FACULTY OF COMPUTER AND INFORMATION SCIENCE

# New York Parking Tickets Report

Student name: *Maja Kolar (63220476), Anja Hrvatič (63220458)*

Course: *Big Data*
Due date: *August 18th, 2024*

This is our report for the Big Data Final Project, where our goal was to perform a comprehensive data analysis on a dataset of New York Parking Violations for the years 2014 - 2024.

## 1. T1 - File Types

Our objective for this task was to import the CSV data of NYC Parking Violations and convert it to Parquet and HDF5 formats.

Parquet is chosen for its columnar storage format, which is efficient for both read and write operations, especially for large datasets [6]. In columnar storage, data is organized by columns rather than rows, allowing for rapid skipping of non-relevant data. This organization significantly speeds up aggregation queries compared to row-oriented databases. As a result, columnar storage has led to hardware cost savings and reduced latency in data access. This is especially beneficial when used in combination with certain cloud service providers that charge based on the amount of data scanned per query (Amazon Athena) or time-based charging services (Google Dataproc). The original CSV files were too big to be simply read into a pandas data frame and then transformed into Parquet files, so we read the CSV files in chunks and created the Parquet files. The rows were divided into row groups of 1048576 rows. This allows for efficient reading of specific parts of the data, as you can read or skip entire row groups without needing to process the entire file. Additionally, snappy compression was used to save disk space without significantly slowing down the write and subsequent read operations.

Hierarchical Data Format version 5 (HDF5) file format can be thought of as a file system contained and described within one single file. The term "Hierarchical" in HDF5 refers to its tree-like structure, similar to a file system. This structure allows for an organized way to store datasets and metadata. In an HDF5 file, you can have groups that contain other groups or datasets, much like folders containing files and subfolders in a computer's file system. It enables fast read and write operations. You can append new data to an existing HDF5 file without altering its structure, making it a highly flexible and extensible format. HDF5 files offer on-the-fly compression, which conserves disk space. This feature is especially beneficial when handling very large datasets. The files were produced in the same way as Parquet files - reading chunks of data and adding them to files. HDF5 files offer different types and levels of compression that help decrease the file size and retain speed.

A comparison of file format sizes can be seen in Table 1. We observe that Parquet files are almost 10 times smaller than CSV files. HDF5 files can be created with compression or without it. We created files with no compression and with the highest level

Table 1: Different file formats have different sizes.

| File | CSV [GB] | Parquet [GB] | HDF5 [GB] | HDF5 compressed [GB] |
|---|---|---|---|---|
| 2014 | 2.5 | 0.311 | 2.9 | 1.4 |
| 2015 | 3.1 | 0.495 | 3.4 | 2.1 |
| 2016 | 2.7 | 0.313 | 3.1 | 1.5 |
| 2017 | 3.0 | 0.553 | 3.6 | 2.2 |
| 2018 | 3.2 | 0.397 | 4.0 | 1.9 |
| 2019 | 3.0 | 0.363 | 3.9 | 1.8 |
| 2020 | 3.2 | 0.376 | 4.1 | 1.9 |
| 2021 | 3.9 | 0.441 | 4.9 | 2.0 |
| 2022 | 4.0 | 0.438 | 5.0 | 2.1 |
| 2023 | 4.4 | 0.487 | 5.6 | 2.3 |
| 2024 | 3.1 | 0.323 | 3.9 | 1.6 |

of compression. The non-compressed files are sometimes twice as big. We used Blosc compression which uses a blocking technique to reduce activity on the memory bus as much as possible. In this case, the compression is beneficial as loading the compressed data to memory and decompressing it using the CPU can be faster than just loading the uncompressed data.

## 2. T2 - Data Augmentation

For our next task we had to augment the Parking Violations data with additional information sources such as the weather and the vicinity of the following factors: schools, events, businesses and major attractions.

**2.1. Weather.** We extracted the weather data with NCEI Data Service API [2], which provides historical weather data for various locations, allowing us to match the weather conditions to the dates of the parking violations. We chose the location of Central Park, Manhattan, arguably because further localization of weather for a city is redundant. We retrieved data for each fiscal year (July 1 of the previous year to June 30 of the current year) from 2014 to 2024, since Parking Violations data is organized likewise. The specific weather parameters we focused on were:

- Precipitation (PRCP): Total daily precipitation measured in inches.
- Snowfall (SNOW): Total daily snowfall measured in inches.
- Average Wind Speed (AWND): Daily average wind speed measured in miles per hour.
- Average Temperature (TAVG): Daily average temperature measured in degrees Fahrenheit.
- Minimum Temperature (TMIN): Daily minimum temperature measured in degrees Fahrenheit.

- Maximum Temperature (TMAX): Daily maximum temperature measured in degrees Fahrenheit.

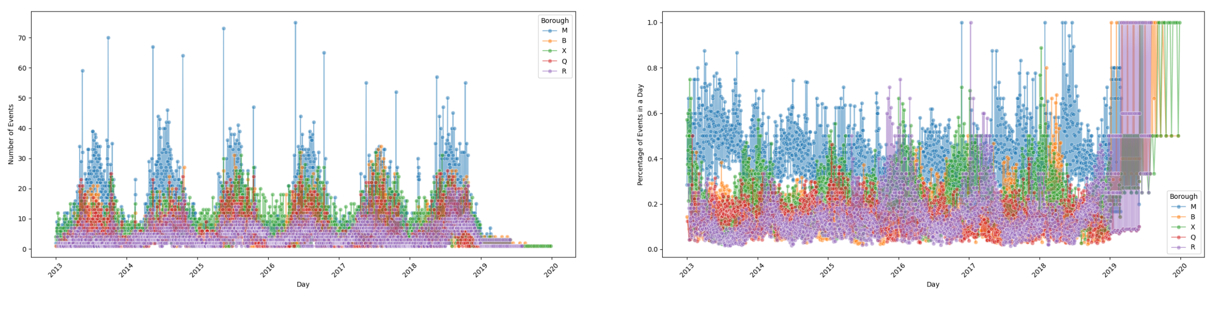
Each day's weather data was matched with the corresponding parking violation records based on the date. This integration allowed us to analyze how different weather conditions influenced parking violations on a daily basis.

