UNIVERSITY OF TWENTE

SUGGESTING ELECTRIC CAR CHARGING STATIONS' LOCATIONS

[Final Project Report]

ABSTRACT

Suggesting the locations for electric car's charging stations based on frequent routes. This project is undertaken at University of Twente under the supervision of Professor Doina Bucur for the course Managing Big Data 1B (2019 – 2020).

Group Members:

Ashish Singhal – s2279444
Mike Pingel – s1752065
Smrithi Menon – s2336413
Yash Gupta – s2323303

1. ABSTRACT

Governments have been encouraging citizens to use electric vehicles for a long time now. However, people are still hesitant since it takes a lot of time for them to even reach charging the stations let alone the charging time. The project aims to help government organizations and companies to identify ideal locations for planting charging stations based on the most frequent routes of the city. A dataset of a city has been taken and used to cluster out the most frequent routes. Ideal charging station points have then been identified based on user density inside clusters. Real time effects and results of charging stations built at these identified locations have been discussed at the end.

2. INTRODUCTION

Electric vehicles are the future of transportation in this century and a dire need for the environment. Various automobile companies have started manufacturing and selling electric vehicles foreseeing the future already and the consumers are still hesitant to buy electric vehicles due to poor batteries. A frequent user of electric vehicle needs to charge his vehicle daily. Hence there is a high demand for charging stations and to make electric vehicles as the future of transportation it is important to meet this demand efficiently. To promote the use of electric vehicles, governments are coming forward and planting public electric charging stations at various sites. This deployment of charging infrastructure is the prerequisite for the spread of electric vehicles. A well-established charging network increases vehicle mile which relieves range anxiety among the consumers and encourage them to trust in electric vehicles more.

The objective of this project is to deduce the method by which we can help the government or the concerned city's municipality in identifying the sites where electric car charging stations can be planted by which a consumer can drive an electric car for a long time and long distance without any kind of anxiety.

2.1 PROBLEM STATEMENT

It is found that, in cities generally charging stations are placed concentratedly in one area, preferably in the center of the city or sparingly planted at the boundaries of the city. Many municipalities plants charging stations in the center of the city as it is easily accessible from all the ends of the city in terms of routes and hence there is a high demand as well. But this demands a larger distance to travel along with a large amount of time to spend to charge the vehicle if a user is nowhere near to the center of the city.

Also, the number of charging stations planted near the boundaries are less in number as it is assumed there would be a less demand of charging stations near the boundaries of the city as correspondingly there is a high demand in the center. Additionally, the charging stations are placed far away and are highly inaccessible.

Therefore, can there by a way where charging stations are easily accessible across the city to the user irrespective of where they are and can charge their cars without any anxiety and encourage the use of electric cars more.

2.2 RESEARCH GOAL

The goal of this project is to identify the locations where charging stations can be planted by identifying the most frequently travelled routes by the people and then distributing the charging stations among these frequently travelled routes.

With distributing the charging stations among the most frequently travelled routes, it makes sure the charging stations are easily accessible from anywhere. This reduces the distance to travel to charge the vehicles and in result saving the battery. This saved battery charge now can be used in an actual commute by the user. Additionally, in the scenarios where the battery has a lower charge and there is an urgent need to charge it but cannot travel long distances like travelling to the city center or city edges, charging stations distributed among the frequent travelled comes in handy.

We considered the frequent travelled routes as there is a higher probability of the vehicles to be found on such roads. As a result, charging stations planted near to frequent routes will always be used to its full potential and charging station wouldn't be idle.

3. THE DATASET

This section describes the dataset and the city with the current charging location used in this project.

The dataset used is obtain from the document[2]. This vehicle energy dataset (VED) consists of dynamic data which is (time-stamped naturalistic driving records) and static data of 383 personal vehicles recorded for 1 year from Nov,2017 to Nov,2018 in Ann Arbor, Michigan, USA. This VED datasets contains 22 million records accumulating approximately 374,000 miles of driving ranging from highways to traffic-dense downtown area.

Static data includes all the parameters of the car which are engine type, vehicle type, vehicle class, engine configuration & displacement transmission, drive wheels and generalized weights. Dynamic data represents for each car, various trips taken by it and the corresponding latitude and longitude of the trip. Dynamic dataset captures GPS trajectories of vehicles along with time-series data of fuel, energy, speed and auxiliary power usage. The columns of dynamic data is: DayNum, VehId, Trip, Timestamp(ms), Latitude[deg], Longitude[deg], Vehicle Speed[km/h], MAF[g/sec], Engine RPM[RPM], Absolute Load[%], Outside Air Temperature[DegC], Fuel Rate[L/hr], Air Conditioning Power[kW], Air Conditioning Power[Watts], Heater Power[Watts], HV Battery Current[A], HV Battery SOC[%], HV Battery Voltage[V], Short Term Fuel Trim Bank 1[%], Short Term Fuel Trim Bank 2[%], Long Term Fuel Trim Bank 1[%], Long Term Fuel Trim Bank 2[%].

