E-Logistics Case 3

Hongyi Nie hn327
Chenyang Yan cy477

Part A

Q1
Read ‘station_information.json’ and ‘201907-citibike-tripdata.csv’, and only keep rows of dataframe where ‘start station id’ and ‘end station id’ is in the json file.

Q2.
Create a weekday list and eligible hour list, and only keep dataframe where 'start_hour' is in hours list and where 'start_weekday' is in weekdays list. Now there are only 741 unique station is in the list, then create a matrix of 19 * 741 zeros, and insert total number of bike for each hour to the list, then divide 23 for each number since there are 23 weekdays for 2019 July, so we get a 19 * 741 size matrix which represents 19 time periods and columns represent 741 unique stations, $\mu_t(i)$ = rides that are initiated at each Station i at time t.

Q3.
$p_t(i, j)$ = Pr(destination station is j | a ride leaves Station i in hour t) = Total number of a ride leaves Station i in hour t and destination station in j / Total number of a ride leaves Station i in hour t.

Start from one time period: Create an empty main_list, loop in the dataframe to check if “start station id” is in the unique station id list, and if no rides originate from Station i in hour t then just set the list with corresponding position $p(i, i) = 1$ and $p(i, j)=0$ for j not equal to i, append the main_list; if it is in the unique station id list, then change the right position with its corresponding $p(i, j)$ and append to the main_list. Finally we have a 741*741 size matrix, the rows represent i (where the initiate station is) and columns represent j (where the end station is). Repeat this process for 19 time periods and we will get 19 matrices.

Q4.
According to the following function:

$$\lambda t\,(j) \;=\; \sum_{i}^{741} \mu t(i)\; pt(i,\, j)$$

Since we already have the $\mu t(i)$ and $pt(i,\, j)$ matrix from step two and three, so we only need to find its sum product. It will give a 741 * 19 size lambda matrix

Part B

We find the key value pair for each station and its capacity in a json file, then we use the key(station_it) to find each station's capacity in the dataframe. Then for each station, net_inflow_t(i) = $\lambda t\ (i)\ -\ \mu t(i)$. Set initial_bikes = np.linspace(0, station_capacity+1, station_capacity+1), outages = np.zeros(station_capacity+1). Then for all 19 time period:
when exceed capacity: Outages = Outages + [current_bikes + net_inflow_t(i) - station_capacity]+
when drop below zero: Outages = Outages + [-(current_bikes + net_inflow_t(i))]+
Current bike will be updated by the change of initial bike.

Finally we get the updated initial bike list for each station, and only pick the argmax(outage) of initial bike number as optimal number for that station, and use these numbers to create a new list:optimal_initial_bike.

Q1.
Prediction for the optimal number of bikes in the fleet: Sum(optimal_initial_bike) = 5640.41

Q2.
173 bikes need to be moved overnight assuming ideal positioning each morning from #1 above, since the sum of current_bike according to the argmin(outgage) initial bike number is 5467.23, and we want the initial bike to be 5640.41in total.

