ALMA MATER STUDIORUM – UNIVERSITY OF BOLOGNA
CAMPUS OF CESENA

Engineering and Architecture Faculty
Master's Degree in Computer Science and Engineering

ANALYSIS, DESIGN AND IMPLEMENTATION OF A PARKING RECOMMENDATION SYSTEM APPLYING MACHINE LEARNING TECHNIQUES

Subject
MACHINE LEARNING AND DATA SCIENCE

Thesis supervisor
Prof. MARCO ANTONIO BOSCHETTI

Thesis co-supervisor
Eng. DAVIDE ALESSANDRINI

Presented by
GABRIELE GUERRINI

Single graduation session
Accademic Year 2019 – 2020

KEYWORDS

Urban mobility

Parking recommendation system

Machine learning

Data mining

myCicero

Table of Contents

Introduction

Nowadays, smart city and urban mobility topics are focusing growing interest for business and R&D purposes both, and one of its trendy application regards the study and the realization of parking recommendation systems.
As a matter of fact, understanding the parking behavior and making effective predictions play a key role in multiple optimization aspects. Besides the facilitation of the parking operation itself when looking for a free parking lot, the understanding of the existing dynamics also enables the possibility to better frame the traffic conditions and, consequently, to dislocate resources such as the traffic police. Furthermore, vehicles looking for free parking spaces negatively impact the traffic speed, and long queue can often be observed during peak hours. Accurate feedbacks to users about the current availability of parking lots can reduce the traffic congestion by limiting the "cruising for parking" phenomenon that is affecting the city centers increasingly.

In our case, a parking recommendation system is going to be developed for the city of Bologna. After a brief revision of similar proposed solutions and their approaches and findings, a first data analysis phase is going to be executed to explore the data and to mine any useful information about the parking behavior. Then, once the adopted forecasting model has been exposed, the design and the development of the system are going to be described. In the end, the performances of the system are going to be evaluated under multiple points of view, considering some project-specific constraints too.

Chapter 1

State of the art

Firstly, it follows a revision of the related works about parking forecasting topic in Section 1.1. Then, it is executed a deepening about the application where the parking recommendation system is going to be integrated (Section 1.2) and the competitors in the Italian market that already introduced a similar functionality (Section 1.3).

1.1 Related works revision

Many different projects have already been developed to model and predict the occupation of parking lots, and very different ideas and approaches have been explored. Such a great variance is mainly due to the fact that the domain is very complex, and so each singular project deals with different scenarios depending on multiple factors, among which the features of the available data.

Available data

Besides the occupation of parking lots, information about exogenous factors are often exploited too, including weather data, traffic speed and outlier temporary conditions. This latter aspect is comprehensive of car crashes, city market day, closed roads, holidays.
Also, we can identify further aspects that make it possible to provide different capabilities to the solution, namely:

- Usage of real-time data against usage of the historical data solely: the usage of real-time data permits to have accurate information that otherwise must be approximated (e.g., exact arrival and/or departure time).
- Historical depth of the datasets: the period covered by the data influences the type of seasonalities that can be found.

- Sampling period of the data: it influences the minimum period a new prediction can be generated (every 5 minutes, 15 minutes, etc.).

Forecasting model

As well as the commonplace approach based on neural networks, statistical predictors (e.g., ARIMA, [2]) and other well-known machine learning techniques (e.g., random forest, [3]) have been applied too. However, an acknowledged best performing standard is still not present among the different forecasting models, and there is not a compliance in the type of problem to be solved neither (regression against classification).
Also, further models have been enriched using the fuzzy logic principles to estimate the uncertainty of parking availability during the peak parking demand period [6].

When a regression problem is faced, the time series concept has been used to model the stops trend in most of the cases. Though, the way the time series are built is not always the same and depends upon the forecasting goal of each singular project (find out an occupation percentage, find out a delta in number of arrivals and departures, find out the time a parking lot is going to remain occupied/free, etc.).

Concerning the neural network domain, we have different adopted methodologies straddling between more architectures. Indeed, besides classical MLP networks that are still used, Recurrent Neural Networks (RNN) are widely employed to catch time dependences and, in some cases, Graph Convolutional Neural Networks (CGNN) are employed to catch spatial dependences too [4]. Farther, more complex techniques have also been tried by:

- Ad-hoc redesigning the architecture of the LSTM unit [5]: the input gate is modified to merge the regular input with information about weather conditions and recurrent patterns at three different granularity level (day, week, month).

- Training the neural network using genetic algorithms [1].

- Combining more neural networks and merging their partial results [4] (similarly to bagging techniques principle).

Geometric model

When dealing with the geometric aspect, the common solution is to split the territory up in different regions so that situated forecasts can be executed. The number of regions a city can be divided in highly varies case by case,

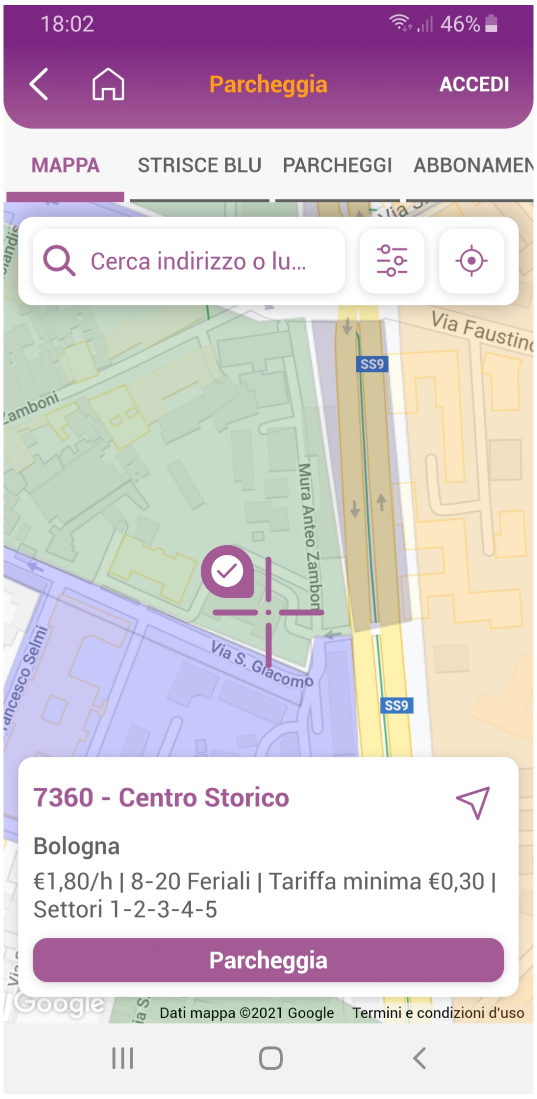
and the methodology adopted to split the territory is generally handmade by considering the districts on the map or the road network.
The correct detection of regions is a crucial aspect since it has been found that different areas may assume completely different behaviors depending on the time of the day, the day of the week and weather conditions. This latter factor significantly influences performances in general, and it has been discovered that different areas suffer occupation changes on different type of weather. Indeed, business areas expose higher values on adverse weather (storm, snow, etc.), while recreational areas follow an opposite behaviour.
Moreover, areas containing the more parking lots seem to perform better because the occupation rate variance is lowered, and so more stable and accurate forecasts can be made. On the contrary, areas containing just a bunch of parking lots are susceptible to fluctuations.
Furthermore, a time correlation between adjacent areas has been found in some cases [4]. Such areas reach a high occupation in similar times during the day, except for a little shift. This is probably due to the filling of a certain block where it is contained a major place of interest firstly. Then, once such a block is full, people begin to look for more distant parking lots, and so near blocks start to grow their occupation too.

In the end, notice that few projects worked on a simplified scenarios where the incoming data covers just one or more restricted areas. Hence, the domain had been considered as atomic, and no particular geometric model had been adopted.

1.2 myCicero

"myCicero" [7] is a digital platform developed by the "Pluservice" [8] company and whose goal is the offering of different services, including info mobility, mobile ticketing, shopping promos, points of interest (POI).
The application is based on a MaaS architecture (Mobility-as-a-Service), and already integrates more than 200 mobility operators.

When talking about the ticketing area, the application handles different typologies of transport, such as railways, taxis, bus routes, car pooling.
The "on street parking" functionality handles the paid parking lots management, and Figure 1.1 reports an example of its usage.



7360 - Centro Storico

Bologna

€1,80/h | 8-20 Feriali | Tariffa minima €0,30 | Settori 1-2-3-4-5

Parcheggia

Figure 1.1: *The "on street parking" functionality of "myCicero" application. Different colors identify different hourly rates.*

This latter functionality enables the payment of the parking through the mobile application so that users can get rid of parking meters just by using their mobile phone. The main advantages are:

- **Faster parking:** the user does not need to reach a parking meter by foot anymore.
- **Easier payment:** the is no need for the user to use coins to pay.
- **Increased flexibility:** the application is going to charge only the effective minutes of the stop. Moreover, the user can extend the stop remotely

without coming back to the car.

The parking recommendation system to be developed will be integrated in this parking functionality, and there is no previous version of it. The goal is to provide a visual feedback for each street on the map ideally.

1.3 Competitors

1.3.1 EasyPark

"EasyPark" [9] is the first competitor that introduced a parking recommendation system in the Italian market, and the company already launched the functionality in several European cities. In Italy, Verona had been the first one in 2018, but then other cities were included too (such as Milan and Rome).
The "Find & Park" functionality allows the users to check the probability to find parking for each street, and such probability is expressed using different colors on the map (Figure 1.2). The goal is to save time when looking for a parking lot, and it has been confirmed that the time spent by the users is cut down from a couple of minutes to the half depending on the situation.
It has been declared that the recommendation system considers the following factors:

- "EasyPark" data: stops executed by the users that pay through the application.
- Data provided by the local mobility operator, also including exogenous factors such as temporary closed streets and city market day.

In the end, as well as displaying the map containing the status of each street, users can also activate a navigator whose calculated journeys are optimized upon the availability of the parking lots instead of the travel time.

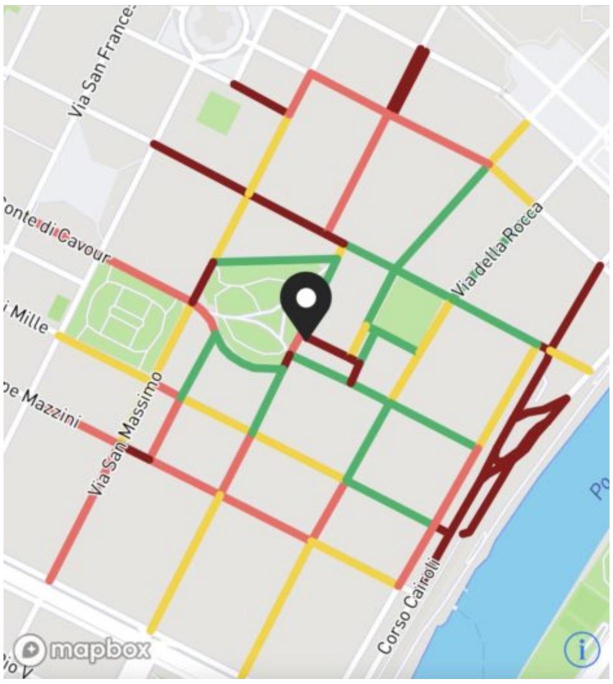


Figure 1.2: *Example of the "Find & Park" functionality.*

1.3.2 APParked

"APParked" [10] is another application providing a similar functionality. However, even though the goal is way similar, the application uses a completely different approach. Indeed, it is based upon the social idea and the collaboration of the users so that free parking lots are reported.
A user can ask for a free parking lot, and the platform will return all the nearby availabilities. Then, the user notifies the platform when the parking lot is effectively occupied (*park-in*), and a suggested duration of the stop can be provided so that it is known an expected time for the parking lot to be released. Obviously, the user can also notify to the platform when he is leaving (*park-out*).
Besides the common benefits, users are enticed to use the application through various rewards. The more information they provide, the more reward they can obtain.

Chapter 2

Data analysis

Firstly, it follows a description of the available data in Section 2.1. Then, once the model adopted to shape the stops has been explained (Section 2.2) and the datasets have been preprocessed (Section 2.3), a data analysis phase is executed to mine any useful information that may be considered during the development of the system itself[1].

Premise: Notice that there is no info about night hours since the parking is free during such period and, consequently, there is no check made by police officers too. Hence, no analysis can be executed nor any feedback to users can be given using the provided data for the time being.
Moreover, a similar reasoning can be made for holidays where the parking is free too.

2.1 Data exploration

There are three admissible ways to occupy a paid parking lot, that is:

- Use a near parking meter to pay the required amount.
- Use an application that allows the payment using a digital wallet (or any similar feature). We focus on information coming from "myCicero" application since data of competitors is obviously unknown.
- Use a seasonal ticket (or any similar special convention).

A dataset for each payment method is available grouping transaction that happened in the period 01-01-2019/31-12-2019.

[1]Analysis that did not lead to any interesting result are not reported for conciseness.

2.1.1 myCicero

The dataset contains an overall of 310k records with a size of 20MB. Each record represents a payment made in a certain moment and in a certain point of the space. Hence, each entry contains few useful information such as:

- Longitude and latitude coordinates
- Transaction timestamp representing the moment the stop begun
- Stop length (minutes)

Records are provided in the format:

(longitude: float, latitude: float, stop length: int, timestamp: string)

Timestamps, already rounded to quarter of hours, are provided in the following format:

Y-M-D h:m:s

Table 2.1 reports few example records.

Longitude	Latitude	Stop length	Timestamp
11.347791	44.508027	27	2019-01-01 11:15:00
11.311599	44.491037	60	2019-01-01 11:15:00
11.354347	44.487044	197	2019-01-01 13:00:00

Table 2.1: *Sample of the "myCicero" transactions dataset.*

2.1.2 Parking meters

We need information about location, time and length for each stop but, unfortunately, such information is split up among multiple datasets. Indeed, besides the transactions dataset storing the arrival time, we must use two more datasets to calculate stop location and length, namely the parking meters' registry (storing location) and the hourly rates dataset (storing the required information to correctly calculate stop length from the paid amount).

Transactions dataset

The dataset contains an overall of 3800k records with a size of 380MB. Each record represents a payment made in a certain moment to a certain parking meter. Hence, each entry contains few useful information such as:

- Parking meter's id
- Transaction timestamp
- Paid amount

A first preprocessing operation[2] is required to correctly shape the data since column separator and amount values use the comma character both. Therefore, amounts containing decimal digits are split upon two columns, and records do not have all the same shape.
After such operation, the records are provided in the format:

(parking meter id: int, paid amount: float, timestamp: string)

Timestamps are provided in the following format:

Y-M-D h:m:s.ms

Table 2.2 reports few example records.

Parking meter id	Amount (€)	Timestamp
10001	2.0	2019-01-01 23:04:00.000
10001	1.0	2019-01-02 08:23:00.000
5516	0.9	2019-01-26 18:31:00.000

Table 2.2: *Sample of the parking meters' transactions dataset.*

Hourly rates dataset

A parking area is a region of the city where a same hourly rate is applied to all parking lots within it.
Each record of the dataset includes the name of an area and the information associated to such area. The rate values hold just for regular days since free parking is applied during holidays (the rate is "0").
Table 2.3 reports the complete hourly rate dataset (just 4 areas).

[2]Such operation is not exposed in next preprocessing section since it is required to actually have a dataset that, otherwise, is not usable.

Area	Rate (€)	Validity
Cerchia del Mille	2.4	8:00 20:00
Centro Storico	1.8	8:00 20:00
Corona Semicentrale	1.5	8:00 18:00
Zona Bolognina-Arcoveggio	1.2	8:00 18:00

Table 2.3: *Hourly rate of each area.*

Registry dataset

The dataset stores the registry of parking meters (819 records in total) with information about:

- Parking meter id
- Longitude and latitude coordinates
- Parking area where a parking meter is located
- Other info of no interest (e.g., human readable location where the parking meter is placed in term of street name and house number)

Table 2.4 reports few example records.

Id	Longitude	Latitude	Area
10001	11.348345	44.508008	Corona Semicentrale
3032	11.348821	44.485008	Cerchia del Mille
3023	11.346152	44.8913	Centro Storico

Table 2.4: *Sample of the parking meters' registry. Only columns of interest are shown.*

2.1.3 Verifiers dataset and seasonal tickets

The verifiers dataset includes all the checks made by police officers during the selected period[3] and it contains an overall of 1800k records with a size greater than 500MB.
Each record represents a check made in a certain location at a certain time, and so it includes the following useful info, that is:

[3]Checks about stops in order are included too.

- Longitude and latitude coordinates representing the point of the space where the check has been registered (and so where the vehicle is parked approximatively)
- Timestamp representing the time the check has been registered
- Type of ticket (regular parking meters' ticket, seasonal tickets, etc.)
- Id of the street (useful to execute aggregated analysis)

The remaining columns report information of no interest and so they are omitted.

Few ETL transformations are required to generate the dataset from the input raw files. Indeed, even though the data is already organized in columns and separated by commas, it is split among 12 text files (one for each month) that need to be converted to CSV format and then merged. When doing the above operations, a preliminary projection is executed contextually in order to drop useless columns.

Besides the complete verifiers dataset that may be useful during the next performance evaluation stages, the target is also to create a dataset representing the slice of occupied parking lots that can not be framed using the two aforementioned datasets, that is stops that did not require any payment thanks to seasonal tickets, hotel conventions, school conventions, etc. Hence, the verifiers dataset is filtered to select only the records of interest, and the resulting dataset's size is about 100MB, reduced by 80% from the starting one.

When talking about the seasonal tickets data, records are provided in the format:

(longitude: float, latitude: float, timestamp: string, street id: string, ticket type: string)

Timestamps are provided in the following format:

D-M-Y 00:00:00 h:m

Table 2.5 reports few example records.

Longitude	Latitude	Timestamp	Ticket type	Street id
11.3412	44.5153	01/10/2019 00:00:00 08:08	ABBONAMENTO	32760
11.316	44.4932	26/11/2019 00:00:00 09:21	ABBONAMENTO	10750
11.316	44.4932	26/11/2019 00:00:00 09:21	INVALIDI	10750

Table 2.5: *Sample of the verifiers' dataset after the ETL operations.*

2.2 Time series model

Times Series (TS) with different time intervals and sampling periods are employed to describe the stop trend at different granularity levels.
Table 2.6 reports the adopted taxonomy for different types of TS.
Note: TS spread over multiple years may be used to find high-level behaviors that live aside the single year (such as an expected lower occupation during summer period), but this would require a dataset with a historical depth of few years.

Type	Time interval	Sampling period	Number of values on time axis
Day by quarters	1 day	quarter (of an hour)	96 [1]
Year by quarters	1 year	quarter	35040-35136 [2]
Month by days	1 month	day	31
Year by days	1 year	day	365-366
Year by months	1 year	month	12

Table 2.6: *All types of time series along with respective defining info.*

Any value of a given TS represents the number of occupied parking lots in such time. Let us consider, for example, the following TS:

$$[1, 5, 4, 3, 2, 10, 7, 8, 3, 2]$$

We have 1 occupied parking lot at time 0, 5 at time 1, 4 at time 2 and so on.

It follows a detailed explanation of the *day by quarters* TS. A similar reasoning, omitted for conciseness, can be made for other types of TS too.

Day by quarters TS

Quarters are indexed in range $[0; 95]$ using the following criteria:

00:00 -> 0, 00:15 -> 1, 00:30 -> 2, . . . , 23:45 -> 95

Each record of the dataset contributes as “1” for any quarter the stop covers, and the number of quarters it contributes to is given by the stop length using the formula:

$$n\_quarters = round(stop\_length/15) + 1$$

[1]24 hours per day * 4 quarters per hour
[2]365/366 day per year depending on leap years * 96 quarters per day

Once covered quarters for each transaction is calculated, we generate the TS just by counting how many times each quarter appears.

It follows a clarifier example. Table 2.7 reports few records and the respective information required to generate the TS.

Input info		Calculated values		Output
Arrival time	Stop length	Arrival quarter index	# quarters	Covered quarters
10:00	48	32	4	[32, 33, 34, 35]
10:15	63	33	5	[33, 34, 35, 36, 37]
23:30	68	94	5	[94, 95, 0, 1, 2]

Table 2.7: *Examples of covered quarters calculation.*

Once we have stop lengths in term of covered quarters, we can merge them all and generate the following TS by counting the number of occurrences of each quarter:

[
$1, 1, 1, 0, 0, 0, 0, 0, 0, 0,\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 1, 2, 2, 2, 2, 1, 1, 0,\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 1, 1$
]

The first element of the TS (i.e., time 00:00 having index "0") has value "1" since we have only one "0" in the covered quarters array.
The second element of the TS (i.e., time 00:15 having index "1") has value "1" since we have only one "1" in the covered quarters array.
The 34th element has value "2" since we have two values "34" in the covered quarters array.
Similarly for all other items of the TS.

2.3 Preprocessing

2.3.1 Timestamp approximation

It is required an approximation of timestamps to execute further analysis because of two main reasons, that is:

- All datasets should use the same time detail on timestamp to correctly merge/compare them. However, "myCicero" timestamps are already rounded to quarters, and so such a round must be executed on other datasets too.

- Even if we had minutes detail for all datasets, such fine-grained information would be an overkill, and it should be reduced since there is no point on dividing between transaction occurred in one-minute distance. On the other hand, just dropping minutes information may be excessive since we may lose dynamics happening within hours. Thus, the quarter approximation may be a reasonable tradeoff.

Therefore, we approximate timestamps using quarter of an hour. There are several possibilities to do such an approximation (to the nearest quarter, to the previous one, etc.). Notice that there is no wrong option, and each one would lead to a different modeling of the data.

The approximation to the nearest quarter has been chosen[4] (Figure 2.1 reports a graphical explanation).

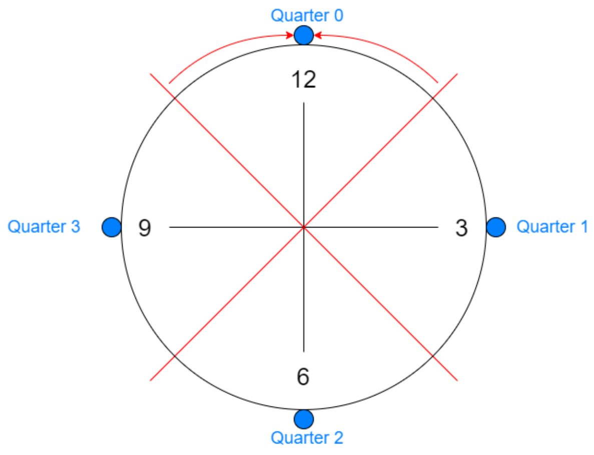


Figure 2.1: *Approximation of each minute to the nearest quarter.*

2.3.2 Aggregation of near parking meters

It is required an aggregation of parking meters for data mining and solution development both. The problem arises from the distribution of parking meters over the territory, and it is addressed to the excessive proximity of few couples of them. This problematic situation occurs in two main casuistries, that is:

- Two parking meters located at the two sides of a street.

[4] Actually, since "myCicero" data is already rounded to quarters, the same approximation should be used to keep consistent data. Unfortunately, the type of that approximation is not known for the time being.

- Two parking meters located at the opposite corners of a crossroad.

Indeed, parking meters of given couple likely share a similar behaviour (intended as the trend of the stops along daylight hours), and so they can be aggregated and reduced to one. Figure 2.2 reports a clarifier example.

