Location suggestion for quick snack outlet

pvelalam (Paritosh Kumar Velalam), 50295537; gauravav (Gaurav Avula), 50138279

Abstract:

The concentration of mobile people at a particular location at a particular time is a major factor which can be made use of by an entrepreneur who wants to open a quick snack outlet in a city. People tend to have quick snacks while travelling to keep themselves stay energized and active. So finding out the most mobile people concentrated location would help the entrepreneur to be successful. The Taxi ride statistics can be used to identify the busiest area. We can then use clustering based machine learning algorithms such as K-means on Taxi ride dataset of NYC Taxi and Limousine Commission containing pickup time, pickup location, drop off time and drop off locations to identify the location with highest number of pickup and drop-off activities. Also the hourly concentration of people can be used to plan the inventory requirement. As spark is useful for parallel processing of big data with iterative algorithms, we will be using spark for implementing the solution. The K-means clustering algorithm implementation of mllib package of spark will be used to form the clusters.

Problem Statement:

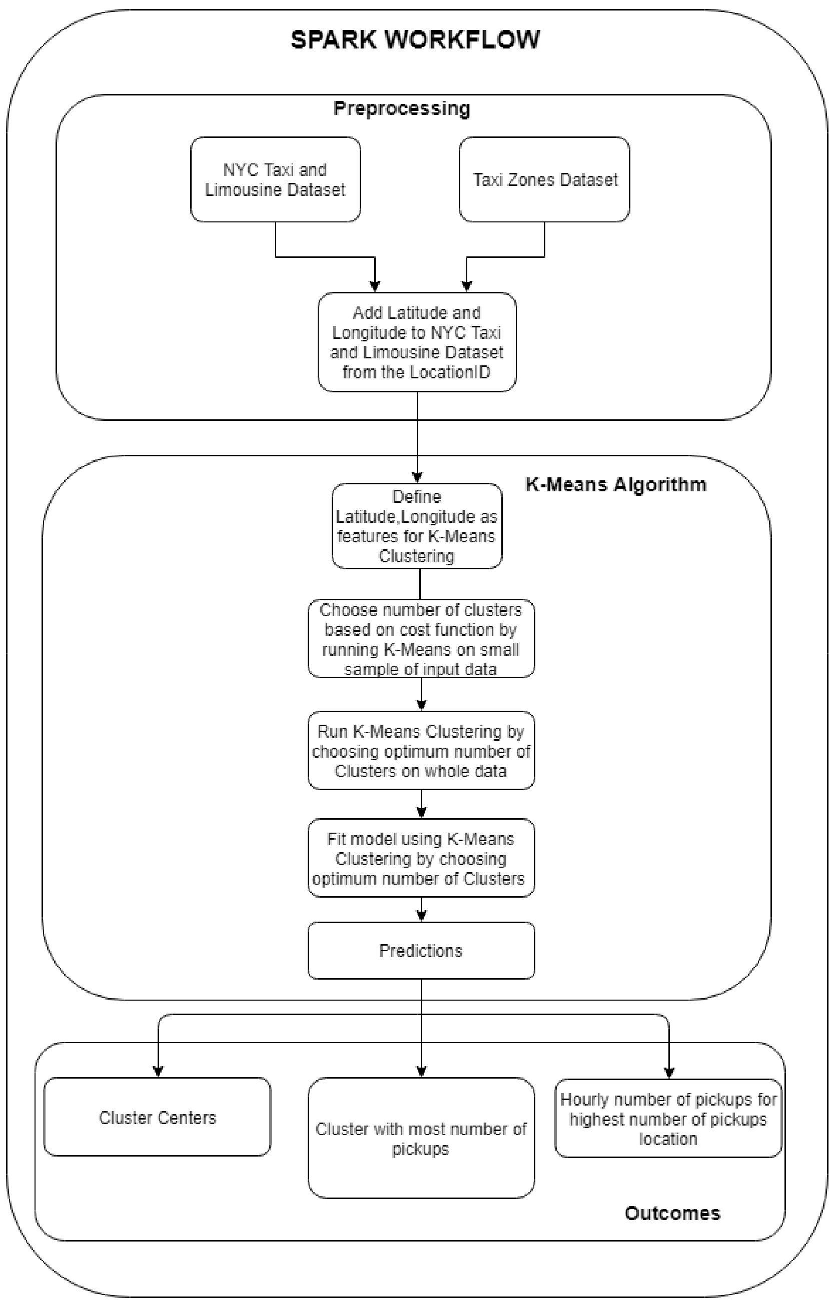
A person who wants to setup a fast food chain type restaurant where people can grab a fresh snack quickly wants to target highly concentrated mobile people locations to maximize the profits. People often grab quick snack when they travel. In this busy modern world, Taxis are being used by people mostly for commute these days. The Taxi ride dataset containing pickup time, pickup location, drop off time and drop off location can offer significant insights on the most concentrated mobile people location and the hourly concentration at a particular location. K-Means Clustering algorithm can be used to form the clusters and identify the suitable location. The main aim of our analysis is to suggest the location in order to establish a quick snack outlet.

