In [ ]:

```python
import pymongo as pm
import pprint
#import MongoClient only
client = pm.MongoClient('bigdatadb.polito.it',
                        ssl=True,
                        authSource = 'carsharing',
                        username = 'ictts',
                        password ='Ict4SM22!',
                        tlsAllowInvalidCertificates=True)
db = client['carsharing']
#Choose the DB to use
active_booking = db['ActiveBookings']
active_parking = db['ActiveParkings']
permenant_booking = db['PermanentBookings']
permenant_parking = db['PermanentParkings']

enjoy_active_booking = db['enjoy_ActiveBookings']
enjoy_active_parking = db['enjoy_ActiveParkings']
enjoy_permenant_booking = db['enjoy_PermanentBookings']
enjoy_permenant_parking = db['enjoy_PermanentParkings']
#---------------------------------------------------------------------------
#How many documnets are present in each collection?
print("active_booking: ", active_booking.count_documents({}))
print("active_parking: ", active_parking.count_documents({}))
print("permenant_booking: ", permenant_booking.count_documents({}))
print("permenant_parking: ", permenant_parking.count_documents({}))
print("enjoy_active_booking: ", enjoy_active_booking.count_documents({}))
print("enjoy_active_parking: ", enjoy_active_parking.count_documents({}))
print("enjoy_permenant_booking: ", enjoy_permenant_booking.count_documents(
print("enjoy_permenant_parking: ", enjoy_permenant_parking.count_documents(
#---------------------------------------------------------------------------
#distinct cities that are served any the system
print( len(active_booking.find().distinct("city")), ' --> ',active_booking.
print( len(permenant_booking.find().distinct("city")), ' --> ',permenant_boo
print( len(enjoy_active_booking.find().distinct("city")), ' --> ',enjoy_act
print( len(enjoy_permenant_booking.find().distinct("city")), ' --> ',enjoy_
#---------------------------------------------------------------------------
from datetime import datetime
collections = [permenant_booking,enjoy_permenant_booking]
for collection in collections:
    print('collection: ',collection.name)
    print('start: ',datetime.fromtimestamp(list(collection.find().sort([("i
    print('end: ',datetime.fromtimestamp(list(collection.find().sort([("fin
#---------------------------------------------------------------------------
#select number of unique cars in the city of Seattle fleet size
#we checked for 1 or 2 weeks to see fleet size is less
print('Torino-enjoy : ',len(enjoy_permenant_parking.find({'city':'Torino'})
print('Seattle : ',len(permenant_parking.find({'city':'Seattle'}).distinct(
print('Stuttgart : ',len(permenant_parking.find({'city':'Stuttgart'}).distin
start_date = datetime.datetime.strptime("15/12/2016", "%d/%m/%Y").timestamp
end_date = datetime.datetime.strptime("22/12/2016", "%d/%m/%Y").timestamp()
start_date2= datetime.datetime.strptime("01/11/2017", "%d/%m/%Y").timestamp
end_date2 = datetime.datetime.strptime("01/12/2017", "%d/%m/%Y").timestamp(
print(len(permenant_parking.find({'city':'Seattle','init_time':{'$gte':star
print(len(permenant_parking.find({'city':'Seattle','init_time':{'$gte':star
#---------------------------------------------------------------------------
#alternative methods
len(list(permenant_booking.find({'city': 'Seattle', 'public_transport.durat
len(list(permenant_booking.find({'city': 'Seattle', 'walking.duration': {'$
len(list(permenant_booking.find({'city': 'Seattle', 'driving.duration': {'$
#walking is not -1 or public_transport is not -1 or driving is not -1
len(list(enjoy_permenant_booking.find({'city': 'Torino', '$or':[{'public_tr
#---------------------------------------------------------------------------
#$tep2 Analysis of the data
#2.1
start_unix_time = datetime(2017, 11, 1, 0, 0, 0).replace(tzinfo=timezone.utc
end_unix_time = datetime(2018, 1, 31, 23, 59, 59).replace(tzinfo=timezone.ut
def piper(city,start_unix_time,end_unix_time):
    return [
    {
        '$match': {
            'city': city,
            'init_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            },
            'final_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            }