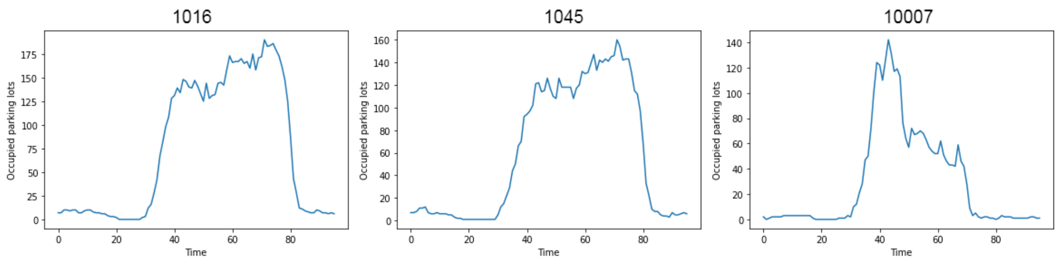


Figure 2.2: *It is reported the trend of stops along the day for three different parking meters (1016, 1045, 10007). 1016 and 1045 are located at the two sides of the same street ("Via Irnerio") and expose an identical trend. Meanwhile, 10007, exposing a totally different behaviour, is a randomly chosen parking meter far from to above ones.*

By aggregating close parking meters, we obtain the following advantages:

- Reduction of the amount of computation to be done.
- Collapse of different items that actually represent the same underlying concept.

This operation is executed iteratively by aggregating couples one by one. When a couple is aggregated, the new parking meter will be located in the middle of the segment that connects the two starting points (Figure 2.3), and the resulting parking meter will own one of the two starting id (it does not matter which one).
The metric used when calculating the distance between any couple of parking meters is the *Haversine distance.*

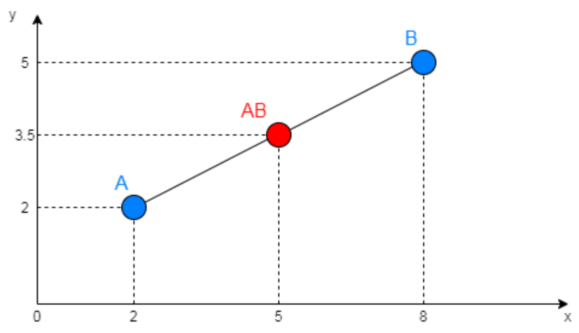


Figure 2.3: *Two points (blue) and their middle point (red).* $(x_{AB},\ y_{AB}) = ((x_A + x_B)/2,\ (y_A + y_B)/2) = ((2 + 8)/2,\ (2 + 5)/2)$.

The starting cardinality of parking meters is 816 (3 already dropped from starting 819 due to missing values). The minimum allowed distance for two parking meters that avoid them from being aggregated is a parameter. Table 2.8 reports few attempts when tuning such a parameter so that only parking meters with same behaviour are collapsed.

Minimum allowed distance (m)	Number of dropped parking meters
50	122
20	38
10	23

Table 2.8: *Number of aggregated parking meters when varying the minimum allowed distance.*

By analyzing the behaviour of a sample of the aggregated parking meters, the usage of 20m distance seems to be a good trade-off. The aforementioned assumption (i.e., merge only parking meter with similar behaviors) cannot be considered true any more when using higher distances (such as 50m), while few couples may be lost and not aggregated when going under 20m.
Thus, the number of aggregated couples is 38, resulting in 778 final parking meters.

2.3.3 Drop of outliers

The considered territory is included in range [44.47; 44.55] and [11.30; 11.38] for latitude and longitude respectively, and records detected to be outside the domain are dropped.

By analyzing “myCicero” dataset, 37 records are detected to be located in different Italian cities, namely:

- Riva del Garda (TN): area 10.83/45.88, 22 records
- Rimini (RN): area 11.71/44.36, 10 records
- Imola (BO): area 12.56/44.06, 5 records

Such outliers are dropped even though it is not clear why they are present in the datasets.

By analyzing parking meters registry, the ones with id 3017, 14109, 15025 are dropped. Such outliers clearly are typing errors (wrong values, missing digits, etc.) but the involved parking meters result on being outside of the domain, and so they are dropped.

In the end, seasonal tickets dataset is cleaned too. In this case, a bunch of records are dropped since they have latitude or longitude set to “0”. Actually, such records are dropped now since they are located outside the domain, but it is obvious how they are not real outliers (probably, “0” value has been put as placeholder to fill missing information).

2.3.4 Filling of missing values

There are four fields to be filled or calculated, namely:

- Stop length for parking meters transactions
- Stop location for parking meters transactions
- Stop length for seasonal tickets
- Stop location for few seasonal tickets records

Parking meters: stop length and location

The location in term of longitude and latitude coordinates is not carried with each transaction, but it can be easily retrieved from the registry dataset. In reverse, there is no explicit stop length for parking meters transactions, but it can indirectly be calculated from the paid amount and the hourly rate using the following formula:

$$length = (amount/hourly\_rate) * 60 \tag{2.1}$$

The stop length is expressed in minutes. The hourly rate varies on the parking area where the parking meter is located in.

In order to retrieve the missing values, parking meters registry is firstly joined with the hourly rates dataset on the *“area”* column to retrieve the

hourly rate associated to each parking meter. Next, the transactions dataset is joined with the registry on the *"id"* column in order to retrieve location and hourly rate both for each transaction, and then the Formula (2.1) is applied to find the stop length out.
In the end, the transaction dataset may contain few missing values if any parking meter's id in the transaction dataset is not also present in the registry (failed join operation). Such records are simply dropped.

Seasonal tickets: stop length and location

The main issue is that there is no information about the stop length for seasonal tickets data. However, this latter value is mandatory to create a TS, and such info is set up to 1 minute. Notice that, since a same value is going to be used for all records, the chosen length is not a knot or an error because any other greater value would just modify the TS by widening the "bells" created by the peaks, and so the overall trend would just be blurred while keeping the same overall shape of the time series (Figure 2.4).

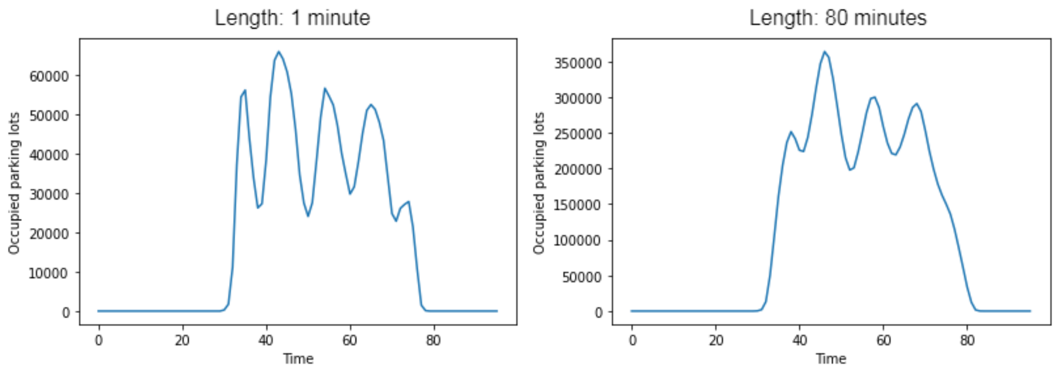


Figure 2.4: *Trend of stops along the day with stop length set to 1 (left) and 80 (right) minutes, respectively.*

All the records contained in the period April 7th - June 11th have missing information about location. This is not a problem when talking about aggregated analysis since they can be dropped, and the remaining dataset is still significant for most of the techniques.
However, it may become an issue of primary importance when executing situated analysis or forecasting tasks. Hence, such an aspect must be kept in mind and fixed if required (e.g., by interpolating missing values of the TS).

2.4 Time analysis

2.4.1 myCicero

Figure 2.5 reports three different time series along with respective Pearson autocorrelation values and normality tests (using Q-Q plots).

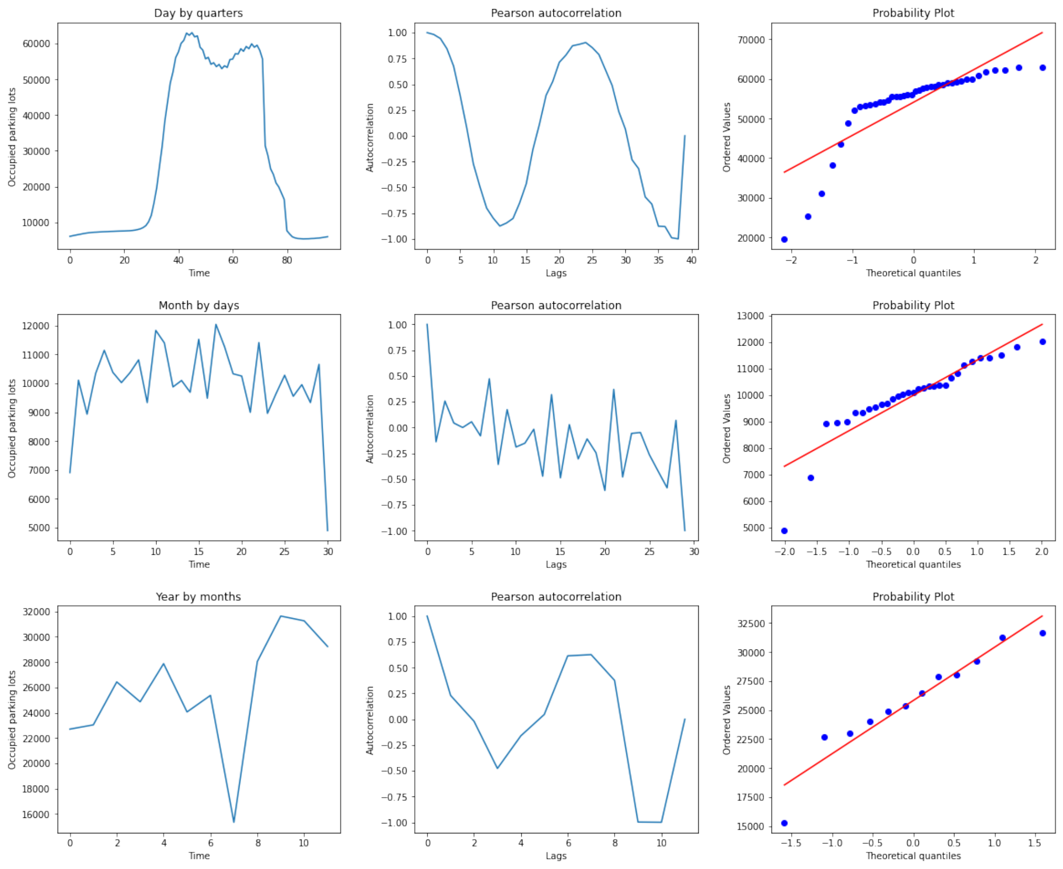


Figure 2.5: *Three different TS along with information about Pearson autocorrelation and normality test using Q-Q plot for daylight hours.*

Day by quarters

It is quite easy to identify a neat division between day and night hours with very low values on the latter ones. From now on, daylight hours are estimated to be the ones in the interval 8 a.m. – 6 p.m., and most of the analysis will be executed on such period since it is the one of interest.

During daylight hours, the time series is shaped as a bimodal one since the almost constant trend occupation is interrupted in the middle. This is

probably due to the launch break, and so it is not a novel fact of particular interest.
In the end, the distribution of data can be assumed as normal if not considering few outliers.

Month by days

The times series does not show any major regularity among different periods of the month, and this is formally confirmed by the Pearson autocorrelation.

Anyhow, we can detect a slightly 7-lags autocorrelation. This may point a similar trend out for the days of the week independently from the month. The issue that may mask such a correlation can be addressed to the shift of the days as months proceed (that is, the 1st of a month is not always Sunday or Monday, etc.).
Hence, we use a different time series modelling each day of the year (*year by days*), and we look to the Pearson autocorrelation index (Figure 2.6). As we can notice, the seasonality is now clear, and the period's length is seven (one week). Moreover, we can conclude there is no sub-seasonality on midweek days.

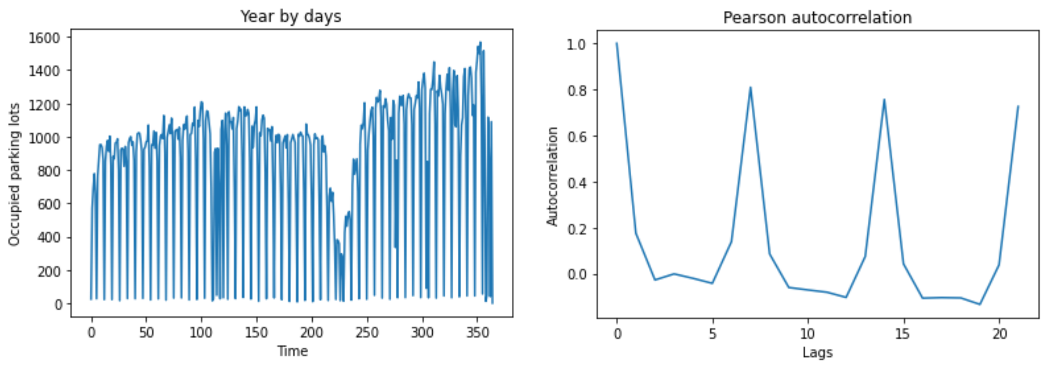


Figure 2.6: *Year by days TS along with associated Pearson autocorrelation index for "myCicero" data.*

In the end, the distribution of data can be associated to a normal one except for 2 outliers that have lower values, that is:

- 1st day of the month: a lower number of stops seems to occur on such day. This is probably due to a combination of holidays and Sundays that occurred in that year so we just have 8 values out of 12 (1st January and 1st November are holidays, 1st September and 1st December are Sundays).

- 31st day of the month: obviously, only 7 months out of 12 have such a day, and so it is normal to find a lower value. Hence, it cannot be considered a real outlier.

Year by months

The time series cannot be analyzed to find any recurring pattern since only one-year period data is available (e.g., higher values and growing trend after summer till the end of the year can be either a general trend among multiple years or a recurrent seasonality within the year). However, as we expected, we find a fall on number of stops in August.
Moreover, we do not have enough values to correctly apply the normality test (or better, to trust its results).
Thus, nothing can be said about autocorrelation and normality, but the respective plots are reported for completeness.

2.4.2 Parking meters

Figure 2.7 reports three different time series along with respective Pearson autocorrelation values and normality tests (using Q-Q plots).

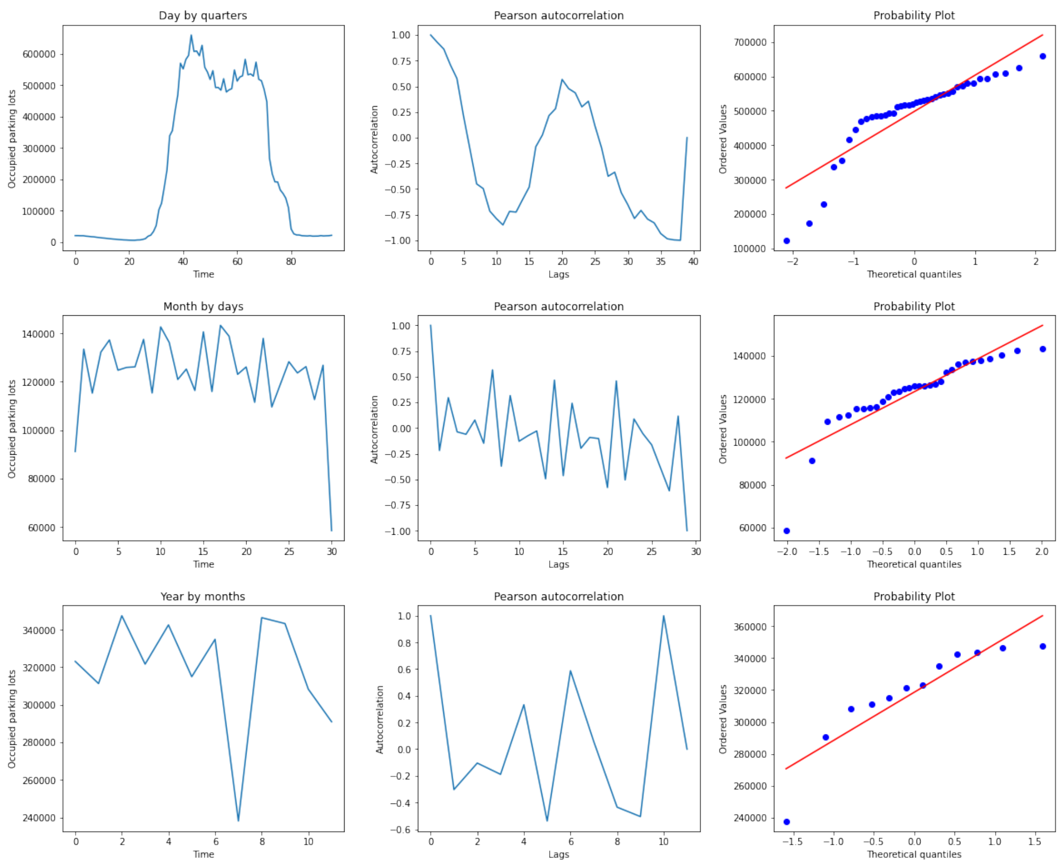


Figure 2.7: *Three different TS along with information about Pearson autocorrelation and normality test using Q-Q plot for daylight hours.*

Day by quarters

The TS trend is very similar to the one of “myCicero” data, and so similar consideration can be exposed. The only outstanding element is the presence of few periodic little spikes that break the “linear trend” of the TS (a deepening about such aspect is made in Section 2.6).

Month by days

Similar considerations to the ones stated for “myCicero” data can be executed here too since similar trends and patterns are observed. Figure 2.8 reports the *year by days* time series.

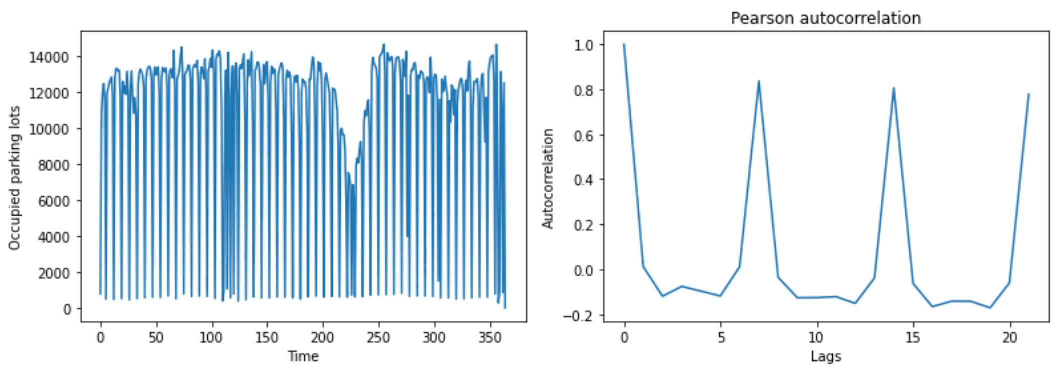


Figure 2.8: *Year by days TS along with associated Pearson autocorrelation index for parking meters.*

Year by months

Despite similar patterns to "myCicero" data such as a fall in stops on August, there is one main difference which resides in the number of stops. Indeed, now the growth of stops after the summer period does not occur any more (may be seen clearly by comparing Figure 2.6 to Figure 2.8). There are three main possibilities that may produce such a gap, namely:

- Parking meters and "myCicero" are used by different classes of people, and so it is normal that they do not share the same trend since they represent different underlying concepts, even though all the other time series create a match between parking meters and "myCicero" with very similar trends.
- The difference is given by a spread of the "myCicero" company and more people using the application, while the parking meters are of public domain and so its usage remains constant. This would imply that there is no real difference in term of habits of the customers.
- The increase is simply given by an outlier period.

None of the three hypotheses can be discarded with the provided data.

2.4.3 Seasonal tickets

Figure 2.9 reports three different time series along with respective Pearson autocorrelation values and normality tests (using Q-Q plots).

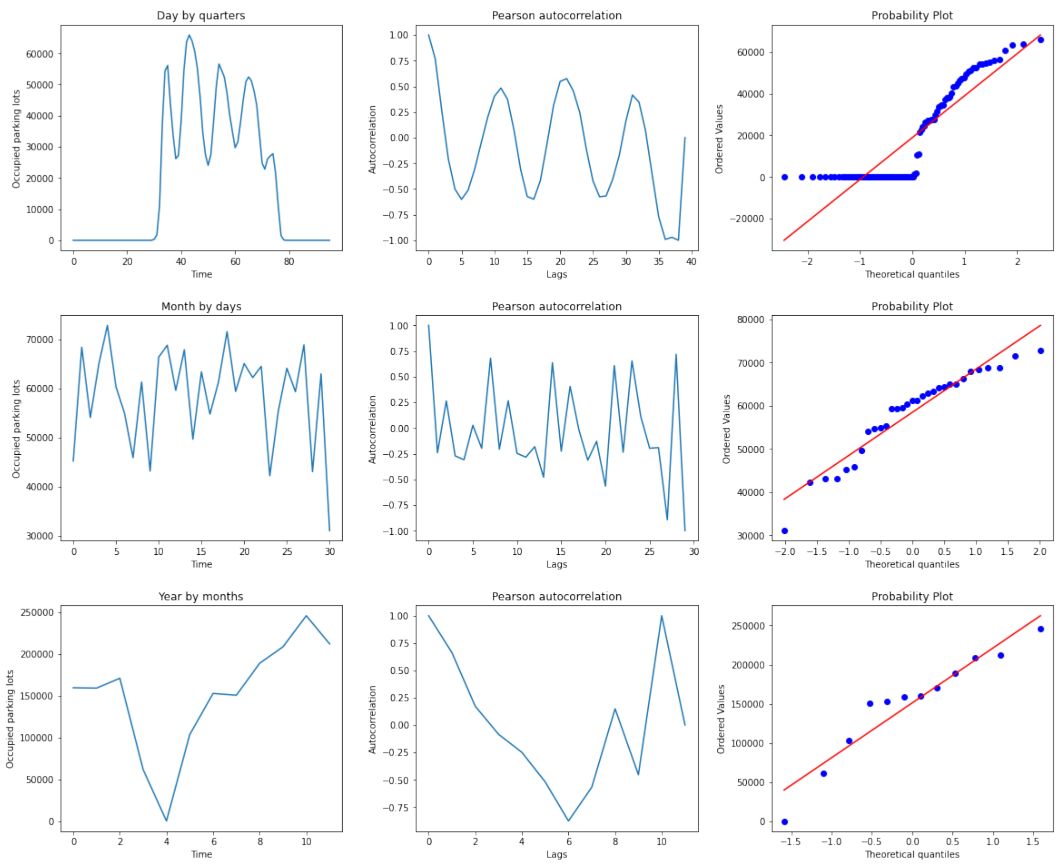


Figure 2.9: *Three different TS along with information about Pearson autocorrelation and normality test using Q-Q plot for daylight hours.*

Day by quarters

The time series shows a well-defined recurrent pattern: high and low valued are interleaved during the daylight hours. This is probably due to the checking activity following a regular frequency along the day, and so the trend of stops is not trustable to be coherent with the real one. Indeed, while for parking meters and "myCicero" we can exactly model the arrival of a customer and the associated stop by using the timestamp and the stop length, now we do not know the real arrival time nor the length of the stop (length is mocked as exposed in previous Section 2.3.4, arrival timestamp is the one when the check had been made).
The TS actually takes snapshots of the occupation of parking lots at a given time, while not considering the stop over the time passing. To better understand this issue, we may just look to night hours where the recorded number of stops is exactly always zero since checking are not executed during such pe-

riod. This is not a problem since the occupation during night hours would be low anyway, but such a concept becomes a big issue when considering daylight hours.

