Universitetet i Sørøst-Norge

Processing and visualizing traffic data and affecting factors from the Norwegian Public Roads Administration

CS4010: Data Management Course

8556

8559

Department of Science and Industry Systems

Faculty of Technology, Natural Sciences and Maritime Sciences

University South-Eastern Norway, 2023

Abstract

Traffic generates a lot of data. There is both static data, such as speed limits, and dynamic data including traffic volume and the speed driven. A lot of this data in Norway is provided by the Norwegian Public Roads Administration.

In this report we present the results and conclusions from our research related to processing and visualizing traffic data from the Norwegian Public Roads Administration.

Keywords: Big Data, Traffic, Visualization, MongoDB, Grafana, Python

Preface

The group is taking the Data Management course (CS4010) while attending the University of South-Eastern Norway in Kongsberg. For our project we have chosen to extract data from the Norwegian Public Roads Administration.

We began exploring data provided by the NPRA primarily because of the abundance of publically available data. After looking a while, we found Traffic Registration Points that monitor road traffic. These TRP's are spread around Norway collecting data from traffic, most of them 24/7, 365 days a year.

Initially, creating a small data set for a "Proof of Concept" posed challenges in data integration. We decided to focus on one specific road in Oslo, RV162, or commonly referred to as "Ring 1". The objective was to streamline the process to further add roads later on.

Some of the data was available to download straight to CSV or EXCEL files. While specifically one source needed to be requested through a GraphQL endpoint. To request, transform, normalize and to load the data we used Python alongside various libraries. For storing the data we used MongoDB as it is a document based NoSQL database suited for big data workloads. To visualize the data we opted to use Grafana as its interface is both user friendly and numerous useful features.

The group learned new skills and utilized new technologies to achieve the result. We are quite pleased with the result. While it might not be deployed by the industry or the NPRA, it gives the group members valuable experience with both industry-standard visualization, data collection, transformation and storage using big data technologies.

List of abbrevations

ACID - Atomicity, Consitency, Isolation and Durability

CAP - Consistency, Availability and Partition tolerance

NPRA - Norwegian Public Roads Administration

TRP - Traffic Registration Point

UTC - Coordinated Universal Time

GDPR - General Data Protection Regulation

BI - Business Intelligence

API - Application Programming Interface

List of Figures

Contents

Chapter 1

Introduction

1.1 Problem Statement

In this project we have been working on gathering, transforming, storing and visualizing traffic data from the Norwegian Public Roads Administration, with the goal of seeing how speed, traffic volume and weather conditions affect each other and accidents.

1.2 Objective

Our main goal was to visualize the collected data in a Grafana dashboard that was connected to a MongoDB database. We wanted to visualize deviations from the speed limit, traffic volume, and accidents over time. If we completed our main goal, the next goal was to collect weather data to visualize to look for correlations. Our last interim goal was to add more roads and expand our dataset beyond RV162.

1.3 Assumption and limitations

We assumed that we would be able to complete the project on time and create the visualisation we wanted. This led us to first look at the datasets from the Norwegian Public Roads Administration and extend the datasets with weather data from the Norwegian Meteorological Institute if we had time.
As for the data from the Norwegian Public Roads Administration, some data is not publicly available by default for privacy reasons. This required us to request access to data

about actual driving speeds, which we were granted. This data lacked records where fewer than five cars passed the registration point during that hour, due to GDPR regulations. It is also worth mentioning that a lot of this data was recorded during Covid-19, which may have resulted in different driving pattern than pre- or post-Covid.

1.4 Project management

During the project, both members did collaborate remotely, and not always at the same time. There were frequent meetings but in order to ensure a smooth workflow, Git and GitHub was used as version control. For collaborative work, the Docker containers and code ran on a server hosted by one of the group members, accessible through a VPN access. For virtual meetings the group used Discord, due to it's ease of use, screen sharing and support for integrations such as webhooks. A webhook was created to notify the group when commits were uploaded and when pull requests were created. Each pull request needed to be reviewed by at least one other team member to ensure quality and to keep each team member updated. For collaboration on the project report, the group used Overleaf to collaborate on the LaTeX code. For additional file sharing a shared Google Drive folder was used.

Chapter 2

Theoretical Background

2.1 Spark

Spark is an open-source engine for big data processing. It uses a cluster architecture in a master and slave configuration. It is built for data parallelism and fault tolerance. Spark processes data in-memory, in contrast to Hadoop MapReduce [1]. Spark abstracts the complexity of distributed systems by providing frameworks for machine learning (MLib), structured data processing (Spark SQL), and streaming (Spark Streaming). These frameworks can be accessed by common programming languages like Java, Python, Scala and R[2].

