New York Parking Tickets - Big Data Project 2022/23

Luka Žontar
Faculty of Computer and Information Science
University of Ljubljana
SI-1000 Ljubljana
lz3057@student.uni-lj.si

Ermin Omeragić
Faculty of Computer and Information Science
University of Ljubljana
SI-1000 Ljubljana
eo3031@student.uni-lj.si

1 Introduction

The task of the project was to perform a comprehensive data analysis on a moderately sized dataset (in terms of big data) [1]. The dataset in question is Parking Violations Issuance dataset, which contains all the violations issued since the 2014 fiscal year. Each row represents a violation at the time it is issued, and does not reflect its violation status (whether it was paid, dismissed via hearing, expired or had any other changes to its status). The dataset covers the parking violations issued only in New York City.[2].

The project was carried out as follows. We first locally imported only the 2023 dataset and converted it to Parquet and HDF5 format. We then found several external datasets with which we augmented the original dataset: weather dataset, school locations, businesses, registered vehicles, events, and meaning and prices of parking code fines. We then created a dashboard on which we could easily see the the table statistics for all used tables: number of rows, freshness (latest date that shows up in the table), uniqueness of the table rows, memory usage, percentage of missing values per column and distribution of row counts by date. Next step was exploratory data analysis, followed by data analysis using streams. Then, we transferred all the code to a HPC cluster where we could work with the whole dataset, and carried out all the experiments there as well. We concluded with the comparison of the Dask, Dask-SQL and DuckDB processing performance with Parquet and HDF5 files.

2 Data Ingestion and Augmentation

The New York City Parking Violation 2023 dataset consists of **14,392,430** rows and **45** columns, occupying **2.52 Gigabytes** of memory when saved in a CSV format. For a medium-end personal computer, this may already exceed the RAM memory at disposal, so we had to resort to distributed processing. We wrote an Extract-Transform-Load (ETL) pipeline to first extract the data from the CSV file, format the date and time columns, and then load it into Parquet and HDF5 formats. We do not use custom partitioning when loading the data into these formats, and we end up with 42 Parquet/HDF5 files representing 42 partitions of the dataset. For HDF5 files, we use level 9 compression as the size of the files explodes if no compression is used. The resulting Parquet directory occupies only **478 Megabytes**, while the resulting HDF5 directory occupies **1.70 Gigabytes** of disk space.

We then proceeded to augment the dataset with several external datasets that can help us understand the data better. We first pull the information from the New York City Department of Finance website about the parking violation codes and their respective fines[3]. There were **4782** rows, or **0.03%**, with illegal violation codes, i.e. codes that are not found in the table. We chose not to differentiate the fines between the inner Manhattan region and everything outside that region (the fines differ), and for all data we use the fines as if the violation was made outside of Inner Manhattan.

[1]The entire code repository can be found on GitHub https://github.com/eomeragic1/big-data-project along with thorough documentation.

[2]Most recent Parking Violations dataset can be viewed and downloaded from this link

[3]Violation Codes data can be viewed on this site

We then augmented the dataset with the weather information. We took hourly temperature and precipitation data from the Open-Meteo Historical Weather API for the period of interest[4]. We took 5 different simultaneous measurements, one for each borough in New York (Manhattan, Staten Island, Bronx, Queens and Brooklyn). Before we joined the two tables, we transformed the "Violation Time" column in the original dataset into an integer column that represents the hour of the day (0-23). We then joined the tables based on the date, time and borough.

Afterwards, we similarly augmented the original dataset with schools[5] and event information[6]. We first grouped the data by regions, boroughs in the case of events and police precincts for schools. We then counted all permitted events that happened per day and per borough, and all the schools per police precinct. We could then augment the original dataset with this information. For parking violations, there were **6 500 534** rows with no police precinct information, or **45.1%**. That is why augmentation with school information, even though it was better localized, was not covering all the data.

The parking violation data on the cluster, on the other hand, contains **118,953,219** rows with 45 columns. Only **0.000084%** of rows have illegal violation codes, and **31,221,376 rows**, or **26.2%**, do not have violation precinct information. The CSV files occupy **24 Gigabytes** of disk space, and the processed Parquet files of the dataset occupy **3.1 Gigabytes** of disk space. On the other hand, the Dask workers timed out every time we tried to save the data into a HDF5 file format. We tried increasing the memory of workers, with no success. The augmented dataset contains **114,415,282** rows, which means we lost **4,537,937**, or **3.8%** of rows because of the missing data in the inner joins with Weather and Permitted Events datasets.

