ICT for smart mobility

Report of Lab1

Group 10
Setareh Pourgholamali S309064
Hossein Zahedi Nezhad S309247
SeyedMohammadhossein Sajjadikaboudi S313242

November 2023

Step 1 – Lab #1 - Preliminary data analysis

To get used to both MongoDB and the data at your disposal, investigate first the collections and get used to the document and field stored in each.

1.1. How many documents are present in each collection?

Car2go:

Active bookings	Active Parking	Permanent Parking	Permanent Booking
8742	4790	28,312,676	28,180,508

Enjoy:

Active bookings	Active Parking	Permanent Parking	Permanent Booking
0	0	6,689,979	6,653,472

1.2. Why the number of documents in Permanent Parkings and Permanent Booking is similar?

When a user makes a car reservation, the information is recorded in the Permanent Booking Collection. Subsequently, once the user has utilized the car and parked it, the relevant data is transferred to the Permanent Parking Collection. The commonality between these two collections is inherent in the system design, as each vehicle is anticipated to undergo both booking and parking processes every time it is utilized.

1.3. For which cities the system is collecting data?

Car2go:
The are 26 cities presented in the PermanentBookings and PermanentParkings including: Amsterdam, Austin, Berlin, Calgary, Columbus, Denver, Firenze, Frankfurt, Hamburg, Madrid, Milano, Montreal, Munchen, New York City, Portland, Rheinland, Roma, San Diego, Seattle, Stuttgart, Torino, Toronto', Twin Cities, Vancouver, Washington DC, Wien.

Enjoy:
Only six cities—Bologna, Catania, Firenze, Milano, Roma, and Torino—are currently featured in the enjoy_PermanentBookings and enjoy_PermanentParkings.

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

Car2go:
Start Time: 2016-12-13 18:38:23.000Z
End Time: 2018-01-31 14:13:03.000Z

Enjoy:
Start Time: 2017-05-05 17:06:21.000Z
End Time: 2019-06-10 19:20:35.000Z

1.5. What about the timezone of the init_date and init_time timestamps? Which timezone do they refer to?

Each document includes four timestamp fields: init time, init date, final time, and final date. The timestamp zone consistently refers to Greenwich Mean Time (GMT), while the date is converted to the local time of a specific city for easier readability by humans.

Considering the three cities of your group, check
1.6.1. What is the total number of cars seen in the whole period in each city?

Vancouver: 1409 Roma Enjoy: 3020 Milano Enjoy: 1870

1.6.2. 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 fleet size is always lower than the counted value because cars can be added or removed from the system during the collection period, including maintenance, resulting in a reduced total number of cars.

1.**7**. **How many bookings have been recorded in November 2017 in each city?**

Vancouver: 276349 Roma Enjoy: 127356 Milano Enjoy: 197803

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

The database does not have recorded information about transport alternatives for Vancouver. However, for Milano, Enjoy has 825,428 records, and Roam Enjoy has 1,381 records.

Step 2 – Lab # 1 - Analysis of the data

Consider each city of your group, and the period of time of November 1st 2017 – January 31st 2018. Consider the time series (city, timestamp, duration, locations). Process it to further analyse it by producing the following plots and results:

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

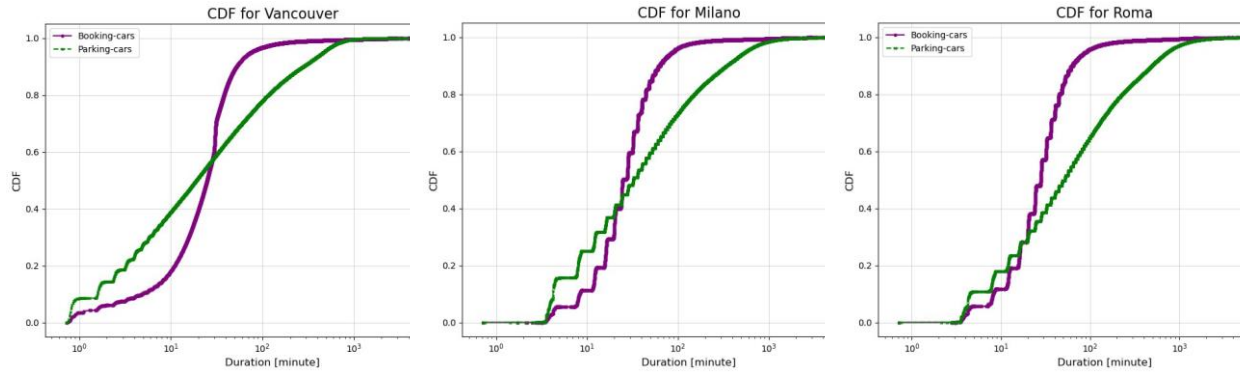

Fig1. CDF of bookings and parkings duration in each city

After Plotting cumulative distribution function (CDF) diagrams for Vancouver, Roma, and Milano, we found that most bookings last less than 100 minutes in all three cities. Another noticeable pattern is a significant rise in the CDF of booking after about 10-15 minutes. In Vancouver, for durations under 50 minutes, there's a higher chance of parking compared to being booked. However, for Milan and Rome, this trend changes, with a higher likelihood of parking than being booked within the first 25 minutes.

