POLITECNICO DI TORINO

Master of science
ICT FOR SMART SOCIETIES

Master's Degree thesis

Big Data analysis of Floating Car Data to identify traffic congestion in urban areas

Supervisors
Prof. Danilo Giordano
Prof. Luca Vassio
Prof. Marco Diana

Candidate
Seyedamirmohammad Sakaki

October 2022

Summary

The inescapable phenomenon of traffic congestion has been steadily worsening over the last several decades, particularly in big urban regions that have been experiencing population growth all over the globe. In its most basic form, congestion occurs when there is a larger demand for space than there is capacity on the roadway to accommodate it. However, these activities, which force people to interact with one another, are necessary for the effective operation of economic systems and cannot be avoided. In today's societies, these demands are frequently generated at the same time, while a large number of people intend to commute to school or their workplace as well as other movements to provide goods and services. Although there is a potential for congestion with every method of transportation, the emphasis of this thesis will be on the congestion that occurs with automobiles on public highways and streets.

The identification of regions that are prone to traffic congestion has been and will continue to be an essential step in the process of developing urban transportation systems. The technologies used to gather data on traffic have seen significant development over the last several years, and access to real-time traffic information is increasingly becoming the norm around the globe. Floating Car Data (FCD) is one of the most well-liked ways, and it has been more widespread in usage over the last several years. FCD is often timestamped geo-localization and speed data that is directly acquired by moving cars utilizing "in-vehicle" devices via mobile phones or GPS. These devices are connected to the internet. Because of the growing quantity of data that are acquired by FCD, it is becoming more desired to find and extract useful traffic-related information from the accumulated historical dataset. Examples of this kind of information include patterns of congestion. Despite this, it is not an easy task since the dataset is rather large, and the characteristics of the traffic, such as its complexity and dynamics, make it quite difficult.

Using Data-Driven approaches and an already-collected large FCD dataset, this thesis aims to identify traffic congestion areas in an urban area. This will be accomplished by extracting the desired meaningful information for the purpose of the research from a large FCD dataset. The collected FCD includes information spanning the entirety of 2019 and includes millions of records from 290,185 distinct vehicles, of which 92% are declared to be personal vehicles and the remaining 8% fall under the category of fleet vehicles. In addition, the vast majority of the data are collected at intervals of one minute for each vehicle. The FCD were

collected over the course of the 2019 calendar year. From these data, the segments of the trips of distinct cars are extracted. Thanks to a grid map consisting of cells that are each uniquely identified, the extracted segments are assigned to their corresponding cells such that each cell could provide key required information for traffic congestion calculation at the next step. In this project, the relative and absolute traffic congestion of morning and evening peak hours compared to off-peak hours is discussed and analyzed based on the average speed (in kilometers per hour) of vehicles on their respective segments of the route. Finally, the traffic congestion severity is extracted and polished using Kernel Density Estimation to demonstrate in a convenient way, how much different zones of the urban area are suffering from traffic congestion 1, so the future transportation system developments could be planned accordingly.

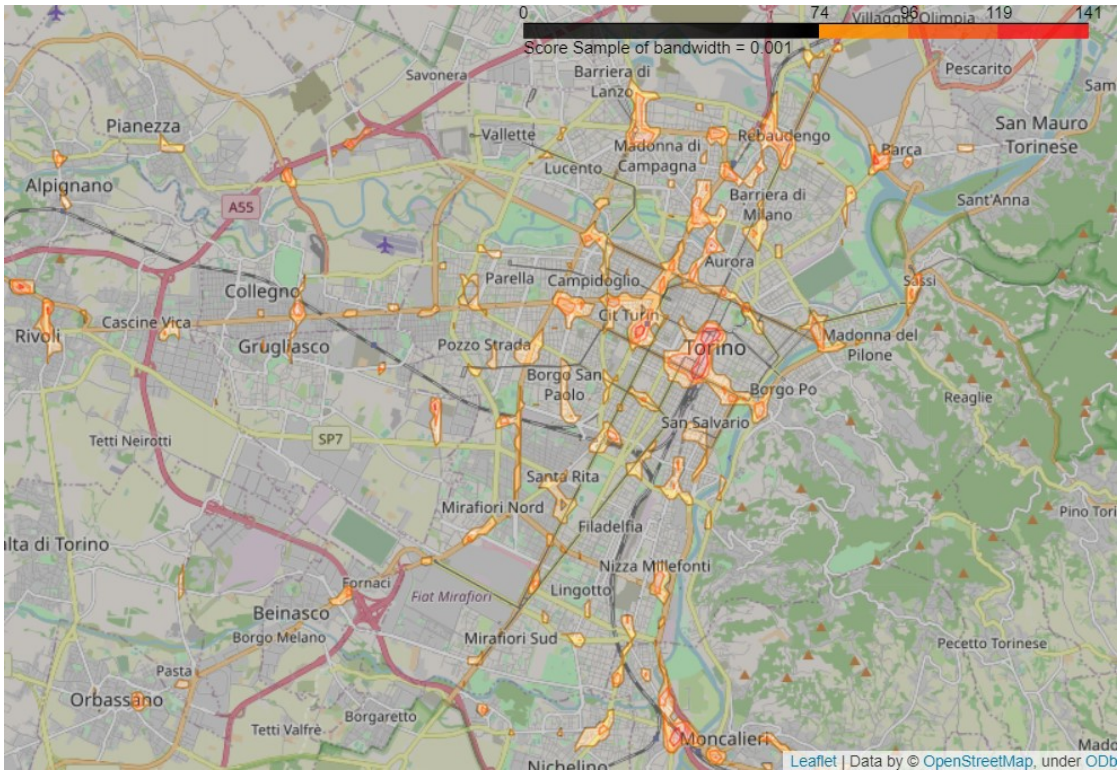

Figure 1: Highlight of the most congested areas, Bandwidth = 0.001

Acknowledgements

My studies at Politecnico di Torino with the purpose of acquiring a Master of Science (MSc) degree in ICT for smart societies have come to a close with the completion of this master thesis report. During the time that I was working on my thesis, I went through an experience that was challenging, but more than that, it was intellectually engaging. Without the substantial and unwavering aid and guidance provided by a number of different persons, it would not have been feasible to do this project. To begin, I would like to offer my most sincere gratitude to Professor Giordano at Politecnico di Torino, who serves as my principal advisor for providing comments on my work and aid me in finding solutions to the various problems I encountered throughout the development process. I also would like to express my sincere appreciation to professors Vassio and Diana for the direction they provided me in determining the objectives of my study and how to assess strategy.

Seyedamirmohammad Sakaki

Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

FCD
Floating Car Data

IoT
Internet of Things

GPS
Global Positioning system

GSM
Global System for Mobile communications

GPRS
General Packet Radio Service

UMTS
Universal Mobile Telecommunications System

ITS
Intelligent Transportation Systems

WGS
World Geodetic System

GDOP
Geometric Dilution Of Precision

HDOP

Horizontal Dilution Of Precision

GNSS

Global Navigation Satellite System

OTM

Open Transport Map

RTMS

Remote Traffic Microwave Sensor Data

LOS

Level Of Service

KDE

Kernel Density Estimation

Chapter 1

Introduction

1.1 Traffic congestion as a growing problem

Congestion is a condition in traffic flow in which an increased volume of traffic on a highway, or "throughput" facility (such as a tunnel), leads to greater queue lengths and therefore increased travel time, lower average speed and longer trip times. Congestion occurs when traffic demand is high enough that interactions among vehicles reduce the velocity of the traffic stream. Since the 1950s, there has been a significant increase in traffic congestion on metropolitan road networks, [**1**] especially during rush hour. Extreme traffic congestion occurs when demand exceeds the capacity of a road (or of the junctions along the road), that leads to a condition under which the vehicles are completely halted for a period of time [**2**] [**3**].A traffic jam can last anywhere from a few minutes to a several hours.

As the traffic congestion tends to be more severe year by year, there are some major reasons causing this increase. The most obvious one is population expansion. More people in an affluent country implies more vehicles; However, overall vehicle kilometers have increased far faster than the population. For instance, Between 1980 and 2000, the overall population of the United States increased by 24%, while total car kilometers driven increased by 80% due to more intensive usage of each vehicle [**4**]. Other main reasons as [**5**] mention, are the following:

- Economic growth
- Increase in the number of vehicles
- Insufficient road capacity
- Ineffective city planning and management
- Accidents

- Natural disasters

1.2 Traffic data collection methods

In recent decades, different approaches to traffic data collection have been employed to collect and analyze relevant data to extract congestion patterns in order to manage and decrease traffic. Data analytic has proven to be crucial in determining various causes of congestion based on data from sensors and cameras placed on roads, GNSS data collected from cars, cell phone signal strength in an area, which also reflects amount of cars and traffic around it, etc. In general there are two principal methods of traffic data collection including data collection by surveys and data collection by using the records of some specific devices [**6**]. The latter one is known as 'in-situ' method and itself is consist of two sub methods of intrusive and non-intrusive [**7**]. The intrusive approaches essentially consist of a data recorder and a sensor placed on or in the road:

- Pneumatic road tubes [**8**]
- Piezoelectric sensors [**9**]
- Inductive loop [**10**]

On the other hand, Remote observations are the foundation of non-intrusive approaches. Even while manual counting is the most often used approach, new technologies that appear to be highly promising have lately emerged:

- Manual counts
- Passive and active infra-red
- Passive magnetic
- Microwave radar
- Ultrasonic and passive acoustic
- Video image detection

Profile surveys are used by experts to assist evaluate the intensity of traffic on a route and the direction of traffic flow. These surveys are frequently set up at junctions to monitor one or more lanes of traffic going in different directions for the most effective data collecting 6.

- **Profile Surveys**: A profiling survey is a type of manual counting in which the data is collected by a human. Most surveyors will design a paper form (in some cases substituded by mobile apps) to assist differentiate between different sorts of vehicle classifications, the direction or lane in which the vehicle is driving, and other pre-selected time intervals.
- **Directional Surveys**: consist of several human surveyors covering various directions along a single length of road or crossroads counting pre-specified criteria

1.3 IoT sensors, big data and data analysis development

The *Internet of Things*, also known as IoT, is a system that consists of interconnected computer devices, mechanical and digital equipment, items, or people that are given the capacity to communicate data across a network with almost no involvement from humans [**11**]. This implies the installation of various types of sensors on a variety of objects that, thanks to the concept of data analysis, are involved in various life scenarios, such as those involving individuals and businesses: household appliances that make up the so-called Smart Home; manufacturing machines in Industry 4.0; vehicles and smart roads, all of which are integrated to form Smart Cities. The steady increase in the installation of *Internet of Things (IoT)* objects leads to the consequent generation of *Big Data*. *Big Data* is a collection of data that is so large, varied, and rapidly transmitted that it is extremely challenging, if not nearly impossible, to analyze using conventional techniques. It has been said that given the right paradigms, it is feasible to filter, arrange, analyze, and get value from them. Therefore, Internet of Things and Big Data are deeply intertwined in this setting.

The dissertation does a wonderful job of addressing the issues raised by the present case study. After a considerable amount of data has been gathered from a large number of cars, it is sent to a data storage facility over the internet. From there, it is eventually analyzed by means of data analysis tools.

1.4 Thesis organization

The thesis is structured as follows:

Backgrounds and definitions of topics used in this thesis are presented in Chapter 2, which includes mathematical background, relevant topological explanations for

the thesis purpose, and technological background to understand how the thesis was constructed.

The implementation of the thesis is discussed in Chapter 3, beginning with pre-processing on initial input data to extract required information for the following step, segment extraction. The projection of a grid of cells on the map is then explained, followed by the methodology for assigning each segment to a grid cell. Furthermore, how to use the cells to calculate and display traffic congestion on a map, and eventually extract the pattern of the congestion, so that it can be used for future transportation development projects.

Chapter 2

Background and concept definition

2.1 The Floating Car Data (FCD)

Floating car data, also known as FCD, is a term used to describe a technology that gathers information on the status of the traffic from a series of individual vehicles that float in the traffic. Each vehicle, which can be thought of as a moving sensor that operates in a distributed network, is outfitted with positioning (GPS) and communication (GSM, GPRS, UMTS, etc.) systems. These systems allow each vehicle to send information about its location, speed, and direction to a centralized control unit, which then uses this information to make decisions [**12**] or simply just stores in a databases.

Since FCD systems transcend the limits of fixed traffic monitoring methods (Poor flexibility, high installation and maintenance costs, etc.), they are being employed in a number of crucial application, therefore, FCD is a replacement or supplement source of high-quality data to existing technologies. They will contribute to the transportation system's safety, efficiency, and dependability. They are becoming increasingly important in the creation of future *Intelligent Transportation Systems* (ITS). In this Thesis, a dataset of collected FCD is being exploit as the initial entry data.

In its most basic forms, there are essentially two primary categories of FCDs, namely GPS and cellular-based systems [**7**]:

2.1.1 GPS-based FCD

Even though the Global Positioning System (GPS) is becoming more popular and more inexpensive, only a small percentage of vehicles, generally those used for fleet management services, are now equipped with this system (e.g. taxi drivers). Due to the usage of dual frequency receivers that make use of several constellations in order to determine position, the level of accuracy achieved is quite high and stable [**13**] and is often less than 30 meters.

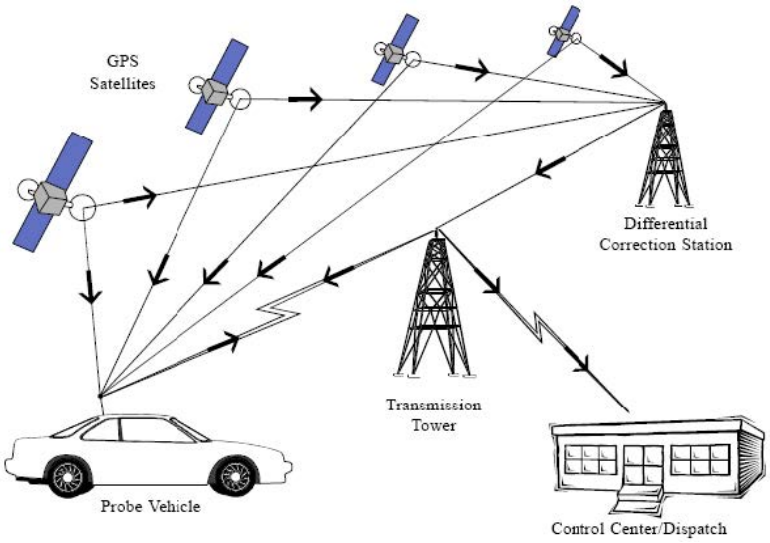

Figure 2.1: Communication from GPS, source [**14**]

In most cases, traffic statistics collected from individual automobiles or trucks are preferable for use in highways and rural areas. In the event of urban traffic, taxi fleets are especially valuable owing to the large number of vehicles in the fleet as well as the on-board communication technologies that are already in place. In the present day, GPS probe data are widely used as a source of real-time information by a variety of service providers. Despite this widespread use, there are only a limited number of vehicles equipped with the necessary equipment, and the cost of such equipment is relatively high in comparison to those associated with floating cellular data.

2.1.2 Cellular network based FCD

Given that the vast majority of cars on the road nowadays are equipped with at least one or more mobile phones, it may be desirable to use mobile phones to

carryout anonymous traffic surveys. The network receives periodic transmissions of the mobile phone's location, which are often accomplished by triangulation but may also be accomplished through other methods (e.g. handover). Mobile phones are required to be powered on, although this does not always imply that they are being used. Due to the shorter distance that separates each antenna in this system, it is especially well suited to the delivery of information that is generally precise in metropolitan areas, which is where traffic statistics are required the most.

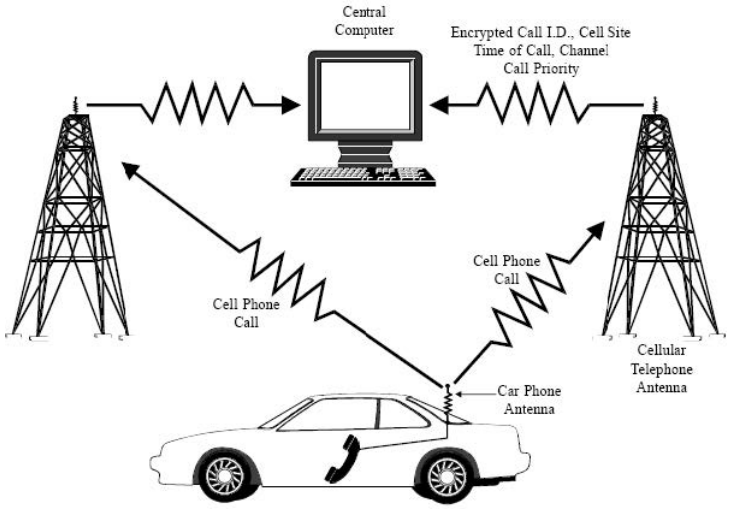

Figure 2.2: Communication from cellular phones, source [**14**]

There is no need for any specialized hardware or software to be installed in moving vehicles, in contrast to fixed traffic detectors and systems that are based on GPS. Additionally, there is no requirement for any unique infrastructure to be erected along the route. As a result, it is more cost-effective than traditional detectors while also providing more capabilities in terms of coverage. Instead of obtaining data at discrete points, continuous data collection is done on the traffic. It takes far less time to set up, it is less difficult to install, and it significantly requires less maintenance. In spite of the fact that the location accuracy is normally rather poor (about 300 meters), this deficiency is largely compensated for by the huge number of devices. It is important to take into account that the UMTS technology should be used in order to collect more precise data. At the moment, FCD is participating in many applications all over the globe that deal with the management and information on real-time traffic.

2.2 Topographical Concepts

2.2.1 Geographical coordinate system and WGS84

The geographic coordinate system, also known as the GCS, is a spherical or ellipsoidal coordinate system that is used to measure and communicate locations directly on the surface of the Earth as latitude and longitude [**15**]. It is the simplest, the oldest, and the one that is utilized the most commonly out of all of the many spatial reference systems that are now in use, and it serves as the basis for the majority of the others [**16**]. The World Geodetic System, also abbreviated as WGS, is a standard that is used in the fields of cartography, geodesy, and satellite navigation, including GPS. The National Geospatial-Intelligence Agency of the United States has been responsible for the establishment and maintenance of WGS since 1984. The most recent modification, which took place in January 2021, is WGS 84, which is also known as the WGS 1984 ensemble [**17**] [**18**]. Decimal degrees, often known as DD, are a method of expressing geographic coordinates such as latitude and longitude as decimal fractions of a degree. GPS devices and various geographic information systems (GIS), online mapping programs like OpenStreetMap, and other mapping applications typically employ DD.

2.2.2 Haversine Distance

The Haversine formula is used to calculate the distance in terms of great circles that separates two locations on a sphere, given the longitudes and latitudes of those places. It is a specific example of a more general formula in spherical trigonometry called the rule of Haversines, which relates the sides and angles of spherical triangles. This law is important in navigation, as it relates the sides and angles of spherical triangles.

Haversine function $hav(\theta)$ is:

$$hav(\theta) = sin^2\left(\frac{\theta}{2}\right) = \frac{1 - cos(\theta)}{2} \tag{2.1}$$

where:
$hav(\theta)$ is the central angle between any two points on a sphere.

By performing *arcsine* on 2.1 the equation could be solved to obtain distance d:

$$d = 2{\cdot}r{\cdot}\arcsin\sqrt{hav\left(\varphi_2 - \varphi_1\right) + \left(1 - hav\left(\varphi_1 - \varphi_2\right) - hav\left(\varphi_1 + \varphi_2\right)\right) \cdot hav\left(\lambda_2 - \lambda_1\right)} \tag{2.2}$$

where:
λ_1, λ_2 are the longitude of point 1 and longitude of point 2
φ_1, φ_2 are the latitude of point 1 and latitude of point 2.

2.2.3 Geometric Dilution Of Precision

The term "dilution of precision," also known as "geometric dilution of precision" (GDOP), is used in satellite navigation and geomatics engineering to describe error propagation as a mathematical effect of navigation satellite geometry on positional measurement precision. The purpose of the GDOP is to specify how inaccuracies in the measurement will have an impact on the estimate of the final state. One possible explanation for this is:

$$GDOP = \frac{\Delta(OutputLocation)}{\Delta(MeasuredData)} \tag{2.3}$$

It is claimed that the geometry is weak and the DOP value is high when visible navigation satellites are close together in the sky; on the other hand, it is said that the geometry is strong and the DOP value is low when the satellites are far away. Imagine two circles, or annuli, that overlap one another and have different centers. When they overlap each other at right angles, the maximum extent of the overlap is much less than when they overlap each other in a nearly parallel fashion. Because of the greater angular separation between the satellites that are used to determine a unit's location, a lower DOP value indicates a higher level of positional accuracy than a higher number. Other factors, like nearby obstacles like mountains or buildings, may also cause the effective DOP to go up [**35**].

DOP can be expressed as a number of separate measurements:

- HDOP – Horizontal Dilution Of Precision
- VDOP – Vertical Dilution Of Precision
- PDOP – Position (3D) Dilution Of Precision
- TDOP – Time Dilution Of Precision
- GDOP – Geometric Dilution Of Precision

DOP calculation for satellite i, consider:

$$R_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} \tag{2.4}$$

where:
x, y, z remark about the location of the receiver, and $x_i, y_i, z_i(\theta)$ represent the location of satellite i. Matrix A is formulated as follow:

$$A = \begin{bmatrix} \frac{(x_1-x)}{R_1} & \frac{(y_1-y)}{R_1} & \frac{(z_1-z)}{R_1} & -1 \\ \frac{(x_2-x)}{R_2} & \frac{(y_2-y)}{R_2} & \frac{(z_2-z)}{R_2} & -1 \\ \frac{(x_3-x)}{R_3} & \frac{(y_3-y)}{R_3} & \frac{(z_3-z)}{R_3} & -1 \\ \frac{(x_4-x)}{R_4} & \frac{(y_4-y)}{R_4} & \frac{(z_4-z)}{R_4} & -1 \end{bmatrix} \tag{2.5}$$

The first three elements in each row of A are the components of a unit vector from the receiver to the specified satellite. The last element of each row is the partial derivative of pseudorange with respect to the clock bias of the receiver. Formulate Q as the covariance matrix with the fewest squares that comes from the normal matrix:

The elements of Q are designated as:

$$Q = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{xz} & \sigma_{xt} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{yz} & \sigma_{yt} \\ \sigma_{xz} & \sigma_{yz} & \sigma_z^2 & \sigma_{zt} \\ \sigma_{xt} & \sigma_{yt} & \sigma_{zt} & \sigma_t^2 \end{bmatrix} \tag{2.6}$$

Eventually, GDOP and HDOP are formulated as [**36**]:

$$GDOP = \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2 + \sigma_t^2} \tag{2.7}$$

$$HDOP = \sqrt{\sigma_n^2 + \sigma_e^2} \tag{2.8}$$

depends on the coordinate system used To match the local horizon plane and vertical in a north, east, or up coordinate system.

2.2.4 DOP value classification

[**37**] interprets the obtained DOP value and classifies it as follows:

Table 2.1: DOP value classification

DOP value	Rating	Description
<1	Ideal	Finest confidence level to be utilized for applications that need the highest accuracy at all times.
1-2	Excellent	At this confidence level location measurements are deemed precise enough for all applications save the most sensitive ones.
2-5	Good	Represents a minimal acceptable threshold for making correct selections. Positional measurements might be utilized to provide the user with credible in-route navigation recommendations.
5-10	Moderate	Although positional measurements might be utilized for computations, the quality of the fix could still be improved. A more open view of the sky is advised.
10-20	Fair	Represents a low degree of confidence. Positional measurements should be ignored or only utilized to provide an approximate estimation of the present position.
>20	Poor	At this level, readings using a 6-meter accurate equipment are erroneous by up to 300 meters and should be ignored.

2.3 ICT background

2.3.1 Python

Python is a general-purpose programming language, which means that it may be used to develop a wide range of different applications and is not specifically tailored to solve any particular issues. Because of its flexibility and the fact that it is easy to learn, it has quickly become one of the most widely used programming languages today. Developed by a group of programmers led by British scientist, Guido van Rossum, Python is actually an interpreted language. The design concept behind it places an emphasis on the readability of the code by making extensive use of indentation [**19**].

The language has applications in a variety of domains, including Web Development, Data Science and Machine Learning, and Simulation; all of these domains, as well as others, are being investigated as part of the same research effort that this thesis is a part of. As a result of this, Python has been preferred over other

languages like R. There are several libraries that can be imported, which is another selling feature for the product. In direct relation to this body of work, the most significant ones that have been applied are as follows:

- **Pandas**:A software library designed for data manipulation and analysis. In particular, it provides data structures and functions for the manipulation of numerical tables and time series [**20**] [**21**].

- **Geooandas**: a free and open-source tool that simplifies the process of dealing with geographical data in Python. The datatypes that pandas uses are extended by GeoPandas so that spatial operations may be performed on geometric types. Shapely is responsible for carrying out geometric operations [**22**] [**23**].

- **Numpy**: Adding support for big, multi-dimensional arrays and matrices, as well as a vast set of high-level mathematical functions to work on these arrays. The "ndarray" data structure, which stands for "n-dimensional array," lies at the heart of NumPy's fundamental capabilities. These arrays have a consistent data type across the board [**24**].

- **Matplotlib**: a complete library for the creation of interactive, animated, and static visualizations. [**25**] [**26**]

- **Shapely**:Python package with the Shapely name that allows for the manipulation and analysis of planar geometric objects under the BSD license. Although Shapely is not concerned with specific data formats or coordinate systems, it can be easily integrated with other packages that are [**27**].

- **Folium**: Folium is a robust Python module that provides assistance in the development of a variety of different Leaflet maps. By default, Folium builds a map in a separate HTML file. Because Folium's findings may be interacted with, this library is an excellent resource for the construction of dashboards. In Folium, you also have the ability to construct inline Jupyter maps.
  Folium takes advantage of the strengths of the Python environment for working with data and the strengths of the Leaflet.js library for making maps. You can alter your data with Python using Folium, and then you can view the results using a Leaflet map [**28**] [**29**].

- **Pyspark**: PySpark is a Python-based interface for the Apache Spark data processing framework. It not only lets you build Spark applications with Python APIs, but it also gives you access to the PySpark shell, which lets you analyze your data in a distributed setting in an interactive way. In other words, Apache Spark is an analytical processing engine designed for use in

applications that need strong distributed data processing on a large scale. Machine learning applications also make use of Apache Spark [**30**] [**31**].