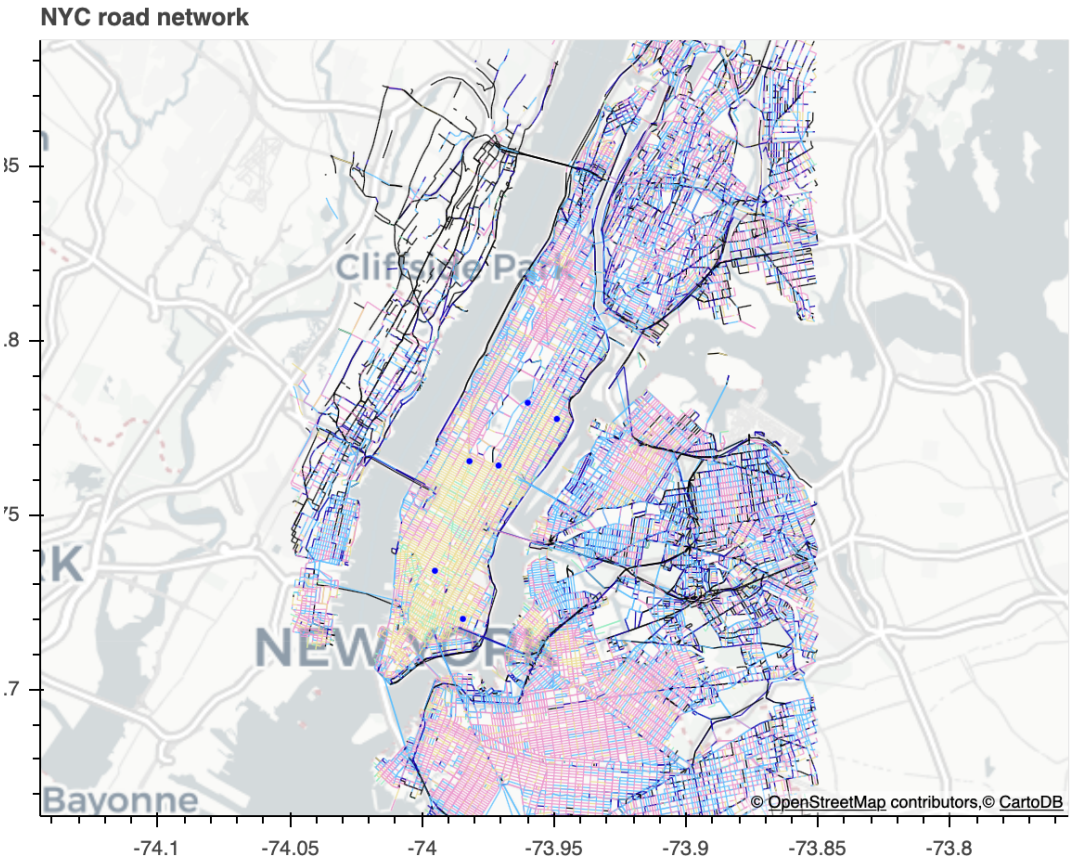
Q3.
There are 2555 outages in total over a typical week day, which is 3.35 outages per station per day. Compare it to the average number of rides taken on a typical weekday, which is 4.38 per station per day. The number of rides is a bit higher than outages.

Q4.
Similar as we did in Q3 but we only consider about the 2pm and suppose that all stations are half full. Then we get the index of adding and reducing station index is as following:
reducing one works at station_id index of [56, 148, 389, 394]
Adding one works at station_id index of [188,331]

And the corresponding position is shown in the flowing:
latlon_add = [(40.7652654, -73.98192338), (40.76350532, -73.97109243)]
latlon_reducing = [(40.73331967, -73.99510132), (40.72066442, -73.98517977), (40.78307, -73.95939), (40.778301, -73.9488134)]

We find the 6 blue points and plot it to the map, the rest 741 - 6 = 735 points will be black but we did not print it since their latitude and longitude is in different decimal place from the NYC make

nodes so we cannot find the exact points of them, but we labeled the 6 optimal points' closest value and plot them on the map.



Q5

1. The limitation of lambda. if there is no car in a station, we cannot count the number of people who wanted to use the bike in that station, so the data we collected is biased. It is very hard to avoid since there is no way to know, except the app add a section says "Collect data: check if you want to use the bike but there is no nike here" and let user contribute the data themselves, even if we can do that there is still probability that user will not report the data by themselves.

2. The limitation of features. We only consider the weekday and hour, but not location, seasonality, holiday, traffic, community (i.e schools, companies, mall .etc) These are also features that will affect the result. To avoid this, divide to four seasons and multiple areas, and import more data to add features.

3. The limitation of time. Using hour to represent time is not accurate since some bikes are rented at the beginning of an hour and some are at the end of an hour. To avoid this, we can change the time to every 10 mins instead of every hour.

4. We did not count the car lost, broken, system issues or any issues that cause the bike cannot be returned. To avoid this, we should count the total number of bikes everyday and calculate the optimal number accordingly.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import json
from ortools.linear_solver import pywraplp
```

Q1

In [3]:
```
f = open('station_information.json')
data = json.load(f)
```

In [4]:
```
station_list = []
for i in range(len(data['data']['stations'])):
    station_list.append(data['data']['stations'][i]['station_id'])
len(station_list)
```

Out[4]:
```
1422
```

In [5]:
```
july = pd.read_csv('201907-citibike-tripdata.csv')
df = july[july['start station id'].isin(station_list)]
df = df[df['end station id'].isin(station_list)]
print(july.shape)
print(df.shape)
```