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

As shown in the Figure 1, In the intervals, such as first 30 minutes, the cumulative distribution function (CDF) for parking indicates a higher probability. This is due to the lower likelihood of a car being booked for a brief duration, resulting in shorter time frame, making parking more probable. However, for durations ranging between 30 and 80 minutes, the CDF for bookings surpasses the CDF of parking. This shift signifies a higher probability of a car being booked during this time frame compared to it being parked.

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

In general, it can be said that the CDF of the two cities of Rome and Milan are very similar, which could be because both are under the supervision of the Enjoy company Additionally, the percentage of very short bookings in Vancouver is greater than in the other two cities. This could suggest that individuals in Vancouver cancel their bookings more frequently than those in the other cities. Alternatively, it might indicate a higher incidence of errors within the car2go system, which operates cars in Vancouver

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

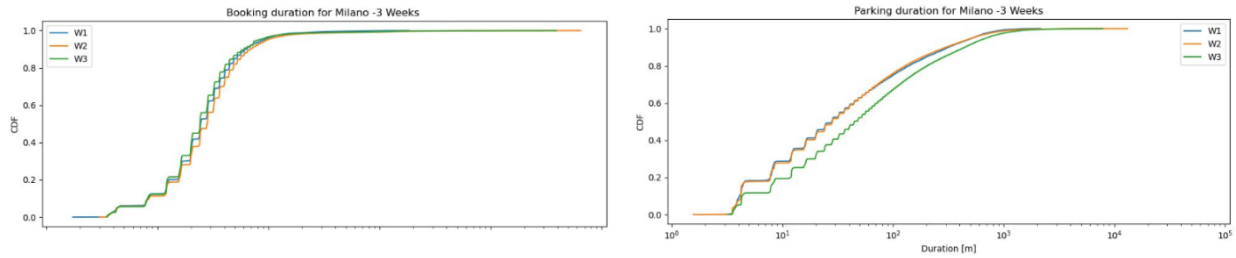

Fig2. CDF of bookings and parkings per different weeks

To complete this task, we examined the cumulative distribution function (CDF) of three distinct weeks in Milan. As depicted in the image, the graphs for each week exhibit remarkable similarity, and follow a completely similar trend for both bookings and parkings during different weeks.

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?

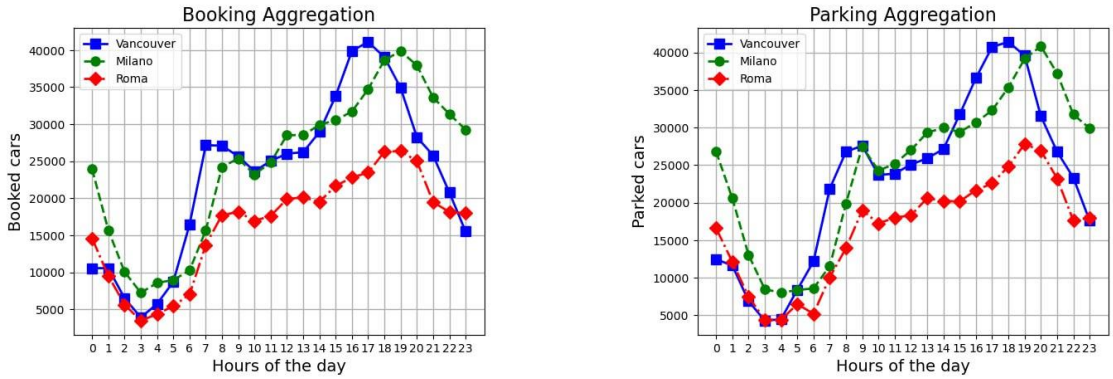

Fig3. Bookings and Parkings per hour of the day for each city

The figures above demonstrate a similar trend across all cities. The number of booked and parked cars typically increases in the morning from 5 to 8 am When people are looking for vehicles to go to work or school and university. The highest count happens in the afternoon, around 5 to 7 pm, coinciding with the end of the workday. After this peak, both rented and parked cars start decreasing for both car-sharing systems.

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.

To create a filter for booking, we established two boundaries for our data: an upper and lower limit for the bookings. Data falling outside these boundaries are identified as outliers and subsequently removed. To define these boundaries, we utilized CDF diagrams, designating 5% of the total data as outliers. This percentage was divided into two parts of 2.5% each, marking the upper and lower limits of each curve. In fact, to create a filter we consider this viewpoint which:

- Excessive Booking time can be flagged as an outlier, indicating potential car issues such as breakdowns, repairs, or system problems.
- In major cities like Milan, Rome, and Vancouver, very short booking durations are unreasonable and might signal system errors or subscriber mistakes leading to cancellations.

Subsequently, for Vancouver, any duration less than 3 minutes or more than 150 minutes is considered an outlier. Meanwhile, for Milan and Rome, the acceptable range is from a minimum of 7 minutes to a maximum of 120 minutes.

In the case of parking, we apply a similar approach, except we designate 5% of the lower limit as outliers. Because Determining the maximum parking duration is challenging, while for the minimum duration, it's crucial to allow adequate time for a customer to get off and another to board. Consequently, Parking, for Vancouver, the parking duration is expected to be more than 3 minutes. In Milan and Roma, this duration is set at 7 minutes for both cities. Any data with a duration less than these limits will be filtered and discarded.

