Big Data Project: New York Parking Tickets

Robert Jutreša, Luka Škodnik

Introduction

Our project tackles the given project with the given parameters, from the perspective of locations data processing. In the following tasks, we attempted to enrich the given base dataset with as much spatial information as we could, adding proximity data on points of interest to the data parking violations. We then utilize this data in the exploratory data analysis phase, and attempt to make predictions based on it.

Data

When we first did comparisons between the paqruet and HDF5 file formats, we observed that the HDF5 files formats were the smallest. This is displayed in Table 1, where we only converted from CSV to one of the target formats. For saving to HDF5, we prepared the data such that we only store an identifier for each of the values in the non-numeric columns. However, we found that these were not necessarily unique which made overall usage of the files more difficutl. Thus we had to use variable length strings in the future.

Formats

File Name	CSV Size	Parquet Size	HDF5 Size
2014.csv	1710.72	330.06	184.14
2015.csv	2393.49	511.80	318.64
2016.csv	1971.17	352.62	192.56
2017.csv	1990.24	519.51	351.12
2018.csv	2073.71	401.29	232.82
2019.csv	1910.91	364.27	211.75
2020.csv	2214.43	382.20	216.46
2021.csv	2618.82	445.62	252.72
2022.csv	2645.07	418.49	218.41
2023.csv	3839.23	617.47	310.59
2024.csv	2645.07	418.49	216.27

Table 1. Comparison of file sizes across different formats in MB.

Additional Sources

Firstly we enrich our data with the addition of daily weather data. New York is not so geographicaly big that there should be significant deviations of the reported weather across it, therefore we simplify our task by including the same weather information regardless of location. The daily weather data was obtained with the help of Visual Crossing API [1] and was merged with the ticket data based on time.

We further augmented our dataset with information about middle [2] and high schools [3], whose data we obtain from the NYC Open Data website [4]. These datasets contain a lot of information we are not interested in so we decided to only include the name and distance of the closest middle and high school to where the ticket was issued. With the same logic, we also aggregate data about businesses and [5] landmarks [6, 7]. We initialy also tried to aggregate event data [8], but in the end omitted it since it proved to be a much harder challenge because of the lack of unified information about the event location.

In order to merge the above mentioned data, we had to obtain spatial coordinates of each issued parking ticket. We did this utilizing the four columns that give us spatial information: Violation County (which had to be unified), Street Code1, Street Code2, and Street Code3. According to the NYC Planning GeoSupport [9] documentation, a true NYC street code is constructed from the Borough Code (for us Violation County) followed by the 5bit street code (given to us in three columns). For this reason, we constructed a street code to spatial coordinates lookup table from the Street Name Dictionary from NYC Open Data [10] using the Nominatim API. We then used this local table, to get the approximate spatial location of each violation, by computing the average of the longitudes and latitudes of given street codes.
The joining process of the main data with the additional data was difficult due to the nature of the joining procedure. For each data entry we had to compute the closest entity of interest for each of the 5 location based datasets we had. To do this as efficiently as possible, we utilized the RTree data structure [11]. We pre-computed the RTree index for each of the location based datasets, and then for each entry fetched the closes entity based on the computed latitude and longitude from the step prior. Additionally, we also computed the distance between the two locations using the Haversine formula [12]. Merging the businesses required some addi-

tional work to merge, as we had to limit the RTree query results to those that were legally operational during the time the ticket was issued, if no such result was found, we recursively expanded the search depth. With these data, we then formed a smaller data frame, which was joined with the tickets data. Due to the row-by-row computations this required, it took some time, as seen in Table 2 and produced files with combined sizes as seen in Table 3. We have two remarks here:

1. Processing data in an HDF5 file (using dask) is faster than processing that same data stored in a parquet file. However, due to the slow I/O operations of the selected HDF5 library, these took a lot longer to fully process. In reality processing was about 50% faster than that same processing of parquet files.

2. The file sizes in Table 3 can be misleading. During our testing we have proven, that HDF5 files can, if structured properly, be of smaller size than parquet files. This can be achieved, by factorizing the categorical data, and potentially storing the mappings in a different file, shortening the string data via acronyms or some other method. As it stands, the biggest reason for the giant file sizes, was the use of a generic variable size byte array data type to store them.