Hence, low peaks on daylight hours can not be assumed as true since they do not represent a real decrease in the number of occupied parking lots. As exposed Section 2.3.4, a possible solution is to create a "blurred" TS by using length set to any random value (e.g., 60 minutes) so low peaks are filled. However, this way we do not obtain any useful information anymore since the trend becomes almost constant, except for a little peak in middle morning.

Month by days

Similar considerations to the ones already stated in previous analysis can be made. *Year by days* TS is exploited to assess that the seasonality is seven. The gap in spring months is not unexpected since those records have been removed during outliers dropping stage.

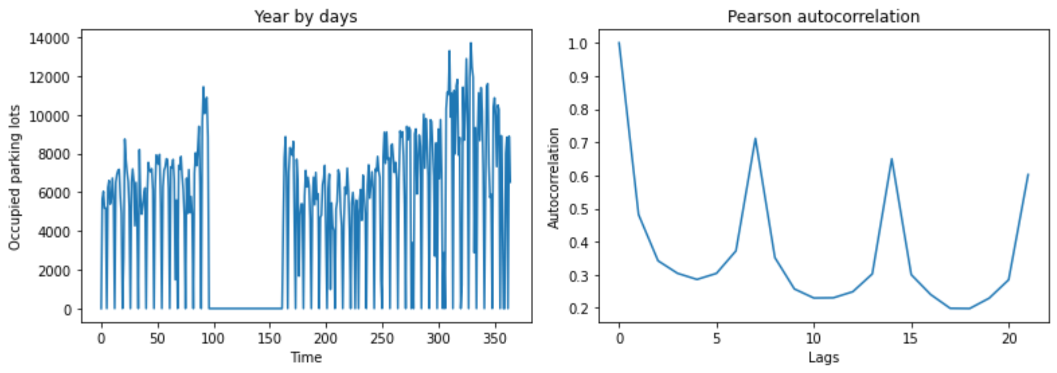


Figure 2.10: *Year by days TS along with associated Pearson autocorrelation index.*

Year by months

Similar considerations to the ones already stated in previous analysis can be made about seasonality, normality test, etc. There are only two remarkable thing that may be underlined, that is:

- As already discovered, the falling during spring period is due to the missing data in such period.
- There is not a clear fall during August month, and this implies that the slice of the population that makes use of seasonal tickets is not greatly affected by summer variations.

2.5 Spatial analysis

Parking meters and “myCicero” data are compared on different areas.
The area sampling is handmade and, in order to minimize the sample bias, different areas are chosen considering few characteristics, such as:

- Downtown areas vs suburbs areas.
- Different shapes: stretched areas, rounded areas, etc.
- Completely random areas vs areas with known potential peculiarities (e.g., university).

It follows an analysis on two of the sampled areas[5] over multiple abstraction levels (analysis on whole area and on sub-areas both). In some cases, a robustness check has been executed too.

2.5.1 University area (north wing)

The considered area, whose schema is reported in Figure 2.11, is the north wing of the university district.

[5]Just few examples are shown since the description of all the analyzed areas would be too much verbose.

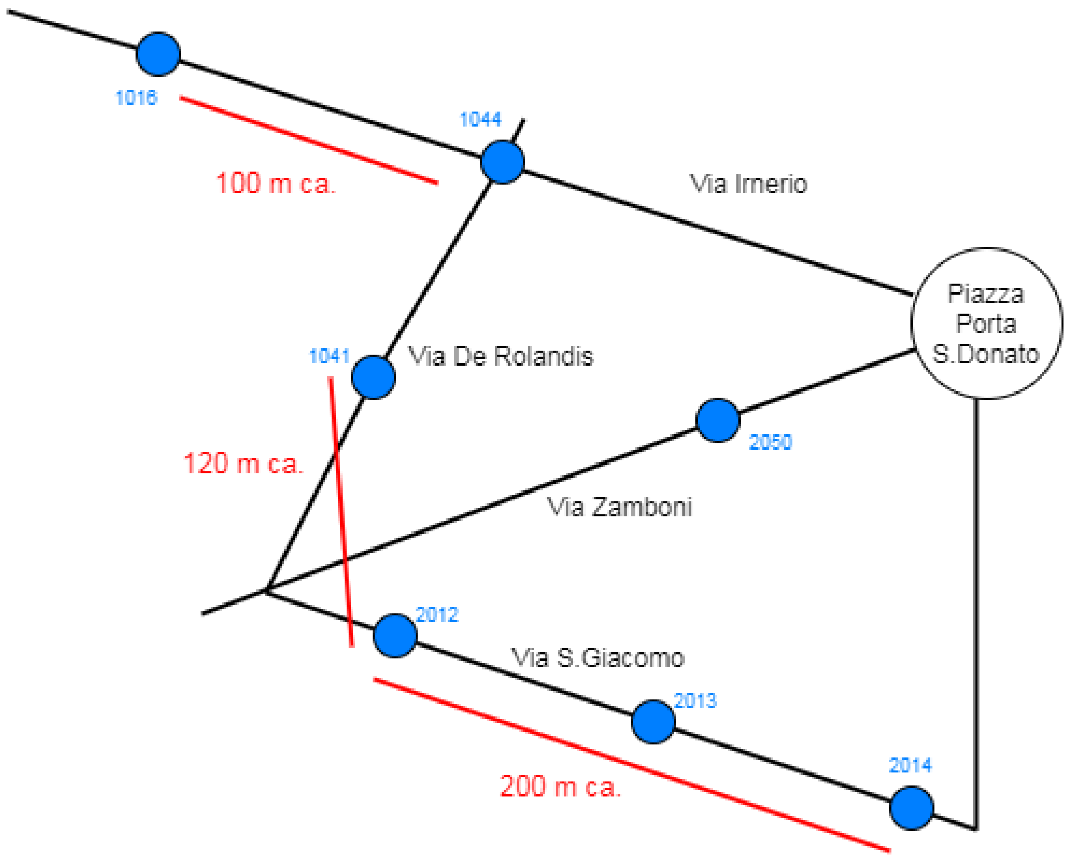


Figure 2.11: *Schema of the selected area. Blue markers identify parking meters (each one with its own id).*

We firstly consider parking meters to understand the trend of stops along the day in Figure 2.12.
We notice different behaviours among different areas. Indeed, as we can see for parking meters 2012, 2013 and 2014, the trend is strongly influenced by the territory peculiarities: they are all located on the same street (*S.Giacomo*) within the university context, and they share the same monotonic increasing trend. This is not true for other parking meters that are located nearby the university though.

Moreover, we understand that the distance is a tricky dimension. Indeed, $d(2012, 2014) > d(2012, 1041)$[6], but parking meters 2012 and 1041 do not share the same behaviour (while it is true for 2012 and 2014). Thus, we can conclude that the little the distance the similar the behaviour (et vice versa) is not a fair assumption in general.

[6]Euclidean distance.

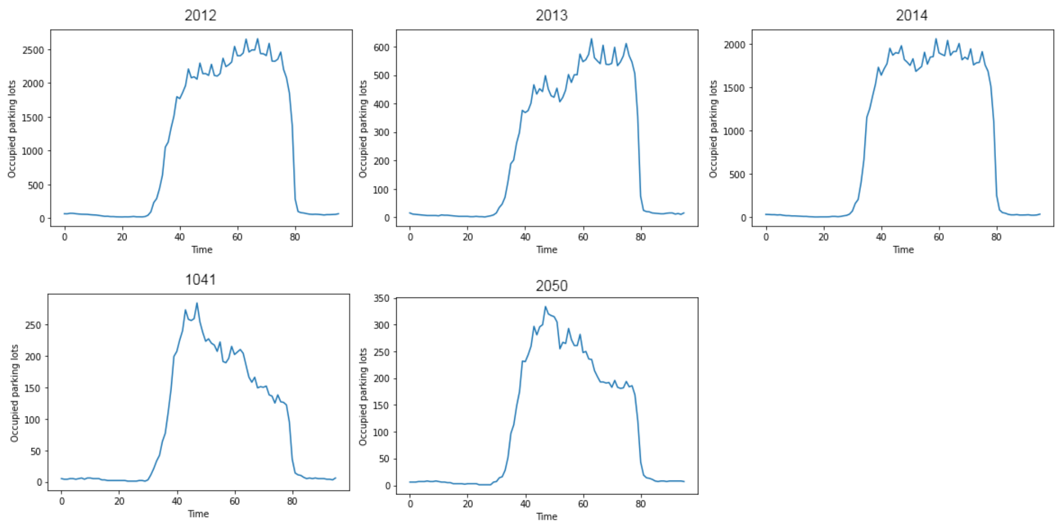


Figure 2.12: *Trend of stops (expressed using the day by quarter TS) of few parking meters within the considered area.*

Now, we look for any evidence of similar behaviors on "myCicero" data. In particular, we consider "S.Giacomo" street (Figure 2.13) and, then, we look to corresponding time series (Figure 2.14).
As we can notice, there is not any correspondence between parking meters and "myCicero" in this area with the two data source modelling completely different behaviours.

Moreover, there is no correspondence neither between the two halves of the street within the solely "myCicero" data. Indeed, by splitting the street up into two halves, the first exposes a peak in the middle of the day and low levels for other hours, while the second one has an almost constant trend.

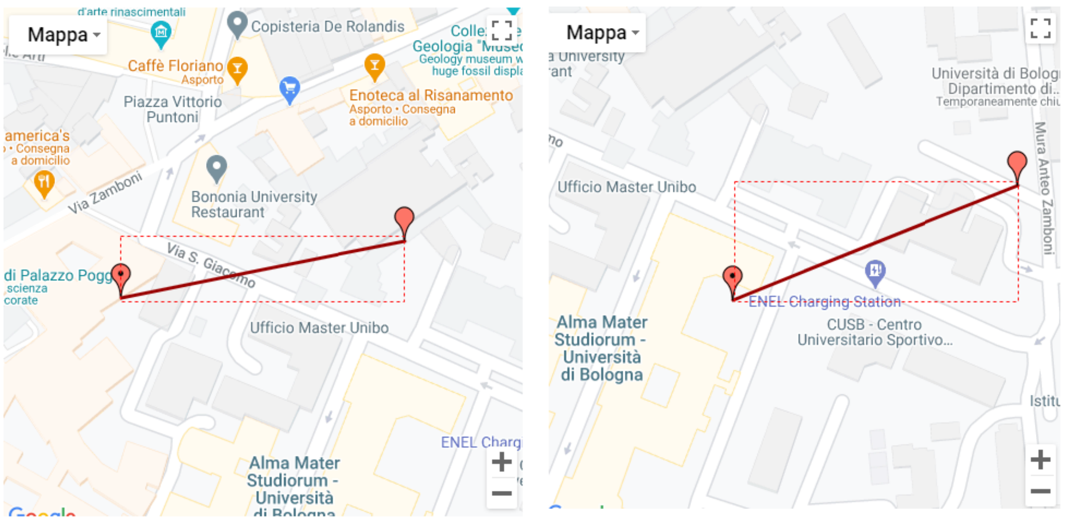


Figure 2.13: *S.Giacomo street on the map. The street is split up in two halves so it can be approximated by the two red rectangles in the best way.*

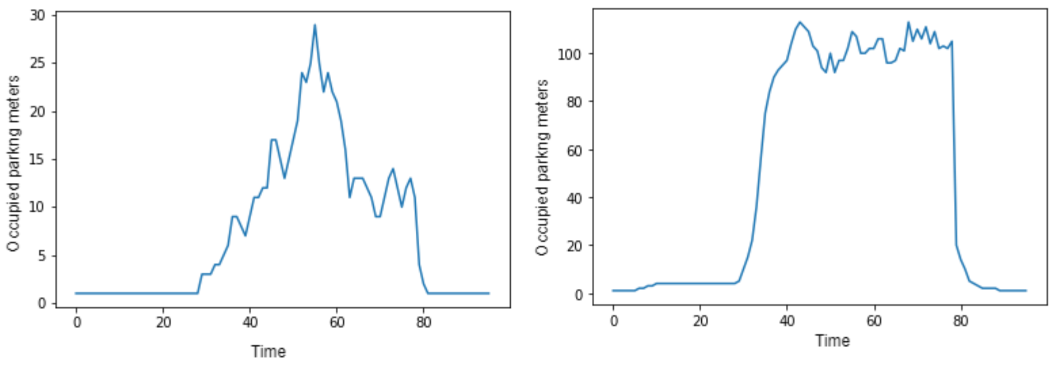


Figure 2.14: *Day by quarters TS for area S.Giacomo street (part 1 on the left, part 2 on the right) using myCicero data.*

Thus, we can conclude that the two data sources are not interchangeable, nor an accurate forecast can be made by using just one of the twos since different behaviours may appear for a same considered area. The mean of such difference is still unknown, and it may rely on two different classes of people that use parking meters and “myCicero”. However, by using the provided information, it is impossible to deepen such a speculation.

Moreover, it is quite evident how territory peculiarities influence the trend of stops along daylight hours, and so they should be taken in consideration when developing the solution.

In the end, we understand how each prediction must rely on a limited portion of territory since the behaviour may vary quite suddenly for nearby points of the space.

2.5.2 Area #2

The selected area is a sampling of the downtown having a circular-like shape (Figure 2.15).

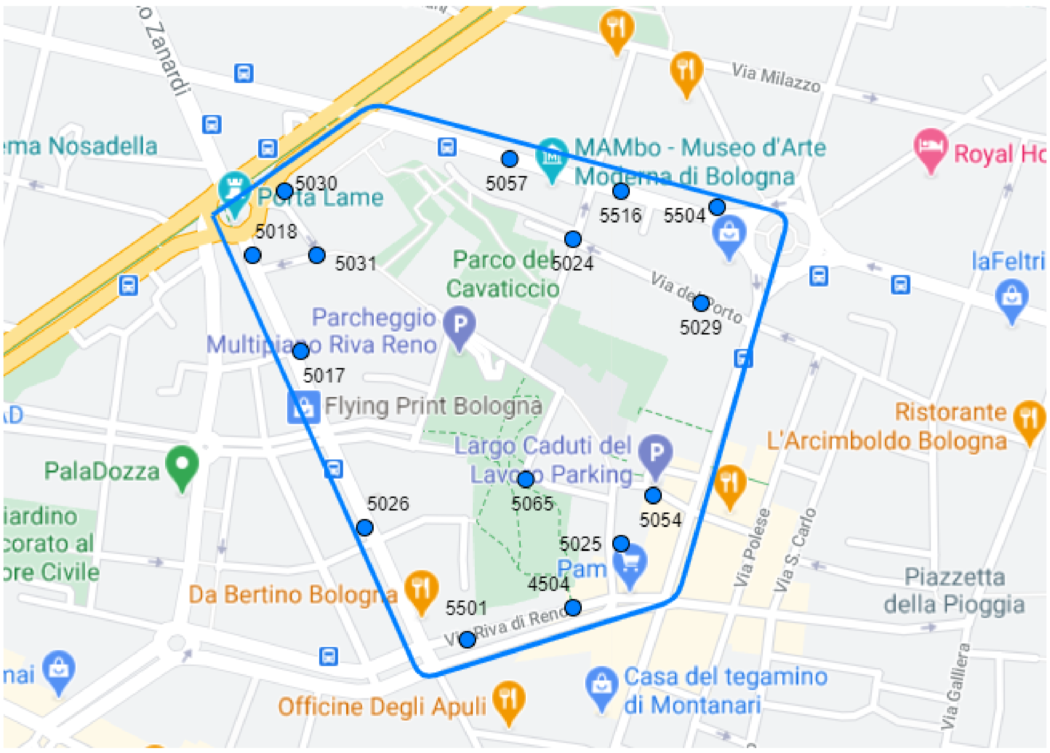


Figure 2.15: *Area made up by the intersections of few different streets: Via delle Lame, Via Riva del Reno, Via Marconi, Via Minzoni, SS9. Parking meters (along with their ids) are highlighted by the blue markers.*

First of all, the stops trend along the day is reported in Figure 2.16. While for "myCicero" data the behavior is not informative at all (almost a constant time series), the parking meters one can provide few novel information about the area with a higher occupation in the afternoon hours at the expense of the morning ones.

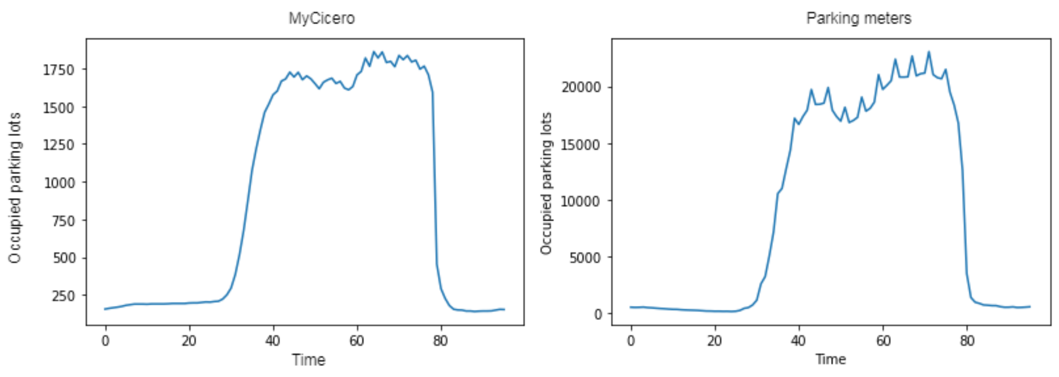


Figure 2.16: *Times series "day by quarters" for "myCicero" and parking meters on the sampled area.*

However, the area is still too wide to be considered atomic since different sub-areas may show completely different patterns, resulting in misleading results. We split the whole area in five sub-areas, namely:

- "Via Gardino" and surrounding area
- "Via Caduti del Lavoro" and surrounding area
- "Via Castellaccio" and surrounding area
- "Via Riva del Reno" and surrounding area
- "Via delle Lame" and surrounding area

It is confirmed that different areas expose completely different behaviors, and such behavior is less informative as the number of parking lots inside that area increases.

Riva del Reno

"Riva del Reno" area, whose behavior is reported in Figure 2.17, exposes the less informative behaviour among the five areas and it is the only one containing a big parking (more than 100 parking lots, map in Figure 2.18). This may underline a correlation between the number of parking lots and the information a time series can provide (a deepening in such direction for all the territory in general is going to be executed in Chapter 5).

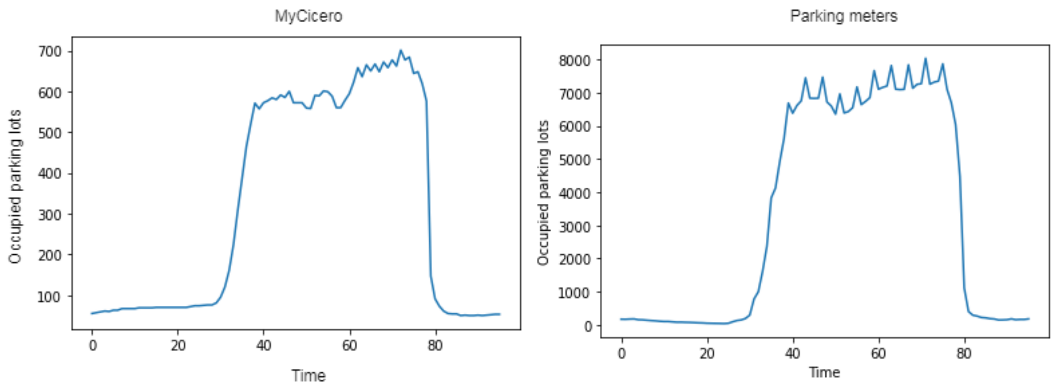


Figure 2.17: *"Riva del Reno" area, day by quarters TS for "myCicero" and parking meters both.*

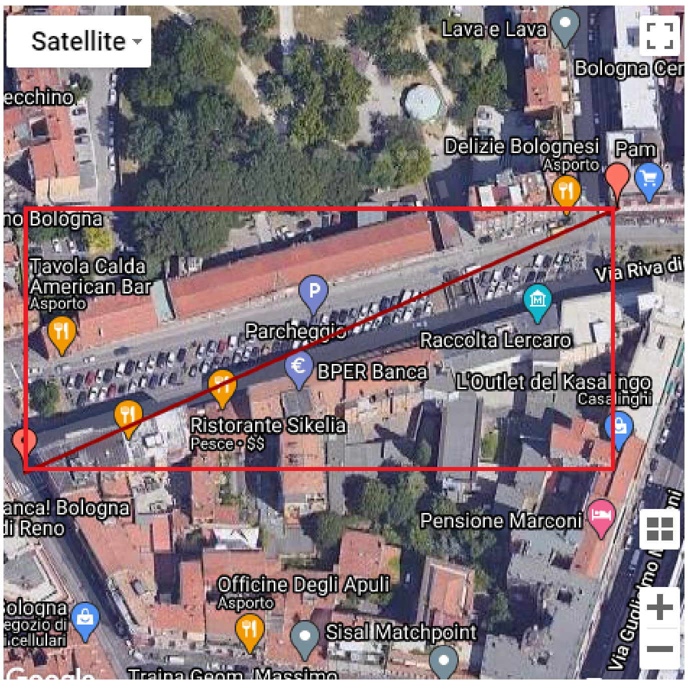


Figure 2.18: *Street view of "Riva del Reno" area.*

Via delle Lame

Boundary and behavior of the area are reported in Figure 2.19 and 2.20, respectively. The parking meters within such area are {5017, 5018, 5030, 5031}. As we can notice, "myCicero" and parking meters follow a compliant trend with an increase of stops during afternoon hours.

