ICT in transport systems

LAB REPORT 1
Group 1

Bobonazarov Abdurasul S263771
Yu Cheng S288483
Khamidov Davron S273331

January 2022

1. Preliminary data analysis

- **How many documents are present in each collection?**

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

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

When someone want to use a car. First they should book car and after using they should park the car. The system stores booking in PermanentBooking collection then it will stores parking in PermanentParking. In system may happen some errors or maintenance issues. And because of this number of documents in PermanentParking and PermanentBooking is not exactly same

- **For which cities the system is collecting data?**

There are 28 cities in collections: ”Wien”, ”Washington DC”, ”Vancouver”, ”Twin Cities”, ”Toronto”, ”Torino”, ”Stuttgart”, ”Seattle”, ”San Diego”, ”Roma”, ”Rheinland”, ”Portland”, ”New York City”, ”Munchen”, ”Montreal”, ”Milano”, ”Madrid”, ”Hamburg”, ”Frankfurt”, ”Firenze”, ”Denver”, ”Columbus”, ”Calgary”, ”Berlin”, ”Austin”, ”Amsterdam”, ”Bologna”, ”Catania”

- **When the collection started? When the collection ended?**

Started time for Car2Go collection: 2016-12-13 17:38:23
Ended time for Car2Go collection: 2018-01-31 13:11:33
Started time for Enjoy collection: 2017-05-05 15:06:21
Ended time for Enjoy collection: 2019-06-10 17:16:20

- **What about the timezone of the timestamps?**

There are four fields related to date: “init_time”, “final_time”, “init_date”, “final_date”. “init_time”, “final_time” are unix timestamps and they are related to GMT (Greenwich Mean Time). “init_date”, “final_date” are in human readable format and refers to local time zone.

First booking in Car2Go with local time 2016-12-13 09:38:23 in Vancouver.

First booking in Enjoy with local time 2017-05-05 17:06:21 in Vancouver.

- **What is the total number of cars seen in the whole period in each city? How does this relate to the fleet size at a given time?**

We will analyze the cities of New York City and Milano in Car2Go and Enjoy dataset, for each city we obtain the following numbers: in Car2Go dataset: 1050 cars in New York City, 1153 cars in Milano. In Enjoy dataset: 1870 cars in Milano. In total, there are 1050 cars in New York City and 3023 cars in Milano.

- **How many bookings have been recorded on the November 2017 in each city?**

In Car2go dataset, there are 53720 bookings in New York City and 183197 bookins in Milano.

In Enjoy dataset, there are 132269 bookings in Milano

- **How many bookings have also the alternative transportation modes recorded in each city?**

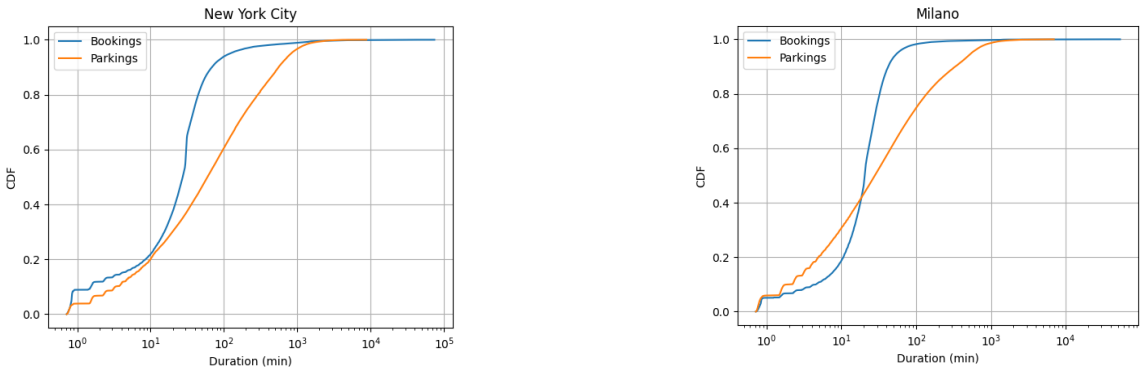
In Car2go dataset there are not any bookings with alternative transportation mode in New York City while in Milano there are 729171 bookings. In Enjoy dataset there are 826683 bookings with alternative transportation mode in Milano.

2. Analysis of the data

In this part only the cities of New York and Milano will be considered for Car2Go database. Milano for Enjoy database will not be considered because there are no data on Enjoy dataset on analyzed period. The analyzed period goes from 1st January 2017 to 28th February 2017.

Cumulative distribution function

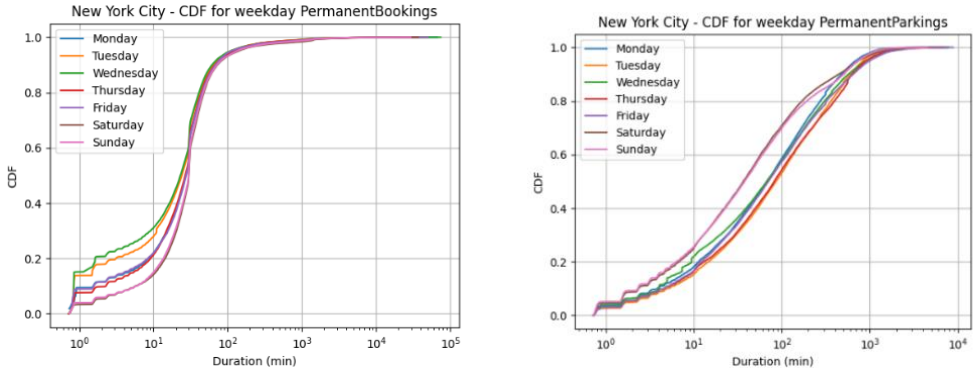
The evaluation of the Cumulative Distribution Functions of booking and parking using the Car2Go database is shown in Figure 1. For both city, parking durations are longer than booking durations. For example, 80 percent of the bookings last less than 60 min, whereas 50 percent of parking last less than 60 min. Also, Outliers are possible because these CDFs are not filtered. Nearly 15% of bookings have a duration of less than 1,2 minutes, as shown at the start of the CDF. A booking/parking is also unlikely to last more than a few hours, as shown at the end of the CDF. So, bookings lasting less than 3 minutes and more than 180 are considered outliers. For Milano and New York, the CDF of bookings shows a similar pattern. On the other hand, the distribution of parking in Milano shows that over 30-35% of them are for less than 10 minutes. This means that in Milano there is a higher usage of the system on average. So cars are rarely parked for long periods of time.