Design:

Methodology and Pipeline architecture:

Spark is used for implementing the solution as the data is big in size and K-Means Clustering algorithm which is iterative will be used. The data frame is formed from the data by importing the data into Spark. The dataset of NYC taxi and limousine commission data does not contain Latitude and Longitude of pickup and drop off locations. So preprocessing has to be performed to get the Latitude and Longitude according to pick up location IDs. After this Latitude and Longitude are defined as the features based on which K-Means algorithm forms clusters. The optimum value of number of clusters is obtained by running K-Means algorithm on a sample of dataset for different values of number of clusters and cost function output is observed. The number of clusters which gives minimum cost function output is chosen. Then the K-Means algorithm is run on the whole dataset with this optimum number of clusters value and cluster centers are obtained. The data is then assigned to clusters according to predictions. The cluster with maximum number of pickup locations is the suggested location for establishing outlet.

Data pipeline Architecture



Spark Workflow

Dataset:

The dataset that is explored for this purpose is New York City taxi and Limousine Commission dataset which contains taxi trip records with fields capturing pick-up and drop-off dates/times, pick-up and drop-off location IDs, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The dataset is organized according to the year and month. Each month dataset is approximately 1GB in size. The taxi ride dataset is available from 2009 year onwards. The dataset can be obtained from https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

Solution:

Platform: Apache Spark configured with jupyter notebook is used to implement the solution.

Preprocessing: The preprocessing has been performed on Yellow taxi ride dataset of January, 2018 dataset of size 736 MB. The taxi ride dataset does not contain the Latitude, Longitude, location name of pickup and drop off locations. This dataset has Locations IDs. Taxi zone dataset from the same website is used to get the locations from location ID and then latitude and longitude is obtained. The Location ID in these two datasets are compared and corresponding Latitude and Longitude is added to the original NYC taxi and limousine dataset.

Analysis: The Latitude and Longitude are defined as features based on which K-Means clustering is performed. The K-Means algorithm of mllib package of Spark is used to implement the Clustering algorithm. The following are the snippets of code. The optimum number of clusters is obtained by fitting the data using K-Means algorithm with a sample of dataset and then observing the cost function. The optimum number of clusters thus determined is used to fit the whole dataset using K-Means algorithm. From the cluster centers obtained, the cluster center with most number of pickups is obtained. This cluster center is our desired location for quick snack outlet.

```
In [1]: import findspark
        findspark.init()

        import pyspark # only run after findspark.init()
        from pyspark.sql import SparkSession
        spark = SparkSession.builder.getOrCreate()

        df = spark.sql('''select 'spark' as hello ''')
        df.show()

        +-----+
        |hello|
        +-----+
        |spark|
        +-----+

In [2]: taxi_df = spark.read.csv("D:\Downloads\yellow_tripdata_2018-01.csv", inferSchema = True, header = True)
```

The above code snippet shows the dataset loading and the schema of dataset.