**2.2. Schools.** We extracted the data for the vicinity of primary and high schools from the School Point Locations dataset from NYC Open Data [4], which includes school point locations and basic information about schools. We aggregated and normalized the data by the total number of schools. We saved the normalized number of schools in each Borough, to join them with Parking Violations data.

**2.3. Events.** We extracted the event data from NYC Parks Events Listing from NYC Open Data [4], which is used to store event information displayed on the public NYC Parks website. We used two related tables from this database: Event Listings and Event Locations. We explored the data and inner joined them on event ID. We extracted Event Count data grouped by Boroughs and Dates to join them on both with Parking Violations data. The distribution of events by Day and Borough is shown in Figure 1.

**2.4. Major Businesses.** We extracted data about NYC major businesses from Legally Operating Businesses from NYC Open Data [3], which features licenses issued by the Department of Consumer and Worker Protection to businesses and individuals so that they may legally operate in New York City. We performed analysis in the same way as with School data, whereas here we had to clean the data first for unanimous names of boroughs. We saved the normalized number of operating businesses in each Borough, to join them with Parking Violations data.

**2.5. Major Attractions.** We extracted data about NYC major attractions from the 348 New York Tourist Locations dataset from Kaggle [1], which contains scraped and cleaned data of 348 tourist locations of New York City. The location data contained only ZIP codes, thus we found the ZIP code data [5] and connected ZIP codes to the Borough. We performed analysis the same way as with School and Businesses data; we cleaned the data and extracted the normalized number of tourist attractions in each Borough, to join them with Parking Violations data.



(a) **Total events** (b) **Normalized by number of daily events.**

Figure 1: **Distribution of Events by Day and Borough.**

## 3. T3 - EDA

Using Dask on Parquet files we examined the data. We used LocalCluster and Client with 6 workers, 2 threads per worker and a 4GB memory limit per worker to explore the full augmented dataset. Firstly we explored the head, column types, and missing values and found the following interesting facts;

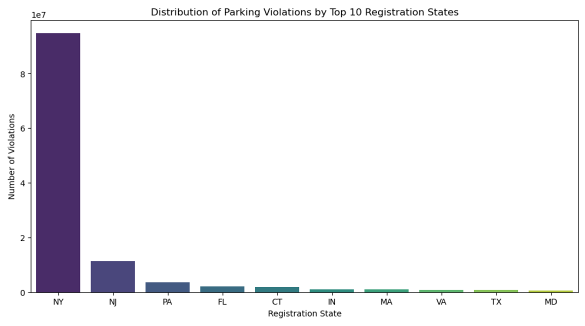
- Out of 43 features, none are continuous except for time features;
- Some columns either have too much missing data, too many empty cells, or are encoded without an available system for decoding (Street Code1/2/3, Vehicle Expiration Date, Violation Location, Violation Precinct, Issuer Precinct, Issuer Code, Issuer Command, Issuer Squad, Time First Observed, From Hours In Effect, To Hours In Effect, Violation Legal Code, Unregistered Vehicle?, Vehicle Year, Meter Number, Feet From Curb, Violation Post Code, No Standing or Stopping Violation, Hydrant Violation, Double Parking Violation, Law Section, Sub Division);
- We fixed the 5 Boroughs (NY, K, Q, BX, R), and the Top 10 Streets with most parking violations (Broadway, 3rd Ave, 5th Ave, Madison Ave, 2nd Ave, Lexington Ave, 1st Ave, Queens Blvd, 7th Ave, 8th Ave) for further analysis. We filtered out the missing data for boroughs (3.49 %), mapped different borough names to the chosen five and discarded all data that does not conform to that;
- Data augmentation was completely successful for the data that connected only through boroughs (Schools, Businesses, Attractions). Weather information had some features that were frequently missing for our time period (AWND and TAVG), while others were successfully augmented. There were 91% rows with no event data, which was connected through borough and time, which can be attributed to a smaller event dataset than the parking violations dataset.

Four EDA plots are shown in Figure 2. We note that most tickets are issued to vehicles registered in New York and that most tickets are issued in the Manhattan borough. Interestingly, we observe no significant correlation between average precipitation and number of daily tickets. Figure 3 shows a map of NYC streets with the most violations, with the heatmap showing the number of violations on that particular street. Staten Island does not have a street among the top 20 streets with the most violations, while Manhattan has the majority.

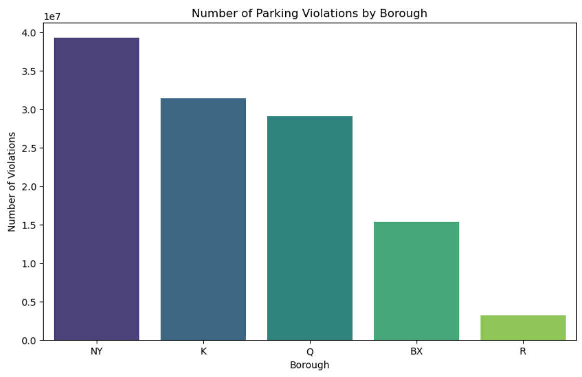
## 4. T4 - Streaming Data

**4.1. Rolling Statistics.** We simulated a stream process with augmented data using Kafka as a broker. We were interested in rolling statistics and clustering the data according to interesting features. The Faust library was used to create agents - consumers for stream processing. We assumed the data was ordered by time, without errors. The goal was to get an update on rolling statistics every week:

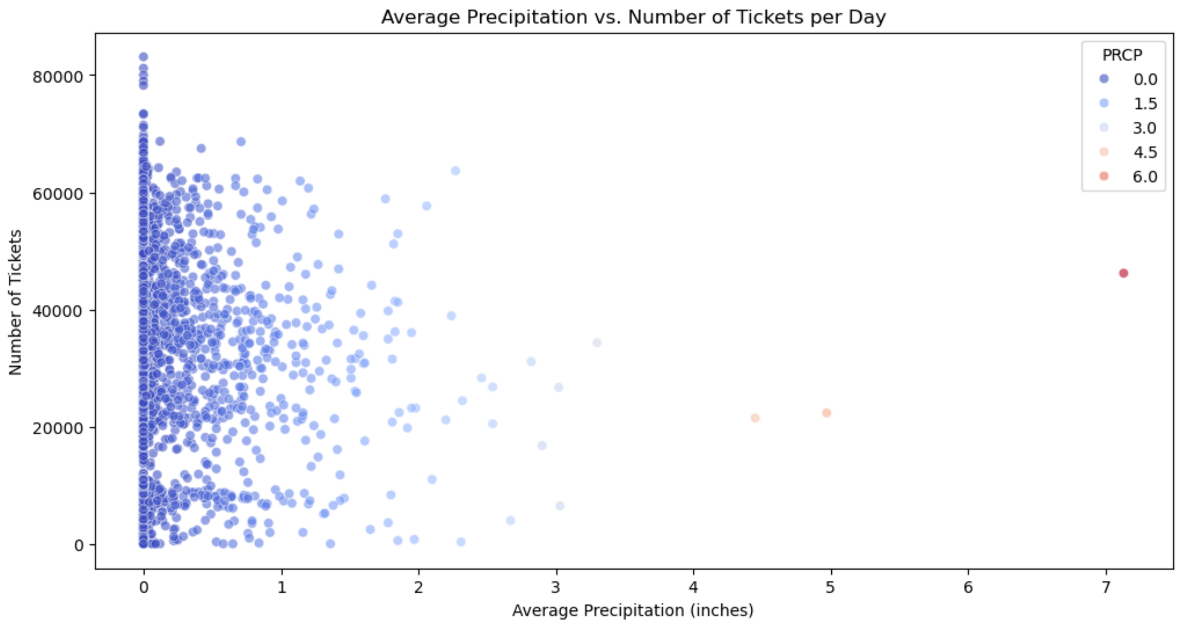
- Mean number of tickets per day of the week,
- Median number of tickets per day of the week,



(a) **Registration States with Most Violations**



(b) **Violations per Borough**



(c) **Effect of Average Precipitation**

Figure 2: **Exploratory Data Analysis**

- Standard deviation of the number of tickets per day of the week,
- Max number of tickets per day,
- Percentage of tickets per vehicle make.

These statistics were obtained for all data, boroughs and the ten most interesting streets: Broadway, 3rd Avenue, 5th Avenue, Madison Avenue, 2nd Avenue, Lexington Avenue, 1st Avenue, Queens Boulevard, 7th Avenue, and 8th Avenue.

**4.2. Clustering.** For clustering Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm was used. It decides the dataset into dense regions separated by sparse regions. It is resilient to outliers because it identifies clusters based on densely populated areas of data, treating isolated points as noise. It doesn't require the user to specify the number of clusters in advance, making it more flexible. DBSCAN can detect clusters with complex shapes and isn't limited by assumptions about cluster

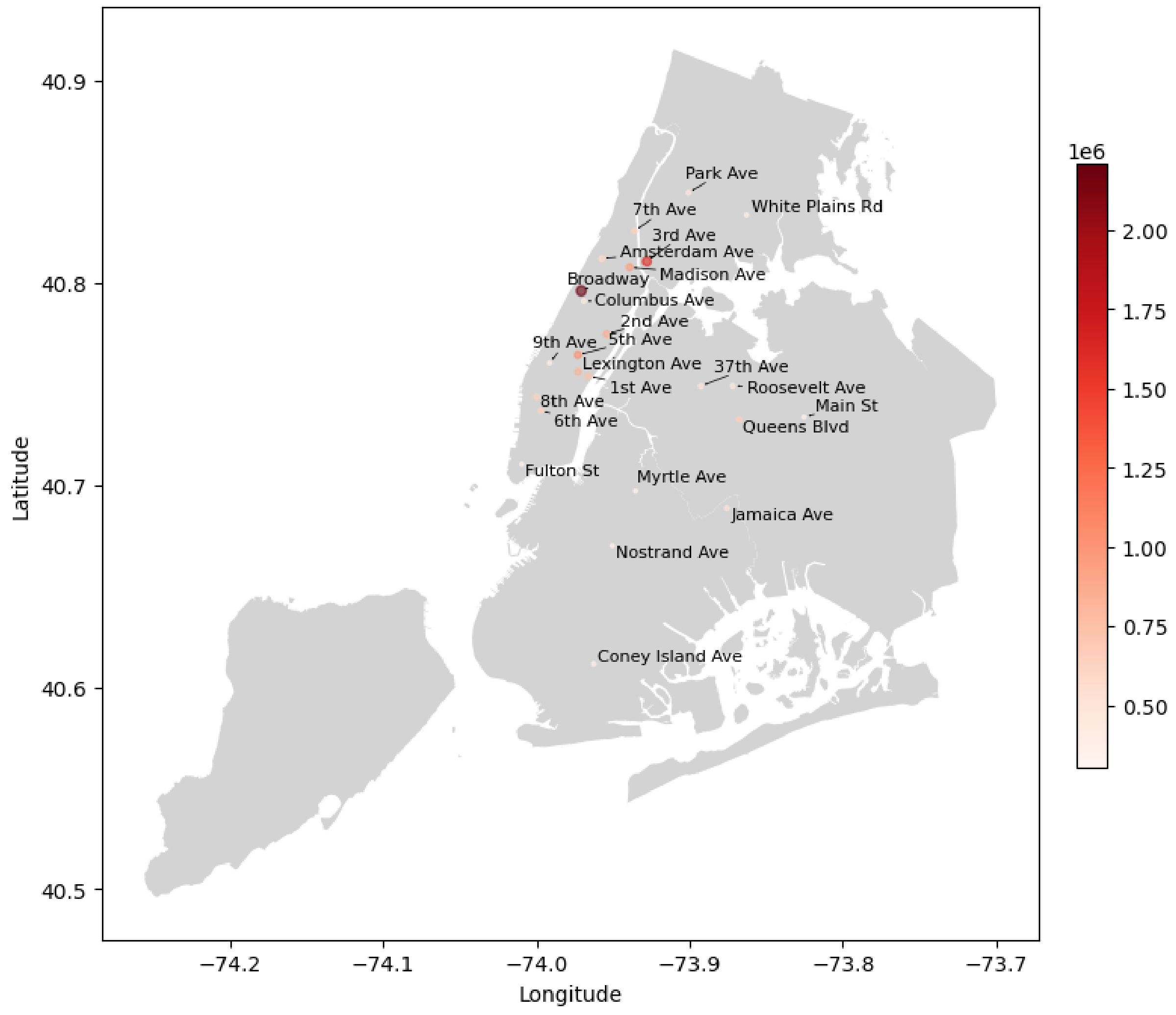


Figure 3: **Map of Streets with Most Violations.** Heatmap shows the number of violations on that particular street.

geometry, making it ideal for datasets with irregular structures. It has two parameters - $\epsilon$ (distance threshold for defining neighbourhood) and the minimum number of points required to form a dense region. To evaluate the quality of the clustering we used the Silhouette score. It is a measure of how similar an object is to its cluster (cohesion) compared to other clusters (separation). The value can be between -1 and 1, the higher number means the data point is well matched to its cluster and poorly to the neighbourhood clusters. A silhouette score higher than 0.5 represents a good clustering.