```
(2181064, 15)
(1878296, 15)
```

In [97]:
```
df
```

Out[97]:

	tripduration	starttime	stoptime	start station id	start station name	start station latitude	start station longitude
1	267	2019-07-01 00:00:05.1780	2019-07-01 00:04:32.4500	3143.0	5 Ave & E 78 St	40.776321	-73.964274
2	2201	2019-07-01 00:00:05.2130	2019-07-01 00:36:46.7490	317.0	E 6 St & Avenue B	40.724537	-73.981854
3	1660	2019-07-01 00:00:08.6010	2019-07-01 00:27:48.8050	249.0	Harrison St & Hudson St	40.718710	-74.009001
4	109	2019-07-01 00:00:12.1580	2019-07-01 00:02:01.5670	3552.0	W 113 St & Broadway	40.805973	-73.964928
5	106	2019-07-01 00:00:12.6680	2019-07-01 00:01:59.6060	3593.0	31 St & Broadway	40.761584	-73.925921
...	...	...	...	...	...	...	...
2181059	644	2019-07-31 23:59:45.2950	2019-08-01 00:10:29.8640	274.0	Lafayette Ave & Fort Greene Pl	40.686919	-73.976682
2181060	589	2019-07-31 23:59:52.7170	2019-08-01 00:09:42.0870	3285.0	W 87 St & Amsterdam Ave	40.788390	-73.974700
2181061	1302	2019-07-31 23:59:52.8280	2019-08-01 00:21:35.1090	331.0	Pike St & Monroe St	40.711731	-73.991930
2181062	353	2019-07-31 23:59:57.1130	2019-08-01 00:05:50.1350	3255.0	8 Ave & W 31 St	40.750585	-73.994685
2181063	274	2019-07-31 23:59:57.4870	2019-08-01 00:04:32.2070	3459.0	E 53 St & 3 Ave	40.757632	-73.969306

1878296 rows × 19 columns

Q2

In [98]:
```
df['start_hour'] = pd.to_datetime(df['starttime']).dt.hour
df['start_weekday'] = pd.to_datetime(df['starttime']).dt.dayofweek
df['end_hour'] = pd.to_datetime(df['stoptime']).dt.hour
df['end_weekday'] = pd.to_datetime(df['stoptime']).dt.dayofweek
new_df = df[['start station id','end station id','start_hour','start_w
```

In [100]:
```
hours = [i for i in range(5,24)]
weekdays = [i for i in range(5)]
limit_df = new_df[new_df['start_hour'].isin(hours)]
limit_df = limit_df[limit_df['start_weekday'].isin(weekdays)]
limit_df.shape[0]
# limit_df.reset_index(inplace = True)
limit_df
```

Out[100]:

	start station id	end station id	start_hour	start_weekday	end_hour	end_weekday	start station latitude	st long
1511	3314.0	3538.0	5	0	5	0	40.793770	-73.9
1515	3118.0	3127.0	5	0	5	0	40.735550	-73.9
1516	3351.0	507.0	5	0	5	0	40.786995	-73.9
1518	402.0	336.0	5	0	5	0	40.740343	-73.9
1519	3459.0	164.0	5	0	5	0	40.757632	-73.9
...	...	...	...	...	...	...	...	
2181059	274.0	3409.0	23	2	0	3	40.686919	-73.9
2181060	3285.0	3356.0	23	2	0	3	40.788390	-73.9
2181061	331.0	309.0	23	2	0	3	40.711731	-73.9
2181062	3255.0	482.0	23	2	0	3	40.750585	-73.9
2181063	3459.0	3134.0	23	2	0	3	40.757632	-73.9

1417865 rows × 8 columns

In [8]:
```
start_end = limit_df['start station id'].append(limit_df['end station
station_ids = sorted(start_end.unique())
len(station_ids)
```

Out[8]: 741

In [9]:
```
# total_bike = np.zeros((19,len(station_ids)))
# for i in range(5,24):
#     hour_df = limit_df[limit_df['start_hour'] == i]
#     for j in range(len(station_ids)):
#         for z in range(hour_df.shape[0]):
#             if hour_df.iloc[z,0] == station_ids[j]:
#                 total_bike[i][j] += 1
# #there are 23 weekdays
# total_bike = total_bike/23
# total_bike
```

In [10]:
```
#miu
total_bike = np.zeros((19,len(station_ids)))
for i in range(5,24):
    hour_df = limit_df[limit_df['start_hour'] == i]
    for j in range(hour_df.shape[0]):
            if hour_df.iloc[j,0] in station_ids:
                total_bike[i-5][station_ids.index(hour_df.iloc[j,0])]
#there are 23 weekdays
total_bike = total_bike/23
total_bike
```

Out[10]:
```
array([[ 1.2173913 ,  0.30434783,  0.30434783, ...,  0.04347826,
         0.04347826,  0.04347826],
       [ 3.60869565,  1.08695652,  1.08695652, ...,  0.04347826,
         0.39130435,  0.        ],
       [10.26086957,  3.30434783,  2.39130435, ...,  0.73913043,
         0.47826087,  0.26086957],
       ...,
       [ 4.08695652,  3.08695652,  1.95652174, ...,  0.34782609,
         0.34782609,  0.69565217],
       [ 3.13043478,  2.34782609,  2.26086957, ...,  0.39130435,
         0.2173913 ,  0.91304348],
       [ 2.30434783,  1.47826087,  0.82608696, ...,  0.17391304,
         0.13043478,  0.7826087 ]])
```

In [72]:
```
np.mean(total_bike)
```

Out[72]:
```
4.378599641155344
```

Q3

In [11]:
```
#p(i,j)
full_matrix = []
for i in range(5,24):
    trans_matrix = []
    hour_df = limit_df[limit_df['start_hour'] == i]
    for start in station_ids:
        if start in hour_df['start station id'].tolist():
            start_df = hour_df[hour_df['start station id'] == start]
            total = start_df.shape[0]
            l=[]
            for end in station_ids:
                end_df = start_df[start_df['end station id'] == end]
                l.append(end_df.shape[0]/total)
            trans_matrix.append(l)
        else:
            l = [0 for j in range(len(station_ids))]
            l[station_ids.index(start)] = 1
            trans_matrix.append(l)
    full_matrix.append(trans_matrix)
    print(trans_matrix)
print(full_matrix)
```

