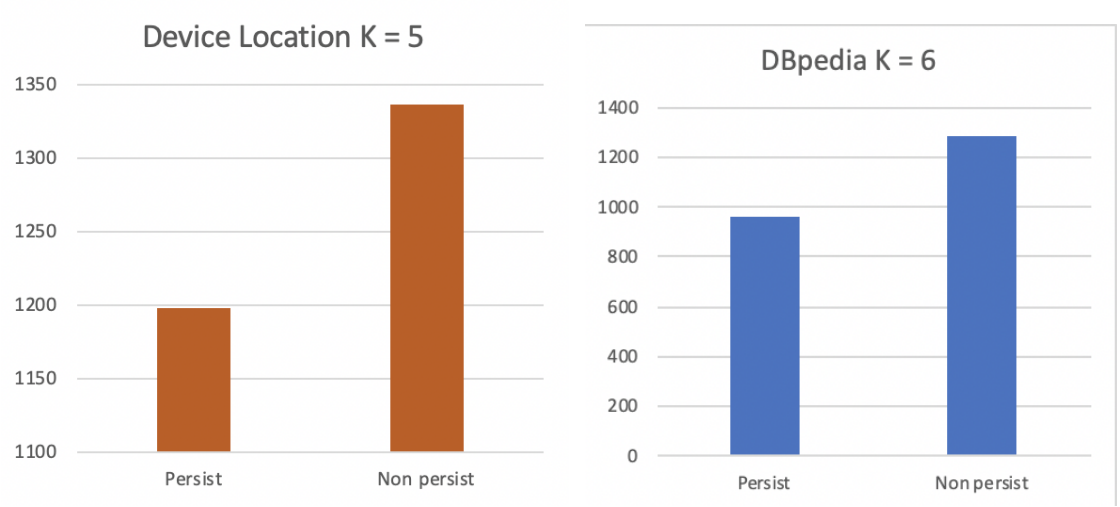
4. Runtime Analysis

Dataset	K	Measure of distance	Persist or	Runtime(sec)
Dbpeida	6	Euclidean	Yes	956.874023
Dbpeida	6	Euclidean	No	1284.34121
synthetic	2	Euclidean	Yes	15.56983
synthetic	2	Euclidean	No	14.34551
synthetic	4	Euclidean	Yes	24.63221
synthetic	4	Euclidean	No	29.87663
device location	5	Euclidean	Yes	1198.087
device location	5	Euclidean	No	1335.87224

Runtime Analysis for various datasets

When we did the runtime analysis, after reading and parsing the dataset into an RDD, we found that without caching/persist, the runtime is larger than for each of the dataset. For dataset in small volume, the difference is not that significant, as shown in the table and graph for synthetic dataset. However, for large dataset like DBpedia, not caching increased the runtime by approximately three times.

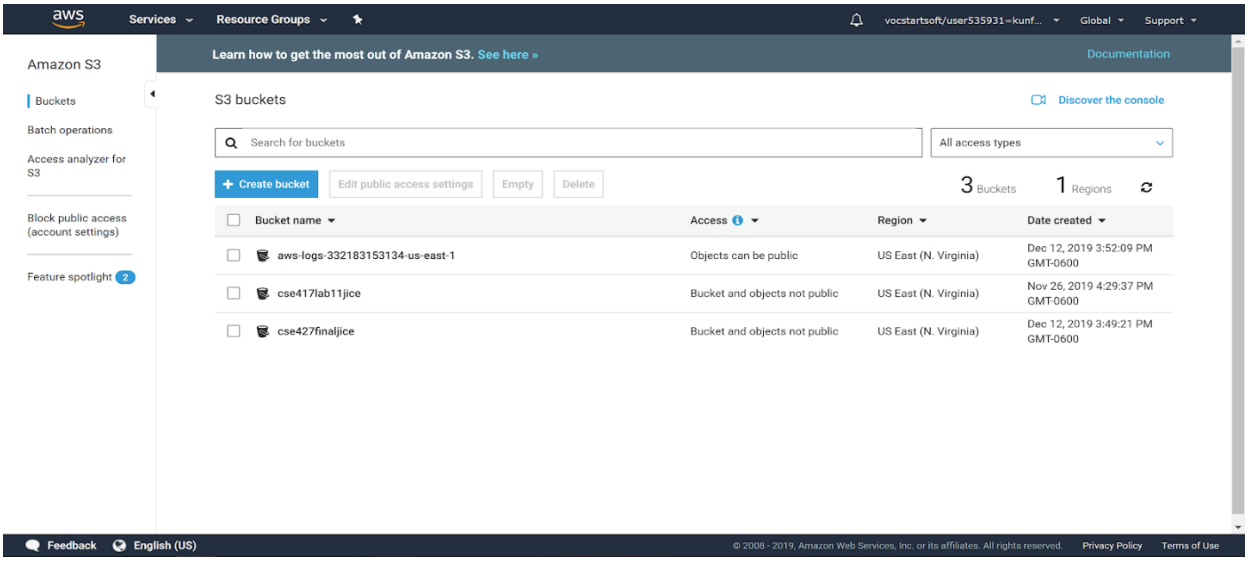
The following bar graphs indicate the performance of the persistent and non-persistent RDDs in the execution with device location data and DBpedia data.