In [3]:
```
taxi_df.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
 |-- tpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: integer (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- PULocationID: integer (nullable = true)
 |-- DOLocationID: integer (nullable = true)
 |-- payment_type: integer (nullable = true)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
 |-- total_amount: double (nullable = true)
```

In [4]:
```
taxi_df.show()
```

```
+--------+--------------------+---------------------+---------------+-------------+----------+------------------+------------
+------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+
|VendorID|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|RatecodeID|store_and_fwd_flag|PULocationID
|DOLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|
+--------+--------------------+---------------------+---------------+-------------+----------+------------------+------------
+------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+
|       1| 2018-01-01 00:21:05|  2018-01-01 00:24:23|              1|          0.5|         1|                 N|          41
         24|           2|        4.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         5.8|
|       1| 2018-01-01 00:44:55|  2018-01-01 01:03:05|              1|          2.7|         1|                 N|         239
        140|           2|       14.0|  0.5|    0.5|       0.0|         0.0|                  0.3|        15.3|
|       1| 2018-01-01 00:08:26|  2018-01-01 00:14:21|              2|          0.8|         1|                 N|         262
        141|           1|        6.0|  0.5|    0.5|       1.0|         0.0|                  0.3|         8.3|
|       1| 2018-01-01 00:20:22|  2018-01-01 00:52:51|              1|         10.2|         1|                 N|         140
        257|           2|       33.5|  0.5|    0.5|       0.0|         0.0|                  0.3|        34.8|
|       1| 2018-01-01 00:09:18|  2018-01-01 00:27:06|              2|          2.5|         1|                 N|         246
        239|           1|       12.5|  0.5|    0.5|      2.75|         0.0|                  0.3|       16.55|
|       1| 2018-01-01 00:29:29|  2018-01-01 00:32:48|              3|          0.5|         1|                 N|         143
        143|           2|        4.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         5.8|
|       1| 2018-01-01 00:38:08|  2018-01-01 00:48:24|              2|          1.7|         1|                 N|          50
```

In [5]:
```
taxi_df.count()
```

Out[5]: 8759874

In [6]:
```
loc_df = spark.read.csv("D:\Downloads/taxi_zones.csv", inferSchema = True, header = True)
```

In [7]:
```
loc_df.show()
```

```
+------------+-----------+--------+-----------+-----------+--------------------+----------+-------------+
|   LONGITUDE|   LATITUDE|OBJECTID| Shape_Leng| Shape_Area|                zone|LocationID|      borough|
+------------+-----------+--------+-----------+-----------+--------------------+----------+-------------+
|-74.17678575|40.68951565|       1|0.116357453| 7.82307E-4|      Newark Airport|         1|          EWR|
|-73.82612577|40.62572424|       2|0.433469667| 0.00486634|         Jamaica Bay|         2|       Queens|
|-73.84947892|40.86588754|       3|0.084341106| 3.14414E-4|Allerton/Pelham G...|         3|        Bronx|
|-73.97702292|40.72415214|       4|0.043566527| 1.11872E-4|       Alphabet City|         4|    Manhattan|
|-74.18992967|40.55034012|       5| 0.09214649| 4.97957E-4|       Arden Heights|         5|Staten Island|
|-74.06777446|40.59906217|       6|0.150490543| 6.06461E-4|Arrochar/Fort Wad...|         6|Staten Island|
|-73.92149057|40.76108473|       7|0.107417171| 3.89788E-4|             Astoria|         7|       Queens|
|-73.92320241|40.77860696|       8|0.027590691|    2.66E-5|        Astoria Park|         8|       Queens|
|-73.78802025|40.75441093|       9|0.099784092| 3.38444E-4|          Auburndale|         9|       Queens|
|-73.79166546| 40.6781247|      10|0.099839479| 4.35824E-4|        Baisley Park|        10|       Queens|
|-74.01061563|40.60397771|      11|0.079211039| 2.64521E-4|          Bath Beach|        11|     Brooklyn|
|-74.01549033|40.70248841|      12|0.036661301|    4.15E-5|        Battery Park|        12|    Manhattan|
|-74.01611967|40.71161208|      13|0.050281323| 1.49359E-4|   Battery Park City|        13|    Manhattan|
|-74.03044705|40.62358428|      14|0.175213698|0.001381778|           Bay Ridge|        14|     Brooklyn|
```