**4.3. Stream process setup.** We used Kafka to set up the producer, which reads the CSV files line by line and sends them to the topic *tickets*. For stream processing we used Faust. In total 5 agents - consumers were utilized:

1. **Process tickets**:

   This agent is subscribed to topic *tickets* and is responsible for obtaining the features needed for calculating and updating rolling statistics. It extracted the day of the week, street name, borough and vehicle make. It counted the number of tickets for each day for all data, for boroughs and the ten interesting streets. At the end of every week, the weekly counts were sent to separate topics: *topic_rolling_stat* (for all data), *topic_rolling_stat_boroughs* (for boroughs) and *topic_rolling_stat_streets* (for interesting streets). To keep track of past weeks to detect the start of a new week, we used a Faust table which is a distributed in-memory dictionary, backed by a Kafka changelog topic used for persistence and fault-tolerance.

2. **Process rolling statistics - all data**:

   This consumer is subscribed to *topic_rolling_stat* which is receiving weekly counts for each day in the week. The mean, median, standard deviation, max and percentage of vehicle brands are calculated.

3. **Process rolling statistics boroughs**:

   This consumer is subscribed to *topic_rolling_stat_boroughs* which gets the weekly counts for each day in the week for each borough. The mean, median, standard deviation, max and percentage of vehicle brands are calculated.

4. **Process rolling statistics streets**:

   This consumer is subscribed to *topic_rolling_stat_streets* which gets the weekly counts for each day in the week for each interesting street. The mean, median, standard deviation, max and percentage of vehicle brands are calculated.

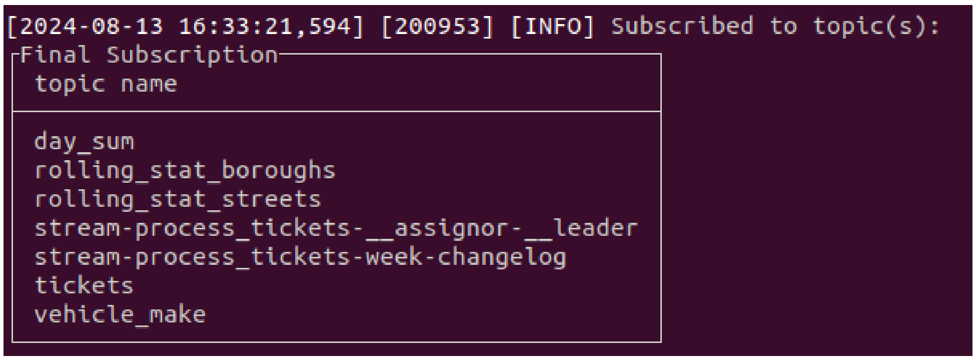
5. **Clustering**:

   The consumer is subscribed to the topic *tickets* and then extracts features that are interesting for clustering: day of the week, time, borough, vehicle year, wind, precipitation, snowfall, daily maximum temperature, daily minimum temperature, normalized number of schools in the area, normalized number of major businesses in the area, and normalized number of attractions in the area. Our chosen clustering algorithm was DBSCAN After 1000 tickets and their features were collected, the numerical ones were normalized and categorical were one-hot-encoded. The clustering model was updated with these 1000 data points. The silhouette score was calculated and we checked the number of clusters.

We separated topics according to tasks for better organization, as only sending data relevant to the task to its topic makes it easier to manage. It is easier to troubleshoot if any problems arise. If something goes wrong with a specific datastream it will not affect the others. Additionally, separating topics allows for the parallelization of tasks. Figure 4 shows the topic our Faust application subscribed to.

**4.4. Results.** The stream process was started with data from 2014. To maintain the order of tickets, they needed to be sent to Kafka 0.01 s apart. Sometimes they arrived at the consumer in different order anyway, which was problematic for time data. Due to large amounts of data, we only waited for the execution of the script for the first few weeks.

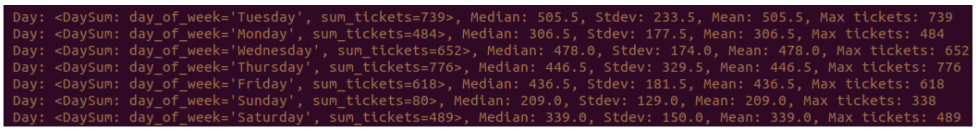
Figures 5, 6, and 7 show examples of the weekly output of rolling statistics for all data, boroughs and interesting streets respectively. We can see that there are the most tickets issued on Tuesdays and Thursdays. The number of tickets issued in Manhattan district (New York City County) is way higher than for other boroughs. The reason might be, that there are more paid parking spots in that area, and that the local traffic



```
[2024-08-13 16:33:21,594] [200953] [INFO] Subscribed to topic(s):
┌Final Subscription──────────────────────────────────┐
│ topic name                                         │
├────────────────────────────────────────────────────┤
│ day_sum                                            │
│ rolling_stat_boroughs                              │
│ rolling_stat_streets                               │
│ stream-process_tickets-__assignor-__leader         │
│ stream-process_tickets-week-changelog              │
│ tickets                                            │
│ vehicle_make                                       │
└────────────────────────────────────────────────────┘
```

