Politecnico di Torino

Laboratory 1(Analysis of the data)

Group 07

Author

Ali Mohammad Alizadeh 308885

Mohammad Eftekhari Pour 307774

Supervisores:

Prof. Marco Mellia

Prof. Luca Vassio

Academic year 2023/24

Introduction

A MongoDB database that is accessible on a server owned by Politecnico di Torino is used for the data analysis of two different car rental companies namely Car2Go and Enjoy. The Appendix contains all the codes.

Step 1 – Preliminary data analysis

1.1 How many documents are present in each collection?

There are 4 collections for the car2go and 4 collections for Enjoy car sharing. Each collection contains documents as below:

Car2Go	#	Enjoy	#
ActiveBookings	8743	enjoy_ActiveBookings	0
ActiveParkings	4790	enjoy_ActiveParkings	0
PermanentBookings	28180508	enjoy_PermanentBookings	6653472
PermanentParkings	28312676	enjoy_PermanentParkings	6689979

1.2 Why the number of documents in PermanentParkings and PermanentBooking is similar?

The Permanent Booking and Permanent Parking collections usually have about the same number of entries. This is because when a car is booked (recorded in Permanent Booking), it’s typically used (recorded in Permanent Parking). Sometimes, things like software starting with cars that aren’t parked or power outages can cause small differences. Also, issues like maintenance or system errors might mix up this order a bit. But generally, each car booking leads to a car being used, so the numbers in both collections should be close.

1.3 For which cities the system is collecting data?

There are 26 cities in the Car2go:

- 'Wien', 'Washington DC', 'Vancouver', 'Twin Cities', 'Toronto', 'Torino', 'Stuttgart', 'Seattle', 'San Diego', 'Roma', 'Rhineland', 'Portland', 'New York City', 'München', 'Montreal', 'Milano', 'Madrid', 'Hamburg', 'Frankfurt', 'Firenze', 'Denver', 'Columbus', 'Calgary', 'Berlin', 'Austin', 'Amsterdam'

There are 6 cities in Enjoy:

- 'Bologna', 'Catania', 'Firenze', 'Milano', 'Roma', 'Torino'

1.4 When did each collection start? When did each collection end?

collection	PermanentBookings	enjoy_PermanentBookings	PermanentParkings	enjoy_PermanentParkings
start	2016-12-13 18:38:23	2017-05-05 17:06:21	2016-12-13 18:37:38	2017-05-05 17:05:36
end	2018-01-31 14:13:03	2019-06-10 19:20:35	2018-01-31 14:13:03	2019-06-10 19:20:35

1.5 What about the time zone of the init_date and init_time timestamps? Which time zone do they refer to?

- UNIX timestamp represents how much second’s past from Unix-time epoch in GMT+0
- init_time is in Unix timestamp which is GMT
- init_date in human readable date which is converted to local time of the city

1.6 What is the total number of cars seen in the whole period in each city? How can you estimate the fleet size in a given period, e.g., one week? How does this relate to the total number of vehicles seen in the whole collection?

The number of cars available in each city for PermanentBooking-PermanentParking in the whole period is:

Torino-enjoy	1901-1900	Seattle	1475-1473	Stuttgart	539-539

The number of PermanentBooking and PermanentParking are the same because each parking is related to a booking done previously and the chain goes on for each car in the dataset. For a specific period of the time we chose two different periods and the result was lower the total fleet size which might be due to some cars being out of service or some cars are added to the fleet after this period, so for a specific period most probably the number of available cars are lower than total fleet size.

1.7 How many bookings have been recorded in November 2017 in each city?

The number of cars in November 2017 in Torino in Enjoy Collection was 301, in Seattle 864 and in Stuttgart 497

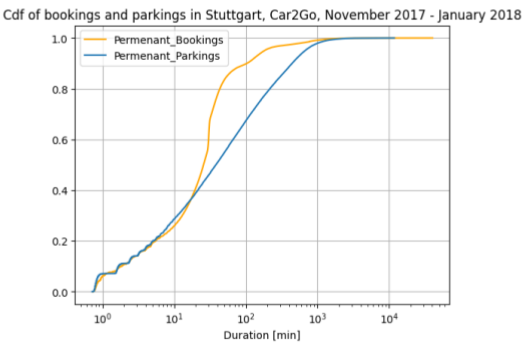
1.8 How many bookings have the alternative transportation modes recorded in each city?