The above code snippet shows the content of data frames

In [8]:
```
from pyspark.sql import *
```

In [9]:
```
new_loc_df = loc_df
```

In [11]:
```
from pyspark.sql.functions import *
```

In [12]:
```
targetDf = taxi_df.withColumn("Long",lit(0.0))
new_taxi_df = targetDf
```

In [13]:
```
for i in range(1,264):
    targetDf = new_taxi_df.withColumn("Long", when(new_taxi_df.PULocationID == i, new_loc_df.filter(new_loc_df.LocationID == i).
    new_taxi_df = targetDf
```

In [14]:
```
targetDf = new_taxi_df.withColumn("Lat",lit(0.0))
new_taxi_df = targetDf
```

```
In [15]: for i in range(1,264):
             targetDf = new_taxi_df.withColumn("Lat", when(new_taxi_df.PULocationID == i, new_loc_df.filter(new_loc_df.LocationID == i).se
             new_taxi_df = targetDf
```

```
In [16]: new_taxi_df1 = new_taxi_df.filter( (new_taxi_df.PULocationID!=264) & (new_taxi_df.PULocationID!=265))
```

```
In [17]: new_taxi_df1.show()
```

```
+--------+--------------------+---------------------+---------------+-------------+----------+------------------+------------
+------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+------------+
-----------+
|VendorID|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|RatecodeID|store_and_fwd_flag|PULocationID
|DOLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|        Long|
Lat|
+--------+--------------------+---------------------+---------------+-------------+----------+------------------+------------
+------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+------------+
-----------+
|       1| 2018-01-01 00:21:05|  2018-01-01 00:24:23|              1|          0.5|         1|                 N|          41
|          24|           2|        4.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         5.8|-73.95206533|
40.80420483|
|       1| 2018-01-01 00:44:55|  2018-01-01 01:03:05|              1|          2.7|         1|                 N|         239
|         140|           2|       14.0|  0.5|    0.5|       0.0|         0.0|                  0.3|        15.3|-73.97827334|
40.78410736|
|       1| 2018-01-01 00:08:26|  2018-01-01 00:14:21|              2|          0.8|         1|                 N|         262
|         141|           1|        6.0|  0.5|    0.5|       1.0|         0.0|                  0.3|         8.3|-73.94582982|
40.77653423|
|       1| 2018-01-01 00:20:22|  2018-01-01 00:52:51|              1|         10.2|         1|                 N|         140
```

The above code snippet shows adding Longitude and Latitude to taxi dataset

```
In [22]: from pyspark.ml.linalg import Vectors
         from pyspark.ml.feature import VectorAssembler
```

```
In [23]: df.printSchema()
```

```
root
 |-- VendorID: integer (nullable = true)
 |-- tpep_pickup_datetime: timestamp (nullable = true)
 |-- tpep_dropoff_datetime: timestamp (nullable = true)
 |-- passenger_count: integer (nullable = true)
 |-- trip_distance: double (nullable = true)
 |-- RatecodeID: integer (nullable = true)
 |-- store_and_fwd_flag: string (nullable = true)
 |-- PULocationID: integer (nullable = true)
 |-- DOLocationID: integer (nullable = true)
 |-- payment_type: integer (nullable = true)
 |-- fare_amount: double (nullable = true)
 |-- extra: double (nullable = true)
 |-- mta_tax: double (nullable = true)
 |-- tip_amount: double (nullable = true)
 |-- tolls_amount: double (nullable = true)
 |-- improvement_surcharge: double (nullable = true)
 |-- total_amount: double (nullable = true)
 |-- Long: double (nullable = false)
 |-- Lat: double (nullable = false)
```

```
In [24]: features_col = ['Long', 'Lat']
         vecAssembler = VectorAssembler(inputCols=features_col, outputCol="features")
```

```
In [25]: df_kmeans = vecAssembler.transform(df)
```

```
In [26]: df_kmeans.show()
```

```
-----------+--------------------+
|VendorID|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|RatecodeID|store_and_fwd_flag|PULocationID
|DOLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|        Long|
Lat|            features|
+--------+--------------------+---------------------+---------------+-------------+----------+------------------+------------
+------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+------------+
-----------+--------------------+
|       1| 2018-01-01 00:21:05|  2018-01-01 00:24:23|              1|          0.5|         1|                 N|          41
|          24|           2|        4.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         5.8|-73.95206533|
40.80420483|[-73.95206533,40....|
|       1| 2018-01-01 00:44:55|  2018-01-01 01:03:05|              1|          2.7|         1|                 N|         239
|         140|           2|       14.0|  0.5|    0.5|       0.0|         0.0|                  0.3|        15.3|-73.97827334|
40.78410736|[-73.97827334,40....|
```

