Politecnico di Milano

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING
Master of Science in Computer Science and Engineering

MASTER THESIS

A General Approach to Value Identification of Large Scale Geospatial Data

Supervisor
Prof. Marco Brambilla

Co-Supervisor
Prof. Ernestina Menasalvas Ruiz
Prof. Alessandro Bogliolo

Candidate
Gioele Bigini
Matr.898741

Academic Year 2018–2019

Virtutibus Itur Ad Astra
9th book of the Aeneid

Contents

List of Figures

List of Tables

Listings

Sommario

Oggi la maggior parte dei dispositivi mobili registra continuamente informazioni come dati geo-spaziali. Grazie all'avvento dell'Internet delle Cose, essi stanno gradualmente dominando il settore dei big data. Un'applicazione che si occupa di collezionare questo genere di dati è SmartRoadSense, un app mobile che è stata sviluppata con lo scopo di collezionare dati geo-spaziali relativamente alle strade percorse mentre si sta guidando un auto.

Uno dei principali problemi relativamente ai dati geo-spaziali è la capacità di fare analytics in maniera efficace su di essi. Due sono le cause principali: la distribuzione nello spazio (dal momento che possono essere registrati in qualsiasi punto del globo) e la distribuzione nel tempo (essi perdono progressivamente valore con il passare del tempo).

Pertanto, se l'aumento di volume rappresenta una sfida per le operazioni di memorizzazione, gestione e trattamento (perlopiù aspetti tecnici), dall'altro lato l'analisi, visualizzazione e veracità rappresentano una sfida ancora più grande perché non c'è alcun modo di poter quantificare lo sforzo necessario per comprenderne il valore.

Lo scopo di questo lavoro è proporre un approccio generale che tenti di aiutare l'operatore nel comprendere la distribuzione dei dati geo-spaziali (senza limiti di forma) nello spazio e nel tempo a partire da metriche personalizzate con l'obbiettivo implicito rivelarne il valore.

Abstract

Nowadays, most mobile devices continuously record information as geospatial data. Thanks to the advent of the Internet of Things, they are gradually dominating the big data sector. An application that collects this kind of data is SmartRoadSense, a mobile app that has been developed with the aim of collecting geospatial data relative to the roads travelled while driving a car.

One of the main problems with geospatial data is the ability to effectively perform analytics on them. There are two main causes: the distribution in space (since they can be recorded at any point on the earth) and the distribution over time (they gradually lose value over time).

Therefore, if the increase in volume represents a challenge for storing, managing and processing operations (mostly technical aspects), on the other hand, the analysis, visualisation and veracity represent an even greater challenge because there is no way to be able to quantify the effort required to understand its value.

The purpose of this work is to propose a general approach that attempts to help the operator understand the distribution of geospatial data (without shape limits) in space and time starting from customised metrics with the implicit goal of revealing its value.

Chapter 1

Introduction

1.1 Context

During these years, the devices involved in the Internet of Things have immensely increased in number and in the amount of information they are able to record. Through cameras, GPS, accelerometers, Wi-Fi and Bluetooth antennas, NFC and other sensors, they are able to collect a huge amount of data, generally referred as Big Data.

When talking about Big Data we often refer to data sets so extensive in terms of volume, speed and variety that It is no longer possible to deal with them through conventional tools and they often require specific technologies and analytical methods for the extraction of value.

Today, geospatial data is everywhere. It is generated continuously while driving, while using social networks or while playing with mobile devices. This data often hide interesting implicit information useful for several reason: i.e. user profiling.

In this context entered Crowd4Roads, an Horizon 2020 project that combines trip sharing and crowd sensing initiatives to harness collective intelligence to contribute to the solution of the sustainability issues of road passenger transport, by increasing the car occupancy rate and by engaging drivers and passengers in road monitoring through SmartRoadSense [14], a mobile application developed by the IT unit of the University of Urbino that uses smartphone's accelerometer and GPS sensor to detect and classify irregularities of the road surface while driving.

1.2 Problem Statement

Many data collection platforms, including SmartRoadSense, collect information from mobile devices such as smartphones and tablets with cross-platform applications. The devices available on the market are obviously not all the same, presenting several differences between them in terms of hardware used and software implemented. This is reflected in the ability to collect data more or less of the same quality. Moreover, the collected data are not necessarily useful if in a good

amount, or rather there is not a univocal relationship between quantity and value for which the greater the data and the greater the value they represent because this relationship is strongly dependent on the problem considered.

For example, SmartRoadSense has been running for about 3 years and has collected millions of geospatial data but the amount available is sparse on time and space. Just to be more specific, considering all the data collected about a specific road, having a large amount of data over that road does not imply that this is equally distributed along the road nor that It has been collected more or less in the same period or recent enough to suggest useful information.

In general, the greater the data and the greater is the probability that a data set contains relevant insights, which makes this possibility not certain. When using geospatial data sets the problems discussed earlier lowers a lot the probability (even more if the data has been collected on a broad spatial basis). These problems are the ones that the big companies all over the world have dragged in a short time and the goal of this work is to try to give a general approach to face with them. They can be summarised as the problems of:

- **Data Volume:** the volume affects the ability to deliver results;
- **Sparsity:** the data is sparse on a spatial and temporal basis which could affect the ability to generate value.

1.3 Proposed Solution

Since the data collected (in SmartRoadSense) as raw data shown obvious deficiencies related to their distribution over time and space, this suggested to focus on solving the problem through a strategy aimed at generating results that can be crossed together in order to find out which records of the data set can be able to generate the value needed.

In order to reach this goal several technologies, methods and techniques have been involved, mainly massively parallel software like Spark, time series techniques, cluster implementations as well as some simple analytical linear model.

The work is based on the challenge of generating a crossed data set that sums up different metrics. Specifically for this work the metrics are:

- **Data Recency:** how much the collected data is recent with respect to the present;
- **Usage Trends:** which is the usage trend of the application along the territories;
- **Data Quantity:** the total amount of data collected over time.

The metrics have been designed to be addressed on a territorial basis, basically the municipalities. As a reference, the Italian peninsula has been considered since It is the pilot of the project.

1.4 Structure of the Thesis

The dissertation is structured as follows:

Chapter 1: Introduction - A brief introduction

Chapter 2: Background - The state of the art and relevant technologies used

Chapter 3: Related Works - Other works from which the problem emerged

Chapter 4: The General Approach for Value Identification - The proposed solution

Chapter 5: Implementation Experience - The implementation of the approach

Chapter 6: Experiments and Discussion - Experiments involved and discussion of what has been obtained

Chapter 7: Conclusions - The final conclusions and future works

Chapter 2

Background

2.1 Internet of Things

The Internet of Things (IoT) refers to the concept that different devices (homogeneous and/or heterogeneous) are able to communicate and interact through the availability of a wireless connection, on the internet.

Initially, the traditional IoT field was relegated to sensor networks, control and automation systems. Today, the IoT means more devices capturing information useful for several reason as: taking strategic decisions or training algorithms.

Thanks to the IoT, a new number of technologies and techniques have been cleared in the fields of real-time analysis and machine learning. However, the IoT is criticised for the big impact it implies on people's lives since the most traditional applications are intrusive, enabling monitoring and control at the expense of privacy and security. And It is not clear the impact on health that some new technologies supporting the IoT have, like 5G communication.

2.1.1 A Bit of History

The derivation of the term "Internet of Things" is rather tight. The concept of a network of devices was first discussed in 1982 when a modified Coke vending machine at Carnegie Mellon University became the first Internet-connected appliance, capable of reporting its inventory and whether the loaded drinks were still cold or not. Later, Mark Weiser's published the article "The Computer of the 21st Century" [1] that is more like the IoT vision of today. It was the 1991.

According to Wikipedia, the term Internet of Things was probably coined for the first time by Kevin Ashton of Procter & Gamble in 1999, later MIT's Auto-ID Center [2]. Then in January 2002, Kary Främling and his team at Helsinki University of Technology more closely matches the modern idea: an information system infrastructure for implementing smart, connected objects. [3]

2.1.2 Challenges

In recent years, the IoT experienced an unprecedented explosive growth with millions of connected devices. The reason why the IoT has taken over is in the opportunities It generated once It abandoned the traditional track. Thanks to technologies improvements into the telecommunications sector, nowadays most of all the objects are smart. Today the Internet of Things is synonymous of:

- Efficiency improvements
- Economic benefits
- Human efforts reduction

But, more devices means more implications. The amount of data available is a lot bigger than before and this impact on the response time required for an IoT device to process some request. This big increase in data volume brought several challenges in the data transmission field. For this reason the services can be generally provided at three different levels [4]:

- IoT Devices
- Edge/Fog Nodes
- Cloud Computing

These three service levels substantially differentiate on how much the system is time dependent. For example:

- If the IoT device must recognise objects in real time then the response time should be minimum to receive the necessary prediction as soon as possible. This decision, which must be very fast, would not be possible if using Cloud Computing only and so, all the power of the IoT system should be inside the device.
- On the other hand, a platform for extracting value from data would not require responsiveness but a very large amount of data, not sustainable from a regular device. Then, the Cloud Computing would prove extremely effective.
- There are then middle cases: Edge or Fog Nodes. In the future, the IoT could be so advanced as to provide autonomous behaviours such as the ability to detect changes in the environment. This goal requires at the same time important characteristics in the space to manage a multitude always greater of devices. With millions of devices in the internet space, Fog Computing will represent a viable alternative for managing the ever-increasing data flow on the internet. The Edge devices can be used to analyse and process data, providing real-time scalability.

2.2 Big Data

The field of Big Data tries to find ways to analyse, extract information and deal with data sets that are too large or complex to be dealt with by traditional data-processing application software [29].

Actually, the field of Big Data tends to refer to the use of predictive analytics, user behaviour analytics, or certain other advanced data analytics methods that extract value from data and seldom to a particular size of data set. Big Data analytics is done to find new correlations, for example to spot business trends, prevent diseases, combat crime and so on [21].

2.2.1 Data Sets Growth

Data sets grow rapidly because they are increasingly gathered by cheap and numerous information sensing devices. The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s.

In this context, relational database management systems, desktop statistics and software packages used to visualise data often have difficulty handling Big Data. To manage and analyse this data sets may require massively parallel software running on several machines.

What qualifies a data set to be "Big" varies depending on the capabilities of the users and available tools, considering that expanding capabilities makes Big Data less big and then a moving target. For example some organisations facing hundreds of gigabytes of data for the first time may trigger a need to reconsider data management options. For others, It may take tens or hundreds of terabytes before data size becomes a significant problem to consider.

So, to identify a Big Data, three main characteristics should be considered along with the concept of *veracity* [12][5]:

1. **Volume:** The quantity of generated and stored data. The size of the data determines the value and potential insights.

2. **Variety:** The type and nature of the data. This helps people who analyse it to effectively use the resulting insights. Big data draws from text, images, audio, video and It completes missing pieces through data fusion.

3. **Velocity:** The speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. The data is often available in real-time. Velocity factors are: the frequency of generation and the frequency of handling, recording and publishing.

4. **Veracity:** It is the extended definition for big data, which refers to the data quality and value. The data quality of captured data can vary greatly, affecting the accurate analysis. Data must be processed with advanced tools

to reveal meaningful information.

2.2.2 Challenges

A high amount of data offer greater statistical power, higher if It is not too complex (high number of attributes or columns), because complexity may lead to a higher false discovery rate [6]. However, this suggest that Big data includes several challenges:

- **Data Collection:** The heterogeneity of devices could generate data difficult to interpret;
- **Data Storage:** Much of that data is unstructured, meaning that it doesn't reside in a database;
- **Data Analysis:** Documents, photos, audio, videos and other unstructured data can be difficult to search and analyse;
- **Data Security:** Big Data stores can be attractive targets for hackers;
- **Data Privacy:** Each collected data should be handled accordingly to the regulations of a given location.

2.3 Mobile Crowdsensing

Crowd-sensing, sometimes referred to as Mobile Crowd-sensing, is a technique where a large group of individuals having mobile devices capable of sensing and computing (such as smartphones, tablet computers, wearables) collectively share data and extract information to measure, map, analyse, predict any processes of common interest.

The term "Mobile Crowdsensing" was coined by Raghu Ganti, Fan Ye, and Hui Lei in 2011 [7] and based on the type of involvement of the user, Mobile Crowdsensing can be classified into two types:

- **Participatory Crowd-sensing:** where the users voluntarily participate in contributing information [8];
- **Opportunistic Crowd-sensing:** where the data is sensed, collected and shared automatically without user intervention and in some cases, even without the user's explicit knowledge [9].

2.3.1 Data Collection

Taking advantage of the ubiquitous presence of powerful mobile computing devices in the recent years (especially smartphones), Crowd-sensing has become an appealing method to businesses that wish to collect data without making large-scale investments. Most smartphones can sense light, noise, location, movement, and so on. The embedded sensors can collect vast quantities of data useful in a variety of ways. Numerous technology companies are using Crowd-sensing to offer services based on the big data collected, some of the most notable examples are Facebook, Google and Uber.

There are three main strategies for collecting this data [10]:

- **Manual Collection:** The user of a device collects data manually. This can include taking pictures or using smartphone applications;
- **Hybrid Collection:** The user can manually control data collection, but some data can be collected automatically, such as when a user opens an application;
- **Automated Collection:** Data sensing is triggered by a particular context that has been predefined, such as a device that reacts to a second nearest device.

2.3.2 Challenges

Collecting data from mobile devices involves several challenges. All the devices are limited in terms of resources and privacy-security represent a major issue.

Limited resources are most of the time related to energy, bandwidth and computation power. For positioning, using sensors like GPS drains batteries and if we like to avoid using it through Wi-Fi and GSM, the results are generally less accurate. The same happens if the sensing process does not care about redundancies: eliminating redundant data can reduce energy and bandwidth costs or, restricting data sensing when quality is unlikely to be high, can improve energy consumption.

The data collected through Mobile Crowd-sensing can be sensitive to individuals, revealing personal information such as home and work locations and the routes used when commuting between the two. Ensuring the privacy and security of personal information collected through Mobile Crowd-sensing is therefore important. Mobile Crowd-sensing can use three main methods to protect privacy [11]:

- **Anonymisation:** It removes identifying information from the data before It is sent to a third party. This method does not prevent inferences being made based on details that remain in the data;
- **Secure Multiparty Computation:** It transforms data using cryptographic techniques. This method is not scalable and requires the generation and maintenance of multiple keys, which in return requires more energy;

- **Data Perturbation:** It adds noise to sensor data before sharing it with a community. Noise can be added to data without compromising the accuracy of the data.

2.3.3 Architecture

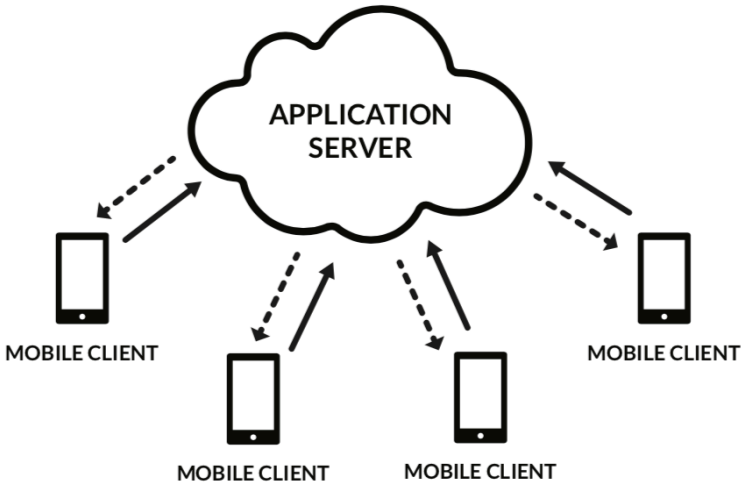

Figure 2.1: General Mobile Crowd Sensing Architecture

As coordinated distributed software platforms, Mobile Crowd-sensing systems are usually composed of a central in-cloud application server and several mobile clients. The central server is responsible for managing all the centralised phases of the whole sensing and processing procedure.

It implicitly or explicitly assigns the sensing tasks to the users and then receives data collected by participants. The sensing task is performed in a decentralised manner through mobile clients of volunteers. The software client can collect data directly or with the help of the user. For a more comprehensive analysis of typical MCS architectures, see the work of Louta et al. [13].

2.4 Libraries and Tools

The project as described needs suitable tools and procedures for the analysis and extraction of information. The database contains millions of records and this number is going to grow exponentially in the future.

