POLITECNICO DI TORINO

Engineering Faculty

Computer Engineering - Data Science

Master's Degree Thesis

Extraction of spatio-temporal sequences

Supervisor

prof. Paolo Garza

Candidate

Andrea Cossio

ACADEMIC YEAR 2019-2020

Contents

List of Figures

List of Tables

Abstract

Considering the field of association pattern mining, FP-Growth is without any doubt one of the most famous algorithms for extracting frequent itemsets. Thanks to Spark's efficient parallel implementation, this algorithm is also the perfect candidate when working with big transactional databases. This work introduces a novel procedure to convert and aggregate multiple multivariate time series into multiple discrete sequences of data, in such a way that it is then possible to extract *spatio-temporal sequences* using FP-Growth.

Chapter 1

Introduction

Association pattern mining is one of the most important branches of data mining: it focuses on the extraction of rules that describe specific patterns within the data. The main goal of this data mining process is the discovery of insights, in terms of rules and patterns, that can be leveraged to generate a wide variety of benefits. Firstly adopted for market-basket data analysis to improve target marketing and shelf placement of items, it is today used for a broad range of applications such as clustering, classification, outlier analysis, text mining, recommendations, and a great variety of other problems.

The aim of this work is to extract *spatio-temporal sequences* of events using association pattern mining algorithms. Starting from a dataset made of multiple time series simultaneously collected at different locations, the newly proposed procedure starts with the extraction of *events of interest* from the time series and proceeds with the conversion to discrete sequences of data. Thanks to the particular structure given to these sequences, that contain information about both time and space of the extracted events, the application of a sequential pattern mining algorithm allows the discovery of *spatio-temporal sequences* of events. Since the Spark framework contains a very efficient parallel implementation of FP-Growth, called PFP [8], the proposed model needs to perform an additional step of conversion from discrete sequences to transactions, so that it is then possible to extract the *spatio-temporal sequences* more efficiently with this algorithm.

The thesis is organized in the following way. Chapter 2 (*Background and related work*) provides a general overview of association pattern mining and contains a brief introduction to the topics of spatio-temporal data and sequential pattern mining. Chapter 3 (*The proposed model*) describes the new procedure for extracting the *spatio-temporal sequences.* Taking into consideration Barcelona's bike sharing dataset, Chapter 4 (*Dataset analysis*) and Chapter 5 (*Experimental results*) contain respectively the exploratory analysis of the dataset and the experimental results obtained by applying the proposed model to it. Finally, Chapter 6 (*Conclusions*) draws some general conclusions and provides also some suggestions that could be useful to improve the model in future developments.

Chapter 2

Background and related work

In this chapter, after a quick revision of association pattern mining's core concepts, two of the most common *state-of-the-art* frequent pattern mining algorithms will be presented: Apriori and FP-Growth. In the last two sections, the topics of spatio-temporal data and sequential pattern mining will be briefly reviewed.

2.1 Association pattern mining

2.1.1 Overview

Association pattern mining [1] is a classical example of an *unsupervised* data mining task that aims at discovering interesting relationships between groups of items inside transactional databases. These relationships, also called *associations*, are rules in the form of an implication between two groups of items. This problem was firstly introduced in the context of supermarket data analysis to find patterns regarding items bought by customers. For example $\{Bread, Butter\} \Rightarrow \{Milk\}$ is a rule that can be interpreted in the following way: *"Whenever a customer buys bread and butter together, he is also likely to get milk in the same purchase"*. As proposed in the original paper, the problem of mining association rules can be decomposed into two sub-problems:

- Extraction of *frequent itemsets* from a transactional database.
- Extraction of *association rules* from the extracted frequent itemsets.

These two steps of the process will be described more in detail in the following sections.

2.1.2 Frequent pattern mining

The definition of *frequent pattern mining* relies on some preliminary notions that will be presented now.

Let T be a database of n transactions denoted by $T_1, T_2, ..., T_n$, where each transaction T_i is a set of items taken from the universe of items U. Let also I be a k-itemset, where k is the exact number of items that it contains taken from the same universe U. The fraction of transactions in which I appears as a subset is called *support*.

Definition 2.1 (Support) *The support of an itemset I is defined as the fraction of transactions T_i in the database T that contain I as a subset.*

$$sup(I) = \frac{\#\{\forall\, T_i \in T \mid I \subseteq T_i\}}{\#T} \tag{2.1}$$

Intuitively, the more often two items appear together in the transactions, the higher the correlation that they have, and therefore the more relevant that they are for extracting association rules. Consequently, the most important itemsets are the ones with the highest support: more precisely, an itemset is said to be *frequent* when its support is higher than a predefined minimum threshold value called $minsup$. The problem of frequent itemsets mining can be formalized with the following definition.

Definition 2.2 (Frequent itemsets mining) *Given a database of transactions T, find all the itemsets $I \subseteq T_i$ that have a support greater than a predefined minimum support $minsup$.*

Due to the fact that transactional databases usually contain a lot of distinct items, it is challenging to develop scalable methods for mining frequent itemsets in big data contexts. More precisely, it is very computationally expensive to count the support of all the combination of items, that given a universe U are $2^{|U|} - 1$. Nevertheless, there are two interesting properties of frequent itemsets that allow to partially solve this problem. Indeed, since they impose a strong constraint on the structure of frequent patterns, these two properties are often exploited to prune the search space of the itemsets.

The first property, called *support monotonicity property*, is the direct consequence of the fact that if a subset I is contained in a transaction T_i, also any subset J of the itemset I must be contained in the same transaction T_i. Therefore, the support of the itemset J will always be greater or equal than the one of I.

Property 2.1 (Support monotonicity property) *The support of every subset J of an itemset I is always greater than or equal to the support of I.*

$$sup(J) \geq sup(I) \qquad \forall J \subseteq I \tag{2.2}$$

The second property, called *downward closure property*, is a straightforward consequence of the support monotonicity one.

Property 2.2 (Downward closure property) *Every subset J of a frequent itemset I is also frequent.*

2.1.3 Association rule mining

As previously mentioned, the second step of association pattern mining involves the extraction of *association rules* starting from the frequent itemsets.

Let r be a rule defined as an implication in the form $X \Rightarrow Y$, where X and Y are two sets of items[1], and respectively the *antecedent* and the *consequent* of the rule. This rule can be interpreted as the conditional probability that a transaction T_i contains Y given that it also contains X; this probability is called *confidence* of the rule.

Definition 2.3 (Confidence) *The confidence of a rule $X \Rightarrow Y$ is defined as the fraction between the support of the itemset $X \cup Y$ and the support of X.*

$$conf(X \Rightarrow Y) = \frac{sup\,(X \cup Y)}{sup\,(X)} \tag{2.3}$$

Obviously, the rules with the highest confidences are the most interesting ones. Therefore, as done with the itemsets, it is possible to define a minimum threshold value to filter out all the irrelevant rules; in this case, this value is called $minconf$.

Definition 2.4 (Association rule) *A rule r in the form $X \Rightarrow Y$ is said to be an association rule with minimum support $minsup$ and minimum confidence $minconf$, when it satisfies the following conditions:*

[1] In reality, in the original paper the consequent of the rule (Y) is intended as a single item. Nevertheless, now the concept has been extended to any set of items.

- *The support of $X \cup Y$ is at least $minsup$.*
- *The confidence of the rule is at least $minconf$.*

From this definition, it is possible to notice how the union of X and Y is required to be a frequent itemset: this guarantees that there is a sufficient number of transactions that are relevant to the rule. It is for this reason that association rules are extracted starting from frequent itemsets.

Differently from the problem of mining frequent itemsets, the extraction of association rules is much simpler and more straightforward. Assumed I to be a frequent itemset, a simple way of generating the rules would be to partition it into all the possible combination of sets X and $Y = I - X$ so that $I = X \cup Y$ and then determine all the rules in the form $X \Rightarrow Y$ that have a confidence greater than $minconf$.

2.2 Apriori

2.2.1 Overview

Apriori [2] is one of the oldest *candidate generation*-based algorithms for mining frequent itemsets. In addition to the adoption of a breadth-first search and a hash tree structure to count itemsets efficiently, Agrawal and Srikant leveraged Property 2.2 (*Downward closure property*) to prune the search space of the candidate itemsets. Since they were the first ones to discover such characteristic of frequent itemsets, this property is also known as the *Apriori principle*.

2.2.2 The algorithm

The Apriori algorithm is based on a *bottom-up* approach. Starting with the generation of the frequent 1-itemsets by counting the support of the individual items belonging to the universe U and identifying those that are frequent, it proceeds with a repetitive procedure that consists of three steps:

- *Candidate generation.* During this first step, the frequent $(k-1)$-itemsets identified during the previous pass are combined together to generate candidate itemsets of length k. Since a k-itemset can be generated by multiple combinations of $(k-1)$-itemsets, to avoid redundancy in candidate generation, it is often useful to impose an ordering for the items. In this way, there is only one way to generate the k-itemsets. Also, thanks to the *Apriori* *principle*, using this approach ensures that no frequent itemset will be missed.

- *Pruning.* In this second step, to further reduce the number of candidates, the downward closure property is used again; indeed, a candidate k-itemset cannot be frequent when it contains as subset a $(k-1)$-itemset that does not belong to the set of frequent $(k-1)$-itemsets. Therefore, such candidates can be safely discarded.
- *Support counting.* Finally, the database is scanned to count the support of the remaining candidates so that all those that are not frequent can be removed.

This process continues until no new frequent itemsets are found.

2.2.3 Conclusions

Even if this new method of pruning candidates has represented a big step-up in terms of efficiency with respect to previous algorithms, the number of candidates in big transactional databases can potentially be so huge that the candidate generation step would still be very costly. Also, as most of the other algorithms, to count the support of the itemsets Apriori needs to scan the database multiple times which is a very computationally expensive operation.

2.3 FP-Growth

2.3.1 Overview

FP-Growth [5] is a *divide-and-conquer* algorithm for mining frequent itemsets that, differently from most of the Apriori-like algorithms, does not require the generation of candidates. The main novelty of this algorithm was the introduction of an extended prefix-tree, called FP-tree, that allowed to store crucial information about frequent patterns in a very compact way. More precisely, this structure can be used to efficiently count the support of frequent itemsets.

This algorithm can be seen as a two-steps process that starts with the construction of a FP-tree from a set of transactions and that proceeds with the extraction of all the frequent itemsets by pattern growth. To better understand these two steps, the algorithm will be presented now alongside with an example: the transactions are reported in Table 2.1.

Transaction ID	Items	Sorted frequent items
1	A, B, C, D, E, F, G, H	C, D, B, E, A
2	B, C, D, E, I, L, M	C, D, B, I, E
3	C, I, M, N, O	C, I
4	A, D, I, P, Q	D, I, A
5	A, B, C, D, E, L, R, S	C, D, B, E, A

Table 2.1: Set of transactions.

2.3.2 FP-tree construction

The first phase of FP-Growth, that consists in the construction of the FP-tree, can be further divided into the following two steps:

- *Header table definition.* During this step, that requires scanning the database for the first time, the *header table* is generated. This table is a list that contains in descending order all the supports of all the individual items that are found in the transactions. For example, imposing $minsup = 3$, starting from the transactions of Table 2.1, the resulting header table is reported in Table 2.2.

Item	Support
C	4
D	4
B	3
I	3
E	3
A	3

Table 2.2: Header table.

- *FP-tree construction.* After having generated the header table, the database is scanned for the second and last time. Each encountered transaction is filtered retaining only its frequent items (last column of Table 2.1) and sorted according to the previously built header table. These itemsets are then inserted inside a FP-tree. The insertion must follow any already existing prefix path and new branches must be created only when needed. In the first case, the frequency count of each item must be incremented by one, while in the second case, the frequency of each item must be set to 1. Figure 2.1 shows the resulting FP-tree.

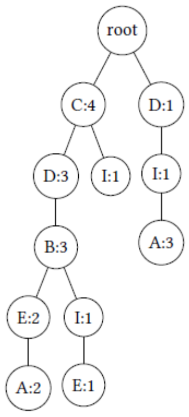

Figure 2.1: FP-tree.

2.3.3 Pattern fragment growth

The second phase, that involves recursively visiting the FP-tree to extract the frequent itemsets, can be divided into two steps:

- *Conditional pattern base.* Starting from frequent prefix patterns of length 1, the conditional pattern bases are generated. Given a prefix pattern, its *conditional pattern base* is the set of frequent paths in the FP-tree that share that pattern. In this "sub-database", the frequencies are adjusted to match the support of the prefix pattern.
- *Conditional pattern tree.* Then, for each conditional pattern base, another FP-tree is created and the process is repeated recursively. The frequent itemsets are obtained by concatenating the prefix paths with the frequent itemsets that are extracted from their conditional FP-trees.

2.3.4 Conclusions

Thanks to FP-trees, FP-Growth needs to scan the database only twice. For this reason, this algorithm achieves better performances than any of the previous algorithms. Also, thanks to the particular structure of the trees, it has been possible to define a parallel implementation, called PFP [8], that scales very well with big databases.

2.4 Contextual data representations

2.4.1 Overview

Contextual data representations are a particular kind of data that is characterized by two types of attributes. The first ones, called *contextual* attributes, provide the reference points in which the behavioural attributes are measured. The second ones, called *behavioural* attributes, contain the values that have been measured at each of the contextual reference points.

2.4.2 Temporal data

Temporal data is the most common contextual data representation in data mining applications and is often generated by continuous measurements over time. This type of data is characterized by a single contextual attribute that is the temporal reference of the measurements, called *timestamp*, and by one or multiple behavioural attributes that describe the recorded values at each timestamp. When the behavioural attributes are numerical, temporal datasets are called *time series*.

