In [6]:
```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import pandas as pd

from sklearn import datasets
from sklearn.model_selection import train_test_split

import datetime
from dateutil.parser import parse

%matplotlib inline
```

BayWheel Station Optimization

Niva Alina Ran

This project looks at the BayWheel bikesharing dataset (source: [https://www.lyft.com/bikes/bay-wheels/system-data (https://www.lyft.com/bikes/bay-wheels/system-data)](https://www.lyft.com/bikes/bay-wheels/system-data)). The idea of this project is to use predictive modeling and optimization to identify optimal placement of bike docking stations in the San Francisco Metro Area. The company BayWheel (now Lyft) has been providing bike sharing services around San Francisco and the Bay Area, and has also been publishing the data they collect about the usage of their stations on a monthly basis.

The first part of this notebook uses monthly usage data and uses it to construct a predictive model of estimated bike demand across San Francisco, indexed by geo coordinates. Next, using the demand model, we estimate the total commuting costs for potential bike users across SF. Finally, we proceed to optimize for the location of potential future bike stations with the objective of minimizing the city-wide total commuting time. In particular, we can then re-optimize the best locations for bike sharing stations, accounting for actual bike needs.

In [7]:
```
#Load data
df_raw = pd.read_csv("201910-baywheels-tripdata.csv")
print(df_raw.columns)
df_raw.head(2)
```

```
Index(['duration_sec', 'start_time', 'end_time', 'start_station_id',
       'start_station_name', 'start_station_latitude',
       'start_station_longitude', 'end_station_id', 'end_station_nam
e',
       'end_station_latitude', 'end_station_longitude', 'bike_id', '
user_type',
       'member_birth_year', 'member_gender', 'bike_share_for_all_tri
p'],
      dtype='object')
```

Out[7]:

	duration_sec	start_time	end_time	start_station_id	start_station_name	start_station_
0	62337	2019-10-31 16:25:01.5970	2019-11-01 09:43:59.0290	148	Horton St at 40th St	37
1	72610	2019-10-31 13:04:11.1950	2019-11-01 09:14:21.8050	376	Illinois St at 20th St	37

In [8]:
```
## Add hour of day and day of week features
dt_vec = df_raw['start_time'].apply(lambda x: parse(x))
hod_col = dt_vec.apply(lambda x: x.hour)
dow_col = dt_vec.apply(lambda x: x.weekday())

df_raw["start_hour_of_day"] = hod_col
df_raw["start_day_of_week"] = dow_col
```

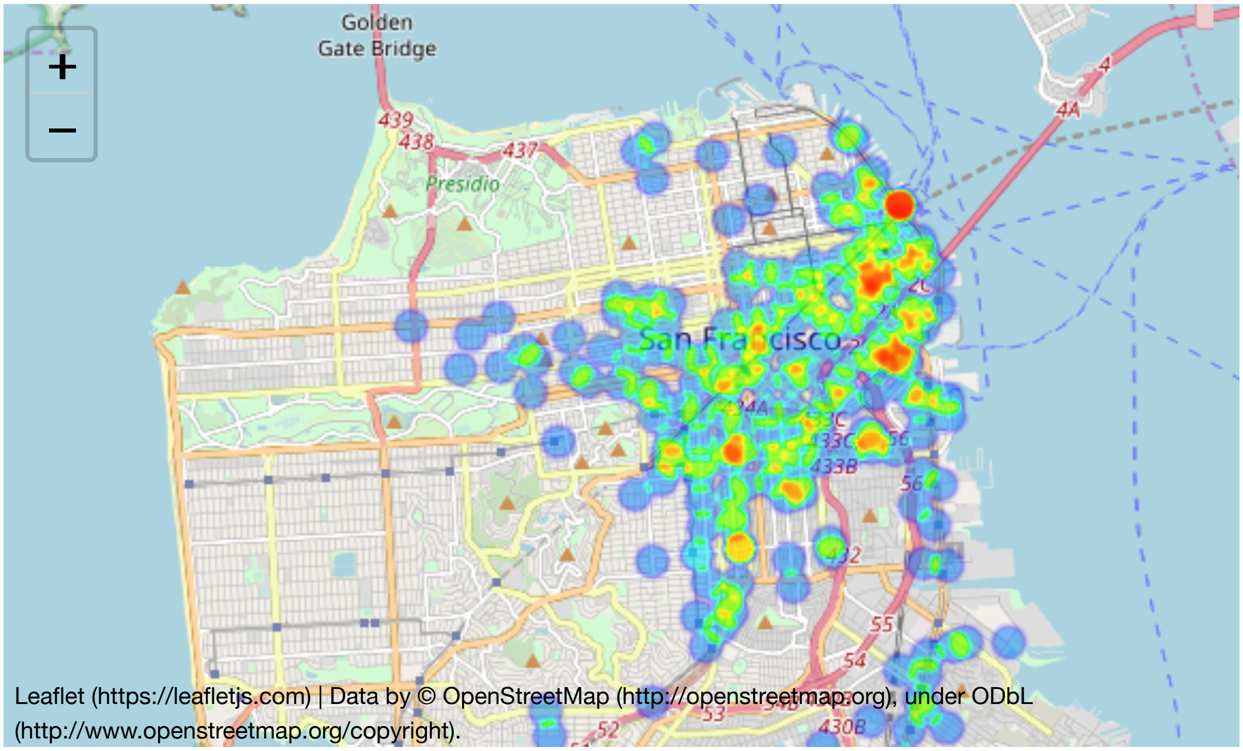
In [ ]:

Exploratory Data Analysis

In [9]:
```python
### Create new data set, grouped by station
df=df_raw
df_stat = pd.DataFrame([])
station_id_list = np.sort(df["start_station_id"].unique())
df_stat["station_id"] = station_id_list

#scan through original data set to find the matching start station ID, and record start station longitude, latitude
df_stat["start_station_latitude"]  = df_stat["station_id"].apply(lambda x: float(df_raw[df_raw["start_station_id"] == x].head(1)["start_station_latitude"]))
df_stat["start_station_longitude"] = df_stat["station_id"].apply(lambda x: float(df_raw[df_raw["start_station_id"] == x].head(1)["start_station_longitude"]))

#scan through original data set to count nubmer of occurrences of start station ID
df_stat["start_trip_count"] = df_stat["station_id"].apply(lambda x: len(df_raw[df_raw["start_station_id"] == x]))
```

In [10]:
```python
### Visualize bike usage by starting station
from folium import plugins #python library that creates interactive maps using Leaflet (Java)
import folium as folium
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

m = folium.Map([37.7756925, -122.4204695], zoom_start=13)
# convert to (n, 2) nd-array format for heatmap
station_data = df_stat[['start_station_latitude', 'start_station_longitude', 'start_trip_count']].values

## Convert startig trip counts to quantiles for visualization
d = station_data[:,2]
qrange = np.arange(0,100,1.25)
quantiles = np.percentile(d, qrange)
station_data[:,2] = np.searchsorted(quantiles, d)/len(qrange)

# Overlay station ID demand with heat map markers
m.add_child(plugins.HeatMap(station_data, max_zoom = 13, max_val=1, radius = 8, blur=2))
```

Out[10]:



In [ ]:

```

```

Learning a bike demand distribution map using K-nearest neighbors

In [11]:

```
#use local interpolation, weighted by distance of points, to predict b
ike usage across the city with K-nearest neighbor regression
from sklearn.neighbors import KNeighborsRegressor

station_data = df_stat[['start_station_latitude', 'start_station_longi
tude', 'start_trip_count']].values
X = station_data[:,[0, 1]] #X = latitude and longitude of station
y = station_data[:,2] #Y = station trip count
neigh = KNeighborsRegressor(n_neighbors=4, weights = 'distance')

#Train the model with real data
neigh.fit(X, y)
```