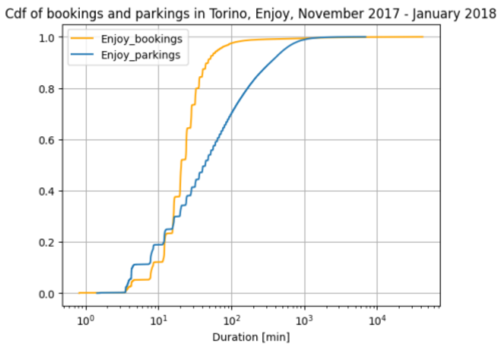
We calculated alternative transportation modes separately. for Seattle and Stuttgart we did not have any alternative transportation modes, But for Torino_Enjoy we had:

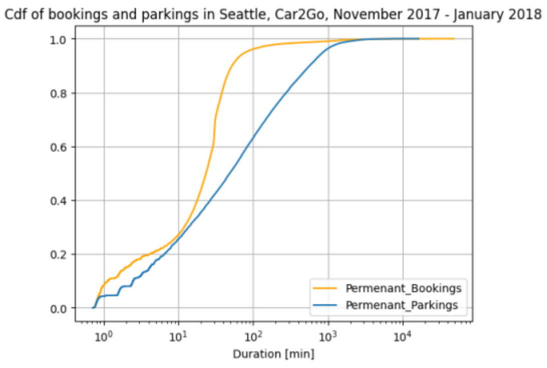
- Public transport: 270219 Walking: 269901 Driving: 270219

Step 2 – Analysis of the data

2.1 Derive the Cumulative Distribution Function of booking/parking duration and plot them. Which consideration can you derive from the results?







The cumulative data was derived from the result of booking/parking and then sorting them and normalizing the data. CDF of booking and parking are represented in the plots above, as it can be observed approximately 25 percent of the bookings and parking last less than 15-20 minutes and about 60 percent of the booking and parking last about less than 100-120 minutes.

2.1.a Which of the CDFs is longer? Is this expected? Does the CDF suggest the presence of some outliers?

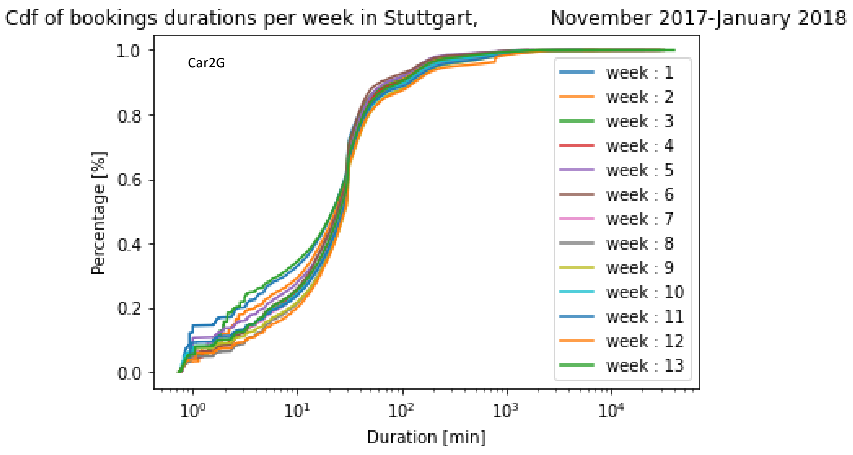
The longest CDF in all 3 figures is associated with Bookings reaching over 10^4 minutes. This is expected and is due to the outliers in the data, this might be due to cars or system failure which cannot be recognized by the system. For example if a car is taken out for repair this is recorded as a booking.

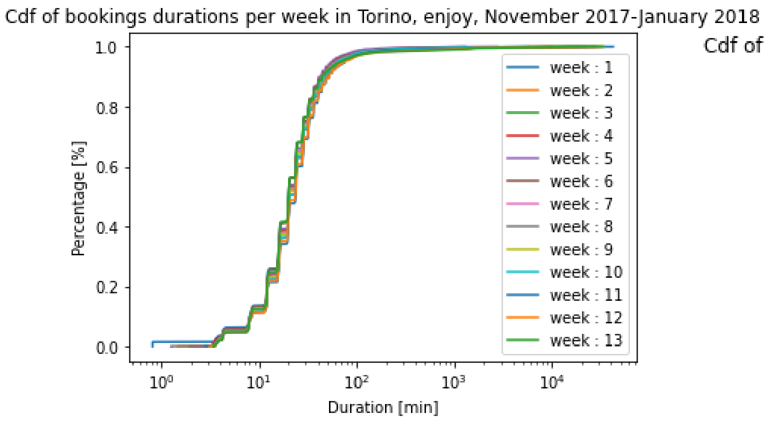
2.1.b Does the per-city CDF change? Why? How can you explain these changes?