Definition 2.5 (Multivariate Time Series Data) *A time series of length n and dimensionality d contains d numeric behavioural attributes collected during each of the n different timestamps $t_1, t_2, ..., t_n$.*

Instead, when the behavioural attributes are categorical, temporal datasets are called *discrete sequences*.

Definition 2.6 (Multivariate Discrete Sequence Data) *A discrete sequence of length n and dimensionality d contains d categorical behavioural attributes collected during each of the n different timestamps $t_1, t_2, ..., t_n$.*

2.4.3 Spatial and spatio-temporal data

Spatial data is a contextual data representation where many behavioural attributes are recorded at different locations at the same time. The contextual attribute is often made of two spatial coordinates.

Definition 2.7 (Multivariate Spatial Data) *A spatial dataset of length n and dimensionality d contains d behavioural attributes collected at each of the n distinct spatial locations.*

Spatio-temporal data is a special form of spatial data where the behavioural attribute are recorded over time; therefore, this type of data is characterized by both spatial and temporal contextual attributes.

Definition 2.8 (Multivariate Spatio-Temporal Data) *A spatio-temporal dataset is made of N multivariate time series of length n and dimensionality d collected simultaneously at each of the N distinct spatial locations.*

2.5 Sequential pattern mining

2.5.1 Overview

Sequential pattern mining [3] is a special form of frequent pattern mining where the input dataset is a list of discrete sequences each containing multiple transactions of items in a specific temporal order. Similarly to frequent pattern mining, the goal of this data mining task is to find all the frequent patterns that satisfy a given $minsup$. However, in this case the frequent patterns are not simply subsets of the input itemsets but must have a precise form, called *sub-sequence*.

Definition 2.9 (Sub-sequence) *Let $S = \{S_1, ..., S_n\}$ and $R = \{R_1, ..., R_n\}$ be two sequences where each of their elements S_i and R_i are sets of items. R is said to be a sub-sequence of S if k elements $S_{i_1}, ..., S_{i_k}$ can be found in S such that $i_1 < i_2 < ... < i_k$ and $R_r \subseteq S_{i_r}$ for each $r \in \{1...k\}$.*

One of the most common applications of sequential pattern mining is the discovery of frequent sequences of items bought by customers in different purchases. For example, in the context of bookstores, one of these sequential patterns could be: *"Whenever a customer buys "The Pillars of the Earth", there is a 80% chance that within the next month he will also buy "World Without End""*. This pattern means that most of the customers that buy the first book will also buy the second one in following purchases.

2.5.2 Algorithms

Most of the algorithms for mining sequential patterns are generalized versions of the already existing ones for frequent itemsets:

- *Generalized Sequential Pattern Mining* [4]. This algorithm was the first to be proposed for mining sequential patterns and it is considered the extension of the Apriori algorithm because it exploits the downward closure property to generate candidates.
- *PrefixSpan* [7]. This algorithm was proposed to improve the performance of GSP by exploiting the same principles of FP-Growth; indeed, it is based on the same *divide-and-conquer* approach that exploits pattern growth structures to recursively mine the frequent patterns.
- *SPADE* [6]. This algorithm is an extension of vertical format-based frequent itemsets mining methods that adopts a *divide-and-conquer* approach to efficiently solve the problem with only three database scans.

A performance comparison of these three models shows that PrefixSpan and SPADE are both faster than GSP. However, with a large number of frequent sub-sequences, all three of these algorithms run very slow. For this reason and due to the fact that a parallel implementation of these algorithms does not exist inside the Spark framework, I have decided to introduce a new procedure that uses the parallel implementation of FP-Growth instead, called PFP [8]. This procedure will be described more in detail in the next chapter.

2.5.3 Spatio-temporal sequences

As mentioned above, the most common goal of sequential pattern mining is the discovery of sequential temporal patterns. Nevertheless, this data mining task can be applied to datasets that can contain any type of sequences. For example, the dataset that will be used to test the model proposed for extracting sequential patterns contains spatio-temporal data: the patterns that will be extracted will not only have a temporal meaning but also a spatial one. For example, referring to the context of the bookstores, the patterns will not only contain sequences of books bought by customers at different periods of time, but also in specific libraries. A more detailed description of this concept will be given in the last section of the next chapter and also in the last chapter when the model will be tested.

Chapter 3

The proposed model

In this chapter, the method proposed for extracting *spatio-temporal sequences* will be presented. After a general description of how the model has been devised, all the several steps that make the overall algorithm will be described in detail.

3.1 Overview of the algorithm

3.1.1 General idea

The main idea that brought to the definition of this new method for extracting *spatio-temporal sequences* is: *"If a spatio-temporal dataset can be converted to a series of transactions with a specific structure, then the rules that are extracted from such transactions can be interpreted as spatio-temporal sequences of events"*. Therefore, apart from the particular kind of interpretation that is given to the rules, the main novelty introduced by this model is the method for converting spatio-temporal datasets into a list of sequences in a way that is then possible to extract the rules.

3.1.2 Steps of the algorithm

From a high-level point of view, the algorithm can be divided into three main phases:

- *Extraction of events of interest.* In this first phase, the input dataset is filtered and its time series are converted to discrete sequences of data.

- *Sequence transformation.* Throughout this phase, the discrete sequences are converted into a new list of sequences that have a specific structure that enables the extraction of the rules.
- *Extraction of spatio-temporal sequences.* In this last phase, the previously generated list of sequences is mined with a frequent pattern mining algorithm that extracts the spatio-temporal rules.

Given the practical nature of the procedure, each of these steps will be presented together with an example in the next sections.

3.2 Extraction of events of interest

3.2.1 Overview

Even if there are algorithms that are specifically designed to work directly with time series, it is often convenient to convert them to discrete sequences because of the richer class of algorithms that has been developed for this second kind of data. For this reason, during this first phase of the process, the spatio-temporal dataset taken as input from the model is converted into another spatio-temporal dataset where the time series have been filtered and converted to discrete sequences of data. This operation can be divided into two steps:

- *Definition of events of interest.* In this step, the time series of the dataset are filtered and converted to discrete sequences of data containing only values that are relevant for the analysis.
- *Reduction of granularity.* In this step, the discrete sequences of data are simplified by reducing the temporal granularity.

3.2.2 Events of interest

This step consists in the identification of which *events of interest* are relevant for the analysis that is being conducted; more precisely, it consists in the identification of those events between which we would like to extract *spatio-temporal sequences.* Depending on the type of data and on the goal of the analysis, there could potentially be several ways to define such events. Nevertheless, a generic definition can be formalized in the following way.

Definition 3.1 (Event of interest) *An event of interest is a specific value*[1] *that corresponds to the occurrence of a particular event in the context of the application and that is relevant for the analysis.*

After having identified the most appropriate definition for the analysis that is being carried out, the extraction of the events of interest is obtained by filtering each time series so that they contain only these events. When the time series contain numerical data an additional step is required: the numerical values must be mapped to categorical ones. Section 3.2.4 contains a practical example that helps to better understand these operations.

3.2.3 Reduction of granularity

In order to create a more robust representation of the sequences, it is often convenient to perform a *reduction of granularity* of the temporal attribute. This operation consists in the temporal alignment of the events to equally spaced timestamps. To obtain this series of timestamps, two values must be defined:

- *Initial timestamp.* This is the starting point in time of the series.
- *Granularity of interest.* This is the fixed distance between the timestamps expressed as a number of minutes.

On one hand, choosing the first value is very simple: even if it is possible to use any timestamp, it is often recommended to use the first one in temporal order that is available in the dataset. On the other hand, choosing the right interval length is very difficult because the appropriate value depends on the context and on the type of patterns that we are looking for: a first possible intuition is that for short term patterns the granularity should be smaller, while for long term patterns it should be larger. However, there are also other factors that need to be considered when choosing the interval length: since these factors depend on future concepts, they will be explained more in detail in Section 3.3.5.

Before describing how the operation of reduction of granularity is carried out, it is necessary to define what an *interval* is.

[1]When the time series are characterized by multiple behavioural attributes the events of interests must be defined in a way that the extracted values are still one-dimensional.

Definition 3.2 (Interval) *Given an initial timestamp t_0 and a granularity g, the following series of timestamps is obtained: $t_n = t_0 + n * g$. An interval is any period of time between two consecutive timestamps t_n and t_{n+1} and is uniquely identified by the number n.*

Following this definition, the reduction of granularity is obtained by aligning all the events to the timestamp t_n, where n is the identifier of the interval they belong to, and by merging together those that belong to the same interval.

Depending on how the events of interest have been defined and on the chosen granularity of time, there could be intervals that do not contain any event of interest, for example when they occur very rarely or when the chosen interval length is too small, or there could be intervals that contain many of them, for example when they occur very often or when the chosen interval length is too big. In the first case, which is also the most common one, no further operation is needed. In the second case, it is important to choose how the events should be merged together, especially when the values are different because there could be some outliers. The following section contains a practical example that helps to better understand this operation.

3.2.4 Example

To better understand this first phase of the algorithm, two examples are now presented. The first example takes as input the numerical time series that is reported in Table 3.1.

	Timestamp	Measure	Event
0	2020-01-01 12:00:00	0	MIN
1	2020-01-01 12:01:00	0	MIN
2	2020-01-01 12:02:00	1	/
3	2020-01-01 12:03:00	2	/
4	2020-01-01 12:04:00	3	MAX
5	2020-01-01 12:05:00	2	/

Table 3.1: Numerical time series and its events of interest.

Imagine that in the context of this time series the measurements can take on values in a range between 0 and 3. A possible definition of *events of interest* could be: *"All those values that are 0 or 3"*. According to this definition, only the first two records and the fourth one should be retained. Also, since the measurements are numerical, the values need to be converted to categorical values; a possible solution consists in mapping 0 to the string "MIN" and 3 to "MAX". Let's now consider an interval length of two minutes: the discrete sequence that is obtained aligning the events is reported in Table 3.2.

	Timestamp	Interval ID	Events
0	2020-01-01 12:00:00	0	{MIN, MIN}
1	2020-01-01 12:02:00	1	{ }
2	2020-01-01 12:04:00	2	{MAX}

Table 3.2: Discrete sequence obtained from the numerical time series.

Notice how the first interval contains two events that are equivalent: it means that the environment stayed in the same state for more than one minute. Also, notice how during the second interval no event has been detected: it means that the sequence will have a missing interval ID. Both of these situations are not a problem because they do not affect how the model will perform.

Let's now consider a second example that takes as input the categorical time series that is reported in Table 3.3.

	Timestamp	Status	Event
0	2020-01-01 12:00:00	FULL	FULL
1	2020-01-01 12:01:00	NORMAL	/
2	2020-01-01 12:02:00	FULL	FULL
3	2020-01-01 12:03:00	FULL	FULL
4	2020-01-01 12:04:00	NORMAL	/
5	2020-01-01 12:05:00	EMPTY	EMPTY

Table 3.3: Categorical time series and its events of interest.

Imagine that in the context of this time series the values represent the fullness of a system. A possible way of defining the *events of interest* could be: *"All those values that are critical for the system and therefore all those situations in which the system is full or empty"*. In this case, since the values are already categorical, the time series only needs to be filtered and aligned. Given an interval length of two minutes, the resulting sequence is reported in Table 3.4.

	Timestamp	Interval ID	Events
0	2020-01-01 12:00:00	0	{FULL}
1	2020-01-01 12:02:00	1	{FULL, FULL}
2	2020-01-01 12:04:00	2	{EMPTY}

Table 3.4: Discrete sequence obtained from the categorical time series.

If the chosen interval length had been of three minutes, the second interval would have contained two different events. As previously mentioned, this is a problem that must be addressed and that can be associated to the presence of outliers or to the choice of interval length too big.

	Timestamp	Interval ID	Events
0	2020-01-01 12:00:00	0	{FULL, FULL}
1	2020-01-01 12:03:00	1	{FULL, EMPTY}

Table 3.5: Example of outliers after reducing the temporal granularity of the discrete sequence.

3.2.5 Summary

After this first phase of the process, the time series of the input spatio-temporal dataset D of size N have been filtered to contain only the events relevant for the analysis that is being conducted and have been converted to discrete sequences of data with a finer granularity of time. Therefore, the resulting dataset is still spatio-temporal but containing shorter discrete sequences of data.

3.3 Sequences generation

3.3.1 Overview

During this second phase of the process, the discrete sequences obtained from the extraction of the events of interest are transformed into a new list of sequences that have a specific structure that allows the extraction of *spatio-temporal sequences*. This phase can be divided into two steps:

- *Joining sequences.* In this step, the discrete sequences are merged together into a new single sequence.

- *Sliding window.* In this step, the newly generated sequence is transformed into a new list of sequences.

3.3.2 Joining sequences

This first step serves mainly as preparation and simplification of the following one. During this step, the N sequences[2] that have been generated in the previous phase are merged together. The new sequence will contain sets of events that occurred during the same interval of time and that have been collected in different places. This operation can be formalized with the following definition.

Definition 3.3 (Merge operation) *Given two discrete sequences $S_1 = a_n$ and $S_2 = b_n$, the third sequence that is obtained by merging them together is $S_3 = c_n = \{a_n, b_n\}_n$.*

To preserve the spatial reference of the events, they must be tagged with the *ID* of the location where they were registered. Section 3.3.4 contains a practical example that helps to better understand how this operation can be performed.

3.3.3 Sliding window

The second step of this phase is the most important one of the whole algorithm because it has the task of generating the sequences with the particular structure that enables the extraction of *spatio-temporal sequences.* This particular structure is obtained by applying a sliding window of length w to the sequence that has been generated in the previous step. This operation can be formalized with the following definition.