Ann Arbor, Michigan city used in this project has real charging locations located in the center as University of Michigan is located there. There are few charging locations around the edges of the Ann Arbor. The charging locations are clearly not distributed across the city.

Figure 1: Ann Arbor, Michigan with real charging locations

4. METHOD

This section describes the pre-processing of the dataset done and methodology implemented to find the most frequent travelled routes followed by how charging stations' sites are distributed.

4.1 DATA PROCESSING

The data is present in the csv format. Various trips passing through same route has different latitude and longitude values. This is because the position of all cars is not same while travelling the same route. For e.g. if we take a horizontal axis on a road, various car can pe present on that horizontal axis but with a different vertical axis. Same analogy can pe applied with taking vertical axis followed by horizontal axis. Hence to bring all such possible positions of car to a single point, latitude and longitude values are transformed to 3-digit decimal precision values. We've taken 3-digit decimal precision difference between two values of latitude and longitude with 3-digit decimal precision is of 43.5 meters to 102.47 meters and this difference is ideal for streets and roads [3].

The column 'VehId' represents the vehicle ids and 'Trip' represents the trip id of the corresponding vehicle. As a result, 'Trip' column has duplicate values as many vehicles could have same trip id at some point of time. Hence, trips cannot be uniquely identified just by 'Trip' column. To resolve this, 'VehId' and 'Trip' columns is merged. The resulting new column 'VehId-Trip' would represent trips. All the rows with same value of 'VehId-Trip' would be a trip.

After doing 3-digit decimal precision, there are values of latitude and longitude which are duplicate in a single trip. Those duplicate values were removed by using 'dropDuplicate()' api.

After all these data processing, data is ready to find the most frequently travelled routes. To use this data it in later stages, the data is written on the cluster in csv format.[Appendix 1]

4.2 FINDING MOST FREQUENTLY TRAVELLED ROUTES

The data after processing in previous step is used here.

The latitudes and longitudes in the data is clustered, which splits the trajectory span area into clusters. These clusters made up of the points are the frequently travelled routes or areas. Clustering helps in part of the trajectories which are not travelled frequently to get merged into nearby clusters and hence are being considered to have a charging station somewhere nearby while being not so frequently travelled. In the case of most frequently travelled routes' areas, these areas will majorly form the clusters. Hence clustering the latitudes and longitudes would result in finding the frequently travelled routes.[Appendix 4]

K-Means algorithm is used to do the clustering. The challenge in using K-means algorithm is find correct number of clusters (K-value). For that, elbow method is used to get the correct number of clusters.

4.2.1 K-Means Clustering Algorithm and Elbow Method

Clustering is the process of dividing the entire data into groups (or clusters) based on the patterns in the data. Clustering is an unsupervised learning problem as there is no target prediction but clubbing similar observations and forming different groups. K-Means clustering is one of the popular unsupervised machine learning algorithms. One of the properties of clustering is that the points in a cluster are similar to each other. In K-means algorithm, the clusters' points have similar distance between themselves and the centroid of the cluster.

To find the optimal value of 'K' for K-Means algorithm, we use elbow method[4]. For a range of values (mostly from 2 to 9), clustering is done and for each point weighted sum of squared error is calculated between the points itself and the centroid of the clusters. In the end, we have two arrays: one array containing K values (2 to 10) and another array containing WSSE value [Appendix 2]. The graph is plotted with K value array on X-axis and WSSE on Y-axis and elbow shape is identified in the graph. The corresponding K value is the optimal number of clusters. In the below example, at K=4 elbow shape is formed and hence optimal number of clusters is 4.

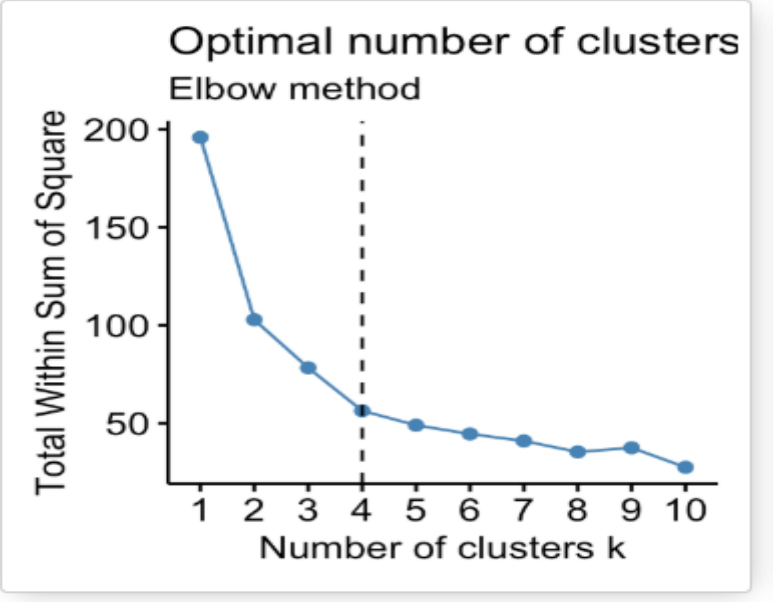


Figure 2: Example Graph for Elbow Method

The pyspark's ml library is used for K-Means algorithm. In many cases, it is difficult to visualize the elbow in the graph when the graph is slow decreasing one. 'Kneed'[5] library is used which identifies the elbow and returns the optimal K value.