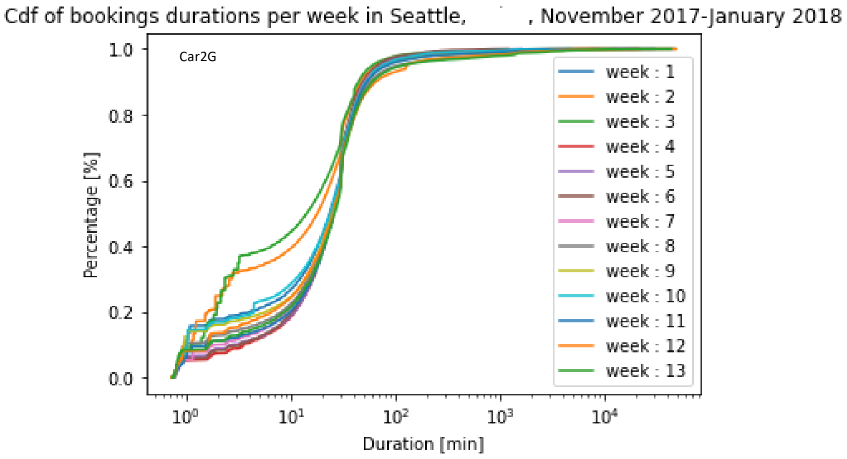
Yes, the CDF for each city changes, this can be a result of the population and area of the city, for example in comparison of the bookings Turin has shorter CDF since it has a smaller area then the others, customers might opt for shorter trips due to proximity of destinations.

2.1.c Does the CDF change over time? E.g., aggregate per different weeks of data, or per different days. Are these CDFs different? Why?

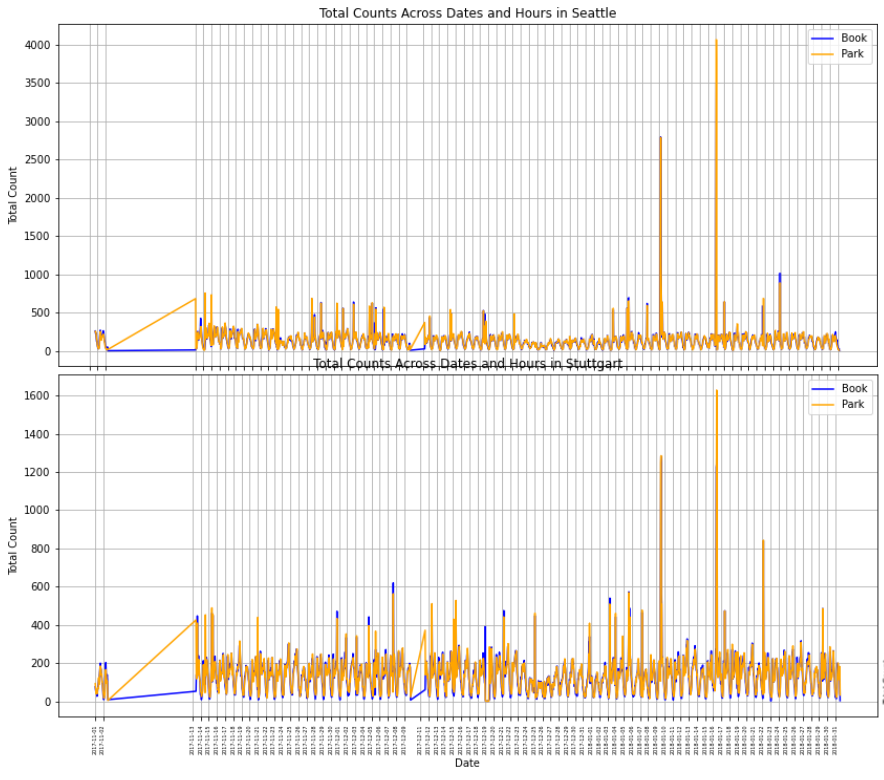
The Trend for different weeks is almost similar except some specific weeks as it can be seen on figure related to Seattle and Stuttgart which can be due to outliers but overall, they follow the same trend and weeks 7 and 8 which corresponds to Christmas holidays have lower number of bookings than the other weeks.