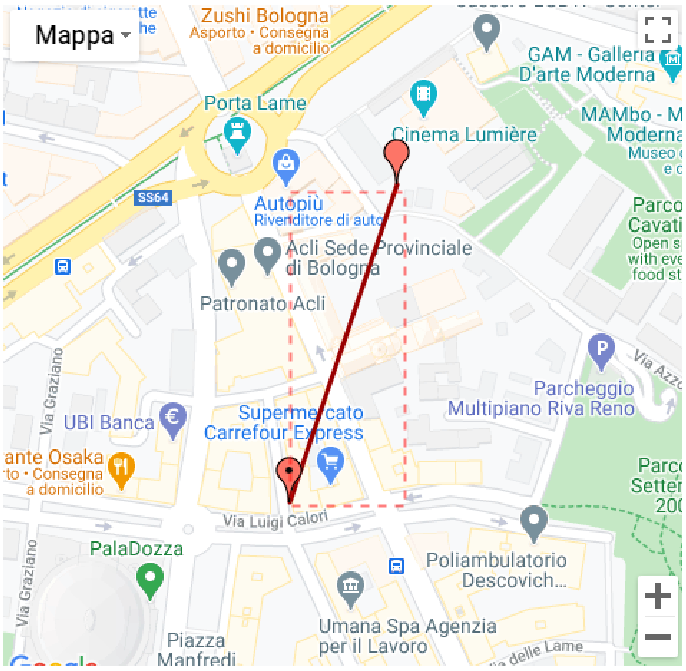

Mappa
Zushi Bologna
Porta Lame
Cinema Lumière
GAM - Galleria D'arte Moderna
MAMbo
Autopiù
Acli Sede Provinciale di Bologna
Patronato Acli
Parcheggio Multipiano Riva Reno
UBI Banca
Supermercato Carrefour Express
Via Luigi Calori
PalaDozza
Poliambulatorio Descovich...
Umana Spa Agenzia per il Lavoro
Piazza Manfredi
Via Graziano
SS64

Figure 2.19: *“Via delle Lame” area.*

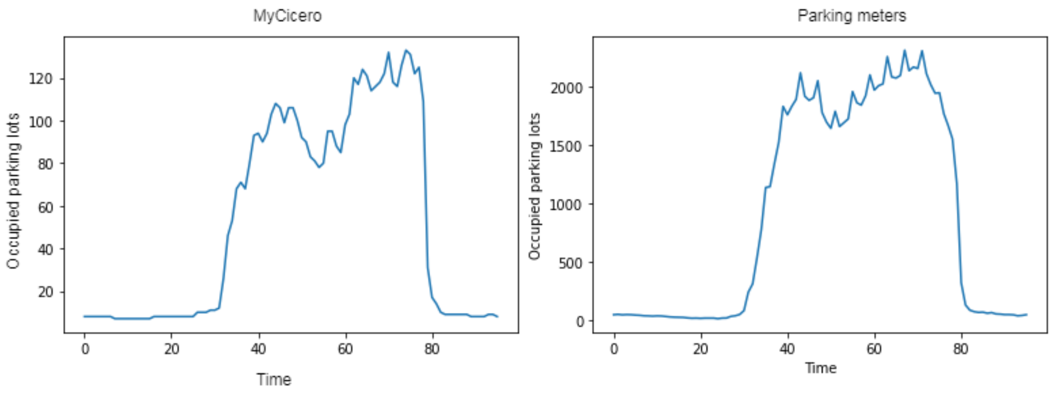


Figure 2.20: *“Via delle Lame” area, day by quarters TS for “myCicero” and parking meters both.*

Now, we execute a robustness check by adding some jitter to the area positioning. The goal is to detect if the behavior substantially changes and how much the positioning may affect the results.
The initial area is identified by the rectangle with vertices: bottom-left = (44.5005, 11.3337), top-right = (44.5022, 11.3345). The jitter will make the area out of phase by moving it along any axis by a quantity equal to the half

of its length, that is ± 0.008 on latitude and ± 0.004 on longitude. Results reported in Figure 2.21 show that:

- *LEFT*: the behavior changes in a non-negligible way since novel parking lots, now considered in the area, influence the results.
- *RIGHT*: similar behavior with little variations. By moving the cell on the right, the introduced portion of the area does not contain paid parking lots at all, and so it is irrelevant for the analysis; meanwhile, we are losing on the left few parking lots, but such a loss does not compromise the behavior that is still captured by the remaining ones.
- *UP*: Almost the same behavior, similar considerations to the *RIGHT* case.
- *DOWN*: Behavior changed in non-negligible way, similar considerations to the *LEFT* case.

Thus, we can conclude that the identified behavior highly depends upon a correct positioning of the area. Obviously, areas containing no paid parking lot will not falsify the results, and so they can be put in one region rather than another. Hence, it is confirmed as a random division of the territory without considering the road network and/or the districts' boundaries is not the best solution at all.

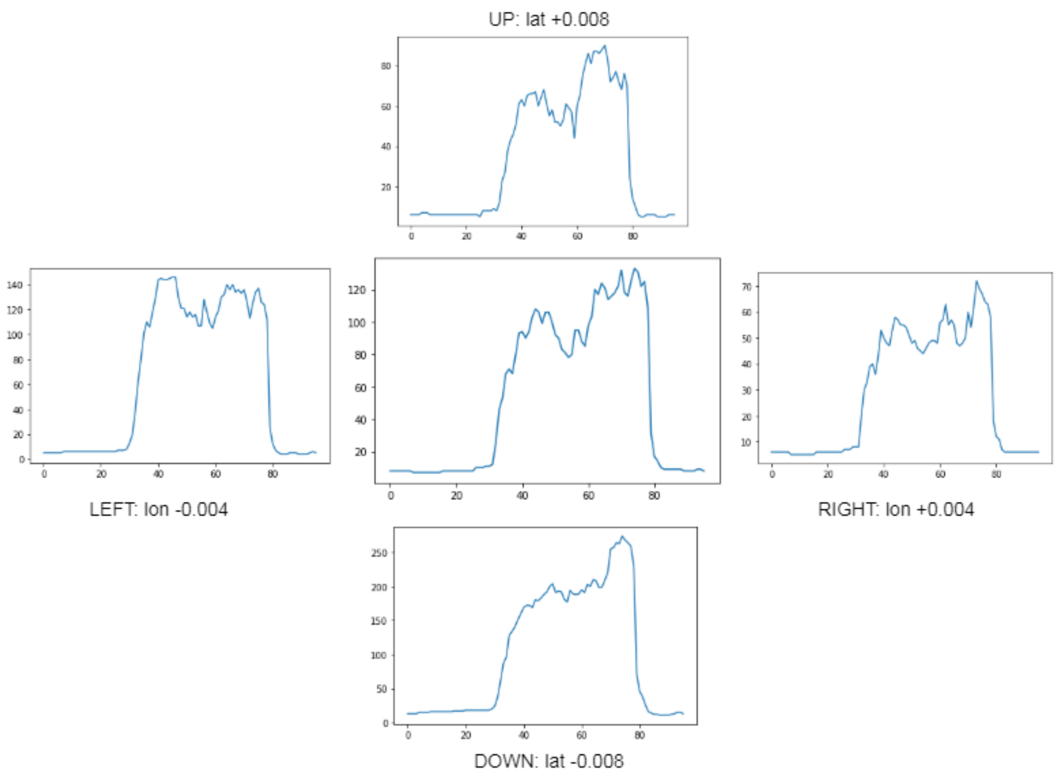


Figure 2.21: *"Via delle Lame" behavior (center) and the four out of phase ones using "myCicero" data.*

Via Castellaccio

In the end, the "Via Castellaccio" area (map in Figure 2.22, behavior in Figure 2.23) is reported to underline two concepts, namely:

- The is no compliance at all between the trends, with the two data sources exposing opposite behaviors (peak of the stops in morning for "myCicero" against peak in the afternoon for parking meters).
- The area has been selected since there is a little number of stops and, at the same time, it is one of those having the most informative behavior. As stated before, a correlation may exist between the number of parking lots and the information provided by the trend of stops.

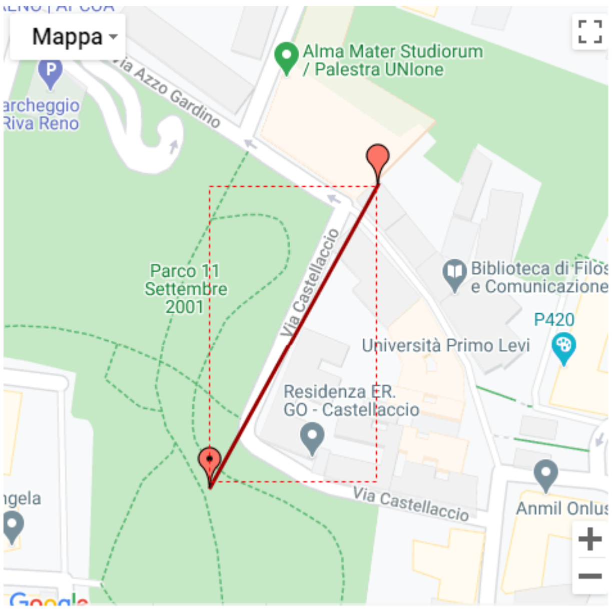


Figure 2.22: *Map of "Via Castellaccio" area.*

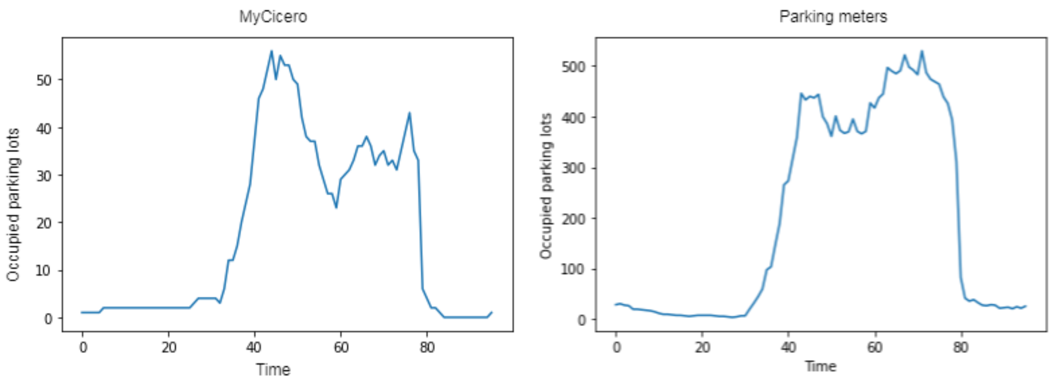


Figure 2.23: *"Via Catellaccio" area, day by quarters TS for "myCicero" and parking meters both.*

2.5.3 Conclusions

We can summarize the following findings from the previous analysis, namely:

- Usage of a random division of the territory might be impossible if the solution must be focused on effectiveness since it does not consider local territory peculiarities that emerged to be of primary importance. Indeed, the correct positioning of regions is essential, and little variations so that regions with different behaviors are intersected and/or mixed make the solution lose effectiveness.

- Euclidean distance is a tricky dimension when dealing with this domain since "the little the distance the similar the behavior (et vice versa)" is not a fair assumption in general since local peculiarities create behavioral patterns in clusters of arbitrary shapes. Hence, equal regions having trivial shapes (circle, rectangle, etc.) for all the territory will not fit well the problem.

  Moreover, regions size must be accurately tuned and very little (edges over 150m might already mix different behaviors) but, on the other hand, creating a division using too shorter edges would generate a huge number of regions, and this might affect the feasibility in term of computational complexity and in term of quantity of data required to correctly generalize assumptions (i.e., too much regions correspond to a too meager quantity of data for each region).

- Given an area, parking meters and "myCicero" data are not compliant in term of behavior during daylight hours in most of the cases. Hence, the usage of "myCicero" data solely might be impossible if solution must be focused on effectiveness. Notice that this is not due to wrong territory split up since analysis on such direction has been executed starting from parking meters (that partially reflects the street network that, in turn, reflect territory features), and then considering "myCicero" data in such portion of the territory.

- Parking meters and "myCicero" data become more and more similar as the size of the considered area grows and as the number of parking lots in such area increases.

2.6 Miscellaneous

Very interesting results emerged from further analysis. However, the utility of such results may live aside from the current project itself or they have no precise classification, and so they are reported in this dedicated section.

2.6.1 Arrivals vs Stops

Datasets have been analyzed by counting the arrivals instead of modeling the stop along the time (merely associating the stop length to 1 minute so that each transaction is exactly associated just to one quarter of an hour).

The main goal is to understand if such simplified model still correctly shapes the behavior of the stops (and so the TS model exposed in Section 2.2 is an overkill).

The comparison of the two models is shown in Figure 2.24 using the *day by quarters* time series for parking meters and “myCicero” both. As expected, the times series trends for arrivals and stops are quite different.

Rather, the most interesting aspect emerges from the arrivals time series of parking meters. Indeed, it is surprising how it comes out an exact seasonality of length four (Pearson autocorrelation diagram is not reported for conciseness) with the values arranged in a “lightning bolt” shape. We have local minimum and maximum associated to the first and fourth quarters, respectively. The difference maximum-minimum in term of number of arrivals in too high to be associated to fluctuations (almost 3x factor), and this pattern is probably explainable by the fact that most of the planned activities (meetings, lessons, openings, etc.) start at the stoke of the hour, and so people tend to arrive few minutes before. Consequently, the minimum is reached just after since the aforementioned activities have just begun, and so few people are expected to arrive.
Moreover, we can suppose that people using parking meters and “myCicero” aim for different goal, and such aspect partially emerged in the Section 2.5 when it was found out that often times series of parking meters and “myCicero” do not share the same behavior along the day when executing situated analysis on restricted areas. Such hypothesis is now strengthened since there is no strong seasonality is “myCicero” data, and so it means that the above motivations do not hold any more for “myCicero” customers. On the contrary, they probably are usual customers that use the application for various purposes (against occasional customers for parking meters).

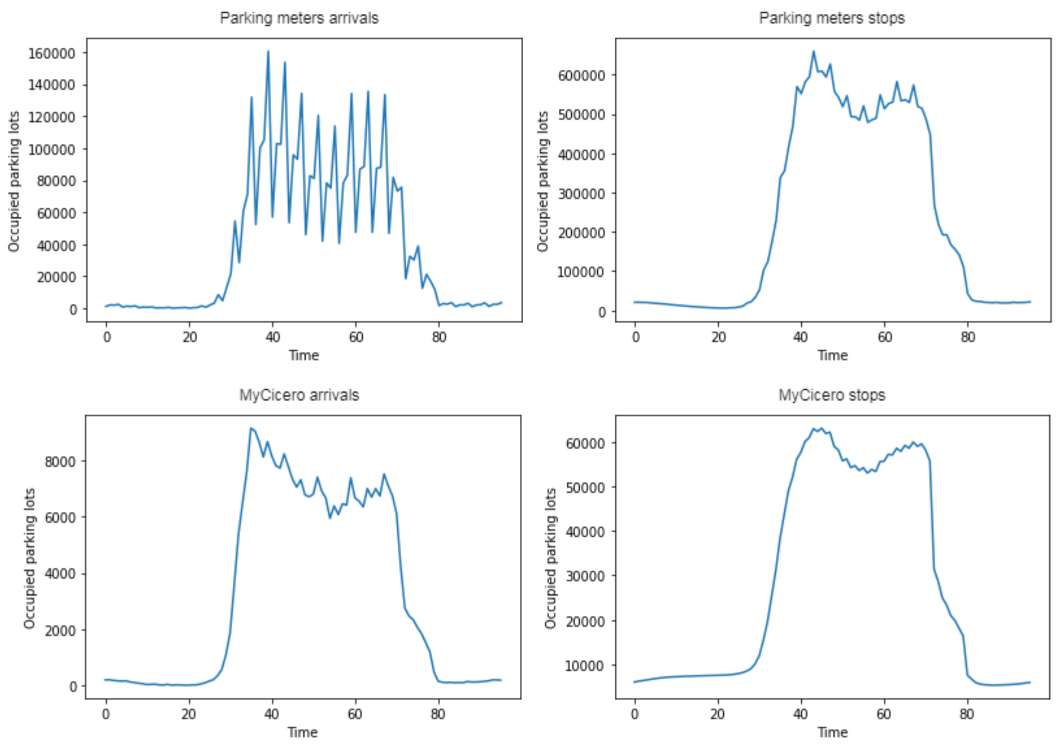


Figure 2.24: *Comparison between arrivals (first column) and stops (second column) for parking meters (first row) and "myCicero" (second row) both.*

2.6.2 Stop length and late afternoon factor

The subject of the analysis is the stop length, and the distribution of its values is reported in Figure 2.25 for "myCicero" and parking meters both.

Firstly, we notice that "myCicero" data is almost all concentrated in few different values corresponding to multiple of hours (60 minutes, 120 minutes etc.), while it does not happen for parking meters values. This is indirectly due to the "myCicero" application logic that hints a default length of one hour to users, and such time seems to be kept by most of the users since the exceeding amount will be refunded if they are going to leave before the validity of the stop expires (for example, a user activates a stop with length 60 minutes, but he leaves after 48 minutes. The cost of the remaining 12 minutes will be paid back). By keeping in mind such principle, few users may select two or more hours if they are going to stay that long just by tapping on hours knob, and so the minutes one is not going to be used so much.

When talking about parking meters, the main subject is the user and the coins it disposes. Hence, there is different type of reasoning that is no more focused on the stop length expressed using time measure (minutes and hours).

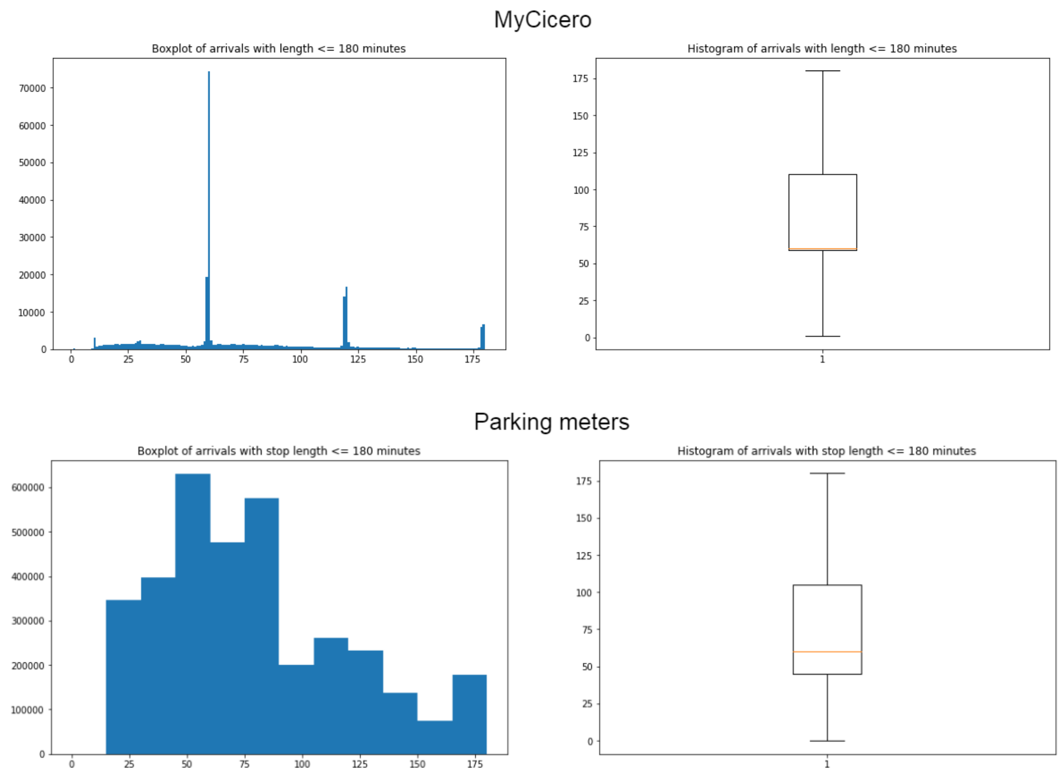


Figure 2.25: *Distribution of stop lengths for "myCicero" and parking meters. Just stops with length less or equal to 180 minutes are considered (few tail values are dropped so that a nice visualization can be provided).*

Seen the previous considerations, the goal is to discover if different types of user do exist. In particular, the main question is about which type of user is going to set a stop length detailed to the exact number of minutes and why. We divide the stops in two groups by their length: the ones that have a stop length multiple to hours and the ones that do not. The stops trend is reported in Figure 2.26.
As we can notice, there is a peak in the late afternoon for the stops whose length is not multiple to hours, and such peak dramatically increases when considering only stops shorter than one hour. Moreover, the major peak before 6 p.m is followed by a minor one just before 8 p.m.
On the contrary, the peak for stops with length multiple to hours are mainly distributed on morning hours.
Such a phenomenon, that actually can be slightly found in parking meters data too where the peak is not so high even though it exists (figure not reported for conciseness), can be addressed to the fact that the paid period ends at 6 p.m. or 8 p.m. depending on the area, and so people tend to do shorter stops during

the previous hour to avoid a waste of money. However, while it is justifiable for parking meters, it should not happen for "myCicero" since any minute in excess out of the paid period is not going to be charged.
The cause of the abnormal peak probably resides in the logic of the application. Indeed, the default suggested length is one hour for the cases when the paid period expires before too (e.g., paid period expires at 6 p.m., user arrives at 5:20 p.m., stop straddling between paid and free parking periods if using a one hour long stop). When in these cases, even though the application is not going to charge the costumer for the exceeding minutes, users normally set the stop length so that an exact length till 6 p.m. (or 8 p.m.) is considered. In order to do that, the minutes are fine-tuned using the proper knob in the application, and so stops with the most disparate minutes emerge (e.g, user arrives at 5:24 p.m. and sets a 36 minutes stop to avoid a possible extra charge of 24 minutes).
This means that most of the users do not trust the application from charging the correct amount when the stop is straddling between the paid and the free parking periods. Hence, in order to solve such a problem, a suggested length till the end of the paid period may be the hinted one instead of the regular value of 60 minutes (or at least, without changing the application logic, a textual feedback may be displayed to tell the safety of the operation).

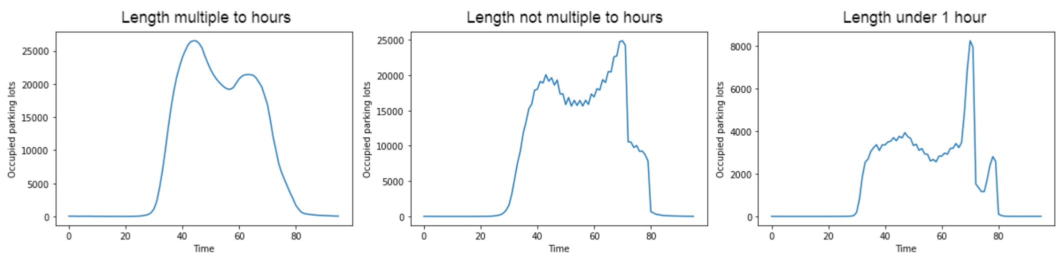


Figure 2.26: *Different day by quarters time series selecting only particular transactions for "myCicero" data.*

Chapter 3

Problem formalization

We should find out a high-level indication of the parking occupation in a given area A and in a given time t. Obviously, such a value cannot be estimated as once for all the territory. Therefore, the domain must be split up in several regions so that a different occupation value can be estimated for each one.

Section 3.1 exposes the way the occupation status is calculated, while Section 3.2 exposes different ways and approaches that can be exploited to obtain a division of the territory.

3.1 Occupation status estimation

Given a time t and an area A, the goal is to obtain a categorical occupation status for the area A at time t. In order to find such occupation status out, we firstly calculate the Occupation Percentage Value (**OPV**), that is a numerical value in range [0,1] representing the fraction of occupied parking lots, and then we map the OPV to a categorical value.

3.1.1 OPV estimation

For the time being, the number of occupied parking lots can be modeled by the sum of the three following components: "myCicero", parking meters, seasonal tickets. Hence, the optimal solution would be calculated using the following formula:

$$OPV(t) = min\{1, \frac{MC(t) + PM(t) + ST(t)}{C}\} \tag{3.1}$$

where:

$MC(t)$ = estimated number of occupied parking lots for “myCicero” at time t;
$PM(t)$ = estimated number of occupied parking lots for parking meters at time t;
$ST(t)$ = estimated number of occupied parking lots for seasonal tickets at time t.
C = capacity of the area in term of number of parking lots[1].

The number of occupied parking lots is not a ground value, but it can be forecasted in any way. In our case, it will be formalized as a *regression* problem, and the TS model already explained in Section 2.2 will be used to shape the data.
The capacity of the area is not known too, and it must be calculated.
For safety, the *“min”* operator is used to bring the final OPV back in [0,1].

As projects constraints force, it must be possible to easily add further data sources (or update the employed ones) to an already existing system, and so the Formula (3.1) can not be directly applied since all data is mixed together[2] and the capacity is calculated once and for all. Hence, each data source should be separately considered, and this does not represent a problem when forecasting the number of occupied parking lots (just creating three different time series and executing a forecast on each one).
The question raises when considering the capacity because it can not be estimated as a unique value, and so partial capacities are going to be calculated on each data source. This way, each data source has its own estimated capacity and it is completely independent from the others.