        }
    },
    {
        '$project': {
            '_id': 0,
            'duration': {
                '$divide': [
                    { '$subtract': ['$final_time', '$init_time'] },
                    60  # Divide by 60 to convert seconds to minutes
                ]
            }
        }
    },
    {
        '$sort': { #since the sorting is done in the pipeline, if the result
            'duration': 1  # Sorting by duration in ascending order
        }
    }
    ]

def normalizer(input_data):
    durations = [el['duration'] for el in input_data] #in minutes
    cumulated_data = np.zeros(len(durations))
    for i in range (len(durations)):
        cumulated_data[i] = (i+1)/len(cumulated_data)
    return durations, cumulated_data

def plotter(booking_durations, booking_cumulated_data, parking_durations, pa
    plt.figure()
    plt.semilogx(booking_durations,booking_cumulated_data,label='Permenant_B
    plt.semilogx(parking_durations,parking_cumulated_data,label='Permenant_P
    plt.grid()
    plt.legend()
    plt.xlabel('Duration [min]')
    plt.title('Cdf of bookings and parkings in {}, {}, November 2017 - Janua
    plt.show()

City = 'Torino'
Nov_Jan_Bookings = enjoy_permenant_booking.aggregate(piper(City,start_unix_t
Nov_Jan_Parkings = enjoy_permenant_parking.aggregate(piper(City,start_unix_t

booking_durations, booking_cumulated_data = normalizer(Nov_Jan_Bookings)
parking_durations, parking_cumulated_data = normalizer(Nov_Jan_Parkings)
plotter(booking_durations, booking_cumulated_data, parking_durations, parkin
#---------------------------------------------------------------------------
#2.1.c
```

```
# aggregated per weeks or days
def daysPipeline(city,start_unix_time,end_unix_time):
    return [
    {
        '$match': {
            'city': city,
            'init_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            },
            'final_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            }
        }
    },
    {
        '$project': {
            '_id': 0,
            'duration': {
                '$divide': [
                    { '$subtract': ['$final_time', '$init_time'] },
                    60  # Divide by 60 to convert seconds to minutes
                ]
            },
            'weekDay': {'$dayOfWeek': '$init_date'},
        }
    },{
        '$sort': { #since the sorting is done in the pipeline, if the resul
            'duration': 1  # Sorting by duration in ascending order
        }
    },
    {
        '$group': {
            '_id': {'week_day': '$weekDay'},
            'durations': {'$push': '$duration'}
        }
    }

]

def weeksPipeline(city,start_unix_time,end_unix_time):
    return [
    {
        '$match': {
            'city': city,
            'init_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            },
            'final_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            }
        }
    },
    {
        '$project': {
            '_id': 0,
            'duration': {
                '$divide': [
                    { '$subtract': ['$final_time', '$init_time'] },
                    60  # Divide by 60 to convert seconds to minutes
                ]
            },
            'week': {'$isoWeek': '$init_date'},
        }
    },{
        '$sort': { #since the sorting is done in the pipeline, if the resul
            'duration': 1  # Sorting by duration in ascending order
        }
    },
    {
        '$group': {
            '_id': {'week': '$week'},
            'durations': {'$push': '$duration'}
        }
    }

]

def plotter(data,City, Company, type):
    data.sort(key=lambda x:x["_id"][type])
    plt.figure()
    for it in range(len(data)):
        el=data[it]
        lenDurations=len(el["durations"])
        cumulate=np.zeros(lenDurations)
        for i in range (lenDurations):
            cumulate[i] = (i+1)/len(cumulate)
        plt.semilogx(el["durations"], cumulate,label = 'Day : '+str(it+1))
    # print("For the {} Day the average duration of bookings is:{}".format(
    plt.legend()
    plt.title('Cdf of bookings durations per Day in {}, enjoy, November 201