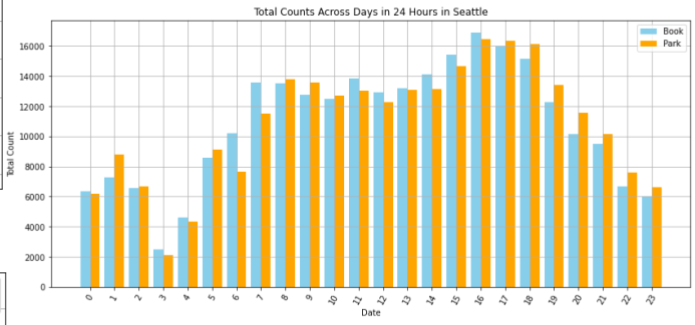


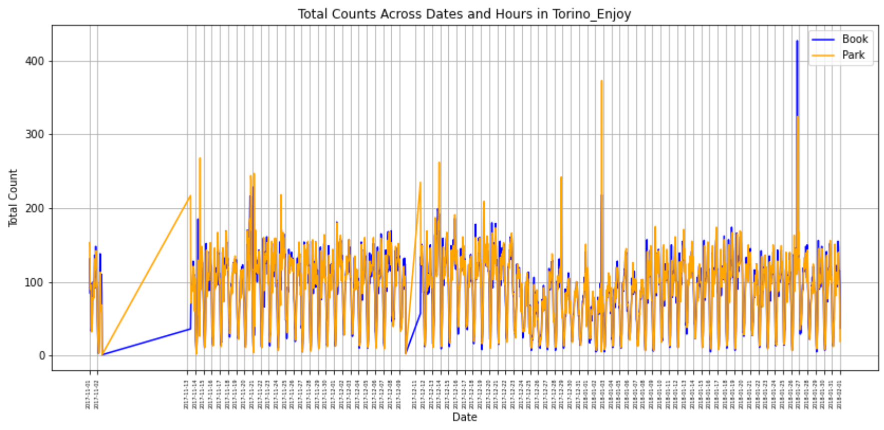


2.2 Consider the system utilization over time: aggregate rentals per hour of the day, and then plot the number of booked/parked cars (or percentage of booked/parked cars) per hour versus time of day. Do you notice any outliers? Can you explain them?



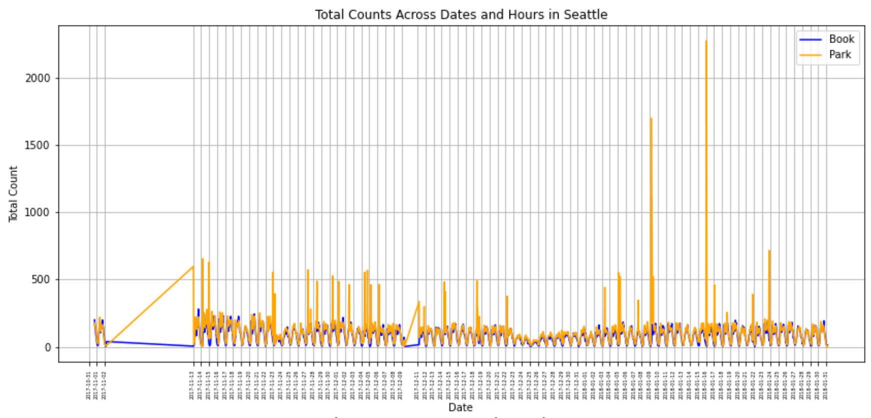
The gap in figures is related to the period which there are no data recorded by the system, also the plot is Date aware so it keeps the period empty. The figure is plotted for each day and each 24 hours of the day for the period. Yes, there are some visible outlier in the figure with unreasonable booking times. Also, it can be observed that the number of bookings has correlation with area of the city and consequently the number of cars and population of the cities, Torino has the lowest number in case of booking and Seattle has the highest numbers.