(a) New York (b) Milano

Figure 1: Cumulative Distribution Functions of bookings and parking

Figure 2 shows the CDF per weekday for both cities and for both bookings and parking durations. Outliers affects the duration because these results are not filtered yet. Even so, we've noticed that weekend and weekday bookings and parking are different. Where parking lasts significantly longer and bookings are slightly less. This trend can be explained by the fact that there are fewer bookings on Sunday, therefore more automobiles are left parked for longer periods of time; at the same time, there is less traffic, so driving takes less time.

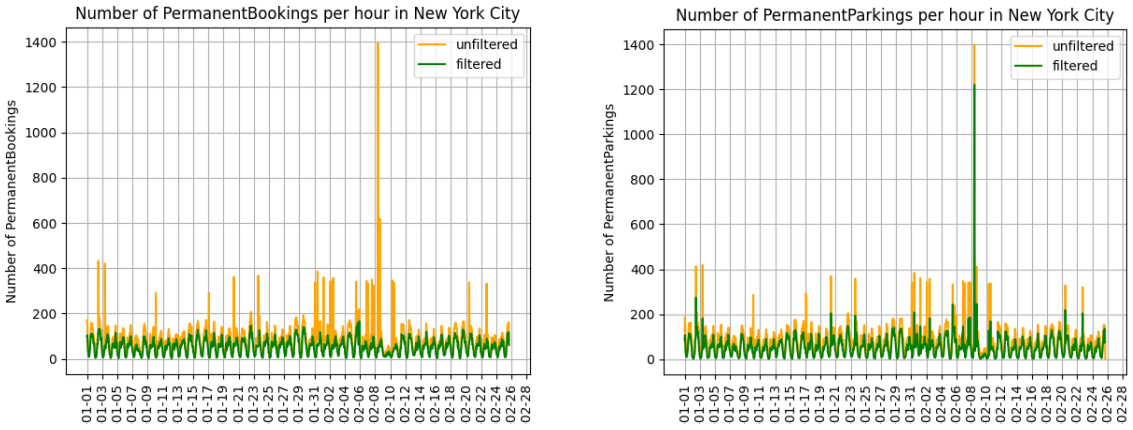


(a) New York Booking (b) New York Parking

Figure 2. Cumulative Distribution Functions of bookings and parking per weekday

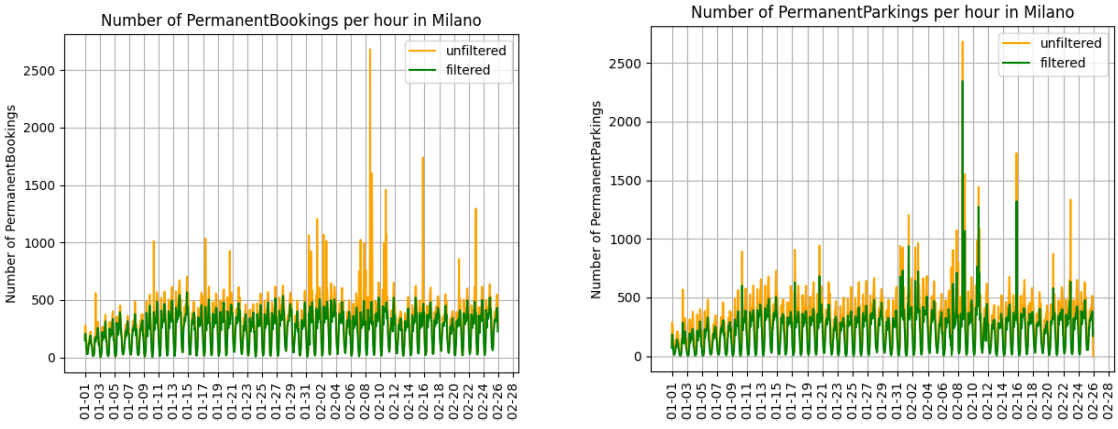
System Utilization Over Time

The system's utilization was analyzed by aggregating bookings and parking by hour of the day. The obtained results for New York and Milano are shown in Figures 3 and 4, respectively. Unfiltered data plotted with yellow lines. First of all, we can see that there are no more data in the days after February 25th. Figures 3 and 4 show that there are some strange peaks in both booking and parking on 8th February. We can consider this as outliers. It can also be seen from the figures that carsharing usage in Milan is higher than in New York.



(a) (b)

Figure 3. Number of booked and parked cars per hour of the day in New York



(a) (b)

Figure 4. Number of booked and parked cars per hour of the day in Milano

Filtering criteria

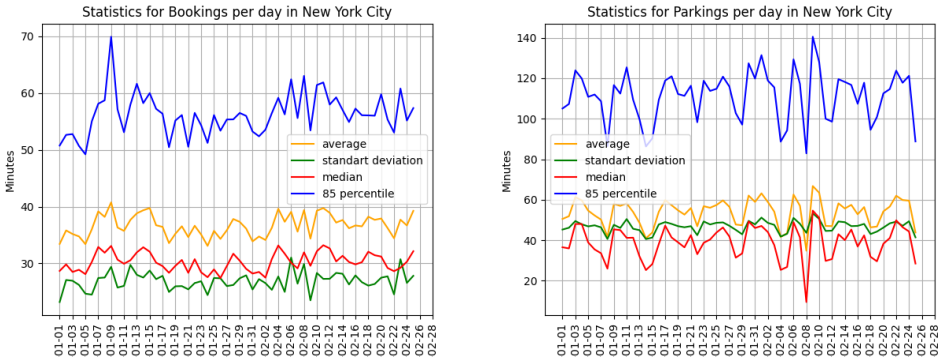
Some outliers are present in the database. As a result, bookings or parking reservations may last longer or shorter than normal behavior. The following filtering criteria were used to filter some outliers:

- Short booking periods: Bookings with rental times of less than 3 minutes are deleted.
- Long booking periods: Bookings with rental times of more than 180 minutes are deleted.
- Invalid rentals: Rentals with same origin and destination coordinates are deleted
- Same criteria applied for parking also.

Result of filtering shown in Figure 3 and 4 with green lines. In Figure 3(a) and 4(a) We see that filtering of bookings data was successful. The graph was smoother after applying filtering. on the other hand, the filtering process, is less visible in parking. Figure 3(b) and 4(b) still presents some peaks.