Definition 3.4 (Sliding window operation) *A sliding window of length w is a multi-step operation that transforms an input sequence of length n into a list of n sequences. During each step i, w consecutive elements of the original sequence starting from position i are collected into a new sequence.*

Following this definition, the new list of sequences is obtained by scanning the sequence of length n obtained in the previous step with a sliding window of length w and by collecting every

[2]Remember that these sequences correspond to the events of interest that have been collected at N different geographical locations.

single group of w consecutive intervals into a sequence each. After this operation, the dataset will contain n sequences at most, each consisting of w sets of events that have been collected in different locations during w consecutive intervals of time at a distance of g minutes each. In this way, the sequences will be characterized by a new contextual attribute that is an index that goes from 0 to w indicating the relative delta of time between the intervals. The following section contains a practical example of this operation.

3.3.4 Example

To better understand and visualise this second phase of the process, an example is now presented. Imagine that the input dataset is the one reported in Table 3.6: it contains two sequences of events that have been collected in two different locations. The first sequence contains two events of interest registered during the intervals 0 and 1, while the second one contains two events registered during intervals 1 and 3.

	Interval ID	Location	Status
0	0	1	FULL
1	1	1	FULL
2	1	2	FULL
3	3	2	FULL

Table 3.6: Spatio-temporal dataset containing 2 discrete sequences.

The sequence that is obtained by merging together these two sequences is reported in Table 3.7. As you can see, the events have been tagged with their location *ID*. Also, the second interval of time does not contain any event: this means that during this period of time no event of interest was captured at any location.

	Interval ID	Events
0	0	{1:FULL}
1	1	{1:FULL, 2:FULL}
2	2	{ }
3	3	{2:FULL}

Table 3.7: Discrete sequence obtained after joining space.

Let's now consider a sliding window of length $w = 3$: the list of sequences that are obtained by performing the sliding window operation is reported in Table 3.8. As you can see, each sequence contains one of the many possible combinations of consecutive intervals of events; the last $w - 1$ sequences contain less elements because they are at the end of the range of intervals.

	Sequence ID	Index 0	Index 1	Index 2
0	0	{1:FULL}	{1:FULL, 2:FULL}	{ }
1	1	{1:FULL, 2:FULL}	{ }	{2:FULL}
2	2	{ }	{2:FULL}	/
3	3	{2:FULL}	/	/

Table 3.8: Discrete sequences obtained after applying a sliding window of length 3.

3.3.5 Summary

After this second phase of the process, the discrete sequences of the spatio-temporal dataset have been merged together and converted to a new list of sequences using the sliding window operation. These new sequences contain sets of events that occurred at a fixed distance of time each, more precisely at a distance that corresponds to the new granularity of time chosen in the previous phase. Therefore, it is now possible to explain which are the other factors that need to be considered when choosing the interval length (g):

- *Interval length too small.* When the interval length is too small, there is the risk of not having enough events that frequently occur at the same distance in a sufficient number of input sequences: the number of generated sequences could be so big that the support of the itemsets and the confidence of the rules may be too low.
- *Interval length too big.* When the interval length is too big, there is the risk of having sequences that contain almost all the same critical events. Even if this results in a lot of frequent sub-sequences with high support, the concept of fixed distance between the events becomes too loose: since the events belonging to the same interval are merged together, we don't know anymore if they happened at the beginning or at the end of the interval. Therefore, two events belonging to two consecutive intervals may have occurred at a real relative temporal distance that can be much smaller or much bigger than the granularity value g.

3.4 Spatio-temporal sequences

3.4.1 Overview

Even if there are algorithms that are specifically designed for extracting sequential patterns, when working with large quantities of transactions these algorithms are not very efficient. Therefore, since the Spark framework contains a parallel implementation of FP-Growth [8], I have decided to use this algorithms instead. During this last phase of the process, the sequences that have been previously obtained with the sliding window operation are converted to transactions and then mined to extract frequent *spatio-temporal* itemsets and rules. This phase can be divided into two steps:

- *Conversion of sequences to transactions.* In this step, the list of sequences is converted to a list of transactions.
- *Spatio-temporal pattern mining.* In this step, the list of transactions is mined to extract spatio-temporal sequences.

3.4.2 Conversion of sequences to transactions

Since I'm planning to use FP-Growth to extract the frequent sub-sequences and rules, it is necessary to transform the sequences to transactions. The key difference between the two structures is that sequences contain temporal ordered sets of elements while transactions do not. Therefore, to prevent the loss of the temporal reference, before taking out the events from their sets of items, they must be tagged with the relative index i of the interval they belong to. In this way, even if the transactions will contain the events in disorder, it will still be possible to trace back the relative index of the interval the events belonged to. This operation will become more clear with the practical example that is presented in Section 3.4.4.

3.4.3 Spatio-temporal pattern mining

The last step of the process consists in the extraction of frequent sub-sequences and rules using FP-Growth; this is achieved by passing to the algorithm the transactions and the two parameters that are required by all the frequent pattern mining algorithms: the minimum support $minsup$ of the sub-sequences and the minimum confidence $minconf$ of the rules.

As previously mentioned, the only problem of using FP-Growth instead of sequential pattern mining algorithms is that there is the possibility of generating redundant frequent sub-sequences and rules: whenever an itemset or a rule does not contain any item with a relative index 0, it means that there exists a shifted version of the same sub-sequences or rule with the same support or confidence. Also, some rules may contain events in the consequent that happened before the ones in the antecedent. All of this sub-sequences and rules must be discarded.

After this operation, the list of rules with a support greater than $minsup$ and a confidence greater than $minconf$ is obtained. As we saw in Chapter 2 (*Background and related work*), the extracted rules can be interpreted as *spatio-temporal sequences* of events: whenever the events in the antecedent part of a rule take place, there is the possibility that the event in the consequent part of the rule will take place in following periods of time. The spatial and temporal references of the events are given by the *ID* of the location and by the relative index of the interval.

The last part of the following section contains an example of a *spatio-temporal sequences* and the interpretation that can be given to it.

3.4.4 Example

To better understand how sequences are converted to transactions, an example is now presented. Imagine that the input sequences are the ones reported in Table 3.9.

	Sequence ID	Index 0	Index 1	Index 2
0	0	{1:FULL, 3:EMPTY}	{1:FULL, 2:FULL}	{4:FULL}
1	1	{1:FULL, 2:FULL}	{4:FULL}	{2:FULL, 3:EMPTY}
2	2	{4:FULL}	{2:FULL, 3:EMPTY}	/
3	3	{2:FULL, 3:EMPTY}	/	/

Table 3.9: List of sequences.

The conversion can be achieved by extracting all the events from their sets of items and by tagging them with the relative index. The transactions that are obtained with this operation are reported in Table 3.10.

	Transaction ID	Events
0	0	{0:1:FULL, 0:3:EMPTY, 1:1:FULL, 1:2:FULL, 2:4:FULL}
1	1	{0:1:FULL, 0:2:FULL, 1:4:FULL, 2:2:FULL, 2:3:EMPTY}
2	2	{0:4:FULL, 1:2:FULL, 1:3:EMPTY}
3	3	{0:2:FULL, 0:3:EMPTY}

Table 3.10: List of transactions obtained after converting the sequences.

Finally let's see an example of a *spatio-temporal sequence*. Imagine that after the application of FP-Growth the following rule is obtained: $\{0:1:FULL, 1:2:EMPTY\} \Rightarrow \{2:3:FULL\}$. This rule can be interpreted in the following way: whenever the event "FULL" is recorded in location number 1 and during the next interval of time the event "EMPTY" is recorded in location number 2, there is the possibility that in the third interval of time the event "FULL" will be recorded in location number 3. The probability that this happens is given by the confidence of the rule.

Chapter 4

Dataset analysis

In this chapter, after a general introduction to bike sharing systems, the dataset that will be used to test the proposed model will be analysed in detail.

4.1 Dataset description

4.1.1 Bike sharing overview

In recent years, *bike sharing* systems have been one of the infrastructures that have grown the most in large cities. Since when these systems have established as permanent component in urban passenger transport, they have increasingly influenced the way we commute inside all the major metropolitan cities. Thanks to the large quantities of data that are collected by these systems, the context of bike sharing lends itself very well to the activity of data analysis.

In this work, a bike sharing system is intended as a *network* of multiple stations located in different geographical locations and containing thousands of bikes. The users of the service can rent such bikes at any station and then, after a short travel, they have to return them at any other station. Usually, each bike sharing station contains a sensor that collects information regarding its status over time; therefore, the datasets generated by bike sharing systems are the perfect candidates for testing the model proposed in Chapter 3 (*The proposed model*): indeed, according to Definition 2.8 (*Multivariate Spatio-Temporal Data*), the data that is collected over time at different spatial locations is defined as spatio-temporal.

4.1.2 Features

To test the model proposed by this work, I have decided to use a dataset that contains historical information about Barcelona's bike sharing system between May and September 2008. The first rows of the dataset are reported in Table 4.1. As you can see, each record is characterized by two contextual attributes, *station* and *timestamp*, and by two behavioural attributes, *used_slots* and *free_slots*.

	station	timestamp	used_slots	free_slots
0	1	2008-05-15 12:01:00	0	18
1	1	2008-05-15 12:02:00	0	18
2	1	2008-05-15 12:04:00	0	18
3	3	2008-05-15 12:06:00	0	18

Table 4.1: First records of the dataset.

The two behavioural attributes describe the number of bikes and the number of available slots that each station has during each timestamp. As you can see from Figure 4.1, that shows the data that has been collected by station number 1 during the first days of June, the two values fluctuate a lot over time. In Chapter 5 (*Experimental results*) we will see how the information provided by these two variables can be exploited to extract *spatio-temporal sequences*.

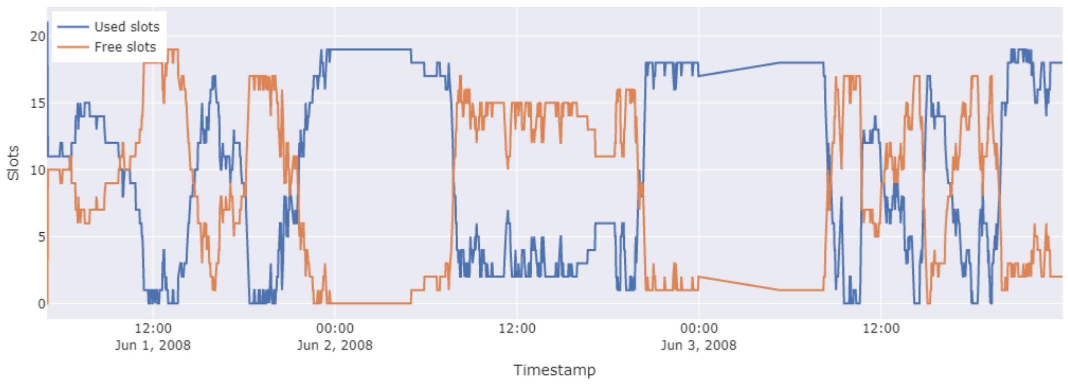

Figure 4.1: Data collected by station 1 during the first three days of June.

4.1.3 Dataset preparation

Before proceeding with the exploratory analysis, I have introduced two new columns in the dataset that provide useful information about the stations. As you can see in Table 4.2, the first columns that has been created is *total_slots*: this value represents the capacity of the station and is obtained by adding together the number of used and free slots. The second columns is *used_ratio*: this value represents the fullness of the station and is obtained dividing the number of used slots by the number of total slots.

	station	timestamp	used_slots	free_slots	total_slots	used_ratio
0	1	2008-05-15 12:01:00	0	18	18	0
1	1	2008-05-15 12:02:00	0	18	18	0
2	1	2008-05-15 12:04:00	0	18	18	0
3	1	2008-05-15 12:06:00	0	18	18	0

Table 4.2: First records of the dataset with the two additional columns.

4.2 Exploratory analysis

4.2.1 Overview

In this section, the dataset will be analysed feature by feature with the goal of getting a better idea of the data and also to find any possible problem that may negatively affect the performance of the model. Before starting with the analysis, it must be pointed out that the dataset does not contain any duplicated records. Also, even if we will see that there are some records that reports a total number of slots equals to 0, fact that could be associated to a missing value, the dataset does not contain any missing value in the traditional way.

4.2.2 Station

The first feature that I'm going to analyse is *station*, which is the spatial contextual attribute of the dataset. Indeed, even if the variable is characterized by numerical values that correspond to the *ID* of the stations, each one of them is also characterized by two geographical coordinates, latitude and longitude. Figure 4.2 shows the position of the stations.

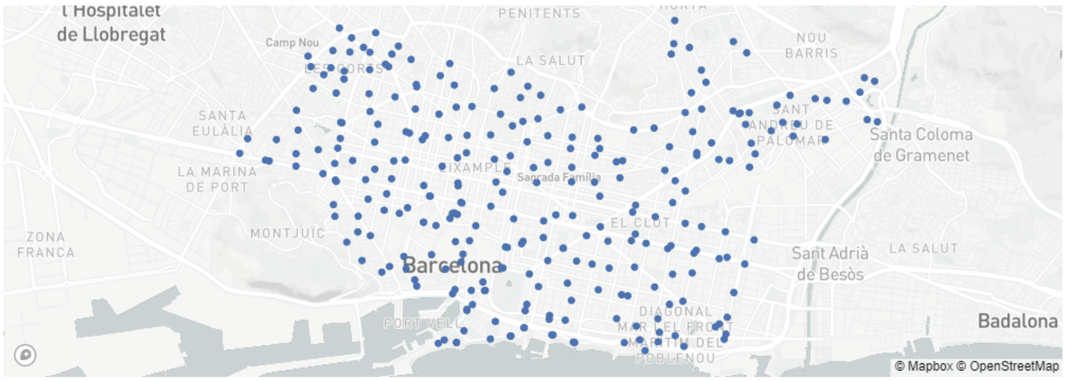

Figure 4.2: Barcelona's map reporting the geographical locations of the station.