4.3 DISTRIBUTING CHARGING STATIONS

After completing the clustering of the latitudes and longitudes, charging stations are distributed within the cluster.

Each cluster formed in previous step is again clustered to find the mini clusters within it and charging stations are placed at the centers of these mini clusters. Finding the mini clusters helps in finding the areas which are smaller but has busy routes/areas. Then placing the charging stations at the center of these mini clusters makes the site accessible every time to the user from anywhere.

For each cluster, optimal value of K is found by elbow method and clustering is done with K-Means clustering algorithm [Appendix 6].

4.4 PLOTTING MAPS

To plot the real-world maps shapefiles are required. Shapefiles of Michigan state is acquired from the OpenStreetMap data extracts maintained by Geofabrik[6]. Ann Arbor is a town within the Michigan state and its boundary is acquired from the government website of Ann Arbor[7]. The intersection of Michigan state map and Ann Arbor boundary is done to get the map of Ann Arbor and this is joined with the roads(routes) data.

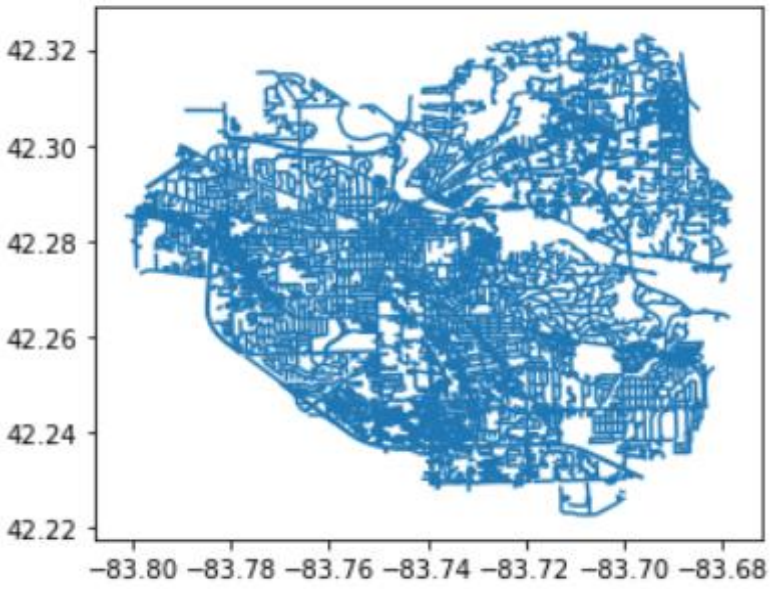


Figure 3: Ann Arbor with roads

'geopandas' library is used reads the shapefiles. EPSG:4326 co-ordinate reference system(CRS) is applied. 'matplotlib' library is used to plot the 'geopandas' data which is maps.

To plot the points (like real charging stations [Fig 1], or clustered points or mini clusters centers as charging sites], points read into the geopandas format and then plotted over the existing Ann Arbor map. While plotting the clustered points, the clustered data is transferred into the local machine and read into the geopandas format and plotted on top of the Ann Arbor map.

To find the route between two points, osmnx[8] and networkx[8] library is used. Osmnx converts the maps data into nodes and networkx uses internal Dijkstras algorithm to find the shortest route with the help of nodes data from the osmnx library. Osmnx library reads shapefiles and converts into graph structure. To uses networkx, we need to convert the latitude and longitude points input into UTM projection which gives node number in osmnx format.

4.5 FINDING AVERAGE DISTANCE BETWEEN THE SUGGESTED CHARGING STATIONS SITES AND ROUTES

After suggesting the charging locations which is centers of the mini clusters (in previous step), we verified the results by calculating the average distance between all the nearest charging stations across the whole route.

We provide the origin point and destination point. There are nodes in the path found between origin and destination. With every node, nearest charging station's distance is summed and divided by the number of nodes in the route (total length). This is done for real existing charging station as well and the result is compared to evaluate the suggested charging locations.