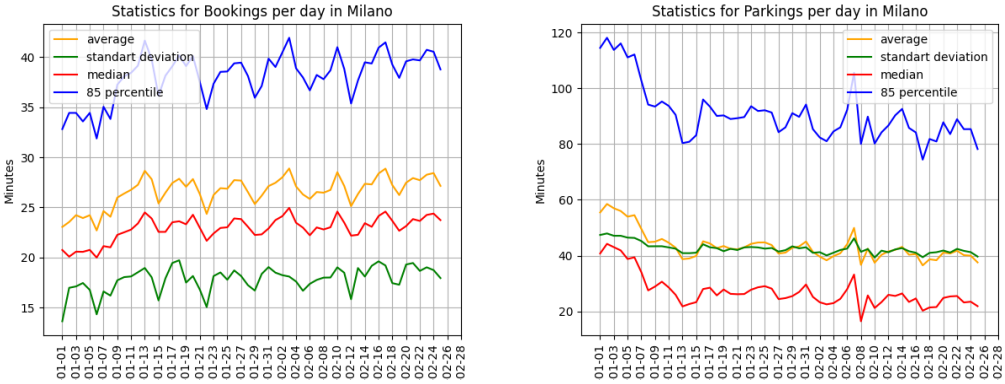
General Statistics of the Filtered Data

After the data has been filtered, averages, medians, 90th percentiles, and standard deviations are calculated for the bookings/parkings in the two cities. Figure 5 represents statistics for New York. Figure 6 represents statistics for Milano. When compared to Milano, New York has the higher average booking duration. As a result of filtering, in the booking graphs, the median and average values appear to be similar once more, and they also follow the same trend. We notice that the standard deviation is lower in the bookings graph than parking graphs. It means that applied filtering was more effective in booking rather than parking data. Also, a periodic pattern with a downward peak on weekends can be seen.



(a) (b)

Figure 5. Statistics for New York



(a) (b)

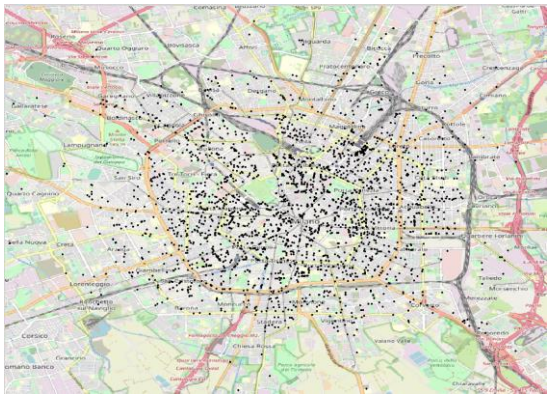
Figure 6. Statistics for Milano

Parking Density, Heat-map and OD Matrix

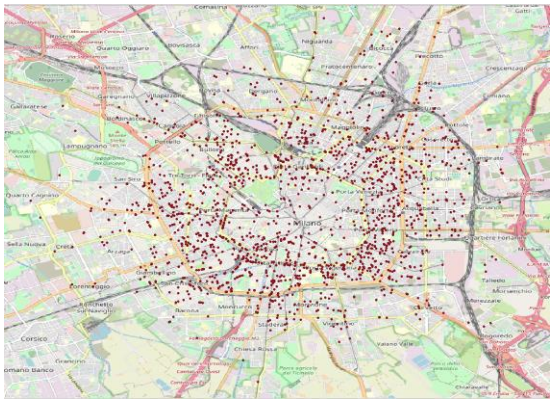
In this section the position of the new parking is plotted on the map of Milano. For this analysis the software QGIS has been used for data visualization. The car positions were obtained from the database and loaded as CSV files into the software. The parking locations were plotted for two different periods of the day (Figure 7,8): from 6:00 to 18:00 and from 18:00 to 6:00, and for two reference days: a holiday (January 10th) and a working day (January 14th) to compare two different situations. It can be noticed that there are more new parkings during the weekend in general, which may be explained by the fact that users use car sharing more during the weekend.

Furthermore, the presence of new parking is higher in both situations between 18:00 and 6:00, demonstrating that the car sharing system is more popular during this time.

To compute the density of cars parked in the city, the city's territory has been divided into squares of approximately 500m x 500m using a conversion of longitude and latitude variations in meters, and the density of cars per area is demonstrated in Figure 9. The results were achieved using the Python gmplot package.

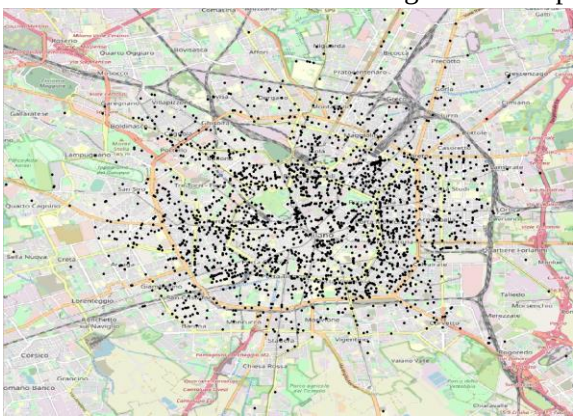


(a) 10th January (6:00-18:00)

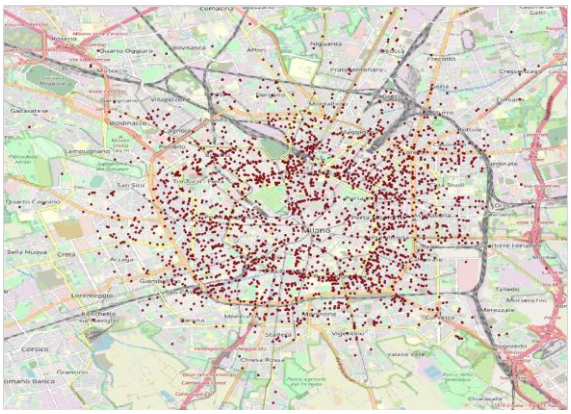


(b) 10th, 11th January (18:00-6:00)

Figure 7. New parkings during the week

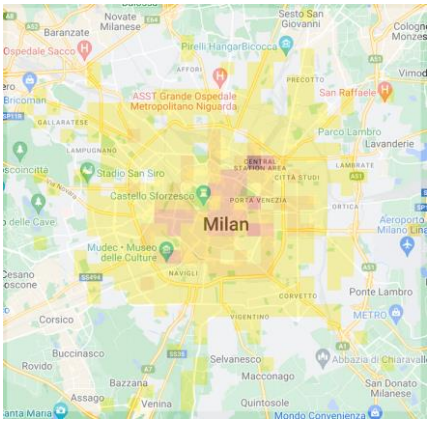


(a) 14th January (6:00-18:00)



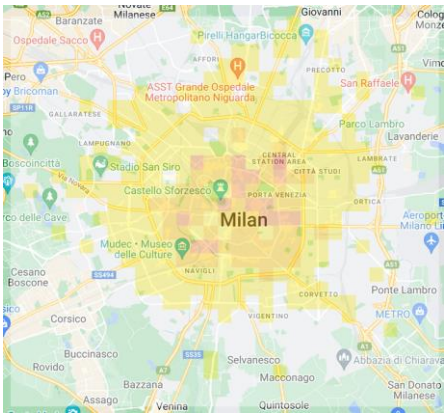
(b) 14th, 15th January (18:00-6:00)