Given a region, we use the following formula to merge the partial results into the final OPV:

$$OPV = min\{1, \frac{\sum_{i=0}^{n} forecast_i}{\sum_{i=0}^{n} capacity_i}\}$$

where:

n = number of employed data sources.
$forecast_i$ = estimated number of occupied parking lots for the i-th data source.
$capacity_i$ = partial capacity of the i-th data source.

[1]Obviously, it is intended as the number of paid parking lots solely.

[2]Indeed, all dataset would be merged, and a unique time series representing the overall stop trend would be created.

Simply, the idea is to calculate the total number of occupied parking lots by summing partial forecasts on each data source, and then to divide the resultant by the sum of partial capacities.
Notice that it is equivalent to a weighted arithmetic mean on the singular OPVs with weights imposed by the partial capacities.
The different weighting is required since the amount of data can highly differ from data source to data source.
For example, let us consider two data sources where we have 3 out of 10 occupied parking lots and 80 out of 100, respectively. This would lead in a resulting OPV of 0.55 ((0.3 + 0.8) / 2) when using a simple mean, while the weighted one will return an OPV of 0.75 ((80 + 3) / (100 + 10)) by implicitly giving more importance to the prominent data source, and this intuitively better represents the real occupation.

3.1.2 Forecasting model: Prophet

The *"Prophet"* forecasting model is the one adopted to estimate the number of occupied parking lots. The model, developed by Sean J. Taylor and Benjamin Letham in 2018 [11] and based on the previous work of A. C. Harvey and S. Peters [12], has been properly designed to catch recurrent business factors that other common approaches may not consider without the development of ad hoc solutions.

The main distinguishing feature of the model is that it straddles between the statistical forecasting approach and the judgemental one, and so it attempts to inherit the advantages of them both, namely:

- **Flexibility:** the model can be easily tuned to handle multiple seasonalities and trend changes. This is a key factor that emerged during the previous data analysis stage (see Chapter 2).

- **Computational complexity:** the computation time of the training stage is limited. This feature is of primary importance because hundreds of models may be potentially created (one for each different region).

- **Explainability:** the model allows to easily tune each additive component, thus producing interpretable changes in the solution.

- **Sampling time:** it is not required a regular sampling over time, and so sparse missing values do not need to be interpolated. Taking into account that we have the availability of full observations for the training stage, this peculiarity may be relevant in the condition where the data of a specific region is temporally unavailable.

The main idea is to use an additive model to decompose a time series into four components:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \qquad (3.2)$$

where:

$y(t)$ = value assumed by the time series at time t.
$g(t)$ = function that models the non-periodic changes, i.e. the trend.
$s(t)$ = function that represents the periodic changes, i.e. seasonality.
$h(t)$ = bias function that represents the effect of holidays.
ϵ_t = error (any idiosyncratic changes not accommodated by the model).

Trend component

Two different designs of the trend are proposed: saturating growth (non linear), piecewise trend (linear).
The piecewise trend was assumed in order to simplify the model and alleviate potential overfitting.

The trend changes its growing rate periodically depending on various factors (especially when dealing with business aspects), and so the *change-point* concept is to be defined. A change-point is a point in the time where the high-level trend changes significantly.
Let us consider S change-points at times s_j, $j = 0, ..., S$, and a basic growing rate of k. The linear model considers a constant growing rate between any two change-points, and variations in the growing rate are stored in a vector δ where δ_i is the rate change that occurs at change-point s_i[3]. Hence, the trend at time t_i is calculated from the starting rate k and summing all changes occurred till t_i.
The total trend component is calculated using the Formula (3.3).

$$g(t) = (k + a(t)^T\delta)t + (m + a(t)^T\gamma) \qquad (3.3)$$

where:

m = offset parameter to connect the endpoints of adjacent segments identified by the change-points.
γ = vector used to make the function continuous ($\gamma_j = -s_j\delta_j$).

[3]Notice that it is the relative change that occurs from the previous rate, and not the overall variation from the basic rate k.

A proper vector $a \in \{0,1\}^S$ is defined to so that the rate a time t simply is $k + a(t)^T\gamma$ where:

$$a_j(t) = \begin{cases} 1, & \text{if } t \geq s_j \\ 0, & \text{otherwise} \end{cases}$$

Change-points can be manually or automatically selected both. In the latter case, a sparse prior $Laplace(0,\tau)$ is added to δ so that few change-points are selected from a large number of candidates (conceptually equivalent to L1 regularization). The parameter τ controls the flexibility of the model in altering its rate (the higher the τ, the higher the trend variations that can be imposed by the model).

Seasonality component

The seasonality component summarizes information about seasonalities at different granularity levels (daily, weekly, etc.).
A periodic function of time $s(t)$ is required to catch recurrent patterns, and a *Fourier series* is adopted to approximate it (Formula (3.4)). Indeed, terms of the series may be seen as refining patterns at different time granularities, and so an increase in number of terms allows the model to consider shorter seasonalites. On the other hand, an higher number of terms increases the overfitting of the model obviously.

$$s(t) = \sum_{n=0}^{N} a_n cos\left(\frac{2\pi nt}{P}\right) + b_n sin\left(\frac{2\pi nt}{P}\right) \tag{3.4}$$

where:

N = number of terms;
P = seasonality length in days (e.g., weekly patterns -> 7);
a_n, b_n = parameters to be tuned by the model.

Holiday component

The holiday component just biases values in correspondence of any holiday. A hyper-parameter k_i (holidays prior scale) that represents the effect on the predicted value is to be defined for each detected holiday. Such effect can be an increase or a decrease either depending on the domain (in our case, it will be a decrease since the parking is free during holidays).

Moreover, the component of a certain holiday may be suffered by the days before and after both, and so the effect is propagated to a window of days (such aspect can be tuned by apposite parameters).

Fitting and forecasting

The fitting is executed by exploiting the L-BFGS technique (an optimization algorithm in the family of *Quasi-Newton* methods) to find a maximum a posteriori estimate, namely to find the best parameters configuration to fit the observed data.

Let H be the time horizon adopted for the current forecasting, i.e. the period in the future we are going to predict (e.g., 1 day).
We define $\hat{y}(t|T)$ as the forecast at time t made with historical data up to time T and $d(y, y')$ as a distance metric to measure the error. The choice of the distance metric is problem-specific (MAE, MAPE, etc.).

3.1.3 Capacity estimation

The capacity of a given area A and calculated on a given data source DS is the ever-maximum observed number of simultaneously occupied parking lots, and such a value can be banally retrieved by taking the maximum value of the *year by quarters* TS. For quite small portions of the territory, such approach would quite well approximate the real capacity of the area since we can fairly assume that (almost) all parking lots were occupied at least in one moment of the history (1-year historical depth should be enough), while it can not be considered true when dealing with wider areas. However, this latter aspect is not a problem of particular importance since predicted values and capacities are both calculated on the same time series coherently.

An issue emerges when considering each data source singularly. Indeed, even though the average number of parking lots occupied by a certain data source may be quite stable over time to an high-level look, an outlier (higher parking occupation than the normal one) may influence the estimated partial capacity in a not negligible way, and so the relative total capacity will be overestimated[4]. On the other hand, this cannot happen if all the data sources were used together: indeed, if the time series reached such a maximum value, it means that (an approximation of) such number of parking lots actually exists. If follows a clarifier example in Figure 3.1. As we can see, an outlier value sets the maximum of B up to a very high value respect to the average ones, and so the partial capacity estimated for B will be distorted.

[4]Overestimated respect to the values registered by the time series, not the actual capacity of the territory.

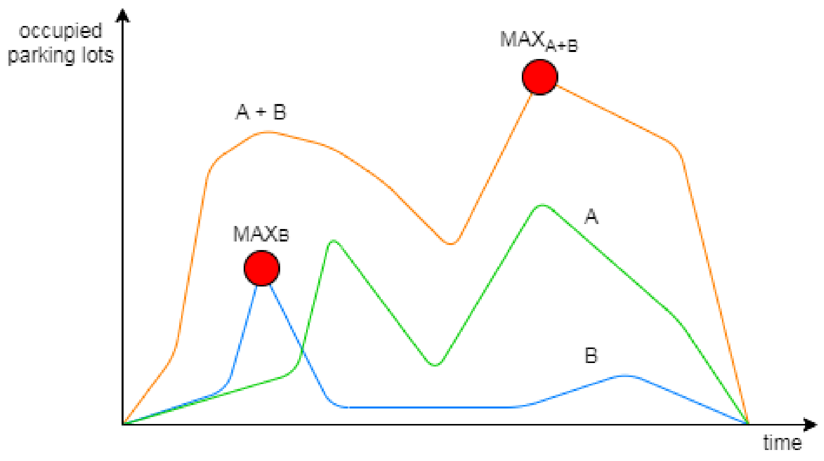


Figure 3.1: *Three example TS are created by data sources A and B. The orange one is the sum element-wise of A and B (i.e., $AB(t) = A(t) + B(t)\ \forall t$). The maximum value of B can potentially vary in range [0; max(A+B)] (0 if B is constant with value 0, max(A+B) if $B(t) = MAX_{A+B}$ and $A(t) = 0$).*

The correct estimation of partial capacities has a primary importance since it will influence the final OPV, and a too high partial capacity dictated by an outlier value will lower all the final occupation results. Hence, to deal with such an issue, the capacity is estimated once outlier values have been fixed. A value v is considered an outlier if its distance from the mean value of the series is greater than N times the standard deviation:

$$abs(v - mean) > N * std$$

where:
$mean$ = mean of the TS;
std = standard deviation of the TS;
N = an hyper-parameter to be tuned; the common suggested value is 3.

Now, the total detected capacity is going to be underestimated, and this carries a further implicit advantage when dealing with uncertain values that can be associated to one status or another depending on the adopted parameters (see Section 3.1.4). Indeed, the underestimation of the capacity is preferred to the previous overestimation since higher occupation status are going to be predicted in the first case, and this may produce “false negatives” where the users can find parking even though the application shows an high occupation (it can be seen as a fluke, and divert the user focus away from the application). On the contrary, when capacities are overestimated and lower occupation status are calculated, the user may not easily find a free parking lot despite the hinted value indicates the opposite, resulting in low a satisfaction of the users.

3.1.4 Occupation percentage to occupation status mapping

We consider four different occupation status, namely:

- Low occupation (displayed color: green);
- Average occupation (displayed color: yellow);
- High occupation (displayed color: red);
- Very high occupation (displayed color: purple).

Any OPV must be mapped to one of the above values.

The range [0,1] is split up in K bins, each one associated to a specific occupation status. We consider four different status, so we use 4 bins (K = 4). Hence, we have K-1 hyper-parameters H* (H1, H2, H3 in this case) to be tuned (Figure 3.2).

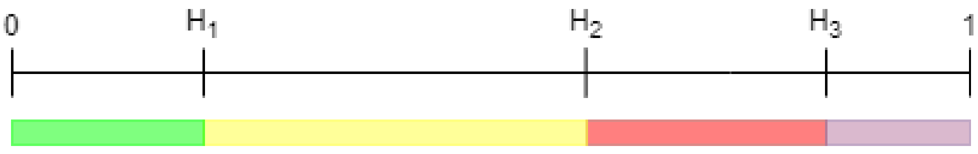


Figure 3.2: *Four probability bins with length tuned by three H*.*

It is clear that a model that contemplates bins of equal length does not suit well to the problem (for example, we would already have a high occupation at 50%). There is not a unique optimal solution but, as common sense suggests, a good tuning for such H* may be [0.6, 0.8, 0.95] (Figure 3.3).

Though the goodness of such a configuration is also supported from data mined about occupation trend along the daylight hours in Section 2.4, it is a theoretical one, and it is made a priori from the development of the solution itself. Hence, such parameters, if required, may be fine-tuned during the performance evaluation step in order to suit the developed models.

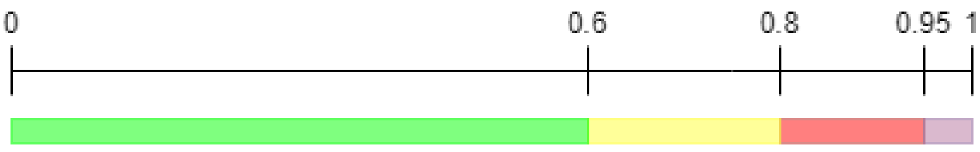


Figure 3.3: *Split points set up in [0.6, 0.8, 0.95]. This way we have a good tradeoff, having low occupation till 60%, average occupation from 61% till 80%, and so on.*

3.2 Geometric models

The territory must be split in several regions so that a forecast can be executed upon each one. We consider two different approaches, namely:

- Polygon-oriented solutions
- Graph-oriented solutions

It follows a brief description of each one (explanation of the theoretical model, pros, cons).

3.2.1 Polygon-oriented models

Given a starting set of points, the whole area is divided in polygons by assigning each point of the space to the closest starting point (i.e., *Voronoi tessellation*).
There are few different possibilities for the choice of the starting points, namely:

- Usage of random points equally distributed over the territory.
- Usage of the locations of parking meters.
- Usage of equally spaced points over the territory.

The latter possibility is a remarkable one since it creates identical squared regions by implicitly imposing the following two constraints, that is:

- Each region will have a squared shape.
- Each region will have the same length of sides.

Figure 3.4 reports an example of all the three possibilities.

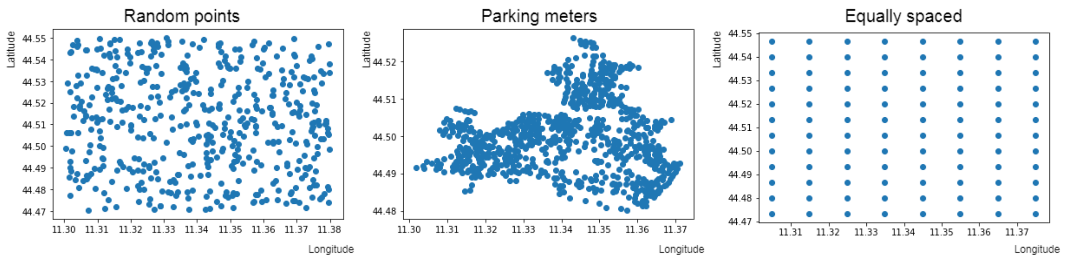


Figure 3.4: *An example of the three possible approaches to choose the starting points: 500 random points, location of parking meters (about 800 points), equally spaced points (8 on longitude axis, 12 on latitude axis).*

Figure 3.5 exposes an example of a Voronoi tessellation, and so a possible result.

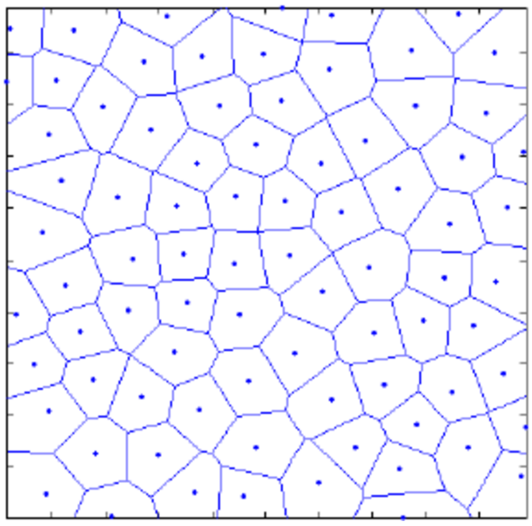


Figure 3.5: *Example of a Voronoi tessellation.*

Pros:

- Geolocated data can be easily managed (just need to associate each transaction to the right region using latitude and longitude).
- It is possible to partially catch each area behaviour by using parking meters coordinates as the set of starting points. Indeed, parking meters reflect the road network that, in turn, reflects the territory peculiarities. If parking meters data source is not available, the solution is still applicable by choosing a different configuration of starting points.
- The model is open to novel approaches for the choice of the starting points (e.g., centroids of the districts using the territory map).

Cons:

- Each region is shaped as a convex polygon, and this implies it is not possible to have interwoven regions. Hence, the expected result will be a map with spots of different colors.
- Since the final feedback to users should be a color to be displayed for each road, it is required a service that permits to extract all roads contained in a given region.

- As shown is Chapter 2, the Euclidean distance may be a tricky dimension in such a context.
- When the starting points are not configured to be the locations of parking meters, the solution does not consider at all any territory peculiarity and the region positioning is quite random (aspects that emerged to be of primary importance both in Chapter 2), and so the solution effectiveness may be limited (Figure 3.6).

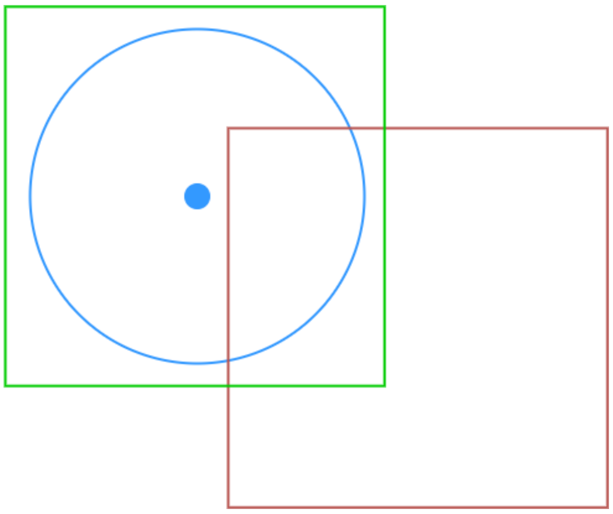


Figure 3.6: *Let us consider a point of interest (any facility, public park etc.) whose features may influence its surrounding area (blue). In the best case, the behaviour will be correctly seized by the region (displayed as a square for simplicity) if its positioning is right. Otherwise (red), a cell may contain partial information that will mix up with external ones, resulting in an informative power loss. When the division of the territory is totally random, such a casuistry is not avoidable.*

3.2.2 Graph-oriented models

The territory is directly evaluated on the street network using a graph where edges are streets and vertices are their intersections. The weight of each edge is the Euclidean distance between the two vertices it connects. The workflow would be divided in few steps, namely:

- Generate a set of starting vertices in the graph. If any point of interest is not already a vertex, the graph is adapted to suit all the starting points.

- Label the graph to associate each edge to the nearest starting point on graph routes.
- Associate each transaction of any data source to an edge using longitude and latitude coordinates and, then, assign such transaction to the starting point the edge is associated to.

Let us consider a slice of the road network (graph reported in Figure 3.7). Then, we suppose to divide such graph in two regions. Figure 3.8 reports the solutions realized by applying the two different approaches. Notice that, taken in consideration the information mined on previous analysis stage too, the graph-oriented approach will likely create a better solution than the polygon-oriented one.

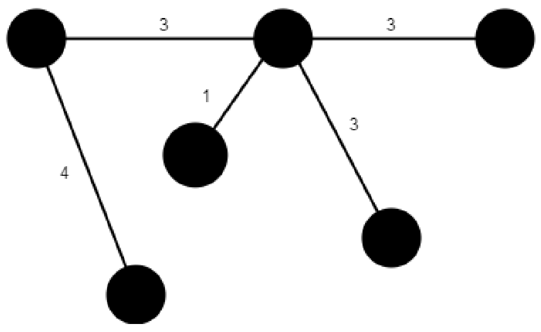


Figure 3.7: *An example graph. Given an edge, its weight is the distance between the two nodes it connects.*

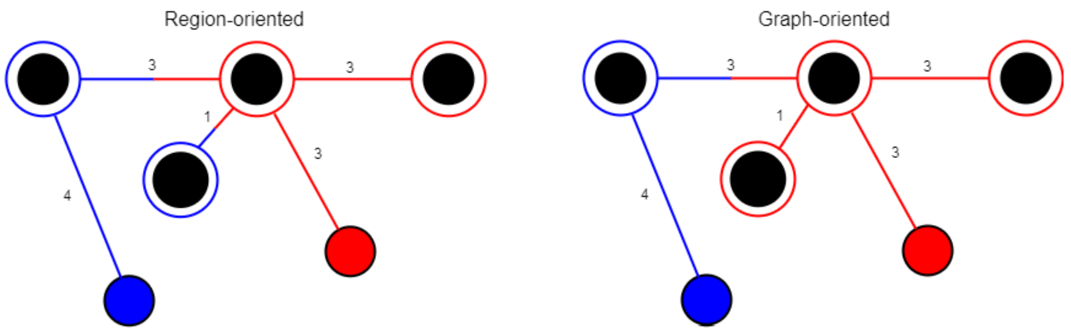


Figure 3.8: *Result of polygon-oriented and graph-oriented models by creating two regions (red, blue) on an example graph. Blue and red nodes are the starting points.*

Pros:

- The behaviour of each area is quite well caught since the division of the territory is made upon the road network (against polygon-oriented models where the division is totally random or such aspect is approximated by using the parking meters locations).
- The distance metric is very accurate since it is calculated on the street network (differently from the Euclidean one that resulted to be tricky, as exposed in Chapter 2).
- There is no need of any auxiliary service to extract roads within a given area since the output is already an estimation for each street (or street segment).
- The street segment is the building block, and so regions of arbitrary shapes can be created and interlaced.

Cons:

- The street network is mandatory to apply the model.
- Complexity[5] of this approach may be higher than to the one required for polygon-oriented models.

[5]Time and space complexity as well as the effort to develop and implement the solution.

Chapter 4

Solution development

4.1 System architecture

Figure 4.1 reports the complete high-level architecture of the system.
The “myCicero” platform is represented by the component **MyCicero**, and it is the only front-end access point to the forecasting service.
The back-end architecture relies upon the Service-Oriented Architecture idea (SOA) and it is made up of seven components, namely:

- **RunningSystem**: Main service of the back-end that periodically updates the occupation status of each region.
- **SystemSetup**: Application used to execute one-off setup tasks.
- **RoadCutterService**: Service used to extract the road segments contained in a given area. Such a service in not included in the **SystemSetup** component since it has been developed separately.
- **ForecastingService**: Service that contains the forecasting engine to estimate the number of occupied parking lots.
- **StorageService**: Service that stores any kind of worthwhile information about regions in a persistent way.
- **ParkingOccupationService**: Facade service used to access the overall forecasting service. It also handles some aspects required to make the functionality work on the final mobile application (e.g., find out all regions close to the user).
- **GatewayService**: Broker service that forwards incoming requests to the right inner service. Indeed, aside from the **ParkingOccupationService**, other functionalities exploited by the “myCicero” application

are provided by different services. Actually, the broker service already exists, and its routing is just to be extended to handle the new requests.

Besides the above components, it is also present an application that encapsulates all the code used to execute the data analysis stage explained in Chapter 2. Such part has been realized using a Python notebook.

Actually, considering that the integration of the system with "myCicero" application is still in progress, the **ParkingOccupationService** has not been implemented yet, and so no detailed description is going to be executed in next sections. Moreover, that service and the **StorageService** may be collapsed into one depending on the adopted solution. For the time being, they are separately considered for a better separation of concerns (prevent incoming requests from contacting the **StorageService** directly).