One of the first things that can be checked is the number of recorded events for each station: this is obtained by grouping the dataset by *station* and then counting the number of records. As you can see from Figure 4.3, the bike sharing network of Barcelona is made of 284 stations, numbered from 1 to 284, and almost all of them have registered the same number of events.

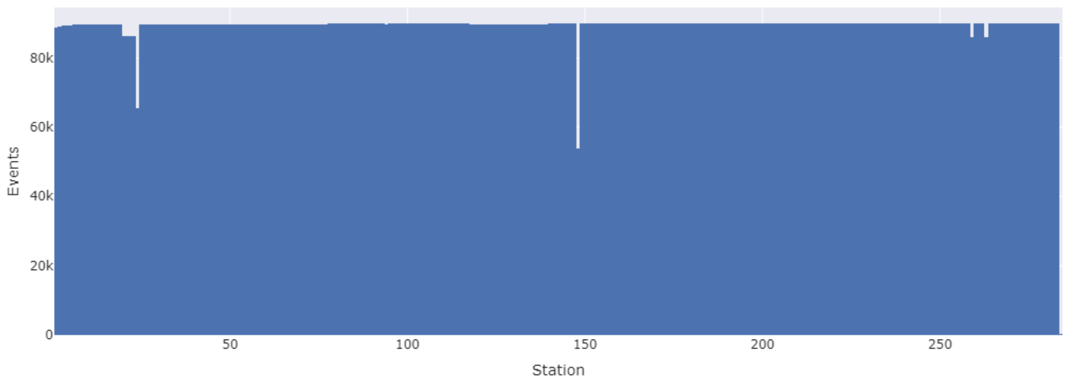

Figure 4.3: Number of events per station.

	station	events
0	284	165
1	148	53892
2	24	65613
3	263	85902
4	259	85903

Table 4.3: Stations with the least number of events.

However, there are some stations with less events than average. Table 4.3 contains the list of the stations with the least number of events: as you can see, stations 24 and 148 have around 40% less events then the maximum, and station 284 has only registered 165 events. Even if stations 24 and 148 recorded during only 60% of the time, Figure 4.4 shows that their periods of activity are rather continuous and without too much noise. Therefore, their available data can be used for the rest of the analysis. Instead, since station 284 has registered very few events, it will be removed from the dataset.

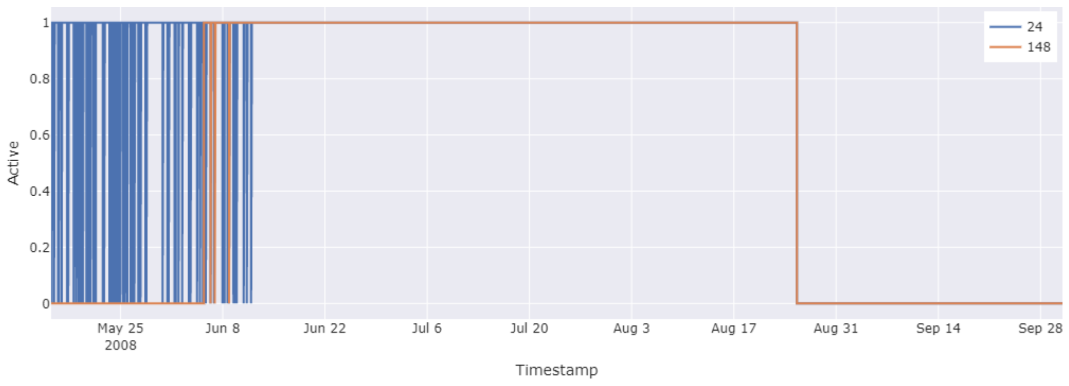

Figure 4.4: Activity of stations 24 and 148.

To analyse the interaction between the variable *station* and the behavioural attributes, it is necessary to group the dataset by *station* and then perform an aggregation function on the grouped data. Starting with *total_slots*, Figure 4.5 shows that the average capacity of the stations is between 20 and 25 slots and that, on average, station 221 has very low capacity. Also, there are some stations that are characterized by a very high variability of capacity. This information is an evident signal that there might be some outliers in the dataset.

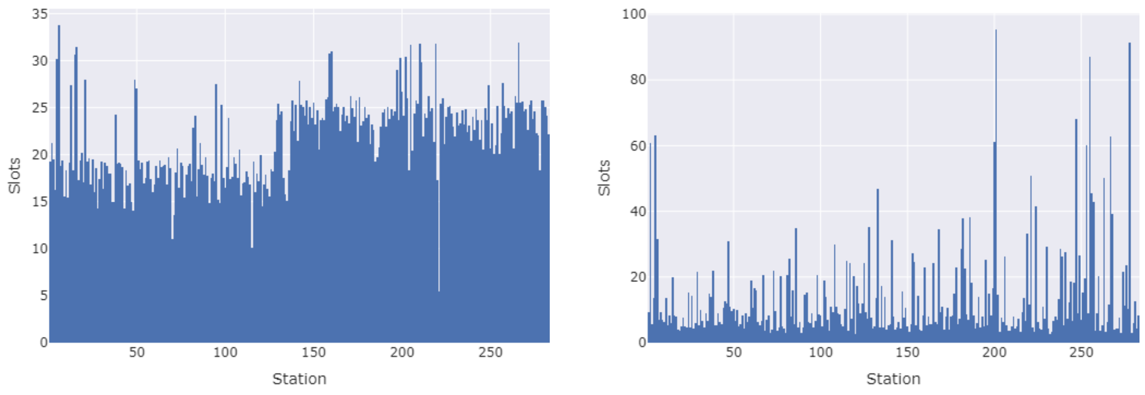

Figure 4.5: Mean (left) and variance (right) of total_slots per station.

Next, moving on to the variable *used_ratio*, Figure 4.6 shows that the stations tend to be almost empty most of the time. Also, it is possible to notice that station 221 is always empty. Therefore, interpreting the data of station 221 as an outlier, this station will be removed from the dataset.

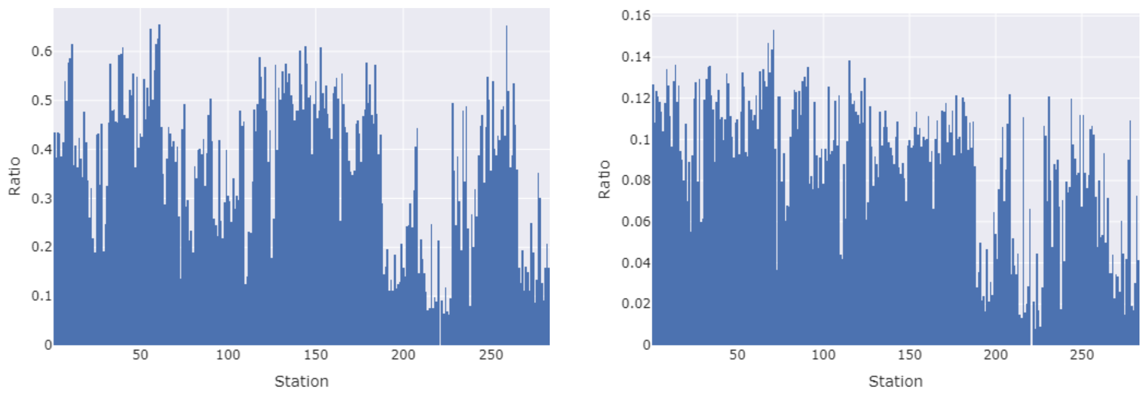

Figure 4.6: Mean (left) and variance (right) of used_ratio per station.

4.2.3 Timestamps

The second feature that I'm going to analyse is *timestamp*, which is the temporal contextual attribute of the dataset. One of the first things that can be checked is the number of active stations over time: this is obtained by grouping the dataset by *timestamp* and then counting the number of distinct station IDs.

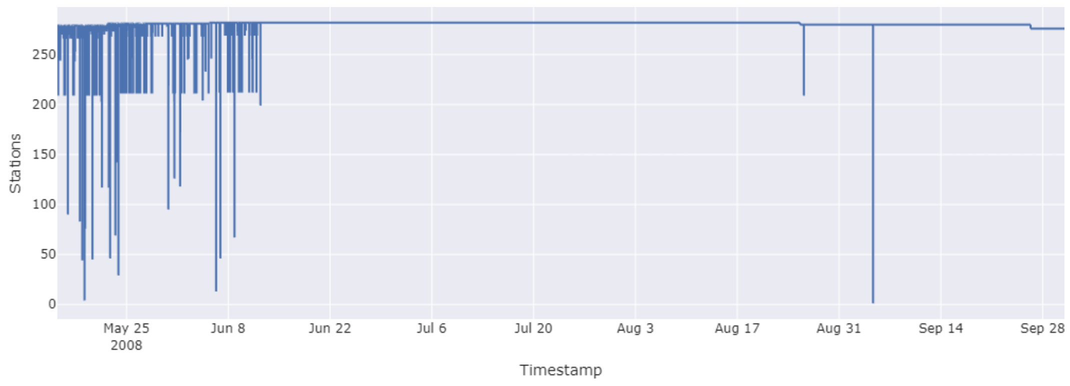

Figure 4.7: Number of active stations over time.

As visible from Figure 4.7, apart from an initial short period of time, almost all the stations have been active during all the available timestamps. Also, it is possible to notice that the events have been registered between *2008-05-15 12:01:00* and *2008-09-30 23:58:00*. Therefore, since the dataset contains four months and a half worth of data, if the sampling rates of the stations had been of one minute, there would have been around 196 thousands distinct timestamps; however, there are only 96 thousands distinct timestamps, which means that the average sampling rate of the stations has been of about two minutes.

It must be pointed out that not all the possible timestamps are represented in the dataset: there are indeed some temporal holes during which the stations didn't record anything. As visible in Figure 4.8, during the first month of data, there are a lot of intervals during which the system didn't record anything for more than 10 consecutive minutes.

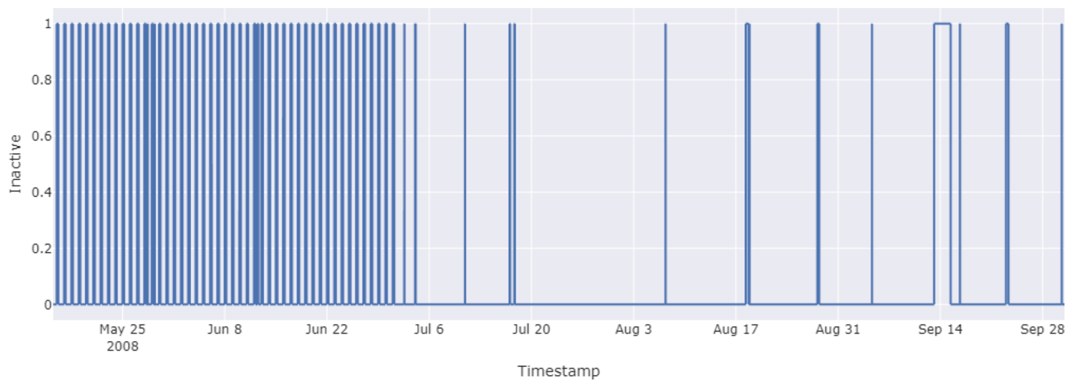

Figure 4.8: Temporal inactivity of the system for more than 10 minutes.

Similarly to what has been done with the stations, it is possible to analyse the interaction between this variable and the behavioural attributes by grouping the dataset by *timestamp* and then performing an aggregation function on the grouped data. In this case, since the aggregation would happen between different stations at a fixed timestamp, it doesn't make much sense to take the mean or the variance. Instead, it is possible to analyse the sum of the different variables.

Starting with *used_slots*, Figure 4.9 shows that on average there are between 2 thousands and 3 thousands bikes parked at the stations at each timestamp. Since this number represents a good indication of how many bikes there are in the system, it is possible to associate the positive trend that occurred during the month of August to the introduction of new bikes into the system. It is also evident how the sum of *used_slots* is very noisy: there are a lot of low peaks that almost reach 0. Since the stations are almost always active all together, it is not possible to associate these low peaks to a reduction of active stations: therefore, they must be outliers. In section 4.3.2 we will see how to remove the outliers that are causing these peaks.

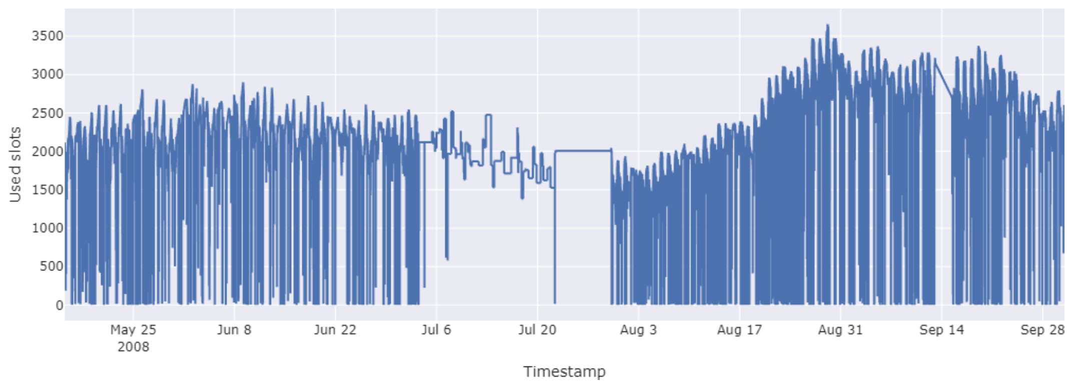

Figure 4.9: Sum of used_slots over time.