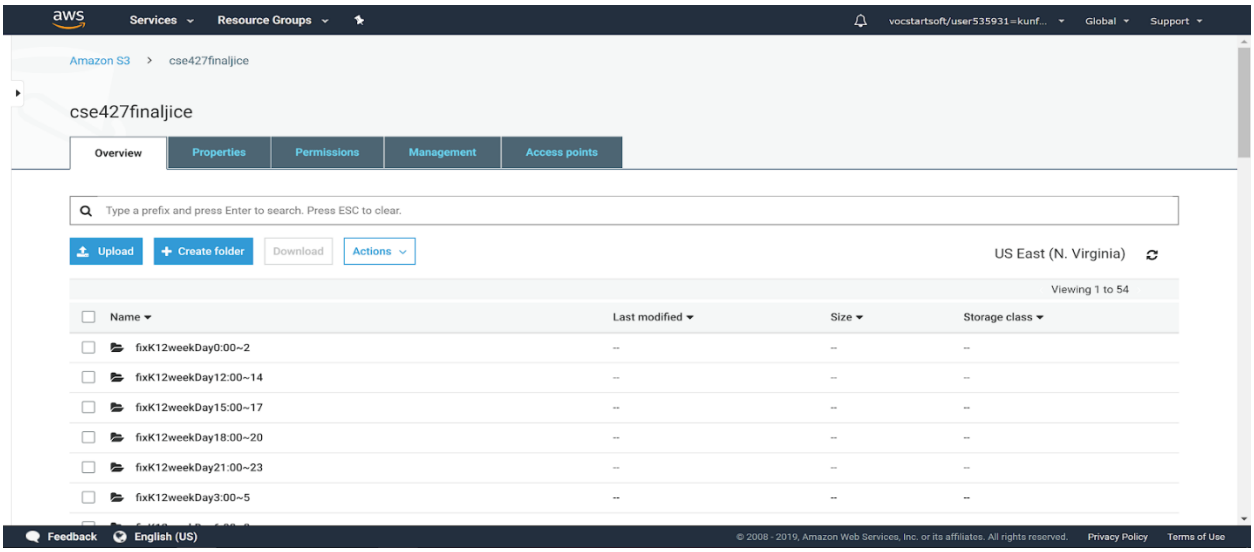
Approach – Running it on the Cloud

After testing small datasets on local pseudo clusters, we then proceed to run the k-means on Amazon Web Services using the Uber pickup location dataset which is much larger than the sample datasets and contains over 4.5 million data points. We first create a computational instance on EC2 and store our preprocessed datasets and Spark/Python program in S3 (Amazon Simple Storage Service). All configurations are set as default.

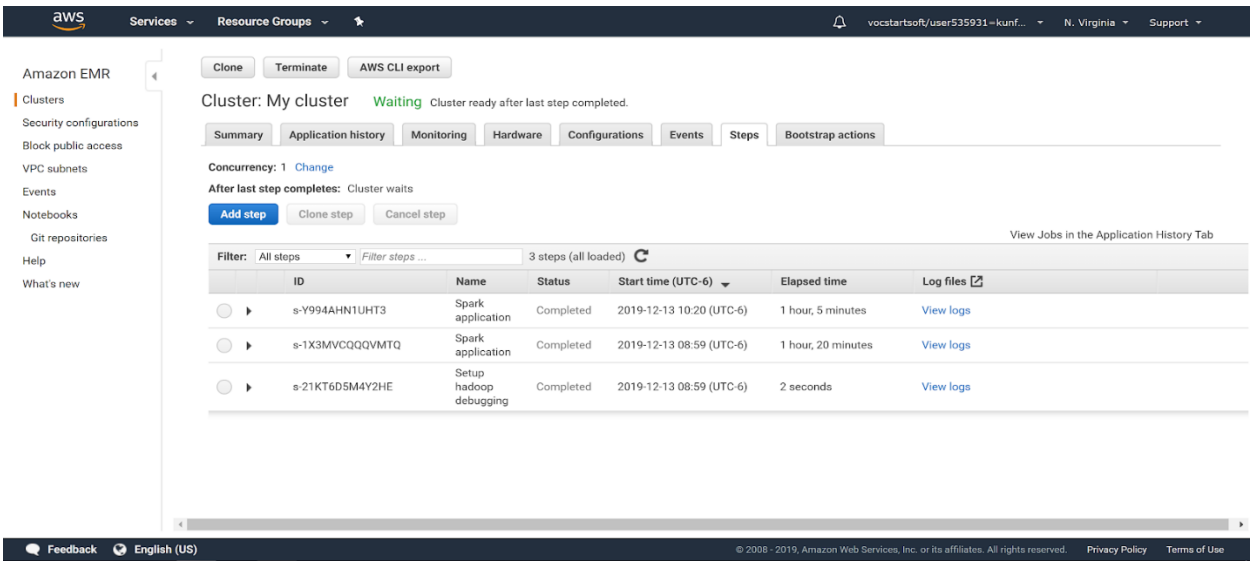
For the project, we created a new bucket called cse427finaljice and put everything we need for the project in it.