2.4 Related works

Studying the patterns of traffic and the conditions of traffic on roads and streets has become an extremely important part of the administration and planning of urban areas due to the ever-increasing amount of automobile traffic in urban areas. This is of utmost significance in large cities, which often have frustratingly slow moving traffic. Magnetic loops, video observation, laser and infrared vehicle detection are some of the more classic methods that have been employed in recent decades to gather information about traffic. However, in more recent times, a new set of other methods and data sources have also been utilized. In recent years, another data source utilized to investigate traffic situations is FCD. In contrast to traditional traffic data, which is typically collected at a fixed location by a stationary device or observer, Floating Car Data in traffic engineering and management is typically timestamped geo-localization and speed data directly collected by moving vehicles that are equipped with GNSS receivers using their on board unit in real-time.

Numerous publications have been published that investigate the efficiency of FCD in a variety of contexts, including but not limited to: costs, penetration rate into automobiles, dependability, and comparison of the acquired findings to more conventional methodologies. As was explained in [**42**], some traffic analysis might be performed using FCD, which would provide findings that are comparable to those of traffic sensors but would incur a far lower cost. In this study, the data that the FCD gathered over the course of 17 continuous hours and included 375,000 records that belonged to 10,000 cars were mapped to the nearest street using Open Transport Map (OTM) so that analysis could be carried out. [**43**] also described an experience of producing a comprehensive traffic analysis by utilizing coherent numbers of FCD and highlighting how this solution provides higher coverage than traffic sensors while also having lower costs. Using FCD and Remote Traffic Microwave Sensor Data (RTMS) data, [**47**] conducted an investigation of the peculiarities of traffic flow on Beijing's ring road expressways. While the average speeds were calculated using both sets of data, the volume calculations were carried out using RTMS data only. The flow-speed relationship in the basic figure was derived by combining the data that were previously collected. After doing a regression analysis on the average speeds obtained from RTMS and FCD, it was discovered that the values obtained from RTMS were, on average, 6% greater than those obtained from FCD. Another comparison was made by [**48**] between the average speeds of road segments acquired from FCD data and Bluetooth data. A statistical analysis was carried out for each

of the four speed categories, and it was discovered that the average Bluetooth speeds were not substantially different from the FCD speeds for any of the speed categories.

One of the disadvantages of using FCD data is the low penetration rate of on board units into cars, which is mentioned in [**42**] which could lead to the results having less reliability than they otherwise would due to the fact that the data were collected from a small percentage of cars. In spite of the fact that [**44**], on the basis of their simulation, suggests that the 1.5% penetration rate of vehicles with on board units is required to have an accurate estimation of travel time, [**45**] suggests that the penetration rate between 3%-5% in order to achieve the confidence level of over 90% would be sufficient. in addition, [**46**] came to the conclusion that the appropriate penetration rate should be anywhere from 5 to 10%, depending on the current traffic condition. Nonetheless, the scope of the investigation for [**46**] is limited to just one highway in Singapore. [**46**] also investigates the variations in the intervals of FCD collection, which revealed that the best results would acquire if the collection interval is between 15 and 20 seconds, despite the fact that this demands a great deal more processing power. Basically, extending the interval for every 15 seconds results in a considerable reduction in the needed amount of processing power.

[**49**] by studying just one parameter, which is average travel speed from FCD, the researchers came to the conclusion that it is still feasible to uncover key patterns along urban roadways, provided that the data is continuous and broad. This research, analyzes the FCD data obtained from automobiles in Ankara over a period of two months with an interval of one minute. The data was taken from the vehicles every minute. In this investigation, the average speed of the road segments is converted to a Level of Service (LOS), and the status of the traffic is determined by a comparison to the speed at which free-flow may occur on the same road segment. Moreover, by comparing the traffic states of two, three, and four successive road segments, one may determine where the bottlenecks are in the road system.

Alongside the implementation of FCD in [**50**] and [**51**], the idea of Big Data is also included. [**50**] blends big data with small data in the creation of elements of transport system models. This is done with the intention of increasing the ability of transport analysts and planners to assess, predict, and prepare for mobility phenomena. [**50**] concludes that the conventional techniques for constructing transport system models are improved with the use of big data for mobility. Similarly,[**51**] presents an original demand estimate framework that is comprised of two steps and argues that the use of Big Data and FCD results in the limitation of not having a strong initial demand matrix being overcome.

2.5 Dataset

The exploited dataset in the thesis is collected from an insurance company. It is a very large dataset, containing more than 700 million rows, that was collected from January of 2019 to January of 2020. The collection was made before the crisis of the COVID-19 pandemic, therefore the dataset represents the behavior of the vehicles in an ordinary time period.

The records inside the dataset are spread over an area, as vast as Piedmont province. However, since the goal of the thesis is to retrieve congestion pattern in a metropolitan area, thus only the records belonging to the city of Turin and some smaller nearby towns are conserved, and the other records are eliminated before proceeding to further steps.
The collected FCD dataset contains following columns and features:

Table 2.2: FCD dataset description

Attribute	Type	Description
deviceId	Text	the unique ID associated to each device 1 to 20 characters
dateTime	DateTime String	Time and date of submitted record format: YYYY-MMDD HH24:MI:SS
latitude	Number (9,6)	expressed in degrees from -90 to 90
longitude	Number (9,6)	expressed in degrees from -90 to 90
speedKmh	Integer	Instantaneous speed in Kilometer per Hour ranging from 0 to 250
heading	Integer	The direction that the vehicle is headed 0 to 360 in 4 degree increments
accuracyDop 2.8	Integer	GPS signal quality expressed in tenths of HDOP 2.1 between 0 (excellent) and 150 (very bad)
EngineStatus	Integer	1 engine on 0 engine off
Type	Integer	1 car 2 fleet

Chapter 3

Implementation and Results

3.1 Input FCD analysis

3.1.1 Data cleaning

The process of detecting and correcting (or removing) corrupt or inaccurate records from a record set, table, or database is known as data cleansing or data cleaning. It also means finding parts of the data that are missing, wrong, inaccurate, or not important, and then replacing, changing, or getting rid of the dirty or coarse data [**32**]. When numerous data sources are combined, there is a greater likelihood that some of the data may be duplicated or incorrectly categorized. Even though the results and methods seem to be right on the surface, their reliability is compromised if the underlying data is inaccurate. Because the procedures differ from dataset to dataset, there is no one method to prescribe the precise stages in the process of data cleaning. This is because there is no one definitive technique to do so. However, it is essential to develop a pattern for each data cleaning procedure in order to ensure that you are always doing the task in the appropriate manner [**33**].

As the main data cleaning steps taken on the input FCD data, below criterias, have been considered:

- **Duplication removal**: During the process of collecting the data, there will most likely be instances of duplicate observations. There is a possibility of creating duplicate data whenever data sets are joined from different locations, whenever data is scraped, whenever data is received from customers or numerous departments, and when scraping data. In this procedure, de-duplication is one of the most important aspects that needs to be addressed.
- **Structural errors and missing data removal**: When doing measurements or uploading data, it is possible to uncover irregular naming conventions,

typos, missing values, or incorrect capitalization. These inconsistencies may result in incorrect categorization of groups or classes.

The work flow of the thesis calls for a couple of data cleaning processes to be conducted at various phases of the development of the project. Each of these steps has its own unique set of requirements that must be met. The first stage in cleaning the data is based on the criteria that were specified before. Nevertheless, this would not be the only component of the thesis that involves cleaning up the data, and the other one is discussed in 3.2.
The original FCD input data spans 12 months of 2019 and consists of 738,969,717 records, of which 5.1% are duplicated rows; following the duplication elimination process, there are a total of 701,536,205 records. In addition, after the 'Not a Number' (NaN) values have been removed, the output comprises 655,798,896 entries. These steps are the very first initial steps of data cleaning.

3.1.2 Area of interest

Choosing the scope of the problem that will be investigated is one of the most important decisions to be made at the first phase of the process. Since the purpose of the thesis is to describe the traffic congestion pattern in an urban area, and the large FCD dataset contains records that are as large as the Piedmont province in Italy, and since the goal of the thesis is to describe the traffic congestion pattern, only the metropolitan city of this region and some small but nearby cities are investigated, and other zones are excluded from the investigation. Figure 3.1.3 shows the interest area for further analysis with the following coordinates that start from minimum latitude and longitude (44.96282106687191,7.502048016422193) to the maximum latitude and longitude (45.19265016665321,7.791812422724604). Eventually, the number of records that are inside this area is: 375,647,074.

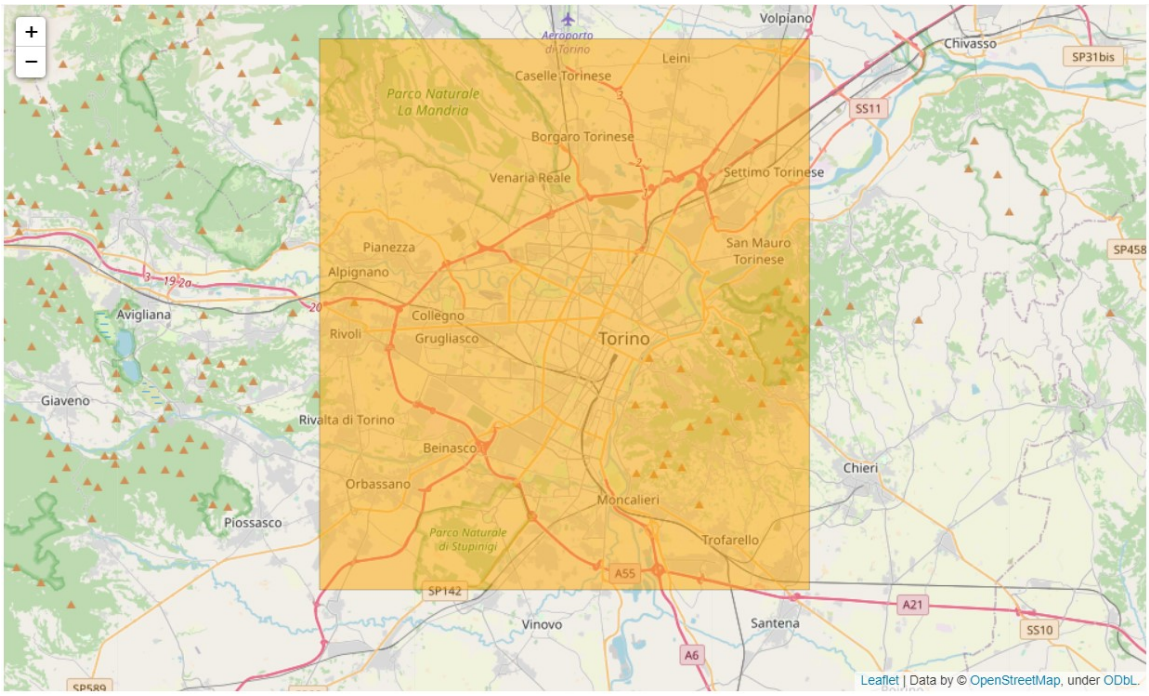

Figure 3.1: The area of interest, including Turin and nearby cities

3.1.3 Data behaviour

Vehicle types

From the total number of 375,647,074 records in the urban area, 310,264,034 records, meaning 92.99% of total, are belong to 'Personal' vehicle type and the rest with 23,367,845 records consist the 'Fleet' vehicle type. Since the 'Personal' type is dominant in the dataset, so the behavior of the vehicles are considered as a whole and not specifically by each type of vehicle.

GPS accuracy DOP

As mentioned in 2.2.3, the accuracy of a GPS signal is affected by an error propagation factor, which is known as dilution of precision. Thus, the quantity of DOP can specify the signal quality of GPS. In figures 3.2 and 3.3 ,both cumulative and non cumulative distribution of DOP values of FCD dataset are demonstrated. according to [**37**], since a DOP value less or equal than 2 is considered as an "excellent" signal in terms of precision, further, based on the behavior on the cumulative distribution function; the value of $HDOP \leq 2$ which contains 94.09%

of the whole dataset, is considered as a threshold to filter the records, and the records with a HDOP greater than 2 are dropped. This is also a part of the data cleaning part.

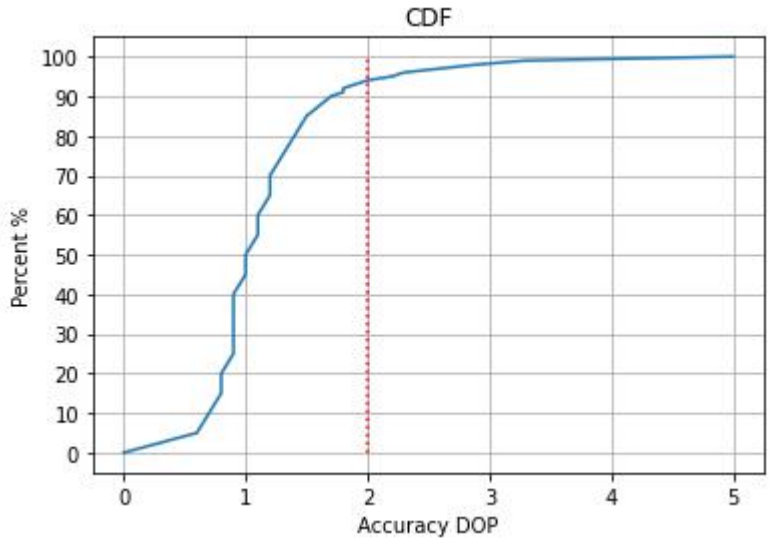

Figure 3.2: Cumulative Distribution Function of GPS accuracy DOP

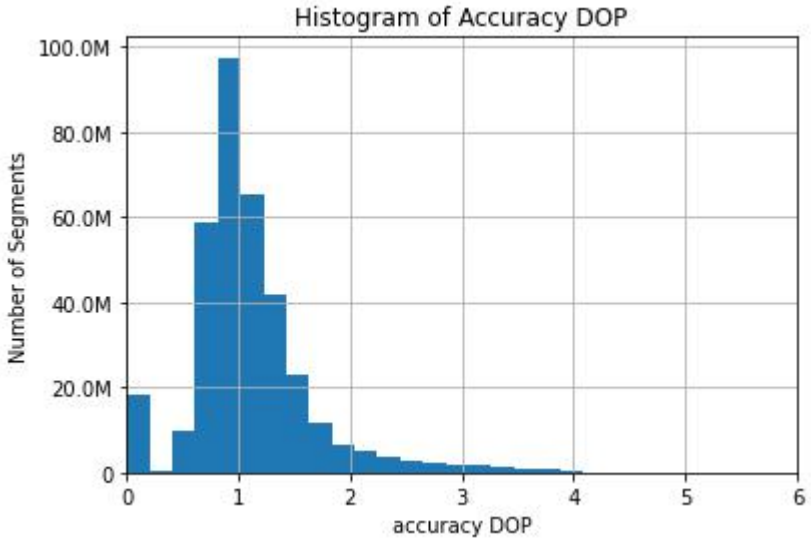

Figure 3.3: Distribution of GPS accuracy DOP

Records daily distribution

Figure 3.4 shows how the FCD records are spread over the year 2019 in a daily time frame. However, for the sake of more readability, the plot is labeled in monthly intervals. As it can be seen, there is a repeating pattern in the behavior of the records. Basically, the pattern shows the weekly behavior. There is a peak on Fridays of each week, and a dip is found on Sundays, which are the weekend. However, in summer, specifically in the month of August, due to the summer holidays, a dramatic decrease in terms of submitted records is happening.

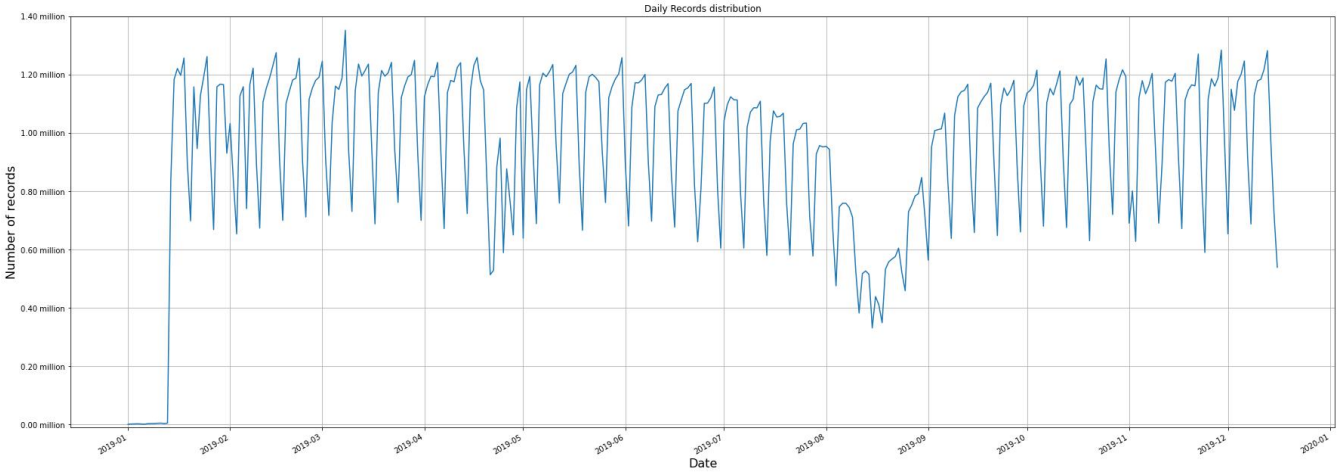

Figure 3.4: Daily Records Distribution

Records hourly distribution

Beside the daily and weekly time frames, it is also interesting to investigate the hourly distribution of the records as well. It could give information about peak hours that are most probably the time intervals in which the most congestion occurs. Figure 3.5 illustrates the diffusion of the FCD records in the hourly time frame. As it can be clearly seen, there are two peaks that exist. One in the morning and the other one, which has a higher peak, in the evening. Furthermore, there has been a drop in the early morning hours, when most people are sleeping and outside activity is at its lowest. Such behavior is quite reasonable. In the morning, when people usually go to school or work, a peak happens, and the other one, when people are returning to their houses after having finished the work activity.

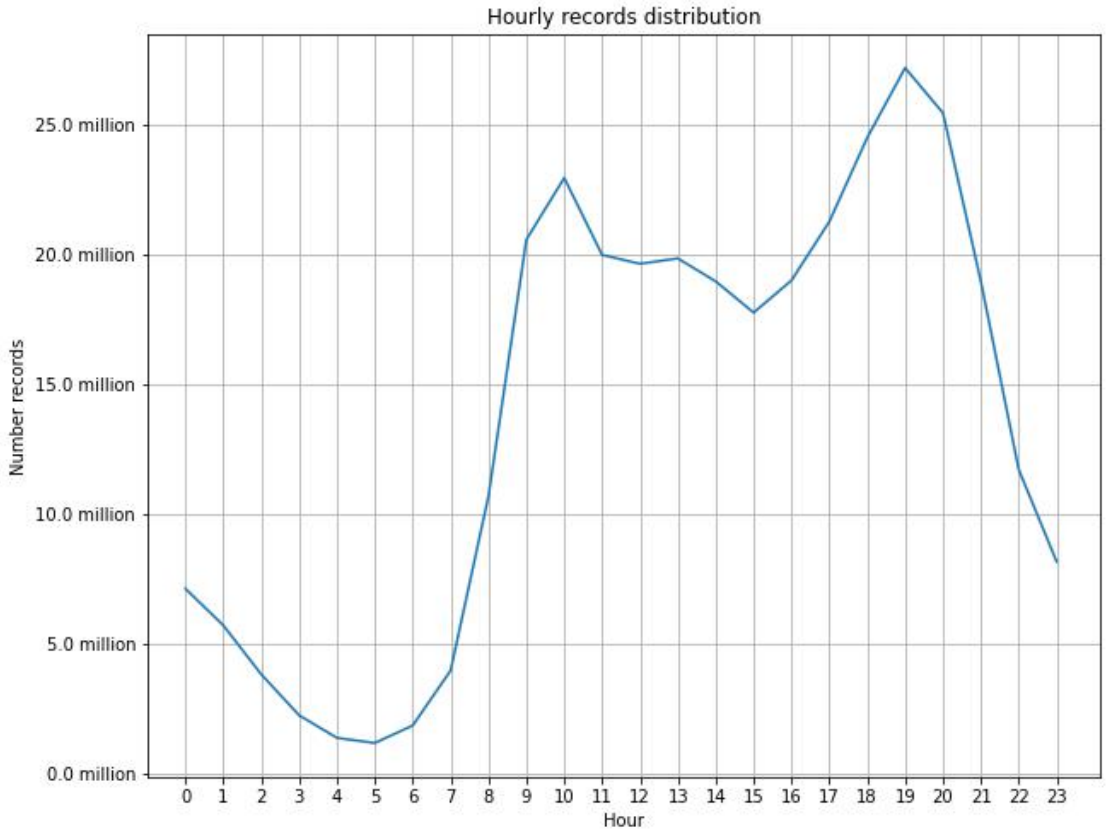

Figure 3.5: Hourly Records Distribution

3.2 Segment Extraction

After conducting basic data cleaning and experimentation on the input FCD data, the first significant step is to extract the path segments of the taken route for each individual vehicle. This is done after examining the FCD data. This is the fundamental prerequisite for the subsequent procedures that will be undertaken to perform analysis.

3.2.1 Segment definition

The movement of an item may be segmented down into trajectories that span the distance between any two significant points. This division differs depending on the application. The movement of a vehicle used by a delivery service, for instance, may be segmented down not just into its daily travels, but also into its moves between individual clients [**38**].

In this thesis, the considered segmentation is defined as the smallest portion of the overall route that is traveled by each individual vehicle. These portions are identified as the traveled path between the time intervals during which the data has been collected.

3.2.2 Segment extraction methodology

The first thing that has to be done in order to extract the path segment is to sort the records that belong to each unique vehicle (which has its own ID) according to the time stamp. Next, the path that was taken between every consecutive pair of recordings is what is known as the path segment for each and every unique vehicle. Then for each path segment, the following parameters are calculated:

Segment distance

Geopy is an extremely useful Python module that is used to determine the distance between segments. This package takes as its input the coordinates of two points, each of which has a latitude and a longitude, and its output is the Haversine distance 2.2.2, which it calculates using meters as the unit of measurement. Combining the segment distance with the calculated segment duration, enables the calculation of the segment speed at a later stage, which is one of the most important characteristics that must be determined.

Segment duration

Is the amount of time that elapses between the beginning of one segment and the beginning of the next. The calculation of the segment length provides valuable information on the time intervals at which each FCD report is sent.

As Figure 3.6 shows the cumulative distribution of duration of the segments, and also Figure 3.7 confirms as well, a majority of the path segments which means 56.93% of the whole, have been recorded with the intervals of 60 seconds. Considering a tolerance of 10 seconds, as a part of data cleaning, the segments with duration more than 70 seconds which form 8.5% of the dataset are identified as outlier and are dropped.

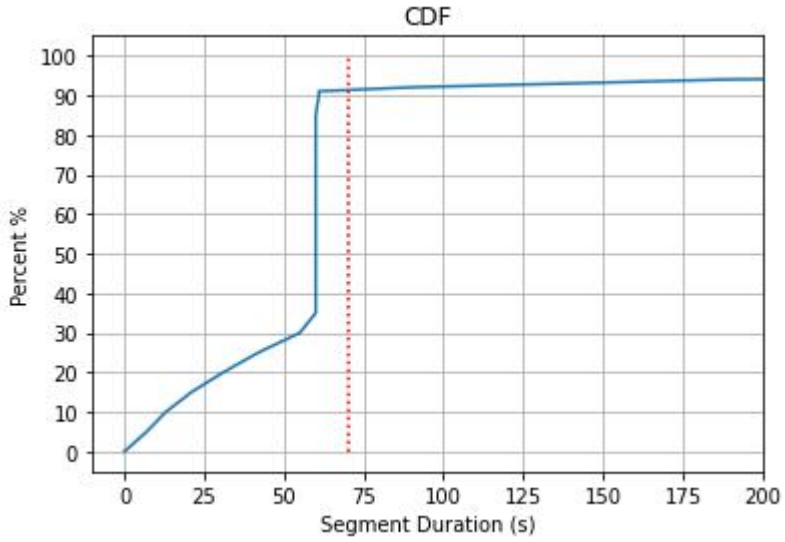

Figure 3.6: Cumulative distribution of Segment duration's

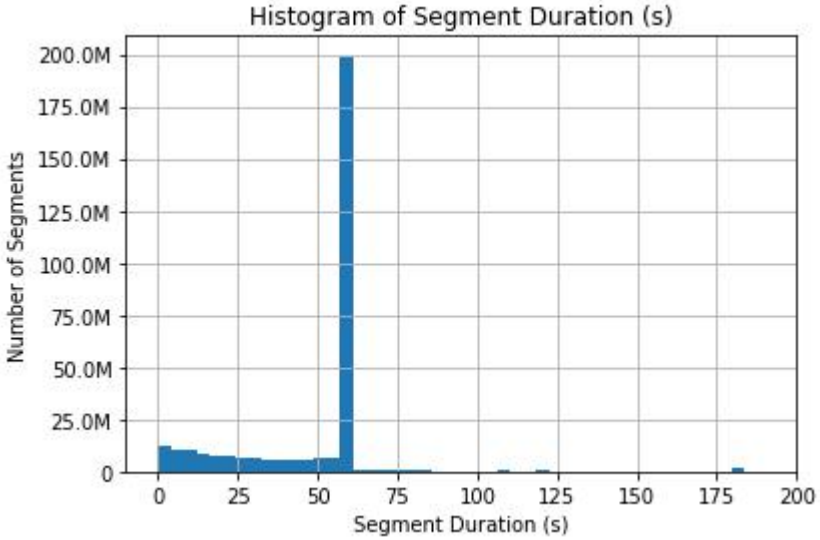

Figure 3.7: Distribution of Segment duration's

Segment speed

It is quite easy to calculate the segment's average speed; all that is required is a simple division of the segment's distance, which is measured in meters, by the segment's duration, which is measured in seconds. After that, a factor of 3.6 is applied to the segment average speed that was determined before in meters per second in order to convert it to the desired unit of kilometers per hour.

To investigate the behavior of the path segments better and following by the applied filter in 3.6, the segment duration of records are grouped based on the different intervals as following:

- Segments with duration between 0-9 seconds
- Segments with duration between 10-19 seconds
- Segments with duration between 20-29 seconds
- Segments with duration between 30-39 seconds
- Segments with duration between 40-49 seconds
- Segments with duration between 50-59 seconds

Following this, the segment speed is computed for each of these groups in order to investigate the behavior of segment speed in each interval. The findings of this investigation are shown in the figure 3.8. It is clear that despite the fact that there are certain insignificant differences that are reasonable, no group is responsible for any kind of bias in the overall behavior, and groups could be considered similar to one another. To determine whether or not more data cleaning is required, this phase is carried out. However, because of the similarities, no special data cleaning action was performed at this phase. Instead, for the next steps, the segment duration of all of the records will be merged into a single group.