Moving on to the variable *free_slots*, it is possible to analyse the evolution of the total number of available slots over time. Figure 4.10 highlights again how there is a lot of noise related to the slots; also, notice how there is a small negative trend in August: this should be interpreted as the direct consequence of the introduction of new bikes in the system.

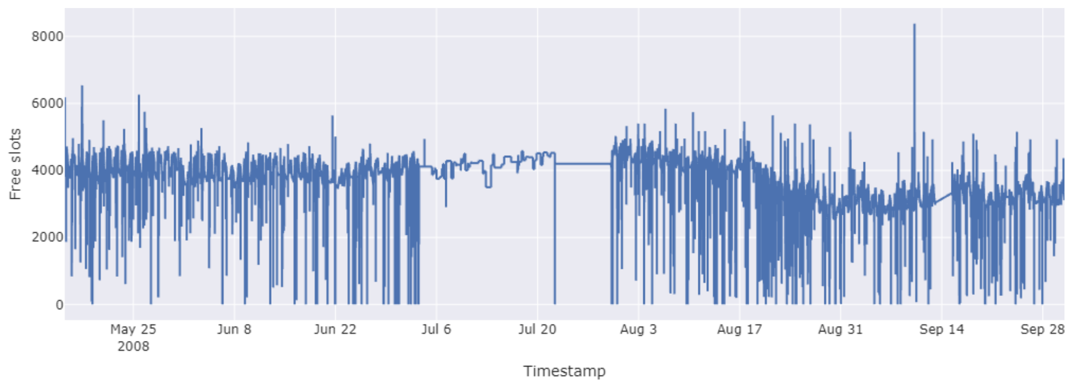

Figure 4.10: Sum of free_slots over time.

Analysing the variable *total_slots*, it is possible to understand how the total capacity of the system evolves over time. Looking at the results reported in Figure 4.11, we have the confirmation that there are a lot of outliers in the dataset: if the fluctuations of used and free slots are expected, the total number of slots of the stations should remain much more stable over time; however, the low peaks are present also here.

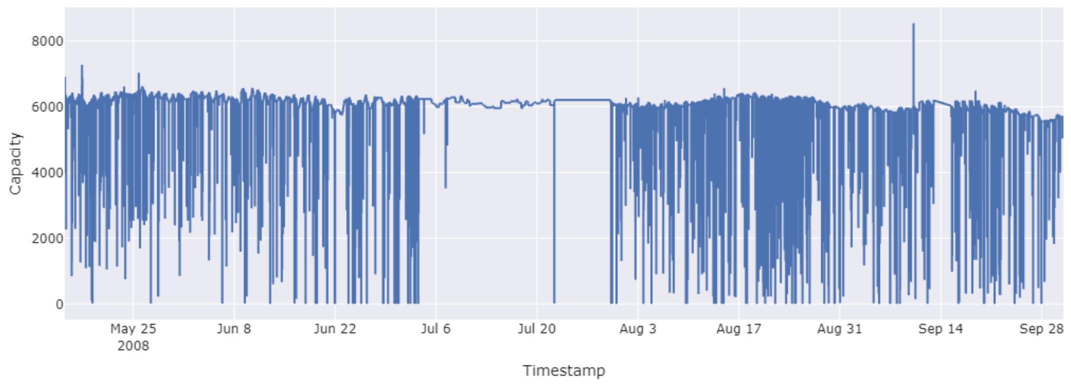

Figure 4.11: Sum of total_slots over time.

The variable *used_ratio* represents the fullness of each station at each timestamp: therefore, it doesn't make sense neither to take the sum across different stations, because it wouldn't be a fraction anymore, and neither the mean, because it would represent the average fullness of the stations, and not of the system. To obtain the average fullness of the system over time, we need to divide the sum of all the used slots by the sum of all the total slots. The result of this analysis is reported in Figure 4.12.

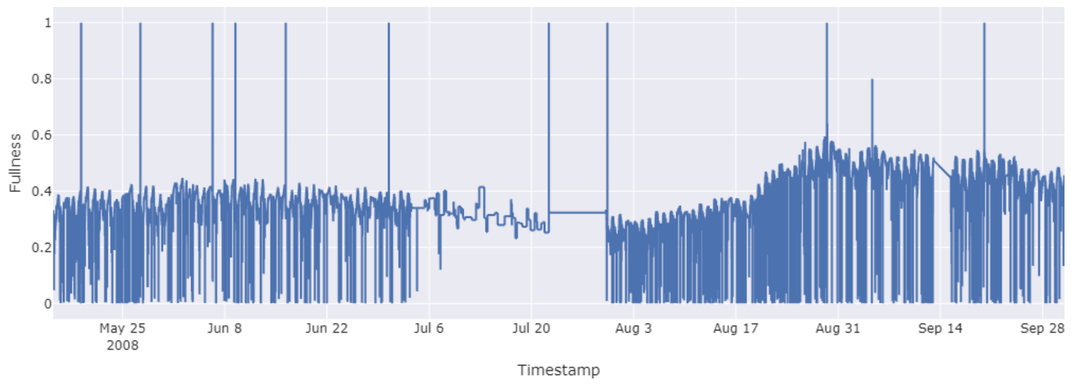

Figure 4.12: Sum of used_slots divided by the sum of total_slots.

All the previous figures show how during the month of July there is not much variability.

4.2.4 Slots

The last variables that I'm going to analyse are the ones regarding the slots of the stations. Since I have already analysed the interaction between these variables and the two contextual attributes, in this section I'm going to focus on their distribution. Starting with the variable *used_slots*, Figure 4.13 shows that the distribution is totally skewed towards 0: this means that the stations tend to be empty most of the time.

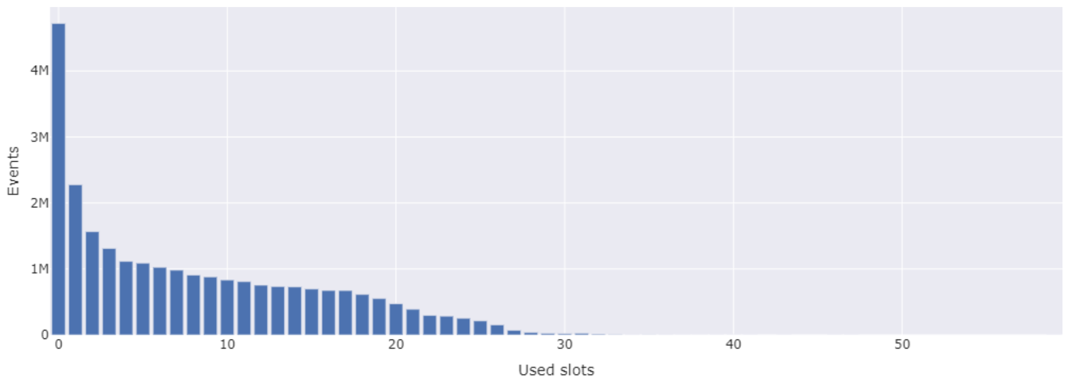

Figure 4.13: Distribution of used_slots.

Next, moving to the variable *free_slots*, from Figure 4.14 it is possible to notice that the distribution is much more uniform than the one of *used_slots*. Nonetheless, the most represented value is again 0, which means that the stations are full very often.

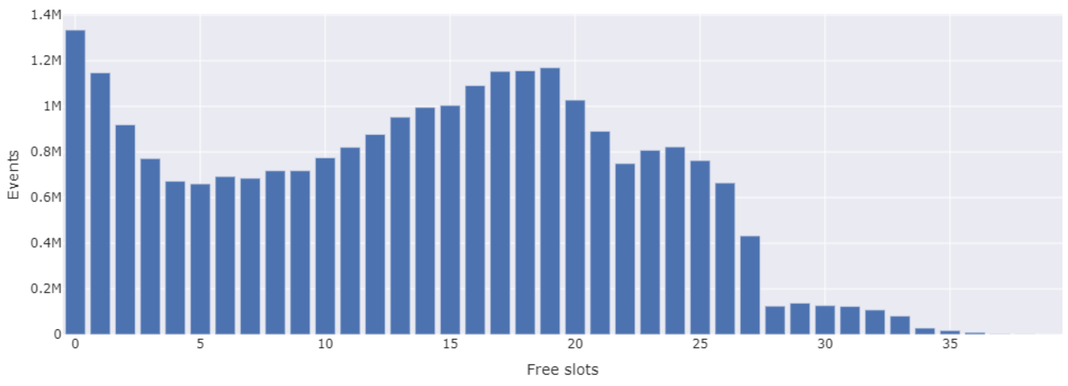

Figure 4.14: Distribution of free_slots.

Next, the distribution of *total_slots*, which is reported in Figure 4.15, confirms that the capacity of the stations is between 20 and 25 slots for most of the time. Notice how there are some records with 0 total slots that can be interpreted as failures or offline periods of the stations: therefore, these records will be removed from the dataset.

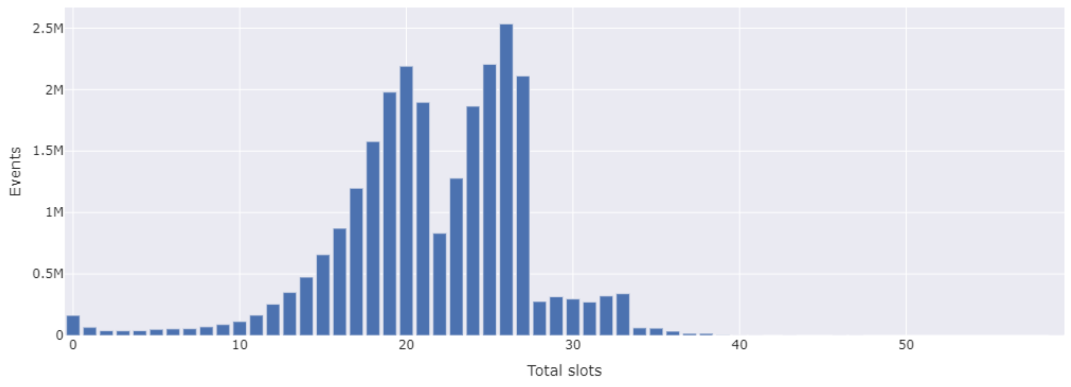

Figure 4.15: Distribution of total_slots.

Finally, the distribution of *used_ratio* is reported in Figure 4.16.

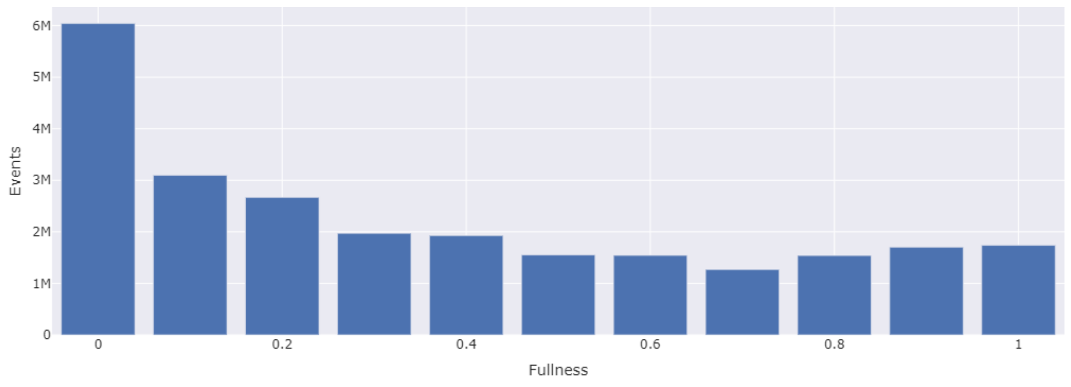

Figure 4.16: Distribution of used ratio.

4.3 Preprocessing

4.3.1 Overview

Before proceeding with testing the model proposed in Chapter 3 (*The proposed model*), all of the problems discovered in the previous sections must be addressed and resolved:

- *Stations.* Station 284 has very few events and station 221 is always empty.
- *System offline.* During the month of July the system appears to be offline.
- *Outliers.* There are outliers that generate noise in the data regarding the number of slots.

On one hand, solving the first two problems is very simple: indeed, it is sufficient to remove all those records that are related to stations 221 and 284 or that occurred during the month of July. On the other hand, detecting and removing the outliers of the time series is a little bit more complicated. In the next section, a method to remove the noise will be proposed and in Chapter 5 (*Experimental results*) we will see how this will affect the performance of the model.

4.3.2 Removing outliers

As we saw during the exploratory analysis, all of the time series of the dataset are characterized by a lot of noise. Even if the fluctuations in the number of *used* and *free* slots go beyond normal expectations, I have preferred to concentrate on the number of *total* slots because this time series should have remained much more stationary over time than the other two. The low peaks that characterize this time series can be classified into two categories: the first one includes all those records where the number of total slots is 0, while the second one corresponds to all the other values that deviate a lot from the mean without reaching 0. Removing the first kind of outliers is very simple because there is a precise rule for detecting them: it is indeed sufficient to remove all those records that have a total number of slots equals to 0. Figure 4.17 shows how there are already less low peaks in the data after removing this first kind of outliers.