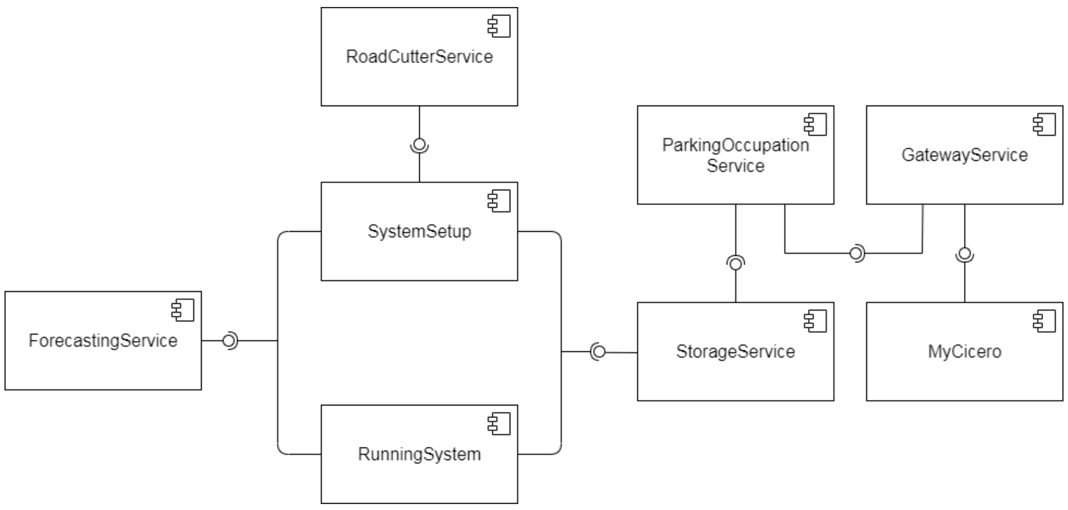


Figure 4.1: *UML component diagram of the high-level architecture of the system.*

4.2 SystemSetup

This component of the system is responsible for all setup tasks, including:

- Definition of the geometric model
- Setup of the regions: split of transactions, calculation of capacities
- Training of the forecasting model
- Setup of the information saved in the **StorageService**

No particular diagram representing the architecture of the system is reported since the application can be described as a linear process with no particular underlying design except for few utility classes used to gather common aspects (e.g., *Region* class to group all regions related concepts).

This setup process has been developed using Python since many useful libraries and functionalities to deal with dataframes (e.g., `pandas`) and geometric aspects (e.g., `scipy.spatial`, `shapely`) are provided. Moreover, since the data analysis part has been executed using Python too, further functionalities can easily be reused (e.g., creation of time series compliant to the model described in Section 2.2). In the end, the `requests` module has been used to easily handle the HTTP client.

4.2.1 Geometric model

The territory is split up using the polygon-oriented model described in Section 3.2.1.

Firstly, we need to create the set of starting points, and all the three alternatives are implemented. The class *VoronoiModel*, containing few useful utility functions and metadata, exposes the respective methods.

Moreover, when talking about the regular grid solution, we consider four different granularity levels (L1, L2, L3, L4).
The L1 grid is initially set up so identical cells of size 792m x 792m[1] are created and, then, we proceed by splitting each cell up in four sub-cells (L2). This way, we double the number of cells along each axis, and the length of the cells' sides is halved (396m x 396 m). Similarly, for L3 and L4 grids.
Within the *VoronoiModel* class, the *regular_grid_shape* dictionary stores the metadata of each grid level (i.e., number of cells along each axis).

It follows a snippet of code showing the three available alternatives to generate the set of starting points (Listing 4.1).

```
points = VoronoiModel.regular_grid(*VoronoiModel.regular_grid_shapes["L1"])
points = VoronoiModel.random_points(500)
points = VoronoiModel.parking_meters_locations(pm_registry) #pm_registry:
    dataframe storing locations of parking meters
```

Listing 4.1: *The three different methods to set the starting points up.*

Once the starting points are defined, it is required a feature that allows to split the domain area up by creating a Voronoi tessellation using the Euclidean

[1] A value of 0.1 degrees on the longitude axis is chosen for convenience, corresponding to 792 meters. Then, the same length is used for latitude too since the cells must be squared.

distance as metric[2]. Such an algorithm is not created by scratch, but the `scipy.spatial` package is exploited. Indeed, the class *scipy.spatial.Voronoi* will automatically create the expected result just by providing the set of starting points arranged as a tensor of shape (n_points, n_dimensions) containing the coordinates of each point (Listing 4.2).

```
from scipy.spatial import Voronoi, voronoi_plot_2d
voronoi = Voronoi(points)
fig = voronoi_plot_2d(voronoi, show_vertices = False, show_points = False)
```

Listing 4.2: *Creation of the Voronoi tessellation and plot of the result.*

Figure 4.2 shows the basic solution merely applying the algorithm to create the tessellation.

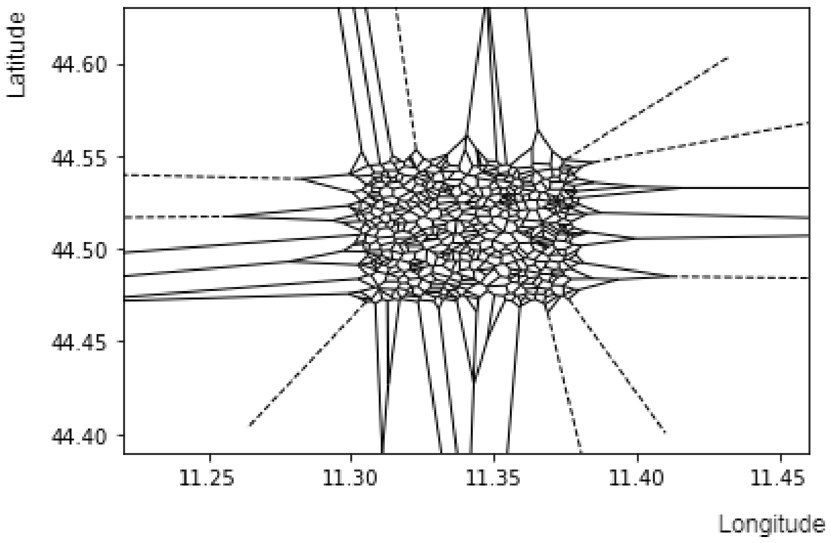


Figure 4.2: *A Voronoi tessellation created by using 500 random points uniformly distributed among the domain.*

We need to extract the edges of each region. Inter alia, we need to point out the vertices so that each region can be expressed as a closed polygon using the *shapely.Polygon* class. As we can notice, it emerges an issue since few regions are unbounded (the ones containing ridge vertices), and no closed polygon can be associated to them. We can avoid such a problem quite easily by adding few additional ad hoc points (control points) so that only these new points will be the ones associated to an unbounded region (control points will be ignored on next steps, and so it is not a problem if they have an infinite region).

[2]The usage of the Haversine distance would be an overkill since each region is quite small, and so the introduced error is very low.

The minimum required number of points to create a subset of the 2-D space is three, but we use four to keep it simpler by creating a rhombus around the real domain. The "real" starting points now handle a limited subset of the whole 2-D space and the remaining part is split upon the control points (Figure 4.3).

Note: Attention must be paid to grant a correct positioning to the control points so that it is impossible for any of them to have part of the domain inside its own region. The proof of the minimum required distance from the domain is trivial, and so it is omitted for conciseness.

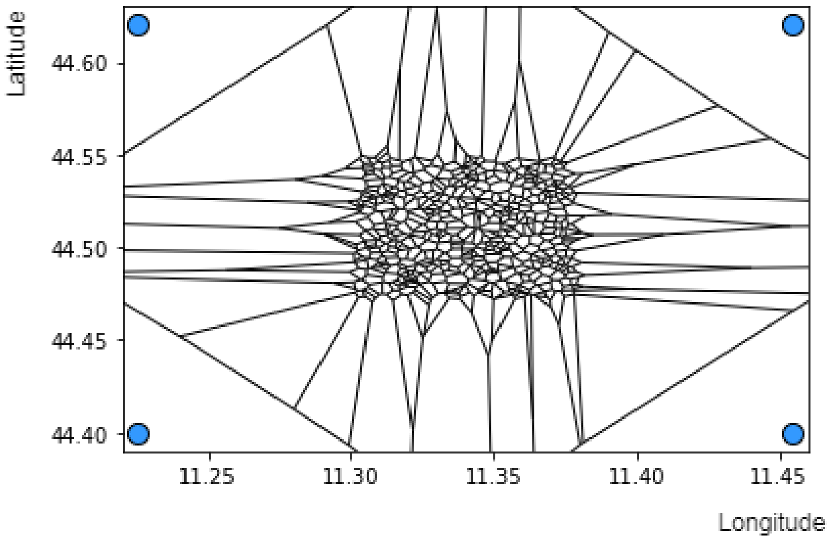


Figure 4.3: *Bounded Voronoi tessellation using 500 random points and 4 control points (blue markers). The "real" points now handle only the middle finite rhomboid area.*

In the end, once the unbounded regions have been dropped, we finalize each region with the assignment of an id and the clipping of portions exceeding from the domain by intersecting the region with the domain bounds. Code stub in Listing 4.3 reports the execution of such operations.

```
#Create a Polygon for each region of the Voronoi tessellation .
#For each region:
#1. Get index of its vertices
#2. Drop empty regions (the first returned region is always empty, len == 0) or
    with ridge vertices (only ctrl points, index == −1) so that only finite
    regions are kept
#3. Get coords of vertices from indices (map function)
#4. Create Polygon from list of vertices coords
#5. Intersect region with bounds so that only the part within the domain is kept
```

```
#6. Associate an id to each region (zip)
from shapely.geometry import Polygon
region_boundaries = [_ for _ in zip(
                                    list(range(len(voronoi.regions) − len(control_points) −
                                        1)), # Drop 4 ridge regions + 1 empty one by default
                                    [Polygon(list(map(lambda index: voronoi.vertices[index],
                                         region_vertices_indices ))).intersection(domain_bounds)
                                     for  region_vertices_indices
                                     in voronoi.regions
                                     if not −1 in  region_vertices_indices  # Drop ridge
                                         regions associated to control points
                                       and len( region_vertices_indices ) > 0
                                    ]
                                  )]
```

Listing 4.3: *Final operations on the regions: clip points out of the domain, assign a progressive id.*

Figure 4.4 shows the clipped regions. As we can see, the territory is quite uniformly covered by regions with almost/exactly the same area when using the random points/regular grid solution, while we notice a different distribution with edge regions having a huge area respect to others in the middle if parking meters locations are employed (parking meters are not uniformly distributed over the domain since Bologna is not a perfect rectangle). This is not a problem during forecasting tasks, but it must be taken into consideration if any analysis based upon regions' area is going to be executed in the future.

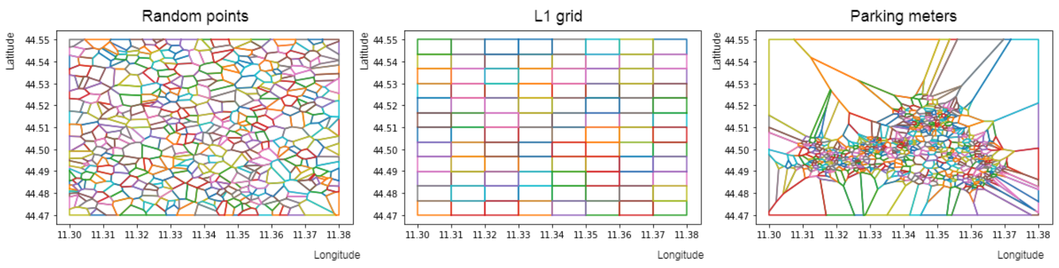


Figure 4.4: *Voronoi tessellation with clipped regions using 500 random points (left), regular grid (center) or parking meters locations (right) as set of starting points.*

In the end, it is also present a feature that permits to load an already existing setup of regions. As well as for convenience (avoid to build the same model each time), this is useful for reproducibility if a same specific setup of regions is to be used once the runtime is reset (e.g., same boundaries when

using random starting points). Obviously, it is present the dual feature that allows to export a given setup of regions too.
Regions' boundary is stored in a Json file structured as in Listing 4.4.

```
{
  regions: [
    {
      id: number
      boundary: [
        {
          longitude: number
          latitude: number
        }
      ]
    }
  ]
}
```

Listing 4.4: *Format of the Json file containing the regions' boundary.*

Such features are simply realized be exploiting the `json` package.

4.2.2 Further setup tasks

Once the boundaries have been defined and data have been preprocessed, the first step is to split the transactions between the regions. Seen the amount of data and the cardinality of the regions for the time being, a trivial algorithm can be implemented and no particular solution is required (e.g, particular arrangement or shuffling of data).
Given a transaction, a *shapely.Point* object is created so that it can be discovered the region containing that transaction using the *contains* method of the *shapely.Polygon* class.
Since any transaction is going to be associated to only one region, an early stopping strategy might be applied when the first matching region is found.
It follows a stub of code showing the core of such task in Listing 4.5.

```python
for arrival in df.values: # (lon, lat, timestamp, length)
  location = Point(arrival[0], arrival[1])
  admissible_region = next((r for r in regions if
      r.boundary.contains(location)), None)
  if (admissible_region is not None):
    admissible_region.add_arrival(data_source_name, arrival[2], arrival[3])
```

Listing 4.5: *Split of the transactions between the regions.*

Then, the ground truth must be modeled, and so a time series *year by quarters* is created for each data source of each region.
Code stub in Listing 4.6 reports the core function containing the workflow used to generate a time series from a given dataset.
The *ArrivalTimeIndex* class is just a container that stores a bunch of time information at different abstraction levels: day in year [1;366], month in year [1;12], etc.
At this point, the time series is created to adhere the model explained in Section 2.2 using an utility function (*calculate_counts*).

```python
def create_TS_year_by_quarters(
        df: pd.DataFrame,
        day_parsing_strategy: Callable[[str], list],
        time_parsing_strategy: Callable[[str], list],
        timestamp_split_strategy: Callable[[str], list],
        quarter_round_strategy: Callable[[int], int],
        time_column: str = "timestamp",
        length_column: str = "length"
        ) -> list:
    period_type = YEAR_BY_QUARTERS_PERIOD
    arrivals = extract_arrival_info(df, day_parsing_strategy,
        time_parsing_strategy, timestamp_split_strategy, time_column,
        length_column)
    for arrival in arrivals:
        arrival.indices.arrival_quarter_in_day =
            calculate_arrival_quarter_index(arrival, quarter_round_strategy)
        arrival.indices.day_in_year = calculate_day_in_year_index(arrival)
        arrival.indices.quarters_in_day = calculate_stop_quarter_index(
            arrival.indices.day_in_year * N_QUARTERS_IN_DAY +
                arrival.indices.arrival_quarter_in_day,
            math.ceil(arrival.length / N_MINUTES_IN_QUARTER),
            period_type.length
        )
    return calculate_counts([quarter for arrival in arrivals for quarter in
        arrival.indices.quarters_in_day], period_type.length)
```

Listing 4.6: *Function used to create a time series from a given dataframe and using the given strategies. The documentation of the function (that should be placed between the declaration and the body) is omitted for conciseness.*

Once all the time series have been defined, the capacity is calculated for each data source of each region as specified in Section 3.1. Moreover, the service hosting the forecasting engine is contacted so that the training phase is executed.

In the end, the regions' data is saved using the **StorageService** so that it can eventually be exploited by the **Running system** and the **MyCicero** application both.

4.3 RunningSystem

This is the core component of the system and it manages the execution of real-time predictions.
It has been developed in Scala using an actor model, and the `akka` library has been used to handle all actor related aspects. Moreover, the `sprayJson` library has been used to handle serialization/deserialization of objects.
Regarding tests, the `scalatest` and `akka.test` libraries have been used to perform unit tests on passive components and actors, respectively. *WordSpec* and *FlatSpec* are the test styles adopted as they seemed, among many, the most expressive ones.

Figure 4.5 reports an high-level state diagram of the application. After the starting bootstrap state where all setup tasks are executed, the application cycles between the "IDLE" and "ExecutingForecast" states.

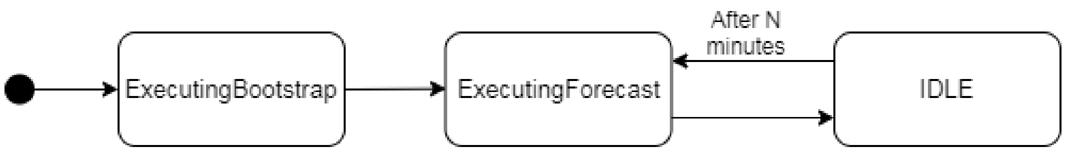


Figure 4.5: *High-level UML state diagram of the system.*

The application architecture is quite simple and it includes an overall of four different actors, namely:

- **CoreAgent**: core of the application that orchestrates other components.
- **ForecastingAgent**: it handles all forecasting aspects (contact the *NN-ForecastingService* to get updated forecasts, estimate the OPVs, merge partial results, etc.).
- **StorageManager**: it intermediates all the communications between the application and the *StorageService*.
- **HttpClient**: utility actor that handles HTTP logic (requests, responses, data format, etc.). For any information about the http requests, the REST interface of each service is described in the proper section.

Figure 4.6 reports the bootstrap operations executed to retrieve the information about each region from the storage service.

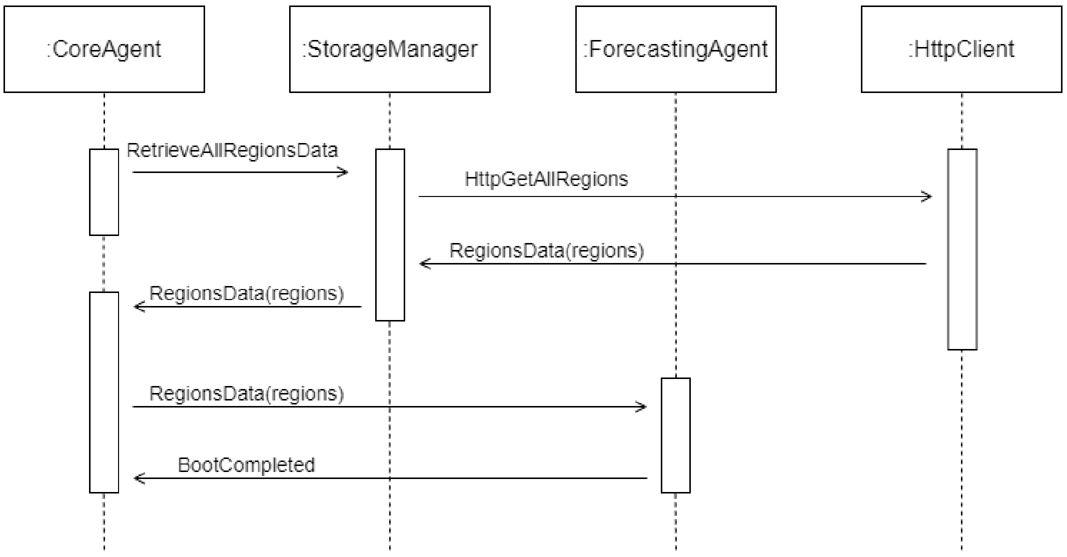


Figure 4.6: *UML sequence diagram for the bootstrap operations illustrating interactions between actors.*

Once the system completed the setup, it starts executing a period forecasting. Figure 4.7 reports the adopted workflow.
The *CoreAgent* periodically sends a message (*ExecuteForecast*) to the *ForecastingAgent* to execute a new forecast. The periodic behavior is simply created using a timer that sends a message to the *CoreAgent* at regular intervals. The *ForecastingAgent* asks for updated predictions to the *ForecastingService* and updates the forecasted value about number of occupied parking lots for each data source of each region. As soon as the new predicted values are available, the agent calculates the new occupation status for each region. New status are communicated to the *RegionManager* that, in turn, updates the **StorageService** coherently.

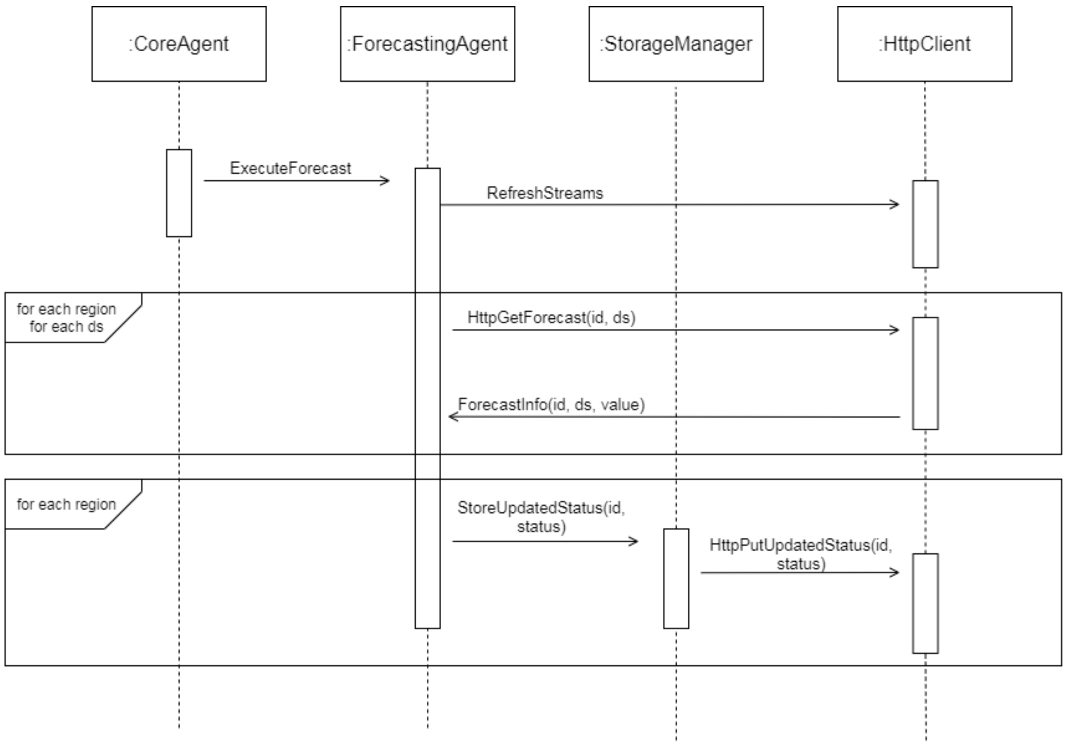


Figure 4.7: *UML sequence diagram for the forecasting operations illustrating the interactions between actors.*

Particular attention must be paid to the outgoing http requests. Indeed, the simple pattern used to send http requests will not work properly when dealing with bursts of requests (sometimes we have a request for each data source of each region, and so too much simultaneous open connections to the same endpoint).
The trivial solution would be the change of the maximum allowed number of connections in the `akka` configuration file. However, such solution statically depends on the number of regions and data sources both, and a high number of parallel connections may slow the system down excessively.
Hence, a different approach has been applied by using the reactive streams, functionality provided by the `akka.stream` library.
A dynamic stream of *HttpRequest* is created using the *Source.queue* method. The stream can use a same http connection for all outgoing requests if configured with the proper *Flow* object, and an handler can be defined to manage the responses (*HttpResponse*). In the end, the stream provides also backpressure functionalities so that the processing rate of requests is correctly handled. Any request to be sent is offered to the stream.
Notice that a stream is "closed" if no information is offered for more than

a certain amount of time (5 seconds by default), and so data can not flow through it any more. To solve such a problem, the *ForecastingAgent* sends a *RefreshAllStreams* message to the *HttpClient* before the beginning of the forecast so that all the streams are reactivated.
Code stub in Listing 4.7 shows an example of creation and configuration of a stream.

```
private val forecastSourceDeclaration = Source.queue[HttpRequest](bufferSize,
    OverflowStrategy.backpressure)
private val forecastFlow = http.outgoingConnection(Routes.Forecast.address,
    Routes.Forecast.port)
private val (forecastSourceMaterialized, forecastSource) =
    forecastSourceDeclaration.preMaterialize()

forecastSource via forecastFlow runForeach { res =>
  Unmarshal(res).to[ForecastResponse] onComplete {
    case Success(value) =>
      forecastingAgent ! ForecastInfo(value)
      res.discardEntityBytes()
    case Failure(exception) => forecastingAgent ! FailResponse(exception)
  }
}
```

Listing 4.7: *Creation of a stream (Source) and definition of its flow and handler. The stream will handle all requests sent to the forecasting service. Once created, new requests can be submitted using the "forecastingSourceMaterialized.offer" method.*

4.4 ForecastingService

This component contains the forecasting engine used to predict the number of occupied parking lots, and its REST interface is reported in Table 4.1.