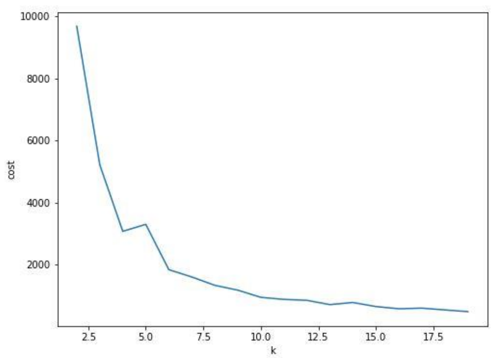
The above code snippet shows the selection of Longitude and Latitude as features

```
In [30]: from pyspark.ml.clustering import KMeans
```

```
In [33]: cost = [None]*18
         for k in range(2,20):
             kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
             model = kmeans.fit(df_kmeans.sample(False,0.1, seed=None))
             cost[k-2] = model.computeCost(df_kmeans)
```

```
In [34]: import matplotlib.pyplot as plt
         from mpl_toolkits.mplot3d import Axes3D
```

```
In [36]: fig, ax = plt.subplots(1,1, figsize =(8,6))
         ax.plot(range(2,20),cost[0:18])
         ax.set_xlabel('k')
         ax.set_ylabel('cost')

Out[36]: Text(0, 0.5, 'cost')
```



The above code snippet shows cost (within set sum of squared error) variation with number of clusters. Number of Clusters =13 is chosen as the optimum number of clusters.

```
In [37]: k = 13
         kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("features")
         model = kmeans.fit(df_kmeans)
         centers = model.clusterCenters()

         print("Cluster Centers: ")
         for center in centers:
             print(center)

         Cluster Centers:
         [-73.98368629  40.76018914]
         [-73.77875443  40.64405592]
         [-73.95231749  40.78679648]
         [-74.00541521  40.7285514 ]
         [-73.98228066  40.74377811]
         [-73.87316434  40.77394568]
         [-74.00618846  40.7025203 ]
         [-73.97669199  40.78274707]
         [-73.9822016   40.72596229]
         [-73.96473599  40.76319306]
         [-73.92964273  40.74444284]
         [-73.99715736  40.74784366]
         [-73.99609612  40.72529912]
```

```
In [107]: Longitudes = [None]*13
          Latitudes = [None]*13
          for i in range(0,13):
              Longitudes[i] = centers[i][0]
              Latitudes[i] = centers[i][1]
```

```
In [39]: transformed = model.transform(df_kmeans)
```

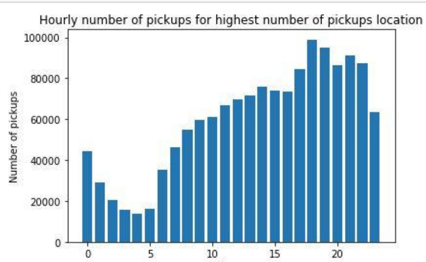
```
In [40]: transformed.show()

+--------+--------------------+---------------------+---------------+-------------+----------+------------------+------------
+------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+------------+
-----------+--------------------+----------+
|VendorID|tpep_pickup_datetime|tpep_dropoff_datetime|passenger_count|trip_distance|RatecodeID|store_and_fwd_flag|PULocationID
|DOLocationID|payment_type|fare_amount|extra|mta_tax|tip_amount|tolls_amount|improvement_surcharge|total_amount|        Long|
Lat|            features|prediction|
+--------+--------------------+---------------------+---------------+-------------+----------+------------------+------------
+------------+------------+-----------+-----+-------+----------+------------+---------------------+------------+------------+
-----------+--------------------+----------+
|       1| 2018-01-01 00:21:05|  2018-01-01 00:24:23|              1|          0.5|         1|                 N|          41
|          24|           2|        4.5|  0.5|    0.5|       0.0|         0.0|                  0.3|         5.8|-73.95206533|
40.80420483|[-73.95206533,40....|         2|
|       1| 2018-01-01 00:44:55|  2018-01-01 01:03:05|              1|          2.7|         1|                 N|         239
```