Table 2. Data Merging Time.

Dataset	Parquet [min]	HDF5 [min]
Weather	2.77	133.29
Middle Schools	102.82	208.61
High Schools	118.64	224.87
Individual Landmarks	102.75	218.64
Scenic Landmarks	57.92	189.38
Businesses	2012.58	1607.84

Table 3. Final Combined Data Sizes.

Format	Size [GB]
CSV	16.33
parquet	4.6
HDF5	71

Exploratory Data Analysis

It is important to mention, that after initial testing, we came to the conclusion that using large enough subsets of data is sufficient to get a representative sample. To more efficiently produce the exploratory data analysis, we thus utilized a 2% sub-sample of the data.

We begin our exploratory data analysis by examining the distribution of parking ticket locations. Figure 1 presents the density of parking tickets across New York. The contours indicate that Manhattan is the most significant hotspot for parking tickets in terms of density, while Staten Island has the lowest density. The other boroughs show similar levels of density. This result makes sense as Manhattan, despite being geographically smaller, is the most densely populated and the most developed of the 5 New York boroughs.

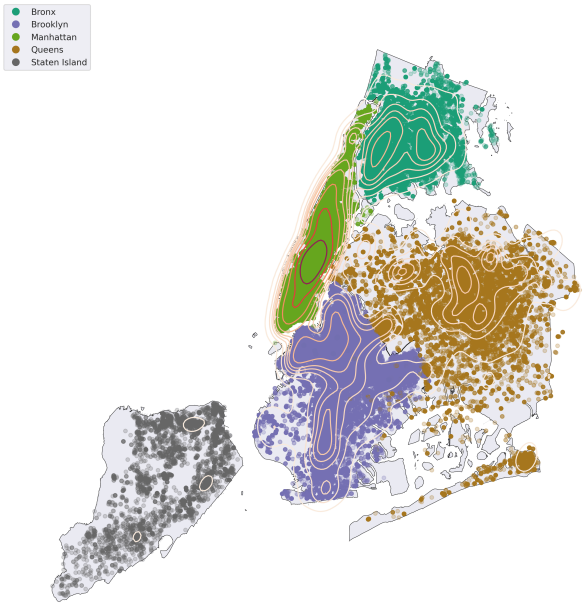

Figure 1. Density of Parking Tickets in New York.

Geographic Entities and Parking Tickets

Next, we explore the relationship between various geographic entities in New York and parking tickets. These entities, shown in Figure 2, include high schools, middle schools, scenic landmarks, individual landmarks, and legally operating businesses. Businesses are evenly distributed across New York, while landmarks are more concentrated in Manhattan. Schools are also spread evenly, although Manhattan has a slightly higher density, and Staten Island has fewer schools. This distribution may be due to population density or other factors.

To determine the most impactful entities, we plot the top 10 entities of each category based on their proximity to issued tickets. As seen in Figure 3, Manhattan shows a high concentration of these entities. Given its smaller size and high density of various entities, this result is not surprising.

We also examine the distribution of distances between entities and the nearest parking tickets, as shown in Figure 4. Notably, businesses are the closest entities to issued tickets, likely due to their overall density. Businesses are followed by individual landmarks, middle and high schools, and lastly scenic landmarks. The sparse distribution of scenic landmarks explains their larger distances from parking tickets. While we gain insights into the proximity of various entities to parking tickets, we lack data to comprehensively explain why some entities are closer than others—whether due to geographical distribu-