2.4.1 Jupyter and Python

The main tools used are Jupyter Notebook (version 4.4.0) and Python (version 2.7 in this work). The Python version is fundamental for the PySpark environment used locally on the machine when experimenting.

2.4.2 Apache Spark

Spark is a unified analytics engine for large-scale data processing created by Apache to distribute the workload on clusters. It provides high-level APIs with which to develop the applications to be submitted to Its cluster manager that takes care about the processing involved.

Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in the main program (called the driver program). Specifically, to run on a cluster, the SparkContext can connect to several types of cluster managers (either Sparks own standalone cluster manager: Mesos or YARN), which allocate resources across applications.

Once connected:

1. Spark acquires executors on nodes in the cluster, which are processes that run computations and store data for the application;
2. It sends the application code (Python for this work) to the executors;
3. The SparkContext sends tasks to the executors to run.

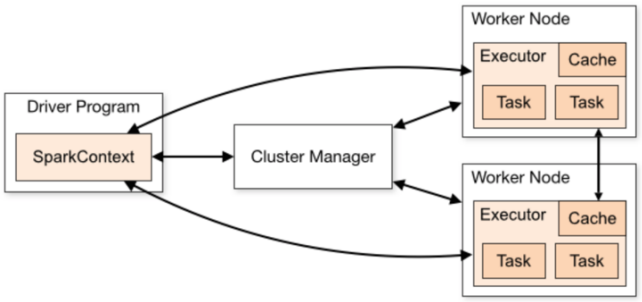

Figure 2.2: Spark Job Flow

There are several things to note:

1. Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads.
   - **PRO:** The applications are isolated each other: on both the scheduling side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs).
   - **CONS:** Data cannot be shared across different Spark applications (instances of SparkContext) without writing it to an external storage system.
2. Spark is agnostic to the underlying cluster manager. As long as it can acquire executor processes, and these communicate with each other, It is relatively easy to run It even on a cluster manager that also supports other applications (e.g. Mesos/YARN).

3. The driver program must listen for and accept incoming connections from its executors throughout its lifetime. As such, the driver program must be network addressable from the worker nodes. Because the driver schedules tasks on the cluster, It should be run close to the worker nodes, preferably on the same local area network.

Resilient Distributed Dataset

Spark revolves around the concept of a Resilient Distributed Dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. Parallel collections are created by calling *sc.parallelize()* method on an existing collection in the driver program. The elements of the collection are copied to form a distributed dataset that can be operated on in parallel.

One important parameter for parallel collections is the number of partitions to cut the dataset into. Spark will run one task for each partition of the cluster. Typically, Spark tries to set the number of partitions automatically based on the cluster properties. However, It is possible to manually pass the number of partitions as a second parameter to the *parallelize()* method.

RDDs support two types of operations:

1. **Transformations:** which create a new RDD from an existing one;
2. **Actions:** which return a value to the driver program after running all the transformations on the RDD.

For example, map is a transformation that passes each RDD element through a function and returns a new RDD representing the results. On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program.

All transformations in Spark are lazy, in that they do not compute their results right away. The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. For example, we can realise that a RDD created through map will be used in a reduce and return only the result of the reduce to the driver, rather than the larger mapped RDD.

By default, each transformed RDD may be recomputed each time an action is performed. However, persisting the RDD in memory using *persist()* or *cache()* method allow to keep the elements around on the cluster for a much faster access.

2.4.3 Google Cloud Platform

Massively parallel processing using tools such as Spark can be performed locally for small data sets and Its local implementation can be used to perform optimisa-

tion operations before proceeding to launch a specific algorithm on a cluster.

If the operations to be performed require clusters with non-negligible resources and not negligible execution times, then locally studying an algorithm should be exploited in this sense.

Google offers a service called DataProc that allows the creation of tailor-made clusters on demand. The service is obviously not free but excellent in all cases It is necessary to perform tasks with tools such as Spark. However, It is very cheap and has been used to pre-process the SmartRoadSense data set.

2.4.4 QGIS

Quantum GIS (QGIS) is an open source GIS desktop application, very similar in user interface and functions to equivalent commercial GIS packages.

The use of QGIS is related to the generation of borders (namely in shapefiles) used to apply the geo-spatial queries to the areas on which the analysis has been performed.

Chapter 3

Related Works

3.1 Mobile Crowdsensing Applications

3.1.1 Data Collection and Processing Workflow

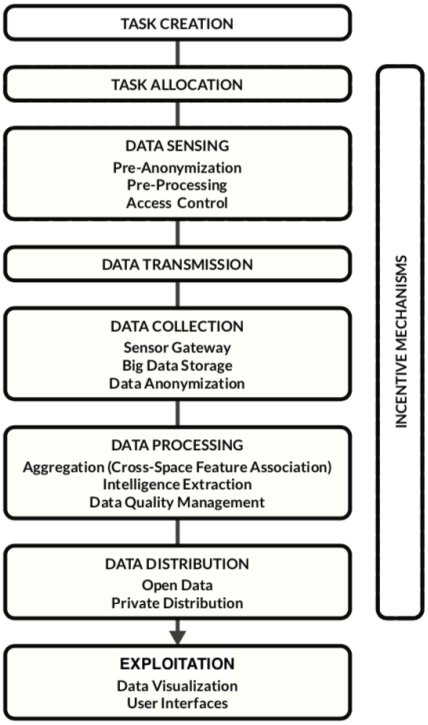

Figure 3.1: MCS Application Workflow

The Mobile Crowd-sensing applications follow a general pattern before coming on the market. There are several definition like the one below, that It is considered the most accepted one with some original change. The workflow is based on the proposal of Dr. Saverio Delpriori [15], shown in Figure 3.1.

Task Creation

In this first phase, the central entity creates specific tasks and provides a detailed description of the required actions. The task creation can be even started by users, the same ones who will consume the data collected using the Mobile Crowd-sensing application.

Depending on the platform used, the description could be either in natural language or domain specific language that software clients are able to understand and present to volunteers. In some cases, the task creation is implicit in the platform structure. Volunteers who join the application are automatically tasked with a defined sensing operation [14].

Task Allocation

The central entity can analyse the sensing task and assign it to specific participants (or nodes of the sensing network) possibly trying to respect given constraints: ensuring area coverage, minimising the task completion estimated time, maintaining the number of volunteers involved under a given threshold, ensuring a minimal average trust value among the selected participants, and so on.

Another approach is to notify all clients that a new task is available and let them choose whether to take part in the sensing task or not. Depending on the motivation incentive systems utilised, some systems also allow approaches like auction based assignments.

Data Sensing

Involves both information sensed from mobile devices and user contributed data from mobile internet applications. MCS applications usually have to tackle security and privacy issues, thus providing users with automated or semi-automated mechanisms determining what kind of information they want to publish and whom to share them with is fundamental.

Many MCS systems resort to access control mechanisms and pre-anonymisation techniques. In order to reduce transmission costs and size, data is often pre-processed on board of the user device. Finding the appropriate trade-off between the amount of processing to be done onboard of smartphones and in cloud after the data has been transmitted is a crucial parameter for a MCS application.

Data Collection

In this phase, data is received by mobile clients and stored into appropriate memory supports. Privacy-preserving techniques are applied to ensure security and to avoid that malicious users acquire collected data and can track them back to users. The sensor gateway module provides a standard approach - usually implementing common web services technologies - to enable data collection from

crowd-sensed sources supplying a unified interface.

MCS applications may collect a vast amount of heterogeneous data and big data storage systems are usually employed. Big data techniques simplify the collection of large-scale and complex data like noise level measured across an urban area. Sensing tools used by participants to evaluate the phenomenon at stance typically varies a lot, leading to significant differences in the accuracy of crowd-sensed data. Therefore, data is commonly transformed and unified before being stored and passed to the next phase in order to boost further processing.

Data Processing

Aims to derive high-level intelligence from raw data received. Using logic-based inference rules and machine-learning techniques this step focuses on discovering frequent data patterns in order to extract crowd-intelligence starting from data sensed by mobile users and user-contributed data from other mobile internet applications mixed together.

The first step of the data processing is the data aggregation phase, in that, raw data from different users, time and space are combined on different dimensions and associated with reference known features (e.g., map-matching [16]).

Then further data processing techniques are applied to extract the three kinds of crowd-intelligence (namely: user awareness, ambient awareness, and social awareness). When information passes through this phase, different statistic methods are applied to classify the quality of processed data.

Data Distribution

Once data have been aggregated, and the crowd-intelligence has been extracted from them, this information is usually made publicly available to be re-used (often as Open Data) or only shared in a private way with authorities, communities, companies, etc.

Exploitation

Finally data arrive at the stage where they are re-used, exploited or just shown. The implementation of a usable user interface and of data visualisation techniques (such as mapping, graphing, animation) are essential to fully exploit the crowd-intelligence extracted by the underlying system starting from raw data.

Cooperation Incentives

As shown in Figure 3.1, the cooperation incentives phase influences almost every other stage of the proposed framework. Users can be motivated to participate in a sensing task by using incentives in the task allocation phase. In the sensing phase,

the idea of a possible future reward can motivate users to collect better or more data.

3.1.2 Real-World Applications

Mobile Crowd-sensing applications can serve as sensing and processing instruments in many different fields. Due to mobile devices inherent mobility, they can be utilised for sensing tasks aimed to gain better awareness and understanding of urban dynamics. Acquiring knowledge in such context is of prime importance in order to foster sustainable urban development and to improve citizens life quality in terms of comfort, safety, convenience, security, and awareness.

Many other studies have investigated urban social structures and events starting from crowd-sensed data. Crooks et al. studied the potential of Twitter as a distributed sensor system. They explored the spatial and temporal characteristics of Twitter feed activity responding to a strong earthquake, finding a way to identify impact areas where population has suffered major issues [17].

Large-scale data can be also collected by means of MCS platforms to analyse the actual social function of urban regions, a kind of data which is usually very difficult to obtain and that can be of primary importance concerning urban planning. For instance, Pan et al. started from the GPS log of taxi cabs to classify the social functions of urban land [18], while Karamshuk et al. tried to identify optimal placements for new retail stores [19].

Awareness of user location is the foundation of many modern and popular mobile applications, such as location search services, indoor positioning, location based-advertising, and so forth. But more useful and precise services can be enabled harnessing all the peculiar characteristics of personal mobile phones. As an example, Zheng et al. used crowdsourced user-location histories to build a map of points of interest which can be of help for people who are familiarising with a new city [20].

Other cases are Geo-Life [22], a Mobile Crowd-sensing platform able to suggest new friendship looking at similarities in user-location logs. Or CrowdSense@Place [24], a framework able to exploit advanced sensing features of smartphones to opportunistically capture images and audio clips to better categorise places the user visits.

In many cases, the development of a particular platform has been the answer to issues raised by pre-existing communities or grassroots initiatives. Citizens and policy makers have usually strong interests in matters like environmental monitoring, public safety and healthcare, where the participatory and mobile essence of the Mobile Crowd-sensing approach provides a novel way for collaboratively monitor the issue being considered.

3.1.2.1 Crowdsensing for the common interest

The moving nature of these topics draws the attention of online and offline communities. The potential of a community can be harnessed by MCS approaches to engage people and to make them participate in the data collection. It is not just a matter of the number of participants, rather someone who is moved by a topic not only will be more disposed to contribute but will also be prone to provide better and more complete data.

As an example, Ruge et al. described how their application SoundOfTheCity [25] allowed users to link their feelings and experiences with the measured noise level, helping in providing information essential to have a more clear understanding of the context (is the high noise level in a party, at a festival or just in a crowded street?). This is an illustrative case of how qualitative data provided by users can enrich the quantitative data gathered through personal smart devices.

In short, to fully harness their potentialities when analysing such contexts, MCS applications should aim not only to collect as much data as possible but also to provide ways for users to enrich the collection with thick data.

Then there are other examples of MCS applications analysing topics of common interest as:

- **NoiseTube** [26] which was a system able to exploit volunteers smartphones to collect data about environmental noise in users daily lives and to aggregate them to obtain a fine-grained noise map;
- **U-Air** [23] inferred air quality data by heterogeneous crowd-sensed data comparing them against information from sensing stations and traffic information.

Healthcare is another field where MCS is helping a lot by collecting a wealth of data for applications more and more useful for an ageing society like ours. Google researchers did pioneering work in 2006 using health-related search queries to estimate illnesses distribution in US [27], while Wesolowski et al. exploited the widespread diffusion of mobile phones to analyze malaria spreading in Kenya [28].

3.1.2.2 SmartRoadSense

An application only based on data sensed using personal user smartphone is SmartRoadSense [14]. The platform is a crowd-sensing system used to monitor the surface status of the road network. The SmartRoadSense mobile app is able to detect and classify the road surface irregularities by means of accelerometers and send them to an in-cloud server. Aggregated data about road roughness are shown on an interactive online map and made available as open data[1].

[1] http://www.smartroadsense.it

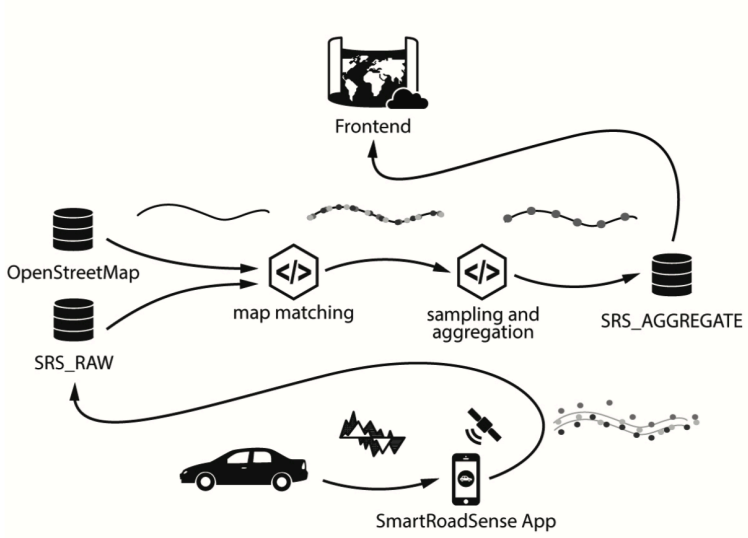

Figure 3.2: SmartRoadSense Process Flow

The system has been developed to provide quantitative estimations of road network surface roughness. The approach at data sensing and processing is general enough to be employed also in different contexts. As many other MCS platform, sensing tasks in SmartRoadSense are performed by multiple distributed sensing devices by means of which volunteers contribute to gauge the quantity of interest in a specific location, within a specific time-window.

As shown in Figure 3.2, the architecture of the SmartRoadSense platform is characterised by the following three layers:

- **Mobile Application:** An app running on users smartphones during a given car trip. The application makes use of the smartphones accelerometers and computation capabilities to collect and process acceleration values the device is subject to. The result, representing the estimated roughness of the travelled road in a given point at a given time, is geo-referenced, time-stamped, and transmitted to a server by means of radio connectivity.
- **Cloud Platform:** A cloud-based back-end service in charge of collecting, aggregating and storing data from multiple users. The platform is in charge of two tasks:
  - **MapMatching:** geo-referenced roughness indexes stored in the database of raw-data (SRS_RAW) are projected on digital cartography maps, specifically OpenStreetMap. Map-matching entails the association of GPS points to features on a digital cartography maps.
  - **Sampling and Aggregation:** data is subsequently aggregated to provide a single evaluation (for a given spatial coordinate) of the roughness index, given the data made available for that point by multiple users. Aggregated data is used to populate the related database (called SRS_AGGREGATE).

- **Visualisation:** A front-end service providing visualisation capabilities of the geospatial information produced by the SmartRoadSense processing pipeline. The same front-end also allows interested end-users to download a continuously updated version of the database containing all SmartRoadSense aggregated data in a ready-to-be-reused fashion (the schema of the table is visible in Table 3.1). Each row of the open-data dataset contains a set of information relative to the roughness level, the geo-localisation, the quality of the data, and even a indication of the estimated number of occupants of each vehicle that has been involved in the gathering process.