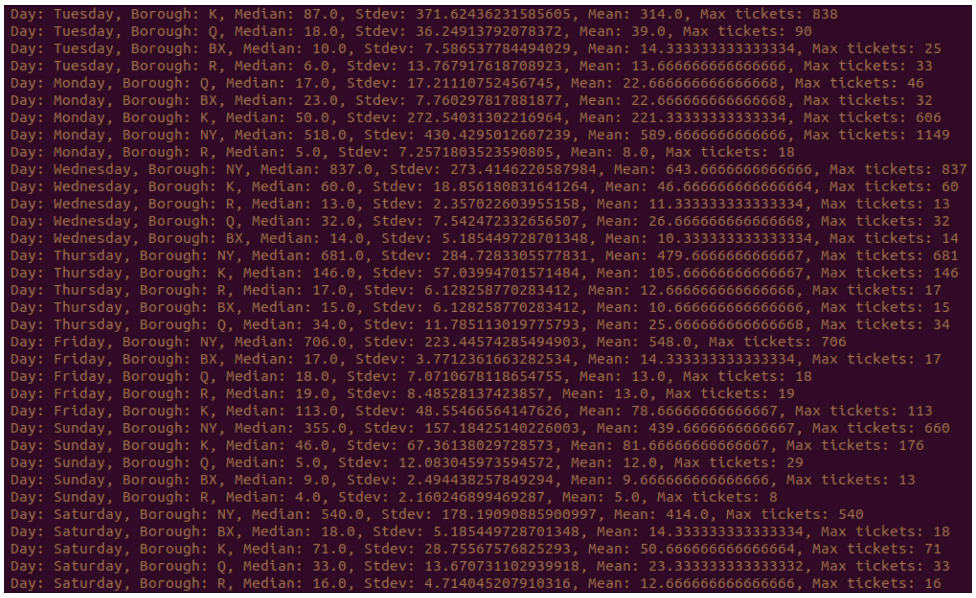
Figure 4: Topics of the stream process.

wardens are more strict. Staten Island (Richmond County) is more suburban compared to the other boroughs. It has more residential neighbourhoods with private driveways and less commercial activity, which reduces the need for on-street parking and the likelihood of parking violations. This is in agreement with results showing a low number of tickets in said borough. For rolling statistics on streets, the streaming process was running for about 2h, after which it only detected tickets from 2 of our most interesting streets. These were Broadway and 3rd Avenue. Broadway is the longest street in New York so it makes sense that there is at one ticket issued there in the first few weeks of fiscal year of 2014. The process would need to run longer to get more interesting data.



```
Day: <DaySum: day_of_week='Tuesday', sum_tickets=739>, Median: 505.5, Stdev: 233.5, Mean: 505.5, Max tickets: 739
Day: <DaySum: day_of_week='Monday', sum_tickets=484>, Median: 306.5, Stdev: 177.5, Mean: 306.5, Max tickets: 484
Day: <DaySum: day_of_week='Wednesday', sum_tickets=652>, Median: 478.0, Stdev: 174.0, Mean: 478.0, Max tickets: 652
Day: <DaySum: day_of_week='Thursday', sum_tickets=776>, Median: 446.5, Stdev: 329.5, Mean: 446.5, Max tickets: 776
Day: <DaySum: day_of_week='Friday', sum_tickets=618>, Median: 436.5, Stdev: 181.5, Mean: 436.5, Max tickets: 618
Day: <DaySum: day_of_week='Sunday', sum_tickets=80>, Median: 209.0, Stdev: 129.0, Mean: 209.0, Max tickets: 338
Day: <DaySum: day_of_week='Saturday', sum_tickets=489>, Median: 339.0, Stdev: 150.0, Mean: 339.0, Max tickets: 489
```

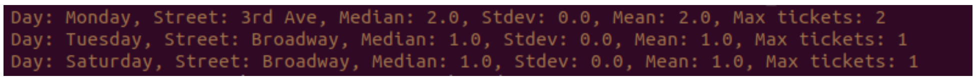
Figure 5: Rolling statistics for all data.



```
Day: Tuesday, Borough: K, Median: 87.0, Stdev: 371.62436231585605, Mean: 314.0, Max tickets: 838
Day: Tuesday, Borough: Q, Median: 18.0, Stdev: 36.24913792078372, Mean: 39.0, Max tickets: 90
Day: Tuesday, Borough: BX, Median: 10.0, Stdev: 7.586537784494029, Mean: 14.333333333333334, Max tickets: 25
Day: Tuesday, Borough: R, Median: 6.0, Stdev: 13.767917618708923, Mean: 13.666666666666666, Max tickets: 33
Day: Monday, Borough: Q, Median: 17.0, Stdev: 17.21110752456745, Mean: 22.666666666666668, Max tickets: 46
Day: Monday, Borough: BX, Median: 23.0, Stdev: 7.760297817881877, Mean: 22.666666666666668, Max tickets: 32
Day: Monday, Borough: K, Median: 50.0, Stdev: 272.54031302216964, Mean: 221.33333333333334, Max tickets: 606
Day: Monday, Borough: NY, Median: 518.0, Stdev: 430.4295012607239, Mean: 589.6666666666666, Max tickets: 1149
Day: Monday, Borough: R, Median: 5.0, Stdev: 7.2571803523590805, Mean: 8.0, Max tickets: 18
Day: Wednesday, Borough: NY, Median: 837.0, Stdev: 273.4146220587984, Mean: 643.6666666666666, Max tickets: 837
Day: Wednesday, Borough: K, Median: 60.0, Stdev: 18.856180831641264, Mean: 46.666666666666664, Max tickets: 60
Day: Wednesday, Borough: R, Median: 13.0, Stdev: 2.357022603955158, Mean: 11.333333333333334, Max tickets: 13
Day: Wednesday, Borough: Q, Median: 32.0, Stdev: 7.542472332656507, Mean: 26.666666666666668, Max tickets: 32
Day: Wednesday, Borough: BX, Median: 14.0, Stdev: 5.185449728701348, Mean: 10.333333333333334, Max tickets: 14
Day: Thursday, Borough: NY, Median: 681.0, Stdev: 284.7283305577831, Mean: 479.6666666666667, Max tickets: 681
Day: Thursday, Borough: K, Median: 146.0, Stdev: 57.03994701571484, Mean: 105.66666666666667, Max tickets: 146
Day: Thursday, Borough: R, Median: 17.0, Stdev: 6.128258770283412, Mean: 12.666666666666666, Max tickets: 17
Day: Thursday, Borough: BX, Median: 15.0, Stdev: 6.128258770283412, Mean: 10.666666666666666, Max tickets: 15
Day: Thursday, Borough: Q, Median: 34.0, Stdev: 11.785113019775793, Mean: 25.666666666666668, Max tickets: 34
Day: Friday, Borough: NY, Median: 706.0, Stdev: 223.44574285494903, Mean: 548.0, Max tickets: 706
Day: Friday, Borough: BX, Median: 17.0, Stdev: 3.7712361663282534, Mean: 14.333333333333334, Max tickets: 17
Day: Friday, Borough: Q, Median: 18.0, Stdev: 7.0710678118654755, Mean: 13.0, Max tickets: 18
Day: Friday, Borough: R, Median: 19.0, Stdev: 8.48528137423857, Mean: 13.0, Max tickets: 19
Day: Friday, Borough: K, Median: 113.0, Stdev: 48.55466564147626, Mean: 78.66666666666667, Max tickets: 113
Day: Sunday, Borough: NY, Median: 355.0, Stdev: 157.18425140226003, Mean: 439.6666666666667, Max tickets: 660
Day: Sunday, Borough: K, Median: 46.0, Stdev: 67.36138029728573, Mean: 81.66666666666667, Max tickets: 176
Day: Sunday, Borough: Q, Median: 5.0, Stdev: 12.083045973594572, Mean: 12.0, Max tickets: 29
Day: Sunday, Borough: BX, Median: 9.0, Stdev: 2.494438257849294, Mean: 9.666666666666666, Max tickets: 13
Day: Sunday, Borough: R, Median: 4.0, Stdev: 2.160246899469287, Mean: 5.0, Max tickets: 8
Day: Saturday, Borough: NY, Median: 540.0, Stdev: 178.19090885900997, Mean: 414.0, Max tickets: 540
Day: Saturday, Borough: BX, Median: 18.0, Stdev: 5.185449728701348, Mean: 14.333333333333334, Max tickets: 18
Day: Saturday, Borough: K, Median: 71.0, Stdev: 28.75567576825293, Mean: 50.666666666666664, Max tickets: 71
Day: Saturday, Borough: Q, Median: 33.0, Stdev: 13.670731102939918, Mean: 23.333333333333332, Max tickets: 33
Day: Saturday, Borough: R, Median: 16.0, Stdev: 4.714045207910316, Mean: 12.666666666666666, Max tickets: 16
```