3 Data Quality

3.1 Running ETL Jobs

We spent a lot of time trying to develop code that is scalable. Even in a smaller project (in terms of big data) such as this one, it did not make sense to write a script for each augmentation table separately. Instead, we tried to develop code that is scalable. This means that it should require little maintenance and little effort to add another dataset. To this purpose we developed modular libraries and all the work can be re-run by executing 3 different ETL jobs, one for the initial transformations and augmentation, one for the analysis and one for postprocessing that essentially simply calculates the table health checks.

3.2 Health Check Dashboard

Our work also focused a lot on data quality evaluation and testing different methods to evaluate data quality.To this purpose, we first developed a health check dashboard in Plotly with which, we are able to quickly check whether the generated datasets have some obvious issues.

In that dashboard we check how much memory and time was used in execution. In order to check whether or not a certain dataset and column are of value to our augmented dataset, we must be able to check if a column contains values. We measure that by calculating the number of null rows per column and normalize it with number of columns to get the nullness score. The table completeness score is calculated by subtracting the nullness score from 100. Since some of our datasets are connected with the original dataset through timestamp, we also check the distributions over time.

As shown in Figure 1, we see that we managed to successfully retrieve historic data for weather, permitted events and legally operating businesses. While, there are much more rows for parking violations issued, we see approximately uniform distribution of frequency of rows by year, which is a good sign of data quality.

Going forward to table completeness, you will notice that only 3 tables, one of which is the original dataset, contains null values in considered columns. The majority of columns with missing values

[4]Weather dataset can be downloaded from this link

[5]School locations dataset can be viewed and downloaded on this link.

[6]Permitted Events dataset can be viewed and downloaded on this link.

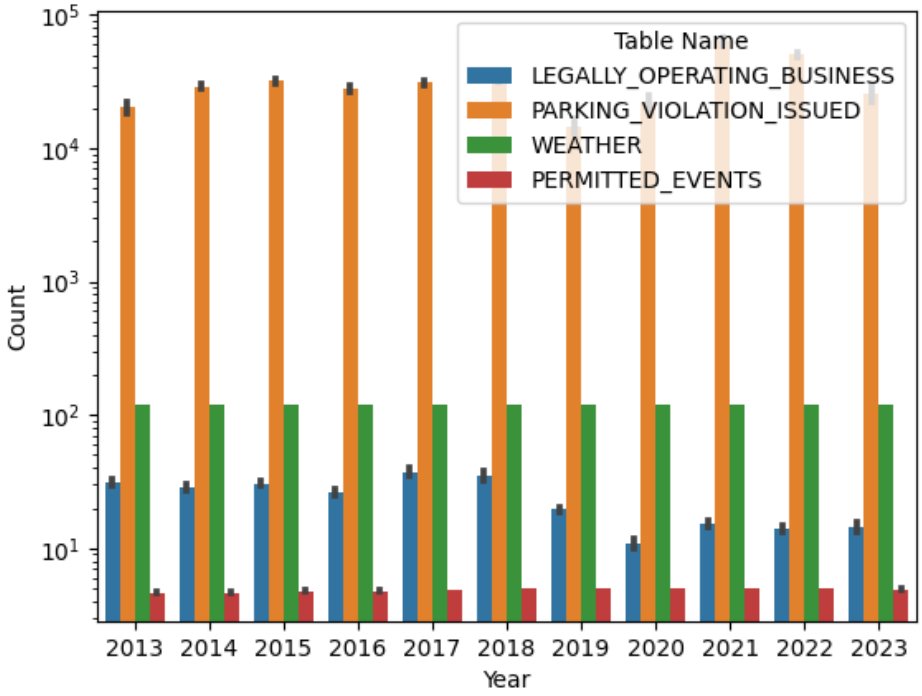


Figure 1: Row count by year and initial datasets.

come from the original dataset. Regarding the datasets chosen for augmentation, there almost no null values, which is once again a good sign of data quality.