COLUMN	FORMAT	DESCRIPTION
LATITUDE	DECIMAL DEGREES	The latitude coordinate of center of the section of the road where the PPE value has been estimated
LONGITUDE	DECIMAL DEGREES	The longitude coordinate of center of the section of the road where the PPE value has been estimated
PPE	DECIMAL	The average roughness level of the road section
OSM_ID	LONG INT	The ID of the road in the OpenStreetMap
HIGHWAY	TEXT	The road category according to the OpenStreetMap classification2
QUALITY	DECIMAL	A numerical estimate of the quality of this particular PPE value. This quality index has been calculated using our bootstrap-based method, in our case-of-study
PASSENGERS	DECIMAL	The average of the number of passengers in vehicles involved in the process
UPDATED_AT	DATE (ISO 8601)	The last update of the data for that particular road section

Table 3.1: SmartRoadSense Open Data Schema

In SmartRoadSense (and possibly in other MCS Systems), the sensing process can be divided into time epochs, during which data is continuously gathered, processed and aggregated. At the end of a given time epoch the system updates current information on the status of measured variables and, in case, It performs a composition operation with data collected in previous epochs (in SmartRoadSense

an epoch represents a week of monitoring activity).

The platform continuously receives values of road roughness from end users. Roads are spatially segmented into landmark points, then all values associated to positions falling within a given range (typically 20 meters) of a landmark point p are aggregated and concur to the overall roughness index of p (the mean value of contributed points is taken by default). At the end of each week current epoch terminates, and the roughness value of each point p is updated by taking the average between the value of current epoch and the value of previous epoch.

This processing inherently implements a form of infinite impulse response filter, the aim of which is to progressively down-weigh (through an exponential decay of weights) the contribution of older samples to the value assigned to p.

Chapter 4

The General Approach for Value Identification

As mentioned in the previous chapter, a data set is considered "Big Data" if It satisfies the properties of: *volume*, *variety* and *velocity*.

However, data alone is not enough. Evaluating Its *veracity* is needed in order to understand if It can be useful to exploit. To explore this property a specific roadmap has been followed and the concept of value has been defined.

4.1 The Concept of Value

What is value in Big Data? As seen in Chapter 2, the word of *value* is included in the concept of veracity. But what really mean *value* for the data? In this case there is no real definition.

If the meaning of value is referable to an economic concept and therefore to an intrinsic property of an object, such as the currency, then the quantity would determine the amount of value held and the greater the coins owned, the greater is the value. Then we could converge on the same idea: a large amount of data represents a great value. However, some could argue: for the same amount of coins held, the one that owns more value is the one who possess those with the higher intrinsic value. But a third party intervenes and claims that It is the market that gives value to the currency and since some currencies have greater market value, in the end the one who has more value is who possesses the higher amount of coins with the greater intrinsic value and market value.

The problem is therefore to ask if the datum represents an intrinsic value: my answer is no. If the datum represented an intrinsic value then we should not verify information or quality of the data, as this property would be intrinsic to the information itself. But in fact, this is not the case for data, which in no way represents an immutable concept. The data is basically a raw material that needs to be processed to generate value.

Therefore, It is not enough to say that: the data acquire value if they are in a big amount. The value is determined by indicators, even multiple, which we could define as *metrics*, that if crossed together can generate the notion of value. For example, the value in the case of geospatial data could be determined by the fact that they must be in large quantities and be recent. However, this statement is strongly dependent on the goal of data handlers. To conclude, by value of the data we mean *the goal generated by the data after a regulated evaluation respecting specific indicators, called metrics.*

4.2 The Proposed Approach

When looking to geospatial data It is interesting to know which records would be used for exploitation (i.e. researches).

In big data sets as geospatial ones, It is a challenge to realise which records are useful for this purpose. The data may present several complications:

- **Size:** disk space can be large to the order of gigabytes or terabytes;
- **Volume:** the amount and the complexity of records may be high and the greater the complexity, the greater the computational cost required to process a record;
- **Velocity:** the records could be updated frequently;
- **Sparsity:** the geospatial points could not be distributed equally on time and space so, It is difficult to have enough information regarding the territory analysed, despite the high volume.

To this end it makes sense to implement an approach developed in different phases, in order to define a process that suggests the correct analysis of geospatial data sets. The process thought is based on 5 main phases:

1. **Exploratory Analysis:** a very first analysis to obtain more information about the data set considered;
2. **Outcome Definition:** the definition of the metrics used to give the definition of value;
3. **Pre Processing:** reducing data set volume avoiding redundancies;
4. **Processing:** using the different metrics to generate results;
5. **Visualisation:** giving a visual representation of what has been obtained.

4.3 Exploratory Analysis

Exploratory analysis is the very first phase to face when having an unknown data set. In this phase It is necessary to take into consideration the shape characteristics of the data set such as its weight and the volume in terms of records, trying to get a comprehensive idea of the data set itself and its critical issues.

4.3.1 Data set

A data set could be structured or unstructured but reducible as a set of rows and column. Analysing a data set means mainly understanding its shape characteristics, i.e. the size, the number of records contained and the amount of information per record. This last point is particularly important since the number of columns generated impacts on the computational complexity and memory occupation. For this reason, this phase focus on carry out a quantitative analysis of the data set that allows to anticipate the dimensions in RAM.

The RAM memory occupancy estimate in Python can be easily performed by having a single data set record available, appropriately transformed according to the criterion that will be used during the Pre-Processing phase. This is because the information that you want to get from the single record is its memory occupation and therefore It is necessary that the record is formatted exactly as It will be. The second information concerns instead the number of rows contained in the data set, information obtainable through the Unix terminal, through Spark or the I/O methods available in Python. At this stage It is clear that the data set should only be read, any attempt to load a data set of the Big Data order into memory will saturate the RAM available on the machine, making the application unresponsive.

The estimate can then be reduced to:

$$memory\ occupancy * number\ of\ lines$$

To be more detailed about the methods that can be used to derive this information is good to cite:

- **UNIX:** *wc -l <dataset.extension>*
- **Spark:** *count()* method
- **Python:** Read and Write operations on files

4.3.2 Insights

Once that the data set form is clear, It is necessary to understand the distributions of data over space and time. Geo-spatial data sets contain millions of records that are potentially distributed throughout the world. This last assumption directly refers to what had been introduced in the previous chapters: in such

a large context of data collection such as geo-spatial data, the risk of not having enough data to be able to generate value is very high and strongly dependent on the problem considered. As anticipated, It is not enough to know quantitatively the distribution of data on the territory to understand Its value because quantity alone is not synonymous of value. The data must be interpreted according to the context in which they are considered.

The exploration is therefore very important at least to understand the obstacles that stand in the ability to generate the value from data, quantifying them qualitatively. For example, in Figure 4.1 a road for which the data collected over time have been obtained (many similar ones have been found). It is clear that for the SmartRoadSense data set there is a problem of fair distribution over time and space, perhaps due to the youth of the application, which in any case hinders the achievement of value.

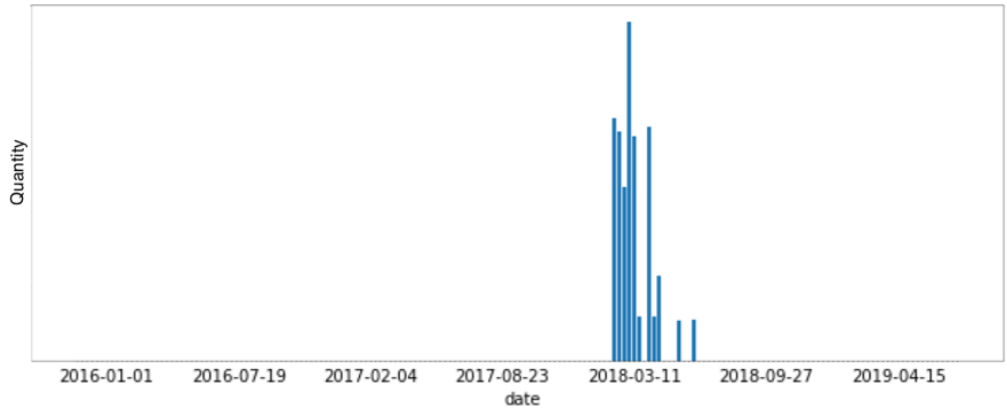

Figure 4.1: Example of bad data collection

4.4 Outcome Definition

After obtaining the insights regarding the data set, It is time to define the goal to achieve, that is the metrics that will be used to define the value.

The idea is that since geospatial data sets could show the previous deficiencies, the focus should be on appropriate metrics that together can generate results that can be crossed together in order to highlight which records can be more promising for one's needs.

The goal should be focused on using metrics that are meaningful for the work considered. One could search for the amount of data in a territory, their degree of topicality, the trend over time of the recordings and so on. Other metrics can be freely added or the previous proposals can be removed if not useful.

These tasks could then be converged into a single solution that can fully describe the data considered, allowing a correct final big picture retailed on its own definition of value.

4.5 Pre Processing

The Pre Processing part is essential for the final solution. Once the metrics are settled, with all probability a large part of the available data will not be useful for solving the problem because they simply represents redundancy.

For this phase, Spark tool is available but in order to perform data Pre-Processing, lots of companies use other solutions like UNIX, perhaps coupled with incremental loading. But many tools are available: Python, or the same SQL could help.

However, these approaches vary from problem to problem and are not always applicable especially with geospatial data, for example:

- Using SQL assumes that a distributed database in which the data is stored is available, such as PostGIS. Instead, often geo-spatial data is provided as Open Data in text files.
- Using SQL assumes that any geospatial operations are procedures within the database, limiting the freedom that a programming language like Python offers.
- Using the UNIX console for non-trivial operations such as a spatial query is not a valid alternative to Spark in addition to introducing perhaps greater restrictions than those offered by SQL with a distributed geospatial database.

Spark's success is therefore guaranteed because of the convenience It represents: its instruments have a great usability and are much more extensive. Despite not finding a real application in local, their use in limited available hardware allows a better settings of the algorithms developed with it.

In fact, the Pre Processing phase if done locally allows to put in place all the necessary tools in the absence of resources, allowing to not worry about the resources available in the code writing and optimisation phase.

Moreover, the Pre Processing phase in the case of geospatial data involves not only the removal of redundancies but also a multitude of other operations such as spatial functions. Spatial functions, or called spatial queries, are useful methods for selecting records on a territorial basis. Through Python It is possible to perform them and with Spark It is possible to develop an algorithm that can exploit the maximum parallelism to execute them. An example of a spatial query application is shown in Figure 4.2 where all the municipalities have been selected across the borders of the Marche Region.

Figure 4.2: Municipalities Selected through Spatial Query in QGIS

If you imagine that this kind of operations should be performed for every single record of the geospatial data set, the complexity of the problem tends to explode. To this end, in this phase the optimisation is fundamental for a subsequent implementation on clusters with the precise objective of providing the results in a short time, since It makes sense to obtain results in a short time in a real-time context.

For the implementation of clusters there are several providers available, for example Amazon, Google and Microsoft. For the project Google DataProc was used for convenience but any other solution is certainly valid.

4.6 Processing

At the end of the Pre-Processing phase a new data set is produced which is almost free of redundancies and tailored in respect of the goal, therefore containing the information necessary for the implementation of the metrics.

In this phase the defined metrics are implemented and subsequently crossed together to form the final data set which should be a correct summary of the three. Python in this case lends itself very well to the purpose, allowing data manipulation, which this time is no longer of unmanageable volumes.

However, before generating the results It is necessary to carry out a further operation that can guarantee the quality of the data set generated in the Pre Processing phase. In particular, in the specific case of SmartRoadSense, but this may happen for any data set, the anomalies may have been easily introduced for some reason. Before generating the results of the predefined metrics It is therefore necessary to verify that the data set produced is free of anomalies, or that the new distributions generated contain data that are consistent with one another.

As an example, once the Pre Processing operation was performed on the SmartRoadSense data set and the new data set was generated with the necessary information for the metrics, It was noticed that many distributions presented data far outside

the average, such as in Figure 4.3. Their removal is therefore necessary before proceeding.

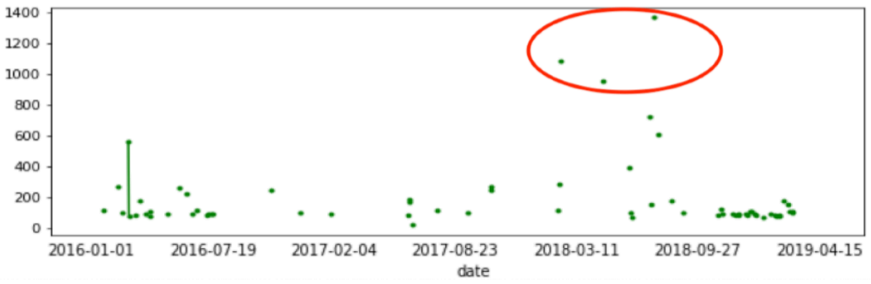

Figure 4.3: Example of Anomalies in a Time Series

Finally, It is possible to proceed doing the operations that aggregate the data to obtain the values for the necessary metric. Once crossed, the results are ready to be shown.

4.7 Visualisation

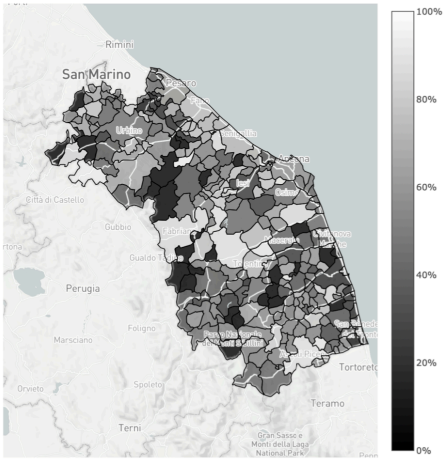

Figure 4.4: Example of Visualisation with Plotly

The visualisation consists in giving a final picture of the results obtained in the Processing phase. On the basis of the Processing phase, It makes sense to visualise the generated results for immediate understanding.

The tools useful for visualisation are many: It is possible to use web apps, mobile applications, libraries available in different languages. The criterion is free but has the common objective of forming an overview of rapid comprehension.

In this work the *plotly* library available in Python was used for convenience, which allows the overlapping of several layers and the insertion of legends and captions as in Figure 4.4

Chapter 5

Implementation Experience

The implementation is based on the process defined in the previous chapter. The meaning in terms of implementation is illustrated in the next sections, point by point. To develop the final solution the Spark environment has been used locally, the drawbacks related to this choice are discussed later on Chapter 6, as well as the solutions to them (the use of a cluster). This evidences of this is not discussed in this chapter since the algorithm written with Spark (locally) can be ported on a cluster with little efforts.

To sum up, the approach for identifying the value in geospatial data sets is as follows:

1. **Exploratory Analysis**
2. **Outcome Definition**
3. **Pre Processing**
4. **Processing**
5. **Visualisation**

5.1 Exploratory Analysis

In the very first part of the process a qualitative analysis of the data is carried out which led to the formulation of several important considerations for the development of the final solution.

5.1.1 Data set

The data set available comes from a PostGIS database of hundreds of millions records. SmartRoadSense is built on two different databases:

- **Raw Database:** It contains the measurements coming from the mobile application that runs on mobile devices. This data is stored for its future aggregation.

- **Aggregate Database:** It contains data generated starting from raw data. It is weekly generated for each road throughout the year.

The data considered is part of the Italian peninsula, country (together with the United Kingdom) from which the pilot of the project started, therefore at least theoretically more populated of information. As a local area for showing the process an even smaller area has been focused (the Marche region) for three main reasons:

- The project started at the University of Urbino, whose municipality resides in the Marche region
- The approach generated must be scalable regardless of the volume of data

5.1.2 Row Attributes

A typical row of the SmartRoadSense dataset contains the following information (only the significant ones will be cited):

- **single_data_id:** Unique ID of the measurement;
- **date:** Time Instant of Sampling;
- **osm_line_id:** OpenStreetMap ID;
- **ppe:** Deterioration Value;
- **speed:** Vehicle Instantaneous Speed;
- **position:** The projection of "Position" on the relative road;
- **track:** The track that registered the information (available only for raw data).

The following attributes: *osm_line_id*, *position*, *track* play an important role in the analysis because they constitute three different concepts:

- **Section:** a section is identified through its *osm_line_id*, the id through which OpenStreetMap identifies a section of a road in its global map. In other words, for each road of any length, OpenStreetMap maps portions of this street within its dataset by associating a given ID. Consequently, a real road is often composed of several sections.
- **Segment:** a segment is identified by the position attribute. Within a section, many segments of a length of twenty meters are identified, therefore the number of segments contained in a section will be equal to the ratio between the length of the section in meters and the length of the segment.