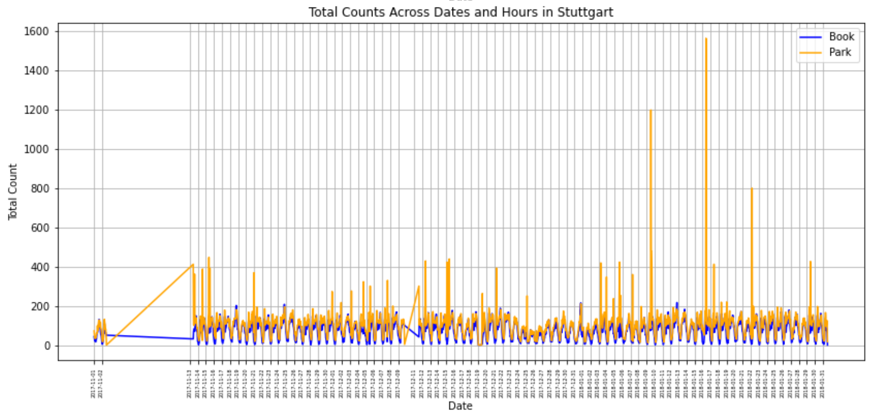


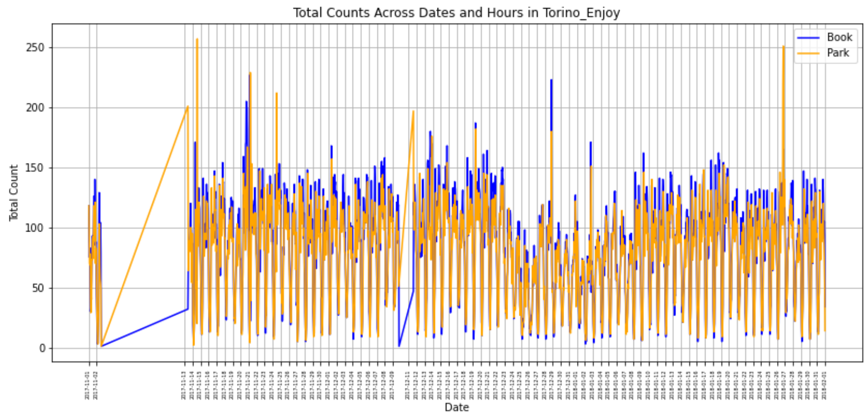


We also aggregated the days of months and calculated the cumulative of booking/parking times which is presented above, in this figure the Peaks and Valleys of booking/parking during different hours of the day. As it can be seen during hours 7-9 and 16-18, we see an increase in number of bookings which is beginning and end of the working day. During 12-13 we see a decrease probably due to lunch time and 3am has the lowest values. Overall trend makes sense and increase during the day, gets to its peak and then has a downward slope.

2.3 Derive a criterion to filter possible outliers (e.g., booking periods that are too short/too long), so as to obtain rentals from bookings, filtering system issues or problems with the data collection.







For the filter we considered some factors for booking and parking, in terms of booking the reasonable shortest time was 5 minutes and longest booking time was 3 Hours, also the origin and destination locations needed to be different.

For the Parking the minimum time of 5 minutes and maximum of 12 hours was considered, also the origin-destination locations must be the same.

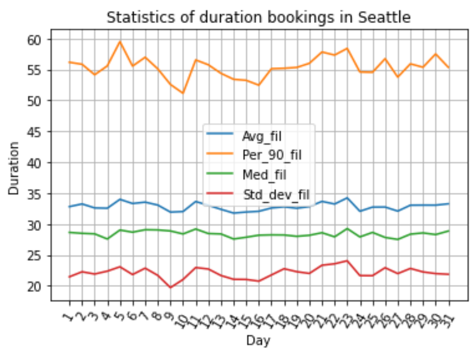
The selected values for the minimum and maximum duration of booking and parking were done considering the cancelation or system errors of maintenance of cars or ….

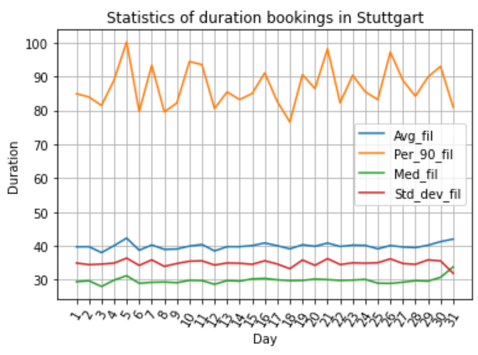
The filtering result is clearly shown in the numbers of cars booked/parked comparing figures of the same city.