5. RESULTS

Through the elbow-method approach described in the previous section, it could be concluded that the optimal value of K for the KMeans algorithm is 4. Using the resulting clusters from this process [image – Appendix 6], an additional clustering was done on the data points belonging to each cluster as mentioned in previous section. Another usage of the elbow-method revealed an optimal K of 4 for every cluster. This resulted in a final number of 20 clusters and the final placement of the 20 charging stations at the center of each mini cluster. A visualisation of the clustering, graphed on a map, can be seen below:

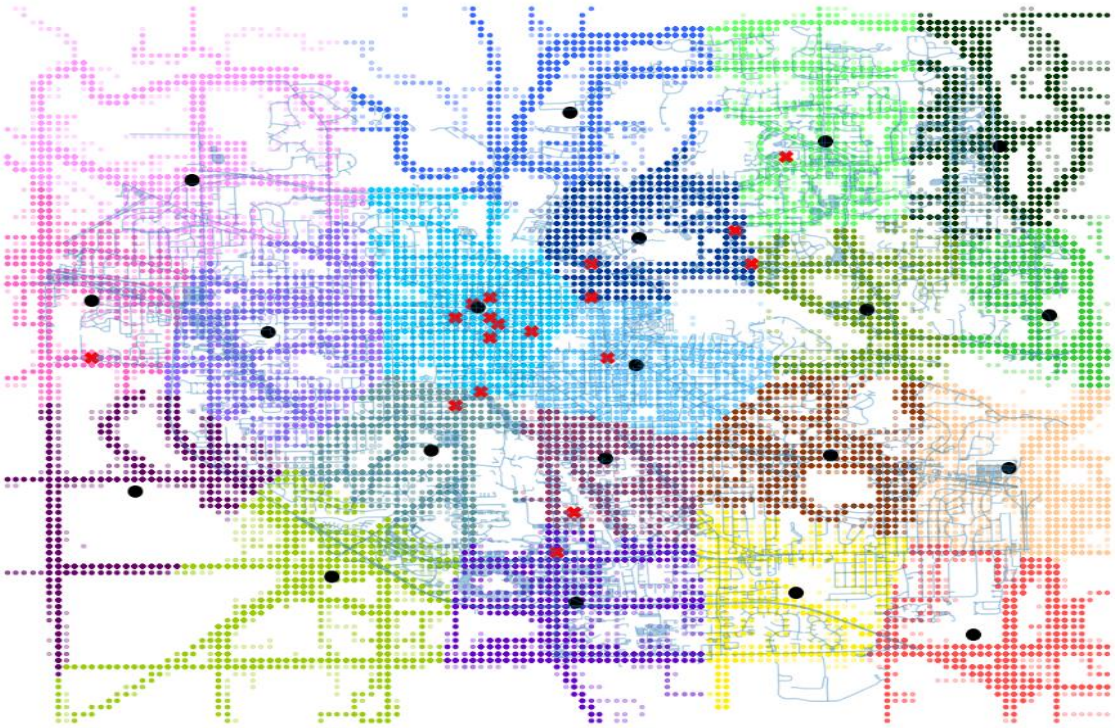
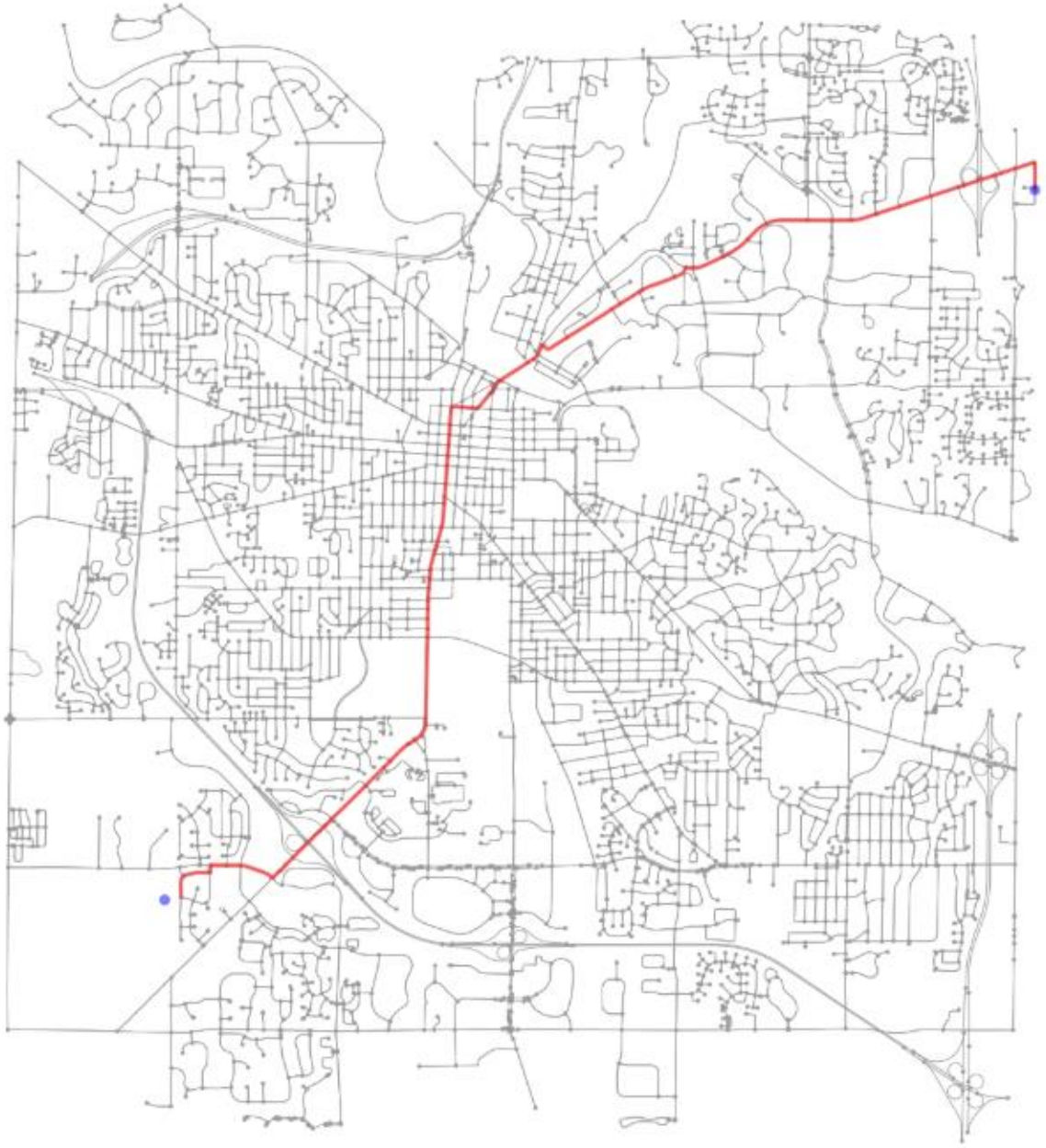