- **Track:** when a user collects data He produces a track. A track can involve several roads, as well as several section. This information is only available for raw data, when the data is collected. This means that when a user travels on some road, an identifier is marking his trip.

5.1.3 Retrieving Informations

A Spark environment was used on the local machine for exploring the data set. The SmartRoadSense data set of the Italian peninsula alone contains about 20 million records and a number of columns for each row equal to 16. Through the code in Listing 5.1 it was possible to estimate the amount of memory needed to load the entire data set in RAM. The result is just over 20 GB of RAM.

```
1 data = sc.textFile('../Datasets/Extraction 10/raw_italy.
    csv') \
2           .map(lambda line: line.encode('ascii', 'ignore'
              ))
3
4 clean = data.filter(lambda row: row != header.value) \
5               .map(lambda row: re.sub(r'\{[\s\S]*\}', '{}'
                  , row)) \
6               .map(lambda row: row.split(",")) \
7               .map(lambda row: row[:4]+[convert_date(row
                  [4])]+row[5:])
8
9 print sys.getsizeof(clean.take(1))*20000000
```

Listing 5.1: Evaluating RAM Occupation

Subsequently, the exploration continued extracting data step-by-step, developing an algorithm to plot results avoiding to load the whole amount in memory. The algorithm developed on Spark, although poorly performing due to local operation, was pretty useful for plotting without the risk of having the RAM saturated. The code is visible on Listing 5.2.

```
1 %matplotlib inline
2 import os
3 for road in unique_roads:
4     quantity = []
5     for elem in tqdm(interval):
6         count = rdd_plot.filter(lambda row: row[0] ==
              road)\
7                         .filter(lambda row: row[1] <=
                             elem and row[1] > elem -
                             datetime.timedelta(days=7))\
8                         .count()
9         quantity.append(count)
```

```
10
11      fig = plt.figure(figsize=(32,12), dpi=100)
12      plt.bar(np.arange(len(interval)), quantity)
13      plt.title('Road '+str(road))
14      plt.xticks(np.arange(len(interval)), interval,
           rotation=90)
15      plt.xlabel('Time')
16      plt.ylabel('Row Data Collected')
17      plt.savefig('Road_'+road+'_Distribution.png', dpi
           =100)
18      plt.close()
```

Listing 5.2: Plotting exploration's results

5.1.4 Insights

What came out analysing the the plots generated is that the data is still too few and scattered even when decreasing the granularity. For example, in Figure 5.2 the granularity is totally reduced: It only shows the quantities of measurements collected over time from the first moment the SmartRoadSense platform was activated (2016) for a certain road. By a quick look of the graph It is clear that the available window of data is too small (18 weeks, more or less 3 months). This plot has been generated by taking the twenty most travelled roads, i.e. those with the greatest number of measurements.

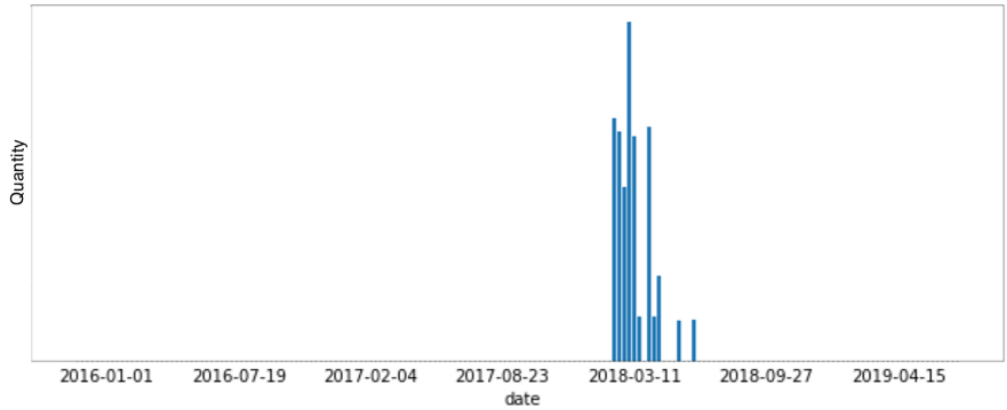

Figure 5.1: Measurements distribution for road 97464677

So, if the latter case was incomplete, trying to increase the granularity looking to each section of one road is useless. Suppose that for a future research we should be able to guess the speed of deterioration or predict when a certain section will deteriorate: with the available data this forecast is not possible even by using filling techniques that can make the time series complete. There is simply no data to generate value regarding that road.

It is easy to imagine that in such a context with lack of continuity in the measurements, no forecasting could be possible regarding a microscopic value like the *PPE* collected along the road. The inaccuracy associated with any model would be too high that the work associated with lots of case study could be found not helpful. However, the amount of data available remains high, that means the data could represent value. But the problem is clear: there is a need of correctly visualise the distribution of the data on the different territories where they have been collected in order to understand which of them could be valuable enough to provide good data. There is the need of identifying the data that can generate the value.

5.2 Outcome Definition

So, since the data distributions built from raw data shown obvious deficiencies related to the distribution over time, this suggested to focus on solving the problem through a strategy aimed at generating results that can be crossed together in order to highlight which region of data can be more promising in generating value.

Since the time series could be used for several reason, i.e. forecasting, the value in this specific work is represented by the idea that the time series found for a specific road is not sparse and so, Its collected data are equally distributed over time and possibly increasing. This goal could be decomposed with three different metrics:

- **Data Quantity:** How many data belongs to a certain region that can be calculated as the summation of the collected samples for each municipality;
- **Data Recency:** Which is the recency of the data over a certain region that can be calculated as a weighted average between all the collected sample for each municipality;
- **Usage Trends:** Which is the trend of data collection over a certain region that can be evaluated by a model (linear in this work) generated considering the quantity of data collected over time.

These three tasks will lead to identify the territories in which the data should generate the value looked for.

5.3 Pre Processing

The computations are based on the amount of data collected over time over a specific area, in this work the Marche region. For such a problem what is needed to compute the solution is to:

- Know the quantity of measurements in a given territory: the *Latitude* and *Longitude*

- Know the time in which these measurements took place: the *Date* in which the record has been collected
- Uniqueness of the record: the ID of the record
- Spatial region: the *Polygon* that defines the area

A raw data set record is composed of the previous fields, this is described in section 5.1.2. Only one of these does not concern the raw data and is the border of the individual municipalities. The perimeters were extracted from the multitude of shapefiles available online.

5.3.1 Spatial Query

The first step is to group data by areas. To obtain this result It is necessary to perform spatial queries for each record considered.

Through *Longitude* and *Latitude* It is easy to know if one collected record is included inside an area or not, which bounds are defined by polygons. To perform this task the shapely library is available in Python. The objects and methods used in the next listings comes directly from this library, as *Point* and *Polygon* objects:

- **Point(latitude, longitude)**
- **Polygon([(latitude, longitude)])**

Through the *Polygon.contains()* function It is possible to check whether a point is included in a specific polygon, as in Listing 5.3:

```
1 # Check if a point lies inside a polygon
2 point = Point(12.50, 43.74)
3 polygon = Polygon(geom['coordinates'])
4 print(polygon.contains(point))
```

Listing 5.3: Polygon and Point Objects

The result of the *contains()* function is a Boolean:

- **True**, if the point is contained;
- **False**, if It falls outside the polygon.

In any case, performing spatial queries on large data sets is particularly complex on a local machine. It is sufficient to think that by testing the algorithm on a local Spark infrastructure (single node) related to the Marche Region took about 3 hours before It terminates. This was possible only thanks to the fact that the data set is still not so big. But, since SmartRoadSense is continuously updating records

in real-time, It is predictable that to obtain results in the future will become incrementally complex that to retrieve results in an acceptable time will not be possible.

So, in order to develop a general solution scalable and ready to be used on a cluster, the algorithm should be general enough to be moved from a local to a cluster environment easily. And this is actually possible with Spark working locally.

In order to do this, the data set has been loaded into the Spark Context, It has been cleaned up of all the redundancies and repartitioned along all the cores of the local machine. Then, finally cached. This is shown in Listing 5.4. Caching the RDD gives the advantage of avoiding that Spark can start from the beginning of the transformations every time an action is called. It basically permits to save a lot of time when processing the same RDD.

```
1 data.filter(lambda row: row != header)\
2     .map(lambda row: row[:row.find('{')-1]+row[row.find(
        "{")+1:row.find("}")].replace(",",";")+row[row.
        find('}')+2:])\
3     .map(lambda row: row.split(","))\
4     .map(lambda row: [row[0]]+[row[6]]+[str(convert_date
        (row[4]))]+[coordinates(row[7])])\
5     .repartition(8)\
6     .cache()
```

Listing 5.4: Transformed and Cached Dataset

In Listing 5.4, two functions are called: *convert_date()* and *coordinates()*. The convert_date() function takes care of simply formatting the dates in the same way: *<year-month-day>*.

```
1 def convert_date(date):
2   try:
3     result = datetime.datetime.strptime(date, '%Y-%m-%d
        %H:%M:%S').date()
4   except:
5     result = datetime.datetime.strptime(date, '%Y-%m-%d
        %H:%M:%S.%f').date()
6   return result
```

Listing 5.5: Date convertion function

The *coordinates()* function takes care of transforming the hex referred to the position. Latitude and Longitude are encoded and saved as *hex* in the SmartRoadSense database, so this step is needed.

```
1 def coordinates(hex_location):
2   point = wkb.loads(hex_location, hex=True)
```

```
3   return str(point.x)+";"+str(point.y)
```

Listing 5.6: From Hex to Lat/Long

Once the RDD has been cached It is possible to perform the spatial queries. Note, as mentioned earlier, the importance of this step since spatial queries, as shown in Listing 5.7, are performed for each individual RDD record. In the absence of caching, this step would risk being considerably slower, of an order as higher as larger is the working RDD.

```
1 intervals = []
2 for municipality in tqdm(marche):
3   boundaries = []
4   for elem in municipality['geometry']['coordinates']:
5     boundaries.append(Polygon(elem[0]))
6     intervals = clean.filter(lambda row: isInside(row
        [3], boundaries) == True)\
7                      .map(lambda row: (row[2], 1))\
8                      .reduceByKey(lambda x,y: x+y)\
9                      .collect()
```

Listing 5.7: Processing function

The spatial query is called by the Spark driver with the *isInside()* function visible in Listing 5.8. The function takes the polygons (the areas) and verifies that the record is included within the boundaries of the polygons themselves.

```
1 def isInside(location, polygon):
2   point = Point(float(location.split(";")[0]), float(
       location.split(";")[1]))
3   for elem in polygon:
4     if elem.contains(point):
5       return True
6   return False
```

Listing 5.8: Spatial Query

The complexity of the algorithm is determined not only by the size of the problem but also by the type of transformations and actions performed on the RDD. Specifically, Spark takes much longer to do grouping and filtering rather than map transformations. This is reflected in the time required for the development of results. Note that the algorithm contains all the three transformations.

Now, considering that these operations should be repeated as many times as all the Italian municipalities, It comes out that this processing is computational intensive. It is not suspicious that the whole processing took so much time before termination.

For this reason, when the Pre Processing is concluded, a file with the results is generated to avoid recomputing it. The file created contains the information related to the quantities for each territory. To be specific, It is a *json* that contains all the municipalities and the dates on which the data was collected, along with their quantities, as in Figure 5.3.

```
{
    'municipality': {
        'date1':quantity,
        'date2':quantity,
        ...
    }
}
```

Figure 5.2: JSON format after Pre Processing phase

5.4 Processing

The Processing phase focuses on using the data set produced during the Pre Processing phase in order to capture the information necessary for the fulfilment of certain metrics.

Consequently, the Processing phase involves two operations:

- **Anomalies Detection** operations that ensure that the data are ready to be used. It can be seen as a last Pre Processing before the Processing for metrics results generation;
- **Development and Implementation** of the different metrics to use for the final value identification.

5.4.1 Anomalies Detection

There is no way of knowing if a given data set contains anomalies without understanding the data set itself and It is not a good practice to leave this task to automated algorithms. It would be like wanting to use a specific screwdriver without knowing the kind of screw.

In the analysed case of SmartRoadSense the anomalies are caused by the campaigns that have been activated over time. In order to encourage data collection the SRS team established several days in which the application was advertised in order to increase its usage. In other words, the quantity of data produced during some days could be totally disproportionate with the real trend of usage of the application and therefore the data collected in correspondence of those days must be excluded from the final evaluation.

The algorithm that deals with detecting and removing the anomalies used in the project is a simple low pass filter, widely used and replaceable with possibly more advanced techniques, but this is not the purpose of the thesis. The procedure used for the removal of the anomalies is described in the next paragraphs. The following examples will take a sample time series (shown in Figure 5.3) to show but the same approach has been used for all the municipalities.

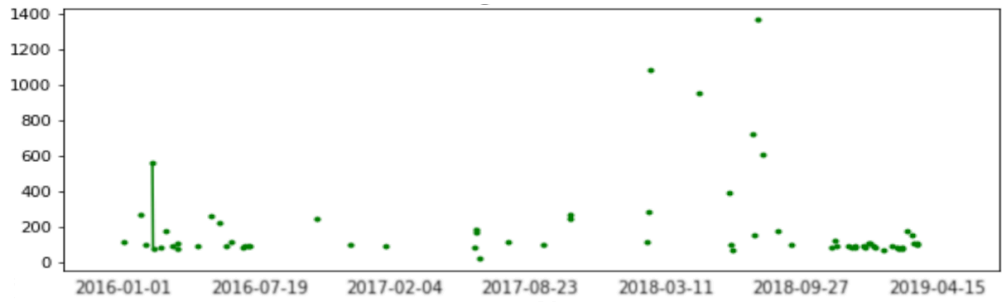

Figure 5.3: Sample time series

Figure 5.3 shows scattered and discontinuous data over time, showing some outliers too. This behaviour is expected as said just before because data collection campaigns have been activated consequently generating anomalous growth of data collected during some days. This time series needs to be cleaned of anomalies to better understand what is really happening to data.

However, the discontinuities in the data need to be managed. In order to build a series without missing data there are different possibilities: *Forward Fill*, *Backward Fill* and *Linear Fill*. But, none of these three is correct and their use should be considered on the basis of the problem to be carried out. In this particular case, the first two techniques would produce the results visible in Figure 5.4 and 5.5:

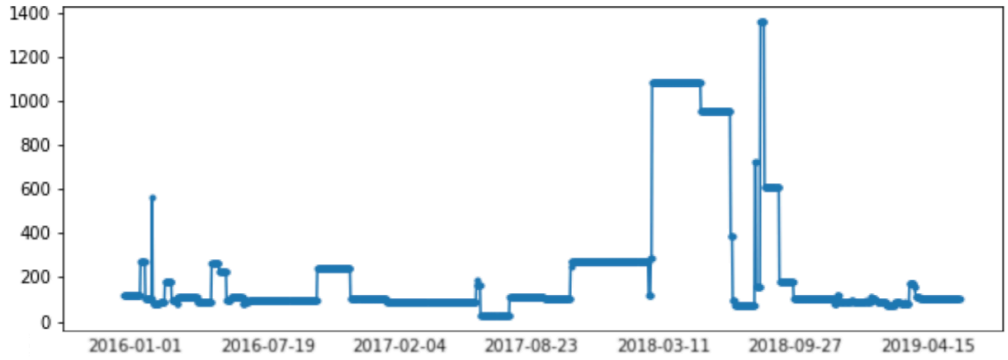

Figure 5.4: Forward Fill on sample time series

These results are not very helpful as they do not allow to have a complete time series for the period under consideration: in the case of Forward Fill we will have absence of data in the queue, while on the contrary for Backward Fill the absence of data could occur in the head. To obtain a complete result, regardless of the

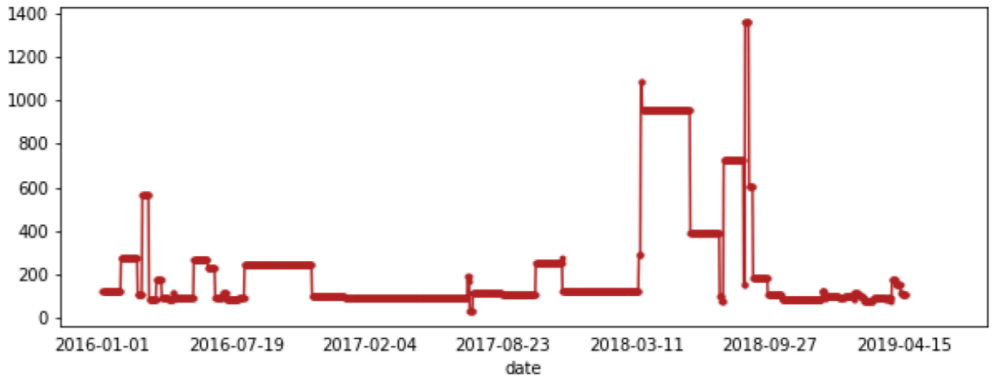

Figure 5.5: Backward Fill on sample time series

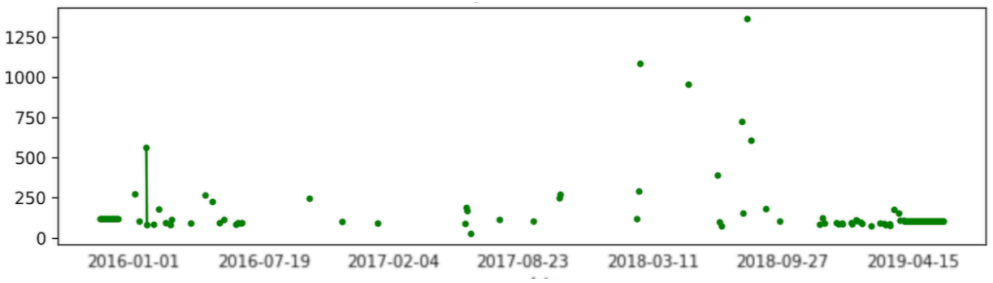

Figure 5.6: Corrected sample time series

technique, we should first model the series as in Figure 5.6.