Figure 6: Rolling statistics for boroughs.

Figure 8 shows a snapshot of percentages of tickets issued for different vehicle brands. Many times this data is missing, but after a few weeks the brands with the



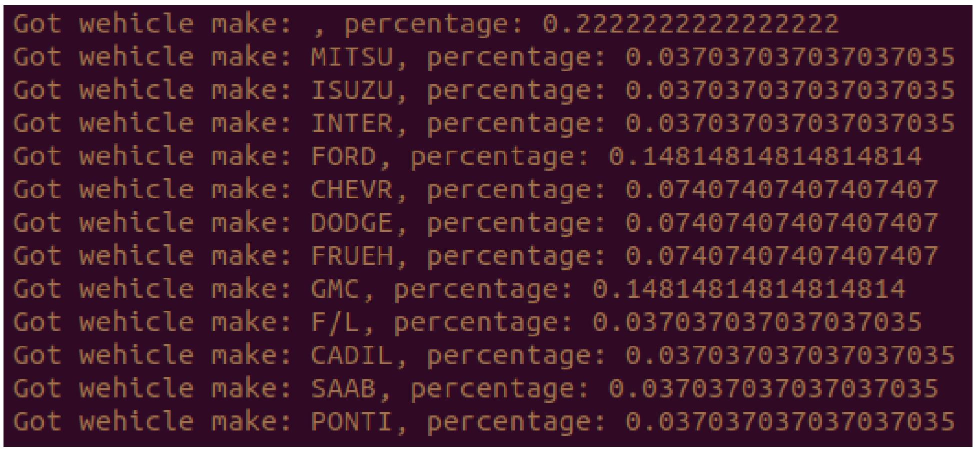
```
Day: Monday, Street: 3rd Ave, Median: 2.0, Stdev: 0.0, Mean: 2.0, Max tickets: 2
Day: Tuesday, Street: Broadway, Median: 1.0, Stdev: 0.0, Mean: 1.0, Max tickets: 1
Day: Saturday, Street: Broadway, Median: 1.0, Stdev: 0.0, Mean: 1.0, Max tickets: 1
```

Figure 7: Rolling statistics for the 10 most interesting streets.

Table 2: Summary of Silhouette Scores and Number of Clusters for first ten batches

| Batch | Silhouette score | Number of clusters |
|---|---|---|
| 1 | 0.4695 | 23 |
| 2 | 0.4671 | 15 |
| 3 | 0.5197 | 13 |
| 4 | 0.5242 | 15 |
| 5 | 0.6067 | 20 |
| 6 | 0.5178 | 17 |
| 7 | 0.5347 | 12 |
| 8 | 0.5427 | 14 |
| 9 | 0.5363 | 8 |
| 10 | 0.5713 | 18 |

largest percentage of tickets are Ford and General Motors Truck Company (GMC). The latter might be some driver of delivery or utility trucks, parking illegally for a few minutes, thinking they won't get a ticket but they do. This distribution heavily relies on the distribution of vehicle brands generally used by people in New York.



```
Got wehicle make: , percentage: 0.2222222222222222
Got wehicle make: MITSU, percentage: 0.037037037037037035
Got wehicle make: ISUZU, percentage: 0.037037037037037035
Got wehicle make: INTER, percentage: 0.037037037037037035
Got wehicle make: FORD, percentage: 0.14814814814814814
Got wehicle make: CHEVR, percentage: 0.07407407407407407
Got wehicle make: DODGE, percentage: 0.07407407407407407
Got wehicle make: FRUEH, percentage: 0.07407407407407407
Got wehicle make: GMC, percentage: 0.14814814814814814
Got wehicle make: F/L, percentage: 0.037037037037037035
Got wehicle make: CADIL, percentage: 0.037037037037037035
Got wehicle make: SAAB, percentage: 0.037037037037037035
Got wehicle make: PONTI, percentage: 0.037037037037037035
```

Figure 8: Percentage of tickets issued for different vehicle brands.