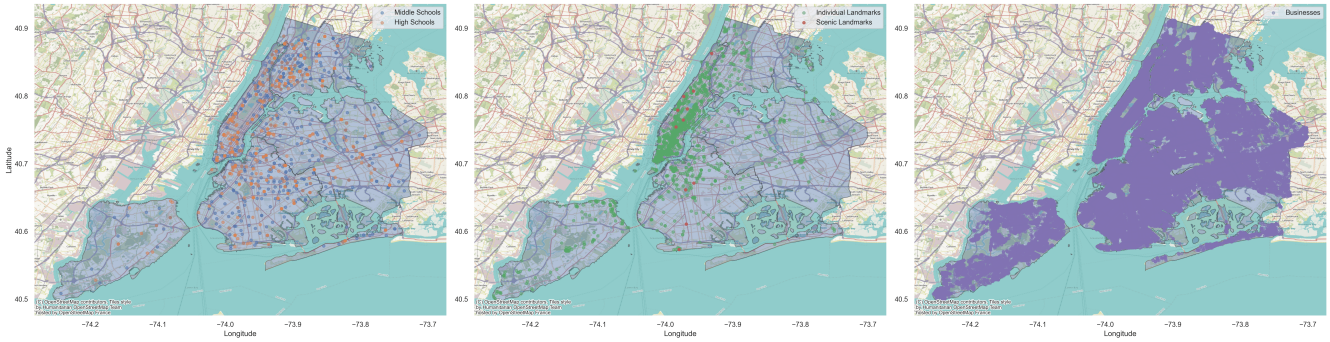

Figure 2. Locations of Various Entities in New York.

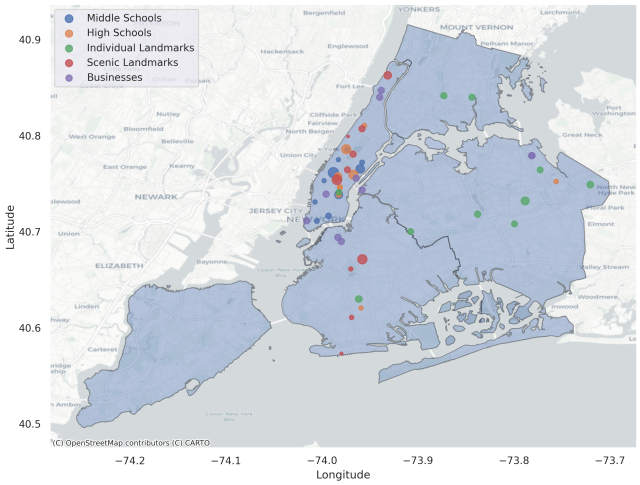

Figure 3. Top 10 Entities by Proximity to Parking Tickets.

tion or other underlying reasons. One thing we can reason upon further inspection of the data, is that many of the closest businesses to the locations of the parking tickets are listed as parking meter authorities, showing, that most likely these violations are because of expired or unpaid parking meters.

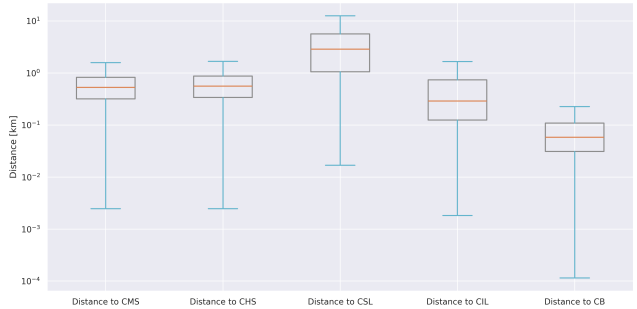

Figure 4. Distances Between Entities and Closest Parking Tickets.

Weather Impact on Parking Tickets

Given that weather affects driving behavior, we also examine its impact on parking violations. Figure 5 shows that most tickets are issued on days with ”Snow, Rain” or just ”Rain” forecasts, which makes sense as adverse weather conditions likely lead to more parking violations. Interestingly, ”Clear” weather also ranks high, indicating that poor parking behavior is not solely dependent on weather conditions.

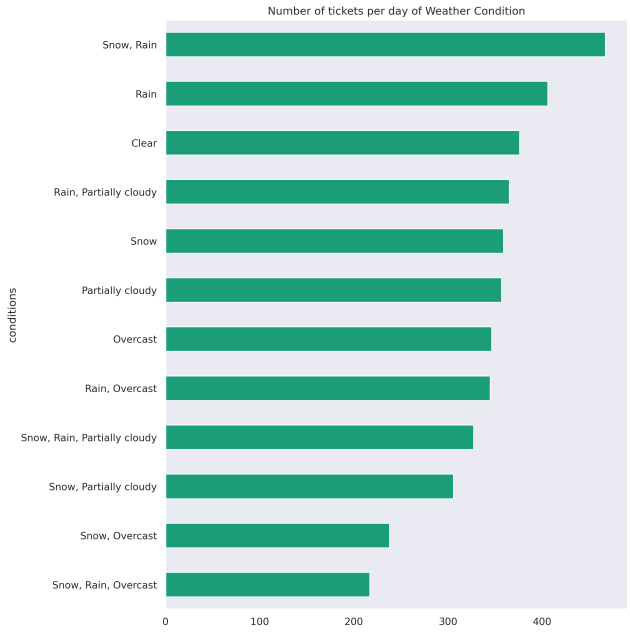

Figure 5. Parking Tickets Issued by Weather Condition.