Figure 8. New parkings during the weekend

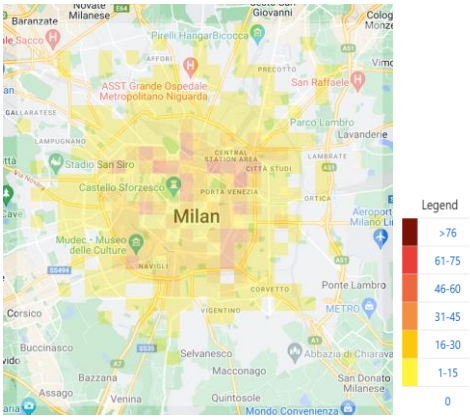


(a)Weekday (6:00-18:00)

(b) Weekday (18:00-6:00)



(c)Weekend (6:00-18:00)



(c) Weekend (18:00-6:00)

Figure 9 Heatmap of parked cars in Milano

Finally, the O-D matrix has been evaluated for the bookings from the same areas obtained with the square grid. As shown in Figure 10, the zones are numbered from 0 to 400, beginning in the top left corner and ending in the lower right corner of the considered region. It can be observed that the majority of the trips started and ended around the city center, since there is a higher concentration in the central part of the matrix, which corresponds to the central zones. This is reasonable because long trips with the Car Sharing system are inconvenient in terms of cost.

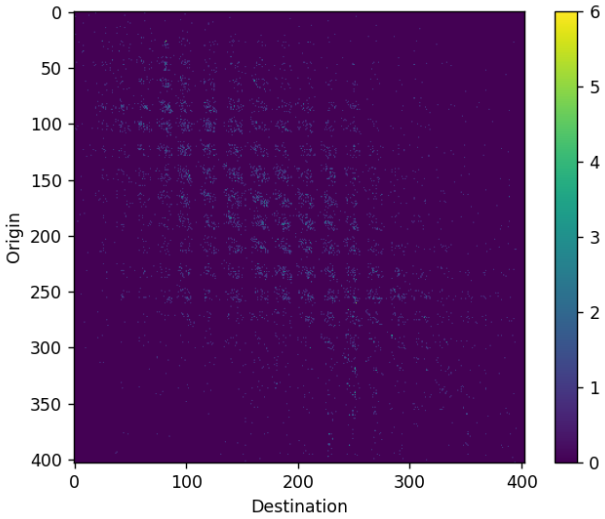


Figure 10. O-D Matrix

APPENDIX

A Step 1

A.1 Connection to Database

```
import csv
import math
from math import pi, floor, cos

import pymongo as pm
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import gmplot
from mne_connectivity.viz import plot_connectivity_circle

COLLECTION_NAMES = ['ActiveBookings', 'ActiveParkings', 'PermanentBookings',
'PermanentParkings', 'enjoy_ActiveBookings', 'enjoy_ActiveParkings', 'enjoy_PermanentBookings',
'enjoy_PermanentParkings']

CITIES = ['New York City', 'Milano']
CITIES_ENJOY = 'Milano'

start_date = datetime(2017, 1, 1, 0, 0, 0)
end_date = datetime(2017, 2, 28, 23, 59, 59)

collection_car2go = ["PermanentBookings", "PermanentParkings"]
collection_enjoy = ['enjoy_PermanentBookings', 'enjoy_PermanentParkings']

def setup_connection():
    client = pm.MongoClient('bigdatadb.polito.it',
                            ssl=True,
                            authSource='carsharing',
                            tlsAllowInvalidCertificates=True)
    db_car_sharing = client['carsharing']
    db_car_sharing.authenticate('ictts', 'Ictts16!')
    return db_car_sharing
```

A.2 How many documents are present in each collection?

```
def count_collections(db):
    for name in COLLECTION_NAMES:
        collection_count = str(db.get_collection(name).estimated_document_count())
        print(name + ": " + collection_count)
```

A.3 For which cities the system is collecting data?

```
def print_cities(db):
    cities = db.get_collection('PermanentBookings').distinct('city')
    enjoy_cities = db.get_collection('enjoy_PermanentBookings').distinct('city')
    print(f'Cities: {cities}')
    print(f'Enjoy_Cities: {enjoy_cities}')
```

A.4 When the collection started? When the collection ended?

```
def duration_of_data(db):
    timestamp_enj = db.get_collection('enjoy_PermanentBookings').find().sort('init_time',
pm.ASCENDING).distinct('init_time')
    start_timestamp_enj = int(timestamp_enj[-1])
    end_timestamp_enj = int(timestamp_enj[0])

    timestamp = db.get_collection('PermanentBookings').find().sort('init_time',
pm.ASCENDING).distinct('init_time')
    start_timestamp = int(timestamp[-1])
    end_timestamp = int(timestamp[0])

    print("Car2Go collection started time {}".format
          (datetime.utcfromtimestamp(start_timestamp).strftime('%Y-%m-%d %H:%M:%S')))
    print("Car2Go collection ended time {}".format(
        datetime.utcfromtimestamp(end_timestamp).strftime('%Y-%m-%d %H:%M:%S')))
    print("Enjoy collection started time {}".format(
        datetime.utcfromtimestamp(start_timestamp_enj).strftime('%Y-%m-%d %H:%M:%S')))
    print("Enjoy collection ended time {}".format(
        datetime.utcfromtimestamp(end_timestamp_enj).strftime('%Y-%m-%d %H:%M:%S')))

    first_timestamp = db.get_collection('PermanentBookings').find_one({'init_time':
start_timestamp})
    date = first_timestamp.get('init_date')
    city = first_timestamp.get('city')
    print("Car2Go local time of first timestamp {} in {}".format(date, city))

    first_timestamp_enj = db.get_collection('enjoy_PermanentBookings').find_one({'init_time':
start_timestamp_enj})
    date_enj = first_timestamp_enj.get('init_date')
    city_enj = first_timestamp.get('city')
    print("Enjoy local time of first timestamp {} in {}".format(date_enj, city_enj))
```

A.5 What is the total number of cars seen in the whole period in each city?

```
def amount_of_cars(db):
    # car2go
    for city in CITIES:
        cars = db.get_collection('PermanentBookings').distinct('plate', {'city': city})
        print('Car2Go cars in {}: {}'.format(city, len(cars)))
    # enjoy
    cars_enjoy = db.get_collection('enjoy_PermanentBookings').distinct('plate', {'city':
CITIES_ENJOY})
    print('Enjoy cars in {}: {}'.format(CITIES_ENJOY, len(cars_enjoy)))
```