```
[[0.03571428571428571, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03
571428571428571, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.03571428571428571, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.
0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.035714285714
28571, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07142857142857142, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.14285714285714285, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03
571428571428571, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03571428571428571, 0
.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03571428571428571,
```

Q4

In [12]:
```
#lambda(i)
lambda_list = []
for i in range(len(station_ids)):
    sum_list = []
    for j in range(19):
        product = []
        for z in range(len(station_ids)):
            product.append(total_bike[j][z]*full_matrix[j][z][i])
        sum_list.append(sum(product))
    lambda_list.append(sum_list)
lambda_list
```

```
 19.739130434782602,
 13.95652173913043,
 9.1304347826087,
 6.347826086956531,
 3.0434782608695663],
[0.043478260869565216,
 0.5652173913043478,
 1.6521739130434778,
 3.1304347826086967,
 2.130434782608695,
 3.043478260869565,
 2.6521739130434785,
 3.347826086956523,
 3.0869565217391317,
 3.8260869565217406,
 3.2608695652173934,
 4.0000000000000036,
 6.565217391304353,
 10.04347826086957,
 6.521739130434787,
```

Part 2

In [81]:
```
station_list = []
for i in range(len(data['data']['stations'])):
    station_list.append(data['data']['stations'][i]['station_id'])
print(len(station_list))
# x(0) = decision variable
capacity_list = []
for i in range(len(data['data']['stations'])):
    capacity_list.append(data['data']['stations'][i]['capacity'])
print(len(station_list))
```

```
1422
1422
```

In [82]:
```
res = {}
for key in station_list:
    for value in capacity_list:
        res[key] = value
        capacity_list.remove(value)
        break
res
```

Out[82]:
```
{'72': 55,
 '79': 33,
 '82': 27,
 '83': 62,
 '116': 50,
 '119': 53,
 '120': 19,
 '127': 31,
 '128': 56,
 '143': 24,
 '144': 58,
 '146': 39,
 '150': 56,
 '151': 33,
 '152': 49,
 '153': 63,
 '157': 23,
 '161': 35,
 '164': 47,
```

In [83]:
```
# solver = pywraplp.Solver.CreateSolver('Case3', 'CBC')
# #.Solver.GLOP_LINEAR_PROGRAMMING or CLP_LINEAR_PROGRAMMING
# # or CBC_MIXED_INTEGER_PROGRAMMING

# objective=solver.Objective()
# objective.SetMinimization()

# #x_0 variable
# x = [None] * 19
# y = [None] * 19

# infinity = solver.infinity()
# for i in range(19):
#     x[i] = solver.IntVar(0, infinity, 'x[{}]'.format(i))
#     y[i] = solver.IntVar(0, infinity, 'y[{}]'.format(i))