Figure 4: Mini-clusters and suggested Charging Sites (in black)

As can be seen on the map, the locations of the charging locations are now more distributed than they were before the clustering.

As explained in the Methods section, the shortest routes between 2 points were graphed to be able to evaluate the placement of the new charging locations against the old locations in the city. The evaluation metric that was used to determine the effect of the new placement is the average distance between a car on its route and the nearest charging stations. For each node placed with the osmnx library, the distance between its location and the nearest charging station was measured and added up. The final sum was finally divided by the total number of nodes to receive the final metric that allowed for a comparison between the original and the custom placement.



6. EVALUATION

To evaluate the placement in a more quantitative way, we evaluated 200 hundred different routes with randomly generated location points for the start and ending points within the city. Of all the average distances for both placements, we took the average again to receive the final evaluation measure. For the old placement of charging stations, the average distance between the cars and the nearest charging stations over 200 routes was 1900.55 meters. For the new placement this average lied at 1216.42 meters. In total, out of the 200 runs, our placement proved to be better 146 times. This is equal to 73.4% of all the runs.

7. DISCUSSION

For the evaluation purpose, we could've used battery percentage column available in the dataset and make a model which would predict how much charge would be required to travel some distance. And if the available charge is not enough to travel the distance, we could find the nearest charging station (suggested and real existing charging locations) and compare how nearby the charging stations are and can evaluate the model. But unfortunately, due to privacy reasons the dataset doesn't contain data for this column and whole column has 'NaN' values.

Another evaluation metric to find how nearby charging stations are in terms of time taken to reach there, but to make the evaluation correct we need to take traffic in consideration. Traffic influences the vehicle's speed and hence time, and though we have speed column in the dataset, but we can't consider it as there is no traffic metric. So, with the time metric, the evaluation wouldn't be accurate.

This approach can help the Ann Arbor municipality to place the charging stations in a distributed manner. In future, when number of electric vehicles are going to increase and hence the demand for the charging stations as well. At this time, this approach can be used to identify the sites. Also, this approach is generic and not specific to just Ann Arbor. Consequently, this approach can be undertaken by other government agencies in different cities across the world. There is difficulty in getting the real-world data of vehicles trajectories due to privacy reasons. One solution to this problem is to generate the synthetic data which simulates the real-world data and scenarios. This approach would work on the synthetic data as well.

8. CONCLUSION

We're probably going to see a huge hike in the number of electric vehicles in the future. Hence, we should be prepared with the services that would facilitate and fuel this growth. One of major issues of electric car owners i.e. the time required for them to reach charging stations is taken up in the project. We used a dataset that provided us with vehicle data of the city Ann Arbor, Michigan. The dataset was used to plot all the coordinates of different trips taken by travelers. Frequent routes were then identified using k-means clustering and the elbow method was used to decide the number of clusters for this data. The same process was repeated on these clusters as datasets to find mini clusters and areas with high density can be established. The centers of these mini-clusters i.e. 20 coordinates have been suggested points for charging stations to be planted. For ensuring that the suggested points would positively affect real-world situations as well, we calculated the average distance b/w every coordinate present on our plot and nearest charging stations on a route taken at random. It is seen that close to 70% of the routes that were taken at random, the average distance was lower considering the suggested points than the ones that exist in real world.