A.6 How many bookings have been recorded on the November 2017 in each city?

```
def bookings_on_november(db):
    start_time = datetime(2017, 11, 1, 0, 0, 0)
    end_time = datetime(2017, 11, 30, 23, 59, 59)
    for city in CITIES:
        total_bookings = db.get_collection('PermanentBookings').count_documents({'$and': [
            {'city': city},
            {'init_date': {'$gte': start_time}},
            {'final_date': {'$lte': end_time}}
        ]})
        print('Car2go bookings in {}: {}'.format(city, total_bookings))

    total_bookings_enjoy =
db.get_collection('enjoy_PermanentBookings').count_documents({'$and': [
        {'city': CITIES_ENJOY},
        {'init_date': {'$gte': start_time}},
        {'final_date': {'$lte': end_time}}
    ]})
    print('Enjoy bookings in {}: {}'.format(CITIES_ENJOY, total_bookings_enjoy))
```

A.7 How many bookings have also the alternative transportation modes recorded in each city?

```
def alternatives(db):
    for city in CITIES:
        total = db.get_collection('PermanentBookings').count_documents(
            {'$and': [{'city': city},
                      {'$or': [
                          {'walking.distance': {'$ne': -1}},
                          {'driving.distance': {'$ne': -1}},
                          {'public_transport.distance': {'$ne': -1}},
                      ]
                      }
                      ]
             }
        )
        print('Car2go alternatives in {}: {}'.format(city, total))
    total_enj = db.get_collection('enjoy_PermanentBookings').count_documents(
        {'$and': [{'city': CITIES_ENJOY},
                  {'$or': [
                      {'walking.distance': {'$ne': -1}},
                      {'driving.distance': {'$ne': -1}},
                      {'public_transport.distance': {'$ne': -1}},
                  ]
                  }
                  ]
         }
    )
    print('Enjoy alternatives in {}: {}'.format(CITIES_ENJOY, total_enj))
```

B Step 2

B.1 Cumulative distribution function

```
def pipline_durations(db, collection, city, start_date, end_date):
    return list(db.get_collection(collection).aggregate(
        [
            {  # filter data for a given city in a given period
                '$match': {
                    '$and': [
                        {'city': city},
                        {'init_date': {'$gte': start_date}},
                        {'final_date': {'$lte': end_date}}
                    ]}
            },
            {  # calculate duration of booking and parking: duration - (final_time-init_time)/60
                '$project': {
                    '_id': 1,
                    'city': 1,
                    'duration': {'$divide': [
                        {
                            '$subtract': ['$final_time', '$init_time']
                        }, 60
                    ]
                    }
                }
            },
            {  # group by duration and calculate sum of unique durations
                '$group': {
                    '_id': '$duration',
                    'total_rentals': {'$sum': 1}
                }
            },
            {  # sort by id(duration)
                '$sort': {'_id': 1}
            }
        ]
    ))


def pipline_durations_days(db, collection, city, start_date, end_date, day):
    return list(db.get_collection(collection).aggregate(
        [
            {  # filter data for a given city in a given period
                '$match': {
                    '$and': [
                        {'city': city},
                        {'init_date': {'$gte': start_date}},
                        {'final_date': {'$lte': end_date}}
                    ]}
            },
            {  # calculate duration of booking and parking: duration - (final_time-init_time)/60 | added
```

```
weekday
            '$project': {
                '_id': 1,
                'city': 1,
                'weekday': {'$isoDayOfWeek': '$init_date'},
                'duration': {'$divide': [
                    {
                        '$subtract': ['$final_time', '$init_time']
                    }, 60
                ]
                }
            }
        },
        {  # select by given weekday
            '$match': {'weekday': day}
        },
        {  # group by duration and calculate sum of unique durations
            '$group': {
                '_id': '$duration',
                'total_rentals': {'$sum': 1}
            }
        },
        {  # sort by id(duration)
            '$sort': {'_id': 1}
        }
    ]
))

def cdf(db):
    # no enjoy data in Milano
    for city in CITIES:
        for collection in collection_car2go:
            # get duration and total rent
            durations = pipline_durations(db, collection, city, start_date, end_date)

            # create list of id and total rent
            duration = [item['_id'] for item in durations]
            totrentals = [item['total_rentals'] for item in durations]

            # sum of total rents
            sum_totalrentals = sum(totrentals)

            # calc cumulative total rents
            cumulative_totrentals = [totrentals[0]]
            [cumulative_totrentals.append(cumulative_totrentals[-1] + totrentals[i]) for i in
             range(1, len(totrentals))]
            cumulative_totrentals = [item / sum_totalrentals for item in cumulative_totrentals]

            plt.plot(duration, cumulative_totrentals, label=collection[len(collection) - 8:len(collection)])
            plt.xscale('log')
            plt.xlabel('Duration (min)')
            plt.ylabel('CDF')
```

```
        plt.title(city)
        plt.grid()
        plt.legend(loc="upper left")
        plt.show()

    days = [1, 2, 3, 4, 5, 6, 7]
    days_label = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
    for city in CITIES:
        for collection in collection_car2go:
            for day in days:
                durations_day = pipline_durations_days(db, collection, city, start_date, end_date, day)

                # create list of id and total rent
                duration = [item['_id'] for item in durations_day]
                totrentals = [item['total_rentals'] for item in durations_day]

                # sum of total rents
                sum_totalrentals = sum(totrentals)

                # calc cumulative total rents
                cumulative_totrentals = [totrentals[0]]
                [cumulative_totrentals.append(cumulative_totrentals[-1] + totrentals[i]) for i in
                 range(1, len(totrentals))]
                cumulative_totrentals = [item / sum_totalrentals for item in cumulative_totrentals]

                plt.plot(duration, cumulative_totrentals, label=collection[len(collection) -
8:len(collection)])
                plt.xscale('log')
                plt.xlabel('Duration (min)')
                plt.ylabel('CDF')
            plt.title(f'{city} - CDF for weekday {collection}')
            plt.grid()
            plt.legend(days_label, loc="upper left")
            plt.show()
```