Table 2 shows the silhouette scores and numbers of clusters for the first ten batches of 1000 tickets. We can see that after 2 batches the score rises above 0.5, making it a sufficient clustering. The score then fluctuates around 0.55 but is expected to rise above 0.6 after more batches. The number of clusters is generally between 10 and 20.

## 5. T5 - Batch Data

For our machine learning task, we aimed to predict the daily number of parking tickets based on grouping the data by two different features: boroughs (Task A) and vehicle makes (Task B). The prediction task was divided into two main objectives: predicting daily ticket counts and evaluating model performance across different configurations of computational resources. We split the temporal data to train and test set with the split date being 2019-07-01.

We applied three different supervised learning algorithms, each of which supports distributed computing to some extent, to predict the daily number of parking tickets:

**a. Linear Regression (Dask-ML):** This simple linear model was implemented using Dask-ML [7], which allows for parallelized computation across multiple workers. Despite its simplicity, this model serves as a good baseline for comparison.

**b. XGBoost (Third-party Library):** XGBoost [9] was chosen for its native support for distributed computing. This model is well-suited for handling large datasets and capturing complex patterns in the data.

**c. SGD Regression (Scikit-learn with Partial Fit):** We used Stochastic Gradient Descent (SGD) from Scikit-learn [8] with the partial_fit option. This method allows for incremental learning, making it suitable for large datasets where the entire data cannot be loaded into memory at once.

**5.1. Preprocessing and Data Transformation.** We kept only the relevant columns from the original data for each task, by which we could group the data for the daily number of tickets. We chose to observe the effect of location (boroughs) on the number of daily tickets for **Task A** and the effect of vehicle brand for **Task B**. Task A required columns *Issue Date & Violation County*, Task B required *Issue Date & Vehicle Make.* We grouped the data according to the chosen columns and performed data augmentation from T2, where we gathered the data according to boroughs. Therefore, we also had to aggregate augmented data for vehicle makes, which was taking the mean or the sum of the test data for each vehicle make. In this step. we had problems with Dask groupby function because of different unannotated errors. We solved it by repartitioning to fewer partitions (10) and using the persist function. However, when using 5 workers with 5GB memory per worker, we had to repartition to more than 10 partitions, otherwise, we had memory problems. We chose 15 partitions.

Since there was some missing data in the augmented datasets (event and weather data) and we did not want to omit all rows with missing values, we replaced them with the test set mean. We transformed the data into Dask arrays, standardized it and fit the models.

**5.2. Results.** The performance of the models was evaluated based on RMSE (Root Mean Square Error), memory usage, and computation time using Arnes cluster. Table 3 presents the results for predicting daily ticket counts for Task A across different configurations of workers and memory limits. In terms of scalability, time AND memory consumption, XGBoost outperformed the other two, particularly in scenarios with limited memory per worker (0.5 GB). It managed to maintain relatively low memory usage and computation time as the number of workers increased. There was no significant difference between using different scenarios of workers and memory per worker with the same final amount of memory for our task (upper half of Table 3). In terms of scalability, using an unnecessarily large amount of workers (20) for the task proved to be time and memory-inefficient (lower half of Table 3).

Table 4 shows the RMSE scores for each model when predicting daily ticket counts for both Tasks. The results indicate that while all models performed similarly, Stochastic Gradient Descend model with partial fit had the lowest RMSE, making it the most accurate model for this task.

Additionally, feature importance was analyzed using the XGBoost model, as shown in Figure 9 for Task A. This analysis helped identify the most influential factors in predicting daily ticket counts; date-specific/temporal features being of the greatest importance. Augmented data showed significant importance as well, where dummy borough variables came last. Notably, all augmented data was borough-specific.

| **Workers** | **Mem** | **LR** | | **XGB** | | **SGD Partial Fit** | |
|---|---|---|---|---|---|---|---|
| | | **Time** | **Mem** | **Time** | **Mem** | **Time** | **Mem** |
| 3 | 9 | 96.71 s | 978.0 MB | 2.63 s | 875.0 MB | 7.67 s | 904.61 MB |
| 4 | 6 | 112.88 s | 1050.9 MB | 3.75 s | 957.4 MB | 9.51 s | 1008.5 MB |
| 5 | 5 | 81.50 s | 951.9 MB | 2.71 s | 843.5 MB | 7.16 s | 879.8 MB |
| 3 | 0.5 | 119.11 s | 884.41 MB | 15.1 s | 849.5 MB | 18.18 s | 921.45 MB |
| 10 | 0.5 | 189.8 s | 852.6 MB | 20.23 s | 795.8 MB | 19.61 s | 814.6 MB |
| 20 | 0.5 | 1267.42 s | 1227.9 MB | 59.70 s | 860.0 MB | 180.34 s | 943.7 MB |

Table 3: **ML performance of predicting the daily number of tickets.** Time to fit and predict with ML algorithms and their peak memory usage for different scenarios in Task A with varying worker counts (W) and memory limits per worker (Mem) in GB. LR - Linear Regression, XGB - XGBoost, SGD - Stochastic Gradient Descent. The first part of the table shows cases with large total memory for our task (20-30 GB), while the second part shows cases with an increasing amount of workers, where each single worker does not have enough memory to store and process the entire dataset for our task.

| **Task** | **LR** | **XGB** | **SGD Partial Fit** |
|---|---|---|---|
| **Borough** | 5270.0 | 5236.7 | **4912.4** |
| **Vehicle make** | 5788.3 | 5690.1 | **5523.2** |

Table 4: **RMSE scores**. Results of the three different algorithms across two tasks: predicting daily and hourly ticket counts.