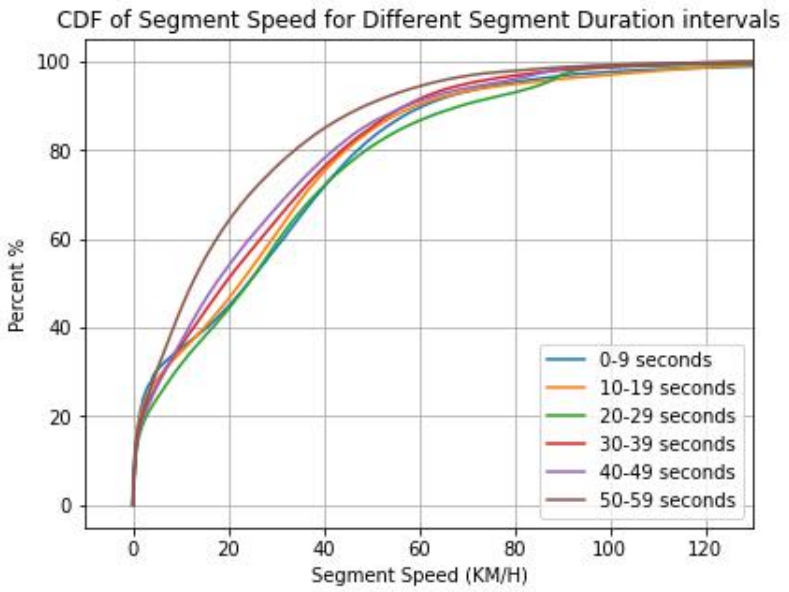

Figure 3.8: Cumulative distribution of Segment Speed (KM/H) of different groups

The cumulative distribution of the average speed of the segments is shown in figure 3.9, which is then followed by the distribution plot in figure 3.10. Because the highest speed that is permitted by law in Italy is 130 kilometers per hour, this limit has been selected as the upper bond in order to retain only the segments that have a value that is lower than or equal to this value. Because of this, 0.52% of the segments are discarded at this step of the data cleansing process. In addition, 2.72% of the segments have been found to have values that are equivalent to 0 on the absolute scale. Since a value of absolute zero indicates that there has been no movement at all for those particular segments, it is also necessary to eliminate them as part of the process of cleaning the data.
In a nutshell, there were two stages of data cleaning that took place in relation to the average speed of the segments: first, the segments whose average speed was equal to or less than 130 kilometers per hour were maintained as the upper bond; second, the segments whose average speed was greater than absolute zero were maintained as the lower bond.

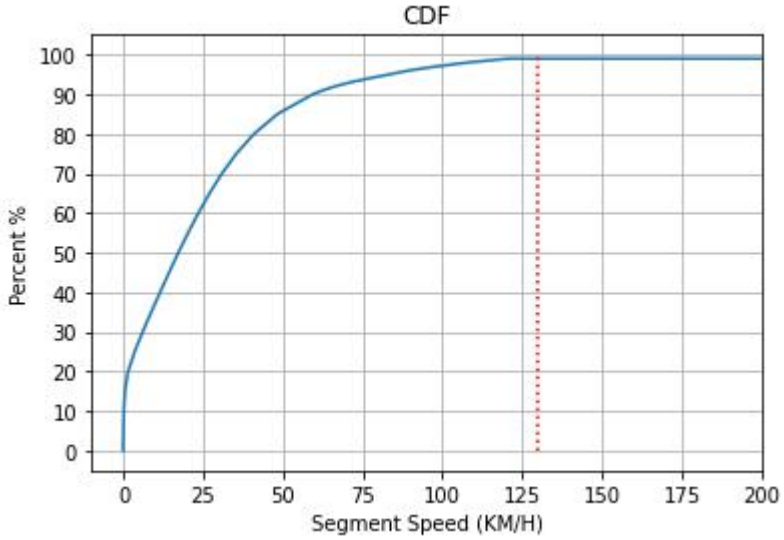

Figure 3.9: Cumulative distribution of Segment Speed (KM/H)

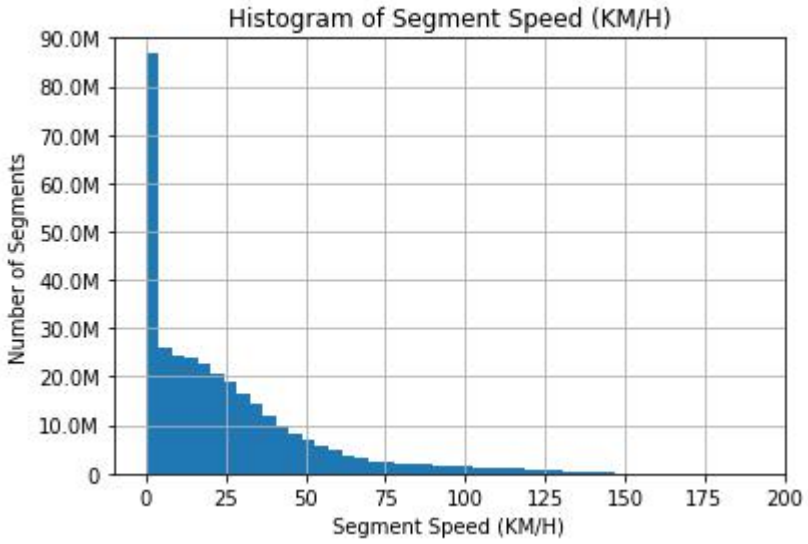

Figure 3.10: Distribution of Segment Speed (KM/H)

3.2.3 Segment table

For the sake of ease and consistency throughout the work, the segment table is produced using the original FCD data as the input. This is based on the segment definition that has been supplied, as well as the technique that describes how the segments are extracted in 3.2.1 and 3.2.2. Due to the fact that it provides the computed average speed as well as the starting and ending location of each segment, the segment table is an extremely important component in the analysis of the thesis. This is because it leads the way for the discovery of traffic congestion. The most important columns of the segment table are shown in table 3.1.

Attribute	Type	Description
deviceId	Text e.g: 2507297	the unique ID associated to each device 1 to 20 characters
Type	Integer	Vehicle type, 1: car , 2: fleet
dateTime	DateTime String e.g: 2019-04-02 10:15:28	Time and date of segment starting point format: YYYY-MMDD HH24:MI:SS
startLatitude	Number (9,6) e.g: 45.08672	latitude of segment starting point expressed in degrees from -90 to 90
startLongitude	Number (9,6) e.g: 7.60882	longitude of segment starting point expressed in degrees from -90 to 90
startEngineStatus	Integer	Engine status at segment starting point 1: engine on , 0: engine off
startAccuracyDop	Integer e.g: 10	GPS signal quality expressed in tenths of HDOP between 0 and 20
endAccuracyDop	Integer e.g: 10	GPS signal quality expressed in tenths of HDOP between 0 and 20
endEngineStatus	Integer	Engine status at segment ending point 1: engine on , 0: engine off
startLatitude	Number (9,6) e.g: 45.08683	latitude of segment ending point expressed in degrees from -90 to 90
startLongitude	Number (9,6) e.g: 7.60893	longitude of segment ending point expressed in degrees from -90 to 90
segmentDistance	Float, e.g: 139.25	travelled distance in the segment in unit of meters
segmentDuration	Float, e.g: 60.0	duration of the the segment in unit of seconds
segmentSpeedKmH	Float, e.g: 26.38	average speed of the segment in unit kilometers per hour

Table 3.1: Segment table description, Other properties derived directly from the latitude and longitude coordinates have been omitted for the purpose of brevity.

3.2.4 Segment distribution

Another aspect to investigate regarding the behavior of the segments, is to examine the distribution of the segments over different time spans, for instance, monthly, weekly, daily, and hourly.

Segments monthly, weekly and daily distribution

Figure 3.11 illustrates how the segments are distributed during the course of 2019 on a daily basis. The figure, however, is labeled in monthly intervals for the convenience of reading. Observably, the behavior of the records follows a recurring pattern. The pattern essentially depicts the weekly activity. Each week has a weekly high on Friday and a weekly trough on Sunday, the weekend. However, throughout the summer, notably the month of August, owing to the summer vacations, the number of submitted records decreases dramatically.

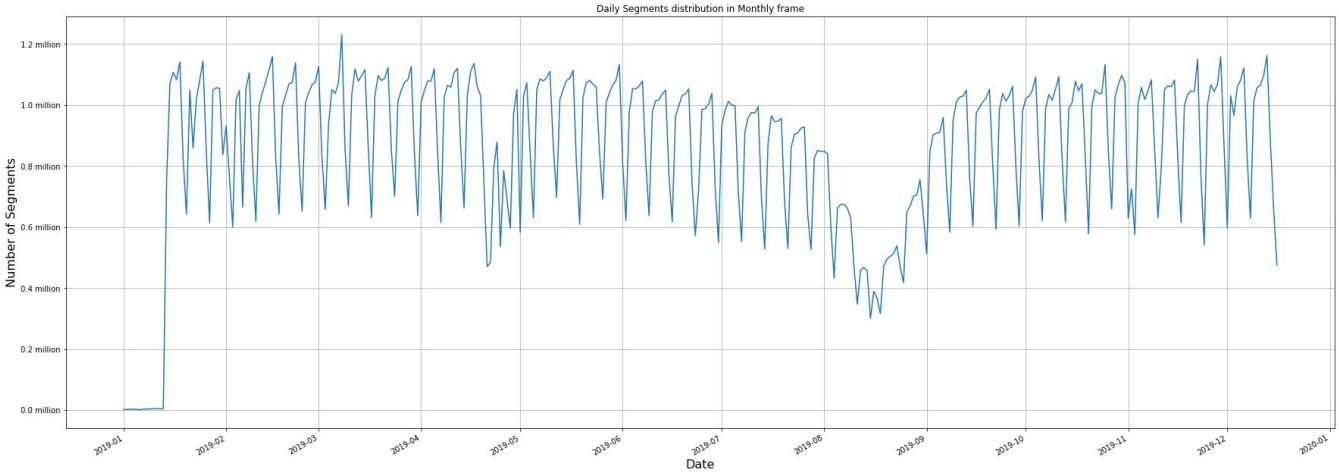

Figure 3.11: Daily Segments Distribution

In addition to the daily and weekly time periods, it is also worthwhile to analyze the hourly distribution of the segments. It provides information on peak hours, which are most likely the most congested periods of time. Figure 3.12 depicts the distribution of segments across an hourly time period. Clearly visible are the existence of two peaks. The one with the greater peak occurs in the evening, while the one with the lower peak occurs in the morning. In the early morning hours, when most people are asleep and outdoor activity is at its lowest, there has also been a decline. Such conduct is very fair. A peak occurs in the morning, when people typically go to school or work, and another occurs in the evening, when people are coming home from work.

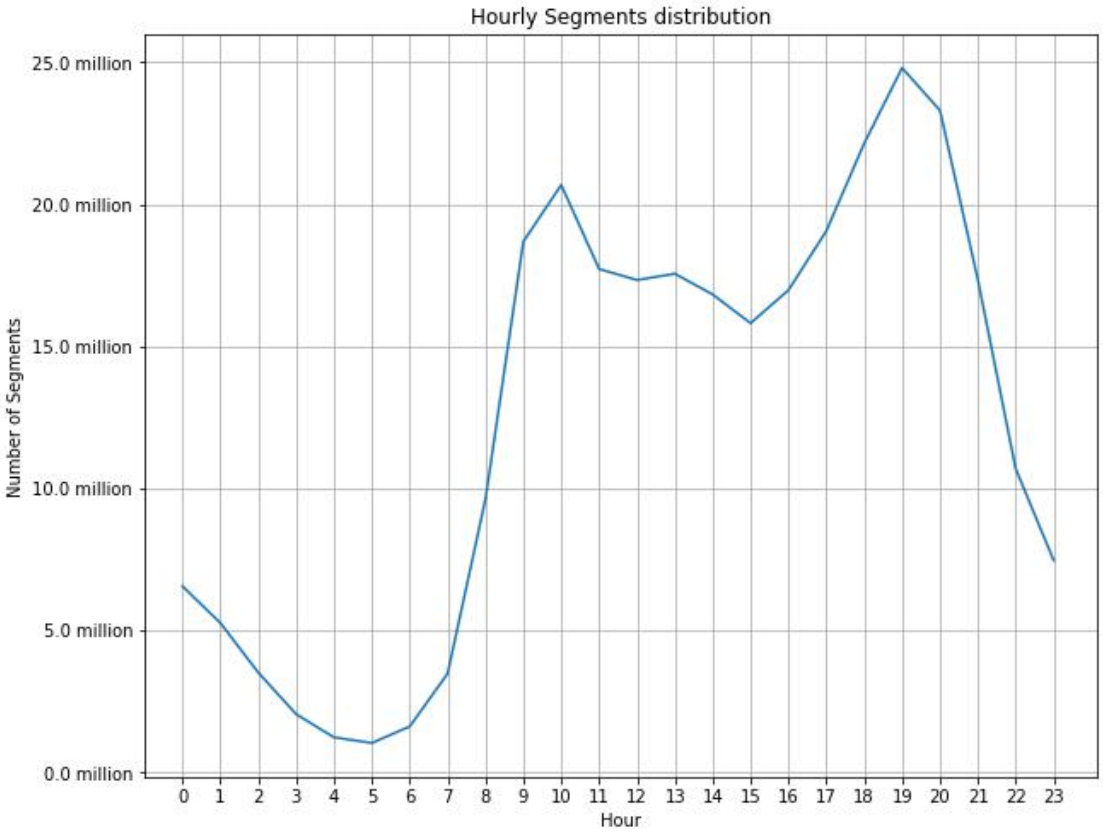

Figure 3.12: Hourly Segments Distribution

3.3 Grid map

Based on the goal of the thesis, the considered area 3.1 is broken down to a grid on the map. A network of horizontal and vertical lines that are equally spaced apart and used to designate places on a map is known as a grid. The row and column labels of a reference grid will often indicate locations that are mentioned in an index for the map [**39**].

3.3.1 Grid size selection

When determining the appropriate cell size inside the grid, some criteria that should be considered include the resolution, the amount of time needed for computation, and the number of segments contained within each cell. There needs to be some sort of compromise between the various aspects that have been brought up, and they are discussed one by one:

Number of segments per cell

The target cell should have a significant number of subcellular segments packed inside of it. Because the influence of each segment's inherent bias would be reduced as the density increased, the cell would become both more informative and more trustworthy as the density increased. Figure 3.13 shows the average number of segments per cell for different cell sizes

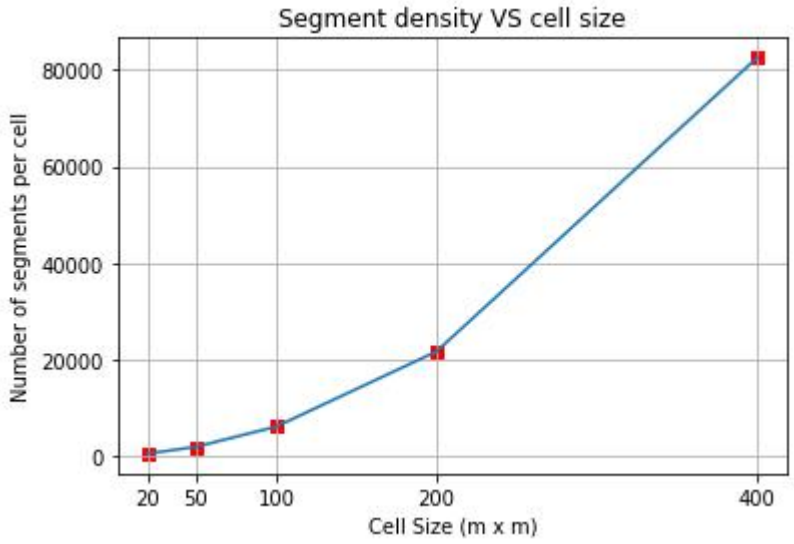

Figure 3.13: Segment density VS cell size

Computation time

To obtain a greater resolution, the size of each cell must be reduced, which results in an increase in the number of cells that must be processed. As a direct consequence of this increase, the amount of calculation time that is necessary also rises. However, in the case of this thesis, the procedure may be completed in a reasonably short period of time because to the extremely powerful computing resources that are given by SmartData@PoliTO [**40**]. This is true even in difficult scenarios that need a significant level of compute power. The amount of time needed for calculation is broken down below in figure 3.14 according to the various cell sizes.

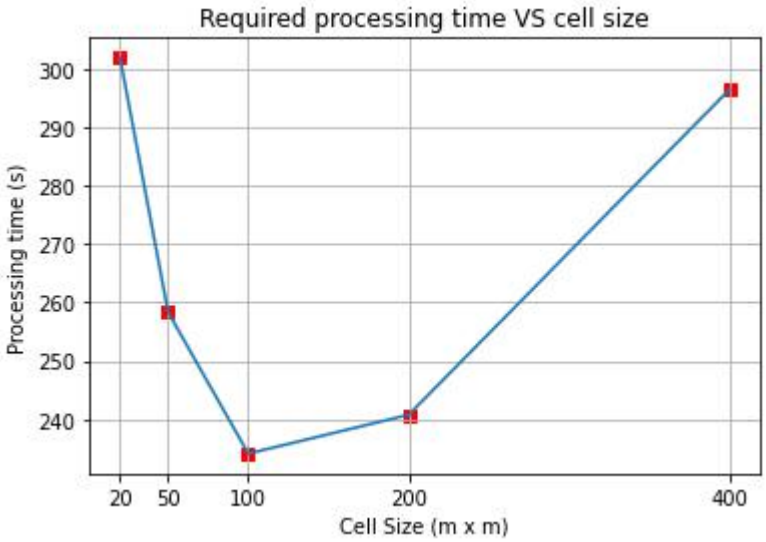

Figure 3.14: Required processing time VS cell size

Resolution

Resolution of cells is one of the most important considerations to make while selecting an appropriate cell size. If the resolution was higher, then there would be more specific information on every little piece of the area. On the other hand, there would be fewer segments for that area, which would result in the information being less accurate. Alternatively, if the resolution is low, the reliability will rise, but there will be less details as a consequence.
As a whole, and on the basis of the findings that were obtained, the size of the grid's individual cells has been decided to be 100 meters by 100 meters.

3.4 Grid table

Following the completion of two key phases, which are represented in 3.1 and 3.3, the segment table and the outcome of dividing the map into a grid of cells are combined into a single entity with the common key which is the identifier assigned to each cell. The consequence of this merge is having the possibility of visualizing on the map how the segments are distributed across the metropolitan area of Turin and some of the smaller cities that are located in the surrounding area in a variety of different aspects. These aspects for each cell of the grid including, the density of the segments, the speed of the segments, and the GPS accuracy DOP of the segments. For each component, one might just compute the minimum, maximum, and the mean; nevertheless, the end result of calculating the mean value is the

point of interest because it relates to the purpose of the thesis.

Attribute	Type	Description
polygon_id	String e.g: 20_63 means	20_63 means cell located in 20th row and 63th column on the grid
geometry	Polygon	The geometry of the corresponding cell
cell_id	String e.g: 20_63 means	20_63 means cell located in 20th row and 63th column on the grid
num_segments	Interger e.g: 1594	Indicates the density of the segments inside the cell
minSpeed	Float e.g: 1.54 (KM/H)	Minimum speed of the segments inside the cell
avgSpeed	Float e.g: 20.39 (KM/H)	Mean of speed of the segments inside the cell
avgSpeed	Float e.g: 120.14 (KM/H)	Maximum speed of the segments inside the cell
minAccuracyDop	Integer e.g: 1	Best GPS signal quality of the segments inside the cell expressed in tenths of HDOP
avgAccuracyDop	Float e.g: 9.32	Mean of GPS signal quality of the segments inside the cell expressed in tenths of HDOP
maxAccuracyDop	Integer e.g: 20	Worst GPS signal quality of the segments inside the cell expressed in tenths of HDOP

Table 3.2: Grid table. Obtained by outer join of 3.1 and 3.3

3.4.1 Grid plots

This current section has, up until this point, provided an explanation of the mechanism behind the extraction of grid tables. As a direct consequence of this, the plots of the grid map in three different aspects—the density of segments in each cell, the mean of the speeds of the segments in each cell, and the GPS accuracy DOP—are presented respectively in 3.15, 3.16, and 3.17.

Segment density per cell

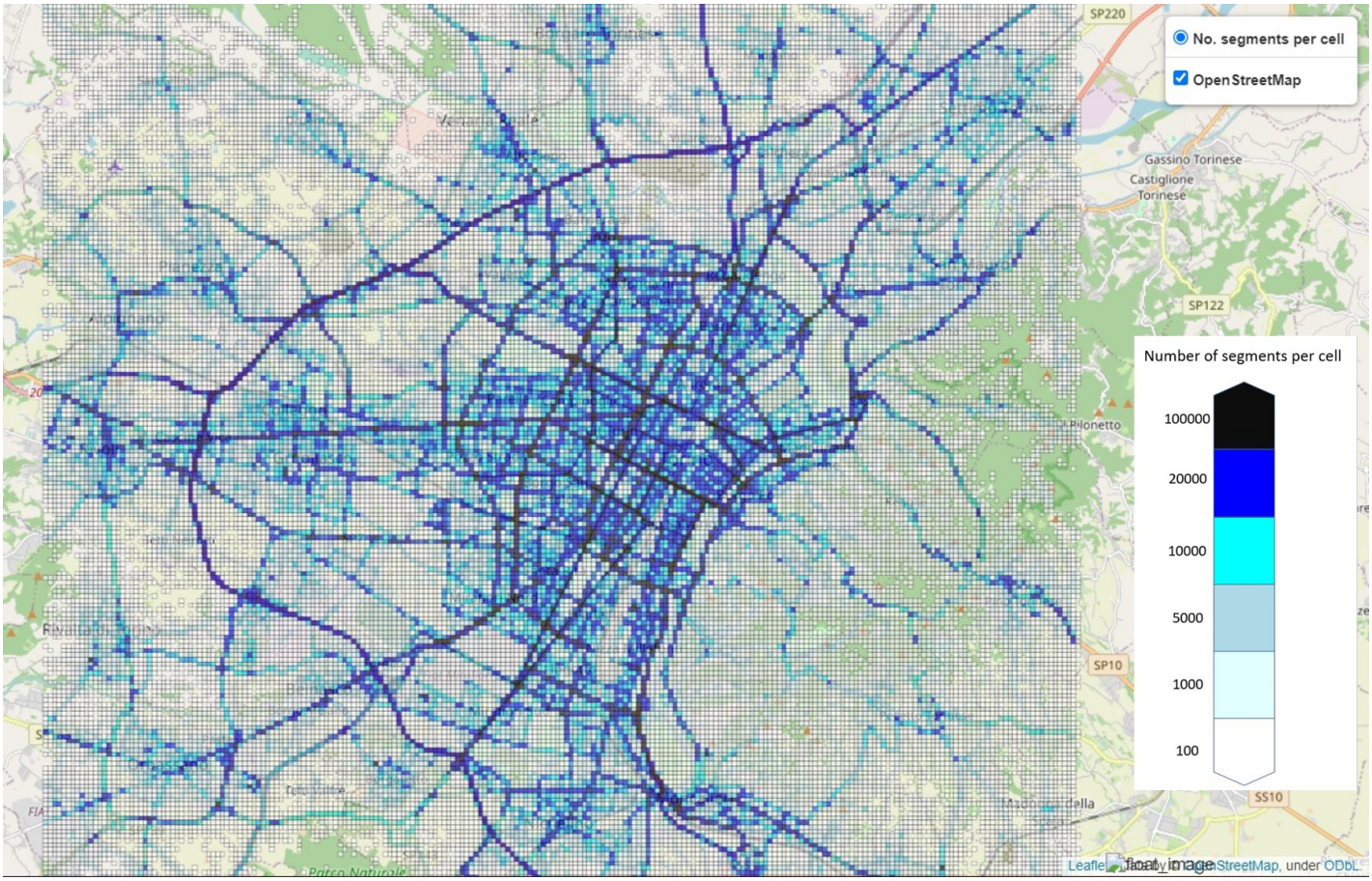

Figure 3.15: Number of segments per cell

It is clear by looking at 3.15 that the density of segments within is substantially higher along the main streets and highways than it is in any other section of the city. This demonstrates the predicted behavior of automobiles and demonstrates that main streets and highways are being utilized to a greater extent by automobiles.

Mean of the speeds of the segments per cell

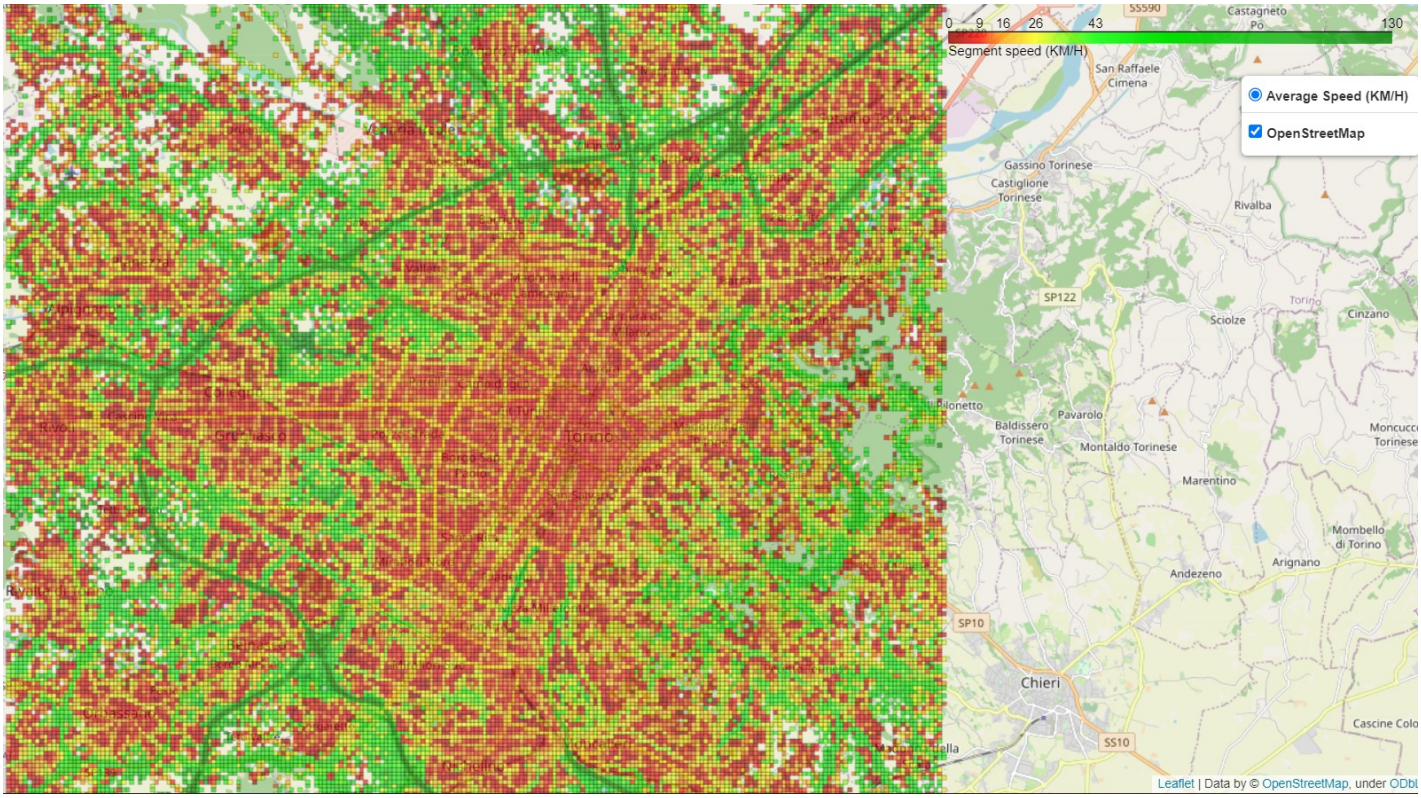

Figure 3.16: Mean of the speeds of the segments per cell

it can be seen in 3.16, highways contain the cells within which, the average speed of the segments is at the highest. In the urban area, along the main streets, the average speed is relatively higher, with respect to the downtown and side streets. As expected, the calculated average speed of the segments is quite low in downtown, where it is expected to be the most crowded area.