B.2 System Utilization Over Time

```
def pipline_cars_per_hour_unfiltered(db, collection, city, start_date, end_date):
    return list(db.get_collection(collection).aggregate(
        [
            {  # filter data for a given city in a given period
                '$match': {
                    '$and': [
                        {'city': city},
                        {'init_date': {'$gte': start_date}},
                        {'final_date': {'$lte': end_date}}
                    ]}
            },
            {  # convert init_date to part of date like:
                # date_parts: {{year: 2017}, {month: 1}, {day: 1}, {hour: 11}, {minute: 20}}
                '$project': {
                    '_id': 1,
```

```
                'city': 1,
                'date_parts': {'$dateToParts': {'date': '$init_date'}}
            }
        },
        {  # group by month,day,hour and calculate sum
            '$group': {
                '_id': {
                    'month': '$date_parts.month',
                    'day': '$date_parts.day',
                    'hour': '$date_parts.hour'
                },
                'total_rentals': {'$sum': 1}
            }
        },
        {  # sort by id(month,day,hour)
            '$sort': {'_id': 1}
        }
    ]
))


def pipline_cars_per_hour_filtered(db, collection, city, start_date, end_date):
    condition_booking = {
        '$cond': [
            {'$ne': ['$origin_destination', 'undefined']},
            {'$ne': [  # match the coordinates
                {'$arrayElemAt': ['$origin_destination.coordinates', 0]},
                {'$arrayElemAt': ['$origin_destination.coordinates', 1]}
            ]},
            True
        ]
    }
    condition_parking = {
        '$cond': [  # condition for identifying is car moved or not
            {'$ne': ['$loc', 'undefined']},
            {'$ne': [  # match the coordinates
                {'$arrayElemAt': ['$loc.coordinates', 0]},
                {'$arrayElemAt': ['$loc.coordinates', 1]}
            ]},
            True
        ]
    }
    # choice correct condition according to collection
    condition = condition_booking if collection == 'PermanentBookings' or collection ==
'enjoy_PermanentBookings' else condition_parking

    return list(db.get_collection(collection).aggregate(
        [
            {  # filter data for a given city in a given period
                '$match': {
                    '$and': [
                        {'city': city},
```

```
                {'init_date': {'$gte': start_date}},
                {'final_date': {'$lte': end_date}}
            ]}
        },
        {  # convert init_date to part of date like:
            # date_parts: [{year: 2017}, {month: 1}, {day: 1}, {hour: 11}, {minute: 20}]
            # moved status
            # calculate duration of booking and parking: duration - (final_time-init_time)/60 | added
weekday
            '$project': {
                '_id': 1,
                'city': 1,
                'moved': condition,
                'duration': {
                    '$divide': [
                        {
                            '$subtract': ['$final_time', '$init_time']
                        },
                        60
                    ]
                },
                'date_parts': {'$dateToParts': {'date': '$init_date'}}
            }
        },
        {  # filter the data
            '$match': {
                '$and': [
                    {'duration': {'$gte': 3}},
                    {'duration': {'$lte': 180}},
                    {'moved': True}
                ]
            }
        },
        {  # calculate sum of rentals per hour on filtered data
            '$group': {
                '_id': {
                    'month': '$date_parts.month',
                    'day': '$date_parts.day',
                    'hour': '$date_parts.hour'
                },
                'total_rentals': {'$sum': 1}
            }
        },
        {  # sort by id(month,day,hour)
            '$sort': {'_id': 1}
        }
    ]
))

def system_utilization(db):
    date_by_day = period_of_date()
```

```
    for city in CITIES:
        for collection in collection_car2go:
            # get unfiltered data
            car_per_hour_unfiltired = pipline_cars_per_hour_unfiltered(db, collection, city, start_date,
end_date)

            date_by_hour_unfiltered = [
                datetime(2017, item['_id']['month'], item['_id']['day']) + timedelta(hours=item['_id']['hour']) for
                item in car_per_hour_unfiltired]
            totrentals_unfiltired = [item['total_rentals'] for item in car_per_hour_unfiltired]

            plt.plot(date_by_hour_unfiltered, totrentals_unfiltired, color='orange', label='unfiltered')

            # get filtered data
            car_per_hour_filtired = pipline_cars_per_hour_filtered(db, collection, city, start_date,
end_date)

            date_by_hour_filtered = [
                datetime(2017, item['_id']['month'], item['_id']['day']) + timedelta(hours=item['_id']['hour']) for
                item in car_per_hour_filtired]
            totrentals_filtired = [item['total_rentals'] for item in car_per_hour_filtired]

            plt.plot(date_by_hour_filtered, totrentals_filtired, color='green', label='filtered')

            plt.xticks(date_by_day, rotation=90)
            plt.xlabel('Day of January and February 2017')
            plt.ylabel('Number of {}'.format(collection))
            plt.grid()
            plt.legend()
            plt.title('Number of {} per hour in {}'.format(collection, city))
            plt.show()
```

B.3 General Statistics of the Filtered Data

```
def pipline_avg_duration_per_day(db, collection, city, start_date, end_date):
    condition_booking = {
        '$cond': [
            {'$ne': ['$origin_destination', 'undefined']},
            {'$ne': [
                {'$arrayElemAt': ['$origin_destination.coordinates', 0]},
                {'$arrayElemAt': ['$origin_destination.coordinates', 1]}
            ]},
            True
        ]
    }
    condition_parking = {
        '$cond': [
            {'$ne': ['$loc', 'undefined']},
            {'$ne': [
                {'$arrayElemAt': ['$loc.coordinates', 0]},
                {'$arrayElemAt': ['$loc.coordinates', 1]}
            ]},
```

```
            True
        ]
    }

    condition = condition_booking if collection == 'PermanentBookings' or collection ==
'enjoy_PermanentBookings' else condition_parking

    return list(db.get_collection(collection).aggregate(
        [
            {  # filter data for a given city in a given period
                '$match': {
                    '$and': [
                        {'city': city},
                        {'init_date': {'$gte': start_date}},
                        {'final_date': {'$lte': end_date}}
                    ]}
            },
            {  # convert init_date to part of date like:
                # date_parts: [{year: 2017}, {month: 1}, {day: 1}, {hour: 11}, {minute: 20}]
                # moved status
                # calculate duration of booking and parking: duration = (final_time-init_time)/60 | added
weekday
                '$project': {
                    '_id': 1,
                    'city': 1,
                    'moved': condition,
                    'duration': {
                        '$divide': [
                            {
                                '$subtract': ['$final_time', '$init_time']
                            },
                            60
                        ]
                    },
                    'date_parts': {'$dateToParts': {'date': '$init_date'}}
                }
            },
            {  # filter the data
                '$match': {
                    '$and': [
                        {'duration': {'$gte': 3}},
                        {'duration': {'$lte': 180}},
                        {'moved': True}
                    ]
                }
            },
            {  # calculate average and standard deviation duration per day on filtered data
                '$group': {
                    '_id': {
                        'month': '$date_parts.month',
                        'day': '$date_parts.day'
                    },
```

```
                'avg_duration': {'$avg': '$duration'},
                'std_duration': {'$stdDevPop': '$duration'},
                'durations': {'$push': '$duration'}
            }
        },
        {  # sort by id(month,day)
            '$sort': {'_id': 1}
        }
    ]
  ))
def rental_statistics(db):
  date_by_day = period_of_date()

  for city in CITIES:
    for collection in collection_car2go:
      avg_per_day = pipline_avg_duration_per_day(db, collection, city, start_date, end_date)

      date_duration = [datetime(2017, item['_id']['month'], item['_id']['day']) for item in
avg_per_day]
      avg_duration = [item['avg_duration'] for item in avg_per_day]
      std_duration = [item['std_duration'] for item in avg_per_day]
      median_duration = [np.median(item['durations']) for item in avg_per_day]
      percentile_duration = [np.percentile(item['durations'], 85) for item in avg_per_day]

      plt.figure()
      plt.plot(date_duration, avg_duration, color='orange', label='average')
      plt.plot(date_duration, std_duration, color='green', label='standart deviation')
      plt.plot(date_duration, median_duration, color='red', label='median')
      plt.plot(date_duration, percentile_duration, color='blue', label='85 percentile')

      plt.xticks(date_by_day, rotation=90)
      plt.xlabel('Day of January and February 2017')
      plt.ylabel('Minutes')
      plt.grid()
      plt.legend()
      plt.title('Statistics for {} per day in {}'.format(collection[-8:], city))
      plt.show()
```