2.2 NoSQL

NoSQL, also refered to as "not only SQL" and is a non-relational database that is used to store data in a non-relational form. There are different types of NoSQL databases but they all have in common that they differ from traditional relational databases, as do not fulfill the ACID properties of traditional databases. They compromise on consistency, usually providing a form of "eventual consistency" and rather favor availability and partition tolerance [3][1]. There are four main types of NoSQL databases [3]:

- Document-based, like MongoDB [4] which stores documents (equivalent to rows) in a collection (equivalent to tables) or Apache Couchbase [5].
- Key-value stores like Redis [6], which store data as a key-value pair.

- Column-oriented like CassandraDB [7] and Hbase [8], that store data per column and not row.
- Graph-based databases like ArrangoDB [9] or Neo4j [10] that store data in graphs. This is popular with social media platforms, who use it to find "friends of friends".

Chapter 3

Literature Review

In our search for literature or projects that do exactly the same as us, we have found very little. A somewhat comparable project is found in the paper "A timeline visualization system for road traffic big data" by A. Imawan and J. Kwon [11]. Here they look at how a timeline visualization of road traffic data can improve the ability to see trends over time compared to the many solutions today that only provide current information about traffic such as google traffic [11].

Another project that is also somewhat relevant is found in the paper "Analysis of the New York City's Vehicle Crash Open Data" by A. Saxena and S. A. Robila [12]. This paper presents the development of a New York City (NYC) vehicle crash analysis tool. It enables direct data visualization, trend analysis, and examination of influencing factors [12].

Our project differs from the two mentioned above in terms of using an existing open source interactive visualization tool rather than developing our own or have code to plot static figures. The amount of data used is also smaller, but it combines both traffic volume, speed and accidents.

Chapter 4

Research Methodology

4.1 Choice of technologies

When we started thinking about tools to use for this project we wanted to use tools that were:

- Appropriate for Big Data
- Free to use
- Well documented and maintained

The requirement for big data tools arose from the requirement to ensure our solution scaled. Even though the amount of data in this project did not need the use of big data tools, an expansion to more roads and a larger time span would require such solutions without changing technology stack.

We needed to collect data from API's to store and process it a later stage. This "scraper" could have been written in any language, however the API in question was a GraphQL API. GraphQL is a graph query language for API's, and since Python has rich libraries for data processing, it is well documented and maintained and both members have experience with it, it was a natural choice.

To make sure we used big data tools, we decided to use Spark. There is a Python library called PySpark that allows us to run our Python code to a Spark cluster [13]. This allows us to process the data across a cluster of nodes. Since the group did not want to use another language PySpark was the natural choice.

For data storage the group settled on using MongoDB. This is because it meets the

aforementioned requirements and the group has prior knowledge with it. Furthermore MongoDB offers the flexibility to change the schema per document, this was important because we did not know how the schemas were going to look. Alternatively, column based databases could have been used. For example: CassandaDB[7] or Hbase[8], however these would have limited our flexibility.

Finally to visualize the data we went with Grafana, specifically the open-source version. Grafana has a free, open-source community driven MongoDB plugin. Grafana does also provide an in-house developed MongoDB plugin, however it is an enterprise edition plugin only accessible through an enterprise licence or Grafana cloud account. As a result, we opted for the community edition plugin with limited functionality [14]. Alternatives to Grafana could have been Microsoft PowerBI [15] or Tableau [16], but they are both paid software. The Grafana interface is also easy and intuitive for users that are going to use and observe the data. It also user-friendly for the developers designing the dashboard. However, with the way we have designed our storage it is easy to integrate with any other general BI tool that have MongoDB support.

4.2 Data collection process

The Norwegian Public Roads Administration provided road traffic volume data in a GraphQL API. The API only allowed us to query for four days of data at a time, and thus we created a Python script that repeated the query until we had data for three years. The script had to wait between each query to ensure the connection didn't time out or get blocked by the NPRA [17]. The data was then saved to a CSV file. The speed limits and accidents data was downloaded from the National Road Data bank [18]. We chose the type of data, the roads we wanted data from and downloaded as a CSV. As for the speed data we had to submit a request, as it was not available for the public. We then got an Excel workbook for each of the traffic registration points on our selected road.