Violations by Borough

We then investigate the violation time of tickets per borough, as seen in Figure 6. Despite having the lowest density and number of tickets, Staten Island shows the longest violation times. This might be due to a higher number of business vehicles that are not driven regularly, leading to fewer but longer violations. For other boroughs, the violation times correspond to the density of parking tickets issued.

We also analyze the most prevalent car makes among violators in each borough, shown in Figure 7. There are some differences in the most prevalent makes, which might indicate brand preferences among violators in different boroughs. However, the top makes remain consistent across all boroughs. It is unclear if these brands are common among violators or

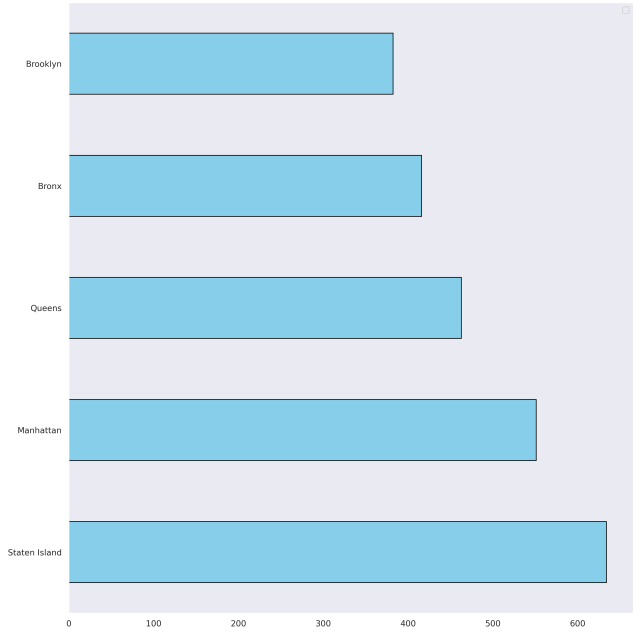

Figure 6. Violation Times of Parking Tickets by Borough.

the general population, as we lack the data to determine this.

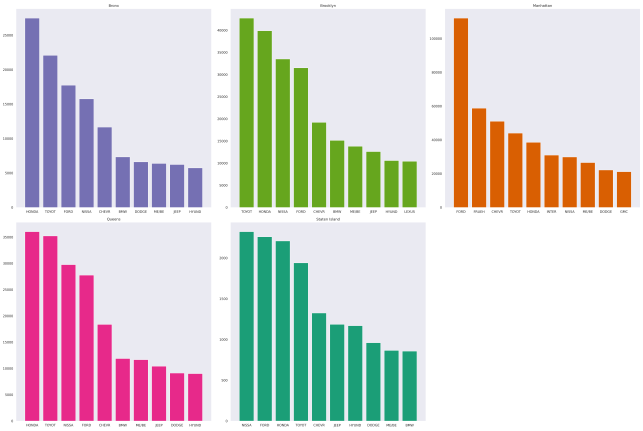

Figure 7. Prevalent Car Makes Among Parking Violators by Borough.

Comparison by Format and Method

We prepared the plots with 3 different methods where applicable: with dask on parquet files, with dask on HDF5 files, and with dask and duckdb on parquet files. The resulting times taken to plot, are shown in Table 4. We can see that duckdb was much faster in every aspect. For the N/A values, we implemented the first plot in geopandas, which was required to ensure the desired result, thus implementation via duckdb is not possible. For the second one, we just didn't think it was efficient to hand compute all the boxplot values. Overall we get a different picture on the parquet - HDF5 dynamic, as we can see much longer computational times for HDF5 compared to what we previously saw.