3.3 Data Augmentation Quality

One of the initial tasks of this project was to augment the original dataset. As already mentioned, we did that by joining new tables to the original dataset. Figure 3 shows that we were quite successful in performing the data augmentation. We note that the majority of augmented columns have nearly no missing values. Columns from the schools augmentation table had the most missing values at around 25%.

4 Data Analysis

We started by looking at the dynamics of the parking violations. It is interesting to see during which hours of the day are the tickets mostly issued, and what are the reasons for it. This is shown in figure 4. We can see that most of the tickets are on average issued between 6 AM and 3PM, which corresponds to the traffic trends. The average traffic trends are taken from the historical data about traffic on 7 bridges/tunnels on the edges of New York City[7]. We also wanted to check whether this ratio of issued parking violations per hour also corresponds to the ratio of the number of police officers working the daily shifts and the night shifts, but we couldn't find public data about this. We can also see that most of the tickets are issued in the busiest boroughs: Manhattan, Brooklyn and Queens.

We then wanted to see whether there has been an increase in the amount of parking violations issued throughout the time. This is shown in figure 5. The data for the 2020 fiscal year is missing, hence the dip in the amount of parking violations issued during that time. We also normalized the plot with

[7]The Traffic dataset can be found on the following link

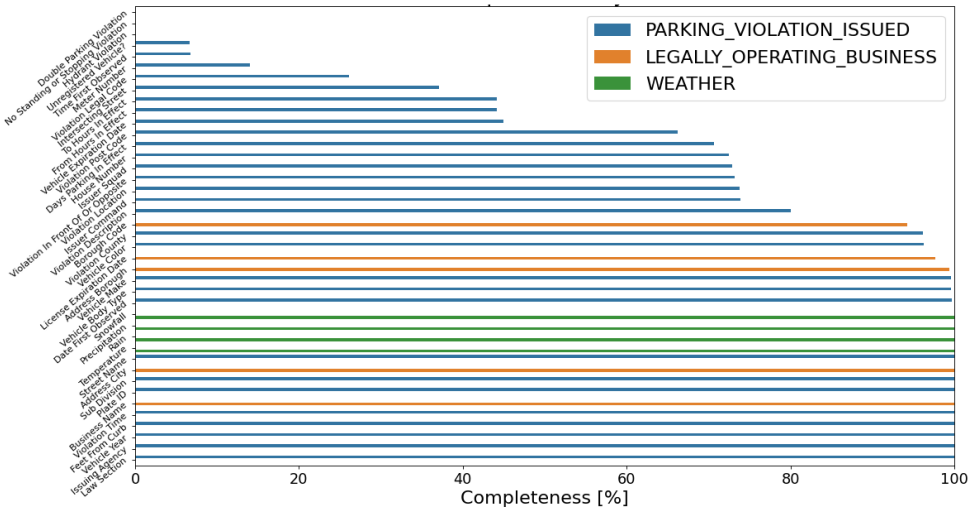


Figure 2: Table completeness of the initial datasets.

respect to the population density in the boroughs. We can see that the there is an upwards trend in the amount of violations issued with each year. Queens stands out in this area - this may be due to the most of the middle-rise and low-rise buildings can be found here, and relatively low population density (8542 people/sq.km, compared to 13482, 15227 and 28872 people/sq.km. in Bronx, Brooklyn and Manhattan respectively).

What is also interesting to see is at what days of the week are most of the violations issued. Looking at the figure 6, it is obvious there are less violations issued during weekends, as there is much less traffic and the officers' working times are more restricted.

Next figure shows which vehicle makers have the most tendencies to get a ticket. We took the number of parking violations issued by a vehicle maker and divided this number by the total number of registered vehicles in New York State for that vehicle maker. The results are shown in figure 7. Outliers in this plot, which have been left out, are Hino, Rover and Freuheur, which have relative ratios equal to 16206, 15929 and 3799, respectively. This may be due to them having alternative vehicle maker codes in the registered vehicles dataset along with the one that is present in the parking violations dataset.

We can also see which vehicle makers' drivers paid the largest amount of fines in the last 10 years, shown in figure 8.