4.3 Data preprocessing

After the data was collected we used various scripts to preprocess our data. This included turning Excel workbooks to CSV files, dropping unnecessary columns, renaming columns, transforming coordinates from EPSG 5973 [19] to EPSG4326 [20] in decimal degree nota-

tion [21]. The group tried to use Spark as much as possible for the preproccesing as well, but were not able to do this for every dataset due to issues with user-defined functions in Pyspark [22]. This led to usage of Pandas and local preproccesing in addition to Pyspark where Pyspark alone did not suffice.

4.4 Data transformation

All the data transformations were done using Spark through the Pyspark library. This included converting date-time to timestamps in UTC, renaming and selection of columns. For traffic volume and speed we did the same time conversions, renamed and selected columns and joined the dataframes together based on the timestamp. For the metadata we first joined the speed and traffic volume dataframes, then renamed and dropped columns, joined the speed limit dataframe based on the 'from' and 'to' meter values of the stretch of road to obtain the speed limit for the location of the TRP.

4.5 Data storage

All our finished dataframes were stored in a MongoDB collection using the MongoDB Spark Python connector [23]. Our data was stored in three separate collections, accidents, traffic_data and traffic_registration_point. With accidents and traffic_data being timeseries collections. The collection schemas can be found in appendix A.

4.6 Data visualization

The visualization was done through a Grafana dashboard. We made the dashboard by creating different panels, in these panels we wrote MongoDB aggregated queries that were sent directly to the database. All of the panels, excluding the map over accidents, also includes variables that link to the timeframe that can be adjusted in Grafana. The graphs and gauges in the dashboard are linked directly to a TRP through a Grafana variable in the query. This makes it so that each of the panels affected by the variable shows information related to that specific TRP.