So, Backward Fill or Forward Fill should be chosen based on two considerations:

- It is assumed that the data preceding a known datum is at the same value
- It is assumed that the data following a known datum is at the same value

In our case, generating an artificial datum equal to a known one could correspond to consider that the same quantity was collected on a previous day (Backward) or later day (Forward). This is something the is highly improbable but at least It allows to avoid inserting anomalies.

The solution is found using a combination of two of the techniques: the *Forward Fill* for data from the head to the beginning of the tail of the Time Series and the *Backward Fill* to fill the tail as shown in Figure 5.7.

However, the *Forward Fill* has some limitations in terms of representation: a function as a moving average would produce poorly credible results. The fact that It is a piecewise constant function due to the lack of data would end up creating a very distant from reality function, much more focused on the traits than on the real points of the function itself, as visible in Figure 5.8. The final goal should not be forgotten: eliminate the outliers.

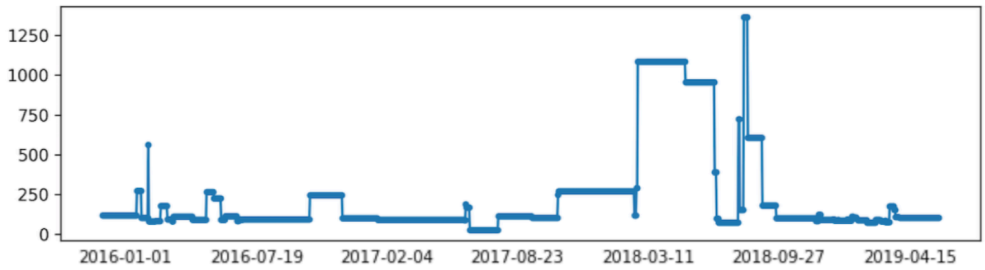

Figure 5.7: New time series originated by the corrected one

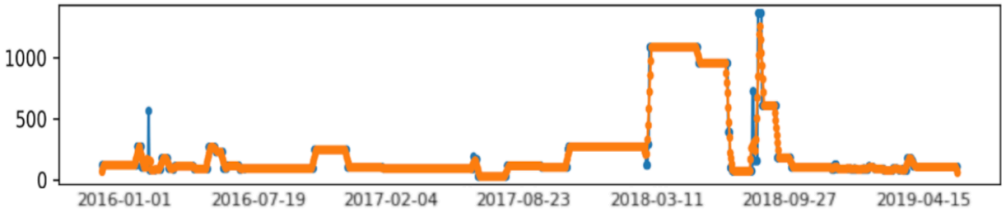

Figure 5.8: Removing anomalies through Forward Filled Series

So, what if using a linear interpolation instead of forward filling then? Linear interpolation consists substantially in generating artificial data that is not constants at all, but rather the result of the straight line joining the two neighbouring points as in Figure 5.9.

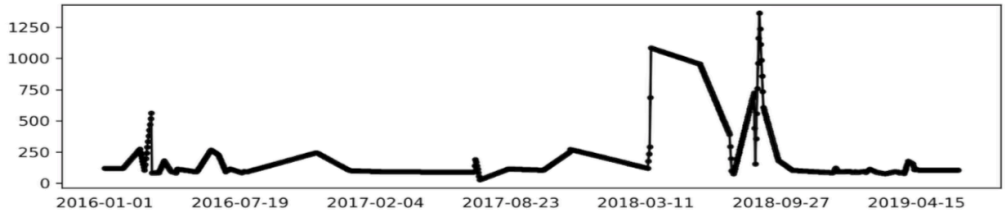

Figure 5.9: New time series filled with Linear Interpolation

Using the *Linear Fill* for the point between the head and the tail could give better results when looking for anomalies. And in fact, the function already identifies the outliers more clearly as shown in Figure 5.10. Therefore, It is possible to identify them through a low-pass filter, one of the most used techniques for the identification of anomalies in the time series.

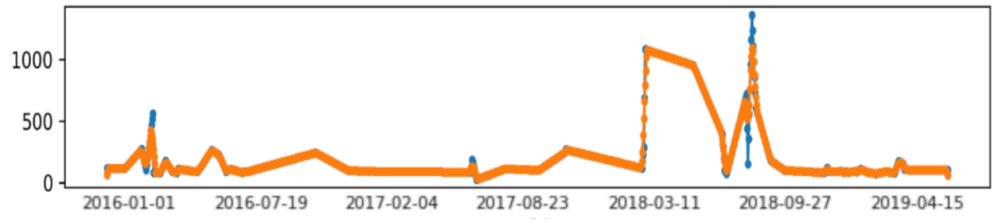

Figure 5.10: Removing anomalies through Linear Filled series

Identified the outliers with the previous technique, It is possible to remove them from the initial time series, generating a new one free of anomalies (Figure 5.11).

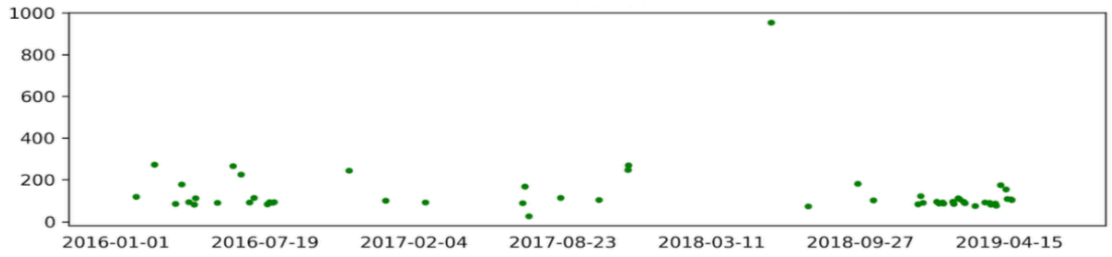

Figure 5.11: Time series cleaned of the anomalies

5.4.2 Usage Trend Metric

The general idea is to define a trend (i.e. stable, increasing or decreasing) for each area considered, in our case the municipalities. In order to do this, a definition of the term *usage* in our context should be carried out.

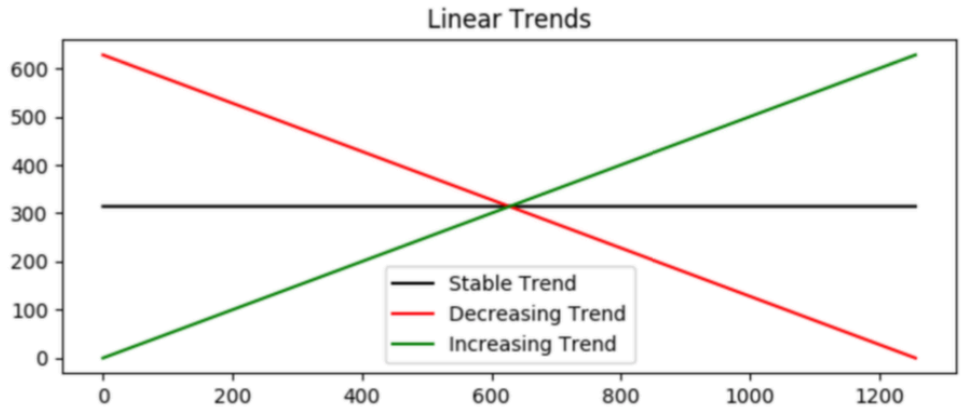

Figure 5.12: Examples of different trend cases

In this specific context, the meaning of usage can refer to the number of points collected over time. When the collected data is increasing, a positive trend should be highlighted, as well as when the data is decreasing, a negative trend should be highlighted. The same should happen when the data is not decreasing nor increasing, in that case the trend will be stable.

So, what is expected is to reach a result of this kind based on the data, breaking down the problem on territorial basis: municipality by municipality.

In Italy there are more or less 7998 municipalities[1] and in the case examined in

[1]The shapefile of the Italian Peninsula is updated to 2016. Today there are fewer municipalities, according to Wikipedia they are 7918 [https://en.wikipedia.org/wiki/Comune]

this work, the region of Marche has 236 of them, about the 2.95% of the national territory.

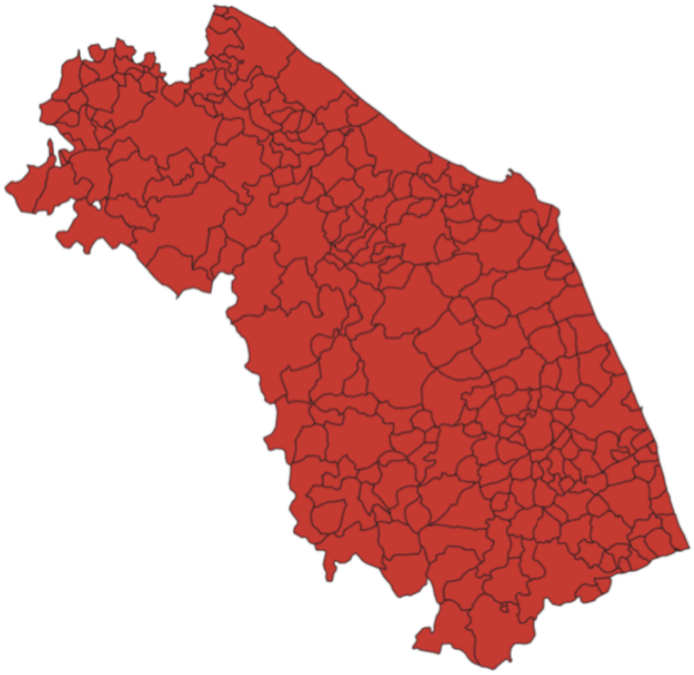

Figure 5.13: All the municipalities of Marche Region in Italy

Finding the Trend

Filling the time series by linear interpolating the point is not valid for the purpose of identifying the trend. In this case, linear interpolation is a function that risks to negatively influence the trend since It is not correct to think that the points absent in the series grow linearly over time.

Actually, none of the three proposals is correct: neither that they are constant (Backwards or Forward Fill), nor that they grow linearly. However, to define the trend between these three functions it is preferable to use a constant function that, in a certain sense, does not give a negative or positive contribution compared to neighbouring points and assumes that in those days the collected date were almost the same as in the other days.

Applying the Forward Fill and using a linear model the trend is defined as in the Figure 5.14. The value of the slope that is the indicator highlighting the trend is represented by the coefficient of the line, in this case positive and equal to +0.18

- **Positive:** the trend is growing
- **0:** the trend is stable

- **Negative:** the trend decreases

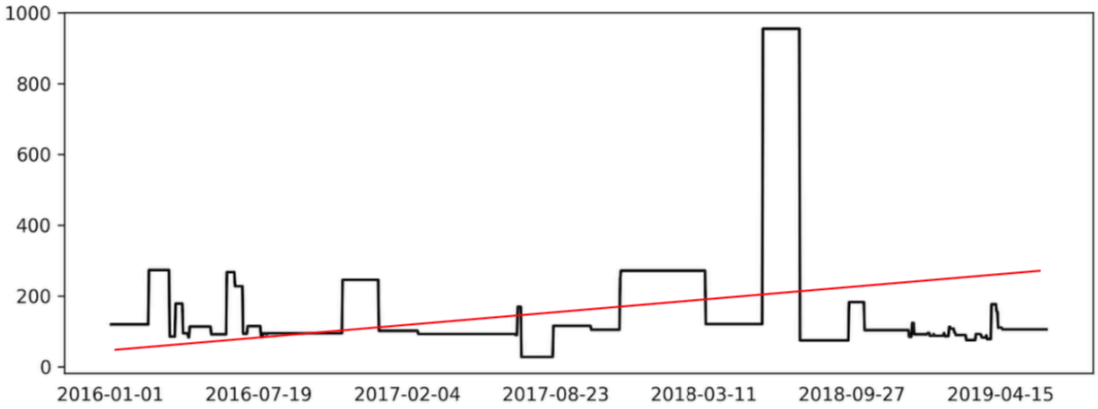

Figure 5.14: Trend of Acqualagna Municipality

The code that performs the previous operation is shown below. The operation consists in finding the linear model based on the points, with the help of *numpy* library.

```
# Generating Trend Values
def trendline(data, order=1):
  coeffs = np.polyfit(data.index.values, list(data),
     order)
  return [float(elem) for elem in coeffs]

trend_d = {}
for key, value in tqdm(sorted(municipalities_d.iteritems
   ())):
  df = pd.DataFrame(list(sorted(value.iteritems())),
     columns=['year','quantity'])
  df_ffill = df.ffill()
  coeffs = trendline(df_ffill['quantity'],1)
  slope = coeffs[-2]
  trend_d[key] = slope
```

Listing 5.9: Trend Value Function

5.4.3 Data Recency Metric

The idea behind the recency of the data is that within each single considered area an indicative value of the distribution over time of the quantity of collected points can be returned.

The process should therefore return a normalised value between 0 and 1 that can visually communicate the status of the data when the analysis is performed

on a consistent scale.

The generated Pre-Processing data set cleaned of anomalies contains all the information necessary for the recency evaluation. The information used for the final solution are two:

- Creation date;
- ID.

Basically, if a road is totally recent then all its points are updated to date. As the points are old, their weight decreases in the recency calculation. For example, if a road possess 10 points collected globally and all of these have been collected today, then their weight will be equal to 1 and the final average will give a 100% recency. This is expressed by the following equation:

$$Recency = \frac{\Sigma_i weight\ of\ date_i * municipality\ collected\ samples\ on\ date_i}{total\ municipality\ collected\ samples}$$

The result will be equal to 1, which means the data is 100% recent. The weight for the collected data in a specific date is chosen based on a decaying window (Figure 5.16) in which: if the data have been collected near the present then the weight will be near 1, or on the contrary near 0. In this work, the window is considering three years before weighting a record with 0 but the choice of the amplitude of it can be modified.

Figure 5.15: Recency Decaying Window

The code that generates the recency value is shown below. The operation consists in generating a new value between 0 and 1 based on the moment in time in which data have been collected.

So, first of all the width of the interval is retrieved, starting from January 1st 2016 up to today.

```
1 # Generating Interval
2 minimum = datetime.date(2016, 1, 1) today = datetime.
     date.today()
3 width = today - minimum
4 width.days
```

Listing 5.10: Date interval for Recency

And then the value calculated:

```
# Generating Recency Values
recency_d = {}
for key, value in tqdm(sorted(municipalities_d.iteritems
    ())):
  collected = 0
  for k, v in value.iteritems():
    date = datetime.datetime.strptime(k, '%Y-%m-%d').
       date()
    weight = (date - datetime.date(2016, 1, 1)).days /
       float(width.days)
    recency += weight * v
    collected += v
  result = recency/collected
  recency_d[key] = result
```

Listing 5.11: Data Recency Value Function

Where v is the amount of points collected in a certain date k, for a specific municipality *key*.