# #constrains
# outage = []
# for i in range(len(station_ids)):
#     y_list = []
#     for j in range(18):
#         lam = lambda_list[i][j]
#         miu = total_bike[j][i]
#         d = res[str(station_ids[i])]
#         solver.Add(x[i+1] = x[i] + lam - miu)
#         solver.Add(y[i+1] = max(0,x[i] + lam - miu - d) + max(0, -(x
#         y_list.append(y[j+1])
#     outage.append(sum(y_list))

# solver.Minimize(sum(outage))
```

In [84]:
```
net_inflow = [[0 for i in range(19)] for j in range(len(station_ids))]
for i in range(len(station_ids)):
    for j in range(19):
        net_inflow[i][j] = lambda_list[i][j] - total_bike[j][i]
```

In [85]:
```
optimal_initial_bike = []
optimal_current_bike = []
optimal_outages = []
for i in range(len(station_ids)):
    station_capacity = res[str(int(station_ids[i]))]
    initial_bikes = np.linspace(0, station_capacity+1, station_capacit
    outages = np.zeros(station_capacity+1) # Contains number of outage
    current_bikes = np.copy(initial_bikes)
    for j in range(19):
        outages += np.maximum(0, current_bikes + net_inflow[i][j] - st
        outages += np.maximum(0, -(current_bikes + net_inflow[i][j]))
        current_bikes += net_inflow[i][j]
        current_bikes = np.maximum(current_bikes, 0) # Now pin the inv
        current_bikes = np.minimum(current_bikes, station_capacity)
    initial = initial_bikes[np.argmin(outages)]
    optimal_initial_bike.append(initial)
    optimal_current_bike.append(current_bikes[initial_bikes.tolist().i
    optimal_outages.append(min(outages))
print(sum(optimal_initial_bike))
print(sum(optimal_current_bike))
print(sum(optimal_outages))
print(np.mean(optimal_outages))
```

```
5640.410118715889
5467.232878831566
2555.8623323822667
3.4492069262918585
```

In [93]:
```
current_outages = []
for i in range(len(station_ids)):
    station_capacity = res[str(int(station_ids[i]))]
    initial_bikes = station_capacity/2 # initial bikes can be 0, 1, ..
    outages = np.zeros(1) # Contains number of outages as a function o
    current_bikes = np.copy(initial_bikes)
    for j in range(9,10):
        outages += np.maximum(0, current_bikes + net_inflow[i][j] - st
        outages += np.maximum(0, -(current_bikes + net_inflow[i][j]))
        current_bikes += net_inflow[i][j]
        current_bikes = np.maximum(current_bikes, 0) # Now pin the inv
        current_bikes = np.minimum(current_bikes, station_capacity)
    current_outages.append(outages)
# adding one
adding_one_outages = []
for i in range(len(station_ids)):
    station_capacity = res[str(int(station_ids[i]))]
    initial_bikes = station_capacity/2 + 1 # initial bikes can be 0, 1
    outages = np.zeros(1) # Contains number of outages as a function o
    current_bikes = np.copy(initial_bikes)
    for j in range(9,10):
        outages += np.maximum(0, current_bikes + net_inflow[i][j] - st
        outages += np.maximum(0, -(current_bikes + net_inflow[i][j]))
        current_bikes += net_inflow[i][j]
        current_bikes = np.maximum(current_bikes, 0) # Now pin the inv
        current_bikes = np.minimum(current_bikes, station_capacity)
    adding_one_outages.append(outages)
reducing_one_outages = []
for i in range(len(station_ids)):
    station_capacity = res[str(int(station_ids[i]))]
    initial_bikes = station_capacity/2 - 1 # initial bikes can be 0, 1
    outages = np.zeros(1) # Contains number of outages as a function o
    current_bikes = np.copy(initial_bikes)
    for j in range(9,10):
        outages += np.maximum(0, current_bikes + net_inflow[i][j] - st
        outages += np.maximum(0, -(current_bikes + net_inflow[i][j]))
        current_bikes += net_inflow[i][j]
        current_bikes = np.maximum(current_bikes, 0) # Now pin the inv
        current_bikes = np.minimum(current_bikes, station_capacity)
    reducing_one_outages.append(outages)
```

In [95]:
```
adding = []
reducting = []
for i in range(len(current_outages)):
    if current_outages[i] - adding_one_outages[i] > 0:
        adding.append(i)
        print("adding one works at " + str(i))
    if current_outages[i] - reducing_one_outages[i] > 0:
        print("reducing one works at " + str(i))
        reducting.append(i)
```