The above code snippet shows K-Means algorithm usage and getting the clusters. The data is assigned to clusters according to the prediction.

```
In [41]: nhp = transformed.select(hour(transformed.tpep_pickup_datetime).alias("hour"),transformed.prediction)

In [43]: nhp.groupby("hour","prediction").agg(count("prediction").alias("count")).orderBy(desc("count")).show()

+----+----------+-----+
|hour|prediction|count|
+----+----------+-----+
|  18|         0|98956|
|  19|         0|95224|
|  21|         0|90969|
|  18|         9|88056|
|  22|         0|87152|
|  20|         0|86437|
|  17|         0|84380|
|  19|         4|81393|
|  18|         4|81101|
|  19|         9|80177|
|  17|         9|77804|
|  14|         0|75829|
|  20|         4|74024|
|  15|         0|73961|
|  16|         0|73579|
|  14|         9|73449|
|  15|         9|73292|
|  13|         0|71625|
|  12|         9|70827|
|  12|         0|69905|
+----+----------+-----+
only showing top 20 rows

In [46]: numpickups = transformed.groupBy("prediction").count().orderBy(desc("count"))

In [47]: numpickups.show()

+----------+-------+
|prediction|  count|
+----------+-------+
|         0|1434744|
|         9|1210570|
|         4|1195725|
|         7| 970455|
|         2| 865432|
|        11| 772594|
|        12| 541666|
|         3| 451078|
|         8| 316285|
```

The above code snippet shows the cluster with highest number of pickups.

```
In [95]: x = [None]*24
         for i in range(0,24):
             x[i] = pred0_hourdf[i][0]

In [98]: y = [None]*24
         for i in range(0,24):
             y[i] = pred0_hourdf[i][2]

In [103]: fig = plt.bar(x=x, height=y, width=0.8)
          plt.title('Hourly number of pickups for highest number of pickups location')
          plt.xlabel("hour")
          plt.ylabel("Number of pickups")
          plt.show()
```




```
In [120]: import gmplot

          gmap = gmplot.GoogleMapPlotter(40.7128,-74.0060,13)
          gmap.scatter( Latitudes, Longitudes, '#FF0000', size = 120, marker = False )
          # gmap.plot(Latitudes, Longitudes,'cornflowerblue', edge_width = 2.5)
          gmap.draw( "D:\Downloads\\map2.html" )

In [121]: spark.stop()
```