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 parkings? Consider also to plot the CDF of the filtered events. How do these compare to the unfiltered versions?

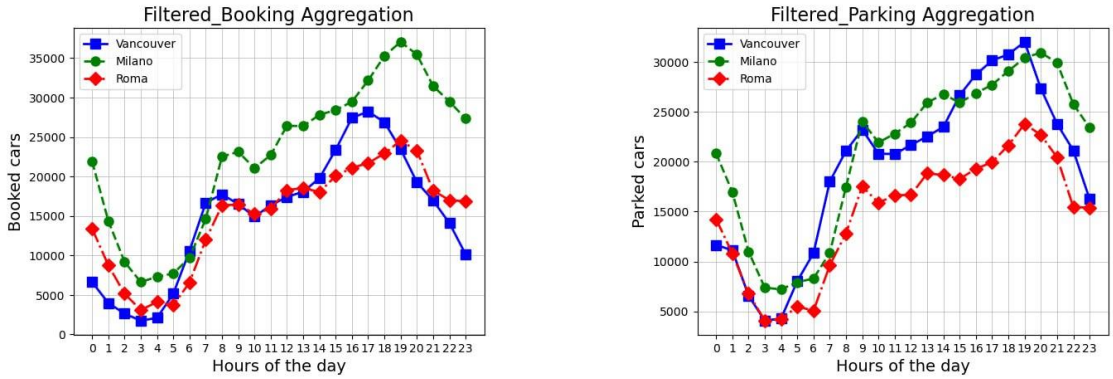

Fig4. Bookings and Parkings per hour of the day for each city after filtering

The depicted figure illustrates a decline in the count of bookings and parkings in all three cities following the identification and removal of outliers such as invalid bookings/parkings. Notably, the reduction in Vancouver surpasses that of the other two cities, which may indicate potential issues within the C2G system or a higher number of cancellations in this city.

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).

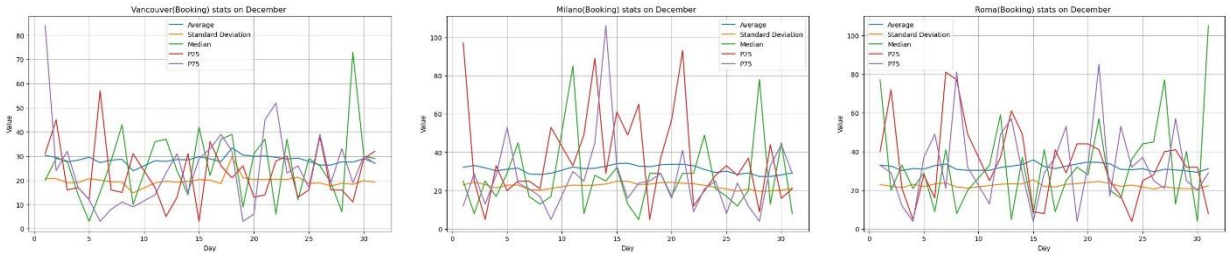

Fig5. Statistical Analysis of Bookings for Vancouver, Milano, and Roma

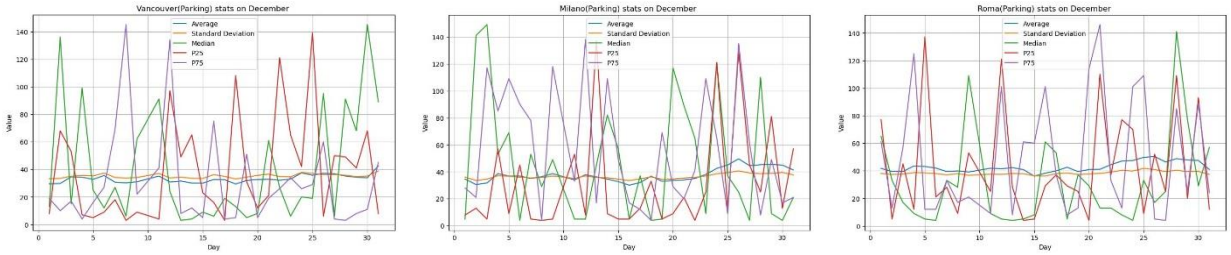

Fig6. Statistical Analysis of Bookings for Vancouver, Milano, and Roma

a. Do these figures change over time?

To get a clearer picture of changes due to the unique mix of holidays and workdays, we specifically focused on December. The data presented in figures 5 and 6, illustrates that the average booking time for cars in Rome and Milan closely aligns with each other and is slightly higher than the average booking time in Vancouver. Additionally, the parking duration curves in Milan and Rome follow similar patterns. Furthermore, regarding both parking and booking metrics, aside from the average and standard deviation, the other parameters display significant fluctuation. Since only the data representing the lower limit of

parking time was excluded as outliers, the diagrams for all three cities display a longer maximum parking duration.

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

The minimum average time of using cars at the end of the week (Saturday and Sunday) decreases compared to other days of the week, and this can be due to less use of cars to transport people to their workplace.