Combining the weather data with the violations data, we can investigate if the outside weather conditions impact the amount of violations issued. We take a look at temperature values and precipitation (rain and snowfall). The plots are shown in figures 11 and 10. We normalize the amount of parking violations issued when that temperature/rain/snowfall was outside with the total amount of times that temperature/rain/snowfall was recorded during the same period. We see that there is a slight negative correlation with the temperature records and the amount of parking violations issued, while the rain and snowfall graphs are too unstable to make any conclusions.

Final analysis plot shows how school vicinity proxy affects the amount of parking violations. We compared the amount of parking violations that happened in certain violation precinct to the number of schools registered in that precinct. The results are shown in figure **??**. We see that there is actually a slight negative trend in this regard, which is somewhat surprising.

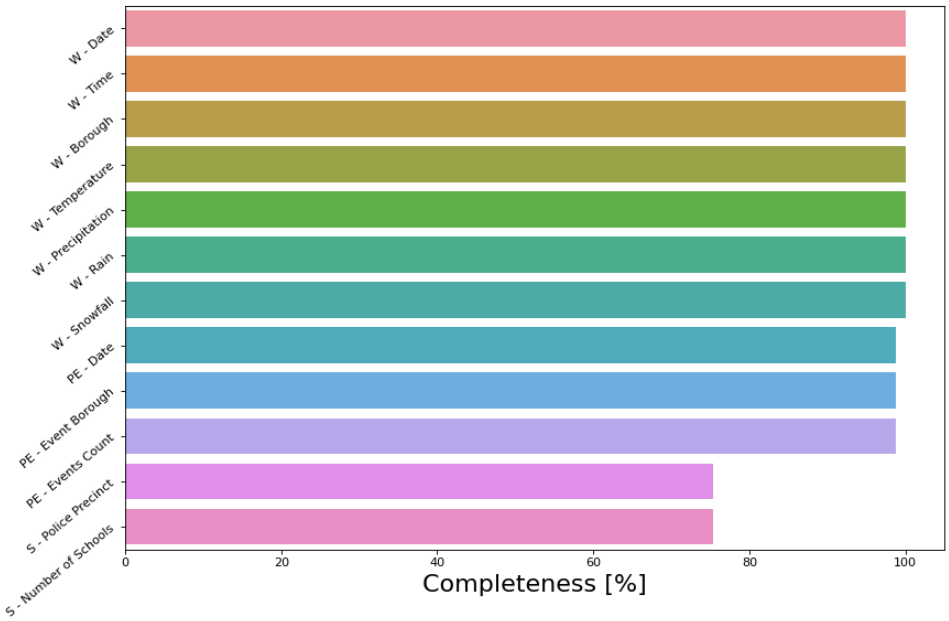


Figure 3: Data augmentation column completeness.

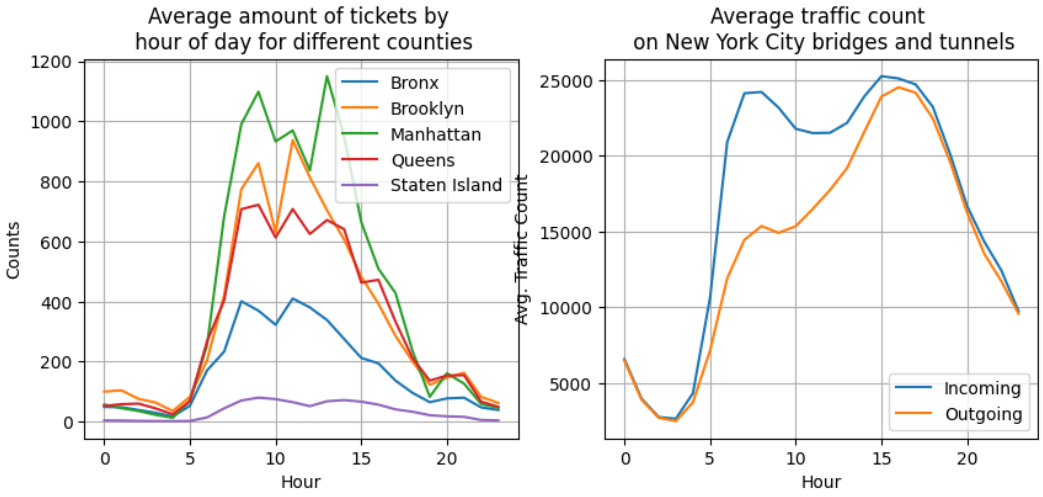


Figure 4: Hourly trends of Parking violations issuance compared to hourly trends in traffic on the outskirts of New York City

5 Data Streams

A big challenge was to adopt the distributed way of processing to streaming data. Unfortunately, there is no straightforward way of streaming data from Dask DataFrames. That is why we chose to use Apache Kafka as a data ingestion engine. We first convert the parking violation data to Parquet format, so we can load separate partitions one by one with Pandas, which should fit into memory. Then, we send each row of data to a Kafka topic.