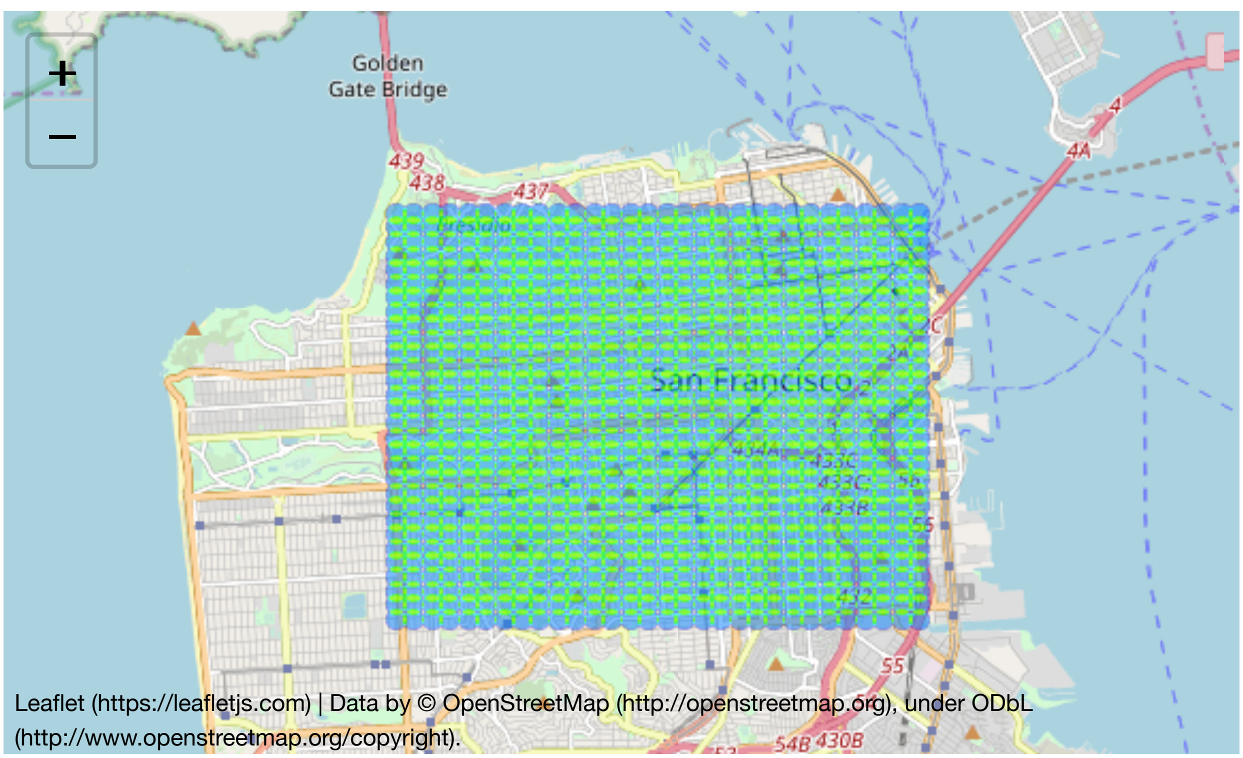
Out[11]:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowsk
i',
                    metric_params=None, n_jobs=None, n_neighbors=4,
p=2,
                    weights='distance')
```

In [12]:

```
## Define grid coordinate range
coor_ll = np.array([37.748566, -122.477034]) #lower left coordinate
coor_ur = np.array([37.801819, -122.389905]) #upper right coordiante
grid_size = 30 #number of points on each side of the grid

grid_lat, grid_long = np.mgrid[coor_ll[0]:coor_ur[0]:(coor_ur[0]-coor_
ll[0])/grid_size, coor_ll[1]:coor_ur[1]:(coor_ur[1]-coor_ll[1])/grid_s
ize]  #mgrid: returns a dense mesh-grid
grid_lat = np.ndarray.flatten(grid_lat)
grid_long = np.ndarray.flatten(grid_long)

# Predicting Demand
X = np.stack((grid_lat, grid_long), axis= -1)
y_pred = np.transpose(neigh.predict(X))
grid_data = np.stack((grid_lat, grid_long, y_pred), axis=-1)

## Convert startig trip count to quantiles for visualization
d = grid_data[:,2]
qrange = np.arange(0,100,1.25)
quantiles = np.percentile(d, qrange)
grid_data[:,2] = np.searchsorted(quantiles, d)/len(qrange)

# Visualization with map
cor_ref_start = (coor_ll+coor_ur)/2
m = folium.Map(cor_ref_start, zoom_start=13)

# plot heatmap
m.add_child(plugins.HeatMap(grid_data, max_zoom = 13, max_val=1, radiu
s = 5, blur=1))
m
```

Out[12]:



Part 2: Station Optimization using Evolutionary Strategy

In [13]:

```
#Define distance measure between two points
def manhattan_distance(a,b):
    return np.sum(np.absolute(a-b))

def geo_vec_to_tab(geo_vec):
    n_stat = int(len(geo_vec)/2)
    return np.stack((geo_vec[0:n_stat],geo_vec[n_stat:2*n_stat]), axis
= -1)