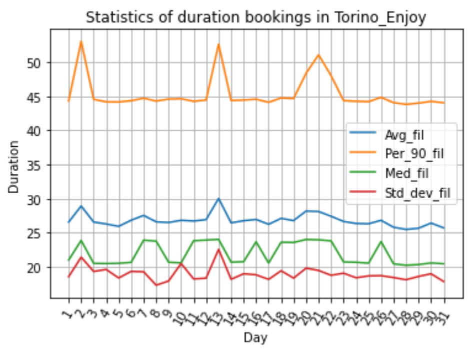
2.4 Filtering data as above, consider the system utilization over time. How do they change compared to the unfiltered versions? Are you able to filter outliers efficiently for both bookings and parking? Consider also to plot the CDF of the filtered events. How do these compare to the unfiltered versions?

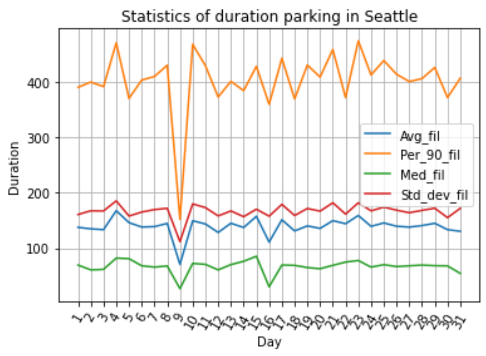
After applying the filter over the data, the number of bookings experienced a significant reduction, and the unusual peaks in the plot are removed. Comparing the Turin's filtered and unfiltered data is the best example of the impact of removing outliers. Some peaks of the parking are still visible which might be due to system failures and the reason needs an in-detail investigation.

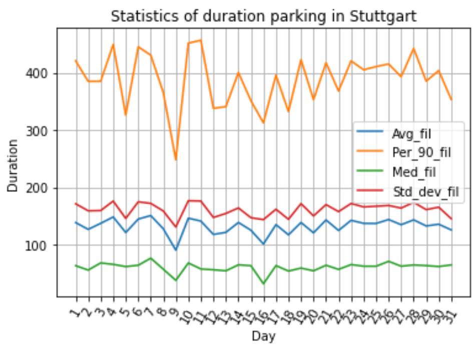
2.5 Filtering the data as above, compute the average, median, standard deviation, and percentiles of the booking/parking duration over time (e.g., per each day of the collection).

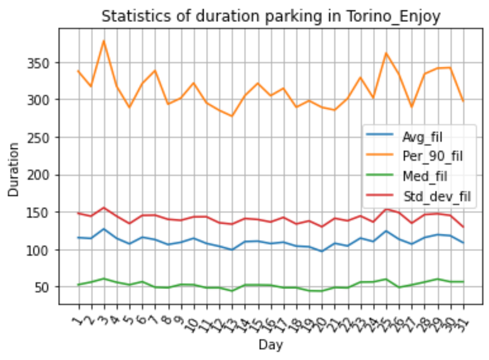












2.5.a Do these figures change over time?

2.5.b Is it possible to spot any periodicity (e.g., weekends vs weekdays, holidays versus working periods)?

2.5.c Is it possible to spot any trend (e.g., increasing, decreasing, holiday periods)?

The figures are plotted using filtered data showing statistics including average, median, standard deviation and percentile 90 that shows where the 90% of the data fall in. the data was provided aggregating the same days of each month. As shown in the booking plots of the cities, Stuttgart has the highest average of bookings reaching over 40 and the rest are lower than 35 and 30 for Seattle and Turin. There is a more fluctuation in the average of booking and parking which is due to simpler filtering on the parking data, this results in higher values for parking average. Percentile 90 shows the largest fluctuations and the other statistics have constant behavior. Considering the weekend days, the number of bookings grows as the number of parking decreases which is the result of weekend trips. There is a weekly periodicity in the average booking duration, with the average booking duration being highest on Fridays and Saturdays, and lowest on Sundays and Mondays. This suggests that people are more likely to book longer car rentals on the weekends. The weekly periodicity in the parking duration plot is similar to the weekly periodicity in the booking duration plot, with the average parking duration being highest on the weekends. This suggests that people are more likely to rent cars and park them for longer periods of time on the weekends.