We put the preprocessed datasets in “input” in “cse427finaljice” and store the Spark/Python executable and the results directly in “cse427finaljice”. We store the logs created during the execution in “cse427finaljice” also.

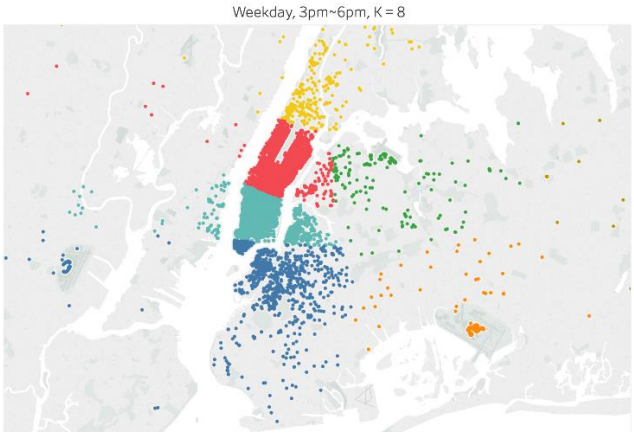


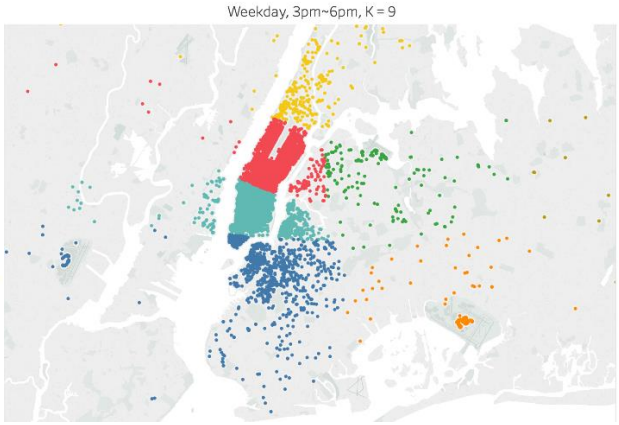
We then create a Spark cluster on Amazon EMR and run the Spark/Python executable. For each job, arguments are set as k, input location, output location. We test the situation when k=6, k=8, and k=12 using the same configurations and datasets. The executable took 1 hour and 25 minutes to complete for k=12, 1 hour and 20 minutes for k=8, and 1 hour 5 minutes for k=6. We can then thus infer that with k increasing the runtime increases.