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

Because of the substantial fluctuation within curves, finding trends is challenging but this is possible to some extent for instance it can be stated that, During the Christmas holiday (from December 20 onwards), there is a noticeable decline in the average use of cars across all three cities. Concurrently, there is an increase in average parking time during this period.

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.

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?

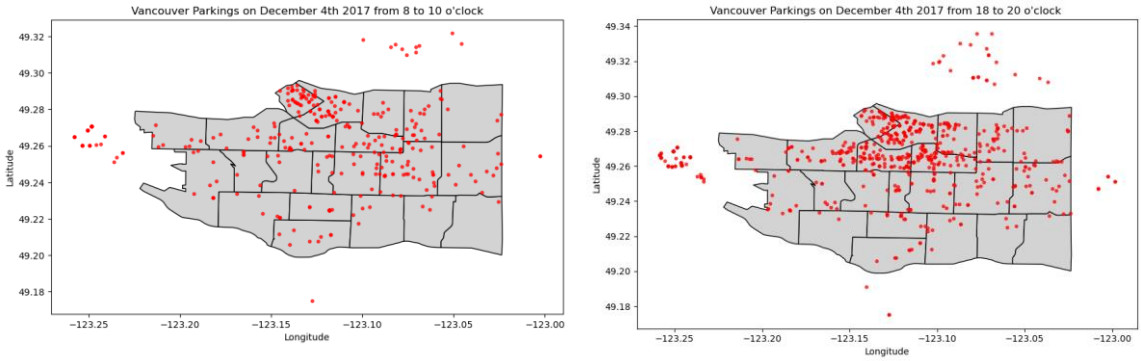

Fig7. Position of Parked cars in Vancouver for different times

To carry out this task, we examine the quantity of parked vehicles during two distinct time slots on Monday, December 4, 2017, as depicted in the image. The concentration of parked cars is notably higher, particularly in the northern region of the city and near the city center from 6:00 to 8:00 PM compared to the morning period. This surge might be attributed to an increased number of bookings between 8:00 and 10:00 AM, likely for commuters heading to work places and Offices.

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 colour to each square to represent the densities of cars).

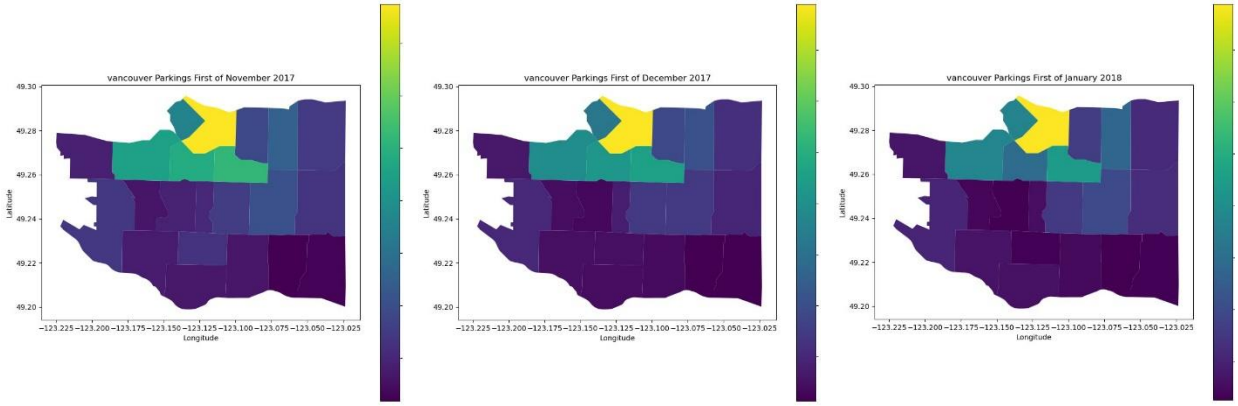

Fig8. Density of Parked cars in Vancouver for different times

The result obtained from the figure above is that the density of parked cars in the north of Vancouver, which is actually the city center and the area where offices and companies are located (yellow area), is stable during the considered period and is more than other places. Also, the areas close to this area have a high density of parked cars, noting that this density has decreased from November to January, which can be due to holiday and closed offices. Also, the lowest density is in the southern areas of Vancouver**.**

Appendix

1.1

```
db.getCollection("PermanentParkings").count()
```

1.3

```
db.getCollection("enjoy_PermanentParkings").distinct('city')
```

1.4

```
db.getCollection("PermanentBookings").find({},{init_date:1,city:1}).sort({init_time:1})
db.getCollection("PermanentBookings").find({},{final_date:1,city:1}).sort({final_time:1})
```

1.6

```
db.PermanentParkings.aggregate([
{
$match: { "city": "Vancouver" }
},
{
$group: {
_id: "$plate",
count: { $sum: 1 }
}
},
{

$group: {
_id: 0,
totalDistinctPlates: { $sum: 1 }
}
}
]);
```

1.7

```
var cities = [ "Milano", "Roma"], len = cities.length
var startUnixTime = new Date("2017-11-01") / 1000
var endUnixTime = new Date("2017-12-01") / 1000
for(i=0; i<len; i++){
c=cities[i]

print("Checking " + c + " bookings: " +
db.enjoy_PermanentParkings.find({
city: c,
init_time: { $gte: startUnixTime, $lte: endUnixTime }}).count()
+" Parkings: " +
db.enjoy_PermanentBookings.find({city: c,init_time: { $gte: startUnixTime, $lte: endUnixTime }}).count())}
```