```
reducing one works at 56
reducing one works at 148
adding one works at 188
adding one works at 331
reducing one works at 389
reducing one works at 394
```

In [103]:
```
reducting
```

Out[103]:
```
[56, 148, 389, 394]
```

In [107]:
```
latlon_add = []
for i in adding:
    position_df = limit_df[limit_df['start station id'] == station_ids
    lat = position_df.iloc[0,6]
    lon = position_df.iloc[0,7]
    latlon_add.append((lat,lon))
print(latlon_add)

latlon_reducting = []
for i in reducing:
    position_df = limit_df[limit_df['start station id'] == station_ids
    lat = position_df.iloc[0,6]
    lon = position_df.iloc[0,7]
    latlon_reducting.append((lat,lon))
print(latlon_reducing)
```

```
[(40.7652654, -73.98192338), (40.76350532, -73.97109243)]
[(40.73331967, -73.99510132), (40.72066442, -73.98517977), (40.78307,
-73.95939), (40.778301, -73.9488134)]
```

In [22]:
```
lon_add = [(40.7652654, -73.98192338), (40.76350532, -73.97109243)]
lon_reducing = [(40.73331967, -73.99510132), (40.72066442, -73.9851797
```

In [34]:
```
dfn.loc[(dfn['lon'] >= latlon_reducing[3][1]-0.001) & (dfn['lon'] <= l
```

Out[34]:

	name	lon	lat	x	y
566	42454196	-73.948881	40.777552	-8.231952e+06	4.979586e+06
1092	42445748	-73.948425	40.778177	-8.231901e+06	4.979678e+06
2111	42436551	-73.947967	40.778809	-8.231850e+06	4.979771e+06

In [39]:
```
target1 = [42428328,42445953,42449067,42432075,42454189,42454196]
```

In [59]:
```
# x = [[0 for i in range(19)] for j in range(len(station_ids))]
# y = [[0 for i in range(19)] for j in range(len(station_ids))]
# variable_1 = [[0 for i in range(19)] for j in range(len(station_ids)
# variable_2 = [[0 for i in range(19)] for j in range(len(station_ids)
# indicator1 = [[0 for i in range(19)] for j in range(len(station_ids)
# indicator2 = [[0 for i in range(19)] for j in range(len(station_ids)
# infinity = solver.infinity()
# for i in range(len(x)): #741
#     for j in range(len(x[0])): #19
#         x[i][j] = solver.IntVar(0, solver.infinity(), "x_"+str(i)+",
#         y[i][j] = solver.IntVar(0, solver.infinity(), "x_"+str(i)+",
#         indicator1[i][j] = solver.IntVar(0, 1, "indicator1_"+str(i)+
#         indicator2[i][j] = solver.IntVar(0, 1, "indicator2_"+str(i)+
#         variable_1[i][j] = solver.NumVar(0, solver.infinity(), "vari
#         variable_2[i][j] = solver.NumVar(0, solver.infinity(), "vari

# #constrains
# outage = []
# for i in range(len(station_ids)):
#     y_list = []
#     for j in range(18):
#         lam = lambda_list[i][j]
#         miu = total_bike[j][i]
#         d = res[str(int(station_ids[i]))]
#         solver.Add(x[i][j+1] == x[i][j] + lam - miu)

#         solver.Add(variable_1[i][j] <= x[i][j] + lam - miu - d + (1
#         solver.Add(variable_1[i][j] >= x[i][j] + lam - miu - d - (1
#         solver.Add(variable_1[i][j] <= indicator1[i][j] * 10000000)
#         solver.Add(x[i][j] + lam - miu - d <= indicator1[i][j] * 100

#         solver.Add(variable_2[i][j] <= (-(x[i][j] + lam - miu)) + (1
#         solver.Add(variable_2[i][j] >= (-(x[i][j] + lam - miu)) - (1
#         solver.Add(variable_2[i][j] <= indicator2[i][j] * 10000000)
#         solver.Add((-(x[i][j] + lam - miu)) <= indicator2[i][j] * 10

#         solver.Add(y[i][j+1] == variable_1[i][j] + variable_2[i][j])
#         y_list.append(y[i][j+1])
#     outage.append(sum(y_list))
# solver.Minimize(sum(outage))
```

In [ ]:
```
# #find the optimal solution by solver
# status = solver.Solve()
# if status == solver.OPTIMAL:
#     print("Solver reaches the optimal solution with objective functi

# else:
#     print('Solver cannot find optimal solution')
```

In [ ]:
```

```