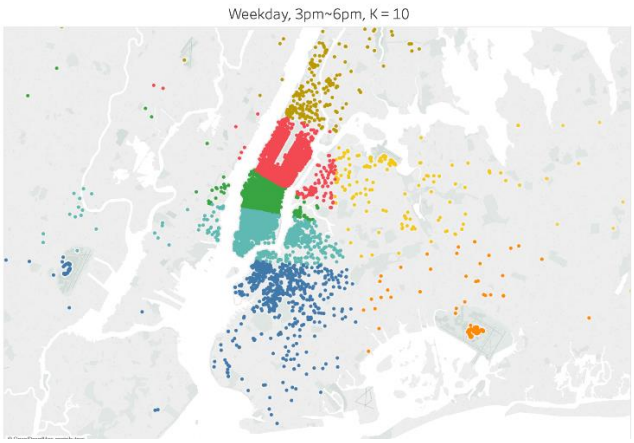


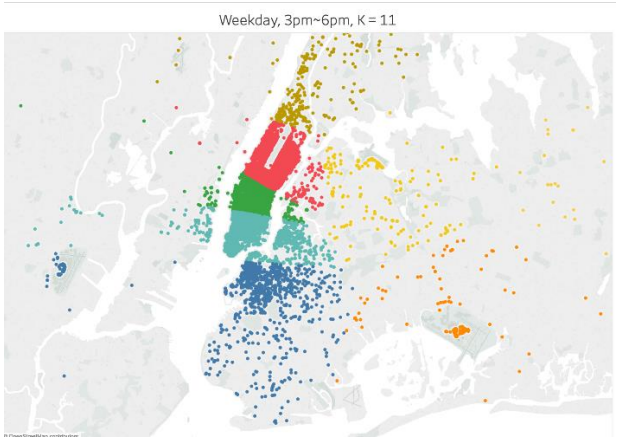
Result – Cloud

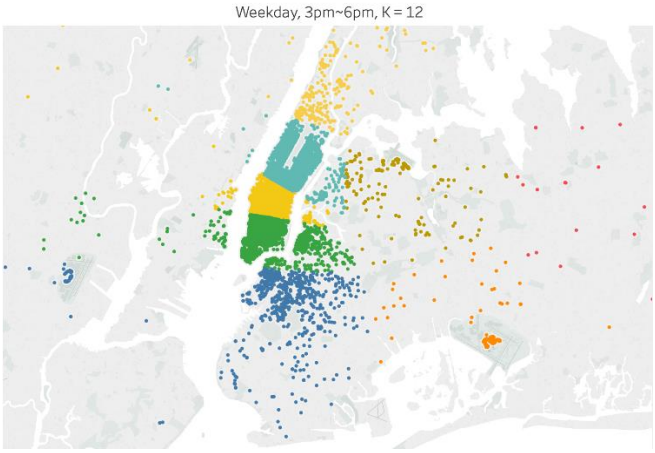
First, we ran k-means on our sample of 3pm ~ 6pm on the cloud with different k-values to decide which k value suits our database the best.