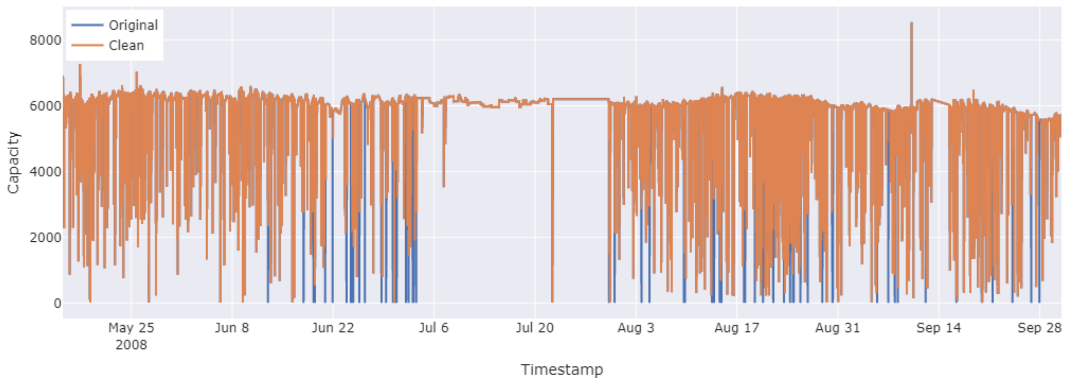

Figure 4.17: Total number of slots in the system before and after cleaning.

To remove the second category of outliers, I have used a method that consists in the identification of *point outliers*, which are data points that deviate significantly from their expected value. This operation can be divided into three steps:

- *Determine the forecasted values.* For each data point I have decided to define its expected value as the mean of the previous and subsequent 50 points.

- *Compute and normalize the deviations.* The deviation of each data point can be obtained by subtracting its value from the expected one. The normalized deviation of each data point can be obtained by subtracting the mean of the deviations from its deviation and then dividing by the square root of the variance of the deviations.
- *Remove outliers.* The resulting normalized deviations are equal to the Z-value of a normal distribution and provide a continuous alarm level of outlier scores. Therefore, the outliers can be detected and removed by using a threshold value on these scores. Often, because of the Z-value interpretation, a good value for the threshold is 3.

In the following section we will see how the removal of this second type of outliers definitely improves the stability of the time series.

4.3.3 Results

To better understand how the cleaning has affected the time series, I have analysed the number of total slots of two different stations before and after the removal of the outliers during the first week of August. Starting with station number 1, from Figure 4.18 it is possible to notice how removing the first kind of outliers is almost already enough to remove all the low peaks. The remaining ones have been removed with the second operation, the one that removes the point outliers.

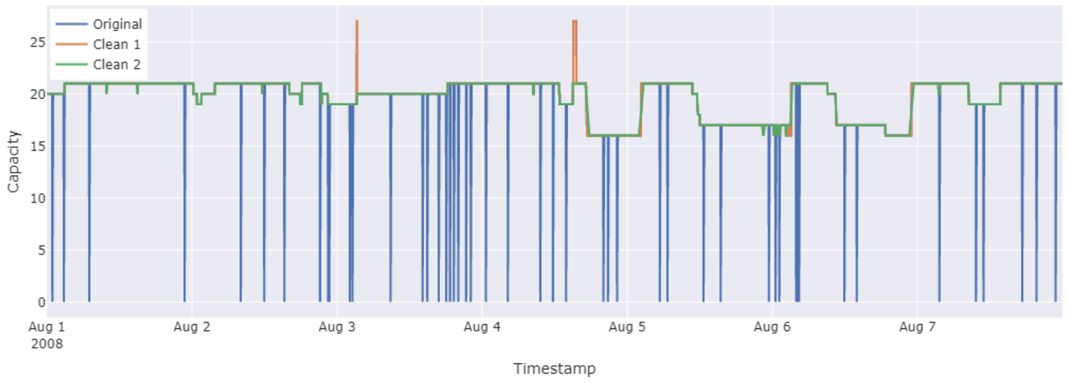

Figure 4.18: Total number of slots of station 1 before and after cleaning.

Moving to the time series of station 219, from Figure 4.19 it is possible to notice that in this case removing the records with total slots equal to 0 is not enough to remove all the low peaks. Indeed, almost all of them do not reach 0: nonetheless, the second operation successfully removes the remaining low peaks.

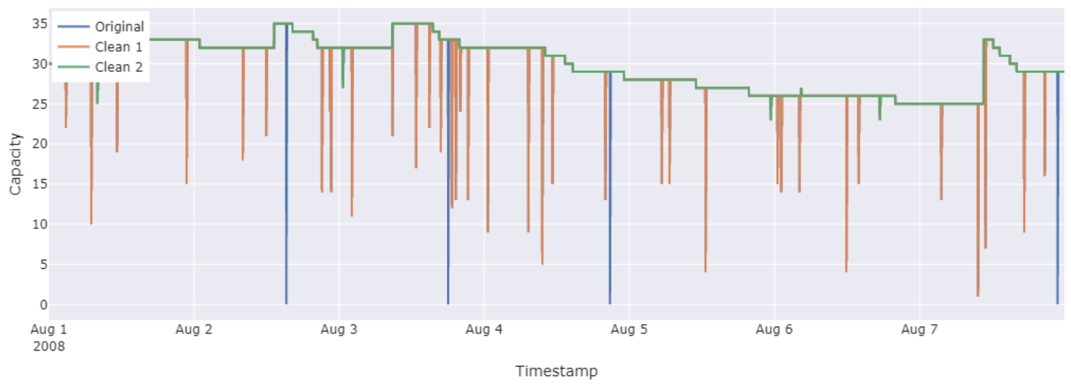

Figure 4.19: Total number of slots of station 219 before and after cleaning.

Chapter 5

Experimental results

In this chapter, the model proposed for extracting the *spatio-temporal sequences* will be tested on the dataset that has been analysed in Chapter 4 (*Dataset analysis*). To better understand how various factors affect the performance of the model, I have conducted multiple analysis in parallel with different parameters and compared the results.

5.1 Chapter overview

After having analysed the spatio-temporal dataset containing information about Barcelona's bike sharing system, it is now time to test the proposed model. As we saw in Chapter 3 (*The proposed model*), the overall procedure can be divided into three main steps: conversion of the time series to discrete sequences, transformation of the sequences to transactions, and extraction of the spatio-temporal rules. During each of these phases, there are crucial decisions that must be made and that influence the way *spatio-temporal sequences* are extracted. Therefore, this chapter has been divided into four major sections, each containing the description of how each of these factors has been addressed and the results that have been obtained.

As we saw in Chapter 4 (*Dataset analysis*), the original dataset contained some outliers that could have affected the performance of the model: to verify this hypothesis, the model will be applied to both the original[1] and cleaned versions of the dataset and the results will be compared.

[1]The problems regarding stations 221 and 284 and the month of July have been removed also from the original dataset. The difference with the cleaned version is only about the removal of the point outliers.

5.2 Events of interest

5.2.1 Overview

As we saw in Chapter 3 (*The proposed model*), the first phase of the process consists in the definition of the *events of interest*. Since these are the events between which the patterns will be extracted, it is important decide the goal of the analysis first. In this work, I would like to find out if there are any patterns regarding the *critical events* of the various bike stations. A station can be characterized by two types of critical event: "empty", when it does not contain any bike, or "full", when all of its slots are occupied by a bike. To better understand how the events of interest influence the model, two definitions have been analysed:

- *"All" critical events*. This definition includes all the critical events of the stations.
- *"First" critical events*. This definition includes only the critical events that are located at the beginning of a series of critical events.

The first possible intuition is that the patterns extracted from the first kind of critical events will be less significant than the ones extracted from the second kind: intuitively, the more a station remains in a critical status the less its critical events will impact on the critical situations of other stations. The difference between the two definitions is highlighted in Figure 5.1.

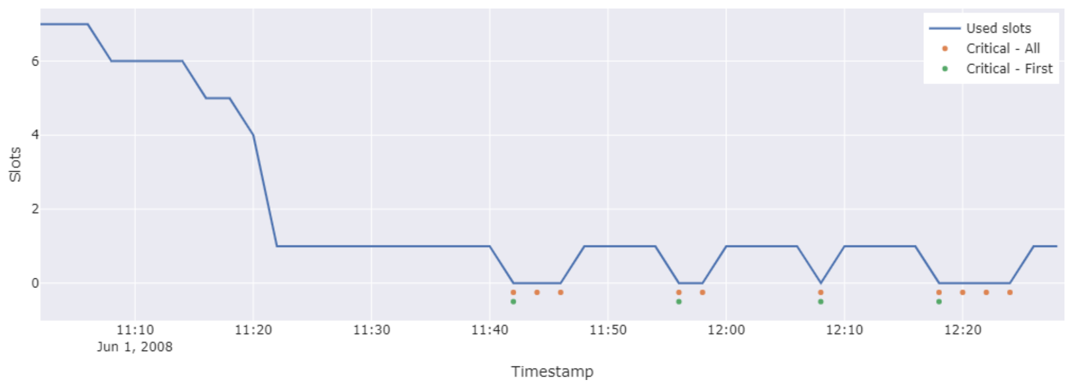

Figure 5.1: Difference between "all" and "first" critical events.

5.2.2 Extraction of critical events

As we saw in Chapter 4 (*Dataset analysis*), the variable *used_slots* indicates how many bikes are at the station and *free_slots* how many available slots the station has. Therefore, to extract the *critical events*, the records that must be filtered out are those where the number of used or free slots is 0. Since the values are numerical, they must also be converted to categorical: the "empty" events have been mapped to the string "E", while the "full" ones to the string "F". Table 5.1 contains a sample of records and the extracted critical events with both definitions.

	station	timestamp	used_slots	free_slots	"all" critical	"first" critical
0	1	2008-06-01 11:40:00	1	18	/	/
1	1	2008-06-01 11:42:00	0	19	E	E
2	1	2008-06-01 11:44:00	0	19	E	/
3	1	2008-06-01 11:46:00	0	19	E	/
4	1	2008-06-01 11:48:00	1	18	/	/

Table 5.1: Sample of records and the corresponding critical events.

5.2.3 Results

The first thing that can be noticed analysing the resulting datasets is that the number of extracted records depends on how the events of interest have been defined: as visible from Figure 5.2, the number of "all" the critical events is around 20% of all the original records (around 19 millions) and around 2% with the "first" critical events.

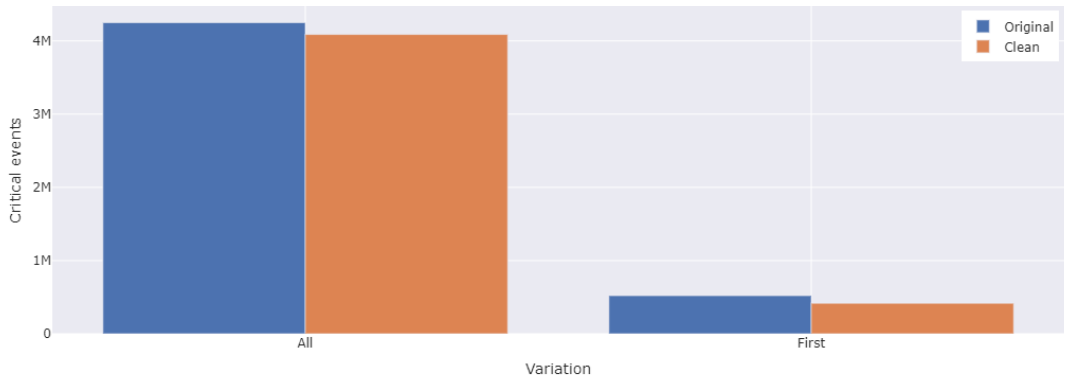

Figure 5.2: Number of critical events comparison.

The difference in size between the two datasets gives an important information about the stations: on average, they remain in a critical situation for around 10 consecutive timestamps, which, given the sampling rate of two minutes, is equivalent to 20 minutes. Figure 5.3 reports the average duration of a critical situation of each station.

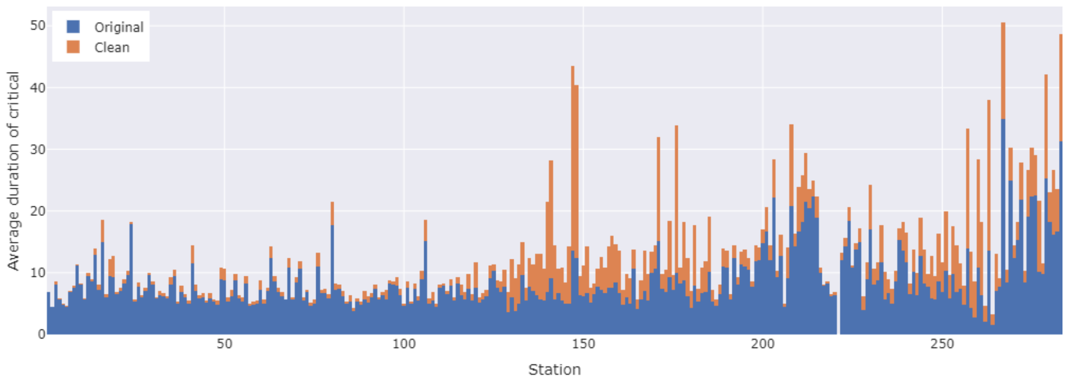

Figure 5.3: Average duration of a critical situations for each station.

Both of the two previous figures also highlight how cleaning the point outliers brings to an expected reduction in the number of extracted events: indeed, most of the point outliers corresponded to peaks in the data that could have resulted in critical events.

The second thing that can be noticed analysing the dataset is the distribution of the type of critical events: Figure 5.4 shows that the percentage of "empty" critical events is around 80% with both definitions. Even if cleaning the point outliers did not affect much this distribution, it is possible to notice that the percentage is a little bit lower with the cleaned datasets: this means that the point outliers where mostly related to "empty" critical events rather than "full" ones.