GitHub Repository Link :- (Link)

Appendix 1

trajectories.py

```
'''
        This file returns the two results.
        First Result -> Data written in csv file which contains unique points per unique
        route id.
        Second Result -> Data written in csv file which contains the unique points of
        routes.
        '''
        from pyspark.sql import SparkSession
        from pyspark.sql.functions import *

        spark = SparkSession.builder.getOrCreate()

        #UserDefinedFunction
        mergeLatLong = udf(lambda col1, col2: [str(col1)+","+str(col2)])
        mergeVehIDTrip = udf(lambda col1, col2 : col1+"-"+col2)

        #Reading one file for now
        readData       = spark.read.csv("/user/s2279444/projectData/*", header=True)

        #Making points to 3 decimal digits precision
        readData       = readData.withColumn("Latitude[deg]",
        round(readData["Latitude[deg]"],3) )
        readData       = readData.withColumn("Longitude[deg]",
        round(readData["Longitude[deg]"],3) )

        tempDataToWork = readData.select("VehId", "Trip",
        col("Latitude[deg]").alias("Latitude"),
        col("Longitude[deg]").alias("Longitude"))

        tempDataToWork = tempDataToWork.withColumn("Points", mergeLatLong(
        col("Latitude"), col("Longitude")))
        dataToWork     = tempDataToWork.withColumn("Routes", mergeVehIDTrip(
        col("VehId"), col("Trip") ))

        #Points Data
        pointsData     = dataToWork.select("Routes","Latitude","Longitude")
        pointsData     = pointsData.dropDuplicates(["Routes","Latitude","Longitude"])
        pointsData     = pointsData.select("Latitude","Longitude")
        pointsData.write.csv("PointsResult")
        ####
```

```
finalData      = dataToWork.select("Routes", "Points")
finalData      = finalData.dropDuplicates(['Routes','Points'])

#Accumulating all points based on Routes
temp = finalData.rdd
temp = temp.map(lambda row: (row[0],row[1]))
temp = temp.reduceByKey(lambda x,y: x+y)
temp = temp.toDF()

finalData = temp.select(col("_1").alias("Routes"), col("_2").alias("Points"))
finalData.write.csv("trajectoriesResult")
print("ASHISH: Trajectories is done")
```

Appendix 2

getK.py

```
'''
        This will give the array containing K from 2-9 and WSSSE values for each
        respective K.
        '''
        from pyspark.sql import SparkSession
        from pyspark.sql.functions import *
        from pyspark.ml.feature import VectorAssembler
        from pyspark.ml.clustering import KMeans

        spark = SparkSession.builder.getOrCreate()

        #Reading points
        pointsData    = spark.read.csv("/user/s2279444/PointsResult/*")
        pointsData    = pointsData.selectExpr("_c0 as Latitudes", "_c1 as Longitudes")
        pointsData    = pointsData.withColumn("Latitudes",
        pointsData["Latitudes"].cast("double"))
        pointsData    = pointsData.withColumn("Longitudes",
        pointsData["Longitudes"].cast("double"))



        #Pre Processing the dataset
        columns       = pointsData.columns
        assembler     = VectorAssembler(inputCols=columns,outputCol="features")
        dataset       = assembler.transform(pointsData)

        # Getting K for K-Means
        def getK(dataset):
               K    = []
               WSSSE = []
               for k in range (2,10):
```

```
                kmeans = KMeans().setK(k).setSeed(1)
                model = kmeans.fit(dataset)
                wssse = model.computeCost(dataset)
                K.append(k)
                WSSSE.append(wssse)
        return K,WSSSE

K,WSSSE = getK(dataset)
print("ASHISH -> Result is: ",K,WSSSE)
```

Appendix 3

kneed.py

```
''
'
    This gives the elbow which will find the K value of the cluster.
    '''
    import kneed as K
    from kneed import KneeLocator


    x = [2,3,4,5,6,7,8,9] #This reoresents different K values

    #Following 'y' values reoresents different WSSSE values respective to K values
    y1 =
    [367.77152670800854,271.10371705436893,190.08994176717303,138.94120466193476,122.0
    1468451528916,106.54955421942,88.80304403670353,85.8122857167995]
    y2 = [36.78219362962361, 26.253973534059455, 20.2281722140284, 13.871612988649709,
    12.116600943761293, 10.183542276824495, 9.081984154933728, 8.07464243307015]
    y3 = [1.4757790870741818, 0.9085881372647246, 0.6192845010706716,
    0.5138362626685297, 0.43798180025193484, 0.3952640350275361, 0.3304673920311346,
    0.3118153449550147]

    kneedle = KneeLocator(x,y3, S=1.0, curve='convex', direction='decreasing')

    print("Elbow at",round(kneedle.elbow, 3))
```