# primary data structure for SGD optimization
class bay_wheel_station_opt:
    def __init__(self, grid_data):
        self.geo_coords = grid_data["geo_coords"]
        self.demand = grid_data["demand"]

    def eval(self, stat_geo_list_vec):
        # for each unit of demand, let cost be the Manhattan distance
to the nearest station
        # station geo list is a vector [Lat...., Long....]
        tot_cost = 0
        n_stat = int(len(stat_geo_list_vec)/2)
        stat_geo_list_table = np.stack((stat_geo_list_vec[0:n_stat],st
at_geo_list_vec[n_stat:2*n_stat]), axis= -1)

        for n in range(len(self.demand)):
            x = self.geo_coords[n,]
            dis_vec = np.zeros(n_stat)
            for i in range(len(dis_vec)):
                dis_vec[i] = manhattan_distance(x,stat_geo_list_table[
i,])
            tot_cost = tot_cost+np.min(dis_vec)*self.demand[n]
        return tot_cost
```

In [14]:

```
## Stochastic gradient descent using Evolutionary Strategy

def es_optimize(funct, x_init, es_coef):
    #funct: function class object
    #x_init: array, initial solution
    #es_coef: optimization coefficients
        # "n": number of steps
        # "m": nubmer of mutations
        # "alpha" :  step size parameter
        # "sigma" : search distribution variance

    n = es_coef["n"]
    m = es_coef["m"]
    alpha = es_coef["alpha"]
```

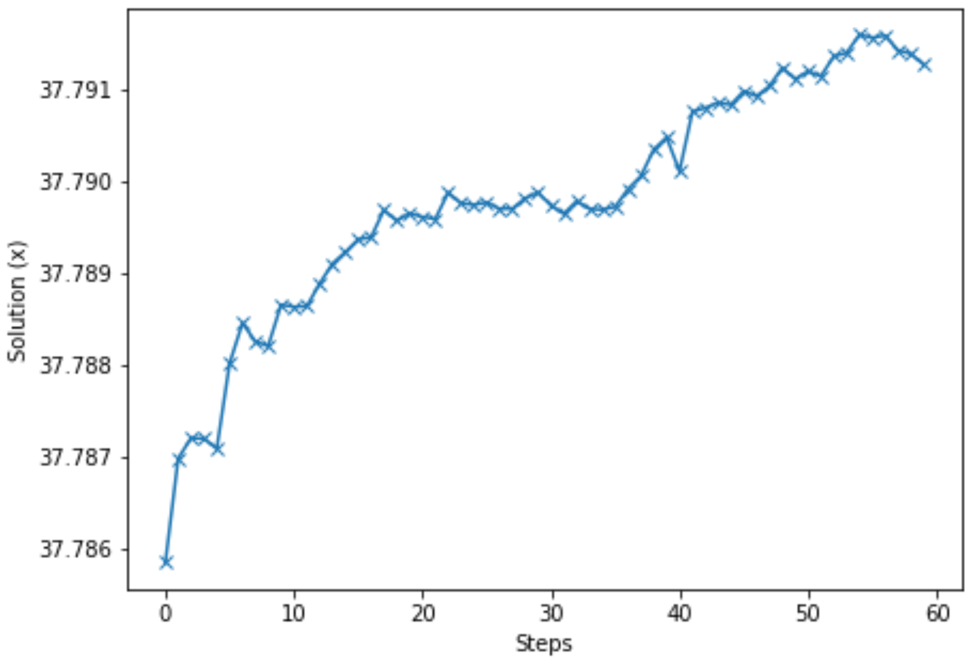
```python
    sigma = es_coef["sigma"]
    xlim_u = es_coef["xlim_u"]
    xlim_l = es_coef["xlim_l"]

    p = len(x_init) #dimension of the solution

    ## Initialization
    x = np.zeros([n, p])
    y = np.zeros([n, p])
    y_muta = np.zeros(m)

    x[0,] = x_init
    y[0] =  funct.eval(x[0,])

    for i in range(n-1):
        c_sum = np.zeros(p)
        d_muta = np.random.normal(0, sigma, [m,p]) #Generate mutations

        #Estimate gradient using weighted mutations
        for j in range(m):
            y_muta[j] = funct.eval(x[i,]+d_muta[j,])
            c_sum = c_sum + (y_muta[j]-y[i])*d_muta[j,]