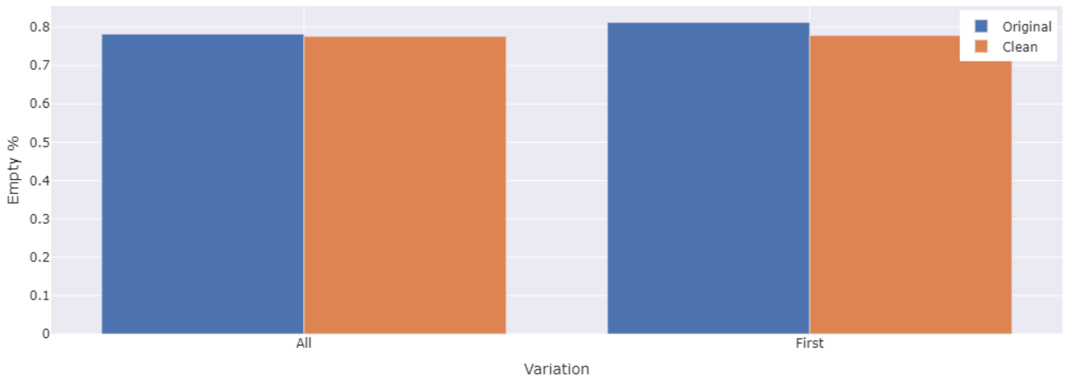

Figure 5.4: Percentage of "empty" critical events comparison.

The last thing that can be noticed regards the number of stations that reported critical events at the same time: Figure 5.5 shows that with "all" critical events the average number of simultaneously critical stations is between 50 and 100, while Figure 5.5 shows that the average number of simultaneously critical stations with the "first" critical events is below 50. Both figures highlight how cleaning the point outliers has heavily reduced these numbers.

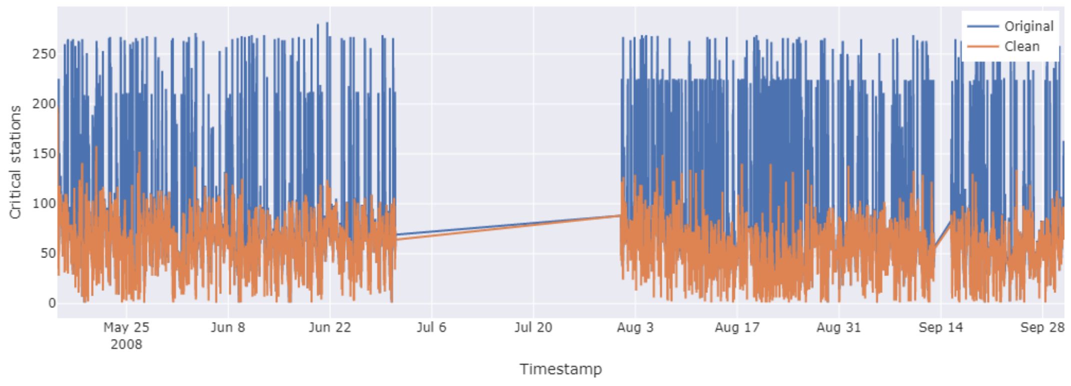

Figure 5.5: Number of simultaneously critical stations with "all" critical events.

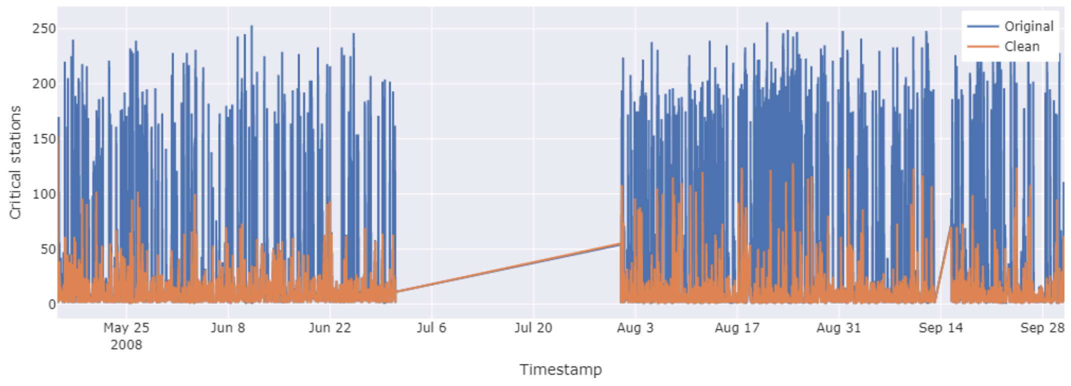

Figure 5.6: Number of simultaneously critical stations with "first" critical events.

5.3 Reduction of granularity

5.3.1 Overview

After the extraction of the events of interest, the second step consists in the reduction of the *temporal granularity* of the discrete sequences. To do so, two values must be defined: an initial timestamp and an interval length. As we saw in Chapter 3 (*The proposed model*), the granularity parameter g has a great impact on the extracted frequent patterns: with small interval sizes the extracted rules will describe short term patterns, while with larger interval sizes they will describe long term patterns. To analyse how this parameter influences the performance of model, I have decided to test a wide variety of values that could be appropriate for discovering patterns between the critical events. In this analysis, the interval lengths of choice had been $[4, 8, 12, 16, 20, 24, 32, 44, 60]$ and the initial timestamp that has been used is the first one available in the dataset, which is *2008-05-15 12:00:00*.

For convenience, the operation of merging together the events registered by different stations will be carried out together with the operation of reducing the granularity.

5.3.2 Alignment of critical events

After having decided the initial timestamp and the granularity of interest, each timestamp in the dataset must be mapped to the integer that represents the ID of the interval it belongs to. For example, Table 5.2 contains the aligned critical events of both stations 1 and 50 in the period of time that goes from *2008-06-01 11:40:00* to *2008-06-01 11:48:00* using an interval length of 4 minutes. After having aligned all the timestamps of the dataset to the ID of the interval they

	Station	**Interval ID**	**"all" critical**	**"first" critical**
0	1	6115	/	/
1	1	6115	E	E
2	1	6116	E	/
3	1	6116	E	/
4	1	6117	/	/
5	1	6117	/	/
6	50	6115	F	/
7	50	6115	/	/
8	50	6116	/	/
9	50	6116	/	/
10	50	6117	F	F
11	50	6117	F	/

Table 5.2: Sample of aligned critical events.

belong to, the critical events belonging to the same interval must be merged together. In Chapter 3 (*The proposed model*) we saw that the operation of merging together the events with the same interval ID can be split up into two steps: before the events of the same station are merged together and then the ones belonging to different stations. However, for convenience, these two operations can be carried out together without any problem. Of course, the problem of different events belonging to the same station in the same interval must still be addressed: in this work, I have decided to remove them. The result of merging the events of Table 5.2 is reported in Table 5.3. As you can see, after this operation a single sequence is obtained: each of its elements corresponds to the set of events that occurred during the same interval of time.

	Interval ID	"all" critical	"first" critical
0	6115	{1:E, 50:F}	{1:E}
1	6116	{1:E}	/
2	6117	{50:F}	{50:F}

Table 5.3: Stations 1 and 50 aligned events and merged.

5.3.3 Results

The first thing that can be noticed analysing the resulting datasets is that the number of generated intervals is inversely proportional to the interval length. Figure 5.7 shows also that when the interval length is smaller there is greater chance of having missing intervals, especially with the "first" critical events.

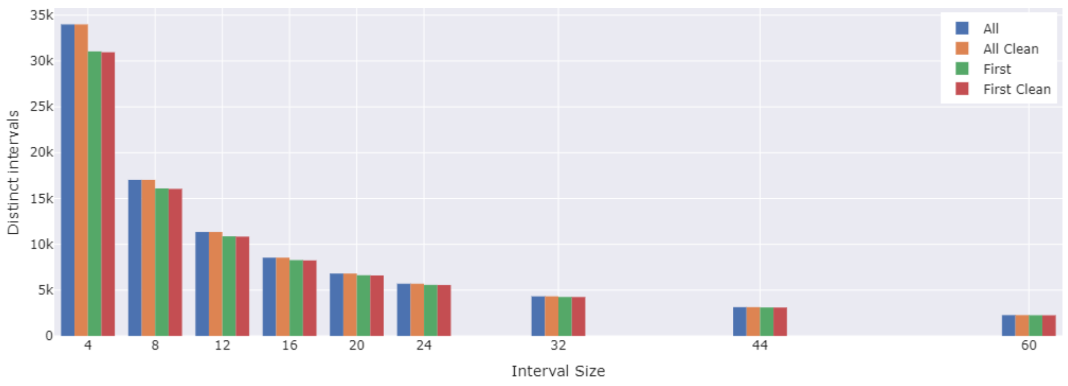

Figure 5.7: Number of distinct intervals for each interval size.

The second thing that can be noticed is that the higher the interval length, the higher the number of repeated events of the same station in the same interval. Figure 5.8 shows the average number of duplicates that each event has in an interval: of course, given the nature of the definitions of events of interest, the "first" critical events almost never have duplicates.

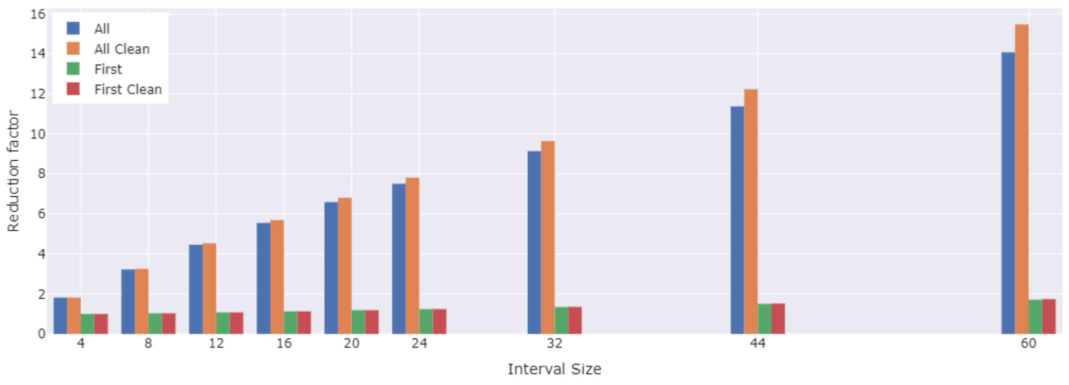

Figure 5.8: Average number of duplicates that each event has in an interval.

The last thing that can be noticed, which is also the most important one, is the average number of stations that registered different critical events in each interval. As you can see in Figure 5.9, the number increases when the interval length is bigger. Also, notice how the cleaning of point outliers definitely helped to reduce this number.

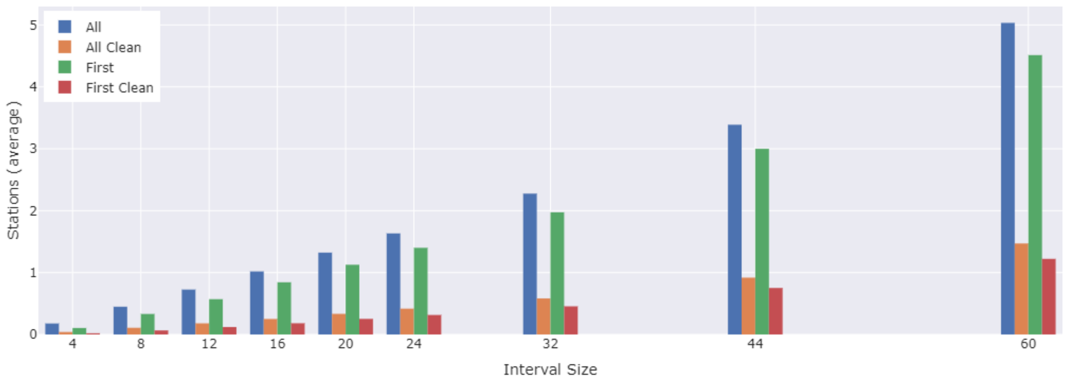

Figure 5.9: Average number of stations for each interval that registered both critical events.

5.4 Sequences generation

5.4.1 Overview

The last step before extracting the *spatio-temporal sequences* consists in the generation of the transactions. More precisely, the previously obtained sequence containing the events of interest must be transformed with a sliding window into a list of sequences that will be then converted to transactions. Since we are working with a lot of data, we cannot afford to test big window lengths because the problem of mining the *spatio-temporal sequences* would become too computationally expensive. Also, we don't want the window length to be too small because the extracted rules would be too simple. Moreover, since the length of the window simply controls the maximum temporal distance between the events in the rules that will be extracted, it is not necessary to analyse multiple values because we already know how it would affect the rules. Therefore, I have decided to test only a window length of 3.

5.4.2 Sliding window and transactions

As we saw in Chapter 3 (*The proposed model*), the sliding window operation consists in scanning the database to extract all the sets of events belonging to all the possible combinations of w consecutive intervals. Table 5.4 contains the sequences that are obtained after applying the sliding window operation of length 3 to the sequence of critical events belonging to stations 1 and 50 in the period of time that goes from *2008-06-01 11:40:00* to *2008-06-01 12:00:00.* If before there was only a sequence containing the critical of all the stations for each interval, now there are multiple sequences, one for each interval, that contain the sets of events that occurred in 3 consecutive intervals of time.