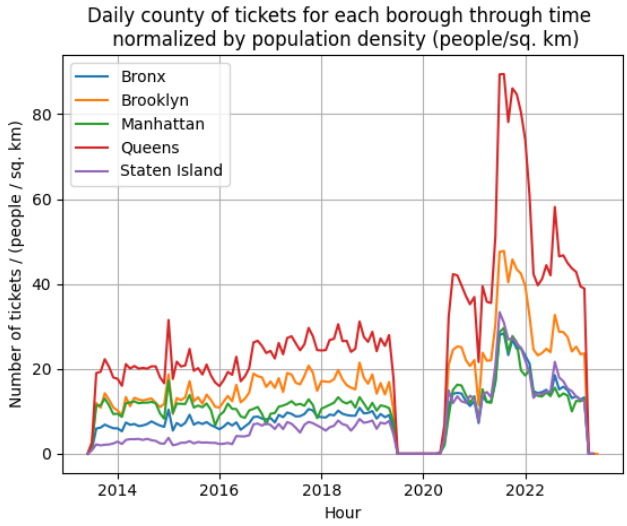


Figure 5: Number of parking violations issued throughout the time.

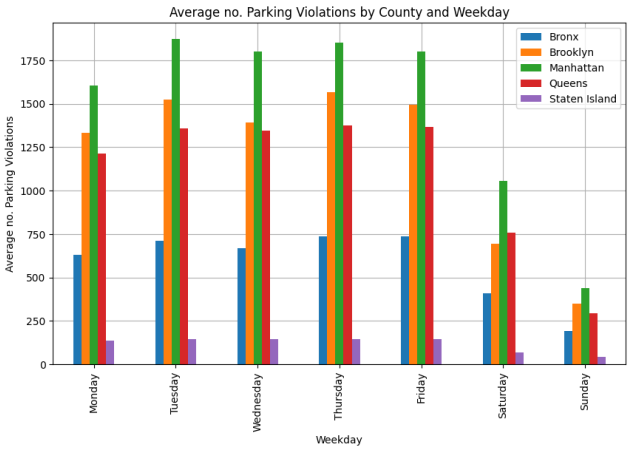


Figure 6: Average number of parking violations issued with respect to weekdays.

For data stream processing, we opted to test two approaches. We first use Streamz library to connect to the Kafka broker and process the data. A big disadvantage of this approach is that the Streamz library is very limited in its functionalities, and processing comes down to using several low level functions. However, Streamz stream connects easily to plotting library HoloViz, so we can easily create dynamic real-time visualizations.

5.1 Scaling data streams

Another challenge occurred, when we wanted to scale data streams. For this task, Streamz library has no apparent solution. While there are several costly cloud solutions that make data streams extremely easy to adopt, we wanted to use an open-source, free approach that we were able to scale on our high-performance cluster.

For this approach, we found a library named Faust. Faust has several advantages over the Streamz library. First of all, it allows users to write regular Python code and functions when processing the data. The biggest advantage over Python Streamz library was that Faust is able to run concurrently with more workers, which would allow us to adopt cluster resources when processing data streams as well. While we did not run data streams on the cluster, we prepared a proof of concept on how to use Faust for distributed computing. This is described more thoroughly in the GitHub repository.