Http Method	**Path**	**Payload**	**Description**
POST	/regions	{id:int, ds:string, values:array}	Create and train the model for the given region and data source.
GET	/regions/:id/:ds /forecast	None	Get an updated forecast for the time being.

Table 4.1: *REST interface of the forecasting service.*

Python has been used as programming language, and the service implements the forecasting model exposed in Section 3.1.2 using the library `fbprophet`.

The `fbprophet` library is compliant to other machine learning well-known API (e.g., *keras*, *scikit-learn*), exposing the *fit* and *predict* main functions.
During the training stage and differently from other machine learning libraries where data is usually shaped using tensor-like structures, the input data must be arranged on a *pandas.Dataframe* object containing two columns "ds" and "y" representing datestamp[3] and time series values, respectively.
Proper APIs make it possible to setup all model's aspects such as seasonalities, holidays, etc. In our case, daily and monthly seasonalities have been specified both, and local holidays such as the patron day have been added to the Italian regular festivities.
In the end, a log transform operation on time series values is executed before the training of the model. Obviously, predicted values are then subjected to the reverse operation (exp).
It follows a stub of code in Listing 4.8 summarizing the training of a model and the execution of the forecasting.

```
# Creation of the model
model = Prophet(changepoint_prior_scale=0.9, holidays=patron) # tau = 0.9
model.add_seasonality(name='daily',period=1, fourier_order=5)
model.add_seasonality(name='monthly',period=30.5, fourier_order=5)
model.add_country_holidays(country_name='IT')

# Training
model.fit(df)

# Forecasting
```

[3]All timestamp formats accepted by `pandas` can be provided.

```
predicted_period = model.make_future_dataframe(periods =
    N_QUARTERS_IN_DAY, freq = "15min")
predictions = model.predict(predicted_period)
```

Listing 4.8: *Example of training and forecasting upon a Prophet model. The observed data is contained in the "df" variable.*

4.5 StorageService

Data about regions must be stored persistently, and the **StorageService** fulfils this purpose. Indeed, such service simply creates a layer upon a database storing the required information.
The service has been realized as a web-service using the *MEAN stack*, and its REST interface is reported in Table 4.2.

Http Method	**Path**	**Payload**	**Description**
GET	/regions	None	Retrieve info about all regions.
GET	/regions/:id	None	Retrieve info about the specified region.
POST	/regions	Json file of the region (Listing 4.9)	Create a new region.
PUT	/regions/:id/status	{ status: string }	Update the status of the given region.
GET	/regions /deleteAllRegions	None	Utility service: remove all the data in the database about regions.

Table 4.2: *REST interface of the service.*

The database contains just one collection "regions" whose documents are shaped as in Listing 4.9.

```
{
  id: number,
  boundary: [{
    latitude: number,
    longitude: number
  }],
  data_sources: [{
   name: string,
   capacity: number
  }],
  roads: string,
  status: string
}
```

Listing 4.9: *Structure of a region using JSON format.*

The adopted architecture of the service is a classical one and it is organized into three main packages, namely:

- **routes**: it administrates routing by declaring handlers for all allowed requests.
- **controllers**: it contains the application logic specifying callbacks for incoming requests.
- **models**: it defines the format of data.

The **app.js** file is the entry-point of the application and it is responsible for all preliminary configuration tasks (connection to DB, enabling of CORS requests, running of the server).

4.6 RoadCutterService

Table 4.3 reports the REST interface of the service: there is only one utility service that permits to extract all the roads contained within a given area.
The boundary of a region, submitted as a set of (latitude, longitude) pairs, is defined in the query string by the parameters "lat" and "lon", and the desired information is returned using the WKT format (Well-Known Text).
It follows an example request using three points {(0,0), (1,1), (2,3)}:

http://serverAddress:port/extractRoads?lat=0&lon=0&lat=1&lon=1&lat=2&lon=3

Http Method	**Path**	**Payload**	**Description**
GET	/extractRoads	None	Retrieve all roads segments contained within the given area in WKT format.

Table 4.3: *REST interface of the RoadCutter service.*

The implementation of the logic itself has not been executed personally, and so it is not described in detail.

Chapter 5

Performance evaluation

Independently from the adopted solution, there is an error between the forecasted occupation of parking lots and the real one. More precisely, we can distinguish three different components illustrated in Figure 5.1, that is:

- **Real-world gap**: the difference between the real occupation and the one considered as ground truth (Section 5.2).
- **Generalization error**: the difference between the occupation modeled using total and partial data (Section 5.3).
- **Forecasting error**: The classical error introduced by the forecasting model (Section 5.4).

Each type of error can be limited, but none of them can be completely eliminated.

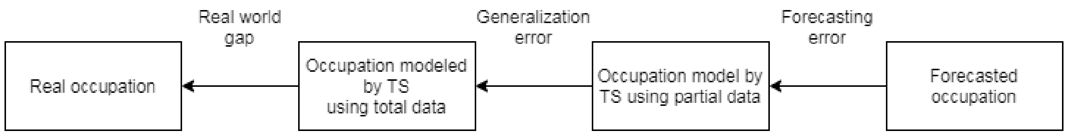


Figure 5.1: *Each type of error and the stage where it is introduced.*

Remind: The generalization error must be considered as a component because only "myCicero" data source is mandatory. Taken in consideration that all data sources are available in the Bologna case study, such a fact is not ensured to be true anymore in other cities the model will applied to.

5.1 Experimental setup

The system is tested using the following setup:

- **Data sources**: For the time being, verifiers data is not used because of the motivations already explain in Chapter 2 (that is, a long period of missing values still to be filled and incoherences in the trend to be solved). Hence, just “myCicero” and parking meters data are exploited.
- **Period:** all the available data is used (01-01-2019/31-12-2019).
- **Geometric model:** the L2 grid solution is the one chosen at the moment.
- **Prophet**
  - *Trend*: automatic selection of change-points, $\tau = 0.9$.
  - *Seasonalities*: we select a Fourier order of 10, 5 and 5 for yearly, monthly and daily seasonality respectively.
  - *Holidays*: No particular setup besides the basic configuration.

5.2 Real-world gap

Differently from other classical domains where the ground truth can be modeled with a high confidence despite little approximations (sales, stocks, etc.), the parking occupation of real scenarios can hardly be represented if proper data is not provided since it must be rebuild from scratch. Indeed, the usage of information registering the real-time occupation of each parking lot is the only way to create trustworthy values for occupation and capacities both. Actually, similar real-time sensors already exist (e.g., positioned under the asphalt), but their spread is still restricted to some limited areas.

In our case, we do not have the aforementioned type of information, and so the comparison of the ground truth is going to be executed against the the data registered by verifiers for the number of occupied parking lots and handmade calculated values for the capacities. Obviously, only few example regions will be compared since the whole process cannot be automatized. Though, quantitative analysis in this direction have not been executed yet.

5.3 Generalization error

As emerged from previous analysis, the forecast can be quite inaccurate if only partial data is employed since different data sources can expose totally different behaviors even in a same region.
For the time being, we only consider the daily trend since it is the most informative one, but similar analysis can be executed at different granularity levels too.

As we can notice in Figure 5.2, the ratio between parking meters and "myCicero" data, which is about 10:1 considering the total territory, highly varies from region to region. Moreover, it assumes very different values without any marked detectable pattern, except for a specific restricted area of the old town where find a widespread low ratio (green spot on the map, area included in [11.33, 11.35] and [44.49, 44.50] for longitude and latitude, respectively).

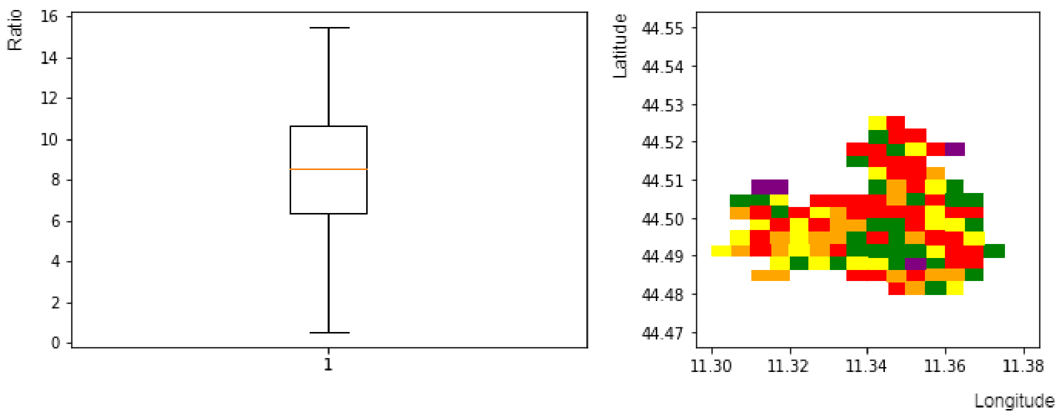


Figure 5.2: *Ratio of parking meters and "myCicero" data on the different regions. Boxplot outliers having ratio greater than sixteen are not reported so that a nice representation can be obtained. The heatmap gives a qualitative information about the distribution of the ratio values on the territory: green $->$ low ratio, yellow $->$ average ratio, red $->$ high ratio.*

Figure 5.3 shows few example regions exposing opposite situations (different ratio, different behavior).

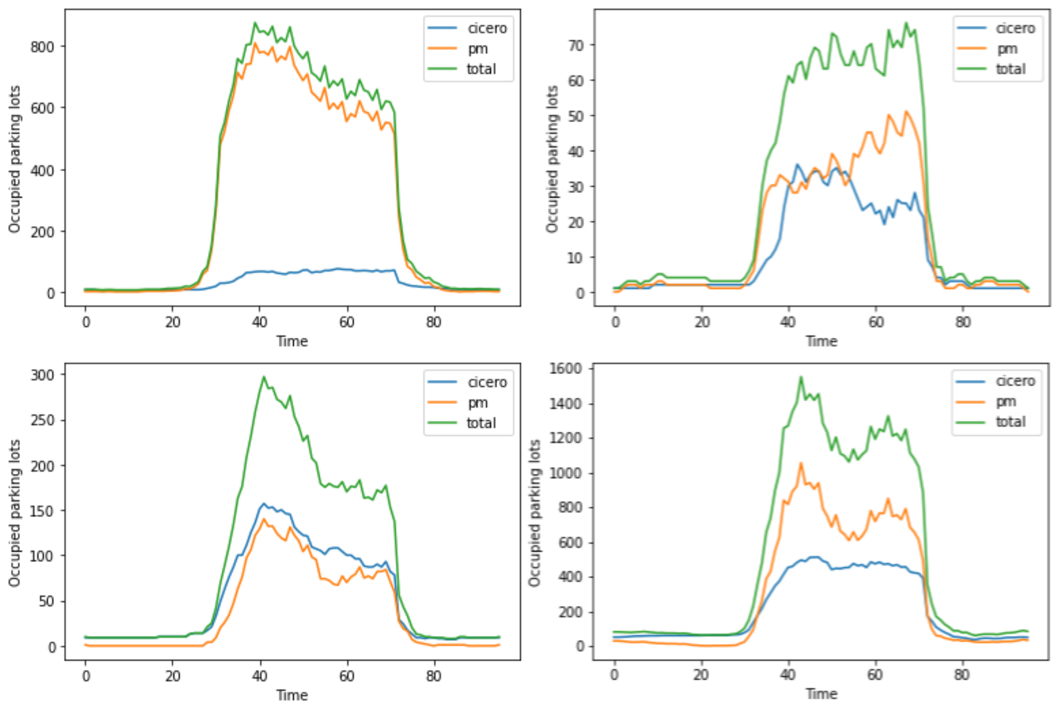


Figure 5.3: *Comparison of "myCicero" and parking meters behaviors on some example regions ad hoc chosen to show different possible casuistries: compliant and opposite behaviors, high and low ratio, cases where "myCicero" data even is higher than the parking meters one.*

The exploiting of the parking meters solely is going to work quite well in regions where a high ratio is found because "myCicero" trends become irrelevant when considering the total ones, and so they represent a negligible factor during the OPV calculation.
On the contrary and differently from parking meters, the exploiting of the "myCicero" data solely is not feasible since a quite erroneous prediction is going to be made in most of the regions.
Thus, we can conclude that no rule can be inferred to model the total trend from a partial data, and none of the data sources can be dropped if an accurate forecast is required in all regions.

5.4 Forecasting error

The *forecasting error* is the classic error made by a predictor when comparing the forecasted values with the ground truth. We consider the following two metrics to evaluate the model: *MAE*, *r2score*.

MAE

Firstly, we merely measure the error made by the Prophet model when estimating the number of occupied parking lots. Figure 5.4 reports raw information about the mean absolute error in all regions for “myCicero” and parking meters both. As we can notice, though a different order of magnitude in the values, the errors share a very similar distribution, and the higher errors are mostly associated to regions within a same restricted critic area (approximatively, [11.33, 11.34] for longitude and [44.50, 44.51] for latitude).

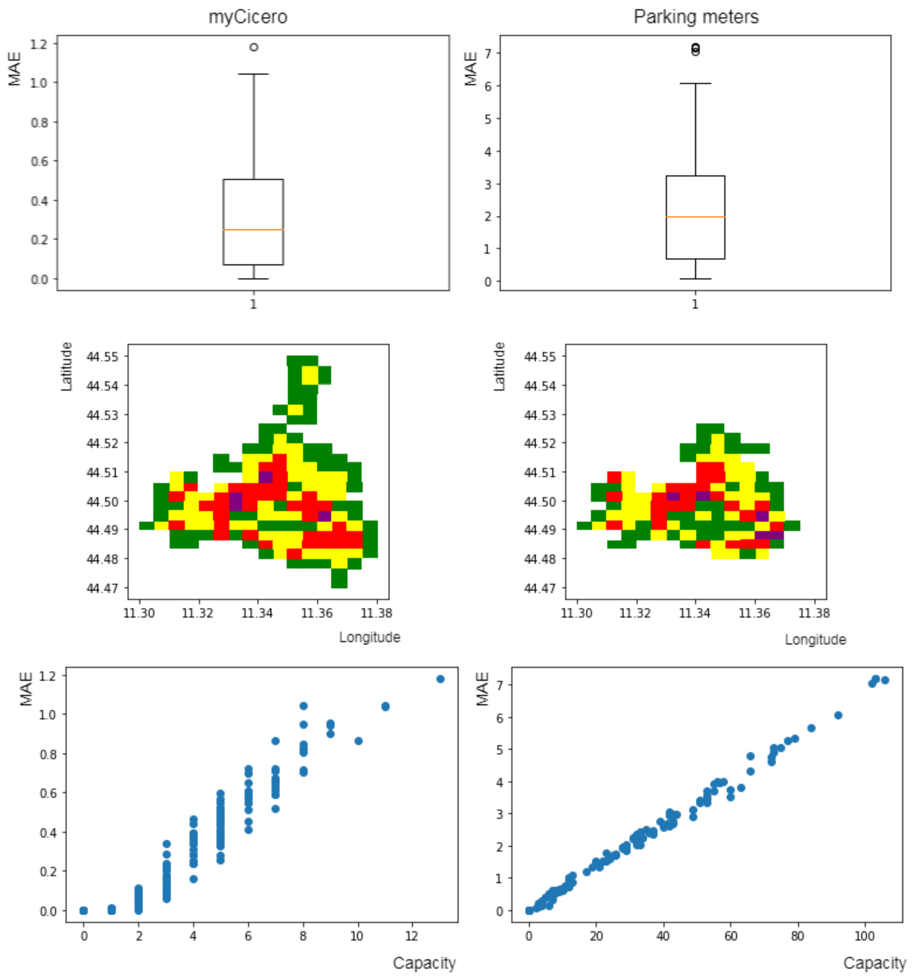


Figure 5.4: *MAE information about “myCicero” (left) and parking meters (right). Heatmaps give a qualitative information about the errors distribution above the territory, and their parameters are not reported for conciseness.*

Obviously, the adopted measure is tricky, and we can find a remarkable positive correlation between the capacity of a region and the introduced error for “myCicero” and parking meters both. Hence, we refine the evaluation by considering the OPVs so that we can compare regions independently by their capacities, and Figure 5.5 reports the respective information.
The errors now assume values in the same domain [0, 1]. Hence, they can be compared, and novel interesting information emerges.

Though a same median (about 7% of the capacity), “myCicero” errors expose a great variance, while parking meters one assume a very little set of values. This implies a great robustness for parking meters’s forecasting model, independently from region specific traits.

As a matter of fact, notice how the previous critic area does not create any real particular problem when considering parking meters, and its performances are absolutely average. Just a small cluster of red regions can be found in a small area of the old town (outliers with high errors) while, on the contrary, outliers representing low errors are distributed on suburban areas, and can be all associated to region with very low capacities.
On the contrary, we notice that the same critic area initially identified is still present for “myCicero”, and so the model truly does not perform well in such regions.

Besides, a further surprising finding can be observed when considering the correlation between capacity and MAE. Indeed, the behavior of parking meters resembles the dynamics of a complex system, exposing a phase transition at a precise critical value of the capacity (about 10-15): errors are sparse with no particular logic or pattern for regions having low capacities, while the system reaches a steady state and the MAE suddenly stabilizes as the threshold is overcome.

On the other hand, such observation can not be tested on “myCicero” data since the capacities all assume low values (beneath 15).
Anyhow, lower errors for regions having the little capacities were expected, and they can be addressed to an excessive bias, despite of a low variance[1]. To be exact, the issue does not reside on the forecasting model itself, but instead on the TS that can not frame the behavior of the region. Indeed, TS associated to such regions contain just sparse peak values in the extreme cases where the capacity is close to zero, and the forecasting model, that tends to be very simple (like a constant function), correctly foresees the value that is “0” in most of the cases. For increasing capacities until fifteen, TS progressively become more and more “complex”, and so the errors start to increase. This justifies the positive correlation between capacity and MAE for “myCicero”

[1]Bias and variance terms are referred to the bias-variance decomposition model.

(Spearman correlation = 0.98), but it is not clear yet why such correlation is not also present on parking meters regions having low values (Spearman correlation = 0.25).
Then, when a TS is build using enough data and so the capacity estimated on it increases, the error gets quite stable independently from the employed amount since the behavior gets "complete", exposing well-defined seasonalities and trends.

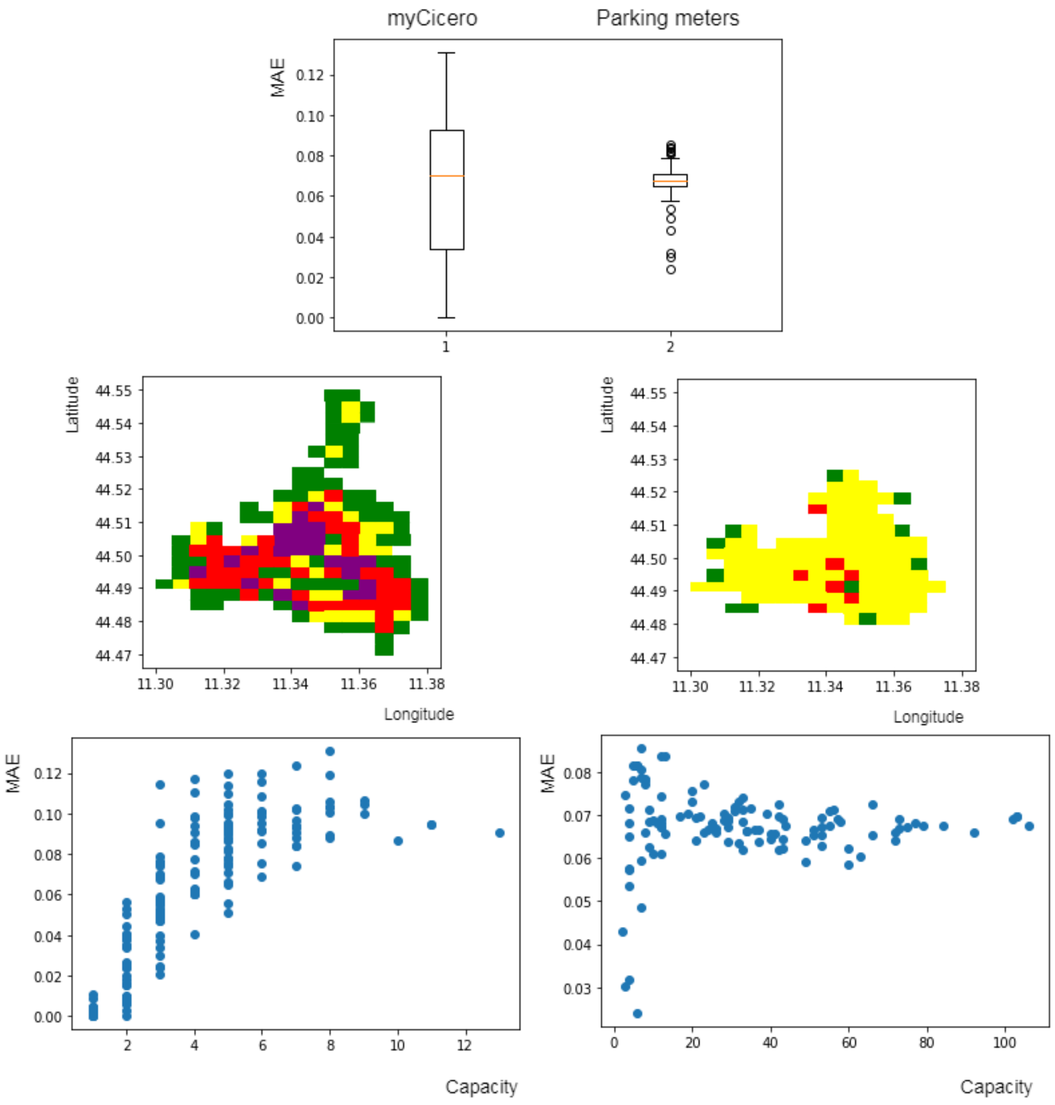


Figure 5.5: *MAE information normalized by the capacities about "myCicero" (left) and parking meters (right). Heatmaps give a qualitative information about the errors distribution above the territory, but their parameters are not reported for conciseness.*

r2score

Besides the corroboration of the above considerations, the r2score results also expose novel findings.

Firstly, we discover a positive correlation between the capacity and the r2score in Figure 5.6. It may seem contradictory to previous results emerged from MAE analysis where little errors were detected for the lowest capacities but, actually, it is not and we can understand why it happens by properly looking to r2score definition formula where the numerator is zero in most of the cases. Anyway, we can conclude that the model better explains regions containing the more data, and it is compliant with the previous findings since these are the regions where the behavior is not particularly influenced from fluctuations and can be predicted easily.