2.6 Consider one city of your collection and check the position of the cars when returned. Then compute the density of cars at rental ending time (the destination matrix) during different hours of the day.

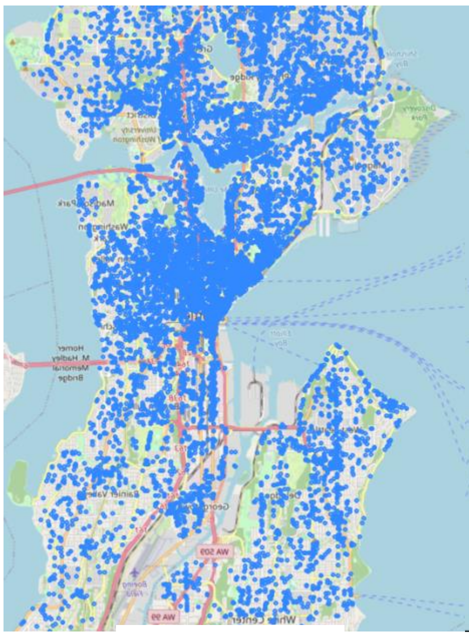
2.6.a Plot the parking position of cars at different times using a mapping service of your preference. For instance, how different are the destination zones on Mondays between 8- 10 and 18-20? Or the same time, but on a different day? Or weekends and weekdays?

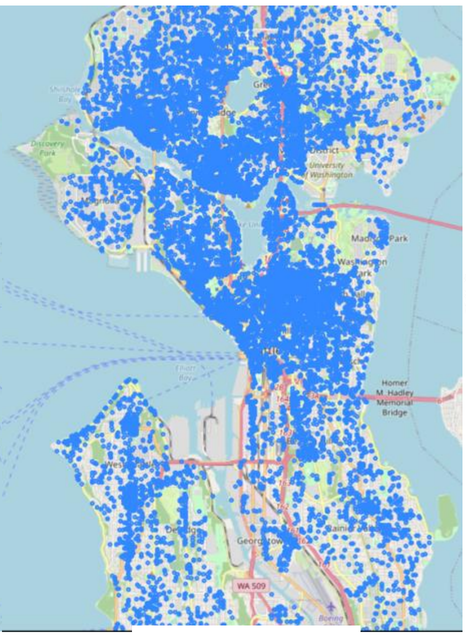
2.6.b Divide the area using a simple squared grid of approximately 500mx500m and compute the density of cars in each area. Plot the results using a heatmap (i.e., assigning a different color to each square to represent the densities of cars).

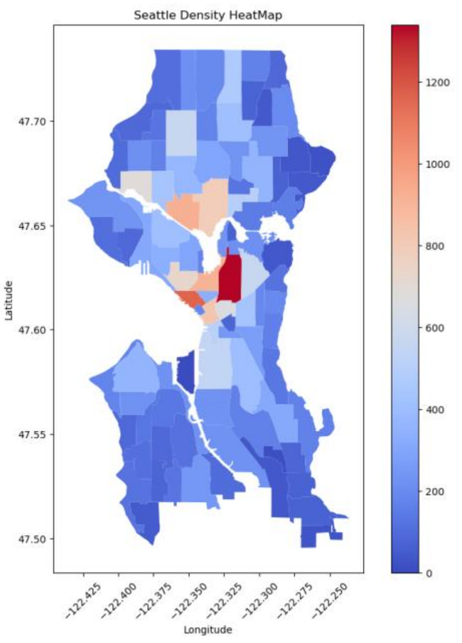
2.6.c Compute then the O-D matrix, i.e., the number of rentals starting in area O and ending in area D. Try to visualize the results in a meaningful way.

For the heatmap we used the shape format file of the city which contains different district with geographical data and boundaries of their latitude and longitude. In this task we are asked to visualize the density of parked cars in different days (weekend and weekdays) and time between 8-10 a.m. and 18-20 p.m. in this task we consider Seattle. As we can see in the figures in the weekdays at morning time the number of cars which is parked in the city center grater then weekdays evening, but in vice-versa at the evening most of cars are parked in the suburbs. This pattern is the same for the weekends.