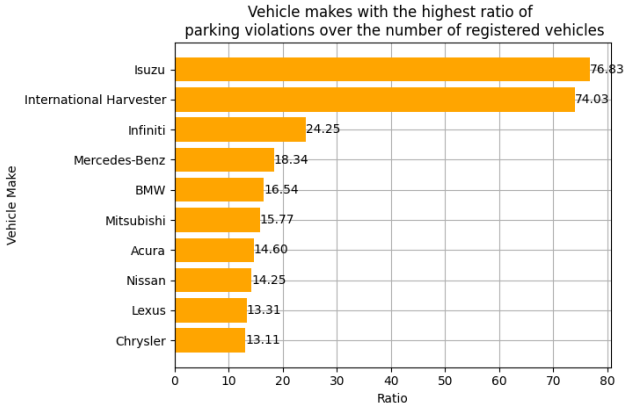


Figure 7: Ratio of the amount of violations to the number of registered vehicles per vehicle maker

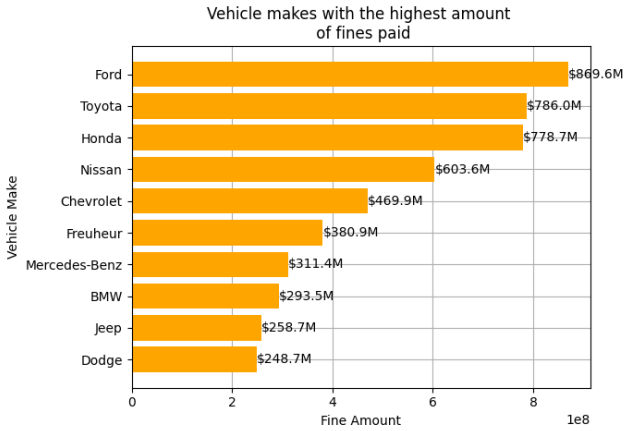


Figure 8: Amount of fines paid by each vehicle maker's drivers.

For distributed computing with Faust, one must first create a Kafka topic that will transmit data. This Kafka topic must be created in a way to enable partitioning. Then, as we are transmitting data over the Kafka producer node, we must introduce a partitioning logic, which may be random, that is, the produced can randomly decide which worker node gets the data. Then, we initialize multiple Faust workers and introduce a sink node that will aggregate the data back together. This enables us to either visualize, save or print the results.

The only disadvantage we noticed with Faust was that the library does not have any visualization possibilities, so we had to send the output of the Faust processed stream to a new Kafka topic that served as the sink node. Then, we were able to ingest the curated stream data from Kafka using asynchronous callbacks that then updated the plots created with Plotly library.

6 Processing Performance Comparison

In order to ensure the optimal processing method is used in the project, we tested several different processing methods. As shown in Figure 12, we split those processing method in 2 groups. The first is comparing **parquet** file storage versus the **hdf5** file storage and the other is comparing regular Dask with Dask SQL and DuckDB tool.

The most appropriate tool for processing data first and foremost depends on the size of the dataset and user's preference of Python versus SQL. While there exist libraries that put DuckDB over a Dask engine and consequently make it scalable, we only tried the default setting. Due to this reason, DuckDB is only appropriate for data that can be processed in-memory. Dask on the other hand,

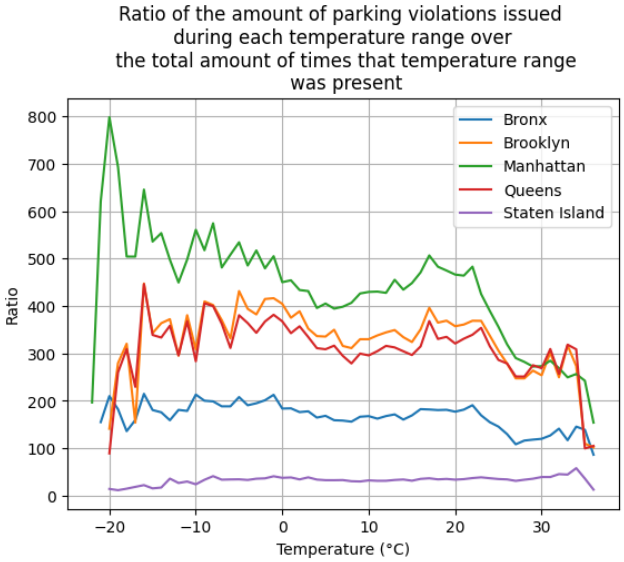
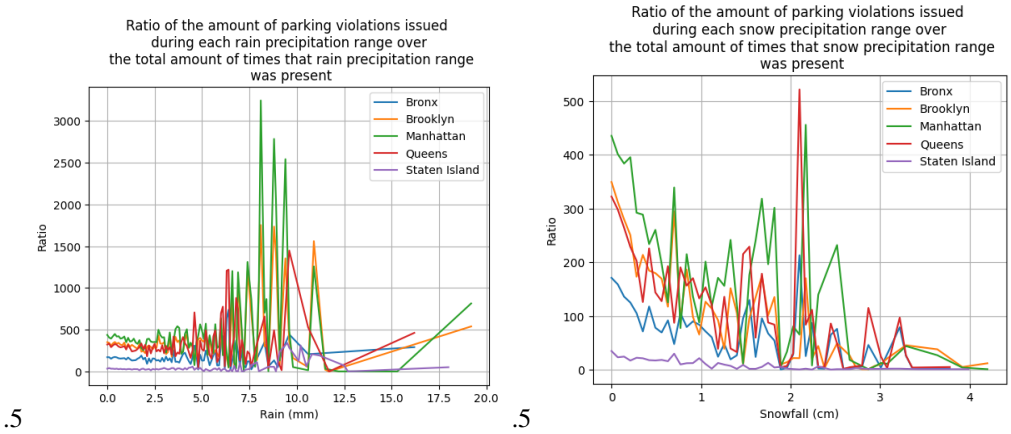