B.4 Parking Density, Heat-map and OD Matrix

B.4.1 Parking Density

```
def pipline_parking_per_hour(db, collection, city, start_date, end_date):
  return list(db.get_collection(collection).aggregate(
    [
      {  # filter data for a given city in a given period
        '$match': {
          '$and': [
            {'city': city},
            {'init_date': {'$gte': start_date}},
            {'final_date': {'$lte': end_date}}
          ]}
```

```
            },
            {
                "$project": {
                    "_id": 1,
                    "init_long": {"$arrayElemAt": [
                        "$loc.coordinates", 0
                    ]},
                    "init_lat": {"$arrayElemAt": [
                        "$loc.coordinates", 1
                    ]},
                    "final_long": {"$arrayElemAt": [
                        "$loc.coordinates", 0
                    ]},
                    "final_lat": {"$arrayElemAt": [
                        "$loc.coordinates", 1
                    ]},
                    "moved": {
                        "$ne": [
                            {"$arrayElemAt": ["$loc.coordinates", 0]},
                            {"$arrayElemAt": ["$loc.coordinates", 1]}
                        ]
                    },
                    "duration": {
                        "$divide": [
                            {
                                "$subtract": ["$final_time", "$init_time"]
                            },
                            60
                        ]
                    }
                }
            },
            {  # filter the data
                '$match': {
                    '$and': [
                        {'duration': {'$gte': 3}},
                        {'duration': {'$lte': 180}},
                        {'moved': True}
                    ]
                }
            }
        ]
    ))
def rental_location(db):
    # weekday 6-18 18-6 weekend day 6-18 18-6
    dates = [
        {"period": "weekday_morning", "time": [datetime(2017, 1, 10, 6, 0, 0), datetime(2017, 1, 10, 18, 0,
0)]},
        {"period": "weekday_evening", "time": [datetime(2017, 1, 10, 18, 0, 0), datetime(2017, 1, 11, 6, 0,
0)]},
        {"period": "weekend_morning", "time": [datetime(2017, 1, 14, 6, 0, 0), datetime(2017, 1, 14, 18, 0,
0)]},
```

```
    {"period": "weekend_evening", "time": [datetime(2017, 1, 14, 18, 0, 0), datetime(2017, 1, 15, 6, 0,
0)]}
  ]

  for date in dates:
    parking_per_hour = pipline_parking_per_hour(db, collection_car2go[1], 'Milano',
date["time"][0],
                                        date["time"][1])
    for item in parking_per_hour:
      print(item)

    with open(f'{date["period"]}_parking.csv', 'w') as fp:
      fp.write('latitude,longitude')
      for item in parking_per_hour:
        fp.write(
          '\n{},{}'.format(item['init_lat'], item['init_long']))
```

B.4.2 Heat-map

```
def calculate_zones(parking, ):
  df = pd.read_csv(parking + '.csv')

  # Definition of the square grid of 500mx500m
  longitude_variation = 0.006358  # East variation of 500 m long
  latitude_variation = 0.004495  # North variation of 500 m lat

  min_longitude = min(df['longitude'])  # weastern longitude e
  max_longitude = max(df['longitude'])  # eastern longitude e
  min_latitude = min(df['latitude'])  # southern latitude n
  max_latitude = max(df['latitude'])  # northern latitude n

  NS_side = math.ceil((max_latitude - min_latitude) / latitude_variation)  # vertical side of the
minimum square cell
  WE_side = math.ceil(
    (max_longitude - min_longitude) / longitude_variation)  # horizontal side of the minimum
square cell

  zone_coordinates = []
  density = np.zeros((NS_side, WE_side), dtype=float)

  with open(f"{parking}_heatmap.csv", 'w') as fp:
    grid_writer = csv.writer(fp, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
    grid_writer.writerow(['latitude', 'longitude', 'count'])

    for i in range(NS_side):
      north_min = (i * latitude_variation) + min_latitude
      north_max = (i * latitude_variation) + latitude_variation + min_latitude

      for j in range(WE_side):
        east_min = (j * longitude_variation) + min_longitude
        east_max = (j * longitude_variation) + longitude_variation + min_longitude
        count = 0  # count - > number of parkings

        for k in range(1, len(df)):
```

```
                if (east_min < float(df['longitude'][k]) < east_max) & (
                        north_min < float(df['latitude'][k]) < north_max):
                    count += 1

            density[i][j] = count
            grid_writer.writerow([north_min, east_min, count])

            coord = [(north_min, east_min), (north_min, east_max), (north_max, east_max),
(north_max, east_min)]
            zone_coordinates.append(coord)

    return zone_coordinates, density
def generate_heatmap(parking, density, zone_coordinates):
    apikey = 'AIzaSyDP_MryIaPjHs2CKssrqFCMl40jkWJ7_jw'

    centre_lat = 45.4773
    centre_long = 9.1815
    gmap = gmplot.GoogleMapPlotter(centre_lat, centre_long, 12, apikey=apikey)
    z = 0

    for i in range(density.shape[0]):
        for j in range(density.shape[1]):
            if density[i][j] == 0:
                color = 'white'
            if 0 < density[i][j] <= 15:
                color = '#fff33b'
            if 15 < density[i][j] <= 30:
                color = '#fdc70c'
            if 30 < density[i][j] <= 45:
                color = '#f3903f'
            if 45 < density[i][j] <= 60:
                color = '#ed683c'
            if 60 < density[i][j] <= 75:
                color = '#e93e3a'
            if density[i][j] > 75:
                color = '#7a1204'
            gate = zip(*zone_coordinates[z])

            gmap.polygon(*gate, color=color, edge_width=0.5, face_alpha=0.4)
            z += 1

    gmap.draw(f'{parking}_densities.html')
    print(f'{parking}_densities.html DONE')
def heatmap(db):
    parking_csv = ['weekday_morning_parking', 'weekday_evening_parking',
'weekend_morning_parking',
                   'weekend_evening_parking']

    for parking in parking_csv:
        zone_coordinates, density = calculate_zones(parking)

        generate_heatmap(parking, density, zone_coordinates)
```