1.8

```
var a=db.getCollection('enjoy_PermanentBookings').find({city: "Milano"}).count()
var d=db.getCollection('enjoy_PermanentBookings').find({city: "Milano", "walking" : {

"duration" : -1,
"distance" : -1
}, "public_transport" : {
"duration" : -1,
"distance" : -1,
"arrival_date" : -1,
"arrival_time" : -1

}}).count()
print('\nUse Alternative Transport: '+ (a-d))
```

TASK 2

2.1

```
tz_offsets = {'Vancouver': -8}
start_time_str = '2017-11-01 00:00:00'
end_time_str = '2017-11-30 23:59:59'
init_nov = time.mktime(datetime.datetime.strptime(start_time_str, '%Y-%m-%d %H:%M:%S').timetuple())
final_nov = time.mktime(datetime.datetime.strptime(end_time_str, '%Y-%m-%d %H:%M:%S').timetuple())

print(f"Time start = {start_time_str}, {init_nov}")
print(f"Time end = {end_time_str}, {final_nov}")

pipeline1 = [
    {
        '$match': {
            '$and': [
                {'init_time': {'$gte': init_nov + (tz_offsets['Vancouver'] * 60 * 60)}},
                {'init_time': {'$lte': final_nov + (tz_offsets['vancouver'] * 60 * 60)}},
                {'city': 'Vancouver'}]}},
    {
        '$project': {
            '_id': 0,
            'city': 1,
            'duration': {'$divide': [{'$subtract': ["$final_time", "$init_time"]}, 60]}
        }},
    {
        '$group': {
            '_id': "$duration",
            'count': {'$sum': 1}}},{'$sort': {'_id': 1}}]
pile1 = PB_C2.aggregate(pipeline1)
defvan = pd.DataFrame(list(pile1))
C_dfP = np.cumsum(Df_P['count'])      #Obtaining the cumsum of the duration column for booking collection
P_sum = np.sum(Df_P['count'])         #Obtaining the sum of all values of the duration for booking collection
C_df_P_prob = C_dfP/P_sum             #Normalizing the cumsum value for booking collection

plt.figure()
plt.plot(Df_B['_id'], C_df_B_prob, Df_P['_id'], C_df_P_prob)
plt.title("CDF Booking/Parkings duration for Tvancouver")
plt.xlabel("Duration [m]")
plt.ylabel("CDF")
plt.legend(["Bookins", "Parkings"])
plt.grid(b=True, which='minor', color='xkcd:grey', linestyle=':')
plt.xscale("log")
```

2.2&3

```
var startDate = new Date(2017, 11, 1, 0, 0, 0);
var startUnixTime = startDate.getTime() / 1000;
var endDate = new Date(2018, 1, 31, 23, 59, 59);
var endUnixTime = endDate.getTime() / 1000;
var pipeline = [
{
$match: {
init_time: { $gte: startUnixTime, $lte: endUnixTime },
city: "Vancouver"
}
},
},
{
$project: {
_id: 0,
city: 1,
init_date: 1,
duration: { $ceil: [{ $divide: [{ $subtract: ["$final_time", "$init_time"] }, 60] }] },
hour: { $hour: "$init_date" }
}
},
{
$group: {
_id: "$hour",
count: { $sum: 1 }
}
},
{
$sort: { _id: 1 }
}
];
var result = db.PermanentParkings.aggregate(pipeline).toArray();
printjson(result);
```

```
# Data
Vancouver_permanentB = [10557,10562,6500,3907,5724,8715,16424,27164,27097,25662,23637,25037,26019,26219,28951,33815,39887,41111,39051,34896,28257,25782,20784,1552
Vancouver_permanentP = [12442,11624,6877,4264,4466,8385,12175,21859,26787,27578,23678,23862,25080,25895,27135,31739,36600,40750,41411,39583,31510,26788,23324,1760
                        ]



Milano_permanentB = [23941,15686,10056,7237,8567,8961,10252,15650,24236,25353,23211,24849,28528,28516,29917,30546,31705,34728,38656,39857,37940,33618,31380,29260
Milano_permanentP = [26790,20568,13024,8460,7995,8340,8576,11538,19896,27468,24259,25169,27013,29324,29968,29393,30675,32297,35349,39167,40842,37224,31740,29905 ]



Roma_permanentB = [14586,9525,5636,3463,4352,5457,6987,13690,17738,18145,16950,17615,19901,20115,19532,21666,22806,23458,26266,26373,25025,19485,18137,17990]

Roma_permanentP = [16590,12110,7456,4388,4380,6475,5187,10059,13938,18975,17220,18004,18303,20597,20218,20164,21646,22553,24818,27799,26935,23215,17622,17977]

# Aggregation of Bookings per Hour
fig, ax1 = plt.subplots()
ax1.plot(h, Vancouver_permanentB, 's-', label='Vancouver', color='blue', markersize=8, linewidth=2)
ax1.plot(h, Milano_permanentB, 'o--', label='Milano', color='green', markersize=8, linewidth=2)
ax1.plot(h, Roma_permanentB, 'D-.', label='Roma', color='red', markersize=8, linewidth=2)
ax1.legend()
ax1.set_title('Booking Aggregation', fontsize=16)
ax1.grid(linewidth=0.5)
ax1.set_xticks(h)
ax1.set_xlabel('Hours of the day', fontsize=14)
ax1.set_ylabel('Booked cars', fontsize=14)
```

```
# Aggregation of Parkings per Hour
fig, ax2 = plt.subplots()
ax2.plot(h, Vancouver_permanentP, 's-', label='Vancouver', color='blue', markersize=8, linewidth=2)
ax2.plot(h, Milano_permanentP, 'o--', label='Milano', color='green', markersize=8, linewidth=2)
ax2.plot(h, Roma_permanentP, 'D-.', label='Roma', color='red', markersize=8, linewidth=2)
ax2.legend()
ax2.set_title('Parking Aggregation', fontsize=16)
ax2.grid(linewidth=0.5)
ax2.set_xticks(h)
ax2.set_xlabel('Hours of the day', fontsize=14)
ax2.set_ylabel('Parked cars', fontsize=14)

plt.show()