Streaming

Streaming data analysis enables real-time processing of continuously flowing data, facilitating immediate insights and actions. This project employs Apache Kafka as a message broker to manage the data stream, using custom producers and consumers to handle data flow and processing. This section details the setup, implementation, and functionality of the streaming components in our project.

Setup and Implementation

Since we are running this code locally, we use a single broker for our Kafka cluster along with Zookeeper for metadata management. A custom producer reads data from a .parquet file and sends it as JSON-encoded strings to a topic designed for ingesting raw data. Our Faust application consumes this data and processes batches of 100 values continuously. Larger window sizes complicate processing, leading to errors and data loss. We attempted to mitigate this by running more complex tasks in a separate thread, which helped to some extent, but the issue persists if processing takes too long. After processing the buffered values, we send the processed statistics to a separate topic. For convenience, we also save the processed data of each batch in a file.

Rolling Statistics

For our data, the rolling statistics are not particularly insightful. Most columns lack numerical data of sufficient granularity to make rolling descriptive statistics such as mean, median, and standard deviation interesting. For example, aggregated daily weather data for New York yields repetitive statistics across 100 values. Similarly, rolling frequency plots of categorical columns are often uninformative.

Despite this, we need to present five or more rolling statistics for all data, data split by boroughs, and per the most interesting streets. Thus, we calculate rolling statistics for the vehicle year column, including mean, standard deviation, median, 25th percentile, 75th percentile, minimum, and maximum statistics. Results for fiscal year 2024 are shown in Figure 8. We present a subsample of the mean and percentiles for clarity. Although individual values do not show clear results, the fitted line indicates an upward trend in all statistics, suggesting that people tend to drive newer cars over time. Interestingly, at any given time, people tend to drive cars that are around eight years old. The calculated statistics per borough and the most interesting streets show the same conclusions, so their plots are not included, as the existing plot effectively conveys the point.

Clustering

As described in [13], various stream clustering approaches exist. Due to issues with drift and data loss, we chose a simple and fast algorithm adapted from classic k-means, a partition-based stream clustering method. This algorithm splits data into a predefined number of clusters based on distance to the cluster centroids, which are continuously updated as new

Table 4. Data Merging Time.

Plot	Dask-Parquet [min]	Dask-HDF5 [min]	DuckDB-Parquet [min]
Location Density [geopandas]	192.95	736.85	N/A
Distances Statistics	16.62	138.86	N/A
Car Make Per Borough	13.94	137.61	6.76
Interesting PoTs	130.24	674.32	28.13
Violations Per Weather Condition	26.08	138.32	5.872
Average Violation Time per Borough	26.08	191.61	0.78

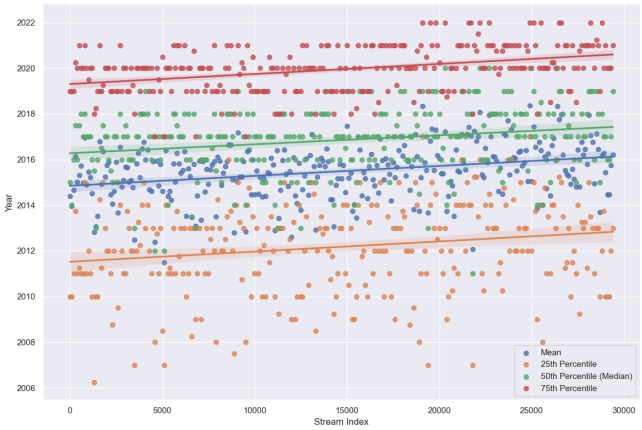

Figure 8. Rolling Statistics for Vehicle Year (Fiscal Year 2024).