4.7 System architecture

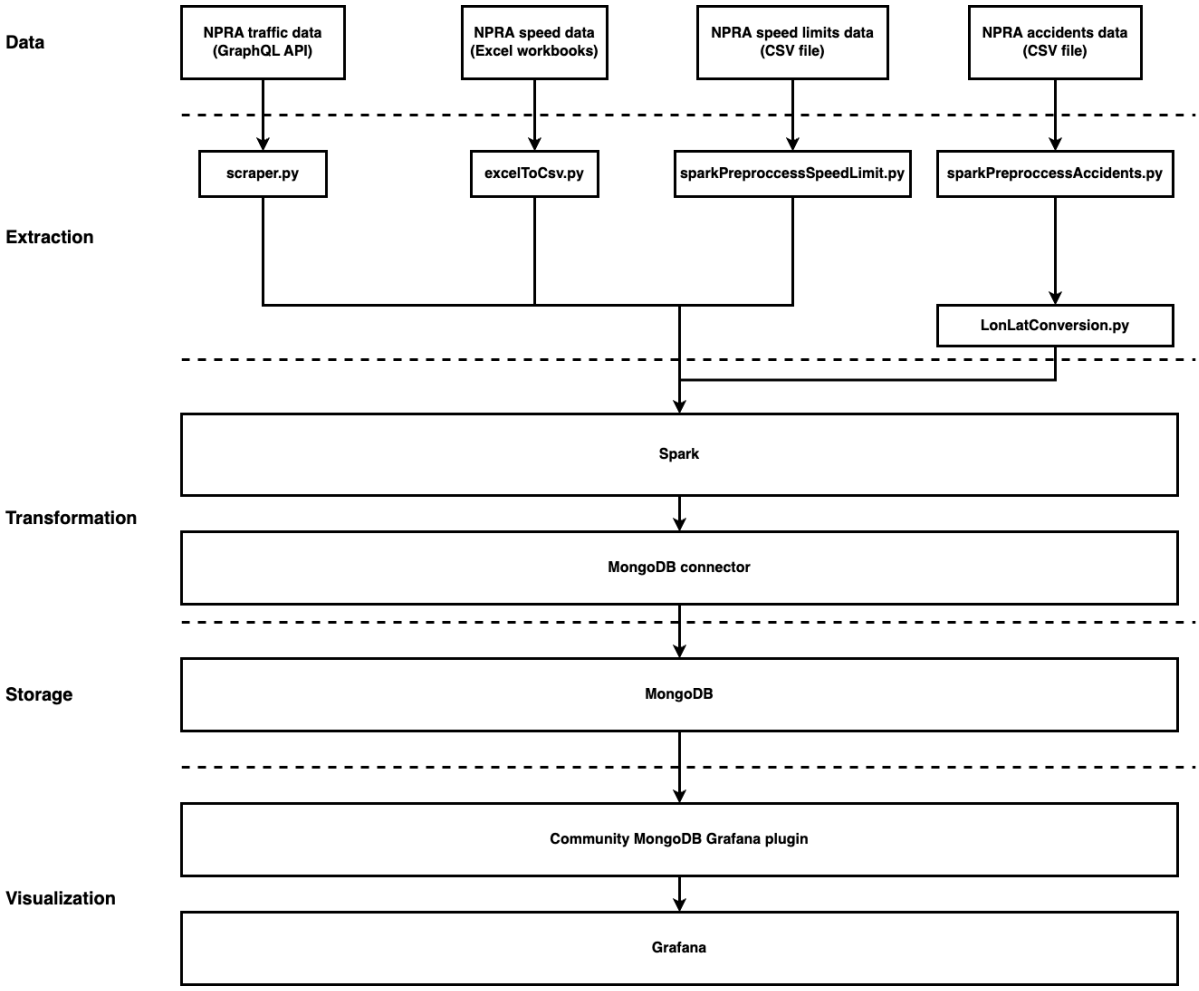

Figure 4.1: Overview of system architecture

Chapter 5

Results

We have managed to extract, transform, store and visualize data of different formats coming from the NPRA. In total there is three years of data from five traffic registration points. In total, we ended up with about 116,000 MongoDB documents in our traffic_data collection. Additionally, we have an accidents collection and a traffic_registration_point collection for metadata. They follow the schema provided in appendix A. We were able to use this data to create a Grafana dashboard to visualize our data.

From figure B.1 and B.2 in appendix B we can see different visualizations of our data. The accident data had relatively few data points and is therefore plotted statically on a map and not displayed based on the selected time interval. We can hover over the accidents on the map to see details about the accident.

All the other visualizations query data exclusively from the selected time interval in Grafana. This let us see how things evolve over time, and how things evolve during the day. From this we can see how the deviation from the speed limit is often negative during rush hour and how people usually drive a little over the speed limit on average. The 85-quantile drives a substantial amount above the speed limit. Because our dataset predates Covid-19, there is indications that the traffic dropped substantially during march of 2020 (first lockdown), and it varied during 2020 and 2021. During the start of 2022 it increased to a volume close to the start of 2020, and now in 2023 it is back to normal. To further analyze how the pandemic affected traffic, we need a bigger dataset. There is also a heat map over the traffic volume of the different TRP's, and for all the gauges and timeseries visualizations we can use a variable to pick the TRP we want to see data for.

Chapter 6

Discussion and Conclusions

The reason for going with data from the Norwegian Public Roads Administration is because they possess a lot of data, although not all of it is interconnected. Initially, we wanted to look at traffic volume, speed, speed-limit deviation, and accidents and try to see how they affected each other. If we had time, we wanted to include weather data as well to assess its impact.

The project had a duration of three weeks in total. The first week was spent on setting up the environment. The second week was used to perform the implementation, and the third week was spent writing the report. During the three weeks, both group members had contributed equally. We each collected and prepared some datasets, worked on the transformation using Spark together, and both created different visualizations for the Grafana dashboard.

During the project, we encountered several difficulties. The first and biggest was with the Spark cluster. We got it up and running and could submit jobs to it. But when we had issues with getting our local files to be processed on the cluster. We tried to manually place it on the server and through spark-submit options, but the PySpark script never found our files. This led us to run Spark locally instead of in cluster mode.

Another recurring difficulty was the MongoDB connector to the PySpark that finally worked once we used a specific version. Any newer version and it broke. This was also the issue with Grafana and the community plugin we had to use. Another challenge with the Grafana MongoDB plugin was that it only supported aggregated queries. It lacked support for helper functions, which led to some extra effort in crafting queries that worked and made use of variables from the Grafana dashboard. The difficulties we experienced

prevented us from having the time to gather additional weather data as we had originally hoped to.

To conclude, we are quite satisfied with our end result. However, we recognize that with more time, we could have added weather data, more traffic and speed data (several extra years) and more road sections. It is also possible that we had gotten our jobs to process on the Spark cluster. With more time we could also constructed a pipeline for automatic data gathering, pre-processing, transforming and storage. This could have expanded the project indefinitely, with the exception of speed data, which required approval by the NPRA.

References

[1] M. Guller, *Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large Scale Data Analysis* (Books for professionals by professionals), eng. Berkeley, CA: Apress L. P, 2015, ISBN: 9781484209653.