```

2.4

```
#task4
import matplotlib.pyplot as plt
import numpy as np

Vancouver_permanentB = [6623,3979,2637,1738,2100,5159,10589,16674,17740,16507,14917,16343,17420,18065,19777,23383,27424,28226,26878,23453,19286,16967,14111,10162]
Vancouver_permanentP = [11661,11117,6561,4070,4279,8031,10843,18016,21095,23220,20804,20776,21679,22535,23497,26669,28741,30147,30769,32013,27330,23791,21131,1625



Milano_permanentB = [21908,14380,9184,6626,7312,7715,9713,14642,22524,23120,21053,22718,26434,26371,27816,28408,29469,32158,35264,37043,35448,31458,29498,27323]
Milano_permanentP = [ 20833,16963,10987,7369,7210,7920,8295,10861,17477,24034,21929,22808,23914,25922,26782,25914,26871,27716,29100,30411,30946,29905,25777,23410]



Roma_permanentB = [13400,8759,5145,3097,4100,3681,6584,11995,16342,16453,15275,15906,18264,18555,18020,20071,21086,21701,22908,24546,23271,18225,17004,16855]

Roma_permanentP = [14244,10819,6775,4062,4192,5489,5063,9616,12827,17519,15889,16634,16680,18825,18670,18284,19289,19953,21648,23794,22685,20436,15453,15395]

# Aggregation of Bookings per Hour
fig, ax1 = plt.subplots()
ax1.plot(h, Vancouver_permanentB, 's-', label='Vancouver', color='blue', markersize=8, linewidth=2)
ax1.plot(h, Milano_permanentB, 'o--', label='Milano', color='green', markersize=8, linewidth=2)
ax1.plot(h, Roma_permanentB, 'D-.', label='Roma', color='red', markersize=8, linewidth=2)
ax1.legend()
ax1.set_title('Filtered_Booking Aggregation', fontsize=16)
ax1.grid(linewidth=0.5)
ax1.set_xticks(h)
ax1.set_xlabel('Hours of the day', fontsize=14)
ax1.set_ylabel('Booked cars', fontsize=14)

# Aggregation of Parkings per Hour
fig, ax2 = plt.subplots()
ax2.plot(h, Vancouver_permanentP, 's-', label='Vancouver', color='blue', markersize=8, linewidth=2)
ax2.plot(h, Milano_permanentP, 'o--', label='Milano', color='green', markersize=8, linewidth=2)
ax2.plot(h, Roma_permanentP, 'D-.', label='Roma', color='red', markersize=8, linewidth=2)
ax2.legend()
ax2.set_title('Filtered_Parking Aggregation', fontsize=16)
ax2.grid(linewidth=0.5)
ax2.set_xticks(h)
ax2.set_xlabel('Hours of the day', fontsize=14)
ax2.set_ylabel('Parked cars', fontsize=14)

plt.show()
```