Figure 9: Average amount of violations issued in each temperature range



.5 .5

Figure 10: Average amount of violations issued in each snowfall and rain range

is capable of great scalability, but performs poorly on smaller datasets. This can also be seen in Figure 12, where we tested all three tools on different tables of smaller size due to DuckDB memory disadvantages. On such datasets, DuckDB outperforms Dask approaches by far. On the other hand, the regular Dask is slightly faster than Dask SQL. Dask also has an advantage to change between SQL and Python and make transformations wherever is easier. Regarding the file storage, it seems that data processing on parquet files slightly outperforms data processing on HDF5 files.

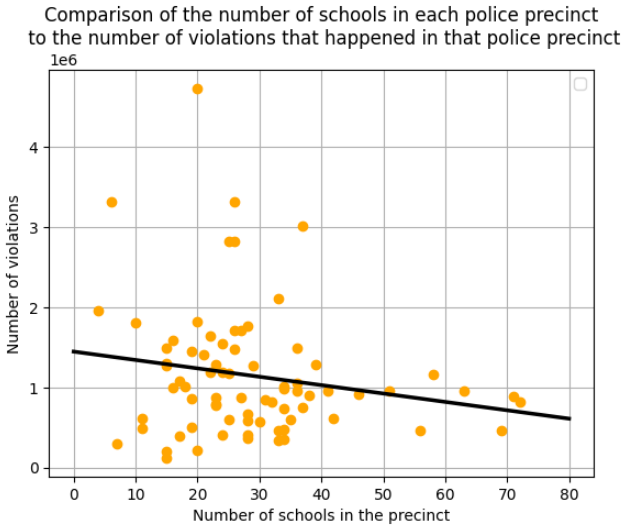


Figure 11: Comparison of amounts of violations issued in one police precinct and the amount of schools registered in the precinct