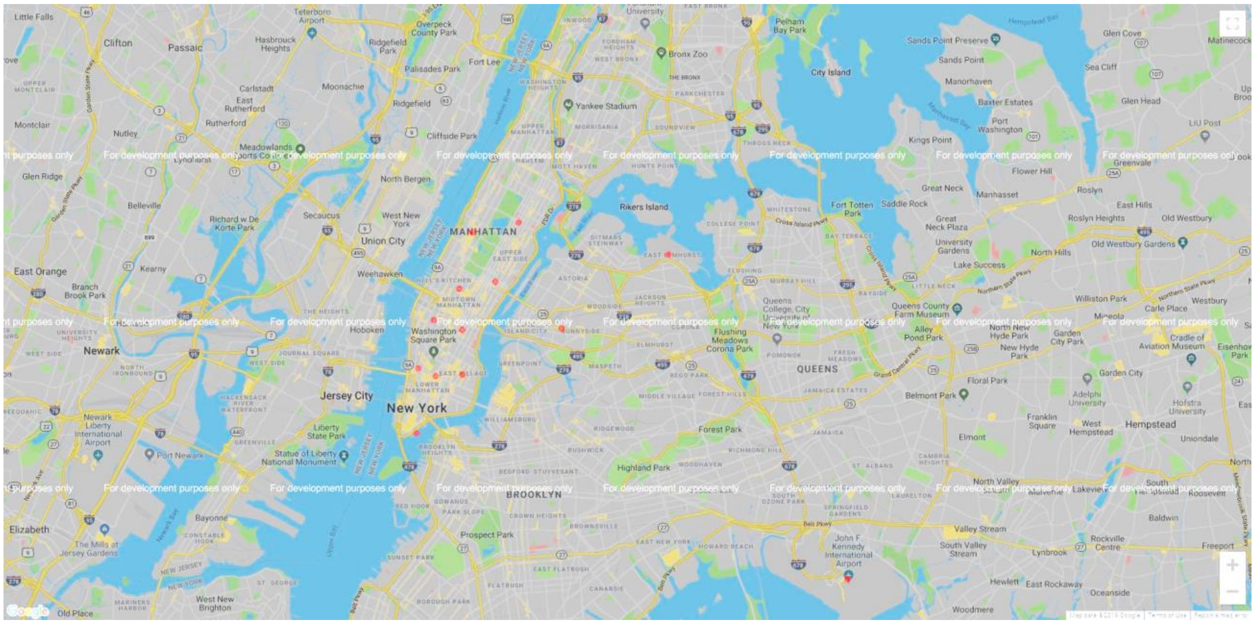
The above code snippet shows plotting outputs

Outcomes and Visualizations:

The cluster centers are

```
Cluster Centers:
[-73.98368629  40.76018914]
[-73.77875443  40.64405592]
[-73.95231749  40.78679648]
[-74.00541521  40.7285514 ]
[-73.98228066  40.74377811]
[-73.87316434  40.77394568]
[-74.00618846  40.7025203 ]
[-73.97669199  40.78274707]
[-73.9822016   40.72596229]
[-73.96473599  40.76319306]
[-73.92964273  40.74444284]
[-73.99715736  40.74784366]
[-73.99609612  40.72529912]
```

The clusters are plotted on the map below using gmplot package:

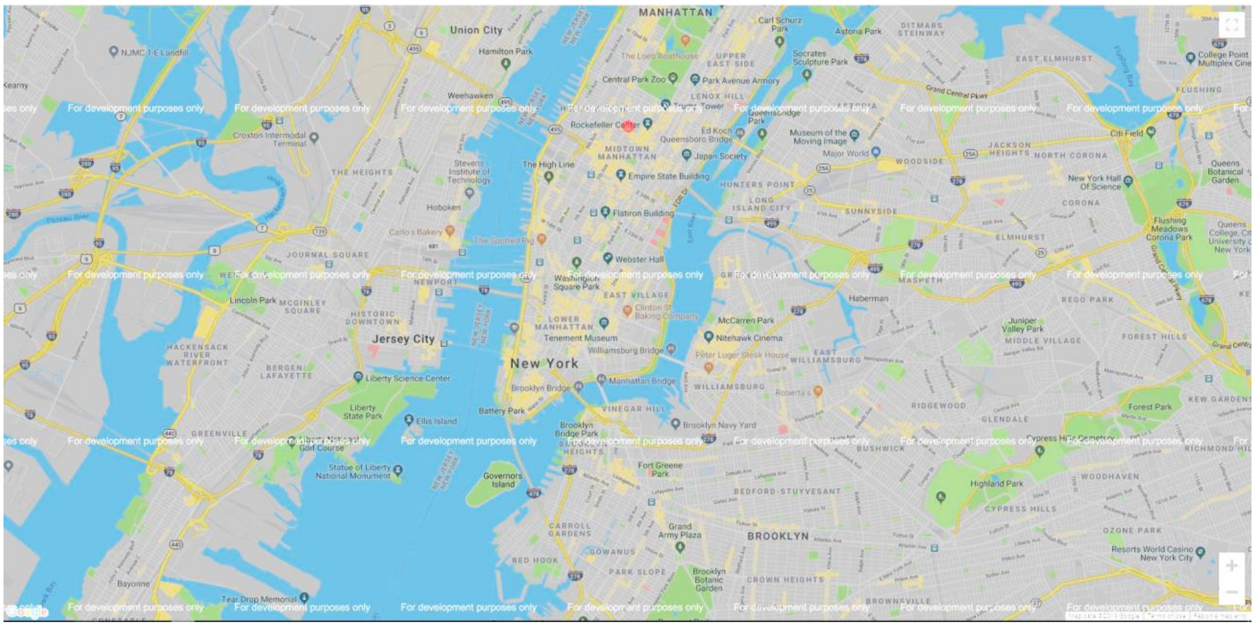


The red dots show all cluster centers.

The below table shows the number of pickups for each clusters. Prediction0 cluster which has center at (-73.98368629, 40.76018914) has the highest number of pickups.

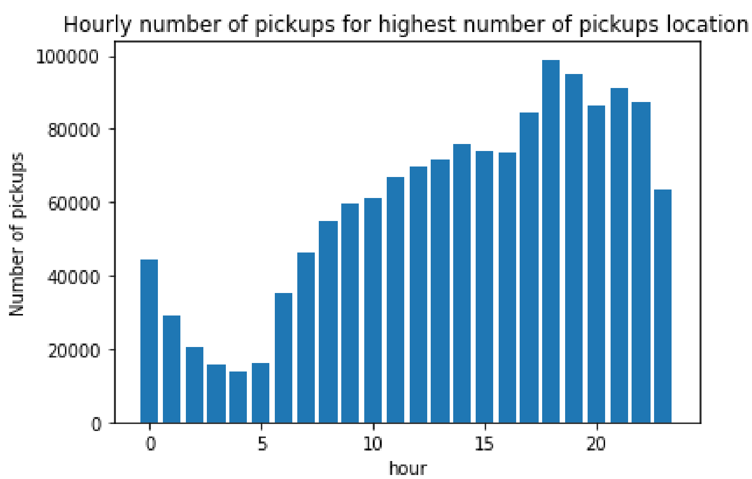
```
+----------+-------+
|prediction|  count|
+----------+-------+
|         0|1434744|
|         9|1210570|
|         4|1195725|
|         7| 970455|
|         2| 865432|
```

```
|         11| 772594|
|         12| 541666|
|          3| 451078|
|          8| 316285|
|          6| 302063|
|          5| 254746|
|          1| 210142|
|         10|  75463|
+----------+-------+
```



The red dot in the above figure shows the coordinates (-73.98368629, 40.76018914) which has the highest number of pickups. Quick snack outlet can be established at this location.

The following figure shows the hourly concentration of mobile people at this location



The above figure can be used to plan the inventory requirement.

Summary:

1. Spark has been chosen for implementing the solution, as it is useful for parallel processing of big data with iterative algorithms.
2. The taxi ride dataset has been used to obtain the location of most mobile people concentrated location.
3. Preprocessing of the dataset has been performed to find the actual location Latitude and Longitude from the Location IDs.
4. K-Means clustering has been performed on a sample of preprocessed dataset with different number of clusters to find the optimum number of clusters based on cost function (within set sum of squared errors). The optimum number of clusters is found out to be 13.
5. K-Means clustering is performed on whole dataset with 13 as the number of clusters.
6. The Cluster center with maximum number of pickups is (-73.98368629, 40.76018914). This location can be used for establishing quick snack outlet as this is the most mobile people concentrated location.
7. The hourly concentration of people at this location can be used for inventory planning.

References:

1. Apache Spark. http://spark.apache.org/, last viewed 2019.
2. Spark Programming guide: https://spark.apache.org/docs/1.2.0/programming-guide.html, last viewed 2019.
3. Spark Quickstart: https://spark.apache.org/docs/latest/quick-start.html, (interactive), last viewed 2019
4. Spark ML guide: https://spark.apache.org/docs/latest/ml-guide.html