Mean of GPS accuracy DOP per cell

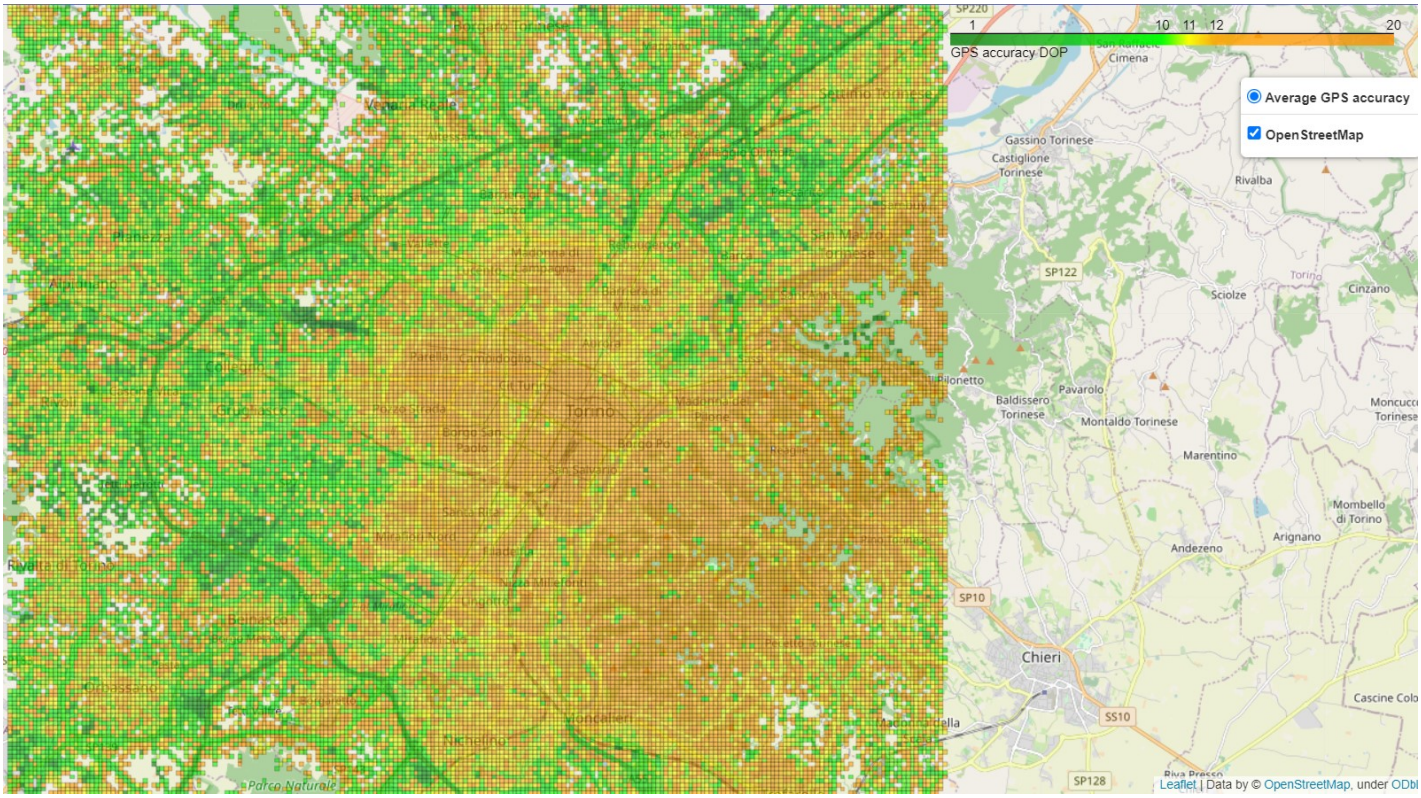

Figure 3.17: Mean of GPS accuracy DOP per cell

As Figure 3.17 demonstrates, the accuracy of received GPS signal in urban areas and also in the hills located on west side of Turin, are relatively lower than the other pats. This could be due to the existence of the buildings which cause multipath errors. The receipt of signals that have come not only straight from satellites but also reflected or diffracted from the objects in the immediate vicinity is what causes GPS multipath that eventually resulting a shift in the location that was estimated [**41**].

3.5 Traffic congestion index

In this part of the article, the methodology behind calculating the traffic congestion index is broken down and explained. This is made possible as a direct result of the creation of the grid table 3.4, which makes this particular option available. Two independent measures of traffic congestion are used in the calculation of the traffic congestion index: absolute traffic congestion and relative traffic congestion. In spite

of this, the focus will mostly be on the latter since it is the objective of the thesis. The reason for this is because the region being researched has several various kinds of roadways. As an example, on a roadway that has an average speed of 100 KM/H during off-peak hours and 80 KM/H during peak hours, the difference of 20 KM/H is not considered to be severe congestion. On the other hand, the difference of 20 KM/H is significant in a neighborhood in the downtown area where the average speed is 40 KM/H.

One key step to be taken before the traffic congestion index calculation, is to identify the peak and off-peak of hours over a complete day cycle of 24 hours. Based on the hourly distribution of the segments 3.12, two peaks the chart are considered as peak hours, morning and evening peak. morning peak hours start from 9AM to 11AM, while the evening peak period is from 6PM to 8PM. The off-peak interval is chosen for the interval of 1AM to 6AM. The reason for choosing the off-peak period longer than the peak hours is due to the low number of records submitted during this period that could lead the analysis to be less reliable. The selected intervals are depicted in figure 3.18.

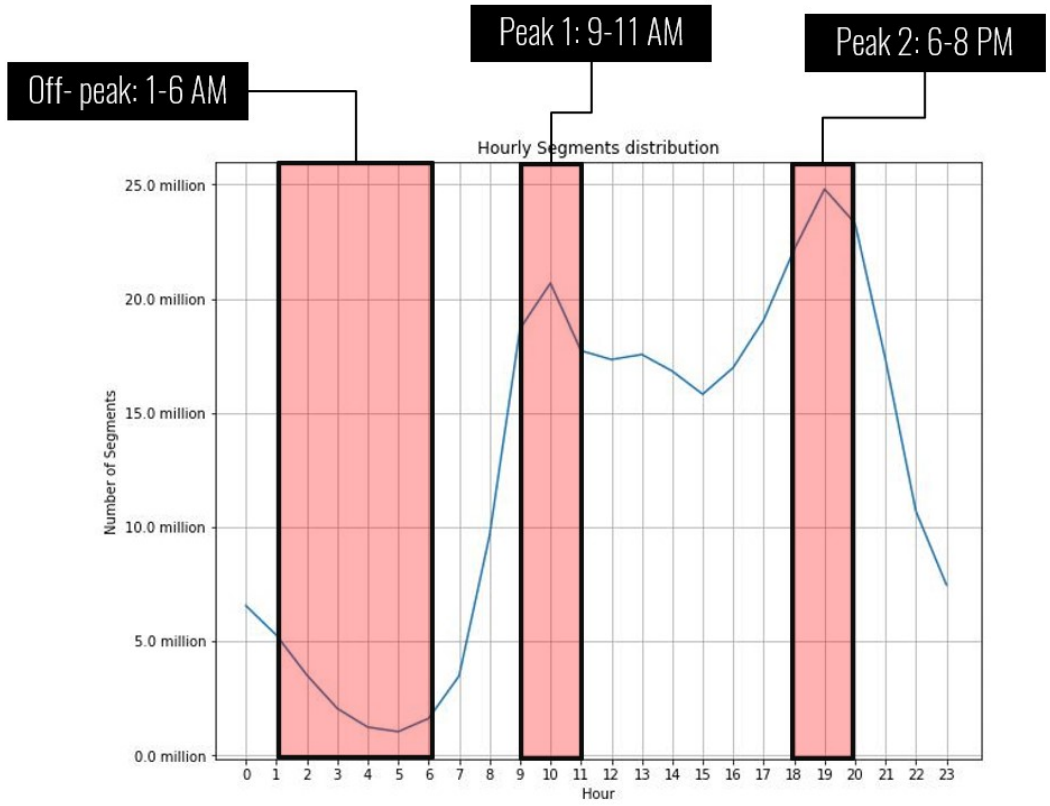

Figure 3.18: Peak and off-peak hours selection

Dropping uncommon cells

One of the first steps toward calculation and comparison of the traffic congestion is to keep only the cells that contain segments in both peaks and off-peak intervals. This is obvious, since in the case of a lack of data in either interval, it is not possible to make a comparison and calculate the congestion.

Traffic index calculation

In this thesis, two different ways of traffic congestion index calculations consist of absolute traffic congestion and relative traffic congestion are experimented that are respectively explained in 3.5.1 and 3.5.2, however, as mentioned earlier the focus is mainly on the relative congestion index

3.5.1 Absolute traffic congestion

Absolute traffic congestion index is simply calculated as the difference of the speed between peak and off-peak hours for the same cell in the grid:

$$V_{OP} - V_P \tag{3.1}$$

Where:
V_{OP}, is the Speed (KM/H) in the Off-peak period
V_P, is the Speed (KM/H) in the Peak period

Distribution

The cumulative distribution of cell-wise traffic congestion is shown in Figure 3.19, which compares the morning and evening peaks with the off-peak period individually. As can be seen, 34.74% of the cells in the morning peak and 29.78% of the cells in the evening peak are exhibiting negative congestion. This indicates that the calculated average speed of cells during the off-peak period is lower than the one during the peak period, which is behavior that is somewhat undesirable. It is to be anticipated that it will be more congested during the peak hours than during the off-peak hours in the regular scenario. In addition, the distribution of cell-wise absolute traffic congestion is displayed in Figure 3.20 below. It is abundantly evident that there is a behavior that is more favorable for the evening peak. The primary reason is because there are more data obtained for the evening peak in comparison to the data collected for the morning peak, which simply means that the results are more reliable. Therefore, the focus will be mostly the comparison between evening peak and the off-peak hours in the very early morning

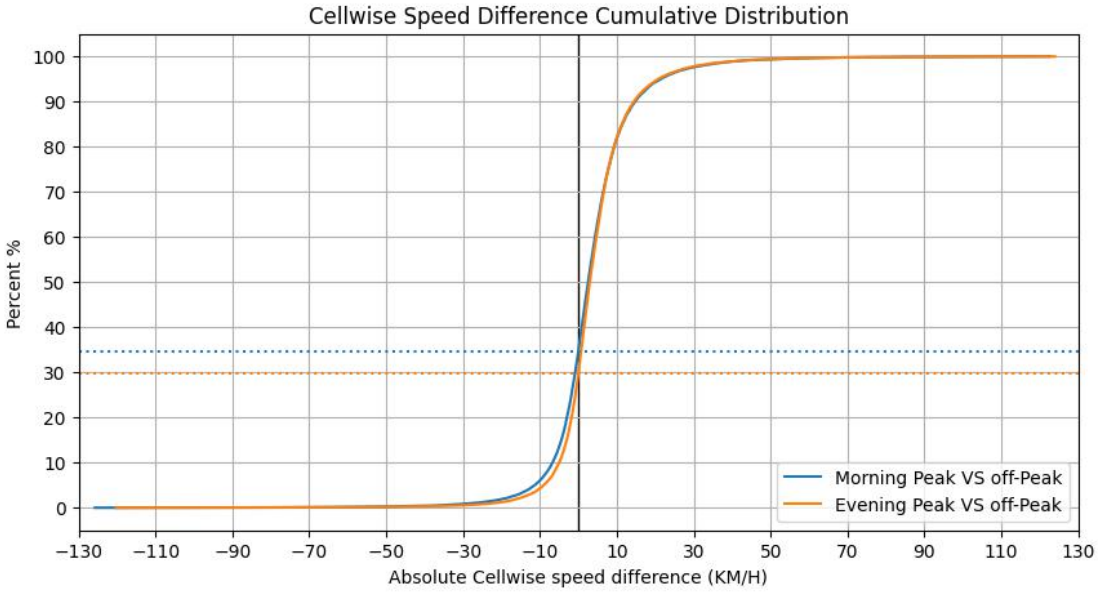

Figure 3.19: Cell-wise cumulative distribution of absolute traffic congestion

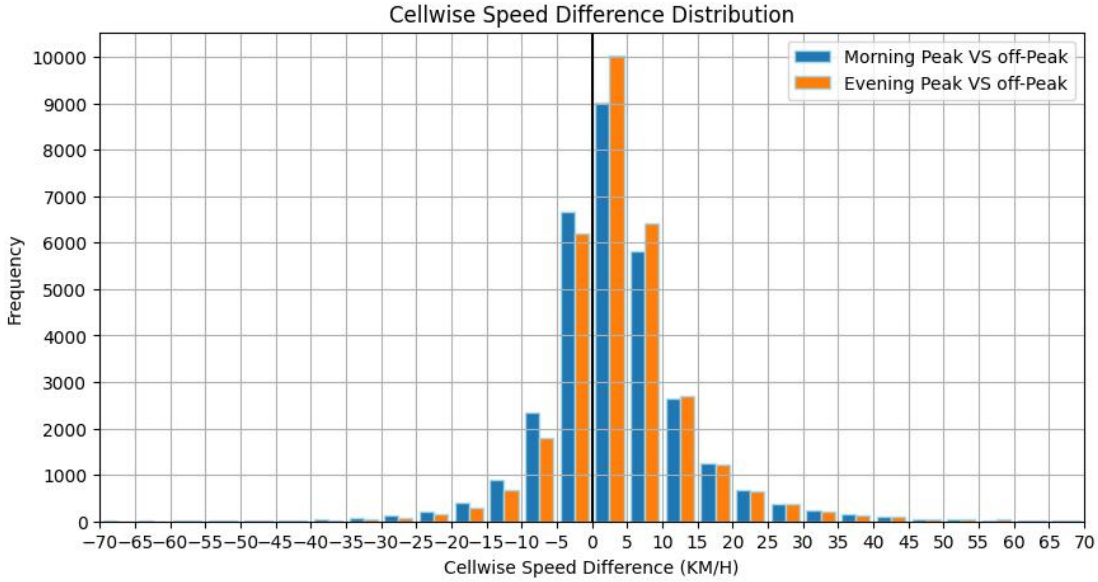

Figure 3.20: Cell-wise distribution of absolute traffic congestion

3.5.2 Relative traffic congestion

The relative congestion index uses a simple formula 3.3 to measure the relative variations in speed during peak hours against off-peak hours:

$$\left(\frac{V_{OP} - V_P}{V_{OP}}\right) * 100 \tag{3.2}$$

Where:
V_{OP}, is the Speed (KM/H) in the Off-peak period
V_P, is the Speed (KM/H) in the Peak period

To standardize the scale and keep it between -100% and 100%, the greater velocity (either in the peak or off-peak interval) is always used as the base to compute the relative changes in speed. As a result, in circumstances when negative congestion is observed, the formula will be slightly different:

$$-\left(\frac{V_P - V_{OP}}{V_P}\right) * 100 \tag{3.3}$$

Distribution

Similarly to 3.5.1, the distribution of the traffic congestion index indicates similar behavior. Negative congestion is present in 34.74% of the cells in the morning peak and 29.78% percent of the cells in the evening peak. The more reliable behavior of the evening peak interval is obvious in the relative traffic congestion index as well. Figures 3.21 and 3.22 show the distribution of relative traffic congestion in cumulative and normal ways respectively.

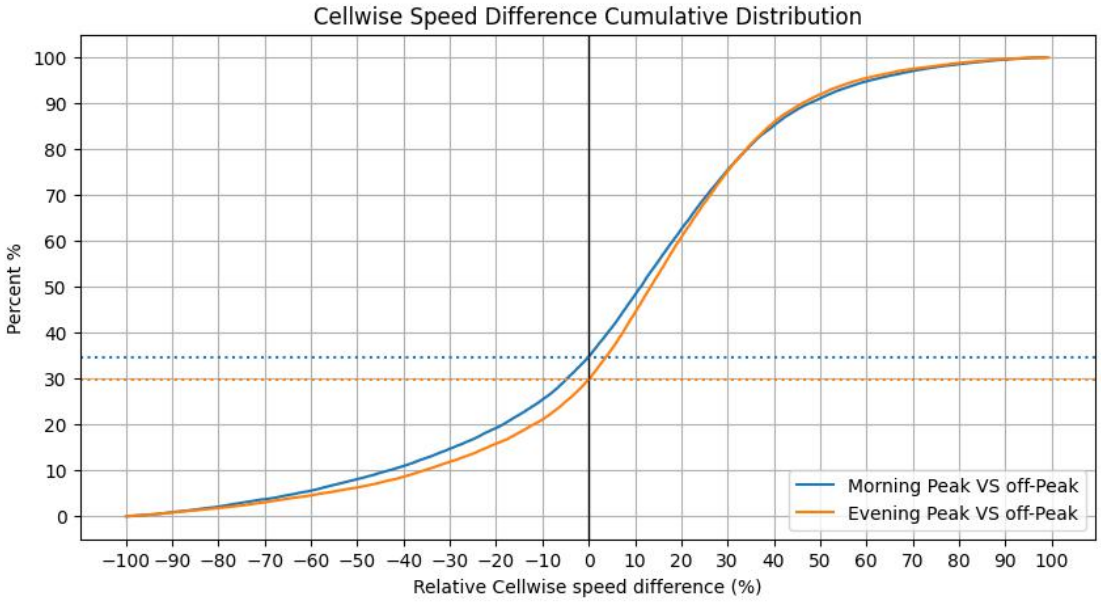

Figure 3.21: Cell-wise cumulative distribution of relative traffic congestion

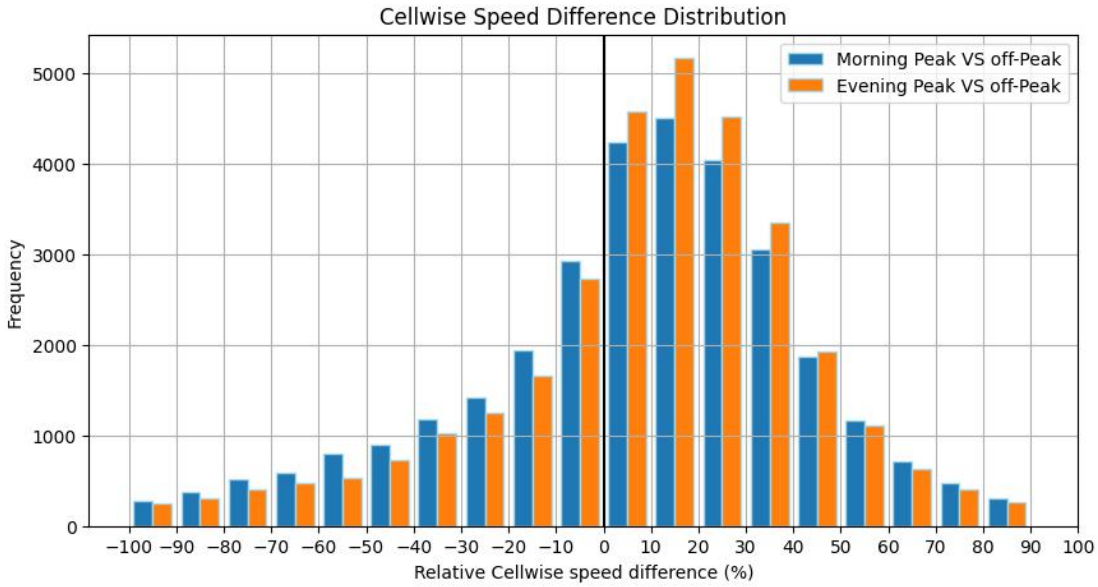

Figure 3.22: Cell-wise distribution of relative traffic congestion

3.5.3 Cells with negative congestion

As indicated in 3.5.1 and 3.5.2, there exist some cells in the grid that demonstrate negative congestion, implying that the average speed in the off-peak interval is

lower than in the peak hours. To understand in which cells this scenario is more likely to happen, the behavior of the number of segments per cell is investigated as the main feature of each cell based on the purpose of the thesis. As shown in 3.23, cells with fewer segments inside are much more likely to exhibit negatively congested behavior. This is mainly due to the unreliability of the results with a low amount of input. In addition, as it can be seen also in figure 3.24 the distribution of the negatively congested cells, tend to fade out (green bar) as the number of the segments per cell increase and the blue and orange bars will be aligned eventually. Therefore, it would be reasonable to filter out the cells with fewer segments inside for further steps in the analysis.

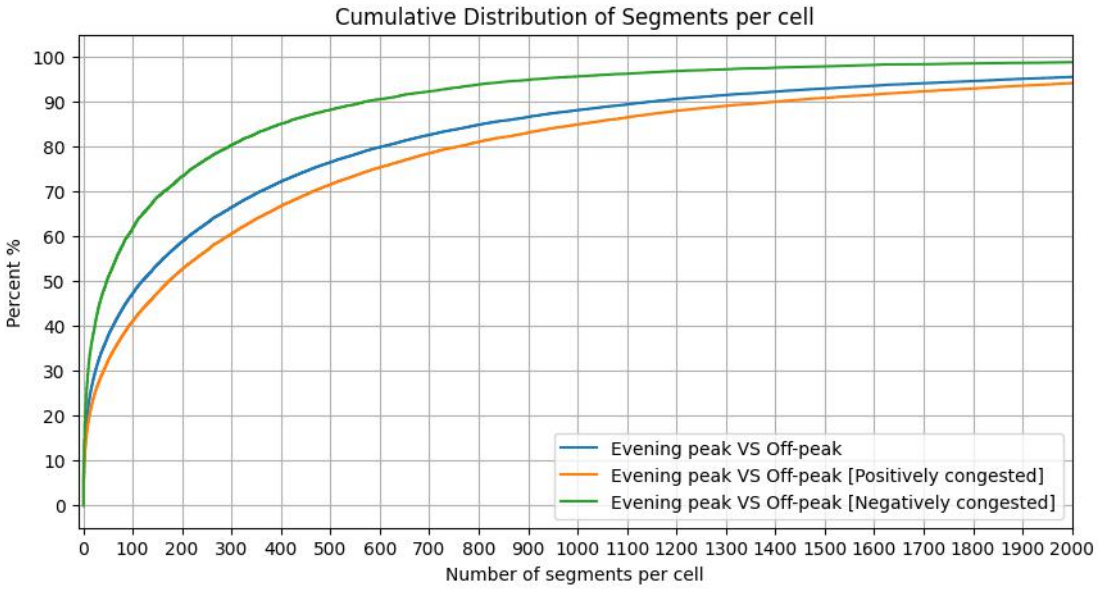

Figure 3.23: Cumulative distribution of cell-wise traffic congestion

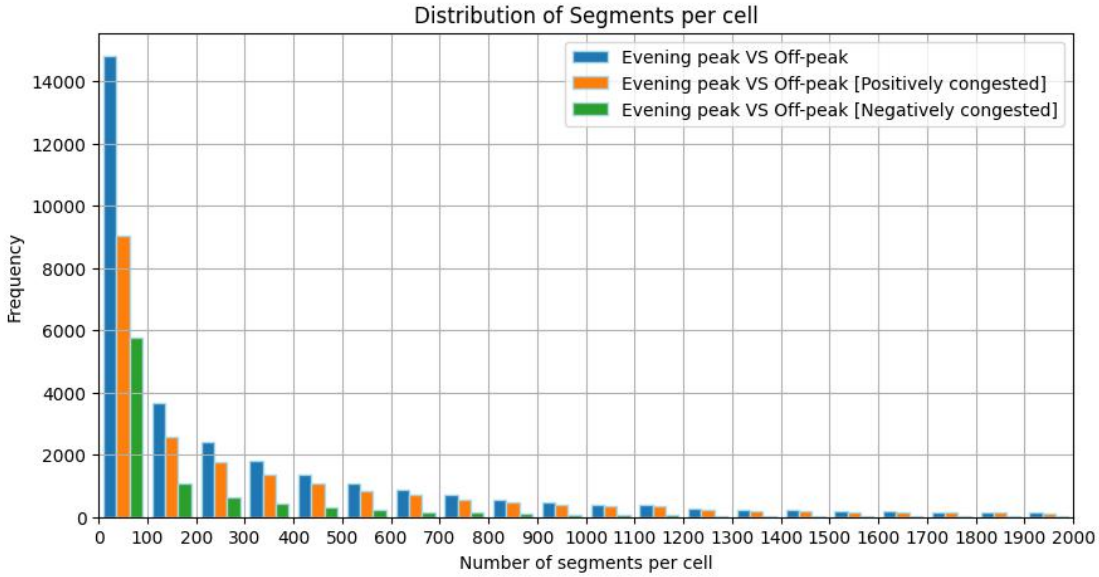

Figure 3.24: Distribution of cell-wise traffic congestion

Filtering cells

Based on the cells behavior that is discussed in 3.5.3, it is already observed that the cells with a lower number of segments inside are more prone to demonstrating unreliable results like negative congestion. As a result, the cells with less than 100 segments and also segments with less than 200 segments are filtered separately, and then they are plotted against the not filtered cumulative distribution and the results are shown in the figure 3.25

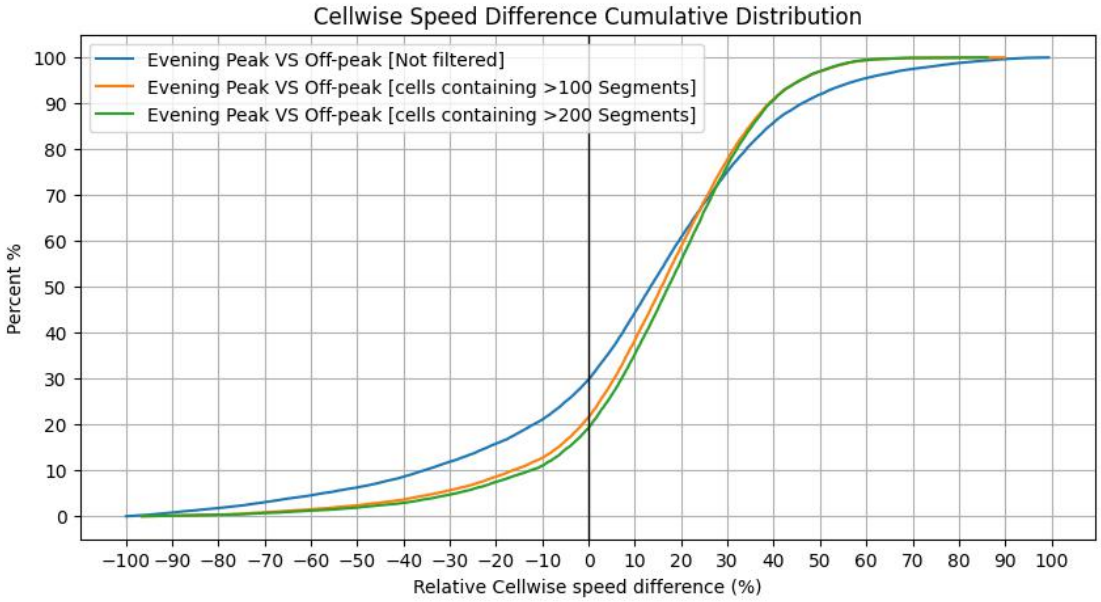

Figure 3.25: comparison of cell-wise cumulative distribution of relative traffic congestion in filtered and not filtered scenarios

As can be observed, the overall behavior of the relative congestion index is significantly better after doing the filtration because there are less cells with negative congestion. Furthermore, in the region with positive congestion, the non-filtered line (blue) is flatter and hence less informative. When cells with less than 100 segments per cell are filtered, the improvement in behavior is more noticeable. When the filtering criteria is increased to 200, the improvement is not significant, thus the threshold of 100 segments per cell is deemed to filter.