5.4.4 Data Quantity Metric

Finding the quantity of the data is an easy task since It only relies on the amount of data over a territory. This means that It does not matter how the data is distributed over time but simply how much of It is in the considered area.

The generated dataset contains all the information necessary for the formulation of this visualisation. Getting a solution in this case is much simpler and after removing the anomalies, It is possible to find the results. The information needed is simply the number of records in a certain area that can retrieved as shown in listing 5.12.

```
quantity_d = {}
for key, value in tqdm(sorted(municipalities_d.iteritems
    ())):
  collected = 0
  for k, v in value.iteritems():
    collected += v
  quantity_d[key] = collected
```

Listing 5.12: Data Quantity Function

As in the previous case, v is the amount of data collected in a certain date k, for a specific municipality *key*.

5.5 Visualisation

Once the files containing the results of the metrics to be displayed have been generated, they can be used in any way, for example with the *plotly* library made available on python. An explanation of the generated files with the referred values to the single municipalities is shown in Table 5.1.

Metric	Values	Meaning
Usage Trend	(-1, 0, 1)	-1 Negative Trend, 0 Stable Trend, 1 Positive Trend
Data Quantity	[0, inf[	Quantity of Data Collected
Data Recency	[0, 1]	Recency of data collected. 0% Recent to 100% Recent

Table 5.1: Metrics Explained

The results of the three metrics are visible in Figures 5.16, 5.17 and 5.18 and as you can see, all three give different outcomes, which makes It difficult to choose the portion of data to be investigated for future studies.

The colours used differ in what they must represent. A grayscale was used to represent the quantity and recency metrics. If white is intense, then the municipality will have more data and the data collected will be more recent. Conversely, if the color is black, the amount of data and recency will be lower. As far as the usage trend is concerned, a Red-Yellow-Green chromatic scale has been chosen to represent the three different conditions in which a trend can appear: negative (Red), positive (Green) or stable (Yellow). The grey color emerges in those municipalities where the data did not allow the formulation of the trend, perhaps due to a lack of information.

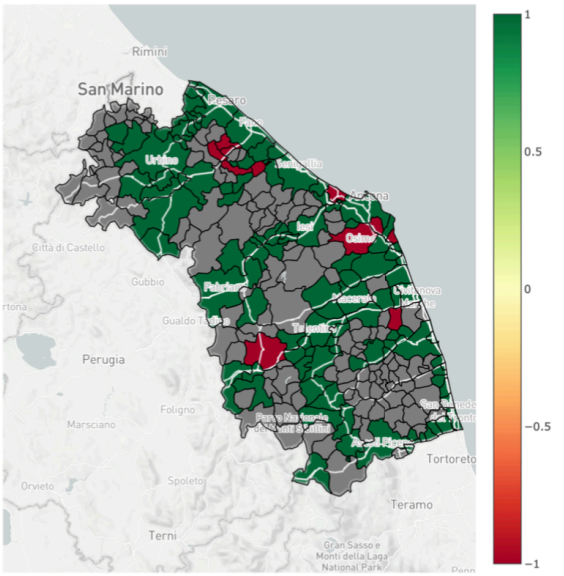

Figure 5.16: Usage Trends Metric

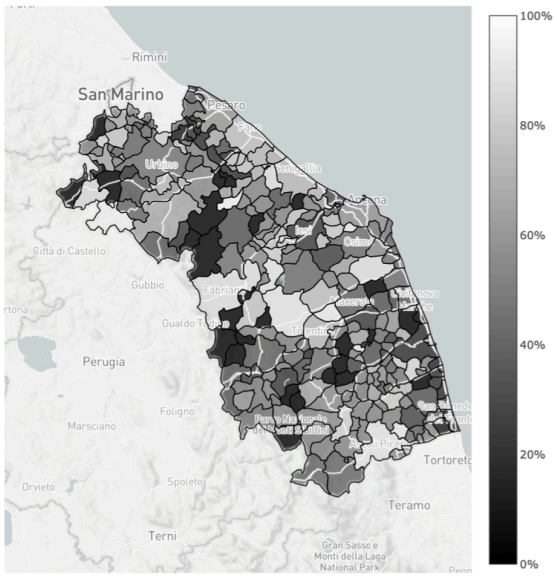

Figure 5.17: Data Recency Metric

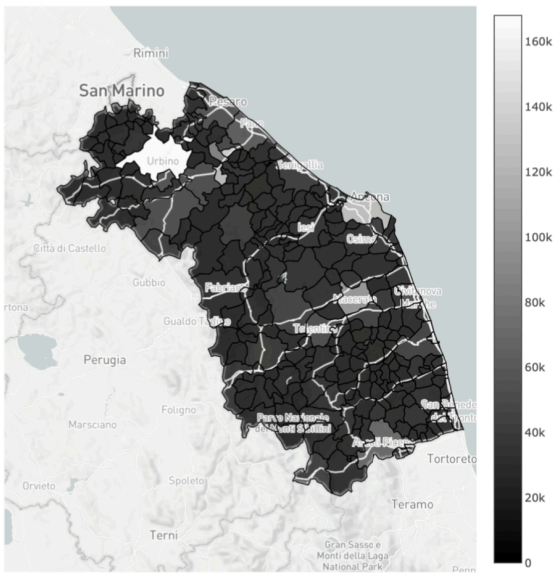

Figure 5.18: Data Quantity Metric

5.5.1 Crossed Visualisation

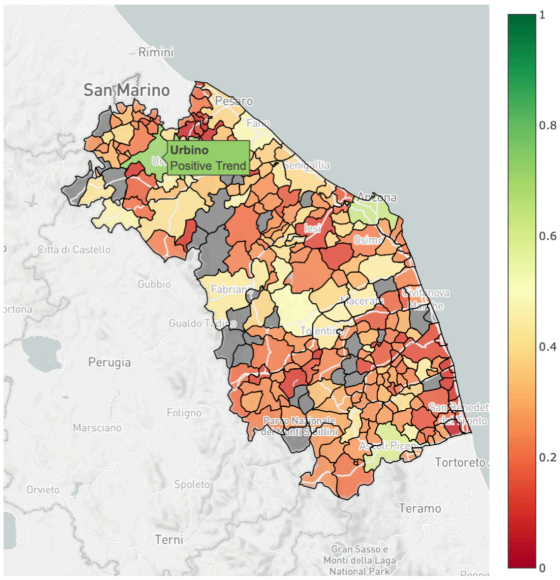

Figure 5.19: Crossed Metrics Result

So, the three metrics alone cannot explain the data correctly, given that the results may be in contrast to each other or not very useful if considered atomically.

Hence the desire to combine the three results in a unique solution that can better describe the situation of the data in the area, allowing the identification of the best records that accomplish the goal: value.

Quantity and recency are information that can be easily combined with an aver-

age, perhaps weighting them according to a certain criterion. As for the trend, the combination is not so trivial. Trend information is all about saying if the collection over time is increasing or decreasing, therefore it would be a mistake to make a simple average. However, It is possible to pair it with the result, for example by showing It as information added to the display produced by quantity and recentness.

For this reason, the final solution consists of the combination of quantity and recency, leaving the task of inferring the trend through a pop-up window, as shown in Figure 5.19. The areas that show a color closer to green are those where the data is more numerous and recent while the increasingly red ones are those in which the data are few and obsolete. Exceptions are the grey areas that instead show a lack of information.

Chapter 6

Experiments And Discussion

Several problems have been detected during the implementation and they are now discussed. The implementation has brought out the limitations that Big Data causes on systems with limited resources. This was already budgeted and It is topic of experiments and discussions in order to achieve the goal.

In fact, from the proposed approach the two critical points emerge which are largely the subject of all Machine Learning problems. Tackling Pre-Processing and Processing. Pre-processing is always the most operational one, which requires the ability of the Data Scientist to use different tools to prepare the data necessary for the processing to be performed. Processing is increasingly oriented to the production of results, therefore oriented to the exploitation of previously refined data during Pre Processing.

On the basis of this, two very interesting discussions emerge on the basis of the work: one oriented on solving the problem represented by data volume, the other one relative to the value identification by applying the proposed approach.

6.1 Local vs Cluster Solution

The implementation on Chapter 5 was developed locally and restricted to the Marche region for demonstration purposes only. The processing took 3 hours locally for its complete termination with the following machine:

- MacBook Pro i7 Haswell QuadCore 2.5 Ghz
- 16 GB RAM
- 512 SSD

Spark is not meant to be used locally, at least during production. This means that its use in a local machine is just a great way to test algorithms before they are brought to a distributed system.

When used locally, Spark automatically divides the resources for the correct execution of normal operations among the available workers, which are generally the machine's CPU cores. It is clear that such a system is affected by traffic on the machine relative to I/O operations, network traffic and the availability of the machine's computational power. Therefore in continuous competition for system resources and not usable for serious processing.

However its usefulness is effective: through Spark in a local machine It is possible to access data indiscriminately from their size. As illustrated in chapter 5, the availability of RDDs (Resilient Distributed Database) allows you to load a very large data set even on a local machine and easily explore it. This is due to the nature of the RDD itself which does not load the entire data set into memory but rather simple pointers to the data.

Being able to display large data in a text file containing unstructured data much larger than the available RAM is not new, Spark has not helped change the world in this sense. UNIX functions like "grep" already did their work in the past (and present too). In any case Spark makes everything much more concrete and close to the user, enabling languages such as Python and entering evaluation opportunities directly into a single environment.

Therefore, the local solutions were very useful first of all for the experimentation and optimisation. This way of using Spark led to the following results in terms of execution times relative to the size of the problem:

Problem	**Processing Time**
Municipality	~2 minutes
Region	~8 hours
Nation	~9 days

Table 6.1: Local Execution Results

The use of the system locally has its obvious limitations. The records in the data set available are around 18 million with 16 columns, a number that is not too high. Performing the Pre Processing phase in these terms is unthinkable, especially considering that in general these could be even larger.

6.1.1 Local Optimisation

Analysing the algorithm that deals with the Pre Processing from a higher level of abstraction It could be summarised as in Figure 6.1. Once the data set is loaded into the RDD, It is shredded. The caching operation is very laborious, but once executed It allows to process the data with very high speed, precisely because they are more easily accessible by the driver who does not have to repeat the transfor-

mations whenever he needs data contained in the RDD.

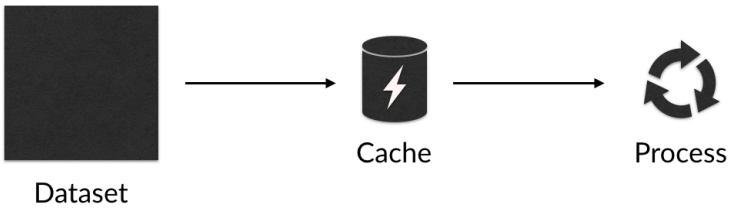

Figure 6.1: Pre Processing Algorithm Core

However, data processing is as onerous as the data set in which to search is larger, even if when cached. The number of operations performed for pre-processing can be easily estimated as:

$$number\ of\ records \times number\ of\ municipalities$$

In other words, billions and billions of operations considering that the available records are more than 20 million and only Italian municipalities are about 7998. In practice, every time the Pre-Processing activity is started, or whenever a spatial query is made, identifying whether the record considered is contained within the boundaries of a municipality or not is like looking for a needle in a haystack, from here to optimisation: reduce haystack.

Reducing the haystack means reducing the size of the cached data set in such a way that the geo-spatial identification operation is easier. This is possible through the decomposition of the original RDD into smaller RDDs processed from time to time, as shown in figure 6.2.

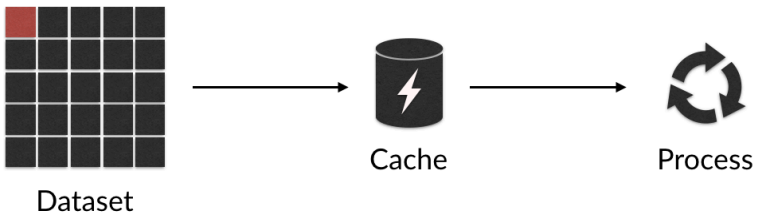

Figure 6.2: Optimised Pre Processing Algorithm Core

This type of processing has several critical points:

1. Reducing the data set implies the need for multiple caching, a rather slow operation;

2. The operations on the various RDDs are executed sequentially, so every time a new RDD is to be processed, It is necessary to wait until the previous one has finished processing.

However, reducing the problem to the regions, generating 20 RDDs (Italy has 20 regions) the results were remarkable, as shown in table 7.2.

Problem	Processing Time
Municipality	~40 seconds
Region	~3 hours
Nation	~3 days

Table 6.2: Optimised Local Execution Results

However, 3 days for Pre Processing are still extremely high but It is possible to bring the solution into a cluster.

6.1.2 Cluster Implementation

The cluster implementation does not differ from the one presented in the previous chapter. In the previous chapter the optimisation was not given and the Google API are not there for obvious reasons.

The machine chosen for the final solution is a cluster on Google DataProc with the following configuration:

Nodes	3
Workers	2
CPU	48 Core
RAM	96 GB
DISK	150GB

Table 6.3: DataProc Cluster Configuration

The considered machine is clearly superior to the machine used locally. The advantages deriving from the execution on clusters of the Pre Processing algorithm are obvious: Spark has a much greater amount of resources than local execution. It can run faster.

The choice of only 3 nodes, where one of these acts as a Master, is dictated by the fact that Spark is much more powerful when the cluster scales vertically. As a result, few nodes but as a high computing power provide the perfect solution to solve the problem.

In fact, increasing the number of nodes, would only increase the bottleneck on the network of data necessary for processing. In all likelihood a configuration with

a greater number of nodes would have given better results, but also greater costs, for this reason it was preferred to test immediately the vertical scale.

The algorithm (whose complete code is visible in the appendix) changes as follows:

The files are read directly from Google Cloud Storage, which is a persistent storage offered by Google since the Cluster is removed when the cluster is deleted.

```
from google.cloud import storage client = storage.Client()

bucket = client.bucket('dataproc-2earbb1b-9101-4127-281s-a3ff37ebb8a2-us-east1')
blobs = list(bucket.list_blobs(prefix='GeoJSON/'))
target_blob = blobs[1] # read as string
read_output = target_blob.download_as_string()
```

Listing 6.1: Read GeoJSON From Bucket

The data is loaded for processing as seen in the previous chapter (they are not shown again for obvious reasons) and the algorithm that executes what was described in section 6.1.1 is started.

```
first = True

for region in regions:
  header = data.first()
  clean = data.filter(lambda row: row != header)\
                .row: row.split(","))\
                .map(lambda row: [row[0]]+[row[6]]+[str(convert_date(row[4]))]+[coordi .filter(lambda row: row[3] != "ERROR")\
                .filter(lambda row: isInside(row[3], region_borders[region]) == True)\
                .repartition(32)\
                .cache()

  plots_data = []
  for mun, polygon in tqdm(reg_poly[region]):
    quantities = clean.filter(lambda row: isInside(row[3], polygon) == True)\
                        .map(lambda row: (row[2], 1))\
                          .reduceByKey(lambda x,y: x+y)
                          .collect()
      plots_data.append((mun, quantities))
```

```

  if first:
    d = {}

    for x,y in plots_data:
      dates = {}
      for elem in y:
        dates[elem[0]] = elem[1]
      d[x] = dates

    with open('data.json', 'w') as outfile:
      json.dump(d, outfile)

    first = False
  else:
    with open('data.json', 'r') as p: x = json.load(p)

     = {}
    for a,b in plots_data:
      dates = {}
      for elem in b:
        dates[elem[0]] = elem[1] y[a] = dates
      d = merge_two_dicts(x, y)

    with open('data.json', 'w') as outfile:
      json.dump(d, outfile)
```

Listing 6.2: Start Processing

With the previous step all the municipalities are processed region by region. This reduce a lot the time required for the processing. Finally, the results are moved from the cluster to the Google Cloud Storage again.

```
from google.cloud import storage

def upload_blob(bucket_name, source_file_name,
   destination_blob_name):
  """Uploads a file to the bucket."""
  storage_client = storage.Client()
  bucket = storage_client.get_bucket(bucket_name)
  blob = bucket.blob(destination_blob_name)
  blob.upload_from_filename(source_file_name)
  upload_blob('dataproc-2earbb1b-9101-4127-281s-
     a3ff37ebb8a2-us-east1, 'data.json', 'data_Italy.
     json')
```

Listing 6.3: Upload Results to Google Cloud Storage

Moving the processing on cluster has reduced the time by at least 70%, processing one region in about 20 minutes at the cost of 3.50 euros per hour. In the table 6.4 these new performances are shown.