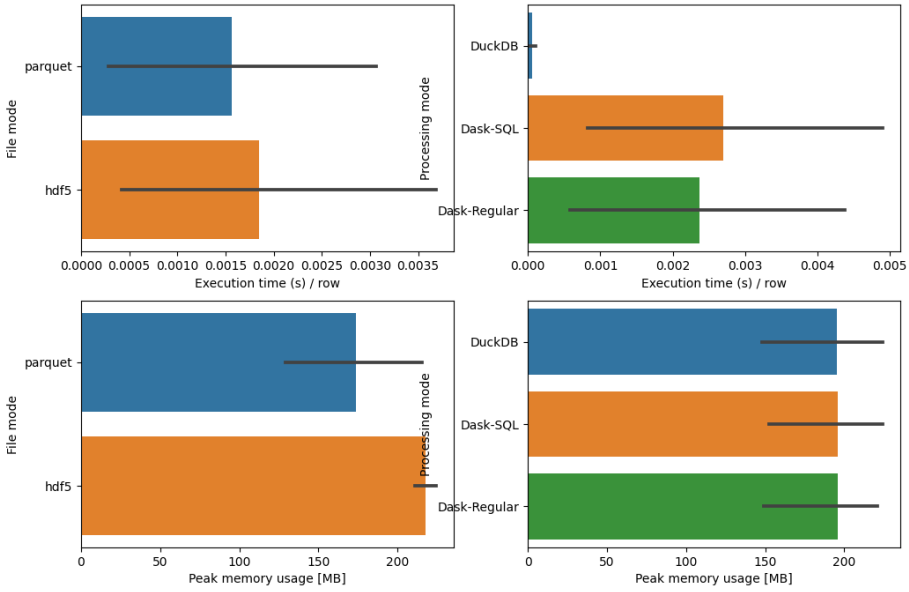


Figure 12: Performance comparison between file storages (hdf5, parquet) and tools (DuckDB, regular Dask, Dask SQL

7 Scaling on high-performance cluster

A great deal of our efforts went to efficiently scale our code on a high-performance cluster. All the data processing was performed via SSH connection using a Dask SLURM cluster that requested 4 nodes, with 4 processes, 12 cores and 8GB of memory each. To be able to run the same code locally and remotely, we set up a configuration file that holds all the differences between the local and remote

	Size [MB] (csv)	Execution time (s)	Number of tasks	Compute time	Transfer time	Disk-write time	Workers	Threads	Memory [GiB]
Parking Violations Issued	23,798	490.29	386	2 hr 17 m	120.11 ms	/	8	24	59.60
Permitted Events	3493	436.05	68	3 hr 9 m	21.96 ms	34.74 ms	16	48	119.20
Registered Vehicles	1298	8.73	27	118.94 s	268.69 ms	/	16	48	119.20
Legally Operating Businesses	62	458.94	2	3.60 s	/	/	4	12	29.80
Traffic	52	2.92	6	2.61	/	/	16	48	119.20
Weather	21	2.62	2	2.30	/	/	16	48	119.20
Parking Violation Codes	1	210.78	2	17.65	/	/	4	12	29.80
Schools	1	861.97	4	198.97	/	/	16	48	119.20
Violation County	1	95.72	2	14.51	/	/	16	48	119.20

environment. In our GitHub repository, we also documented how to connect to the cluster, execute the initial transformation script and even connect to Dask dashboard and Jupyter notebooks via SSH tunneling. Connecting to the Dask dashboard enables us to better understand what is happening with our workers and why and if they are inefficient. Connecting directly to Jupyter notebooks allows us to allocate cluster resources and perform ad-hoc analyses very quickly.

One of the main parts of our results are shown in Table 7, where we see how the data was handled in the cluster environment. The data was extracted from Dask performance reports that mimic the Dask dashboards. Note that the biggest table (Parking Violations Issued) had almost 24 GB of data and was processed in under 10 minutes, along with all the other tables that all together summed up to almost 30 GB of data. As we can see the original data set processing required running almost 400 tasks and using 8 workers used more than 2 hours of compute time. This shows how scalable is our solution and how distributed computing enables processing of volumes that were unseen before. We note a few inconsistencies, where smaller tables have high execution times, however this may simply be due to other people using the same cluster. The more interesting columns are transfer time and disk-write time, which essentially show the time when workers are not working but either communicating or writing to files. Communication between workers is inefficent and should be minimized if possible. We see that registered vehicles table is an appropriate candidate for procedure optimization.

8 Conclusion

Data processing is on the rise and processing big and complicated datasets with appropriate velocity, will be crucial to the developement of the field. In this project we tried several approaches to implement data processing and analysis on a moderately sized dataset. While we did not use the data to train machine learning models and test them on big datasets, we quite successfully developed means to both process the data, extract some interesting insights and eventually scale these processes to clusters of computers. Besides the analytics and insights, we were able to find a solution to perform data quality checks, scale data stream processing and regular data processing and learned which file storage and tools are more appropriate in which situations.

In future work, we should continue focusing on scaling solutions, because real time data can rarely fit in the memory of a personal computer. Additionally, we should also try some machine learning methods that will give us insights in how machine learning can be done in a distributed manner. We might also test some open-source solutions such as *dbt* for data quality and Apache Airflow for orchestration.