The next filtering criteria is to remove the cells that are showing negative congestion. Because achieving the goal of the thesis requires concentrating on locations that are experiencing the highest levels of traffic congestion, the point of interest is not a single cell but rather the behavior of a group of cells that are adjacent to one another. Figures 3.26 and 3.27 show the distribution of cell-wise relative speed difference after applying the filter

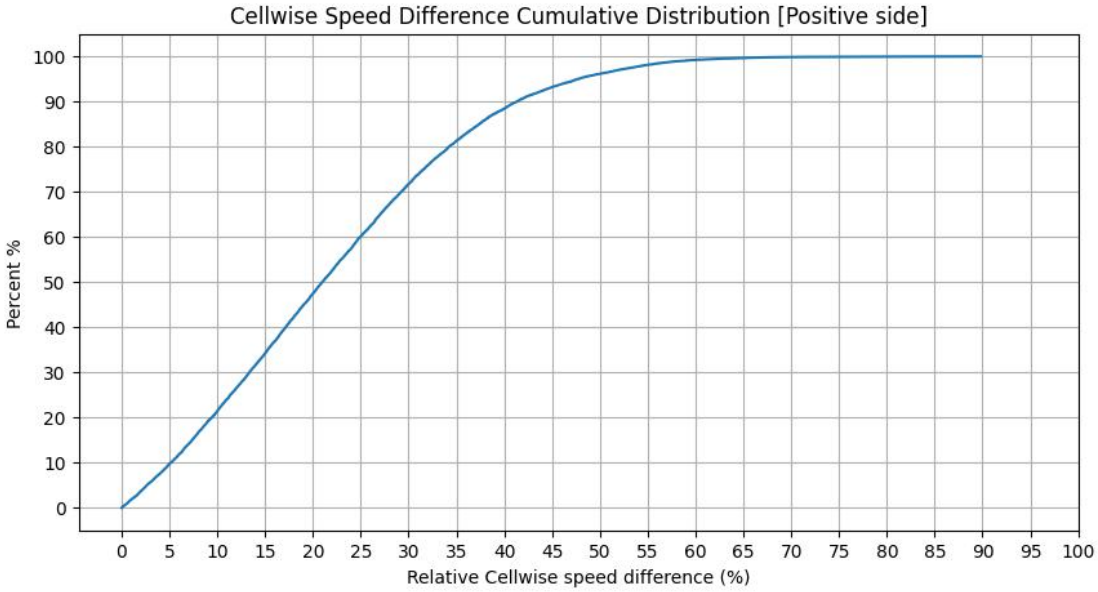

Figure 3.26: Filtered cell-wise cumulative distribution of relative traffic congestion

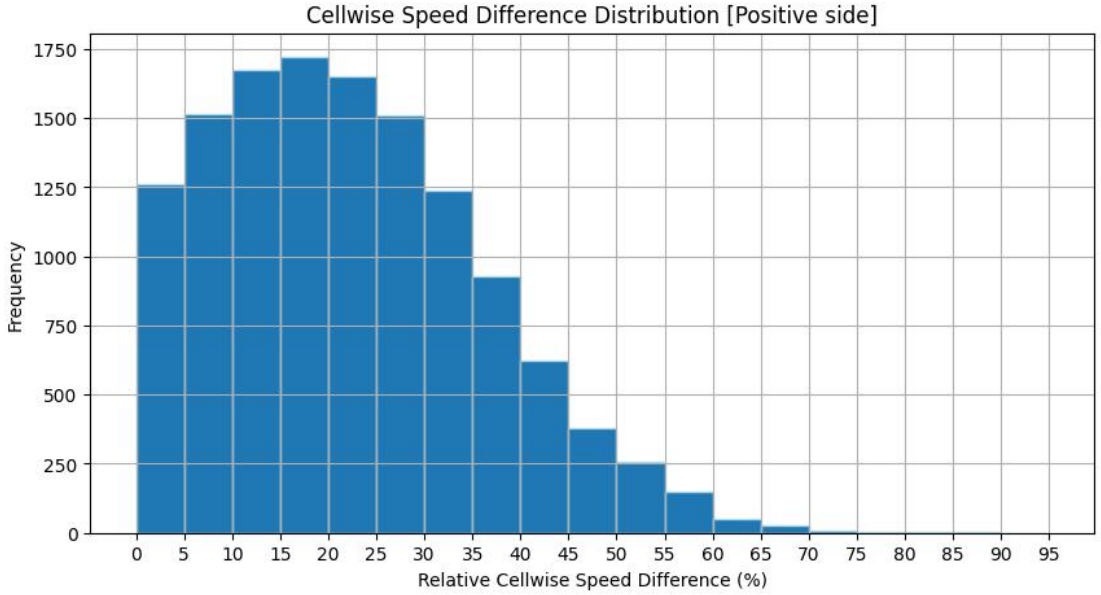

Figure 3.27: Filtered cell-wise distribution of relative traffic congestion

As it can be seen, the relative congestion index is peaked at the range between 15% and 20% that means the severity of the congestion is mostly occurs around this range.

3.6 Identification of congested areas

The criteria for filtering the cells in the grid are broken down and explained in 3.5.3. After the filters have been applied, everything is ready to show which cells and eventually which areas are the most congested based on the relative congestion index 3.5.2. It should not come as a shock that the downtown area is the most likely

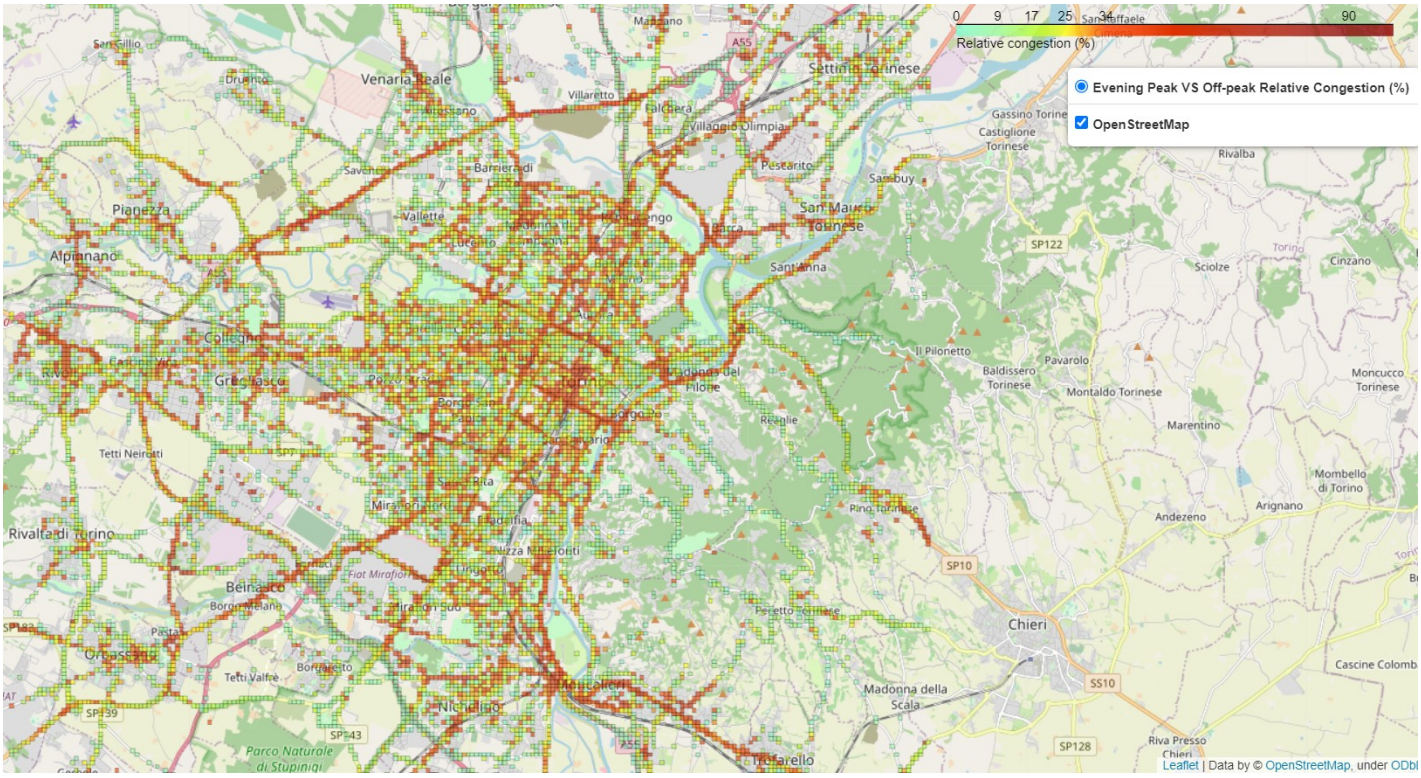

Figure 3.28: Relative traffic congestion

zone to be highly congested given the evidence presented here; it is the downtown area, after all. However, this is not the only area with a significant amount of traffic congestion; it is also possible to see that the traffic congestion occurs on the highways, despite the fact that the average speed may not be particularly slow; however, the speed difference in terms of percentage demonstrates that the highways may also have a remarkable amount of traffic congestion. This was able to be seen because of the relative congestion calculations, which makes it possible to observe this phenomenon. On the other hand, such a thing was not able to be observed when calculating the absolute traffic congestion index.

3.6.1 Kernel Density Estimation: a method for smoothing

Based on the fact that the structure of the project is a grid of cells, the overall image is still a grid of individual cells, each of which depicts the congestion in corresponding coordinates individually. This is the case even after the congestion index of the cells has been identified. However, According to the objective of the thesis, which is to identify the zones, and because each zone is most likely composed of a couple of cells, a method is required to demonstrate the behavior of a group of cells in an area rather than the behavior of individual cells. To smooth out the density, one of the approaches that might be used is to implement Kernel Density Estimation (KDE), which is a method that estimates density. This would be one of the available methods. In this instance, the relative congestion index, which is stated as a percentage, is taken into consideration as density, and several KDE models with varying amounts of bandwidth are trained. The amount of available bandwidth is the primary factor that determines how smooth the outcomes will be. The findings are shown in the charts that follow.