Problem	Processing Time	Cost
Municipality	~5 seconds	~> 1 cent
Region	~20 minutes	~1 euros
Nation	~7 hours	~20 euros

Table 6.4: Cluster Execution Results and Costs

6.2 Metrics

The results obtained following the implementation of the metrics and cross-referencing the data suggest the areas highlighted in the Figure 5.20 (Chapter 5). In the following section the results are discussed.

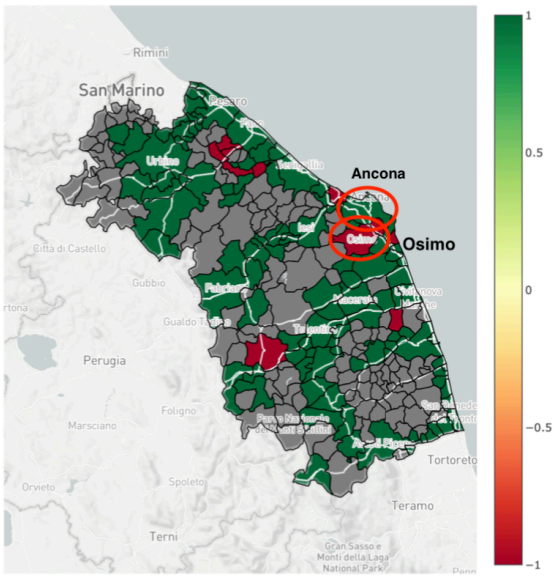

Figure 6.3: Ancona and Osimo Municipalities Study Case

The three metrics used were: **usage trend**, **data quantity** and **data recency**. For the first one, the results are shown in Figure 6.3. They have been already explained in the previous chapter but It is time to go deeper, taking two municipalities as examples: Ancona and Osimo, two diametrically opposed case studies since the first shows a positive trend while the second one a negative one.

Before starting, It should be noted the total absence of the yellow color that should represent the stability of the trend. The reason why this happens is related to the definition of stability as a horizontal line. It is obvious that in the real world this case is pretty difficult to obtain. Maybe would have been useful to define a confidence interval within which the slope of the trend would have been considered horizontal. However this is part of the design of the metric and will not be discussed.

So, in Figure 6.4 the data distributions over time (deprived of anomalies) are shown. The two municipalities show consistent trends with the predefined color scheme (green and then positive for Ancona; red and then negative for Osimo).

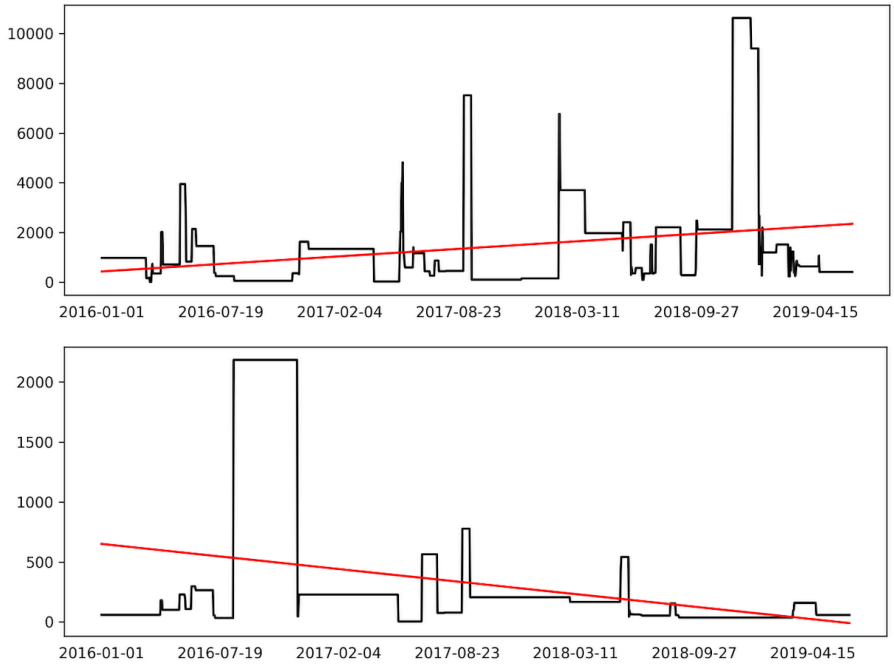

Figure 6.4: Ancona (on top) and Osimo (bottom) Municipalities Trends

Regarding recency and quantity, they can be considered together. For both the metrics the same color scale has been chosen. In this case the correctness of the metrics should be investigated as well as the consequences of merging them together. When a municipality is recent with a big amount of data, It is expected that the graphs should show more collected data on Its right half, or viceversa. For this reason the municipalities of Fano and Macerata have been chosen, shown in Figure 6.5.

The previous assumption is verified in Figure 6.6. In the results, It is possible to verify that the number of samples collected and the fact that the data is recent is effective. They both show the same quantity of data collected but Fano municipality is more recent in respect to Macerata and this is reflected in how the data are distributed over time.

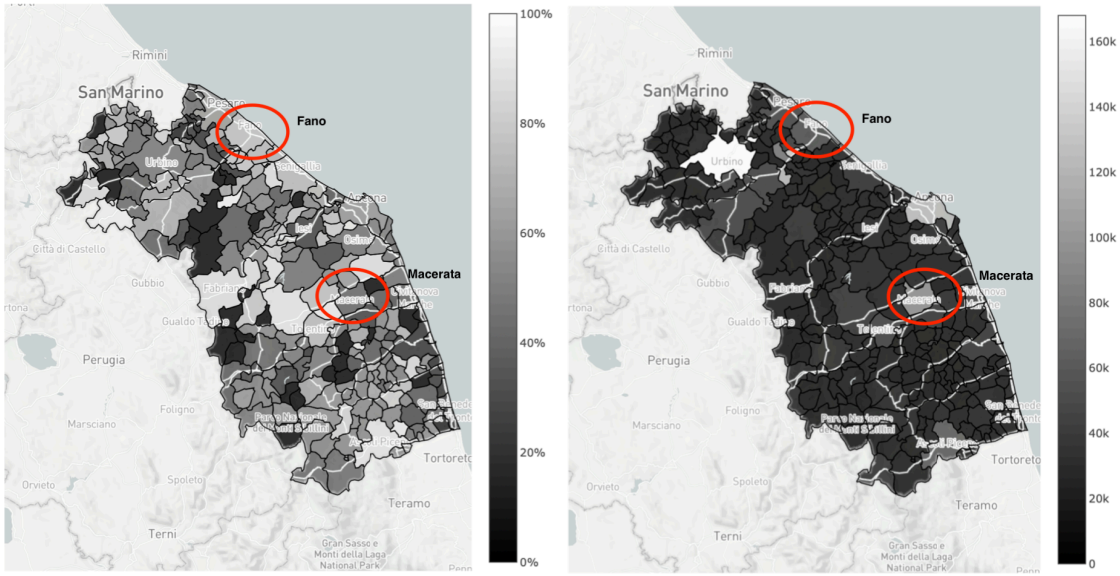

Figure 6.5: Fano/Macerata municipalities with their recency and quantity

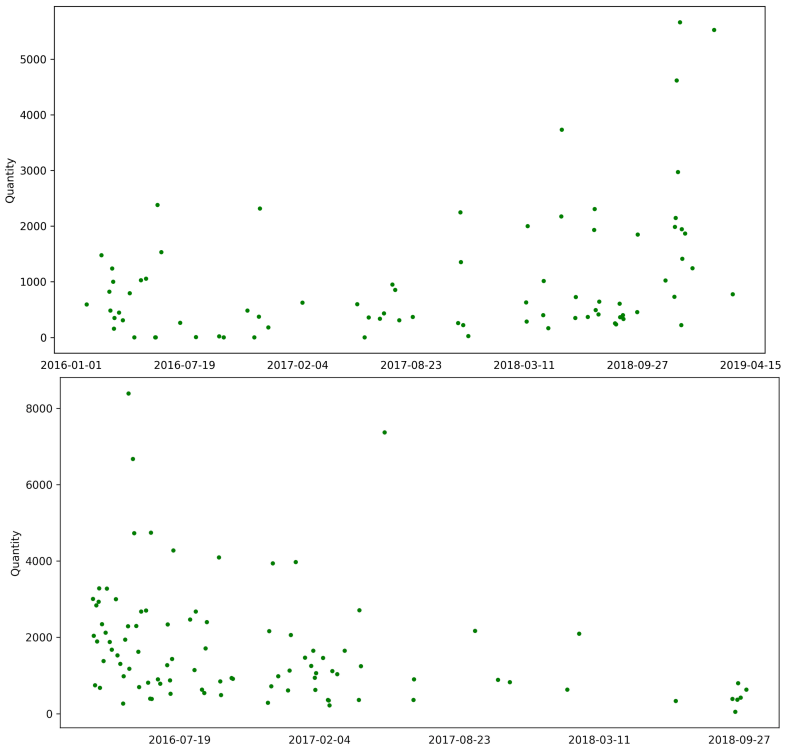

Figure 6.6: Fano (on top) and Macerata (bottom) Municipalities Recency and Quantity

As a final comment, in Figure 6.5 It is evident to refer above all to the case of Urbino, in which the quantity of data is excessively high compared to the neighbouring municipalities. Although this was predictable, It is interesting to underline the potential of the ability of designing metrics with this approach. Choosing a

metric means having the freedom to design it as best suits it: in this case the municipality of Urbino has a much greater impact in terms of data than the rest of the territory, risking to hide in a certain sense the potential of neighbouring municipalities and so, the programmer could be free of weighting (or down weighting) the collected quantity in respect to others in some way bu just re-design the metric.

In any case, the final visualisation permits to generate an overall visualisation able to suggest more concretely which are the best areas. And in fact, by normalising the quantity values and mediating them with the recency, the final result is visible in Figure 6.7. The legend from Red to Green shows the best (green) and the worst (red) municipalities in which to look for valuable data.

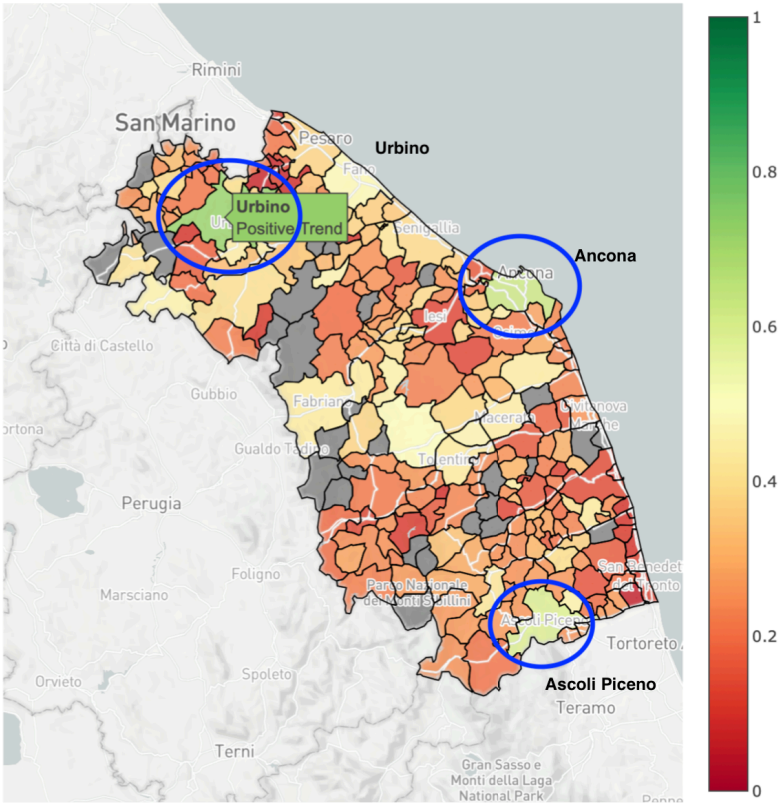

Figure 6.7: The municipalities of Urbino, Ancona and Ascoli Piceno

The best municipalities seems to be those of Urbino, Ancona and Ascoli Piceno in the Marche region. The roads belonging to these three municipalities are approximately 11900 and must be investigated. Following the exploration of the data in these areas different roads emerge with a quantity of data in the time necessary to carry out subsequent studies. Some of these are shown in Figure 6.8.

These results suggest that the approach used worked as expected and allowed to restrict the focus to the most important data, the ones able to generate value based on the metrics chosen. The approach guaranteed the expected results.

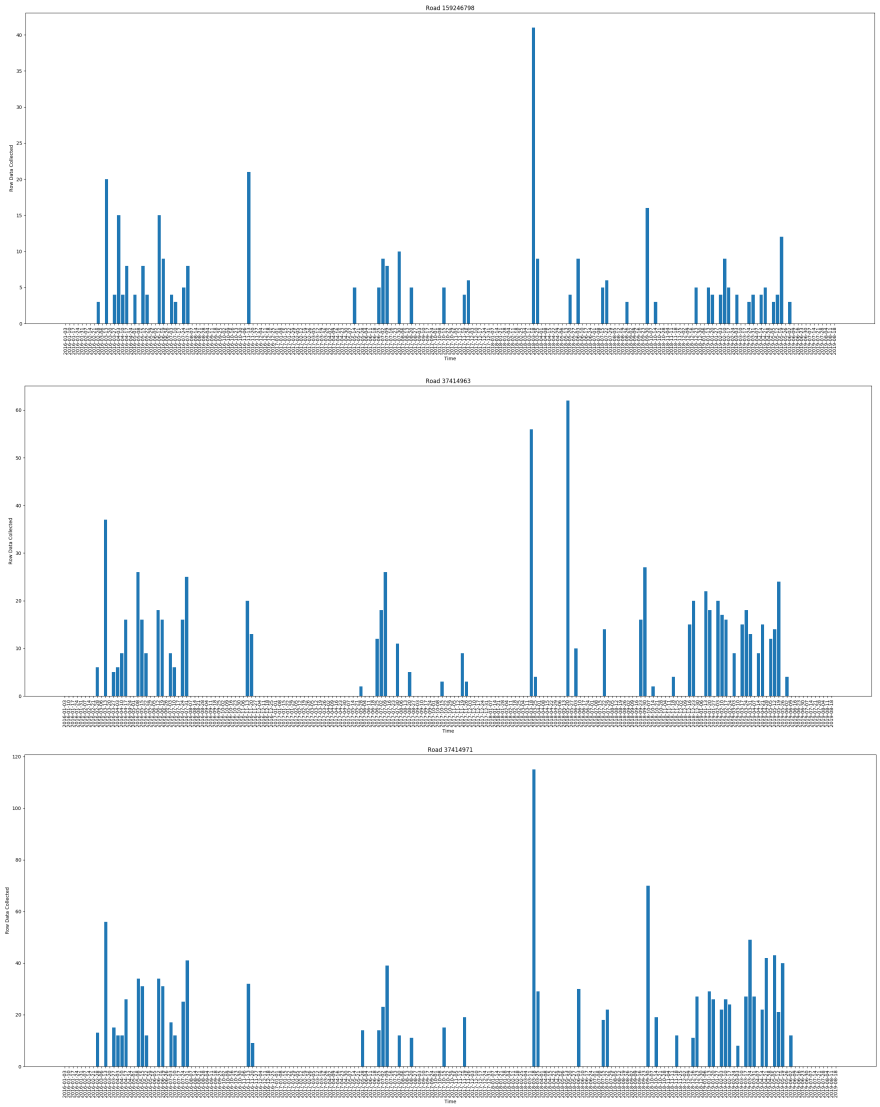

Figure 6.8: Results found applying the approach

Chapter 7

Conclusions

During the dissertation several challenges and many different variables have been faced, most of them critical to the final development. The data produced by SmartRoadSense has proven to be really interesting but a bit hard to handle. The amount of data available is pretty big considering the ages of the project but still It seems not enough to be used for more complex challenges.

This has been reflected on the work carried out with this thesis. In the very beginning of this work, It was thought to use the PPE information of road roughness for forecasting. But as an evidence of this work, that idea has been proven to be immature and distant from being able to take place since the difficulty to find valuable data for that scope is actually challenging. Perhaps, even if some studies on specific use cases may also be possible, the lack of a constant amount of data available over time is actually a huge problem, so important that the SRS team is currently planning to introduce a game into the application itself as an attempt to entertain children in cars and to push them into use. However, this problems have been repeatedly underlined along this work as the need for correct strategies for the value identification.