2.5

```
def print_result(data):
    print('{\n"Day" : ' + str(data["_id"]) + ',\n"Total Rentals" : ' + str(data["count"]) +
        ',\n"Average" : ' + str(data["avg"]) + ',\n"Std Dev" : ' + str(data["stdDev"]) +
        ',\n"Median" : ' + str(data["median"]) + ',\n"P25" : ' + str(data["p25"]) +
        ',\n"P75" : ' + str(data["p75"]) + '\n}')

# start and end dates
start_date = datetime.datetime(2017, 12, 1, 0, 0, 0)
end_date = datetime.datetime(2017, 12, 31, 23, 59, 59)

# Convert dates to Unix time
start_unix_time = int(start_date.timestamp())
end_unix_time = int(end_date.timestamp())

# Generating range for the data
date_range = [{"day": day} for day in range(1, 32)]

# aggregation pipeline
pipeline = [
    {
        "$match": {
            "init_time": {"$gte": start_unix_time, "$lte": end_unix_time},
            "city": "Vancouver"
        }
    },
    {
        "$project": {
            "_id": 0,
```

```
            "city": 1,
            "init_date": 1,
            "duration": {"$ceil": {"$divide": [{"$subtract": ["$final_time", "$init_time"]}, 60]}},
            "moved": {
                "$ne": [
                    {"$arrayElemAt": ["$origin_destination.coordinates", 0]},
                    {"$arrayElemAt": ["$origin_destination.coordinates", 1]}
                ]
            }
        }
    },
    {
        "$match": {
            "duration": {"$gte": 3, "$lte": 150}
        }
    },
    {
        "$project": {
            "city": 1,
            "duration": 1,
            "moved": 1,
            "day": {"$dayOfMonth": "$init_date"}
        }
    },
    {
        "$group": {
            "_id": "$day",
            "count": {"$sum": 1},
```

```
            "avg": {"$avg": "$duration"},
            "stdDev": {"$stdDevPop": "$duration"},
            "durations": {"$push": "$duration"}
        }
    },
    {
        "$sort": {"_id": 1}
    },
    {
        "$project": {
            "_id": 1,
            "count": 1,
            "avg": 1,
            "stdDev": 1,
            "median":{"$arrayElemAt":["$durations",
              {"$floor":{ "$multiply": [0.5,{"$size": "$durations"}]}}]},
            "p25": {"$arrayElemAt": ["$durations", {"$floor": {"$multiply": [0.25, {"$size": "$durations"}]}}]},
            "p75": {"$arrayElemAt": ["$durations", {"$floor": {"$multiply": [0.75, {"$size": "$durations"}]}}]}
        }
    }
]

# Execute the pipeline
result_cursor = db.PermanentBookings.aggregate(pipeline)
result_list = list(result_cursor)

```

```
# Print the results
flag = True
for i, date in enumerate(date_range):
    existing_data = next((data for data in result_list if data["_id"] == date["day"]), {
        "_id": date["day"],
        "count": 0,
        "avg": 0,
        "stdDev": 0,
        "median": 0,
        "p25": 0,
        "p75": 0
    })

    if flag:
        print('[')
        flag = False

    # Moved the print_result function call here
    print_result(existing_data)

    if i < len(date_range) - 1:
        print(',')

print(']')
import matplotlib.pyplot as plt

# Execute the pipeline
result_cursor = db.PermanentBookings.aggregate(pipeline)
```

```
result_list = list(result_cursor)

# Extract relevant data for plotting
days = [data["_id"] for data in result_list]
avg_values = [data["avg"] for data in result_list]
stdDev_values = [data["stdDev"] for data in result_list]
median_values = [data["median"] for data in result_list]
p25_values = [data["p25"] for data in result_list]
p75_values = [data["p75"] for data in result_list]

# Plotting
plt.figure(figsize=(10, 6))

plt.plot(days, avg_values, label='Average')
plt.plot(days, stdDev_values, label='Standard Deviation')
plt.plot(days, median_values, label='Median')
plt.plot(days, p25_values, label='P25')
plt.plot(days, p75_values, label='P75')

plt.xlabel('Day')
plt.ylabel('Value')
plt.title('Vancouver(Booking) stats on December ')
plt.legend()
plt.grid(True)
plt.show()
```

2.6

a)

```
PB_C2 = db['PermanentBookings']

init_nov_c = '2017-12-04 08:00:00'
final_nov_c = '2017-12-04 09:59:59'
init_nov = time.mktime(datetime.datetime.strptime(init_nov_c, '%Y-%m-%d %H:%M:%S').timetuple())
final_nov = time.mktime(datetime.datetime.strptime(final_nov_c, '%Y-%m-%d %H:%M:%S').timetuple())

recorded = PB_C2.find({"$and": [{"city": 'Vancouver'},
                                {"init_time": {"$gte": init_nov }},
                                {"init_time": {"$lte": final_nov }}]},
                      {'_id': 1, 'city': 1, 'origin_destination': 1})

test = pd.DataFrame(recorded)
resultado = [d.get('coordinates') for d in test['origin_destination']]
destination = np.zeros((len(test), 2))
for i in range(len(test)):
    destination[i, 1] = resultado[i][1][0]
    destination[i, 0] = resultado[i][1][1]

destination_coordinates = pd.DataFrame(destination)
destination_coordinates.rename(columns={0: "latitude", 1: "longitude"}, inplace=True)
destination_coordinates.to_csv(r'C:\Users\ASUS\destinationn.csv', index=False, header=True)

df = pd.read_csv("destinationn.csv")
geometry1 = [Point(xy) for xy in zip(df["longitude"], df["latitude"])]
geo_df1 = gpd.GeoDataFrame(df, geometry=geometry1)
```

```
# Reading data from the shapefile of Vancouver
Vancouver_map = gpd.read_file(r'D:\Program Files (x86)\dlgame\local-area-boundary\local-area-boundary.shp')
Vancouver_map.to_crs(epsg=4326, inplace=True)

# Plotting the Vancouver map
fig, ax = plt.subplots(figsize=(10, 10))
Vancouver_map.plot(ax=ax, color='lightgrey', edgecolor='black')

# Plotting the parking locations
geo_df1.plot(ax=ax, markersize=10, color='red', alpha=0.7)

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title(" Vancouver Parkings on December 4th 2017 from 8 to 10 o'clock ")
plt.show()
```