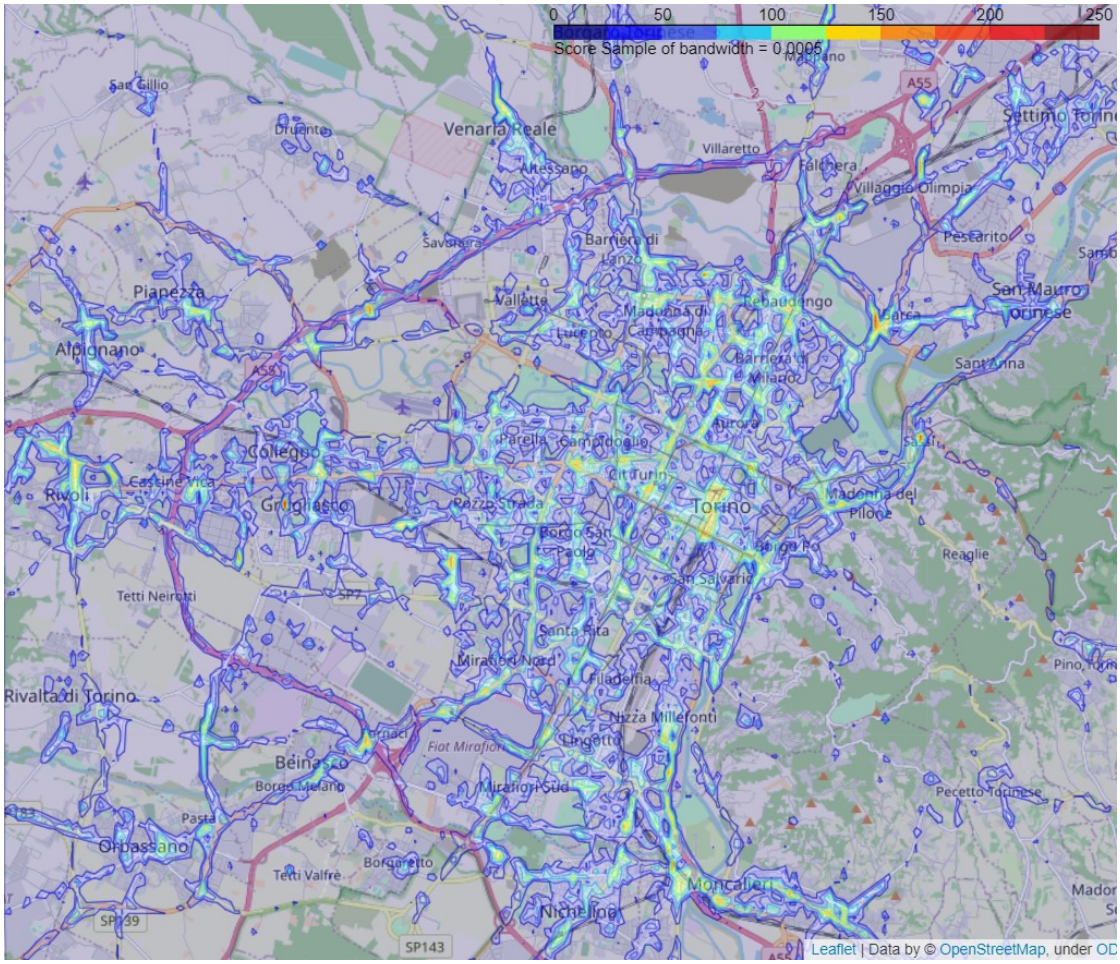

Figure 3.29: KDE of Relative traffic congestion, Bandwidth = 0.0005

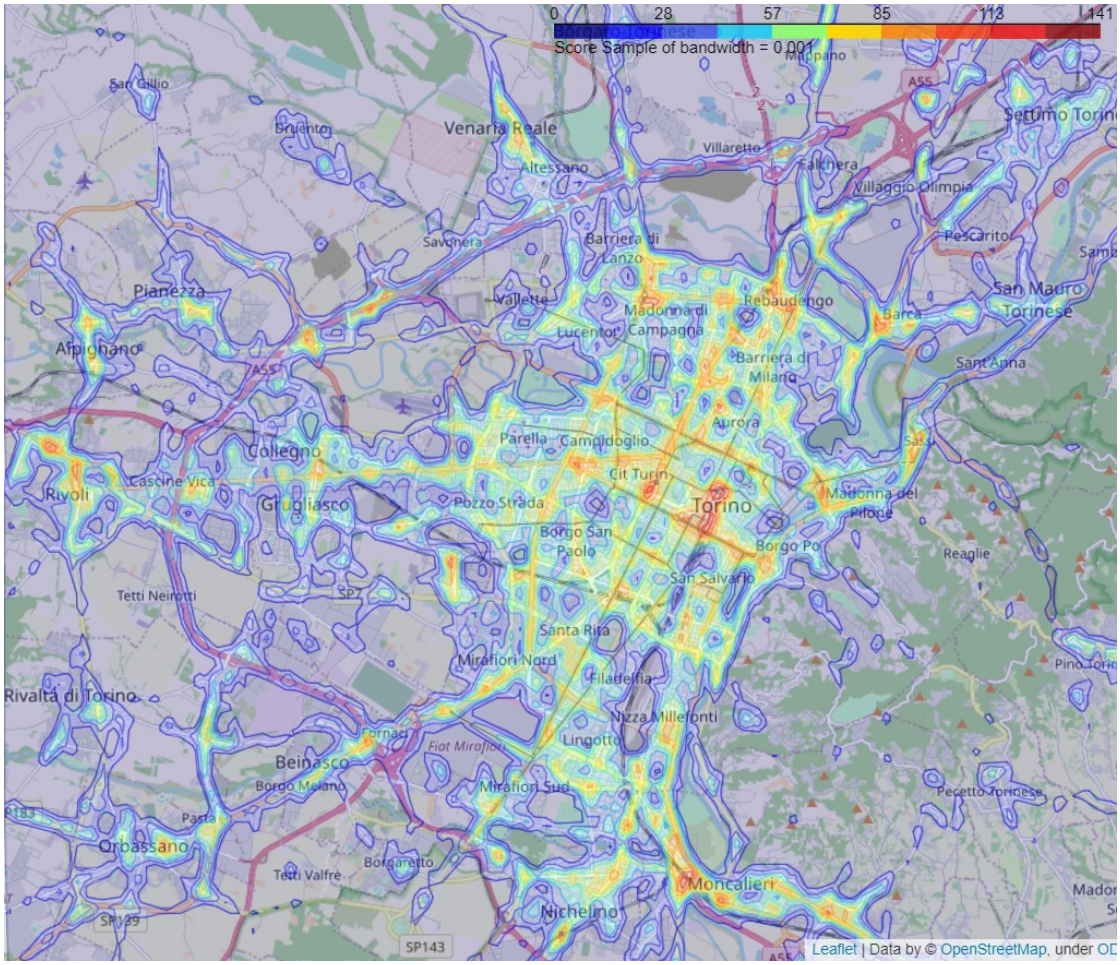

Figure 3.30: KDE of Relative traffic congestion, Bandwidth = 0.001

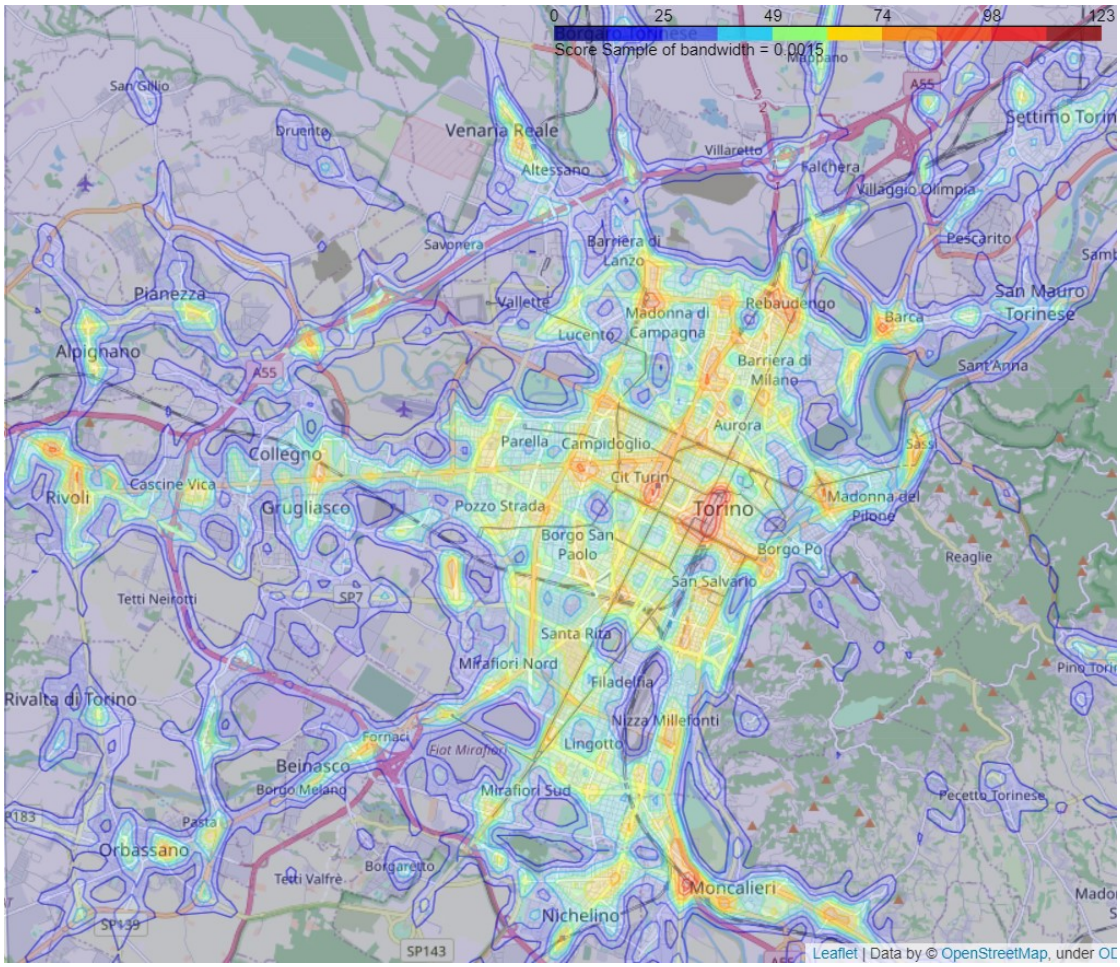

Figure 3.31: KDE of Relative traffic congestion, Bandwidth = 0.0015

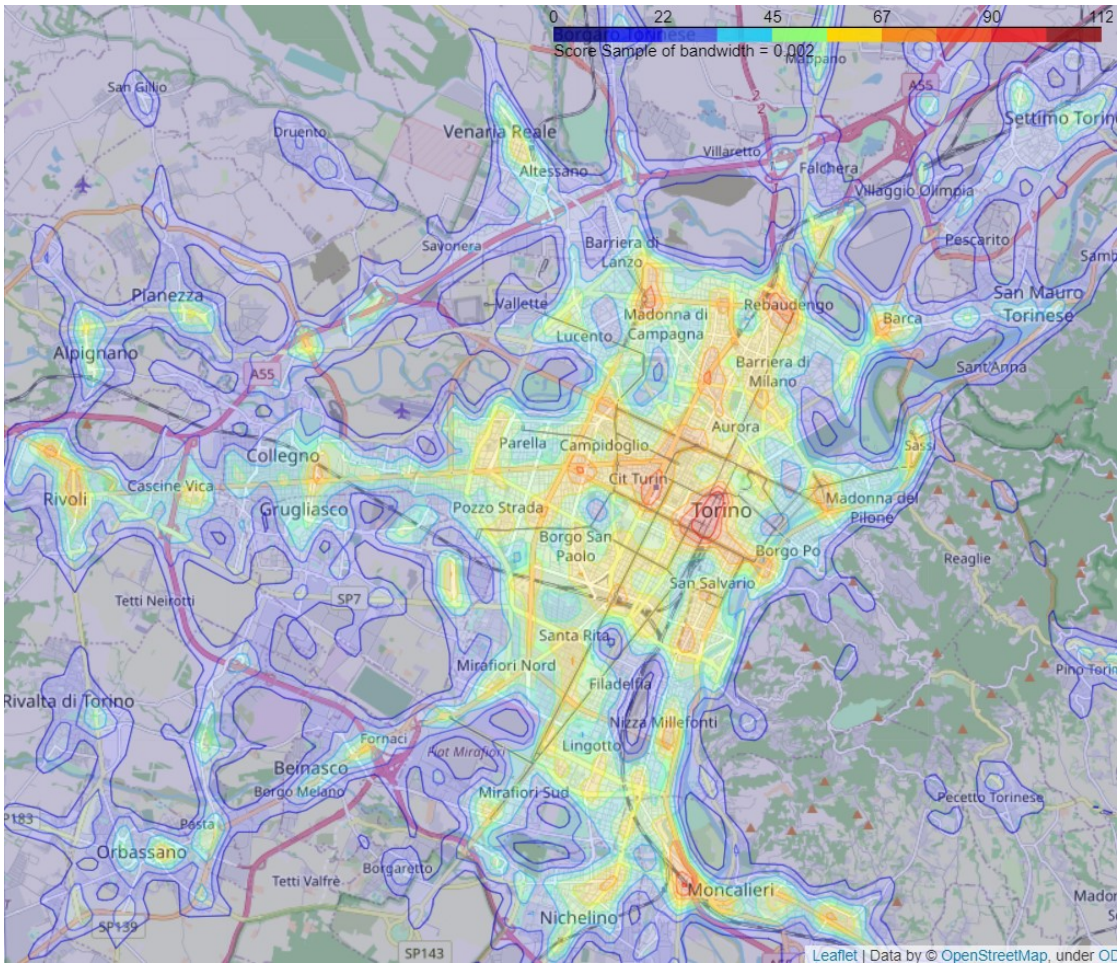

Figure 3.32: KDE of Relative traffic congestion, Bandwidth = 0.002

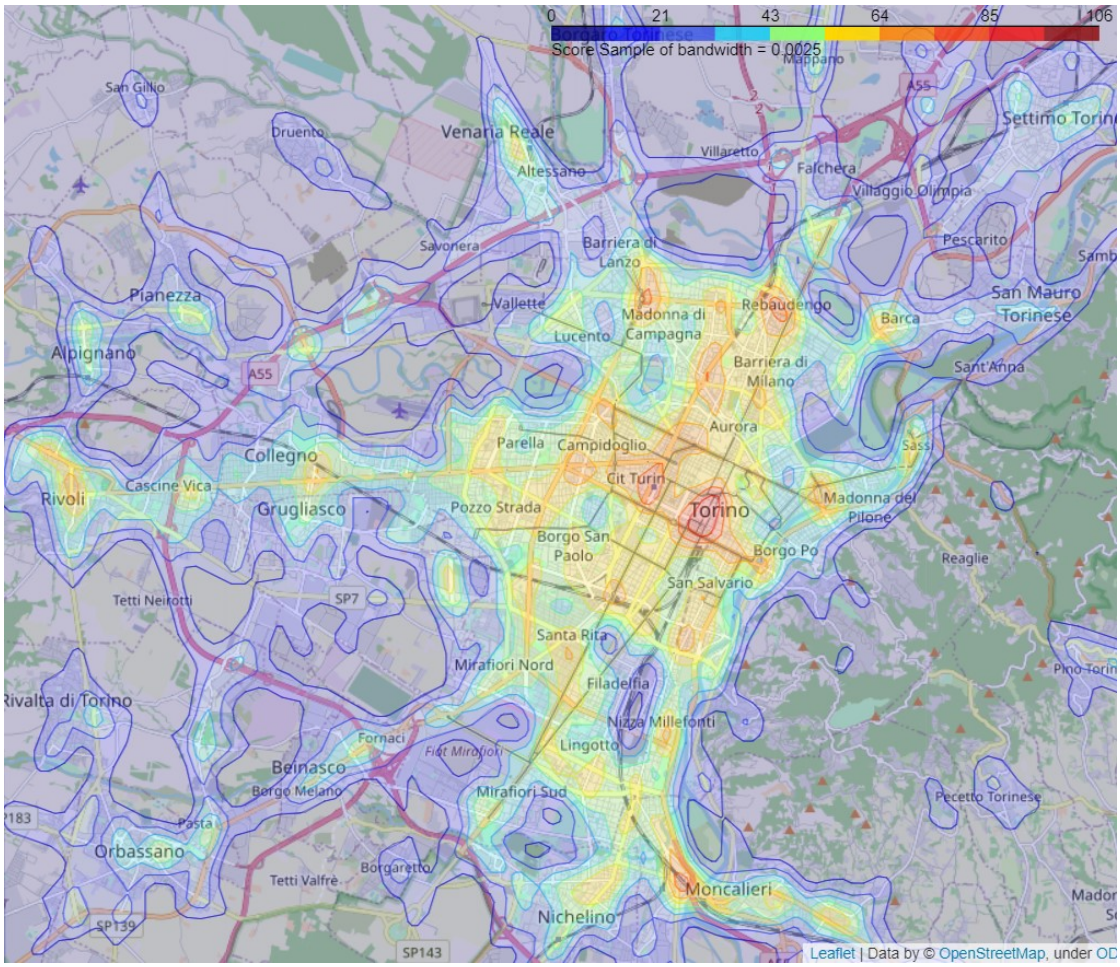

Figure 3.33: KDE of Relative traffic congestion, Bandwidth = 0.0025

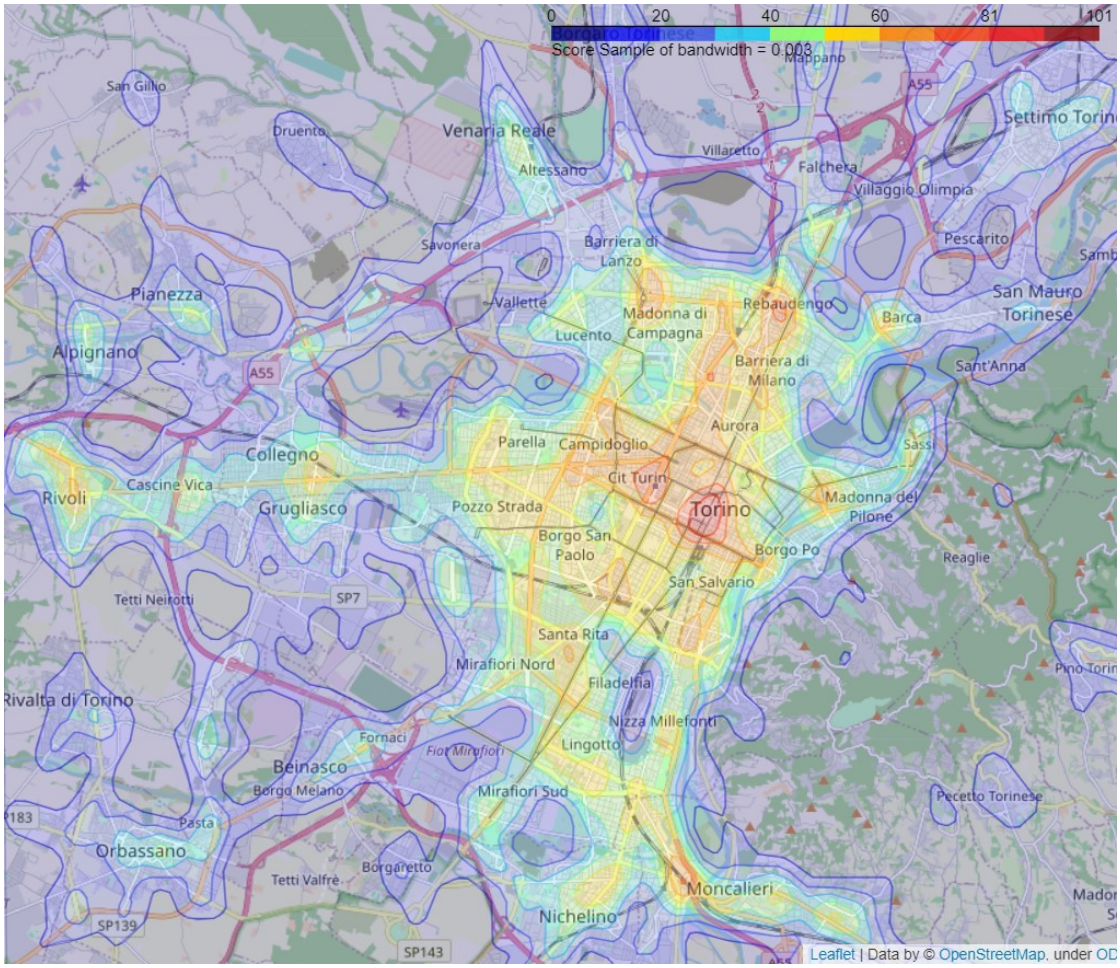

Figure 3.34: KDE of Relative traffic congestion, Bandwidth = 0.003

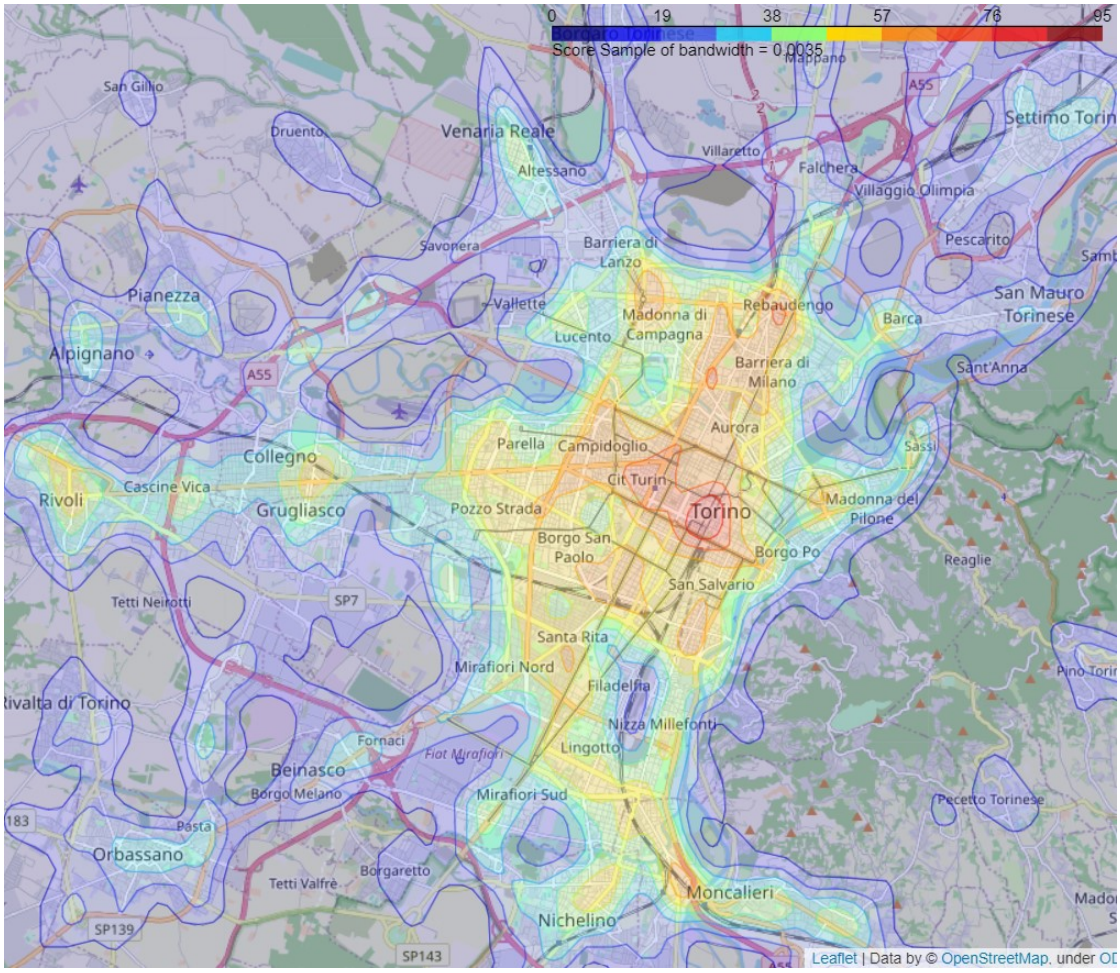

Figure 3.35: KDE of Relative traffic congestion, Bandwidth = 0.0035

As can be seen in 3.29 3.30 3.31 3.32 3.33 3.34 3.35, as bandwidth grows, surrounding cells begin to merge more and more, and this continues until the whole of the city is combined into one single cell. A bandwidth of 0.001 is deemed to be an appropriate bandwidth for the purpose of highlighting the locations that are suffering the most from traffic congestion.

Plot of the congested areas

as the final step of the identification of congested areas, in order to highlight the congested areas, a filter is applied to the bandwidth of 0.001 to demonstrate the congested areas clearly. The result is shown in 3.36.

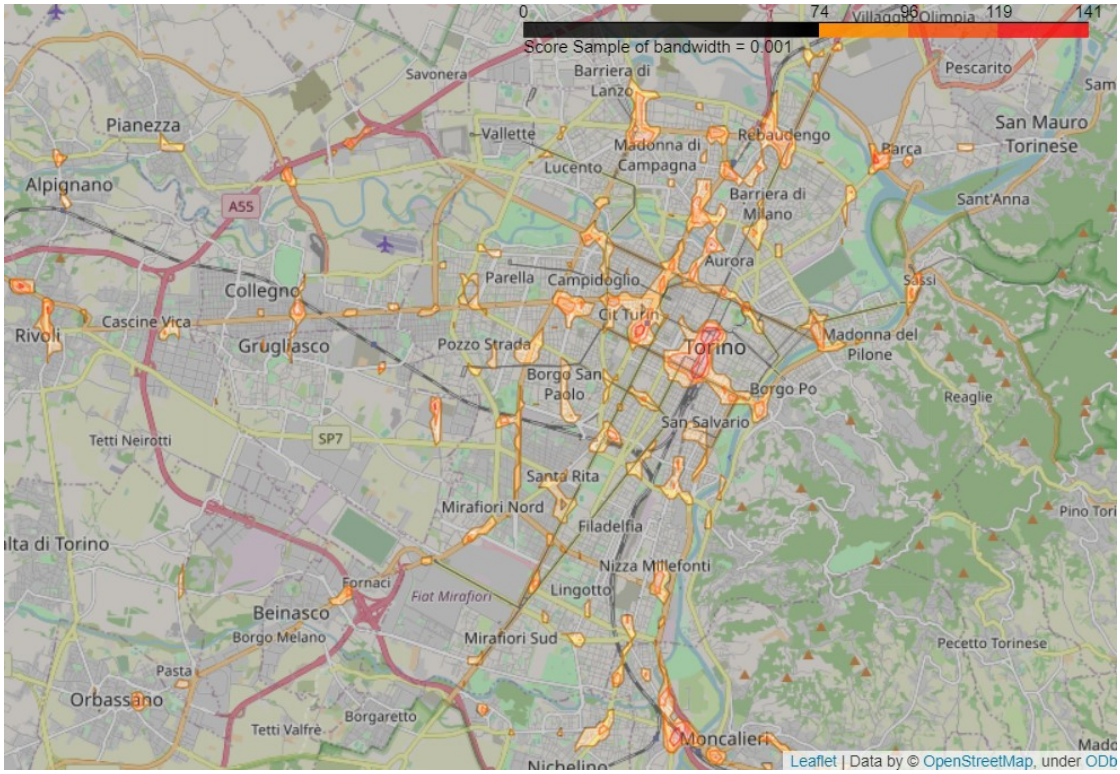

Figure 3.36: Highlight of the most congested areas, Bandwidth = 0.001

Chapter 4

Conclusions

The subject that was suggested for this study was motivated mostly by the two key questions that are outlined below.

- How is it possible to extract traffic congestion pattern from FCD using a data-driven approach?
- How much is it scalable to expand the area under investigation? and what could be the best resolution to achieve?

We realized that it is possible to extract the traffic congestion pattern by making a comparison of the average speed of vehicles in the same area but at different time intervals throughout the day. We did this by integrating a large quantity of FCD that had already been collected into the analysis of this thesis. We came to this conclusion after realizing that it is possible to do so. In addition, we were aware that, despite the fact that Politecnico di Torino has a very robust hardware infrastructure, the experiment's reach could not be expanded beyond the confines of a metropolitan urban region. In addition, because to the limitations imposed by the computing capability of the hardware, the resolution may be reduced the greater the region that is the subject of the study.

A further consideration is the application of the grid over the top of the map. As we have seen, it is necessary to take into account the behavior of a group of cells that are adjacent to one another in order to draw a conclusion about the degree to which there is an issue with traffic congestion in the region. This is the case even though some of the individual cells of the grid have displayed undesirable behaviors. This could be done by doing a smoothing step on all of the grid cells as the last step. This would give a better picture of the areas with a lot of traffic.

The existing findings might be used in a later piece of research on this subject in order to apply certain machine learning techniques in order to create predictions regarding the congestion index in the future.

Bibliography

[1] Caves, R. W. (2004). Encyclopedia of the City. Routledge. p. 141.

[2] Treiber, Martin; Kesting, Arne (2012-10-11). Traffic Flow Dynamics: Data, Models and Simulation. Springer Science Business Media. ISBN 978-3-642-32459-8.

[3] May, Adolf Darlington (1990). Traffic Flow Fundamentals. Prentice Hall. ISBN 9780139260728.

[4] https://www.brookings.edu/research/traffic-why-its-getting-worse-what-government-can-do/.

[5] Agyapong F and Ojo T K 2018 Managing traffic congestion in the accra central market, Ghana Journal of Urban Management. 7 pp 85-96

[6] `https://translineinc.com/traffic-data-collection/#:~:text=Other%20methods%20of%20collecting%20traffic,the%20direction%20of%20traffic%20flow.`

[7] Leduc, Guillaume. (2008). Road Traffic Data: Collection Methods and Applications.

[8] G.S. Larue, C. Wullems, A new method for evaluating driver behavior and interventions for passive railway level crossings with pneumatic tubes [J]. J. Transportation Saf. Secur. 11(2), 150–166 (2019)

[9] S. Rajab, M.O. Al Kalaa, H. Refai, Classification and speed estimation of vehicles via tire detection using single-element piezoelectric sensor[J]. J. Adv. Transportation 50(7), 1366–1385 (2016)

[10] M. Grote, I. Williams, J. Preston, et al., A practical model for predicting road traffic carbon dioxide emissions using Inductive loop detector data[J]. Transportation Res. Part D Transp. Environ. 63, 809–825 (2018)

[11] `https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT`

[12] Llorca, D.F., Sotelo, M.A., Sánchez, S. et al. Traffic Data Collection for Floating Car Data Enhancement in V2I Networks. EURASIP J. Adv. Signal Process. 2010, 719294 (2010)

[13] https://www.septentrio.com/en/learn-more/about-GNSS/why-multi-frequency-and-multi-constellation-matters
[14] Travel Time Data Collection Handbook, FHWA report, chapter 5, ITS Probe Vehicle Techniques, 1998.
[15] Chang, Kang-tsung (2016). Introduction to Geographic Information Systems (9th ed.). McGraw-Hill. p. 24. ISBN 978-1-259-92964-9.
[16] https://en.wikipedia.org/wiki/Geographic_coordinate_system
[17] https://www.ga.gov.au/scientific-topics/positioning-navigation/wgs84
[18] https://www.gpsworld.com/data-collection-of-wgs-84-information-or-is-it/
[19] Kuhlman,Dave."APythonBook:BeginningPython,AdvancedPython, andPythonExercises"
[20] Wes McKinney et al. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference, volume 445, pages 51–56. Austin, TX, 2010.
[21] https://pandas.pydata.org/
[22] Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, Jacob Wasserman, James McBride, Jeffrey Gerard, Jeff Tratner, Matthew Perry, Adrian Garcia Badaracco, Carson Farmer, Geir Arne Hjelle, Alan D. Snow, Micah Cochran, Sean Gillies, Lucas Culbertson, Matt Bartos, Nick Eubank, maxalbert, Aleksey Bilogur, Sergio Rey, Christopher Ren, Dani Arribas-Bel, Leah Wasser, Levi John Wolf, Martin Journois, Joshua Wilson, Adam Greenhall, Chris Holdgraf, Filipe, and François Leblanc. geopandas/geopandas: v0.8.1, July 2020.
[23] https://geopandas.org/
[24] https://numpy.org/
[25] J. D. Hunter. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, 9(3):90–95, 2007.
[26] https://matplotlib.org/
[27] https://pypi.org/project/Shapely/
[28] https://www.dominodatalab.com/data-science-dictionary/folium
[29] https://python-visualization.github.io/folium/
[30] https://spark.apache.org/
[31] https://sparkbyexamples.com/
[32] Wu, Shaomin (2013) A Review on Coarse Warranty Data and Analysis. Reliability Engineering and System Safety, 114 . pp. 1-11. ISSN 0951-8320.
[33] https://www.tableau.com/learn/articles/what-is-data-cleaning
[34] Dudek, Gregory; Jenkin, Michael (2000). Computational Principles of Mobile Robotics
[35] https://en.wikipedia.org/wiki/Dilution_of_precision_(navigation)
[36] https://web.archive.org/web/20141122153439/http://www.gmat.unsw.edu.au/snap/gps/gps_survey/chap1/149.htm

[37] Isik, O.K.; Hong, J.; Petrunin, I.; Tsourdos, A. Integrity Analysis for GPS-Based Navigation of UAVs in Urban Environment. Robotics 2020, 9, 66. https://doi.org/10.3390/robotics9030066

[38] Biljecki, F., Ledoux, H., Van Oosterom, P. (2013): Transportation mode-based segmentation and classification of movement trajectories. International Journal of Geographical Information Science, 27(2), pp. 385-407

[39] `https://desktop.arcgis.com/en/arcmap/latest/map/page-layouts/what-are-grids-and-graticules-.htm`

[40] `https://smartdata.polito.it/computing-facilities/`

[41] T. Kos, I. Markezic and J. Pokrajcic, "Effects of multipath reception on GPS positioning performance," Proceedings ELMAR-2010, 2010, pp. 399-402.

[42] Ajmar, A., Arco, E., Boccardo, P., Perez, F. (2019). Floating car data (fcd) for mobility applications. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, 42, 1517-1523.

[43] Schäfer, R.P., Thiessenhusen, K.U., Wagner, P., 2002. A Traffic Information System by Means of Real-Time Floating- Car Data. In: 9th World Congress on Intelligent Transport Systems.

[44] BS Kerner, C Demir, RG Herrtwich, SL Klenov, H Rehborn, A Haug, et al. Traffic state detection with floating car data in road networks. In Intelligent Transportation Systems, Proceedings. IEEE, pages 44–49. IEEE, 2005.

[45] Xiaowen Dai, Martin Ferman, Robert P Roesser, et al. A simulation evaluation of a real-time traffic information system using probe vehicles. In Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE, volume 1, pages 475–480. IEEE, 2003

[46] Sunderrajan, Abhinav, et al. "Traffic state estimation using floating car data." Procedia Computer Science 80 (2016): 2008-2018.

[47] Zhao, N., Yu, L., Zhao H., Guo, J., and Wen, H., 2009. Analysis of Traffic Flow Characteristics on Ring Road Expressways in Beijing: Using Floating Car Data and Remote Traffic Microwawe Sensor Data. Transportation Research Record: Journal of the Transportation Research Board, pp. 178-185.

[48] Haghani, A., Hamedi, M., Sadabadi, K.F., Young, S., and Tarnoff, P., 2010. Data Collection of Freeway Travel Time Ground Truth with Bluetooth Sensors. Journal of Transportation research record, pp. 60-68.

[49] Oruc Altintasi, Hediye Tuydes-Yaman, Kagan Tuncay, Detection of urban traffic patterns from Floating Car Data (FCD), Transportation Research Procedia, Volume 22, 2017, Pages 382-391, ISSN 2352-1465, https://doi.org/10.1016/j.trpro.2017.03.057.

[50] Croce, A.I.; Musolino, G.; Rindone, C.; Vitetta, A. Transport System Models and Big Data: Zoning and Graph Building with Traditional Surveys, FCD and GIS. ISPRS Int. J. Geo-Inf. 2019, 8, 187. https://doi.org/10.3390/ijgi8040187

[51] Cantelmo, Guido, and Francesco Viti. "A big data demand estimation model for

urban congested networks." Transport and Telecommunication (2020).

Appendix A

Source code

Reading and filtering initial data

```
1  from pyspark.sql import functions as F
2  import matplotlib.pyplot as plt
3  from pyspark.sql.types import *
4  import seaborn as sns
5  import pandas as pd
6  import numpy as np
7  import geopy.distance
8  import matplotlib
9  import datetime
10 import os
11 from haversine import haversine, Unit
12
13 import fastplot
14 %matplotlib inline
15
16 #schema of initial data
17 schema_initial_data = 'idRequest string, deviceId string,
       dateTime string, \
18 latitude double, longitude double, speedKmh Integer,
       heading Integer, \
19 accuracyDrop Integer, EngineStatus Integer, Type Integer'
20 #reading csv
21 path = os.path.abspath(os.getcwd())
22 df = spark.read.csv('file:///%s../../shared/data/
       FCD_complete/*/*'%path, sep=",", schema =
       schema_initial_data)
```

```
23
24 #dropping duplicates
25 df = df.distinct()
26
27 df = df.na.drop("any")
28
29 #dropping points with accuracy higher than 20
30 df = df.filter(df['accuracyDrop'] <= 20)
31
32 #dateTime format
33 df = df.withColumn('dateTime', F.to_timestamp('dateTime'))
34
35 # ========== Torino and countryside
      ==========
36 minLat = 44.96282106687191
37 minLon = 7.502048016422193
38 maxLat = 45.19265016665321
39 maxLon = 7.791812422724604
40 # ========================================
41 @F.udf(returnType=BooleanType())
42 def drop_outside_points(lat, lon):
43     condition = (lat >= minLat) & (lat <= maxLat) & (lon >=
          minLon) & (lon <= maxLon)
44     return condition
45
46 df = df.filter(drop_outside_points('latitude', 'longitude')
      )
47
48 # Adding calculated segment parameters to initial data
49 schema_inital_segment = 'index Integer,idRequest string,
      deviceId string,dateTime timestamp,\
50 latitude double,longitude double,speedKmh Integer,
      heading Integer,\
51 accuracyDrop Integer,EngineStatus Integer,Type Integer,
      segmentDistance double,\
52 segmentDuration double,segmentSpeedKmH double'
53
54 @F.pandas_udf(schema_inital_segment, functionType=F.
      PandasUDFType.GROUPED_MAP)
55 def adding_segment_parameter(df):
56     os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"
```

```
57
58      df = df.sort_values(by="dateTime")
59      df.reset_index(inplace=True)
60      distance=[float(0)]
61      duration=[float(0)]
62      speed=[float(0)]
63
64      for index,row in df.iterrows():
65          if index == len(df)-1:
66              pass
67          else:
68              time_difference = df.loc[index+1,'dateTime'] -
                    df.loc[index,'dateTime']
69              time_difference = time_difference.total_seconds
                    ()
70              duration.append(time_difference)
71
72              org = (df.loc[index,'latitude'],df.loc[index,'
                    longitude'])
73              des = (df.loc[index+1,'latitude'],df.loc[index
                    +1,'longitude'])
74
75              distance.append(float("{0:.2f}".format(geopy.
                    distance.distance(org,des).m)))
76
77              try:
78                  speed.append(geopy.distance.distance(org,
                        des).m/time_difference)
79              except:
80                  speed.append(float("NaN"))
81
82      df['segmentDistance'] = distance
83      df['segmentDuration'] = duration
84      df['segmentSpeedKmH'] = [v*3.6 if not pd.isna(v) else
            float("NaN") for v in speed ]
85      return df
86
87  df_added_parameters = df.groupby("deviceId").apply(
        adding_segment_parameter)
88  df_added_parameters = df_added_parameters.drop('index')
```

```
89 df_added_parameters.write.csv('file:///%s/all_merged/'%path
       , sep=",")
```