B.4.3 OD Matrix

```
def calculate_zones(parking, ):
    df = pd.read_csv(parking + '.csv')

    # Definition of the square grid of 500mx500m
    longitude_variation = 0.006358  # East variation of 500 m long
    latitude_variation = 0.004495  # North variation of 500 m lat

    min_longitude = min(df['longitude'])  # weastern longitude e
    max_longitude = max(df['longitude'])  # eastern longitude e
    min_latitude = min(df['latitude'])  # southern latitude n
    max_latitude = max(df['latitude'])  # northern latitude n

    NS_side = math.ceil((max_latitude - min_latitude) / latitude_variation)  # vertical side of the minimum square cell
    WE_side = math.ceil(
        (max_longitude - min_longitude) / longitude_variation)  # horizontal side of the minimum square cell

    zone_coordinates = []
    density = np.zeros((NS_side, WE_side), dtype=float)

    with open(f"{parking}_heatmap.csv", 'w') as fp:
        grid_writer = csv.writer(fp, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        grid_writer.writerow(['latitude', 'longitude', 'count'])

        for i in range(NS_side):
            north_min = (i * latitude_variation) + min_latitude
            north_max = (i * latitude_variation) + latitude_variation + min_latitude

            for j in range(WE_side):
                east_min = (j * longitude_variation) + min_longitude
                east_max = (j * longitude_variation) + longitude_variation + min_longitude
                count = 0  # count - > number of parkings

                for k in range(1, len(df)):
                    if (east_min < float(df['longitude'][k]) < east_max) & (
                            north_min < float(df['latitude'][k]) < north_max):
                        count += 1

                density[i][j] = count
                grid_writer.writerow([north_min, east_min, count])

                coord = [(north_min, east_min), (north_min, east_max), (north_max, east_max), (north_max, east_min)]
                zone_coordinates.append(coord)

    return zone_coordinates, density
def pipline_booking_od(db, collection, city, start_date, end_date):
    return list(db.get_collection(collection).aggregate(
        [
            {  # filter data for a given city in a given period
```

```
            '$match': {
                '$and': [
                    {'city': city},
                    {'init_date': {'$gte': start_date}},
                    {'final_date': {'$lte': end_date}}
                ]}
        },
        {
            "$project": {
                "origin": {
                    "$arrayElemAt": ["$origin_destination.coordinates", 0]
                },
                "destination": {
                    "$arrayElemAt": ["$origin_destination.coordinates", 1]
                },
                "moved": {
                    "$ne": [
                        {"$arrayElemAt": ["$origin_destination.coordinates", 0]},
                        {"$arrayElemAt": ["$origin_destination.coordinates", 1]}
                    ]
                },
                "duration": {
                    "$divide": [
                        {
                            "$subtract": ["$final_time", "$init_time"]
                        },
                        60
                    ]
                }
            }
        },
        {  # filter the data
            '$match': {
                '$and': [
                    {'duration': {'$gte': 3}},
                    {'duration': {'$lte': 180}},
                    {'moved': True}
                ]
            }
        },
        {
            "$project": {
                "origin": 1,
                "destination": 1
            }
        }
    ]
))
def od_matrix(db):
    s_date = datetime(2017, 1, 10, 0, 0, 0)
    e_date = datetime(2017, 1, 10, 23, 59, 59)
```

```
    city = 'Milano'

    zone_coordinates, density = calculate_zones('weekday_morning_parking')

    trips = pipline_booking_od(db, collection_car2go[0], city, s_date, e_date)

    print('Calculating the travels list through zones...')
    travels = []
    trips_df = pd.DataFrame(trips)

    for trip in range(trips_df.shape[0]):
        for c in range(len(zone_coordinates)):
            if ((trips_df['origin'][trip][0] > zone_coordinates[c][0][1]) &
                    (trips_df['origin'][trip][0] < zone_coordinates[c][1][1]) &
                    (trips_df['origin'][trip][1] > zone_coordinates[c][0][0]) &
                    (trips_df['origin'][trip][1] < zone_coordinates[c][2][0])):
                origin = c
            if ((trips_df['destination'][trip][0] > zone_coordinates[c][0][1]) &
                    (trips_df['destination'][trip][0] < zone_coordinates[c][1][1]) &
                    (trips_df['destination'][trip][1] > zone_coordinates[c][0][0]) &
                    (trips_df['destination'][trip][1] < zone_coordinates[c][2][0])):
                destination = c
        travels.append([origin, destination])

    print('Calculating the OD matrix...')
    OD_matrix = np.zeros((len(zone_coordinates), len(zone_coordinates)), dtype=float)

    for t in travels:
        for i in range(len(zone_coordinates)):
            for j in range(len(zone_coordinates)):
                if (t[0] == i) & (t[1] == j):
                    OD_matrix[i][j] += 1
    print('Saving the OD matrix...')

    # Save the OD_matrix in a .csv file
    np.savetxt('OD_matrix.csv', OD_matrix, delimiter=',')
    print('Process done')
def vis_od_matrix():
    # Processing the OD matrix
    data = np.loadtxt('OD_matrix.csv', delimiter=',')
    data = np.delete(data, np.where(~data.any(axis=0))[0], axis=1)
    data = np.delete(data, np.where(~data.any(axis=1))[0], axis=0)

    # Check if the new matrix is square
    a = data.shape[0] # rows
    b = data.shape[1] # columns

    if a != b:
        if a > b:
            l = (a - b)
            col = np.zeros((a, l), dtype=float)
            data = np.append(data, col, axis=1)
```

```
        else:
            l = (b - a)
            row = np.zeros((l, b), dtype=float)
            data = np.concatenate([data, row], axis=0)

    # Plot the OD matrix
    plt.imshow(data)
    plt.ylabel('Origin')
    plt.xlabel('Destination')
    plt.colorbar()
    plt.show()
```

C.1 Other functions and main function to run the program

```
def pre_data_analysis(db):
    count_collections(db)
    print_cities(db)
    duration_of_data(db)
    amount_of_cars(db)
    bookings_on_november(db)
    alternatives(db)

if __name__ == '__main__':
    db = setup_connection()
    pre_data_analysis(db)
    cdf(db)
    system_utilization(db)
    rental_statistics(db)
    rental_location(db)
    heatmap(db)
    od_matrix(db)
    vis_od_matrix()
```