b)

```
tzVancouver=-8;  #Time zone shift for
time_week = 7*24*60*60
init_nov_c = '2017-12-01 00:00:00'
final_nov_c = '2017-12-01 23:59:59'
init_nov = (time.mktime(datetime.datetime.strptime(init_nov_c,'%Y-%m-%d %H:%M:%S').timetuple()))
final_nov = (time.mktime(datetime.datetime.strptime(final_nov_c,'%Y-%m-%d %H:%M:%S').timetuple()))

recorded = PP_C2.find({"$and":[{"city":'Vancouver'},
                               {"init_time":{"$gte":init_nov+tzVancouver*60*60}},
                               {"init_time":{"$lte":final_nov+tzVancouver*60*60}}]},
                      {'_id':1, 'city':1,'loc':1})
test = pd.DataFrame(recorded)
resultado = [d.get('coordinates') for d in test['loc']]
parking = np.zeros((len(test),2))
for i in range (len(test)):
    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=True)
parking_coordinates.to_csv (r'C:\Users\User\parkinggg.csv', index = False, header=True)

df = pd.read_csv("parkinggg.csv")
geometry1 = [Point(xy) for xy in zip(df["longitude"], df["latitude"])]

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

Vancouver_map = gpd.read_file(r'E:\ict\mobility\local-area-boundary.shp')
```

```
Vancouver_map.to_crs(epsg = 4326 , inplace = True)
c=0
counter=[]
for i in Vancouver_map['geometry']:
    for j in geo_df1['geometry']:
        if i.contains(j):
            c+=1
    counter.append(c)
    c=0
Vancouver_map['counter']=counter

Vancouver_map.plot(column = 'counter',cmap='viridis',legend = True,figsize=(10,10))
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('vancouver Parkings First of December 2017')


# In[23]:




tzVancouver=-8;  #Time zone shift for
time_week = 7*24*60*60
init_nov_c = '2018-01-01 00:00:00'
final_nov_c = '2018-01-01 23:59:59'
init_nov = (time.mktime(datetime.datetime.strptime(init_nov_c,'%Y-%m-%d %H:%M:%S').timetuple()))
final_nov = (time.mktime(datetime.datetime.strptime(final_nov_c,'%Y-%m-%d %H:%M:%S').timetuple()))
```

```

recorded = PP_C2.find({"$and":[{"city":'Vancouver'},
                               {"init_time":{"$gte":init_nov+tzVancouver*60*60}},
                               {"init_time":{"$lte":final_nov+tzVancouver*60*60}}]},
                      {'_id':1, 'city':1,'loc':1})
test = pd.DataFrame(recorded)
resultado = [d.get('coordinates') for d in test['loc']]
parking = np.zeros((len(test),2))
for i in range (len(test)):
    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=True)
parking_coordinates.to_csv (r'C:\Users\User\parkingg.csv', index = False, header=True)

df = pd.read_csv("parkingg.csv")
geometry1 = [Point(xy) for xy in zip(df["longitude"], df["latitude"])]

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

Vancouver_map = gpd.read_file(r'E:\ict\mobility\local-area-boundary.shp')
Vancouver_map.to_crs(epsg = 4326 , inplace = True)
c=0
counter=[]
for i in Vancouver_map['geometry']:
    for j in geo_df1['geometry']:
        if i.contains(j):
            c+=1
```

```
    counter.append(c)
    c=0
Vancouver_map['counter']=counter

Vancouver_map.plot(column = 'counter',cmap='viridis',legend = True,figsize=(10,10))
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('vancouver Parkings First of January 2018')


# In[25]:



tzVancouver=-8;  #Time zone shift for
time_week = 7*24*60*60
init_nov_c = '2017-11-01 00:00:00'
final_nov_c = '2017-11-01 23:59:59'
init_nov = (time.mktime(datetime.datetime.strptime(init_nov_c,'%Y-%m-%d %H:%M:%S').timetuple()))
final_nov = (time.mktime(datetime.datetime.strptime(final_nov_c,'%Y-%m-%d %H:%M:%S').timetuple()))

recorded = PP_C2.find({"$and":[{"city":'Vancouver'},
                               {"init_time":{"$gte":init_nov+tzVancouver*60*60}},
                               {"init_time":{"$lte":final_nov+tzVancouver*60*60}}]},
                      {'_id':1, 'city':1,'loc':1})
test = pd.DataFrame(recorded)
resultado = [d.get('coordinates') for d in test['loc']]
parking = np.zeros((len(test),2))
for i in range (len(test)):
    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=True)
parking_coordinates.to_csv (r'C:\Users\User\parkingg.csv', index = False, header=True)

df = pd.read_csv("parkingg.csv")
geometry1 = [Point(xy) for xy in zip(df["longitude"], df["latitude"])]

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

Vancouver_map = gpd.read_file(r'E:\ict\mobility\local-area-boundary.shp')
Vancouver_map.to_crs(epsg = 4326 , inplace = True)
c=0
counter=[]
for i in Vancouver_map['geometry']:
    for j in geo_df1['geometry']:
        if i.contains(j):
            c+=1
    counter.append(c)
    c=0
Vancouver_map['counter']=counter

Vancouver_map.plot(column = 'counter',cmap='viridis',legend = True,figsize=(10,10))
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('vancouver Parkings First of November 2017')
```