Creating segment table

```
1  from pyspark.sql import functions as F
2  import matplotlib.pyplot as plt
3  from pyspark.sql.types import *
4  import seaborn as sns
5  import pandas as pd
6  import numpy as np
7  import geopy.distance
8  import matplotlib
9  import datetime
10 import os
11
12 import fastplot
13 %matplotlib inline
14
15 #schema of initial data
16 schema_inital_segment = 'idRequest string, deviceId string,
       dateTime timestamp, \
17 latitude double, longitude double, speedKmh Integer,
       heading Integer,\
18 accuracyDrop Integer, EngineStatus Integer, Type Integer,
       segmentDistance double,\
19 segmentDuration double, segmentSpeedKmH double'
20 #reading csv
21 path = os.path.abspath(os.getcwd())
22 df = spark.read.csv('file:///%s/all_merged/*'%path,sep=",",
        schema = schema_inital_segment)
23
24 #dateTime format
25 df = df.withColumn('dateTime', F.to_timestamp('dateTime'))
26
27 schema_segment_inital ='deviceId String,\
28 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣type Integer,\
29 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣dateTime string,\
30 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startLatitude double,\
31 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startLongitude double,\
32 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startEngineStatus Integer,\
33 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startAccuracyDrop Integer,\
```

```
34 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endAccuracyDrop␣Integer,\
35 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endEngineStatus␣Integer,\
36 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endLatitude␣double,\
37 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endLongitude␣double,\
38 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣segmentDistance␣double,\
39 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣segmentDuration␣double,\
40 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣segmentSpeedKmH␣double'
41
42 @F.pandas_udf(schema_segment_inital, functionType=F.
       PandasUDFType.GROUPED_MAP)
43 def segment_table_maker(df):
44     os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"
45     df = df.sort_values(by="dateTime")
46     df.reset_index(inplace=True)
47     df_segment_temp=pd.DataFrame()
48
49     '''Creating empty lists for columns '''
50     deviceId = []; dateTime = [];
51     startLat = []; startLon = []; startEngineStatus = [];
           startAccuracyDrop = [];
52     endLat = []; endLon = []; endEngineStatus = [];
           endAccuracyDrop = [];
53     distance = []; duration = []; speed = [];
54     Type = df['Type'][0]
55     for index,row in df.iterrows():
56         if index == 0:
57             pass
58         else:
59             deviceId.append(row['deviceId'])
60             dateTime.append(str(df.loc[index-1,'dateTime'])
                   )
61             startLat.append(df.loc[index-1,'latitude'])
62             startLon.append(df.loc[index-1,'longitude'])
63             startEngineStatus.append(df.loc[index-1,'
                   EngineStatus'])
64             startAccuracyDrop.append(df.loc[index-1,'
                   accuracyDrop'])
65             endLat.append(row['latitude'])
66             endLon.append(row['longitude'])
67             endEngineStatus.append(row['EngineStatus'])
68             endAccuracyDrop.append(row['accuracyDrop'])
```

```
69                distance.append(row['segmentDistance'])
70                duration.append(row['segmentDuration'])
71                speed.append(row['segmentSpeedKmH'])
72
73
74        df_segment_temp['deviceId'] = deviceId
75        df_segment_temp['type'] = Type
76        df_segment_temp['dateTime'] = dateTime
77        df_segment_temp['startLatitude'] = startLat
78        df_segment_temp['startLongitude'] = startLon
79        df_segment_temp['startLongitude'] = startLon
80        df_segment_temp['startEngineStatus'] =
              startEngineStatus
81        df_segment_temp['startAccuracyDrop'] =
              startAccuracyDrop
82        df_segment_temp['endLatitude'] = endLat
83        df_segment_temp['endLongitude'] = endLon
84        df_segment_temp['endEngineStatus'] = endEngineStatus
85        df_segment_temp['endAccuracyDrop'] = endAccuracyDrop
86        df_segment_temp['segmentDistance'] = distance
87        df_segment_temp['segmentDuration'] = duration
88        df_segment_temp['segmentSpeedKmH'] = speed
89
90        return df_segment_temp
91
92  df_segment_temp = df.groupby("deviceId").apply(
        segment_table_maker)
93
94  '''applying filter: dropping segment durations longer than
        70s
95  as well as segments with speed higher than 130KmH or equal
        to absolute zero'''
96  df_segment_temp = df_segment_temp.filter(df_segment_temp['
        segmentDuration']<=70)
97  df_segment_temp = df_segment_temp.filter(df_segment_temp['
        segmentSpeedKmH']<=130)
98  df_segment_temp = df_segment_temp.filter(df_segment_temp['
        segmentSpeedKmH']!=0)
99
100 '''adding columns related to grid table'''
101 schema_segments_ultimate ='deviceId␣String,\
```

```
102 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣type␣Integer,\
103 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣dateTime␣string,\
104 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startLatitude␣double,\
105 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startLongitude␣double,\
106 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startEngineStatus␣Integer,\
107 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣startAccuracyDrop␣Integer,\
108 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endAccuracyDrop␣Integer,\
109 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endEngineStatus␣Integer,\
110 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endLatitude␣double,\
111 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endLongitude␣double,\
112 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣segmentDistance␣double,\
113 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣segmentDuration␣double,\
114 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣segmentSpeedKmH␣double,\
115 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣xtmp␣double,\
116 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣ytmp␣double,\
117 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣start_x␣integer,\
118 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣start_y␣integer,\
119 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣end_x␣integer,\
120 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣end_y␣integer,\
121 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣cell_id_start␣string,\
122 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣cell_id_end␣string'
123
124 '''shift by 100m in degrees'''
125 shiftInMeterLat = 0.0008983152841182118 #100m
126 shiftInMeterLon = 0.001270644533487797
127 # ========= Torino and countryside =======
128 minLat = 44.96282106687191
129 minLon = 7.502048016422193
130 maxLat = 45.19265016665321
131 maxLon = 7.791812422724604
132 # ========================================
133
134 @F.pandas_udf(schema_segments_ultimate, functionType=F.
    PandasUDFType.GROUPED_MAP)
135 def segment_position_calculator(df_segments):
136     os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"
137
138     df_segments = df_segments.sort_values(by="dateTime")
139     df_segments.reset_index(drop = True, inplace=True)
140
```

```
141     '''calculating the cells that these segments belong to
            '''
142     df_segments['xtmp'] = (df_segments['startLongitude']-
            minLon)/shiftInMeterLon
143     df_segments['ytmp'] = (df_segments['startLatitude']-
            minLat)/shiftInMeterLat
144     df_segments['start_x'] = (df_segments['xtmp'].apply(
            lambda x : np.floor(x)))+1
145     df_segments['start_y'] = (df_segments['ytmp'].apply(
            lambda y : np.floor(y)))+1
146
147     df_segments['xtmp'] = (df_segments['endLongitude']-
            minLon)/shiftInMeterLon
148     df_segments['ytmp'] = (df_segments['endLatitude']-
            minLat)/shiftInMeterLat
149     df_segments['end_x'] = (df_segments['xtmp'].apply(
            lambda x : np.floor(x)))+1
150     df_segments['end_y'] = (df_segments['ytmp'].apply(
            lambda y : np.floor(y)))+1
151     df_segments['cell_id_start'] = [f'{int(df_segments.
            start_y.values[i])}_{int(df_segments.start_x.values[
            i])}'
152                             for i in range(len(df_segments))]
153     df_segments['cell_id_end'] = [f'{int(df_segments.end_y.
            values[i])}_{int(df_segments.end_x.values[i])}'
154                             for i in range(len(df_segments))]
155     return df_segments
156
157 df_segment = df_segment_temp.groupby("deviceId").apply(
        segment_position_calculator)
158
159 df_segment.write.csv('file:///%s/segment_table'%path, sep="
        ,")
```

Grid creation on the map with cells with size 100m by 100m

```
1 import pandas as pd
2 import geopandas as gpd
3 from shapely.geometry import Point, Polygon, shape
4 import math
5
6 df_segments = pd.read_csv(r'segment_table.csv')
```

```
7
8  #%% boundary of interest area
9  minLat = 44.96282106687191
10 minLon = 7.502048016422193
11 maxLat = 45.19265016665321
12 maxLon = 7.791812422724604
13
14 #%% creating empty dataframes to initialize
15 df_geojson = pd.DataFrame()
16 df_geojson['polygon_id'] = df_geojson['polygon'] = ""
17 df_data = pd.DataFrame()
18
19 #%%
20 shiftInMeterLat = 0.0008983152841182118 #100m
21 shiftInMeterLon = 0.001270644533487797
22
23 # grid creation with 100m by 100m steps
24 range_lat = (maxLat - minLat)/shiftInMeterLat
25 range_lon = (maxLon - minLon)/shiftInMeterLon
26 coords={}
27 minLonCopy = minLon
28 #creating cells with corresponding id
29 for i in range(math.ceil(range_lat)):
30     minLon = minLonCopy
31     new_latitude  = minLat + shiftInMeterLat
32     for j in range(math.ceil(range_lon)):
33         new_longitude = minLon + shiftInMeterLon
34         coords[f'{i+1}_{j+1}'] = [[minLon,minLat],[
              new_longitude,minLat],
35         [new_longitude,new_latitude],[minLon,new_latitude]]
36         minLon = new_longitude
37     minLat = new_latitude
38
39 #%% storing in a dataframe
40 df_geojson['polygon_id']=coords.keys()
41 df_geojson['polygon'] = coords.values()
42 df_geojson['polygon'] = df_geojson['polygon'].apply(Polygon
     )
43 df_data['id'] = coords.keys()
44
45 df_data['count'] = 0
```

```
df_data.to_csv('100m/data_TO.csv',index=False)
```

Grid table and plot over the map

```
from pyspark.sql import functions as F
import matplotlib.pyplot as plt
from pyspark.sql.types import *
import seaborn as sns
import pandas as pd
import numpy as np
import geopy.distance
import matplotlib
import datetime
import pickle
import os

#schema of segment table
schema_segments ='deviceId String,\
                  type Integer,\
                  dateTime string,\
                  startLatitude double,\
                  startLongitude double,\
                  startEngineStatus Integer,\
                  startAccuracyDrop Integer,\
                  endAccuracyDrop Integer,\
                  endEngineStatus Integer,\
                  endLatitude double,\
                  endLongitude double,\
                  segmentDistance double,\
                  segmentDuration double,\
                  segmentSpeedKmH double,\
                  xtmp double,\
                  ytmp double,\
                  start_x integer,\
                  start_y integer,\
                  end_x integer,\
                  end_y integer,\
                  cell_id_start string,\
                  cell_id_end string'
#reading csv
path = os.path.abspath(os.getcwd())
```

```
38 df_segments = spark.read.csv('file:///%s/segment_table'%
       path,sep=",", schema = schema_segments)
39 df_segments = df_segments.withColumn('dateTime', F.
       to_timestamp('dateTime'))
40
41 #converting dates to hour (for peak selection)
42 @F.udf()
43 def hour_extraction(dateTime):
44     hour = dateTime.hour
45     return hour
46
47 df_segments = df_segments.withColumn('dateTime',
       hour_extraction('dateTime'))
48
49 '''Peak 1'''
50 @F.udf(returnType=BooleanType())
51 def peak_1(hour):
52     valid_period = [9,10,11]
53     inside_peak = hour in valid_period
54     return inside_peak
55
56 df_peak_1 = df_segments.filter(peak_1('dateTime'))
57
58 '''Peak 2'''
59 @F.udf(returnType=BooleanType())
60 def peak_2(hour):
61     valid_period = [18,19,20]
62     inside_peak = hour in valid_period
63     return inside_peak
64
65 df_peak_2 = df_segments.filter(peak_2('dateTime'))
66
67 '''Non-Peak'''
68 @F.udf(returnType=BooleanType())
69 def off_peak(hour):
70     valid_period = range(1,7)
71     inside_peak = hour in valid_period
72     return inside_peak
73
74 df_off_peak = df_segments.filter(off_peak('dateTime'))
75
```

```
76 # grid table
77 schema_grid_calculations= StructType([
78     StructField("cell_id",StringType(),True),
79     StructField("num_segments",IntegerType(),True),
80 
81     StructField("minSpeed",DoubleType(),True),
82     StructField("avgSpeed",DoubleType(),True),
83     StructField("maxSpeed",DoubleType(),True),
84 
85     StructField("minAccuracyDrop",IntegerType(),True),
86     StructField("avgAccuracyDrop",DoubleType(),True),
87     StructField("maxAccuracyDrop",IntegerType(),True),
88   ])
89 
90 @F.pandas_udf(schema_grid_calculations, functionType=F.
       PandasUDFType.GROUPED_MAP)
91 def cell_data_calculator(df_segments):
92     os.environ["ARROW_PRE_0_15_IPC_FORMAT"] = "1"
93 
94 #       df_segments = df_segments.sort_values(by="dateTime")
95     df_segments.reset_index(drop = True, inplace=True)
96 
97     df_data = pd.DataFrame()
98 
99     #=============================================#
100    #============ min,avg,max calculations for SPEED
           ============#
101    minSpeed = min(df_segments['segmentSpeedKmH'])
102    avgSpeed = np.mean(df_segments['segmentSpeedKmH'])
103    maxSpeed = max(df_segments['segmentSpeedKmH'])
104    #=============================================#
105    #======== min,avg,max calculations for ACCURACY DROP
           ========#
106    accuracyDrops = list(df_segments['startAccuracyDrop'])
           + list(df_segments['endAccuracyDrop'])
107    minAccuracyDrop = min(accuracyDrops)
108    avgAccuracyDrop = np.mean(accuracyDrops)
109    maxAccuracyDrop = max(accuracyDrops)
110    #=============================================#
111    #============ writing calculated parameters in
           df_data ============#
```

```
112         cell_id = df_segments['cell_id_start'][0]
113         df_data['cell_id'] = [cell_id]
114 
115         num_segments = len(df_segments)
116         df_data['num_segments'] = [num_segments]
117         df_data['num_segments'] = num_segments
118 
119         df_data['minSpeed'] = round(minSpeed,2)
120         df_data['avgSpeed'] = round(avgSpeed,2)
121         df_data['maxSpeed'] = round(maxSpeed,2)
122 
123         df_data['minAccuracyDrop'] = minAccuracyDrop
124         df_data['avgAccuracyDrop'] = round(avgAccuracyDrop,2)
125         df_data['maxAccuracyDrop'] = maxAccuracyDrop
126 
127         return df_data
128 
129 df_grid_peak_1 = df_peak_1.groupby("cell_id_start" or "
        cell_id_end").apply(cell_data_calculator)
130 df_grid_peak_1.toPandas().to_csv('grids/grid_table_peak_1.
        csv',index = False)
131 
132 df_grid_peak_2 = df_peak_2.groupby("cell_id_start" or "
        cell_id_end").apply(cell_data_calculator)
133 df_grid_peak_2.toPandas().to_csv('grids/grid_table_peak_2.
        csv',index = False)
134 
135 df_grid_off_peak = df_off_peak.groupby("cell_id_start" or "
        cell_id_end").apply(cell_data_calculator)
136 df_grid_off_peak.toPandas().to_csv('grids/
        grid_table_off_peak.csv',index = False)
```

```
1 #Import Libraries
2 import geopandas as gpd
3 import pandas as pd
4 import numpy as np
5 import folium
6 from folium.features import GeoJsonTooltip
7 
8 #Read the geoJSON file using geopandas
9 geojson = gpd.read_file(r'../../100m/coords_TO.geojson')
```

```
10
11 #Read the grid tables
12 df_grid_peak_1=pd.read_csv(r'../grids/grid_table_peak_1.csv
       ')
13 df_grid_peak_2=pd.read_csv(r'../grids/grid_table_peak_2.csv
       ')
14 df_grid_off_peak=pd.read_csv(r'../grids/grid_table_off_peak
       .csv')
15
16 df_final_peak_1 = geojson.merge(df_grid_peak_1, left_on="
       polygon_id", right_on="cell_id", how="outer")
17 df_final_peak_1 = df_final_peak_1[~df_final_peak_1['
       geometry'].isna()]
18 df_final_peak_1.dropna(inplace = True)
19
20 df_final_peak_2 = geojson.merge(df_grid_peak_2, left_on="
       polygon_id", right_on="cell_id", how="outer")
21 df_final_peak_2 = df_final_peak_2[~df_final_peak_2['
       geometry'].isna()]
22 df_final_peak_2.dropna(inplace = True)
23
24 df_final_off_peak = geojson.merge(df_grid_off_peak ,
       left_on="polygon_id", right_on="cell_id", how="outer")
25 df_final_off_peak  = df_final_off_peak [~df_final_off_peak[
       'geometry'].isna()]
26 df_final_off_peak .dropna(inplace = True)
27
28 #management before plot
29 df_final_dict_peak_1 = df_final_peak_1.set_index('cell_id')
       ['avgSpeed']
30 df_final_dict_peak_2 = df_final_peak_2.set_index('cell_id')
       ['avgSpeed']
31 df_final_dict_off_peak = df_final_off_peak.set_index('
       cell_id')['avgSpeed']
32
33 df_final_dict_count_peak_1 = df_final_peak_1.set_index('
       cell_id')['num_segments']
34 df_final_dict_count_peak_2 = df_final_peak_2.set_index('
       cell_id')['num_segments']
35 df_final_dict_count_off_peak = df_final_off_peak.set_index(
       'cell_id')['num_segments']
```

```
36
37 #adding calculated parameters for plot
38 m = folium.Map(location=[45.0703,7.6869], zoom_start=12,
       tiles=None,overlay=False)
39 import branca
40 import branca.colormap as cm
41 colorScale_speed = cm.LinearColormap(['darkred','red','
       orange', 'yellow','lime','green'], vmin=0., vmax=130.,
       index=(df_final_peak_2['avgSpeed'].quantile
       ((0,0.2,0.4,0.6,0.8,1))).tolist())
42
43 colorScale_speed.caption = 'Average␣speed␣[KmH]'
44 colorScale_speed.add_to(m)
45
46 fg_off_peak  = folium.FeatureGroup(name='Speed[KmH]␣off␣
       peak␣hours␣1-2-3-4-5-6',overlay=False).add_to(m)
47 fg_peak_1 = folium.FeatureGroup(name='Speed[KmH]␣peak␣hours
       ␣9-10-11',overlay=False).add_to(m)
48 fg_peak_2 = folium.FeatureGroup(name='Speed[KmH]␣peak␣hours
       ␣18-19-20',overlay=False).add_to(m)
49
50 #off_peak
51 folium.features.GeoJson(
52                         data=df_final_off_peak,
53                         name='test␣for␣segment␣density␣',
54                         smooth_factor=2,
55                         style_function=lambda feature: {
56                             'fillColor': colorScale_speed(
                                  df_final_dict_off_peak[feature['
                                  properties']['polygon_id']]),
57                             'color' : 'black',
58                             'weight' : .2,
59                             'fillOpacity': .6
60                             },
61                         tooltip=folium.features.GeoJsonTooltip(
62                             fields=['cell_id',
63                                     'num_segments',
64                                     'minSpeed',
65                                     'avgSpeed',
66                                     'maxSpeed',
67                                     'minAccuracyDrop',
```

```
68                                      'avgAccuracyDrop',
69                                      'maxAccuracyDrop'
70                                    ],
71                                aliases=["Cell id:",
72                                      "Number of Segments:",
73                                      'Minimum Speed (Km/H)',
74                                      'Average Speed (Km/H)',
75                                      'Maximum Speed (Km/H)',
76                                      'Best GPS accuracy',
77                                      'Average GPS accuracy',
78                                      'Worst GPS accuracy'
79                                     ],
80                                localize=True,
81                                sticky=False,
82                                labels=True,
83                                style="""
84                                    background-color: #F0EFEF;
85                                    border: 2px solid black;
86                                    border-radius: 3px;
87                                    box-shadow: 3px;
88                                """,
89                                max_width=800,),
90                                    highlight_function=lambda x: {'
                                        weight':3,'fillOpacity': 2
                                        },
91                                ).add_to(fg_off_peak)
92  #peak_1
93  folium.features.GeoJson(
94                          data=df_final_peak_1,
95                          name='test for segment density ',
96                          smooth_factor=2,
97                          style_function=lambda feature: {
98                              'fillColor': colorScale_speed(
                                    df_final_dict_peak_1[feature['
                                    properties']['polygon_id']]),
99                              'color' : 'black',
100                             'weight' : .2,
101                             'fillOpacity': .6
102                             },
103                         tooltip=folium.features.GeoJsonTooltip(
104                             fields=['cell_id',
```

```
105                                         'num_segments',
106                                         'minSpeed',
107                                         'avgSpeed',
108                                         'maxSpeed',
109                                         'minAccuracyDrop',
110                                         'avgAccuracyDrop',
111                                         'maxAccuracyDrop'
112                                        ],
113                                 aliases=["Cell id:",
114                                          "Number of Segments:",
115                                          'Minimum Speed (Km/H)',
116                                          'Average Speed (Km/H)',
117                                          'Maximum Speed (Km/H)',
118                                          'Best GPS accuracy',
119                                          'Average GPS accuracy',
120                                          'Worst GPS accuracy'
121                                         ],
122                                 localize=True,
123                                 sticky=False,
124                                 labels=True,
125                                 style="""
126                                     background-color: #F0EFEF;
127                                     border: 2px solid black;
128                                     border-radius: 3px;
129                                     box-shadow: 3px;
130                                 """,
131                                 max_width=800,),
132                                     highlight_function=lambda x: {'
                                        weight':3,'fillOpacity': 2
                                        },
133                                 ).add_to(fg_peak_1)
134
135 #peak_2
136 folium.features.GeoJson(
137                         data=df_final_peak_2,
138                         name='test for segment density ',
139                         smooth_factor=2,
140                         style_function=lambda feature: {
141                             'fillColor': colorScale_speed(
                                df_final_dict_peak_2[feature['
                                properties']['polygon_id']]),
```

```
142                                 'color' : 'black',
143                                 'weight' : .2,
144                                 'fillOpacity': .6
145                                 },
146                             tooltip=folium.features.GeoJsonTooltip(
147                                 fields=['cell_id',
148                                         'num_segments',
149                                         'minSpeed',
150                                         'avgSpeed',
151                                         'maxSpeed',
152                                         'minAccuracyDrop',
153                                         'avgAccuracyDrop',
154                                         'maxAccuracyDrop'
155                                        ],
156                                 aliases=["Cell id:",
157                                          "Number of Segments:",
158                                          'Minimum Speed (Km/H)',
159                                          'Average Speed (Km/H)',
160                                          'Maximum Speed (Km/H)',
161                                          'Best GPS accuracy',
162                                          'Average GPS accuracy',
163                                          'Worst GPS accuracy'
164                                         ],
165                                 localize=True,
166                                 sticky=False,
167                                 labels=True,
168                                 style="""
169                                     background-color: #F0EFEF;
170                                     border: 2px solid black;
171                                     border-radius: 3px;
172                                     box-shadow: 3px;
173                                 """,
174                                 max_width=800,),
175                                     highlight_function=lambda x: {'
                                        weight':3,'fillOpacity': 2
                                        },
176                                 ).add_to(fg_peak_2)
177 folium.TileLayer('openstreetmap',overlay=True,name="
        OpenStreetMap").add_to(m)
178 folium.LayerControl(collapsed=False).add_to(m)
179 m.save("peaks_100m_speed.html")
```