    plt.xlabel('Duration [min]')
    plt.ylabel('Percentage [%]')
    plt.show()

booking_days_data = enjoy_permenant_booking.aggregate(daysPipeline('Torino',
plotter(list(booking_days_data),'Torino','Enjoy','week_day')
#---------------------------------------------------------------------------
#2.2
#system utilization over time, aggregated per hour of the day
from datetime import datetime, timedelta
def oopsPipeline(city,start_unix_time,end_unix_time):
    return [
    {
        '$match': {
            'city': city,
            'init_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            },
            'final_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            }
        }
    },
    {
        '$project': {
            '_id': 0,
            'duration': {
                '$divide': [
                    { '$subtract': ['$final_time', '$init_time'] },
                    60  # Divide by 60 to convert seconds to minutes
                ]
            },
            'day': {'$dayOfMonth': '$init_date'},
            'hour': {'$hour': '$init_date'},
            'date': {
                '$dateToString': {
                    'format': '%Y-%m-%d',
                    'date': '$init_date'
                }
            }
        }
    },
    {
        '$group':{
            '_id': {'day': '$day', 'hour': '$hour', 'date': '$date'},
            'total_count': {'$sum': 1},
        }
    },
    {
        '$sort': {
            '_id': 1,
        }
    },
    {
        '$group': {
            '_id': '$_id.date',
            'hours': {
                '$push': {
                    'hour': '$_id.hour',
                    'total_count': '$total_count'
                }
            }
        }
    },
    {
        '$sort': {
            '_id': 1,
            'hours.hour': 1
        }
    }
]

book_data = list(permenant_booking.aggregate(oopsPipeline('Stuttgart',start_
park_data = list(permenant_parking.aggregate(oopsPipeline('Stuttgart',start_

flattened_data = []
for entry in book_data:
    date = entry['_id']
    for hour_data in entry['hours']:
        flattened_data.append({
            'date': date,
            'hour': hour_data['hour'],
            'total_count': hour_data['total_count']
        })
all_dates =[]
all_values = []
for entry in flattened_data:
    current_date = datetime.strptime(entry['date'], '%Y-%m-%d')
    all_dates.append(current_date+timedelta(hours=entry['hour']))
    all_values.append(entry['total_count'])

flattened_park_data = []
for entry in park_data:
    date = entry['_id']
    for hour_data in entry['hours']:
        flattened_park_data.append({
            'date': date,
            'hour': hour_data['hour'],
            'total_count': hour_data['total_count']
        })
park_all_dates =[]
park_all_values = []
for entry in flattened_park_data:
    current_date = datetime.strptime(entry['date'], '%Y-%m-%d')
    park_all_dates.append(current_date+timedelta(hours=entry['hour']))
    park_all_values.append(entry['total_count'])

unique_dates = sorted(set(date.date() for date in all_dates))

plt.figure(figsize=(14, 6))
plt.plot(all_dates, all_values, label='Book', color='blue')
plt.plot(park_all_dates, park_all_values, label='Park', color='orange')
plt.xlabel('Date')
plt.ylabel('Total Count')
plt.legend()
plt.grid(True)
plt.title('Total Counts Across Dates and Hours')
plt.grid(True)
plt.xticks(unique_dates,rotation=90, fontsize=5)
plt.show()

plt.show()
#---------------------------------------------------------------------------
#2.3
#drive a critical analysis of the data and filter the outlier
from datetime import datetime, timedelta
def booking_duration_filter_pipeline(city,start_unix_time,end_unix_time):
    return [
    {
        '$match': {
            'city': city,
            'init_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            },
            'final_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            }
        }
    },
    {
        '$project': {
            '_id': 0,
            'duration': {
                '$divide': [
                    { '$subtract': ['$final_time', '$init_time'] },
                    60  # Divide by 60 to convert seconds to minutes