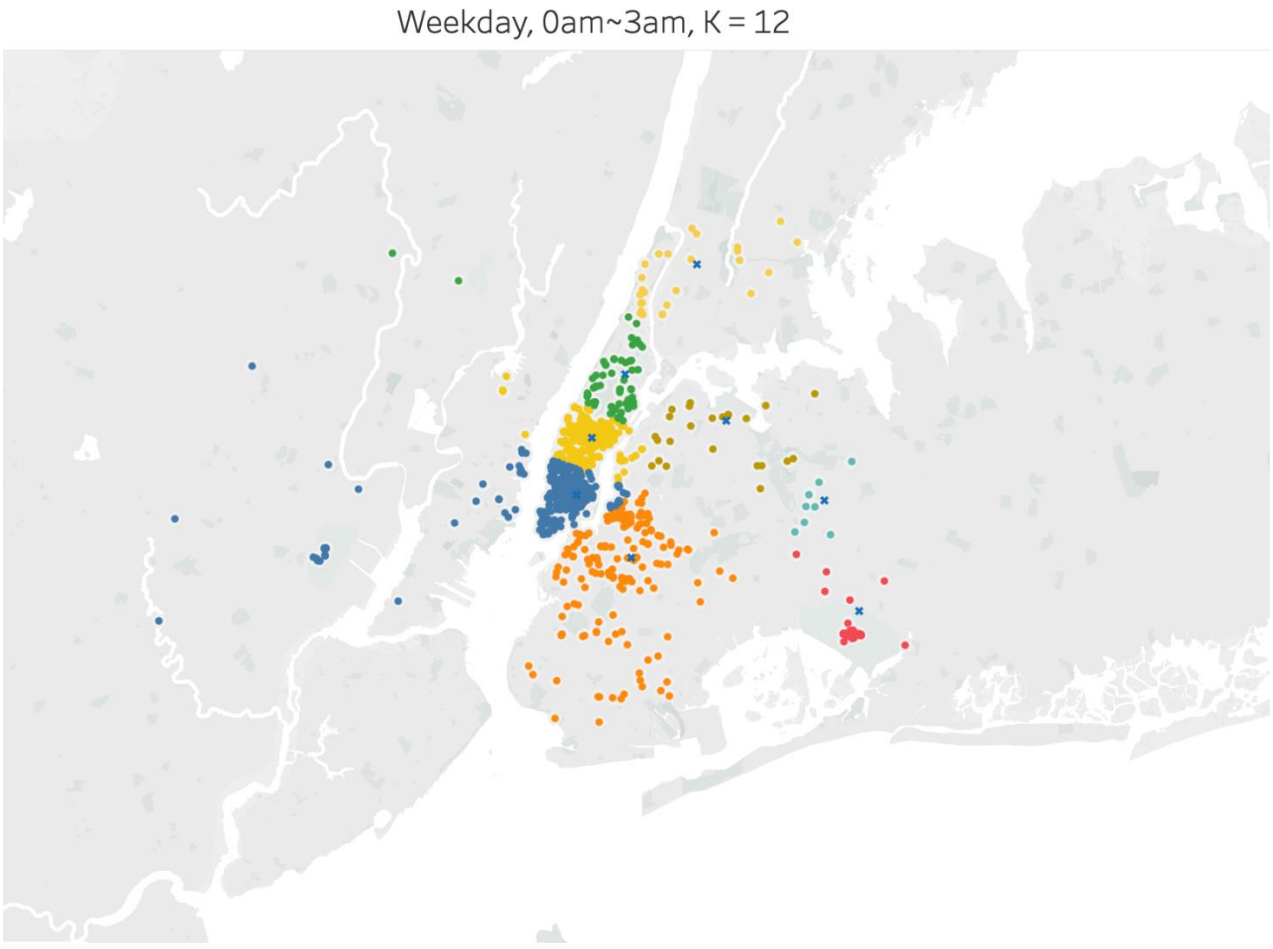




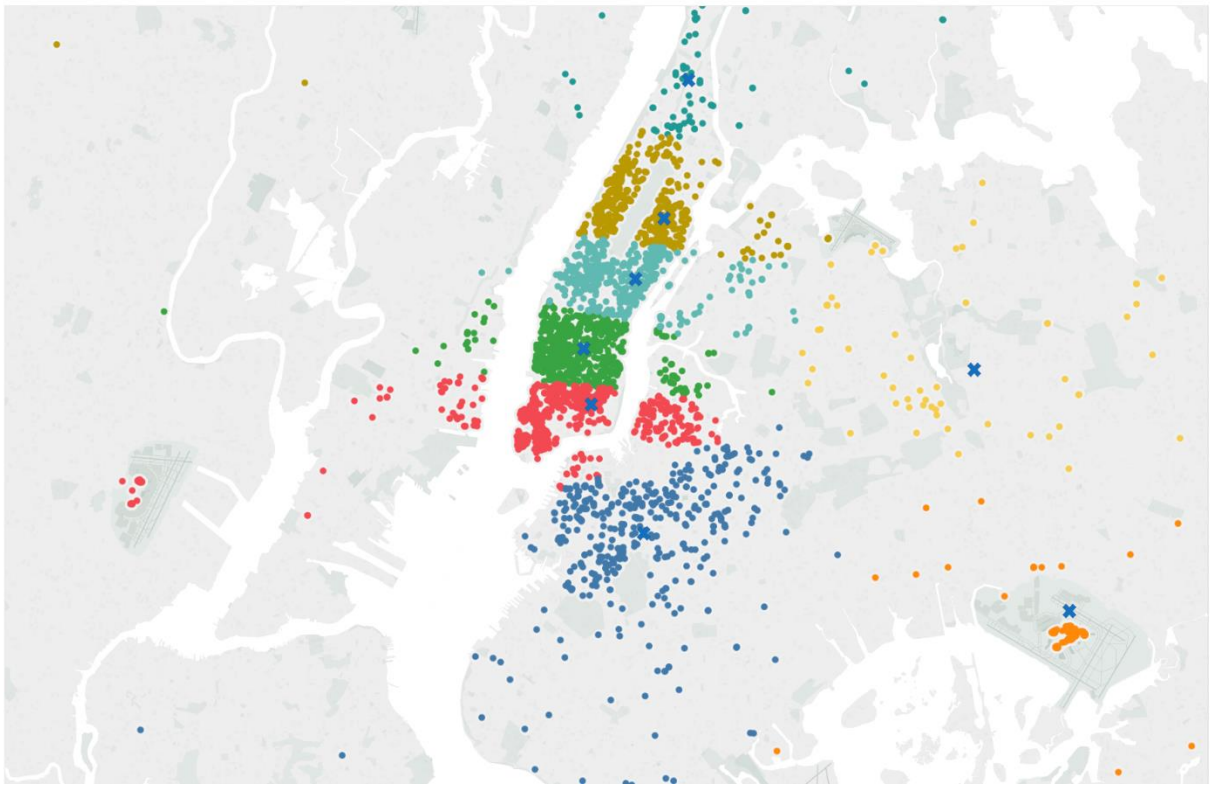


After looking at the following visualization, we concluded that k=12 has the best definition of plausible area that an Uber vehicle can cover when they receive a call from a customer. Therefore, we've selected the k-12 to be the value of our analysis for all 16 (8 time brackets * weekday and weekend) subsets of our data.