        #Update solutions using estimated gradient
        x_now = x[i,]- c_sum*alpha/(sigma**2*m)

        x_now = np.minimum(xlim_u, x_now)
        x_now = np.maximum(xlim_l, x_now)

        x[i+1,] = x_now
        y[i+1] = funct.eval(x[i+1,])

    results = {
        "x_vec": x,
        "y_vec": y,
        "final_x": x[n-1,]
    }

    return results
```

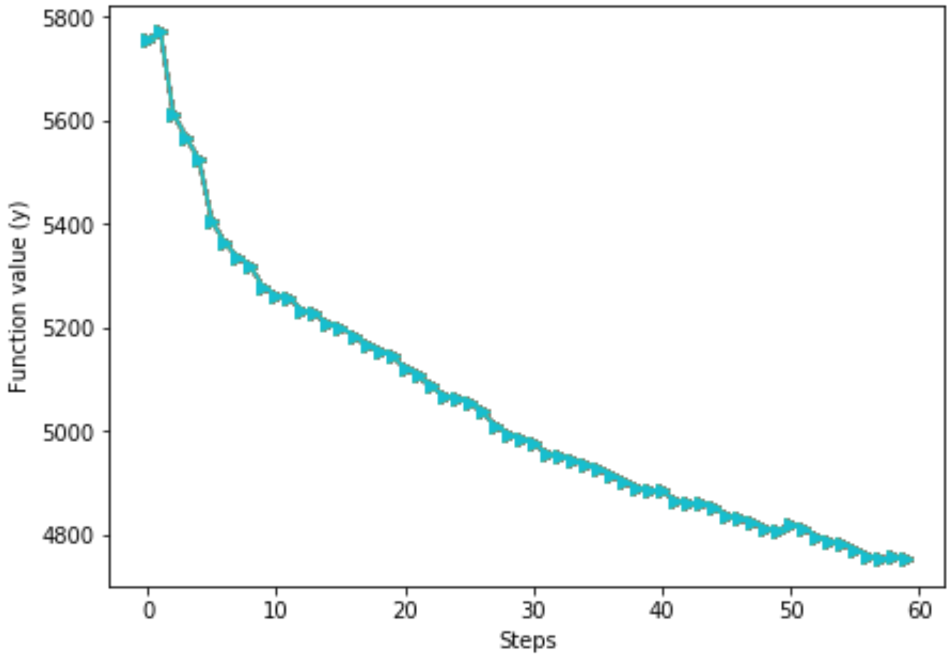
In [15]:

```
n_station = 20

xlim_u = np.concatenate((np.ones(n_station)*coor_ur[0], np.ones(n_stat
ion)*coor_ur[1]))

xlim_l = np.concatenate((np.ones(n_station)*coor_ll[0], np.ones(n_stat
ion)*coor_ll[1]))

es_coef = {
"n": 60, # number of steps
"m": 7, # nubmer of mutations
"alpha" :  0.000000009, # step size parameter
"sigma" : 0.0000000001, #search distribution variance
"xlim_u" : xlim_u,
"xlim_l" : xlim_l,
}

grid_data = {
    "geo_coords" : np.stack((grid_lat, grid_long), axis= -1),
    "demand" : y_pred
}

## Initalize optimization objective function
bw_station_opt = bay_wheel_station_opt(grid_data)

## Set initial station placement
geo_coords = grid_data["geo_coords"]
init_ind = np.random.randint(len(grid_data["geo_coords"]), size=(n_sta
tion, 2))
x_init = np.zeros([n_station,2])
for i in range(n_station):
    x_init[i,] = geo_coords[init_ind][i][0]
x_init = np.ndarray.flatten(x_init, 'F')

## Optmize using Evolutionary Strategy
results = es_optimize(bw_station_opt,x_init, es_coef)
```

In [16]:

```
x_vec = results["x_vec"]
y_vec = results["y_vec"]
final_x = results["final_x"]

plt.figure(figsize=(7,5))
plt.plot(x_vec[:,0],'-x');
plt.ylabel("Solution (x)")
plt.xlabel("Steps");
plt.show()

plt.figure(figsize=(7,5));
plt.plot(y_vec,'->');
plt.ylabel("Function value (y)")
plt.xlabel("Steps");
plt.show()