Traffic congestion index

```
import geopandas as gpd
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

'''reading the grid tables for the peak and off peak hours '''
off_peak_table = pd.read_csv(r'grids/grid_table_off_peak.csv')
peak_table_1 = pd.read_csv(r'grids/grid_table_peak_1.csv')
peak_table_2 = pd.read_csv(r'grids/grid_table_peak_2.csv')
off_peak_cells = list(off_peak_table['cell_id'])
peak_1_cells = list(peak_table_1['cell_id'])
peak_2_cells = list(peak_table_2['cell_id'])

'''finding the intersection of grid cells'''
intersection = list(set(peak_1_cells) & set(peak_2_cells) & set(off_peak_cells))

'''dropping uncommon cells'''
df_peak_1 = peak_table_1.loc[peak_table_1['cell_id'].isin(intersection)]
df_peak_1.reset_index(inplace = True)

df_peak_2 = peak_table_2.loc[peak_table_2['cell_id'].isin(intersection)]
df_peak_2.reset_index(inplace = True)

df_off_peak = off_peak_table.loc[off_peak_table['cell_id'].isin(intersection)]
df_off_peak.reset_index(inplace = True)

#merging grid tables with useful columns
df_merged = pd.merge(pd.merge(df_off_peak[['cell_id','num_segments','avgSpeed']],
                      df_peak_1[['cell_id','num_segments','avgSpeed']],on='cell_id'),
                      df_peak_2[['cell_id','num_segments','avgSpeed']],on='cell_id')
```

```
32
33 df_merged.columns = ['cell_id', 'num_segments_offPeak', '
       avgSpeed_offPeak',
34                      'num_segments_peak1', 'avgSpeed_peak1'
                           ,
35                      'num_segments_peak2', 'avgSpeed_peak2'
                           ,
36                     ]
37
38 min_num_segments = list(zip(df_merged.num_segments_offPeak,
       df_merged.num_segments_peak1))
39 min_num_segments = [min(i[0],i[1]) for i in
       min_num_segments]
40 df_merged['min_num_segments_off_p1'] = min_num_segments
41
42 min_num_segments = list(zip(df_merged.num_segments_offPeak,
       df_merged.num_segments_peak2))
43 min_num_segments = [min(i[0],i[1]) for i in
       min_num_segments]
44 df_merged['min_num_segments_off_p2'] = min_num_segments
45
46 ''' Absolute traffic congestion '''
47 df_absCongestionKMH = df_merged.copy()
48 df_absCongestionKMH['speedDiffKMH_off_p1'] =
       df_absCongestionKMH['avgSpeed_offPeak'] -
       df_absCongestionKMH['avgSpeed_peak1']
49 df_absCongestionKMH['speedDiffKMH_off_p2'] =
       df_absCongestionKMH['avgSpeed_offPeak'] -
       df_absCongestionKMH['avgSpeed_peak2']
50
51 #CDF plot
52 speedDiffs = {'speedDiffKMH_off_p1' : df_absCongestionKMH['
       speedDiffKMH_off_p1'],
53       'speedDiffKMH_off_p2' : df_absCongestionKMH['
             speedDiffKMH_off_p2']}
54 # matplotlib.rcParams.update(matplotlib.rcParamsDefault)
55 plt.rcParams["figure.figsize"] = (10,5)
56 for key,value in speedDiffs.items():
57     x = list(value)
58     x.sort()
59     y = (1. * np.arange(len(x)) / (len(x) - 1))*100
```

```
60
61
62      plt.plot(x, y, zorder=2)
63  #       plt.hlines(y=(len(df_absCongestionKMH.loc[value <0 ])
        / len(df_absCongestionKMH))*100
64  #                   ,xmin = min(x), xmax = max(x),colors =
        colors[0], ls=':'
65
66  title = 'Cellwise Speed Difference Cumulative Distribution'
67  plt.title(title)
68  # title = 'Absolute Congestion'
69  plt.xlabel('Absolute Cellwise speed difference (KM/H)')
70  plt.ylabel('Percent %')
71  # plt.ylim(0,1)
72  plt.xlim(-125,125)
73  plt.grid(True)
74  plt.yticks(np.arange(0,110,10))
75  plt.xticks(np.arange(-130,131,20))
76  plt.legend(['Morning Peak VS off-Peak' , 'Evening Peak VS
        off-Peak'],loc ='lower right')
77
78  plt.axvline(x=0,ymin = 0, ymax = 1,color='black', linewidth
        = 1, zorder=1)
79  plt.axhline(y=len(df_absCongestionKMH.loc[
        df_absCongestionKMH['speedDiffKMH_off_p1'] <0 ]) / len(
        df_absCongestionKMH)*100
80              ,xmin = 0, xmax = 1,color = '#1f77b4', ls=':')
81  plt.axhline(y=(len(df_absCongestionKMH.loc[
        df_absCongestionKMH['speedDiffKMH_off_p2'] <0 ]) / len(
        df_absCongestionKMH))*100
82              ,xmin = 0, xmax = 1,color = '#ff7f0e', ls=':')
83  title='abs_congestion_p1_2'
84  # plt.savefig(f'{title}_CDF.jpg',bbox_inches='tight')
85  plt.show()
86
87  plt.rcParams["figure.figsize"] = (10,5)
88  # matplotlib.rcParams.update(matplotlib.rcParamsDefault)
89  x_p1 = df_absCongestionKMH['speedDiffKMH_off_p1']
90  x_p2 = df_absCongestionKMH['speedDiffKMH_off_p2']
91  bins = np.arange(-125,126,5)
92  plt.hist([x_p1,x_p2],bins=bins,edgecolor="lightblue")
```

```
93  plt.xticks(np.arange(-75,76,5))
94  plt.xlim(-70,70)
95  plt.yticks(np.arange(0,10001,1000))
96  # plt.xlim(-50,10)
97  title='Cellwise Speed Difference Distribution'
98  plt.title(title)
99  plt.xlabel('Cellwise Speed Difference (KM/H)')
100
101 plt.ylabel('Frequency')
102 plt.legend(['Morning Peak VS off-Peak' , 'Evening Peak VS
        off-Peak'],loc ='best')
103 plt.axvline(x=0,ymin = 0, ymax = 1,color='black', linewidth
        = 1.5)
104 plt.grid()
105 title = 'abs_congestion_p1_2'
106 # plt.savefig(f'{title}_Histogram.jpg',bbox_inches='tight')
107 plt.show()
108
109 ''' Relative traffic congestion '''
110 df_relCongestion = df_merged.copy()
111
112 #relative speed difference calculation
113 def relative_speed_difference(row,avgSpeed):
114     if row['avgSpeed_offPeak'] >= row[avgSpeed]:
115         speedDiff = (row['avgSpeed_offPeak'] - row[avgSpeed
            ])/row['avgSpeed_offPeak']
116     else:
117         speedDiff = -(row[avgSpeed] - row['avgSpeed_offPeak
            '])/row[avgSpeed]
118     return speedDiff*100
119
120 df_relCongestion['speedDiff%_p1'] = df_relCongestion.apply(
        lambda row: relative_speed_difference(row,'
        avgSpeed_peak1'), axis=1)
121 df_relCongestion['speedDiff%_p1'] = [float("{0:.2f}".format
        (i)) for i in df_relCongestion['speedDiff%_p1']]
122
123 df_relCongestion['speedDiff%_p2'] = df_relCongestion.apply(
        lambda row: relative_speed_difference(row,'
        avgSpeed_peak2'), axis=1)
```

```
124 df_relCongestion['speedDiff%_p2'] = [float("{0:.2f}".format
    (i)) for i in df_relCongestion['speedDiff%_p2']]
125
126 #CDF plot
127 speedDiffs = {'speedDiff%_p1' : df_relCongestion['speedDiff
    %_p1'],
128     'speedDiff%_p2' : df_relCongestion['speedDiff%_p2']}
129 plt.rcParams["figure.figsize"] = (10,5)
130 for key,value in speedDiffs.items():
131     x = list(value)
132     x.sort()
133     y = (1. * np.arange(len(x)) / (len(x) - 1))*100
134
135 #       matplotlib.rcParams.update(matplotlib.rcParamsDefault
    )
136     plt.plot(x, y)
137
138 title = 'Cellwise␣Speed␣Difference␣Cumulative␣Distribution'
139 plt.title(title)
140 title = 'Relative␣Congestion'
141 plt.xlabel('Relative␣Cellwise␣speed␣difference␣(%)')
142 plt.ylabel('Percent␣%')
143 # plt.ylim(0,1)
144 # plt.xlim(-1000,160)
145 plt.grid(True)
146 plt.yticks(np.arange(0,110,10))
147 plt.xticks(np.arange(-100,101,10))
148 # plt.hlines(y=negative_values_percentage,xmin = min(x),
    xmax = max(x),colors='red', ls=':')
149 plt.legend(['Morning␣Peak␣VS␣off-Peak' , 'Evening␣Peak␣VS␣
    off-Peak'],loc ='lower␣right')
150
151 plt.axvline(x=0,ymin = 0, ymax = 1,color='black', linewidth
    = 1)
152 plt.axhline(y=len(df_relCongestion.loc[df_relCongestion['
    speedDiff%_p1'] <0 ]) / len(df_relCongestion)*100
153             ,xmin = 0, xmax = 1,color = '#1f77b4', ls=':')
154 plt.axhline(y=(len(df_relCongestion.loc[df_relCongestion['
    speedDiff%_p2'] <0 ]) / len(df_absCongestionKMH))*100
155             ,xmin = 0, xmax = 1,color = '#ff7f0e', ls=':')
156 title='rel_congestion_p1_2'
```

```
157 plt.savefig(f'{title}_CDF.jpg',bbox_inches='tight')
158 plt.show()
159
160 plt.rcParams["figure.figsize"] = (10,5)
161 # matplotlib.rcParams.update(matplotlib.rcParamsDefault)
162
163 x_p1 = df_relCongestion['speedDiff%_p1']
164 x_p2 = df_relCongestion['speedDiff%_p2']
165 bins = np.arange(-100,100,10)
166
167 plt.hist([x_p1,x_p2],bins=bins,edgecolor="lightblue")
168 plt.xticks(np.arange(-100,101,10))
169 # plt.yticks(np.arange(0,3000,500))
170 # plt.xlim(-50,10)
171 title='Cellwise Speed Difference Distribution'
172 plt.title(title)
173 plt.xlabel('Relative Cellwise speed difference (%)')
174 title = 'Relative Congestion'
175 plt.ylabel('Frequency')
176
177 plt.grid()
178 plt.legend(['Morning Peak VS off-Peak' , 'Evening Peak VS
    off-Peak'],loc ='best')
179 plt.axvline(x=0,ymin = 0, ymax = 1,color='black', linewidth
    = 1.5)
180 title='rel_congestion_p1_2'
181 plt.savefig(f'{title}_Histogram.jpg',bbox_inches='tight')
182 plt.show()
183
184 ''' Cells behavior based on density investigation '''
185 #negative congestion extraction
186 df_rel_negative_conge = df_relCongestion[df_relCongestion['
    speedDiff%_p2'] < 0]
187
188 df_rel_positive_conge = df_relCongestion[df_relCongestion['
    speedDiff%_p2'] >= 0]
189
190 #CDF of negative congestion based on number of segments per
     cell [relative]
191 matplotlib.rcParams.update(matplotlib.rcParamsDefault)
```

```
192 dfs = [df_relCongestion,df_rel_positive_conge,
        df_rel_negative_conge]
193 for df in dfs:
194     x = list(df['min_num_segments_off_p2'])
195     x.sort()
196 # y = df_negative_conge.speedDiffKMH
197     y = (1. * np.arange(len(x)) / (len(x) - 1))*100
198 
199     plt.rcParams["figure.figsize"] = (10,5)
200     plt.plot( x,y)
201 plt.title('Cumulative Distribution of Segments per cell')
202 # title = 'Absolute Congestion'
203 plt.xlabel('Number of segments per cell')
204 plt.ylabel('Percent %')
205 # plt.ylim(0,1)
206 plt.xlim(-10,2000)
207 plt.grid(True)
208 plt.yticks(np.arange(0,110,10))
209 plt.xticks(np.arange(0,2001,100))
210 plt.legend(['Evening peak VS Off-peak','Evening peak VS Off
        -peak [Positively congested]','Evening peak VS Off-peak
        [Negatively congested]'],loc='lower right')
211 # plt.vlines(x=70,ymin = 0, ymax = max(y),colors='red', ls
        =':')
212 title='congestion_num_segments'
213 plt.savefig(f'{title}_CDF.jpg',bbox_inches='tight')
214 plt.show()
215 
216 bins = np.arange(0,max(x)+1,100)
217 plt.hist([list(df['min_num_segments_off_p2']) for df in dfs
        ],bins=bins,edgecolor="lightblue")
218 title='Distribution of Segments per cell'
219 plt.title(title)
220 plt.xlabel('Number of segments per cell')
221 plt.ylabel('Frequency')
222 plt.xticks(np.arange(0,2001,100))
223 plt.xlim(0,2000)
224 plt.grid()
225 plt.legend(['Evening peak VS Off-peak','Evening peak VS Off
        -peak [Positively congested]','Evening peak VS Off-peak
        [Negatively congested]'])
```

```
226 title='congestion_num_segments'
227 plt.savefig(f'{title}_Histogram.jpg',bbox_inches='tight')
228 plt.show()
229
230 #behavior comparison with and without filtering
231 df_relCongestion_above200 = df_relCongestion.loc[
      df_relCongestion['min_num_segments_off_p2']>200]
232 #CDF plot
233 speedDiffs = {'total' : df_relCongestion['speedDiff%_p2'],
234     'above_100' : df_relCongestion_above100['speedDiff%_p2'
          ],
235     'above_200' : df_relCongestion_above200['speedDiff%_p2'
          ]          }
236 plt.rcParams["figure.figsize"] = (10,5)
237 for key,value in speedDiffs.items():
238     x = list(value)
239     x.sort()
240     y = (1. * np.arange(len(x)) / (len(x) - 1))*100
241
242
243 title = 'Cellwise␣Speed␣Difference␣Cumulative␣Distribution'
244 plt.title(title)
245 title = 'Relative␣Congestion'
246 plt.xlabel('Relative␣Cellwise␣speed␣difference␣(%)')
247 plt.ylabel('Percent␣%')
248 plt.grid(True)
249 plt.yticks(np.arange(0,110,10))
250 plt.xticks(np.arange(-100,101,10))
251 plt.legend(['Evening␣Peak␣VS␣Off-peak␣[Not␣filtered]',
252             'Evening␣Peak␣VS␣Off-peak␣[cells␣containing␣
                  >100␣Segments]','Evening␣Peak␣VS␣Off-peak␣[
                  cells␣containing␣>200␣Segments]'])
253
254 plt.axvline(x=0,ymin = 0, ymax = 1,color='black', linewidth
      = 1)
255 title='rel_congestion_p2_above100'
256 plt.savefig(f'{title}_CDF.jpg',bbox_inches='tight')
257 plt.show()
```

Applying filters on relative congestion index

```
1 import geopandas as gpd
```

```
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import matplotlib
6
7  '''reading the grid tables for the peak and off peak hours
       '''
8  off_peak_table = pd.read_csv(r'grids/grid_table_off_peak.
       csv')
9  peak_table_1 = pd.read_csv(r'grids/grid_table_peak_1.csv')
10 peak_table_2 = pd.read_csv(r'grids/grid_table_peak_2.csv')
11 off_peak_cells = list(off_peak_table['cell_id'])
12 peak_1_cells = list(peak_table_1['cell_id'])
13 peak_2_cells = list(peak_table_2['cell_id'])
14
15 '''dropping uncommon cells '''
16 df_peak_1 = peak_table_1.loc[peak_table_1['cell_id'].isin(
       intersection)]
17 df_peak_1.reset_index(inplace = True)
18
19 df_peak_2 = peak_table_2.loc[peak_table_2['cell_id'].isin(
       intersection)]
20 df_peak_2.reset_index(inplace = True)
21
22 df_off_peak = off_peak_table.loc[off_peak_table['cell_id'].
       isin(intersection)]
23 df_off_peak.reset_index(inplace = True)
24
25 '''merging related columns of off-peak and peak grid tables
       '''
26 df_merged = pd.merge(df_off_peak[['cell_id','num_segments',
       'avgSpeed']],
27                                 df_peak_2[['cell_id','
                                       num_segments','avgSpeed'
                                       ]],on='cell_id')
28
29 #minimum number of segment in cellwise comparison
30 min_num_segments = list(zip(df_merged.num_segments_x,
       df_merged.num_segments_y))
31 min_num_segments = [min(i[0],i[1]) for i in
       min_num_segments]
```

```
32 df_merged['min_num_segments'] = min_num_segments
33
34 def relative_speed_difference(row):
35     if row['avgSpeed_x'] >= row['avgSpeed_y']:
36         speedDiff = (row['avgSpeed_x'] - row['avgSpeed_y'])
              /row['avgSpeed_x']
37     else:
38         speedDiff = -(row['avgSpeed_y'] - row['avgSpeed_x'
              ])/row['avgSpeed_y']
39     return speedDiff*100
40
41 df_relCongestion = df_merged.copy()
42 df_relCongestion['speedDiff%'] = df_relCongestion.apply(
     lambda row: relative_speed_difference(row), axis=1)
43 df_relCongestion['speedDiff%'] = [float("{0:.2f}".format(i)
     ) for i in df_relCongestion['speedDiff%']]
44
45 '''Applying filters'''
46 df_relCongestion = df_relCongestion.loc[df_relCongestion['
     min_num_segments']>100]
47
48 # dropping negative congested cells
49 df_relCongestion = df_relCongestion[df_relCongestion['
     speedDiff%'] >= 0]
50
51 #CDF plot
52 matplotlib.rcParams.update(matplotlib.rcParamsDefault)
53 plt.rcParams["figure.figsize"] = (10,5)
54 x = list(df_relCongestion['speedDiff%'])
55 x.sort()
56 y = (1. * np.arange(len(x)) / (len(x) - 1))*100
57 plt.plot(x, y)
58 title = 'Cellwise␣Speed␣Difference␣Cumulative␣Distribution␣
     [Positive␣side]'
59 plt.title(title)
60 title = 'Relative␣Congestion'
61 plt.xlabel('Relative␣Cellwise␣speed␣difference␣(%)')
62 plt.ylabel('Percent␣%')
63 plt.grid(True)
64 plt.yticks(np.arange(0,110,10))
65 plt.xticks(np.arange(0,101,5))
```

```
66
67 plt.savefig(f'{title}_CDF.jpg',bbox_inches='tight')
68 plt.show()
69
70
71 plt.rcParams["figure.figsize"] = (10,5)
72 bins = np.arange(0,100,5)
73 plt.hist(x,bins=bins,edgecolor="lightblue")
74 plt.xticks(np.arange(0,100,5))
75 title='Cellwise Speed Difference Distribution [Positive
       side]'
76 plt.title(title)
77 plt.xlabel('Relative Cellwise Speed Difference (%)')
78
79 plt.ylabel('Frequency')
80 plt.grid()
81 title = 'rel_pos_congestion'
82 plt.savefig(f'{title}_Histogram.jpg',bbox_inches='tight')
83 plt.show()
84
85 '''creating grid table of positively congested areas'''
86 positive_cells = list(df_relCongestion['cell_id'])
87
88 off_peak_grid = df_off_peak.loc[df_off_peak['cell_id'].isin
       (positive_cells)]
89 off_peak_grid = pd.merge(off_peak_grid[['cell_id', '
       num_segments', 'minSpeed', 'avgSpeed', 'maxSpeed',
90                                             'minAccuracyDrop', '
                                                  avgAccuracyDrop',
                                                   'maxAccuracyDrop
                                                  ']],
91                                     df_relCongestion[['cell_id',
                                        'speedDiff%','avgSpeed_y'
                                        ]],on='cell_id')
92 off_peak_grid.to_csv(r'4_2-grid-congestion-peaks/
       off_peak_above_100.csv',index= False)
93
94 peak_1_grid = df_peak_1.loc[df_peak_1['cell_id'].isin(
       positive_cells)]
95 peak_1_grid = pd.merge(peak_1_grid[['cell_id', '
       num_segments', 'minSpeed', 'avgSpeed', 'maxSpeed',
```

```
96                                                  'minAccuracyDrop', '
                                                        avgAccuracyDrop',
                                                         'maxAccuracyDrop
                                                        ']],
97                                              df_relCongestion[['cell_id',
                                                    'speedDiff%']],on='
                                                    cell_id')
98  peak_1_grid.to_csv(r'4_2-grid-congestion-peaks/
        peak_2_above_100.csv',index= False)
```

Congestion plot on the map

```
1  #Import Libraries
2  import geopandas as gpd
3  import pandas as pd
4  import numpy as np
5  import folium
6  from folium.features import GeoJsonTooltip
7
8  #Read the geoJSON file using geopandas
9  geojson = gpd.read_file(r'../../100m/coords_TO.geojson')
10
11 df_grid_peak_2=pd.read_csv(r'peak_2_above_100.csv')
12 df_grid_off_peak=pd.read_csv(r'off_peak_above_100.csv')
13
14 df_final_off_peak = geojson.merge(df_grid_off_peak ,
       left_on="polygon_id", right_on="cell_id", how="outer")
15 df_final_off_peak  = df_final_off_peak [~df_final_off_peak[
       'geometry'].isna()]
16 df_final_off_peak .dropna(inplace = True)
17
18 df_final_dict_rel_congestion = df_final_off_peak.set_index(
       'cell_id')['speedDiff%']
19
20 m2 = folium.Map(location=[45.0703,7.6869], zoom_start=12,
       tiles=None,overlay=False)
21
22 colorScale_relative_congestion = cm.LinearColormap(['
       aquamarine','palegreen','greenyellow','yellow','
       orangered','darkred'], vmin=0., vmax=100.,
```

```
23                                          index=(
                                                df_final_off_peak['
                                                speedDiff%'].
                                                quantile
                                                ((0,0.2,0.4,0.6,0.8,1)
                                                )).tolist())
24
25 colorScale_relative_congestion.caption = 'Relative␣
       congestion␣(%)'
26 colorScale_relative_congestion.add_to(m2)
27
28 fg_relative_congestion  = folium.FeatureGroup(name='Evening
       ␣Peak␣VS␣Off-peak␣Relative␣Congestion␣(%)',overlay=False
       ).add_to(m2)
29
30 #relative_congestion
31 folium.features.GeoJson(
32                         data=df_final_off_peak,
33                         name='test␣for␣segment␣density␣',
34                         smooth_factor=2,
35                         style_function=lambda feature: {
36                             'fillColor':
                                    colorScale_relative_congestion(
                                    df_final_dict_rel_congestion[
                                    feature['properties']['cell_id'
                                    ]]),
37                             'color' : 'black',
38                             'weight' : .2,
39                             'fillOpacity': .6
40                             },
41                         tooltip=folium.features.GeoJsonTooltip(
42                             fields=['cell_id',
43                                     'speedDiff%',
44                                     'avgSpeed',
45                                     'avgSpeed_y',
46                                     'num_segments',
47                                    ],
48                             aliases=["Cell␣id:",
49                                      'Relative␣Speed␣Difference
                                            ␣(%):',
50                                      'Off-peak␣Speed␣(KM/H):',
```

```
                                        'Evening peak Speed (KM/H)
                                            :',
                                        "Number of Segments:",
                                       ],
                                localize=True,
                                sticky=False,
                                labels=True,
                                style="""
                                    background-color: #F0EFEF;
                                    border: 2px solid black;
                                    border-radius: 3px;
                                    box-shadow: 3px;
                                """,
                                max_width=800,),
                                    highlight_function=lambda x: {'
                                        weight':3,'fillOpacity': 2
                                        },
                                ).add_to(fg_relative_congestion)

folium.TileLayer('openstreetmap',overlay=True,name="
    OpenStreetMap").add_to(m2)
folium.LayerControl(collapsed=False).add_to(m2)

m2.save("peaks_100m_relative_congestion_filt100.html")
```

KDE implementation

```
#Import Libraries
import folium
from folium.features import GeoJsonTooltip
import pandas as pd
import numpy as np
from sklearn.neighbors import KernelDensity
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import branca.colormap as cm
import geojsoncontour

#Read the geoJSON file using geopandas
geojson = gpd.read_file(r'centroids_wgs.geojson')
df_grid=pd.read_csv(r'../4_2-grid-congestion-peaks/
    off_peak_above_100.csv')
```

```
15
16 df_merge = geojson.merge(df_grid, left_on="polygon_id",
       right_on="cell_id", how="outer")
17 df_merge = df_merge[~df_merge['geometry'].isna()]
18 df_merge.dropna(inplace = True)
19
20 #convert float percentages to the closest integer
21 df_merge['speedDiff%'] = [round(i) for i in df_merge['
       speedDiff%']]
22
23 #relative congestion as density
24 df_multiplies = pd.DataFrame()
25 for index,row in df_merge.iterrows():
26     multiplier = row['speedDiff%']
27     df_multiplies = df_multiplies.append([row]*multiplier,
          ignore_index=True)
28
29 #Grid
30 #%% min and max for coordinates
31 minLat = 44.96282106687191
32 minLon = 7.502048016422193
33 maxLat = 45.19265016665321
34 maxLon = 7.791812422724604
35
36 tmp= list(df_merge['cell_id'].str.split('_'))
37 x_num = len(np.unique(np.array(tmp)[:,0].astype(int)))
38 y_num = len(np.unique(np.array(tmp)[:,1].astype(int)))
39
40 #building a grid of point for sampling the resulting KDEs
41 xx = np.linspace(minLon, maxLon, x_num)
42 yy = np.linspace(minLat, maxLat, y_num)
43
44 xg, yg = np.meshgrid(xx,yy)
45 grid_coords = np.c_[xg.ravel(), yg.ravel()]
46
47 # KDE
48 colors = ['navy','blue','royalblue','cyan','yellow','orange
       ','orangered','red','darkred']
49
50 kde_cols = ['centroid_lon','centroid_lat']
51 bandwidths = [.0005,.001,.00150,.002,.0025,.003,.0035]
```

```
52
53 #train model
54 maps = []
55 for i,bandwidth in enumerate(bandwidths):
56     m = folium.Map(location=[45.0703,7.6869], zoom_start
       =12,overlay=False,tiles='openstreetmap')
57     model = KernelDensity( metric="euclidean",kernel="
       gaussian",bandwidth=bandwidth).fit(df_multiplies[
       kde_cols])
58     kde_samples = np.exp(model.score_samples(grid_coords).
       reshape(*xg.shape))
59     levels = np.linspace(kde_samples.min(), kde_samples.max
       (), 10)
60     colorScale = cm.LinearColormap(colors, vmin=round(
       kde_samples.min(),2),
61                                    vmax=round(kde_samples.
                                       max(),2),caption = '
                                       Score Sample').
                                       to_step(10)
62     colorScale.caption = f'Score Sample of bandwidth = {
       bandwidth}'
63     colorScale.add_to(m)
64     contourf = plt.contourf(xg, yg, kde_samples,levels =
       levels, alpha=0.6,colors = colorScale.colors)
65
66     # Convert matplotlib contourf to geojson
67     geojson = geojsoncontour.contourf_to_geojson(
68         contourf=contourf,
69         min_angle_deg=3.0,
70         ndigits=5,
71         stroke_width=.5,
72         fill_opacity=0.5)
73
74     folium_geo = folium.GeoJson(
75         geojson,
76         style_function=lambda x: {
77             'color':     x['properties']['stroke'],
78             'weight':    x['properties']['stroke-width'],
79             'fillColor': x['properties']['fill'],
80             'opacity':   0.8,
81         }).add_to(m)
```

```
    maps.append(m)
for i,m in enumerate(maps):
    m.save(f"bandwidth_{bandwidths[i]}.html")
```

Filtering the KDE result with bandwidth = 0.001

```
#Import Libraries
import folium
from folium.features import GeoJsonTooltip
import pandas as pd
import numpy as np
from sklearn.neighbors import KernelDensity
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import branca.colormap as cm
import geojsoncontour

#Read the geoJSON file using geopandas
geojson = gpd.read_file(r'drafts/centroids_wgs.geojson')
df_grid=pd.read_csv(r'../4_2-grid-congestion-peaks/
    off_peak_above_100.csv')

df_merge = geojson.merge(df_grid, left_on="polygon_id",
    right_on="cell_id", how="outer")
df_merge = df_merge[~df_merge['geometry'].isna()]
df_merge.dropna(inplace = True)

#convert float percentages to the closest integer
df_merge['speedDiff%'] = [round(i) for i in df_merge['
    speedDiff%']]

#relative congestion as density
df_multiplies = pd.DataFrame()
for index,row in df_merge.iterrows():
    multiplier = row['speedDiff%']
    df_multiplies = df_multiplies.append([row]*multiplier,
        ignore_index=True)

#Grid
#%% min and max for coordinates
minLat = 44.96282106687191
minLon = 7.502048016422193
```

```
33 maxLat = 45.19265016665321
34 maxLon = 7.791812422724604
35
36 tmp= list(df_merge['cell_id'].str.split('_'))
37 x_num = len(np.unique(np.array(tmp)[:,0].astype(int)))
38 y_num = len(np.unique(np.array(tmp)[:,1].astype(int)))
39
40 #building a grid of point for sampling the resulting KDEs
41 xx = np.linspace(minLon, maxLon, x_num)
42 yy = np.linspace(minLat, maxLat, y_num)
43
44 xg, yg = np.meshgrid(xx,yy)
45 grid_coords = np.c_[xg.ravel(), yg.ravel()]
46
47 colors = ['black','red']
48
49 #KDE
50 kde_cols = ['centroid_lon','centroid_lat']
51 bandwidth = .001 #approx 100m
52
53 m = folium.Map(location=[45.0703,7.6869], zoom_start=12,
       overlay=False,tiles='openstreetmap')
54
55 threshold = 74
56 kde_samples[kde_samples<threshold] = 0
57 levels = np.array(0)
58 levels = np.append(levels,np.linspace(threshold,
       kde_samples.max(), 2))
59 levels = [round(l,2) for l in levels]
60 colorScale = cm.LinearColormap(colors, vmin=round(
       kde_samples.min(),2), vmax=round(kde_samples.max(),2)).
       to_step(index=levels)
61
62 colorScale.caption = f'Score Sample of bandwidth = {
       bandwidth}'
63 colorScale.add_to(m)
64 contourf = plt.contourf(xg, yg, kde_samples,levels = levels
       , alpha=0.9,colors = colorScale.colors)
65
66 # Convert matplotlib contourf to geojson
67 geojson = geojsoncontour.contourf_to_geojson(
```

```
68      contourf=contourf,
69      min_angle_deg=3.0,
70      ndigits=5,
71      stroke_width=.5,
72      fill_opacity=1)
73
74  folium_geo = folium.GeoJson(
75      geojson,
76      style_function=lambda x: {
77          'color':      x['properties']['stroke'],
78          'weight':     x['properties']['stroke-width'],
79          'fillColor': x['properties']['fill'],
80          'opacity':   1,
81      }).add_to(m)
82
83  m.save(f"congested areas.html")
```