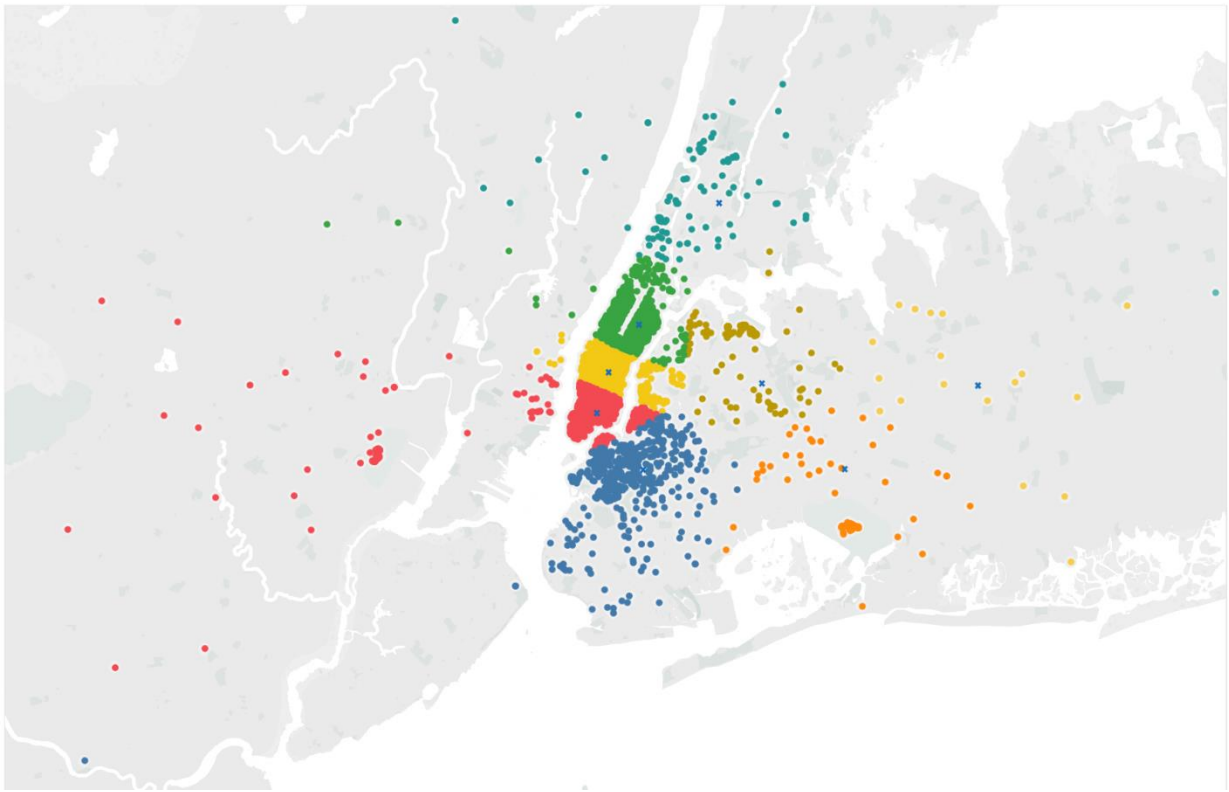
The following is the sample results of our k-means for data at three of our eight time brackets, for weekdays and the weekend.