Appendix 4

cluster.py

```
'''
    This file will cluster the data. This will give the bigger clusters.
    '''
    from pyspark.sql import SparkSession
    from pyspark.sql.functions import *
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import KMeans

spark = SparkSession.builder.getOrCreate()

#Reading points
pointsData    = spark.read.csv("/user/s2279444/Cluster3/*")
pointsData    = pointsData.selectExpr("_c0 as Latitudes", "_c1 as Longitudes")
pointsData    = pointsData.withColumn("Latitudes",
pointsData["Latitudes"].cast("double"))
pointsData    = pointsData.withColumn("Longitudes",
pointsData["Longitudes"].cast("double"))

#Pre Processing the dataset
columns       = pointsData.columns
assembler     = VectorAssembler(inputCols=columns,outputCol="features")
dataset       = assembler.transform(pointsData)

kValue        = 4
kmeans        = KMeans().setK(kValue).setSeed(1)
model         = kmeans.fit(dataset)
predictions   = model.transform(dataset)

finalData     = predictions.select("Latitudes","Longitudes", "prediction")

for i in range(0,kValue):
        data = finalData.filter(finalData["prediction"] == i)
        data = data.select(data["Latitudes"], data["Longitudes"])
        dirName = "Cluster33_"+str(i)
        print("ASHISH: Running for cluster... dirName ->",dirName)
        data.write.csv(dirName)
```

Appendix 5

miniClustering.py

```
'''
        This gives clustering within the cluster.
        '''
        from pyspark.sql import SparkSession
        from pyspark.sql.functions import *
        from pyspark.ml.feature import VectorAssembler
        from pyspark.ml.clustering import KMeans

        spark = SparkSession.builder.getOrCreate()

        generalKValue  = 4
```

```
k0              = generalKValue
k1              = generalKValue
k2              = generalKValue
k3              = generalKValue
k4              = generalKValue

for i in range(0,5):
        pointsData     = spark.read.csv("/user/s2279444/Cluster"+str(i)+"/*")
        pointsData     = pointsData.selectExpr("_c0 as Latitudes", "_c1 as
Longitudes")
        pointsData     = pointsData.withColumn("Latitudes",
pointsData["Latitudes"].cast("double"))
        pointsData     = pointsData.withColumn("Longitudes",
pointsData["Longitudes"].cast("double"))

        #Pre Processing the dataset
        columns        = pointsData.columns
        assembler      = VectorAssembler(inputCols=columns,outputCol="features")
        dataset        = assembler.transform(pointsData)

        kValue        = generalKValue
        if(i==0):
                kValue = k0
        elif(i==1):
                kValue = k1
        elif(i==2):
                kValue = k2
        elif(i==3):
                kValue = k3
        elif(i==4):
                kValue = k4

        kmeans         = KMeans().setK(kValue).setSeed(1)
        model          = kmeans.fit(dataset)
        predictions    = model.transform(dataset)

        print("Cluster Centers of cluster",str(i))
        centers       = model.clusterCenters()
        for center in centers:
                print(center)

        finalData      = predictions.select("Latitudes","Longitudes",
"prediction")

        for j in range(0,kValue):
                data = finalData.filter(finalData["prediction"] == j)
```

```
                    data = data.select(data["Latitudes"], data["Longitudes"])
                    dirName = "Cluster"+str(i)+"_"+str(j)
                    print("ASHISH: Running for cluster... dirName ->",dirName)
                    data.write.csv(dirName)
```

Appendix 6:

getKforSmallerClusters.py

```
'''
        This file will give the array containing K from 2-9 and WSSSE values for each
        respective K for CLUSTERS already formed
        '''
        from pyspark.sql import SparkSession
        from pyspark.sql.functions import *
        from pyspark.ml.feature import VectorAssembler
        from pyspark.ml.clustering import KMeans

        spark = SparkSession.builder.getOrCreate()

        totalNumberOfClusters = 5
        #Reading the Clusters
        clusters = {}
        for clusterNumber in range(0,totalNumberOfClusters):
                read                    =
        spark.read.csv("/user/s2279444/Cluster"+str(clusterNumber)+"/*")
                read                    = read.selectExpr("_c0 as Latitudes", "_c1 as
        Longitudes")
                read                    = read.withColumn("Latitudes",
        read["Latitudes"].cast("double"))
                read                    = read.withColumn("Longitudes",
        read["Longitudes"].cast("double"))
                clusters[clusterNumber] =  read

        #Making pre-processing of data globally
        tempGlobalCluster = clusters[0]
        columns           = tempGlobalCluster.columns
        assembler         = VectorAssembler(inputCols=columns,outputCol="features")

        def getK(dataset):
                K     = []
                WSSSE = []
                for k in range(2,10):
                        kmeans = KMeans().setK(k).setSeed(1)
                        model = kmeans.fit(dataset)
                        wssse = model.computeCost(dataset)
```

```
            K.append(k)
            WSSSE.append(wssse)
        return [K,WSSSE]


result  = {}
for clusterNumber in range(0,totalNumberOfClusters):
        currentCluster          = clusters[clusterNumber]
        dataset                 = assembler.transform(currentCluster)
        result[clusterNumber]   = getK(dataset)

print("Result of WSSSE for each bigger cluster is: ",result)
```