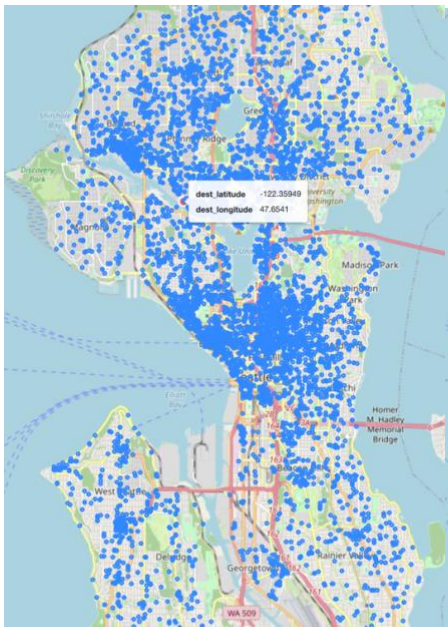
From looking at the density heatmap it can be seen that mostly our population is at the city center and there is a considerable difference in number of the cars between weekdays and weekends, ranging from over 1200 in weekdays to over 400 in weekends.


Weekdays 8-10

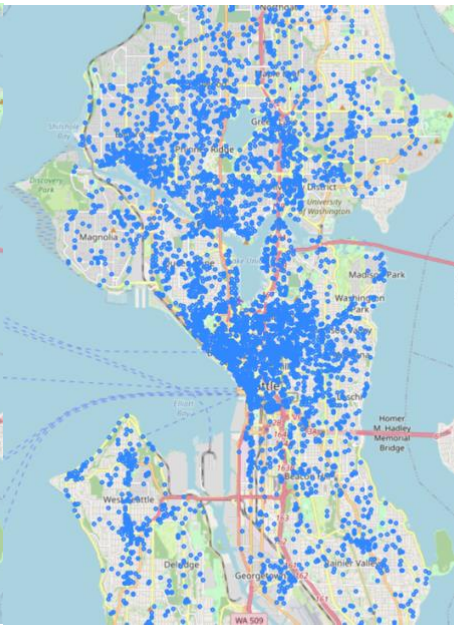

Weekdays 18-20

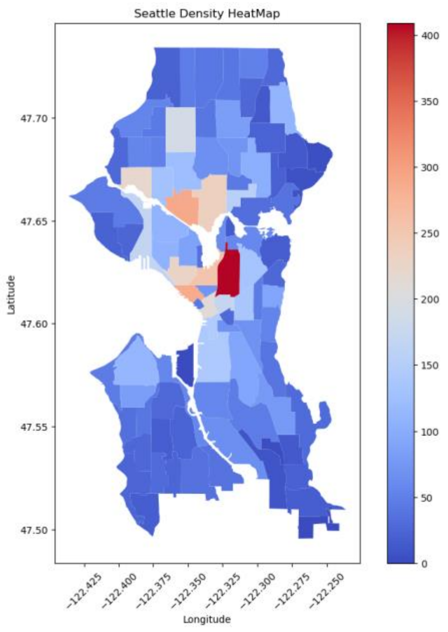


Weekdays 8-10



Weekend 8-10


Weekend 18-20



```
In [ ]: 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_acti
        print( len(enjoy_permenant_booking.find().distinct("city")), ' --> ',enjoy_p
        #-----------------------------------------------------------------------------
        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([("in
            print('end: ',datetime.fromtimestamp(list(collection.find().sort([("fina
        #-----------------------------------------------------------------------------
        #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':{'$gte':start
        print(len(permenant_parking.find({'city':'Seattle','init_time':{'$gte':start
        #-----------------------------------------------------------------------------
        #alternative methods
        len(list(permenant_booking.find({'city': 'Seattle', 'public_transport.durati
        len(list(permenant_booking.find({'city': 'Seattle', 'walking.duration': {'$n
        len(list(permenant_booking.find({'city': 'Seattle', 'driving.duration': {'$n
        #walking is not -1 or public_transport is not -1 or driving is not -1
        len(list(enjoy_permenant_booking.find({'city': 'Torino', '$or':[{'public_tra
        #----------------------------------------------------------------------------
        #$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
```

```python
# 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(i
    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 o
book_total = list(permenant_booking.aggregate(oopsPipeline22('Seattle',start
park_total = list(permenant_parking.aggregate(oopsPipeline22('Seattle',start

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": "$duration
      },
      {
        "$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"}
        }
      },
 ]

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(b

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_da
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_fi
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')
```