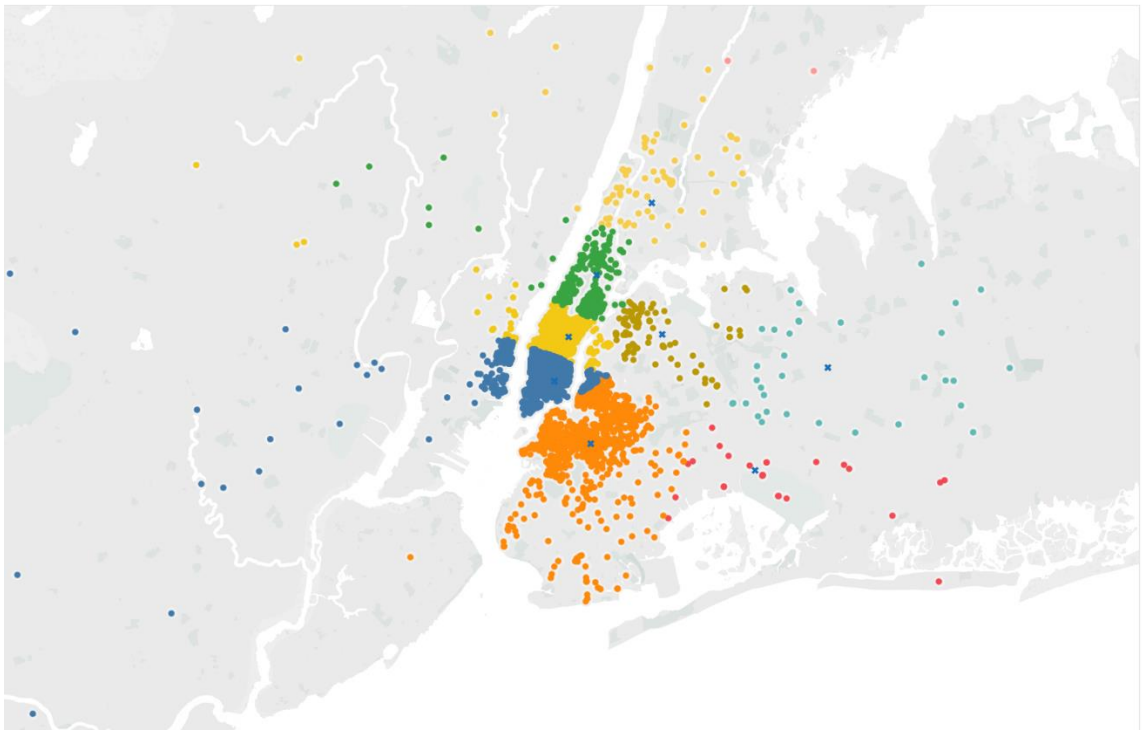
Weekday, 3am~6am, K = 12



Weekday, 9am~12am, K = 12

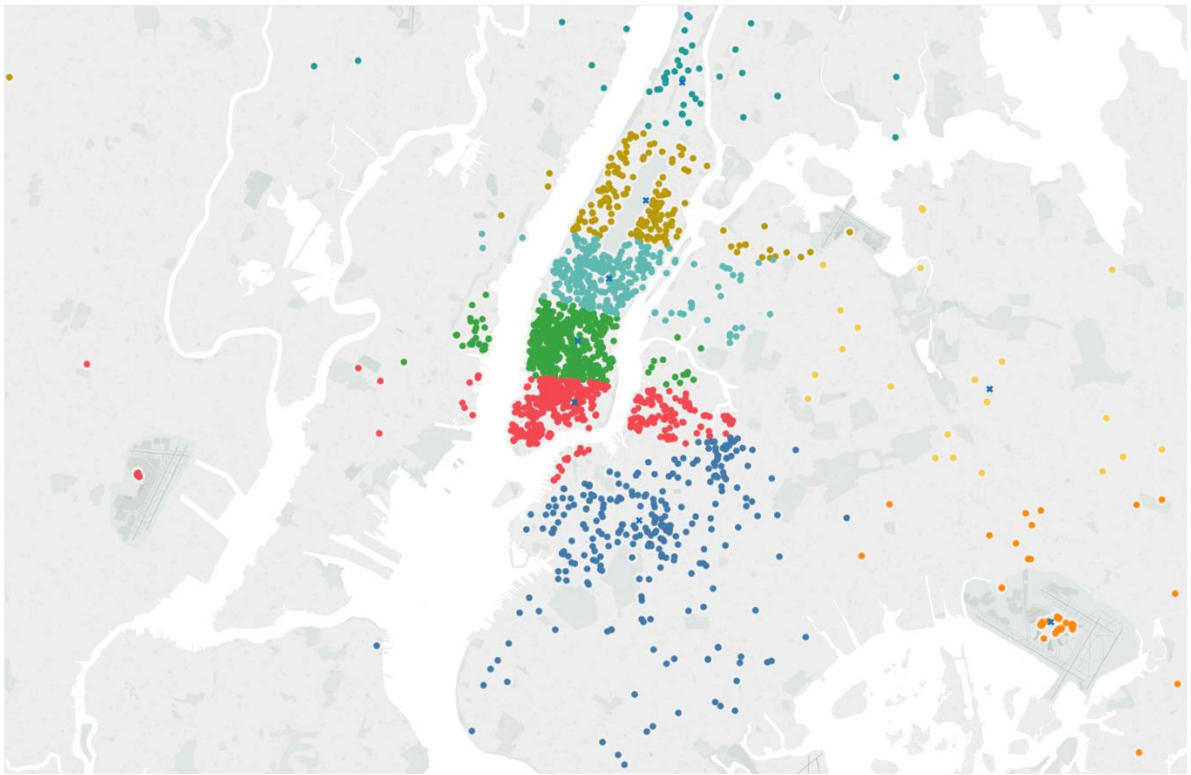


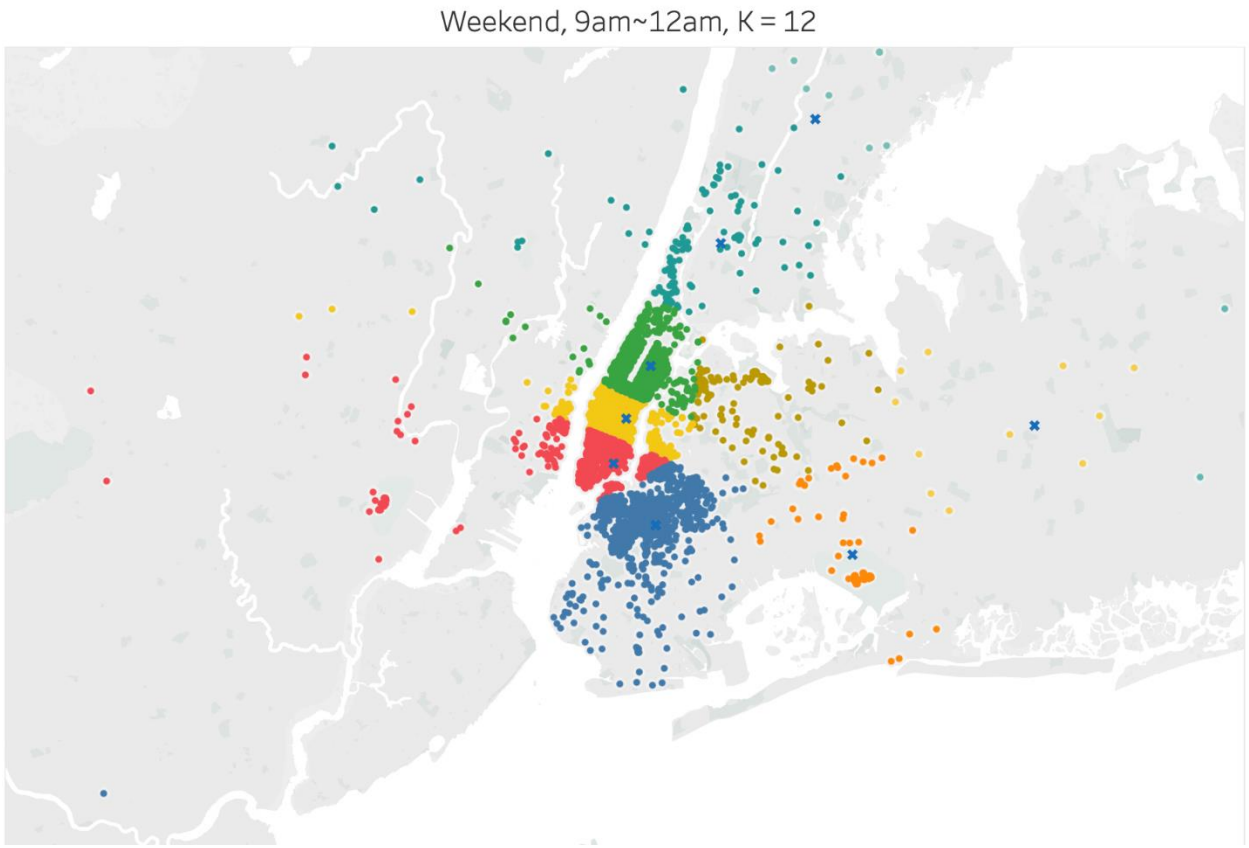
Weekday, 0am~3am, K = 12



<wrongly labeled, weekend 0~3am>

Weekend, 3am~6am, K = 12





We observe the differing behavior of the ride requests made by people and the changing k-means according to the different input. This reflects the different recommendations that will be sent out to the Uber vehicles according to the time and day.

Conclusion

After we processed the filtered Uber pickup data, we've observed that k-means for the clusters can serve as the recommendations for idle locations for Uber rides, since the likeliness of the call happening from the convenient proximity of the k-mean is high. For some time brackets, we've observed that the density of these clusters become high as the population density of the city center becomes oversaturated. We interpret that more and more Uber rides are requested within

these areas and concluded that we can weight the number of idle vehicles needed according to the density of the k-means. For the clusters that are sparser and are located in the outskirt of the city, we can manage to dispatch fewer designated vehicles for standby.

One shortcoming of the algorithm that we thought of is that the overly saturated areas usually have a high rate of traffic on itself, and there will be physical difficulties for the Uber vehicles to standby. We believe if we can integrate the traffic data with the recommendation as a future task, we can improve the recommendation to avoid too much traffic.

Another means to improve the accuracy of our algorithm would be integrating real-time pickup data to the data set we're looking at. Using technologies like Flume, which allows us to receive streaming data, we can update the coordinates being considered for the k-means clustering in real-time. Such methods will help in creating a more customized recommendation for the population behavior of that day.

Sources

Uber Data : https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city