                ]
            },
            'day': {'$dayOfMonth': '$init_date'},
            'hour': {'$hour': '$init_date'},
            'date': {
                '$dateToString': {
                    'format': '%Y-%m-%d',
                    'date': '$init_date'
                }
            },
            'moved': {
                '$ne':[
                    {"$arrayElemAt": [ "$origin_destination.coordinates", 0]
                    {"$arrayElemAt": [ "$origin_destination.coordinates", 1]
                 ]
            }
        }
    },
    {
        '$match': {
            'moved': True,
            'duration':{'$gt':5, '$lt':180},

        }
    },
    {
        '$group':{
            '_id': {'day': '$day', 'hour': '$hour', 'date': '$date'},
            'total_count': {'$sum': 1},
        }
    },
    {
        '$sort': {
            '_id': 1,
        }
    },
    {
        '$group': {
            '_id': '$_id.date',
            'hours': {
                '$push': {
                    'date': '$_id.date',
                    'hour': '$_id.hour',
                    'total_count': '$total_count'
                }
            }
        }
    },
    {
        '$sort': {
            '_id': 1,
            'hours.hour': 1
        }
    }
]


def poarking_duration_filter_pipeline(city,start_unix_time,end_unix_time):
    return [
    {
        '$match': {
            'city': city,
            'init_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            },
            'final_time': {
                '$gte': start_unix_time,
                '$lte': end_unix_time
            }
        }
    },
    {
        '$project': {
            '_id': 0,
            'duration': {
                '$divide': [
                    { '$subtract': ['$final_time', '$init_time'] },
                    60  # Divide by 60 to convert seconds to minutes
                ]
            },
            'day': {'$dayOfMonth': '$init_date'},
            'hour': {'$hour': '$init_date'},
            'date': {
                '$dateToString': {
                    'format': '%Y-%m-%d',
                    'date': '$init_date'
                }
            }
        }
    },
    {
        '$match': {
            'duration':{'$gt':5, '$lt':720},

        }
    },
    {
        '$group':{
            '_id': {'day': '$day', 'hour': '$hour', 'date': '$date'},
            'total_count': {'$sum': 1},
        }
    },
    {
        '$sort': {
            '_id': 1,
        }
    },
    {
        '$group': {
            '_id': '$_id.date',
            'hours': {
                '$push': {
                    'date': '$_id.date',
                    'hour': '$_id.hour',
                    'total_count': '$total_count'
                }
            }
        }
    },
    {
        '$sort': {
            '_id': 1,
            'hours.hour': 1
        }
    }
]
#----------------------------------------------------------------------------
#2.4
#aggregated by the hours of th days to get a better look on the population
book_total = list(permenant_booking.aggregate(oopsPipeline22('Seattle',star
park_total = list(permenant_parking.aggregate(oopsPipeline22('Seattle',star

flattened_data = []
for entry in book_total:
    flattened_data.append(entry['total_per_hour'])

flattened_park_data = []
for entry in park_total:
    flattened_park_data.append(entry['total_per_hour'])
```

```
bar_width = 0.40
x_positions = np.arange(24)

plt.figure(figsize=(14, 6))
plt.bar(x_positions - bar_width/2, flattened_data, bar_width, label='Book',
plt.bar(x_positions + bar_width/2, flattened_park_data, bar_width, label='Pa
plt.xlabel('Date')
plt.ylabel('Total Count')
plt.legend()
plt.title('Total Counts Across Dates and Hours')
plt.grid(True)
plt.xticks(x_positions,range(0,24),rotation=60, fontsize=10)
plt.show()
#-------------------------------------------------------------------------
#2.5
#avg - median - std - percentiles 90
def book_stat_pipeline(city, start_unix_time, end_unix_time):
  return [
      {
        "$match":{
        "city": city,
          "init_time":{"$gte":start_unix_time,"$lte":end_unix_time},