Appendix 7:

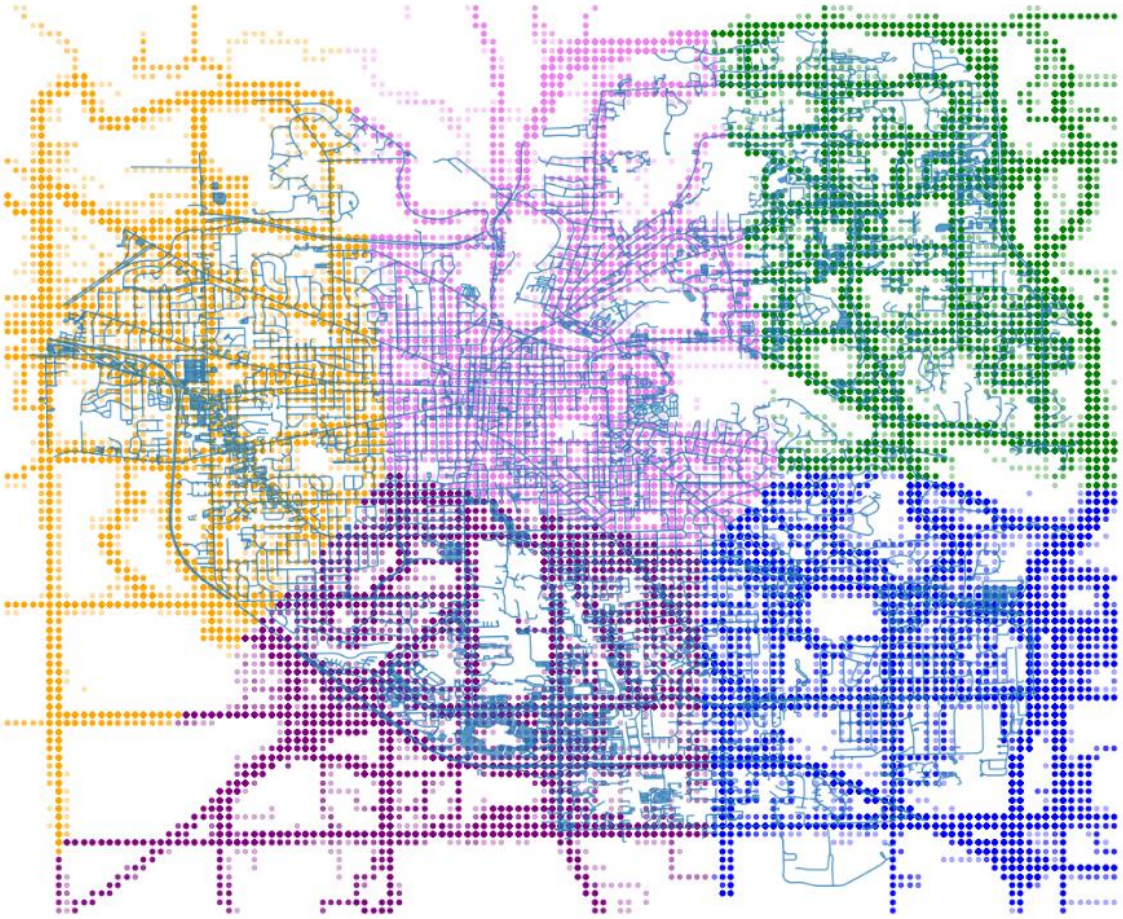


Figure 5: Clustering of Routes

Appendix 8:

Code for plotting maps, routes and clusters.
As this code is done in Jupyter Notebook, adding the notebook in the report would make

the report huge (more than 40 pages). Hence, please refer notebook in GitHub code repository.

References:

[1]. https://en.wikipedia.org/wiki/Charging_station#Locations

[2]. https://arxiv.org/abs/1905.02081

[3]. https://en.wikipedia.org/wiki/Decimal_degrees

[4]. https://en.wikipedia.org/wiki/Elbow_method_(clustering)

[5]. https://pypi.org/project/kneed/

[6]. http://download.geofabrik.de/

[7]. https://www.a2gov.org/services/data/Pages/default.aspx

[8]. https://automating-gis-processes.github.io/2017/lessons/L7/network-analysis.html