print(final_x)
```



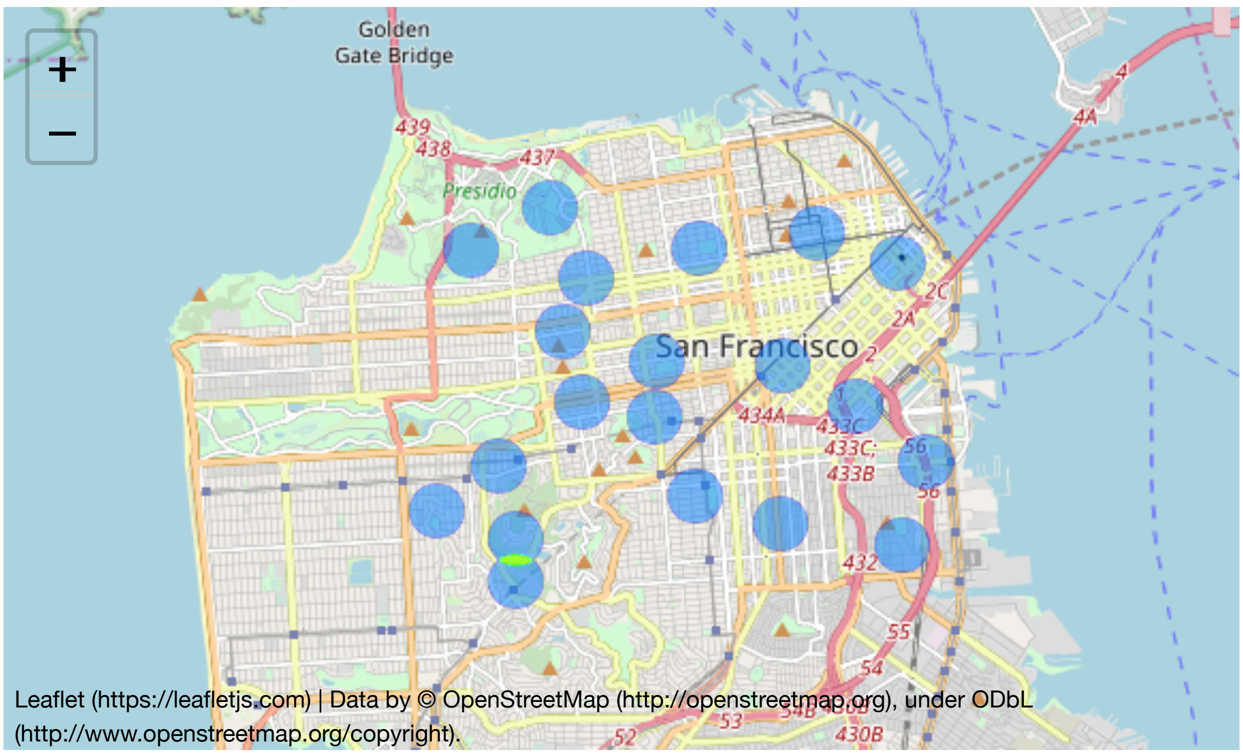



```
[  37.79125989   37.77134786   37.7911139    37.78746726   37.757895
57
   37.74906591   37.78953792   37.79637875   37.79325963   37.776353
59
   37.7643116    37.7543364    37.78081557   37.77000151   37.777079
67
   37.75637603   37.75993587   37.7536186    37.76362041   37.771900
97
 -122.42914461 -122.40411367 -122.4659791  -122.4473686  -122.471628
39
 -122.45881629 -122.39715554 -122.45324133 -122.41050979 -122.415919
92
 -122.3928072  -122.45871089 -122.45109138 -122.43646841 -122.436045
12
 -122.4161779  -122.43005674 -122.39667674 -122.46163947 -122.448021
75]
```

In [17]:

```
##### Visualization with map
final_station_geos = geo_vec_to_tab(final_x)
cor_ref_start = (coor_ll+coor_ur)/2
m = folium.Map(cor_ref_start, zoom_start=13)

# plot heatmap of new stations
m.add_child(plugins.HeatMap(final_station_geos, max_zoom = 13, max_val
=1, radius = 13, blur=1))

## Comparison
grid_data = np.stack((grid_lat, grid_long, y_pred), axis=-1)
## Convert startig trip count to quantiles for visualization
d = grid_data[:,2]
qrange = np.arange(0,100,1.25)
quantiles = np.percentile(d, qrange)
grid_data[:,2] = np.searchsorted(quantiles, d)/len(qrange)
#m.add_child(plugins.HeatMap(grid_data, max_zoom = 13, max_val=1, radi
us = 3, blur=1))
# m.add_child(plugins.HeatMap(station_data, max_zoom = 13, max_val=1,
radius = 6, blur=2))
m
```

Out[17]:



In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```