[2] "Spark documentation - overview." (n/a), [Online]. Available: `https://spark.apache.org/docs/latest/index.html#spark-overview` (visited on 09/13/2023).

[3] M. Housley and J. Reis, *Fundamentals of Data Engineering*, eng. O'Reilly Media, Inc, 2022, ISBN: 1098108299.

[4] "Mongodb." (n/a), [Online]. Available: `https://www.mongodb.com` (visited on 09/13/2023).

[5] "Couchbase." (n/a), [Online]. Available: `https://www.couchbase.com/` (visited on 09/13/2023).

[6] "Redis." (n/a), [Online]. Available: `https://redis.io/` (visited on 09/13/2023).

[7] "Cassandra." (n/a), [Online]. Available: `https://cassandra.apache.org` (visited on 09/13/2023).

[8] "Hbase." (n/a), [Online]. Available: `https://hbase.apache.org/` (visited on 09/13/2023).

[9] "Arrangodb." (n/a), [Online]. Available: `https://www.arangodb.com/` (visited on 09/13/2023).

[10] "Neo4j." (n/a), [Online]. Available: `https://neo4j.com/` (visited on 09/13/2023).

[11] A. Imawan and J. Kwon, "A timeline visualization system for road traffic big data," in *2015 IEEE International Conference on Big Data (Big Data)*, 2015, pp. 2928–2929. DOI: 10.1109/BigData.2015.7364125.

[12] A. Saxena and S. A. Robila, "Analysis of the new york city's vehicle crash open data," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 6017–6019. DOI: 10.1109/BigData52589.2021.9672012.

[13] "Pyspark 3.5.0 documentation." (n/a), [Online]. Available: https://spark.apache.org/docs/latest/api/python/index.html (visited on 09/14/2023).

[14] "Grafana mongodb community plugin." (2023), [Online]. Available: https://github.com/meln5674/grafana-mongodb-community-plugin (visited on 09/14/2023).

[15] "Microsoft powerbi." (n/a), [Online]. Available: https://powerbi.microsoft.com/en-us (visited on 09/14/2023).

[16] "Tableau." (n/a), [Online]. Available: https://www.tableau.com (visited on 09/14/2023).

[17] "Npra traffic data api." (n/a), [Online]. Available: https://www.vegvesen.no/trafikkdata/api/ (visited on 09/14/2023).

[18] "Datakatalog statens vegvesen." (n/a), [Online]. Available: https://datakatalogen.atlas.vegvesen.no/ (visited on 08/28/2023).

[19] "Epsg:5973." (n/a), [Online]. Available: https://epsg.io/5973 (visited on 09/14/2023).

[20] "Epsg:4326." (n/a), [Online]. Available: https://epsg.io/4326 (visited on 09/14/2023).

[21] "Gis wiki - decimal degrees." (n/a), [Online]. Available: http://wiki.gis.com/wiki/index.php/Decimal_degrees (visited on 09/14/2023).

[22] "Pyspark documentation - pyspark.sql.functions.udf." (n/a), [Online]. Available: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.functions.udf.html (visited on 09/14/2023).

[23] "Spark connector python." (n/a), [Online]. Available: https://www.mongodb.com/docs/spark-connector/current/python-api/ (visited on 09/14/2023).

Appendices

Appendix A

Collection schemas

traffic_data

```
{
    "_id" :  "objectid",
    "85fractile_speed": "double",
    "average_speed": "double",
    "coverage": "double",
    "timestamp": "date",
    "trpid": "string",
    "volume": "int32"
}
```

accidents

```
{
    "_id" :  "objectid",
    "lane_type": "string",
    "lon": "double",
    "lat": "double",
    "lighting_conditions": "string",
    "road_conditions": "string",
    "road_lights": "string",
```

```
    "road_reference": "string",
    "road_type": "string",
    "road_width": "double",
    "speed_limit": "int32"
}
```


traffic_registration_points


```
{
    "_id" :  "objectid",
    "lat": "double",
    "lon": "double",
    "name": "string",
    "road_reference": "string",
    "speed_limit": "int32",
    "trpid": "string"
}
```