          "final_time":{"$gte":start_unix_time,"$lte":end_unix_time},
        }
      },
      {
        "$project":{
        "hour": { "$hour": "$init_date" },
        "day": { "$dayOfMonth": "$init_date" },
        "duration":{"$divide" : [{"$subtract":["$final_time","$init_time"]},
        "moved": { "$ne": [
            {"$arrayElemAt": [ "$origin_destination.coordinates", 0]},
            {"$arrayElemAt": [ "$origin_destination.coordinates", 1]}
            ]
        }
        }
      },
      {
        "$match":{
          "moved":True,
          "duration":{"$lte":180, "$gte":5}
        }
      },
      {
        "$sort":{"duration":1}
      },
      {
        "$group":{"_id": {"day":"$day"},"list_values": { "$push": "$duratio
      },
      {
        "$project":{
          "day":1,
          "average":{"$avg":"$list_values"},
          "percentile_90":{"$arrayElemAt":["$list_values",
              {"$floor":{ "$multiply": [0.9,{"$size": "$list_values"}]}}]},
          "median":{"$arrayElemAt":["$list_values",
              {"$floor":{ "$multiply": [0.5,{"$size": "$list_values"}]}}]},
          "st_Dev":{"$stdDevSamp":"$list_values"}
        }
      },
 ]

def park_stat_pipeline(city, start_unix_time, end_unix_time):
  return [
    {
    "$match":{
      "city": city,
      "init_time":{"$gte":start_unix_time,"$lte":end_unix_time},
      "final_time":{"$gte":start_unix_time,"$lte":end_unix_time},
    }
    },
    {
    "$project":{
      "hour": { "$hour": "$init_date" },
      "day": { "$dayOfMonth": "$init_date" },
      "duration":{"$divide" : [{"$subtract":["$final_time","$init_time"]},6
    }
    },{
        "$match":{
          "duration":{"$lt":720, "$gt":5}
        }
      },
    {
    "$sort":{"duration":1}
    },
    {
    "$group":{"_id": {"day":"$day"},"list_values": { "$push": "$duration" }
    },
    {
    "$project":{
      "day":1,
      "average":{"$avg":"$list_values"},
      "percentile_90":{
        "$arrayElemAt":[
          "$list_values",
                        {"$floor":{
                          "$multiply":
                          [0.9,{"$size": "$list_values"}]
                          }}]},
                          "median":{
                            "$arrayElemAt":[
                              "$list_values",
                              {"$floor":
                                {
                                 "$multiply":
                                 [0.5,{"$size": "$list_values"}]
                                 }
                              }
                            ]
                          },
                                 "st_Dev":{"$stdDevSamp":"$list_values"},
      }
    },
 ]

Bookings_per_hours_date_filtered = list(enjoy_permenant_booking.aggregate(bo

Bookings_per_hours_date_filtered.sort(key=lambda x:(x["_id"]["day"]))

averages_filtered=[el["average"] for el in Bookings_per_hours_date_filtered]
percentile_90_filtered=[el["percentile_90"] for el in Bookings_per_hours_dat
median_filtered=[el["median"] for el in Bookings_per_hours_date_filtered]
st_Dev_duration_filtered=[el["st_Dev"] for el in Bookings_per_hours_date_fil
days=[str(el["_id"]["day"]) for el in Bookings_per_hours_date_filtered]

plt.figure()
plt.plot(days,averages_filtered,label="Avg_fil")
plt.plot(days,percentile_90_filtered,label="Per_90_fil")
plt.plot(days,median_filtered,label="Med_fil")
plt.plot(days,st_Dev_duration_filtered,label="Std_dev_fil")

plt.grid()
plt.legend()
plt.xticks(rotation=60)
plt.xlabel('Day')
plt.ylabel('Duration')
plt.title('Statistics of duration bookings')
plt.show()
#---------------------------------------------------------------------------
#2.6
#location of the parked cars on the map
start_unix_time = datetime(2017, 11, 1, 0, 0, 0).replace(tzinfo=timezone.utc
end_unix_time = datetime(2018, 1, 31, 23, 59, 59).replace(tzinfo=timezone.ut

booking_locations = list(enjoy_permenant_booking.aggregate([
    {'$match': {'city': 'Torino'}},
    {'$project': {
        '_id': 0,
        'duration': {'$divide': [{'$subtract': ['$final_time', '$init_time']
        'plate': 1,
        'city': 1,
        'init_time': 1,
        'init_date': 1,