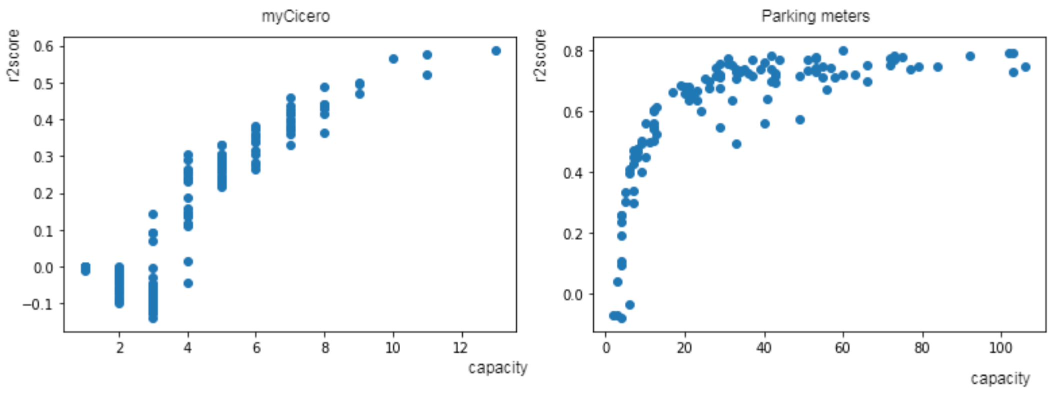


Figure 5.6: *Correlation between the capacity and the r2score for "myCicero" (left) and parking meters (right).*

Furthermore, we notice a general better evaluation for parking meters values despite "myCicero" ones (Figure 5.7). Coherently with previous considerations, such a difference is partially smoothed out when considering only regions having a low capacity for parking meters, but there is still a significant difference between the two medians.

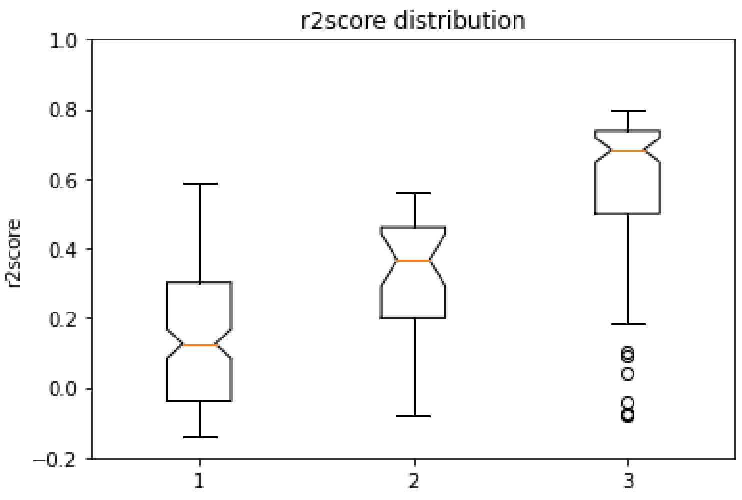


Figure 5.7: *Distribution of the r2score for "myCicero" (1), parking meters having capacity lower than fifteen (2), all parking meters (3).*

5.5 Behavior goodness

The goal is to check how well the adopted geometric model may catch situated behaviors, and so somehow measure its expressiveness. Indeed, as emerged from previous analysis in Chapter 2, the correct split of the territory is a key factor when framing behaviors dictated by local peculiarities of the territory, and it depends upon a set of parameters the regions can be defined on: shape, size, correct positioning. Obviously, there are portions of the territory which expose a high occupation all day long, and so an almost constant daily trend will be detected independently from the applied division. On the other hand, some trends of that type are instead given by a mix of two or more different behaviors that are caught within the same region because of an erroneous split of the territory (see Section 2.5).
The issue is that even a system that exactly predicts the real occupation is quite useless from a business point of view if a same occupation is provided all day long for almost all the regions because of a too raw split of the territory, and so the adoption of a good geometric model assumes an even more important role than the achievement of a great accuracy on the prediction model itself. Indeed, the choice of a particular set of regions will detect behaviors consequently, and the evaluation of its optimality is the matter of the following analysis. Contrariwise, the evaluation of the forecasting model on the identified behaviors is the subject of the previous analysis (real-world gap to measure compliance to real scenarios, forecasting error, etc.).

Figure 5.8 reports a clarifier example that may help to distinguish the above two factors. An erroneous split may consider only the region C, and so the situated behaviors would be lost. At the same time, the forecasting model might behave very well on region C, exposing accurate predictions and high compliance to real scenarios.

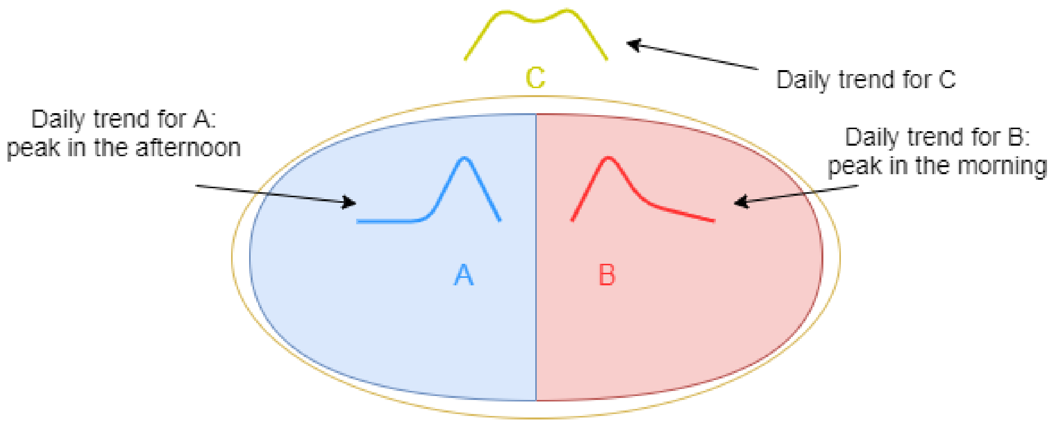


Figure 5.8: *Graphical example of behaviors mixing: region A daily trend exposes a peak during afternoon hours, region B daily trend exposes a peak during morning hours, region C will merge the partial behaviors and create an almost flat daily trend.*

We define a measure of performances (*goodness*) to estimate how well a region behaves so that we can compare the different areas in a quantitative way. The analyzed behavior is the trend of stops along the day, and so the *day by quarters* time series will be the exploited one.

Figure 5.9 reports two example time series: the one on the left is considered "bad" since its trend is almost constant, while the one on the right is considered "good" since there are sharp peaks and great variance in the number of occupied parking lots. Intuitively, we can understand that in the first case no useful forecast can be made (always high occupation), while in the latter one the forecasted values can range over very different values (and worthwhile feedbacks can be provided during the different times of the day). Obviously, considering that the occupation is always close to zero during night hours, such arguments refer to daylight hours solely.

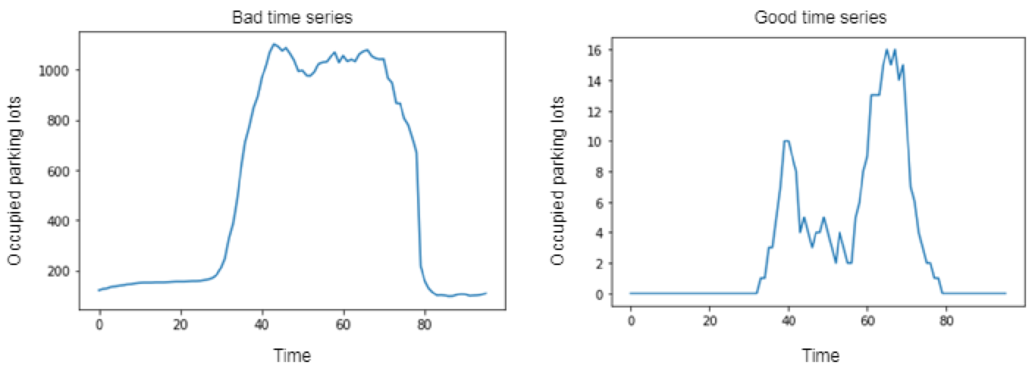


Figure 5.9: *Example of what is considered to be "bad"/"good" time series.*

Firstly we execute two simple preprocessing operations, that is:

- **x-axis:** we keep only the daylight hours values [32;72].
- **y-axis:** we normalize the TS values in range [0;1] so that different time series can be compared.

The idea is to consider the area under the curve (i.e., the integral of the function). Such value and the goodness are bound by an inverse proportionality since the time series tends to flatten as its area increases. The minimum value is 0, while the maximum theoretical value is 40 (period length * value_range = 40 * (1-0) = 40).
In the end, the calculated values are normalized in [0;1] for convenience, and then complement is taken ($1 - normalized\_area$) since the goodness increases as the area decreases.
Such a construction works since we know that each TS will take value "1" at least in one point (so flat series with low values are not possible). Moreover, assuming we have enough data to correctly[2] model a TS and considering that TS model stops for each quarter, it is not possible that the maximum is reached in just one peak point. Indeed, as confirmed by empiric corroboration, we have that the TS trend can vary quite quickly but always in a "differentiable" way to an high-level description (in other words, cannot have values [1,1,2,100,3,2] where there is an isolated peak, but it will presumably be something like [1,10,60,100,80,30,2]). Figure 5.10 shows a clarifier example.

[2]The term "correctly" refers to the modeling of well-defined seasonalities, trends and holidays. As observed in previous sections, few regions have too little data to frame a proper behavior.

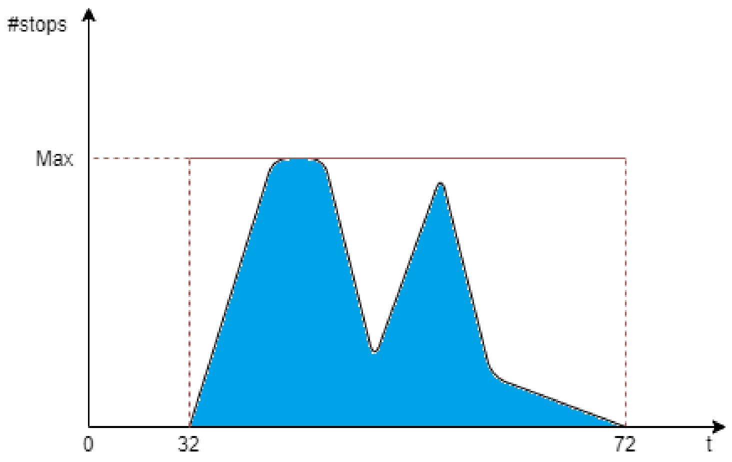


Figure 5.10: *The TS is enclosed in [32; 72] on x-axis and [0; Max] on y-axis. The TS area (blue) is always lower than the one of the worst casuistry (red rectangle). Notice that that is the worst case since it is the only way to create a constant series that reaches the Max value at least once.*

Figure 5.11 reports the goodness for each region of the L2 grid. As we can notice, low goodness values are mainly gathered in the old town and downtown areas, and parking meters seem to hold better behaviors in general.

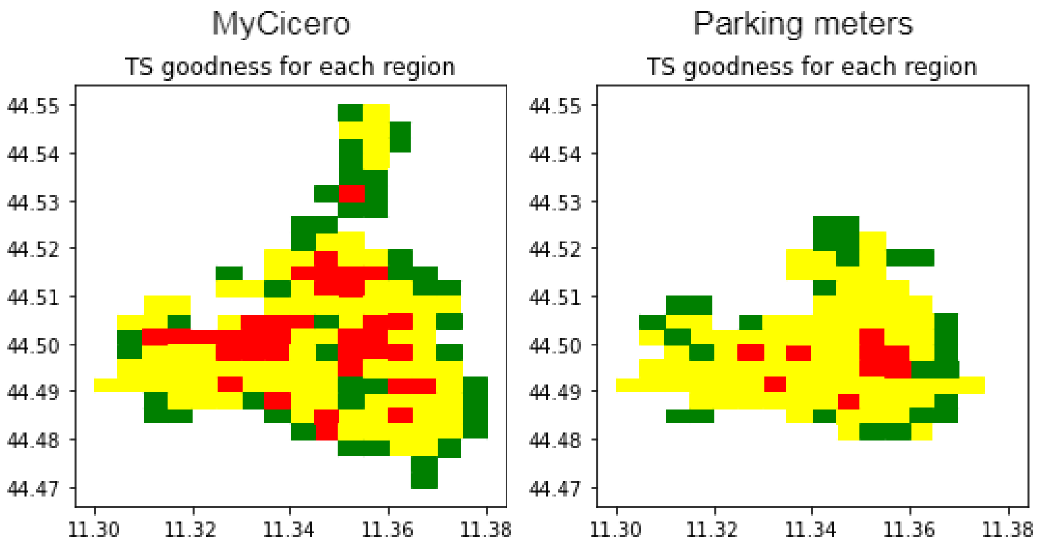


Figure 5.11: *Goodness for each cell of the grid G2. Red -> [0, 0.25], Yellow -> (0.25, 0.4], Red -> (0.40, 1], White -> Areas with no data.*

We compare grids L1, L2 and L3 and the solution based upon the locations of parking meters in Figure 5.12, and we notice the following findings, namely:

- Goodness values tend to increase as the grid refines, and so a correlation between the size of the cells and the goodness may be found. Indeed, it is confirmed that too wide area show an aggregate behavior whose informative power is limited.
- Goodness of parking meters is slightly higher than the one for “myCicero” for the L1 and L2 grids, while the contrary is found when considering L3 grid or model built using the parking meters locations.
- “myCicero” values have higher variance than the parking meters one. Indeed, the first ones range over a wider set of values considering the solely IQR or all values including outliers both.

Furthermore, by looking to the notches of the boxplots:

- Considering a same grid granularity, notches of “myCicero” and parking meters do intersect in all the cases, and so we can conclude there is not a statistically significant difference between the medians.
- Notches never intersect considering grids of different granularities on “myCicero” data, and so the refinement of the grid leads to an increased effectiveness of the solution by gradually detecting situated behaviors.
- Considering parking meters data on different grid granularities, notches intersect only for L1-L2.

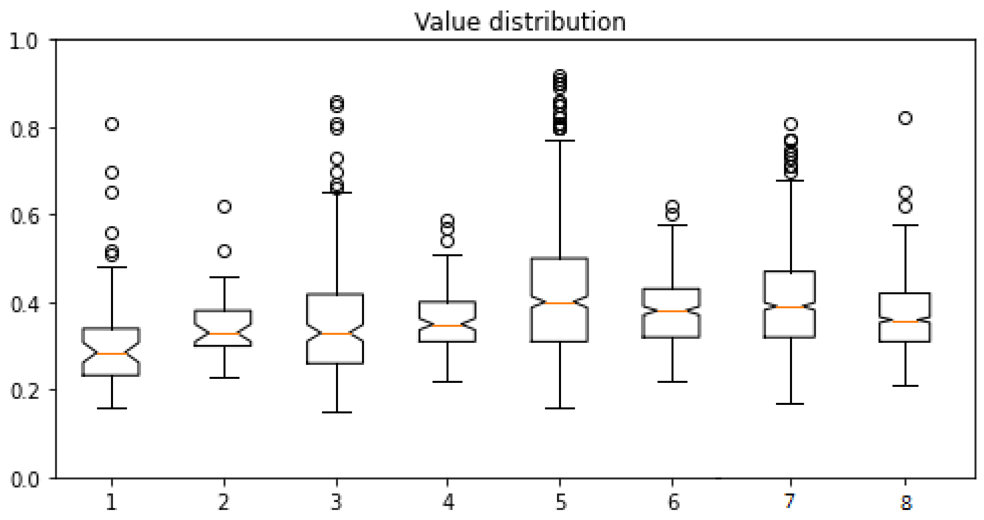


Figure 5.12: *Distribution of the goodness values for different geometric solutions. Areas with no values, whose goodness is 1, are not considered. From left to right: (1) L1 "myCicero", (2) L1 parking meters, (3) L2 "myCicero", (4) L2 parking meters, (5) L3 "myCicero", (6) L3 parking meters. In the end, also goodnesses obtained using the parking meters locations as set of starting points are reported (7,8).*

In the end, Figure 5.13 reports the scatter plot for the grid L2 (the others grids are very similar, and so omitted for conciseness) and for the model built using the locations of parking meters.
In the first case, there is a quite strong non linear negative correlation between the goodness and the quantity of data for "myCicero". Moreover, by looking to values reported in Table 5.1, we find out that such correlation increases as the cells shrink.
On the contrary, such sharp correlation does not exist for parking meters, but it tends to increase as the grid refines.
The existing correlation confirms that areas where the most of the stops are located may provide limited information when considering "myCicero" data, while such a problem does not subsist when dealing with parking meters. Further analysis should be executed in this direction to understand the cause of this difference.
When considering the model built using the locations of parking meters, we find out that there is no correlation at all, and so the behavior goodness of a region does not depend on the quantity of available data.

Thus, we can conclude that this latter model generally performs better since it equalizes the best results in term of goodness exposed by the L3 grid, but it goes beyond since no region is privileged depending on the quantity of available data, factor that emerged to be of a primary importance in previous analysis[3].

	L1	L2	L3
myCicero	-0.77	-0.71	-0.69
Parking meters	-0.33	-0.31	-0.41

Table 5.1: *Correlation between goodness and quantity of data using the Spearman correlation index on grids G1, G2, G3.*

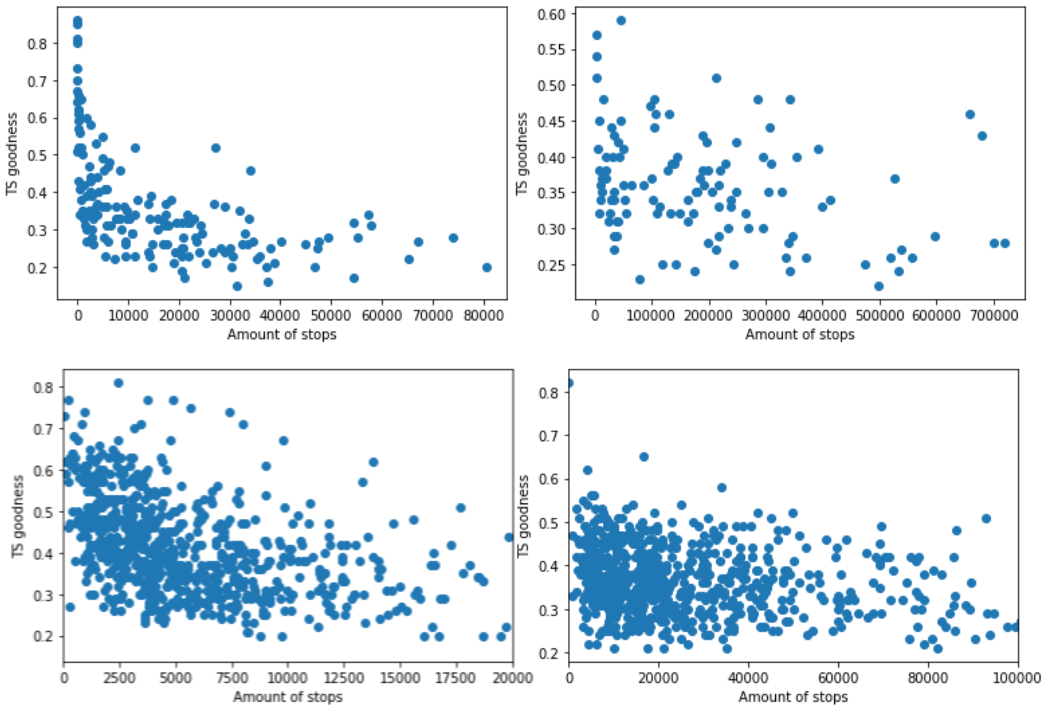


Figure 5.13: *Correlation between goodness and quantity of data (number of stops) for grid G2 (top row) and model built using the locations of the parking meters (bottom row).*

[3]Talking about capacity or quantity of data is equivalent since they are bound by a very strong positive correlation.

Chapter 6

Conclusions

6.1 Summary

Firstly, a data analysis stage was executed to explore the data and to mine any useful information that might come handy during the development of the system itself. In particular, the datasets were probed on time and spatial dimensions both, and new interesting information emerged about trends, seasonalities and features that influence the results (e.g., local territory peculiarities).

Then, starting from the previous findings and considering the topics explored by related works, the underlying mathematical and geometric models were defined to handle few critical aspects such as the division of the territory in regions, the estimation of the capacities, the merging of partial results coming from different data sources.
The prediction of the number of occupied parking lots was framed as a regression problem, and the "Prophet" model [11] was employed so that recurrent business aspects could be caught without any ad hoc countermeasure. For the time being, the forecasting model exploits "myCicero" and parking meters data both.

Finally, all the components of the system were developed exploiting a SOA approach, and various technologies and programming paradigms were to used develop all the services.

In the end, few performance evaluation tasks were executed, and it was found out that the model better behaves in regions containing more than a certain fixed quantity of data and that the adopted geometric model highly influences the final results. Hence, particular attention must be paid to the correct split of the territory so that situated behaviors can be correctly detected. Moreover, different data sources exposed different type of errors and different critic areas of the territory.

6.2 Future developments

Besides the deploy of the system and the completion of some performance evaluation tasks, the following improvements can be made to refine the solution, that is:

- Change of the geometric model and switch to a graph-oriented solution. As emerged from the data analysis stage, this latter solution should better perform than the current one based on a Voronoi tessellation.
- Inclusion of seasonal tickets data into the final forecast.
- Addition of further exogenous factors that resulted to be decisive in related works (e.g., weather).
- Testing of different forecasting models to predict the parking occupation.

Bibliography

[1] Eleni I. Vlahogianni, Konstantinos Kepaptsoglou, Vassileios Tsetsos, Matthew G. Karlaftis. *A Real-Time Parking Prediction System for Smart Cities*. Journal of Intelligent Transportation Systems, 2015, 10.1080/15472450.2015.1037955.

[2] Eleni I. Vlahogianni, Matthew G. Karlaftis. *Testing and Comparing Neural Network and Statistical Approaches for Predicting Transportation Time Series*. Transportation Research Record, 2013, 2399(1):9-22, 10.3141/2399-02.

[3] Ningxuan Feng, Feng Zhang, Jiazao Lin, Jidong Zhai, Xiaoyong Du. *Statistical Analysis and Prediction of Parking Behavior*. Network and Parallel Computing, Springer International Publishing, 2019.

[4] Shuguan Yang, Wei Ma, Xidong Pi, Sean Qian. *A deep learning approach to real-time parking occupancy prediction in transportation networks incorporating multiple spatio-temporal data sources*. Transportation Research Part C: Emerging Technologies, Volume 107, 2019, 248-265, ISSN 0968-090X.

[5] Feng Zhang, Ningxuan Feng, Yani Liu, Cheng Yang, Jidong Zhai, Shuhao Zhang, Bingsheng He, Jiazao Lin, Xiaoyong Du. *PewLSTM: Periodic LSTM with Weather-Aware Gating Mechanism for Parking Behavior Prediction*. Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2020, 4424-4430, 10.24963/ijcai.2020/610.

[6] Zhirong Chen, Jianhong (Cecilia) Xia, Buntoro Irawan. *Development of Fuzzy Logic Forecast Models for Location-Based Parking Finding Services*. Mathematical Problems in Engineering, 2013, 10.1155/2013/473471.

[7] `https://www.mycicero.it/`

[8] `https://www.pluservice.net/it/`

[9] `https://easyparkitalia.it/`

[10] `https://www.digithon.it/startups/apparked`

[11] S. J. Taylor, B. Letham. *Forecasting at scale.* The American Statistician, 2018, 72(1), 37-45.

[12] A. Harvey, S. Peters. *Estimation procedures for structural time series models.* Journal of Forecasting 9, 1990, 89–108.