## 6. T6 - Data Formats and Scenarios

We repeated data augmentation, EDA, and machine learning with DuckDB on parquet and Dask on HDF5 files to see the difference in computation and time resources needed for the completion of tasks. For DuckDB a database was first created from which we could query the needed data. Its creation demanded quite some memory and time (2 min 14s) but it enabled us quick access without the need for a lot of memory allocation. To compare task execution time quantitatively, Table 5 shows the time needed for the execution of two EDA tasks: computing the top 10 registration states with the most tickets and computing the average precipitation for all days. We can see that using DuckDB was considerably faster for both of the tasks. Additionally, there were no problems with memory, whereas Dask had problems with dying workers for the second task. After playing around with the number of workers and memory per

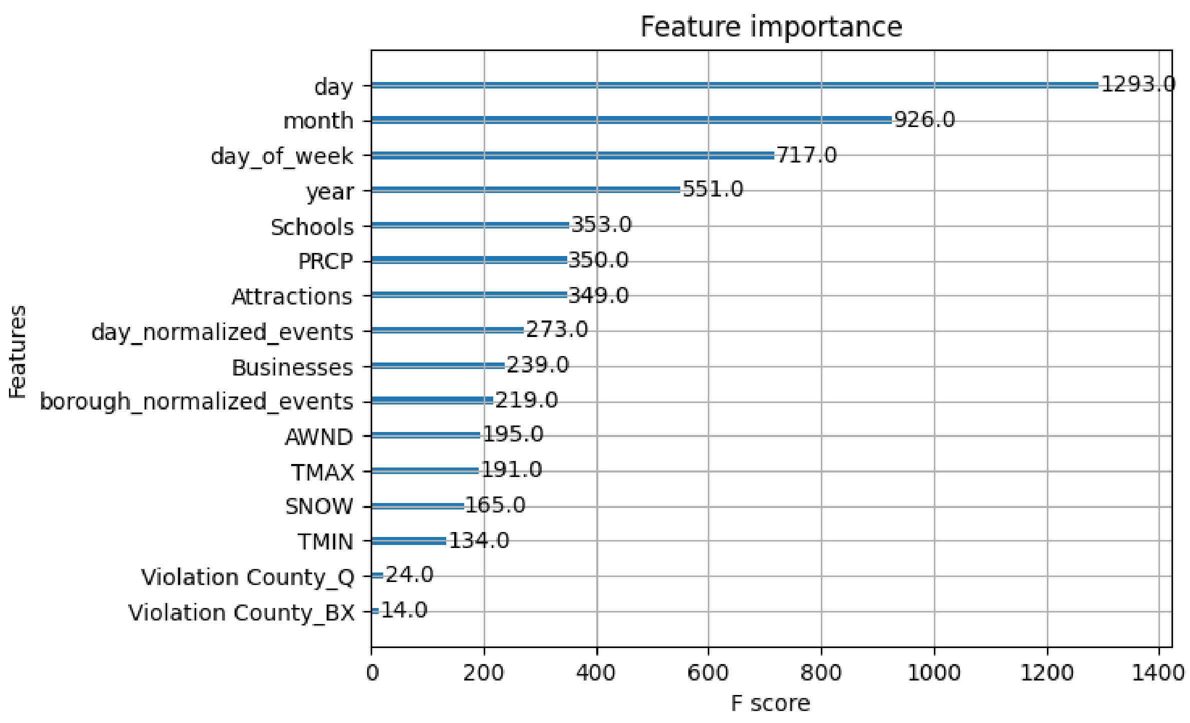


Figure 9: **Feature importance from XGBoost model.** Results shown for Task A, where grouping by boroughs was performed.

Table 5: Time needed for completion of examples from EDA tasks.

| Setup | Task | Time [s] |
|---|---|---|
| Dask + Parquet | Top 10 States | 2.176 |
| DuckDB + Parquet | | 0.411 |
| Dask + Parquet | Tickets vs Precipitation | 7.115 |
| DuckDB + Parquet | | 0.415 |

worker the task was slowly completed. On the other hand, Dask is more scalable as the creation of a database for DuckDB will not be possible if there is not enough memory. If creating smart and efficient SQL queries, DuckDB consistently outperformed Dask. It is also more user-friendly as it is more widely used. Dask lacks good documentation and annotated error messages. Running Dask on Arnes cluster yielded problems that did not occur locally, as the system behaves differently. Even if more memory was available the workers were getting killed all around. Using more memory than locally available does not make sense if we want to prove the usefulness of the method for Big Data analysis, as memory is one of the main limitations. If enough memory is available just use Pandas. Reading HDF5 files with Dask resulted in a lot of dead workers, no matter the Dask cluster configuration and we did not find a way to test it out, which did not run even on the Arnes cluster. The solution probably consisted of creating new HDF5 files with a smarter configuration.

## 7. T7 - Visualizing results on a map

We visualize streets with most tickets in Figure 3.

## 8. Conclusion

In this project, we explored various data storage and processing techniques to manage and analyze large datasets efficiently. Firstly, our takeaway regarding data storage is simpler usage and lower file sizes of parquet over HDF5 and even commonly used CSV file formats. We have never used the parquet file type before this project and have gained great confidence in it. Additionally, we explored the potential of Dask and DuckDB in different scenarios, where we show efficient distributed performance which could not be achieved with e.g. pandas library. This exploration underscores the importance of selecting the right tools for specific data science workflows.

## References

[1] 348 new york tourist locations, 2024. URL: `https://www.kaggle.com/datasets/anirudhmunnangi/348-new-york-tourist-locations`.

[2] Ncei data service api user documentation, 2024. URL: `https://www.ncei.noaa.gov/support/access-data-service-api-user-documentation`.

[3] Nyc legally operating businesses, 2024. URL: `https://data.cityofnewyork.us/Business/Legally-Operating-Businesses/w7w3-xahh/about_data`.

[4] Nyc parks event data, 2024. URL: `https://data.cityofnewyork.us/browse?Data-Collection_Data-Collection=NYC+Parks+Events&sortBy=alpha`.

[5] Nyc zipcodes, 2024. URL: `https://www.kaggle.com/datasets/arbaughp/nyc-zipcodes`.

[6] Lipsa Das. Parquet file format: The complete guide, 2023. URL: `https://coralogix.com/blog/parquet-file-format/`.

[7] Dask developers. Linearregression, 2017. URL: `https://ml.dask.org/modules/generated/dask_ml.linear_model.LinearRegression.html`.

[8] scikit-learn developers. Sgd, 2024. URL: `https://scikit-learn.org/stable/modules/sgd.html#implementation-details`.

[9] xgboost developers. Xgboost, 2022. URL: `https://xgboost.readthedocs.io/en/stable/`.