Appendix B

Grafana dashboard

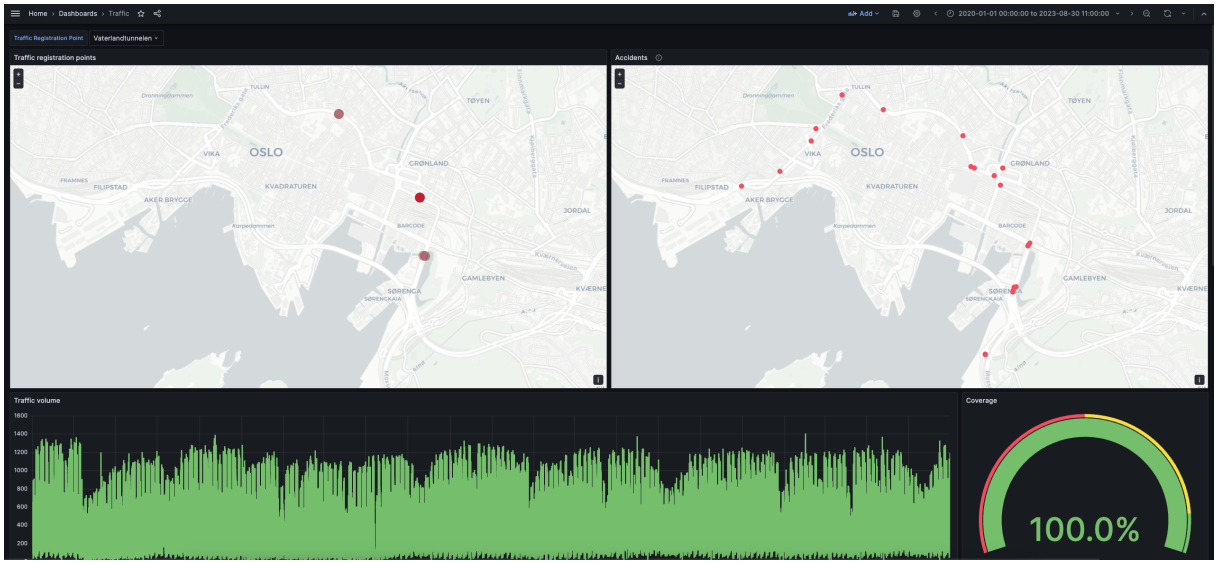

Figure B.1: Grafana dashboard

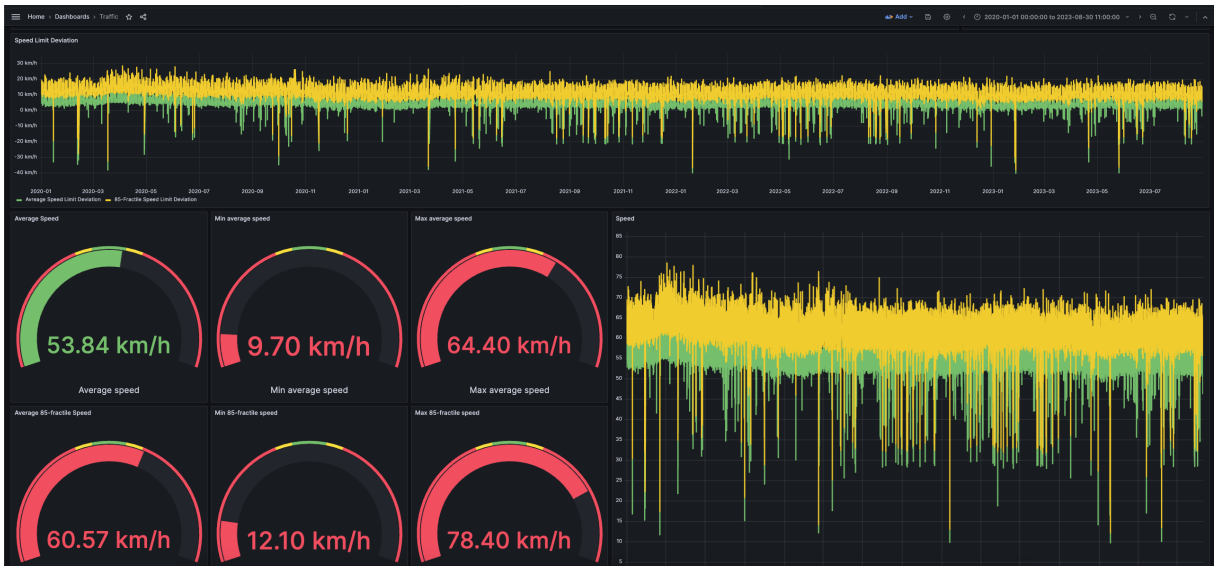

Figure B.2: Grafana dashboard