	Interval ID	"all" critical	"first" critical	Sequence "all"	Sequence "first"
0	6115	{1:E, 50:F}	{1:E}	{1:E, 50:F}, {1:E}, {50:F}	{1:E}, {}, {50:F}
1	6116	{1:E}	/	{1:E}, {50:F}, {50:F}	{}, {50:F}, {}
3	6117	{50:F}	{50:F}	{50:F}, {50:F}, {1:E}	{50:F}, {}, {1:E}
4	6118	{50:F}	/	{50:F}, {1:E}, {}	{}, {1:E}, {}
5	6119	{1:E}	{1:E}	{1:E}, {}, {}	{1:E}, {}, , {}
6	6120	/	/	{}, {}, {}	{}, {}, {}

Table 5.4: Sliding window on a sample of critical events recorded by station 1 and 50.

Since I'm planning to extract the frequent *spatio-temporal sequences* with FP-Growth, it is also necessary to convert the sequences to transactions. Table 5.5 shows how the previously generated sequences are converted to transactions.

	Transaction ID	Events "all"	Events "first"
0	6115	{0:1:E, 0:50:F, 1:1:E, 2:50:F}	{0:1:E, 2:50:F}
1	6116	{0:1:E, 1:50:F, 2:50:F}	{1:50:F}
3	6117	{0:50:F, 1:50:F, 2:1:E}	{0:50:F, 2:1:E}
4	6118	{0:50:F, 1:1:E}	{1:1:E}
5	6119	{0:1:E}	{0:1:E}
6	6120	{}	{}

Table 5.5: Conversion of the sequences to transactions.

5.4.3 Results

Of course, the number of generated transactions corresponds to the number of intervals that the dataset contains. What can be noticed from the resulting datasets is that the density of each transaction increases when the interval length increases. Moreover, Figure 5.10 shows that with smaller interval lengths the transactions containing the "first" critical events are much less dense than the ones with "all" critical events, while with bigger interval lengths the transactions tend to be equally dense in all four datasets.

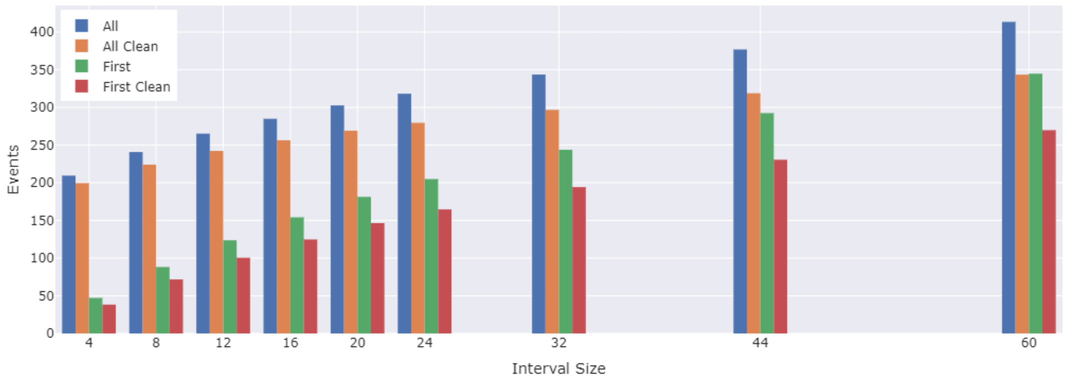

Figure 5.10: Number of events per transactions.

5.5 Spatio-temporal sequences

5.5.1 Overview

The last phase of the process consists in the extraction of frequent itemsets and rules from the list of transactions. As we saw in Chapter 2 (*Background and related work*), the extracted rules can be interpreted as *spatio-temporal sequences* of events. During this phase, two parameters must be defined: the minimum support and the minimum confidence. Since the datasets of transactions are all characterized by different densities and sizes, it is necessary to use different minimum supports for each one of them, otherwise a common low value would make some of the extractions too computationally expensive. The values that have been used are summarized in Table 5.6: I have pushed each one of them to the minimum possible value before the algorithm started to take too much time to complete. Instead, a single value has been used as minimum confidence, which is 0.5.

	Interval Length	Minimum Support - "All"	Minimum Support - "First"
0	4	0.36	0.08
1	8	0.32	0.10
2	12	0.34	0.10
3	16	0.34	0.12
4	20	0.36	0.14
5	24	0.36	0.16
6	32	0.40	0.20
7	44	0.44	0.24
8	60	0.48	0.34

Table 5.6: FP-Growth minimum supports for each dataset.

As you can see from the table, the minimum support is bigger when the interval length increases: this is because the number of transactions is smaller and therefore the itemsets tend to be more frequent. Also, the minimum supports for the "first" critical events are smaller because the transactions tend to be less dense than the ones with "all" critical events.

5.5.2 FP-Growth

As we saw in Chapter 3 (*The proposed model*), after having extracted the frequent itemsets, it is necessary to filter out the ones that do not contain any event with a relative index of 0. Also, all the rules that have events in the antecedent part with a relative index greater than the one in the consequent part must be filtered out.

To improve the quality of the extracted rules, I have decided to remove also those rules where the critical events are related to stations that are too far from each other. More precisely, I have imposed a minimum distance of 1 Km between the stations.

5.5.3 Results

Figure 5.11 shows that the average support of the 100 most frequent itemsets tends to be higher when increasing the interval length. However, as we saw in Chapter 3 (*The proposed model*), when the interval length is too big, there is the risk of losing the concept of relative temporal distance between the events. Therefore, it is essential to find a good *trade-off* between the high support of the itemsets and the concept of temporal distance between the events.

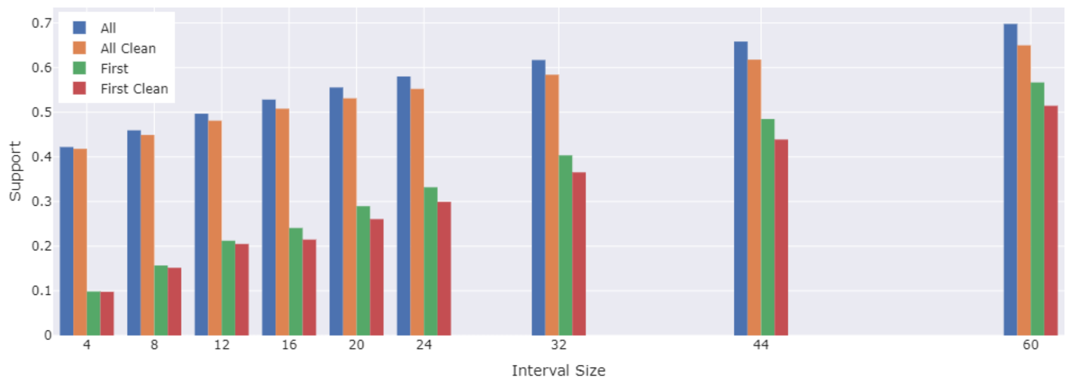

Figure 5.11: Average support of the 100 most frequent itemsets.

Moreover, Figure 5.11 shows also that the itemsets with "all" critical events tend to be more frequent than the ones with the "first" critical events, especially with smaller interval lengths. However, as we had already guessed, the itemsets with "all" critical events tend to be less significant: as visible from Figure 5.12, the average fraction between the number of unique stations and the number of events of the itemsets is much lower with the ones containing "all" the critical events than the ones containing the "first" critical events.

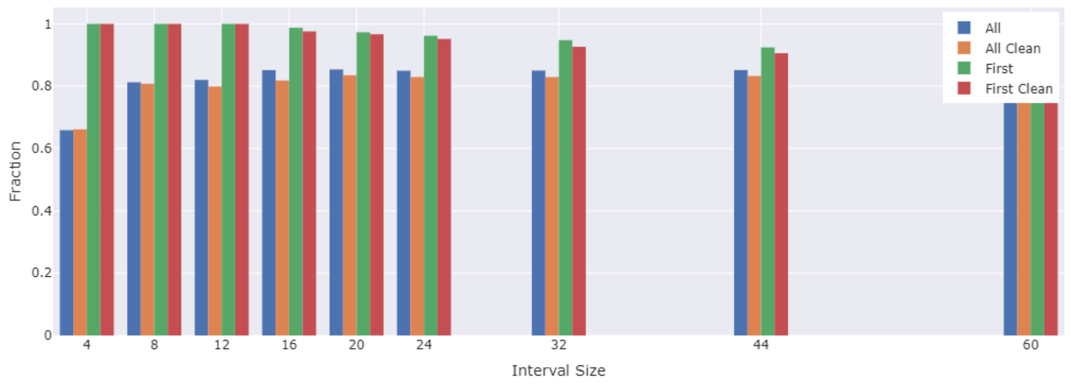

Figure 5.12: Average fraction between the number of unique stations and the number of events of the 100 most frequent itemsets.

Moving to the analysis of the rules, Figure 5.13 shows that when the interval length is too small, some datasets do not contain any rule with a confidence higher than 0.5. Also, it is evident how the definition that includes "all" the critical events has generated rules with higher confidences on average. However, the considerations made for the itemsets also apply to the rules.

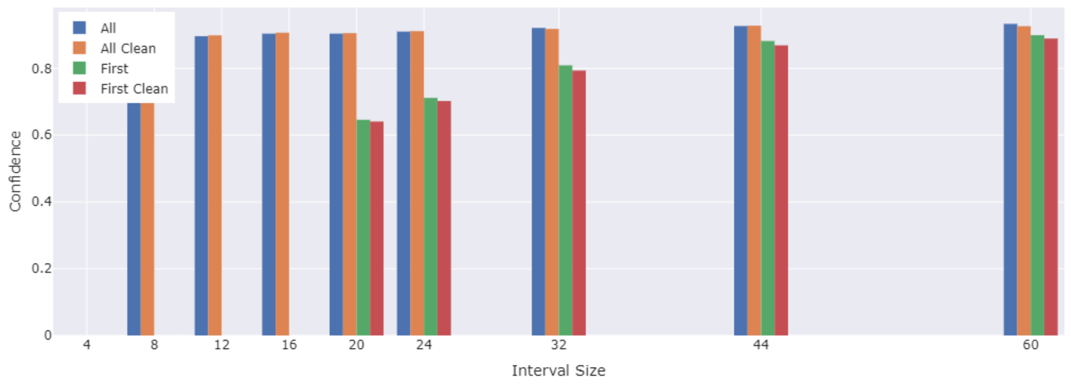

Figure 5.13: Average confidence of the 100 rules with the highest confidence.

An example of rule obtained with the "first" critical events and using an interval length of 20 minutes is: $\{0:72:E, 1:217:E\} \Rightarrow \{2:219:E\}$. The confidence of the rule is 0.68 and the support of the corresponding frequent itemset is 0.15. Therefore, the rule can be interpreted in the following way: *"Whenever station 72 becomes empty and after around 20 minutes also station 217 becomes empty, there is a 67% chance the after other 20 minutes station 219 will also become empty."*. The support of the itemset tells how frequently this situation happens.

Chapter 6

Conclusions

The goal of this work was to introduce a new method for extracting *sequences of events* from spatio-temporal datasets. Considering the results that have been achieved with the bike sharing dataset it is possible to say the new procedure works very well. Also, I believe that this new method can be used in a broad range of applications where the collected datasets are spatio-temporal.

Even if the objective was not to outperform the efficiency of any of the existing methods, I think that this new procedure scales pretty well with large quantities of data thanks to the usage of Spark's parallel implementation of FP-Growth. Nonetheless, there are for sure some optimizations that could be made to improve the efficiency of the algorithm and that could be addressed in future developments:

- *"Online" filtering.* This improvement consists in anticipating the filtering made to the itemsets and rules inside the frequent pattern mining algorithm: in this way it may be possible to prune the search space and therefore improve the efficiency of the algorithm.
- *Divide-and-conquer approach.* In applications where the sequences of events often occur in locations that are near one to each other, it could be possible to perform the analysis by groups of stations. This improvement consists in partitioning the spatial locations into smaller groups and then analyse them in parallel.

Bibliography

[1] Agrawal R., Imielinski T., Swami A. (1993). *Mining association rules between sets of items in large databases.* In: Proceedings of the 1993 ACM-SIGMOD international conference on management of data (SIGMOD'93), Washington, DC, pp 207-216

[2] Agrawal R., Srikant R. (1994). *Fast algorithms for mining association rules.* In: Proceedings of the 1994 international conference on very large data bases (VLDB'94), Santiago, Chile, pp 487-499

[3] Agrawal R., Srikant R. (1995). *Mining Sequential Patterns.* In: Proceedings of the 1995 international conference on data engineering (ICDE'95), Taipei, Taiwan, pp 3-14

[4] Srikant R, Agrawal R. (1996). *Mining sequential patterns: generalizations and performance improvements.* In: Proceedings of the 1996 international conference on extending database technology (EDBT'96), Avignon, France, pp 3-17

[5] Han J., Pei J., Yin Y. (2000). *Mining frequent patterns without candidate generation.* In: Proceedings of the 2000 ACM-SIGMOD international conference on management of data (SIGMOD'00), Dallas, TX, pp 1-12

[6] Zaki M. (2001). *SPADE: an efficient algorithm for mining frequent sequences.* In: Machine Learning, vol 42 pp 31-60

[7] Pei J., Han J., Mortazavi-Asl B., Wang J., Pinto H., Chen Q., Dayal U., Hsu M-C. (2004). *Mining sequential patterns by pattern-growth: the PrefixSpan approach.* In: IEEE transactions on knowledge and data engineering, vol 16 pp 1424-1440

[8] Li H., Wang Y., Zhang D., Zhang M., Chang E. (2008). *PFP: Parallel FP-Growth for Query Recommendation.* In: Proceedings of the 2008 ACM conference on recommender systems (RecSys'08), Lausanne, Switzerland, pp 107-114