This suggested to take a look deeply on how to identify regions of data where the information carried is meaningful. The proposed approach and techniques used could be applied to any geospatial data set in which we would like to avoid the use of SQL, for example when the data is unstructured or placing a geospatial database is not a choice. Furthermore, the use of massively processing tools gives the freedom to the programmer to write simple scripts that can work stand alone that nowadays represents a big advantage in data analytics due to the availability of clusters on demand.

The main difficulties faced have been related to the problem of *volume* handling. During the Pre-Processing phase a big amount of data was taken into consideration. Like usually happens in the field of Data Science, most of the time is spent during this phase. The limits deriving from the analysis of a large data set has required further efforts to guess which solution to adopt if the data set continues its vertical growth. The case of SmartRoadSense is in fact an excellent case representative of what happens when collecting data from the "crowd" in this sense. The amount of

data is so large and tends to grow so quickly that the solutions adopted today may no longer be sufficient tomorrow and must be scaled accordingly. In this sense, the solution found that allows to scale the problem with a cluster-based solution has definitively been a winning choice. Furthermore, this suggested that the system could be updatable rather than working in batch, avoiding recomputing largely the same data every time.

As for the solution regarding the ability to identify records able to generate value, instead, excellent results have been obtained. Through the selection of multiple metrics and their combination, It was actually possible to identify the portion of the data set with the best data for generating the value needed, most continuous over time and in large quantity. However some techniques used should be replaced with more precise and advanced techniques, such as those used for the identification of anomalies, which currently consist of simple low pass filters and could be replaced with more effective and precise machine learning techniques.

For this reason this work will not end here. At this time, further activities are planned such as:

- Introduction of techniques in the field of machine learning for the removal of anomalies in the time series
- Automation of the analytics process to work in real-time

These acts should allow for a more streamlined and complete application that can automatically suggest the best data in which to operates with the final goal of picking the best one that can generate the value needed.

Appendix A

Appendix

A.1 Jupyter Notebook on Google DataProc

```
import datetime
from tqdm import tqdm_notebook as tqdm
from pyspark import SparkContext, SparkConf
from shapely.geometry import Point
from shapely.geometry.polygon import Polygon
import math
from dateutil.parser import parse
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from scipy.interpolate import interp1d
import json
from shapely import wkb
```

Listing A.1: Import Libraries

```
sc
sc.defaultParallelism
```

Listing A.2: View SparkContext

```
from google.cloud import storage client = storage.Client
    ()

bucket = client.bucket('dataproc-2earbb1b-9101-4127-281s
    -a3ff37ebb8a2-us-east1')
blobs = list(bucket.list_blobs(prefix='GeoJSON/'))
target_blob = blobs[1] # read as string
read_output = target_blob.download_as_string()

```

```
geojson = json.loads(read_output)
regions = []
for elem in geojson['features']:
  if elem['properties']['COD_REG'] not in regions:
    regions.append(elem['properties']['COD_REG'])
```

Listing A.3: Read GeoJSON From Bucket

```
data = sc.textFile('gs://dataproc-2eecbb1b-9211-44d7-86d5-a3ff37ebb8a2-us-east1/data/...'
        .map(lambda line: line.encode('ascii', 'ignore'))
header = data.first()
for i, elem in enumerate(header.split(",")):
  print "Index:", i, "Column:", elem data.take(2)
```

Listing A.4: Load Data

```
def coordinates(hex_location):
  try:
    point = wkb.loads(hex_location, hex=True)
    return str(point.x)+";"+str(point.y)
  except:
    return "ERROR"

def convert_date(date):
  try:
    result = datetime.datetime.strptime(date, '%Y-%m-%d %H:%M:%S').date()
  except:
    result = datetime.datetime.strptime(date, '%Y-%m-%d %H:%M:%S.%f').date()
  return result

def isInside(location, polygon):
  point = Point(float(location.split(";")[0]),float(location.split(";")[1]))
  for elem in polygon:
    if elem.contains(point): return True
  return False
```

Listing A.5: Define functions for Processing

```
target_blob = blobs[2]
read_output = target_blob.download_as_string()
geojson_regions = json.loads(read_output)

region_borders = {}
```

```
for elem in geojson_regions['features']:
  boundaries = []
  for poly in elem['geometry']['coordinates']:
    boundaries.append(Polygon(poly[0]))
  region_borders[elem['properties']['COD_REG']] =
     boundaries

reg_poly = {}
for region in regions:
  municipalities = []
  for elem in geojson['features']:
    if elem['properties']['COD_REG'] == region:
      municipalities.append(elem)

  polygons = []
  for municipality in municipalities:
    boundaries = []
    for poly in municipality['geometry']['coordinates']:
      boundaries.append(Polygon(poly[0]))
      polygons.append((municipality['properties']['
         COMUNE'], boundaries))

  reg_poly[region] = polygons

def merge_two_dicts(x, y):
  z = x.copy() # start with x's keys and values
  z.update(y) # modifies z with y's keys and values &
     returns None return z
```

Listing A.6: Take Regions Borders

```
first = True

for region in regions:
  header = data.first()
  clean = data.filter(lambda row: row != header)\
                .row: row.split(","))\
                .map(lambda row: [row[0]]+[row[6]]+[str(
                   convert_date(row[4]))]+[coordi .
                   filter(lambda row: row[3] != "ERROR")
                   \
                .filter(lambda row: isInside(row[3],
                   region_borders[region]) == True)\
                .repartition(32)\
                .cache()

  plots_data = []
```

```
13    for mun, polygon in tqdm(reg_poly[region]):
14      quantities = clean.filter(lambda row: isInside(row
           [3], polygon) == True)\
15                          .map(lambda row: (row[2], 1))\
16                            .reduceByKey(lambda x,y: x+y)
17                            .collect()
18        plots_data.append((mun, quantities))
19
20    if first:
21      d = {}
22
23      for x,y in plots_data:
24        dates = {}
25        for elem in y:
26          dates[elem[0]] = elem[1]
27        d[x] = dates
28
29      with open('data.json', 'w') as outfile:
30        json.dump(d, outfile)
31
32      first = False
33    else:
34      with open('data.json', 'r') as p: x = json.load(p)
35
36       = {}
37      for a,b in plots_data:
38        dates = {}
39        for elem in b:
40          dates[elem[0]] = elem[1] y[a] = dates
41        d = merge_two_dicts(x, y)
42
43      with open('data.json', 'w') as outfile:
44        json.dump(d, outfile)
```

Listing A.7: Start Processing

With the previous step all the municipalities are processed region by region. This reduce a lot the time required for the processing.

```
1  from google.cloud import storage
2
3  def upload_blob(bucket_name, source_file_name,
       destination_blob_name):
4    """Uploads a file to the bucket."""
5    storage_client = storage.Client()
6    bucket = storage_client.get_bucket(bucket_name)
7    blob = bucket.blob(destination_blob_name)
```

```
8   blob.upload_from_filename(source_file_name)
9   upload_blob('dataproc-2earbb1b-9101-4127-281s-
       a3ff37ebb8a2-us-east1', 'data.json', 'data_Italy.
       json')
```

Listing A.8: Upload Results to Google Cloud Storage

A.2 Anomalies Detection

```
1  def moving_average(data, window_size):
2  """ Computes moving average using discrete linear
       convolution of two one dimensional
3    Args:
4      data (pandas.Series): independent variable
5      window_size (int): rolling window size
6    Returns:
7      ndarray of linear convolution
8  """
9    window = np.ones(int(window_size))/float(window_size)
10   return np.convolve(data, window, 'same')
11
12 def explain_anomalies(y, window_size, sigma=1.0):
13 """ Helps in exploring the anamolies using stationary
       standard deviation
14   Args:
15     y (pandas.Series): independent variable
16     window_size (int): rolling window size
17     sigma (int): value for standard deviation Returns:
18     a dict (dict of 'standard_deviation': int, '
          anomalies_dict': (index: value))
19 """
20   avg = moving_average(y, window_size).tolist()
21   residual = [y_i - avg_i for y_i, avg_i in zip(y, avg)]
22
23   # Calculate the variation in the distribution of the
       residual
24   std = np.std(residual)
25   return {'standard_deviation': round(std, 3), '
       anomalies_dict': collections.OrderedDict([(index,
       y_i) for index, y_i, avg_i in izip(count(), y, avg)
        if (y_i > avg_i + (sigma*std)) | (y_i < avg_i - (
       sigma*std))])}
```

Listing A.9: Function Definitions for Anomalies Detection

A.3 Data Quantity

```
quantity_d = {}
for key, value in tqdm(sorted(municipalities_d.iteritems
    ())):
  collected = 0
  for k, v in value.iteritems():
    collected += v quantity_d[key] = collected
```

Listing A.10: Defining Quantities

A.4 Data Recency

```
# Generating Interval
minimum = datetime.date(2016, 1, 1)
today = datetime.date.today()
width = today - minimum
width.days

# Generating Recency Values
recency_d = {}
for key, value in tqdm(sorted(trend_d.iteritems())):
  collected = 0
  for k, v in value.iteritems():
    result = recency/collected recency_d[key] = result
```

Listing A.11: Defining Recency

A.5 Usage Trends

```
# Generating Trend Values
def trendline(data, order=1):
  coeffs = np.polyfit(data.index.values, list(data),
     order)
  return [float(elem) for elem in coeffs]

trend_d = {}
for key, value in tqdm(sorted(trend_d.iteritems())):
  df = pd.DataFrame(list(sorted(value.iteritems())),
     columns=['year','quantity'])
  df_ffill = df.ffill()
  coeffs = trendline(df_ffill['quantity'],1)
  slope = coeffs[-2]
  trend_d[key] = slope
```

Listing A.12: Defining Trend

Acronyms

C4Rs CrowdForRoads

The Crowd4Roads project combines trip sharing and crowd sensing initiatives to harness collective intelligence to contribute to the solution of the sustainability issues of road passenger transport, by increasing the car occupancy rate and by engaging drivers and passengers in road monitoring.
www.c4rs.eu

IoT Internet Of Things

The Internet of things (IoT) is the extension of Internet connectivity into physical devices and everyday objects. Embedded with electronics, Internet connectivity, and other forms of hardware (such as sensors), these devices can communicate and interact with others over the Internet, and they can be remotely monitored and controlled.
www.en.wikipedia.org

MCS Mobile Crowd Sensing

Mobile Crowdsensing, is a technique where a large group of individuals having mobile devices capable of sensing and computing (such as smartphones, tablet computers, wearables) collectively share data and extract information to measure, map, analyze, estimate or infer (predict) any processes of common interest.
www.en.wikipedia.org

SRS SmartRoadSense

SmartRoadSense is a mobile application that uses your smartphones accelerometers and GPS sensor to detect and classify irregularities of the road surface while you are driving.
www.smartroadsense.it

H2020 Horizon 2020

Horizon 2020 is the biggest EU Research and Innovation programme ever with nearly 80 billion euros of funding available over 7 years (2014 to 2020) - in addition to the private investment that this money will attract. It promises more breakthroughs, discoveries and world-firsts by taking great ideas from the lab to the market.
www.ec.europa.eu

Bibliography

References cited in the text

Publications and Manuals

[1] Mark Weiser. "The Computer for the 21st Century". In: *Scientific American*. 1991, pp. 78–89 (cit. on p. 5).

[2] Kevin Ashton. "That 'Internet of Things' Thing". In: *RFID Journal*. 2009 (cit. on p. 5).

[3] Kary Främling et al. "Product agents for handling information about physical objects". In: *Report of Laboratory of Information Processing Science series B, TKO-B 153/03, Helsinki University of Technology*. 2003 (cit. on p. 5).

[4] Mehdi Mohammadi et al. "Deep Learning for IoT Big Data and Streaming Analytics: A Survey". In: *IEEE Communications Surveys and Tutorials*. 2018 (cit. on p. 6).

[5] Rob Kitchin and Gavin McArdle. "What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets". In: *SAGE Journal* (2016) (cit. on p. 7).

[6] Tom Breur. "Statistical Power Analysis and the contemporary crisis in social sciences". In: (2016) (cit. on p. 8).

[7] Raghu Ganti, Fan Ye, and Hui Lei. "Mobile crowdsensing: current state and future challenges". In: *IEEE Communications Magazine* (2011), pp. 32–39 (cit. on p. 8).

[8] Haoyi Xiong et al. "iCrowd: Near-Optimal Task Allocation for Piggyback Crowdsensing". In: *IEEE Transactions on Mobile Computing* (2016) (cit. on p. 8).

[9] Bin Guo et al. "Mobile Crowd Sensing and Computing: The Review of an Emerging Human-Powered Sensing Paradigm". In: *ACM Computing Surveys* (2015) (cit. on p. 8).

[10] Jinwei Liu, Haiying Shen, and Xiang Zhang. "A Survey of Mobile Crowdsensing Techniques: A Critical Component for the Internet of Things". In: *2016 25th International Conference on Computer Communication and Networks*. 2016 (cit. on p. 9).

[11] Raghu Ganti, Fan Ye, and Hui Lei. "Mobile crowdsensing: current state and future challenges". In: *2016 25th International Conference on Computer Communication and Networks*. Vol. 49. 2011 (cit. on p. 9).

[12] M. Hilbert. *Big Data for Development: A Review of Promises and Challenges*. 34 vols. 2016, pp. 135–174. URL: http://doi.org/10.1111/dpr.12142 (cit. on p. 7).

[13] Malamati Louta, Konstantina Mpanti, and Thomas Karetsos Georg end Lagkas. "Mobilecrowdsensing architectural frameworks: A comprehensive survey". In: *2016 7th International Conference on Information, Intelligence, Systems and Applications (IISA)*. 2016 (cit. on p. 10).

[14] Giacomo Alessandroni et al. "Smartroadsense: Collaborative road surface condition monitoring". In: *Proceedings of the UBICOMM*. 2014 (cit. on pp. 1, 16, 19).

[15] Saverio Delpriori. "Mobile Crowd-Sensing: Enabling Technologies and Applications". In: (2018) (cit. on p. 15).

[16] Saverio Delpriori et al. "Efficient algorithms for accuracy improvement in mobile crowdsensing vehicular applications". In: *UBICOMM 2015*. 2015 (cit. on p. 17).

[17] Andrew Crooks et al. *Transactions in GIS*. 2013 (cit. on p. 18).

[18] Gang Pan et al. "Land-use classification using taxi gps traces". In: *IEEE Transactions on Intelligent Transportation Systems*. 2013 (cit. on p. 18).

[19] Dmytro Karamshuk et al. "Geo-spotting: mining online location-based services for optimal retail store placement". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013 (cit. on p. 18).

[20] Vincent W Zheng et al. "Towards mobile intelligence: Learning from gps history data for collaborative recommendation". In: *Artificial Intelligence*. 2012 (cit. on p. 18).

[22] Yu Zheng et al. "Recommending friends and locations based on individual location history". In: *ACM Transactions on the Web* (2011) (cit. on p. 18).

[23] Yu Zheng, Furui Liu, and Hsun-Ping Hsieh. "U-air: When urban air quality inference meets big data". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2013 (cit. on p. 19).

[24] Yohan Chon et al. "Automatically characterizing places with opportunistic crowdsensing using smartphones". In: *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. 2012 (cit. on p. 18).

[25] Lukas Ruge, Bashar Altakrouri, and Andreas Schrader S. "Soundofthecity - Continuous noise monitoring for a healthy city". In: *IEEE International Conference on Pervasive Computing and Communications*. 2013 (cit. on p. 19).

[26] Nicolas Maisonneuve, Matthias Stevens, and Bartek Ochab. "Participatory noise pollution monitoring using mobile phones". In: *Information Polity*. 2010 (cit. on p. 19).

[27] Neil M. Ferguson et al. “Strategies for mitigating an influenza pandemic”. In: *Nature* (2006) (cit. on p. 19).

[28] Amy Wesolowski et al. “Quantifying the impact of human mobility on malaria”. In: *Science* (2012) (cit. on p. 19).

Online Materials

[21] The Economist. *Data, data everywhere*. 2010. URL: https://www.economist.com/special-report/2010/02/27/data-data-everywhere (cit. on p. 7).

[29] Wikipedia. *Big data*. 2019. URL: https://en.wikipedia.org/wiki/Big_data (cit. on p. 7).