        'year': {'$year': '$init_date'},
        'month': {'$month': '$init_date'},
        'day': {'$dayOfWeek': '$init_date'},
        'hour': {'$hour': '$init_date'},
        'origin_longitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_dest
        'origin_latitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_desti
        'dest_longitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_destin
        'dest_latitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_destina
    }},
    {'$match': {
        'duration': {'$gte': 5},
        'init_time': {'$gte': start_unix_time, '$lte': end_unix_time},
        'hour': {'$gte': 8, '$lt': 10},
        'day':{'$gte': 1, '$lte':5}
    }}
]))

my_pd = pd.DataFrame(booking_locations)

import geopandas as gpd
import pandas as pd
from geopandas import GeoDataFrame
from shapely.geometry import Point
import folium
from datetime import datetime, timezone

longitude = my_pd['dest_longitude'].iloc[:]
latitude = my_pd['dest_latitude'].iloc[:]
fina_df = my_pd[['dest_latitude','dest_longitude']]

my_geometry = gpd.points_from_xy(latitude,longitude,crs='EPSG:4326')

gdf = GeoDataFrame(fina_df, geometry=my_geometry)
gdf.explore()
#---------------------------------------------------------------------------
#2.6.b
#HeatMap of the parked cars
# according to the documentation of pymongo and mongodb, $dayOfWeek returns
# so we chose 1 and 7 to represent the weekend
# and 2 to 6 to represent the weekdays
#and this is the refrerence https://docs.mongodb.com/manual/reference/operat

start_unix_time = datetime(2017, 11, 1, 0, 0, 0).replace(tzinfo=timezone.utc
end_unix_time = datetime(2018, 1, 31, 23, 59, 59).replace(tzinfo=timezone.ut

booking_locations = list(permenant_booking.aggregate([
    {'$match': {'city': 'Seattle'}},
    {'$project': {
        '_id': 0,
        'duration': {'$divide': [{'$subtract': ['$final_time', '$init_time']
        'plate': 1,
        'city': 1,
        'init_time': 1,
        'init_date': 1,
        'year': {'$year': '$init_date'},
        'month': {'$month': '$init_date'},
        'day': {'$dayOfWeek': '$init_date'},
        'hour': {'$hour': '$init_date'},
        'origin_longitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_dest
        'origin_latitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_desti
        'dest_longitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_destin
        'dest_latitude': {'$arrayElemAt': [{'$arrayElemAt':['$origin_destina
    }},
    {'$match': {
        'duration': {'$gte': 5},
        'init_time': {'$gte': start_unix_time, '$lte': end_unix_time},
        # 'hour': {'$gte': 8, '$lt': 10},
        # 'day':{'$gte': 1, '$lte':7}
    }}
]))


my_pd = pd.DataFrame(booking_locations)
weekday = my_pd[(my_pd['day'] == 1) | (my_pd['day'] == 7)]
# weekday = my_pd[(my_pd['day'] > 1) & (my_pd['day'] < 7)]
weekday_morning = weekday[(weekday['hour'] >= 8) & (weekday['hour'] <= 10)]
longitude = weekday_morning['dest_longitude'].iloc[:]
latitude = weekday_morning['dest_latitude'].iloc[:]
fina_df = weekday_morning[['dest_latitude','dest_longitude']]
fin_final_df = fina_df.values.tolist()

parking = np.zeros((len(fin_final_df),2))
for i in range (len(fin_final_df)):
    parking[i,1] = resultado[i][0]
    parking[i,0] = resultado[i][1]

parking_coordinates = pd.DataFrame(parking)
parking_coordinates.rename(columns={0:"latitude", 1:"longitude"}, inplace=Tr
parking_coordinates.to_csv (r'/Users/graybook/Downloads/Italy_shapefile/park

df = pd.read_csv(r'/Users/graybook/Downloads/Italy_shapefile/parkinggg.csv')
geometry1 = [Point(xy) for xy in zip(df["longitude"], df["latitude"])]

geo_df1 = gpd.GeoDataFrame(df, geometry=geometry1)

mymap = gpd.read_file(r'/Users/graybook/Downloads/Neighborhood_Map_Atlas_Nei
mymap.to_crs(epsg = 4326 , inplace = True)
c=0
counter=[]
for i in mymap['geometry']:
    for j in geo_df1['geometry']:
        if i.contains(j):
            c+=1
    counter.append(c)
    c=0
mymap['counter']=counter

mymap.plot(column = 'counter',cmap='coolwarm',legend = True,figsize=(10,10))
plt.xticks(rotation=45)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Seattle Density HeatMap')
```