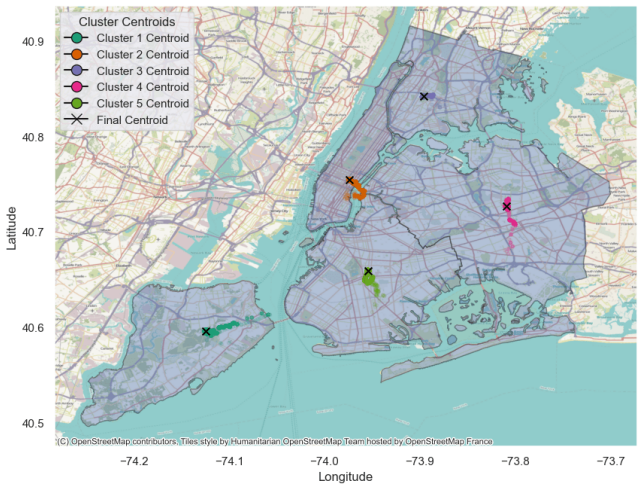

Figure 9. Cluster Centroids for Parking Tickets (Fiscal Year 2024).

data arrives. In our case, we perform spatial clustering based on geographical distance, which is interpretable since we can track centroid changes over time. We use five clusters and update the centroids for batches of 100 data points due to performance issues with larger batches and more clusters. The results, shown in Figure 9, indicate that the cluster centroids fall into separate boroughs and move within them.

Defining a higher number of clusters could potentially provide more detailed hotspots for parking tickets, allowing for a finer-grained analysis of parking violations. This could be particularly useful for identifying specific problem areas within boroughs, offering targeted insights for traffic management and law enforcement. However, increasing the number of clusters would also demand more computational resources and could complicate the interpretation of the results, particularly if the clusters become too small or numerous.

Machine Learning

Machine learning offers powerful techniques for predicting patterns and trends in data. In this section, we focus on predicting days with a high number of ticket submissions. Accurate predictions can help in resource allocation and improve service management. We model this prediction problem using two approaches: regression and classification.

To prepare our data for these tasks, we transform it to a daily granularity. This involves incorporating the most common values of selected categorical columns and the means of numerical columns. For regression, we predict the number of tickets received each day as well as the average daily distance to the closest high school. For classification, we determine whether a given day had more or fewer than 20,000 tickets, which is approximately the daily mean.

The results of these models, using various algorithms and tools, are presented in Tables 5, 6 and 7. We observe only minor differences between the tools and algorithms. We speculate that utilizing different data augmentations and more complex learning problems would yield more significant differences, where some tools would clearly outperform others.

Format and tool comparison

We utilized three different learners: dask-ml Linear and Logistic Regression, the XGBRegressor and XBGClassifier from the xgboost library, and the MLPRegressor and MLPClassifier which utilize the partial fit method from sklearn. We established a memory limit of 8 GB per worker, and from the results displayed in Tables 5, 6 and 7 we came to the following findings:

- On all of our test cases (with the exception of when working with the HDF5 file format), xgboost outperformed other used methods.
- Again processing on the HDF5 files took much longer

Table 5. Machine Learning Performances - Daily Tickets Prediction.

Workers	Memory [GiB]	File Format	ML Method	N Tasks	Execution Time [sec]	RMSE
8	59.60	parquet	dask-ml	178	60.73	8265.30
16	119.21	parquet	dask-ml	176	59.03	8265.30
8	59.60	parquet	xgboost	128	60.26	4576.61
16	119.21	parquet	xgboost	128	59.16	4576.61
8	59.60	parquet	sklearn	714	64.60	9713.65
16	119.21	parquet	sklearn	714	63.81	9713.65

Table 6. Machine Learning Performances - Prediction of Distance to Closest High School.

Workers	Memory [GiB]	File Format	ML Method	N Tasks	Execution Time [sec]	RMSE
16	119.21	parquet	dask-ml	142	58.16	0.88
16	119.21	parquet	xgboost	128	58.17	0.42
16	119.21	parquet	sklearn	714	64.87	17.49
16	119.21	HDF5	dask-ml	102	328.97	1.96
16	119.21	HDF5	xgboost	88	326.38	34.40
16	119.21	HDF5	sklearn	388	332.36	33.73

Table 7. Machine Learning Performances - Classification of Over/Under 20.000 Tickets.

Workers	Memory [GiB]	File Format	ML Method	N Tasks	Execution Time [sec]	Acc	Log Loss
16	119.21	parquet	dask-ml	146	61.91	0.801	0.84
16	119.21	parquet	xgboost	128	62.23	0.918	0.34
16	119.21	parquet	sklearn	714	67.47	0.487	1.84

than the parquet files. Leading us to drop our initial conclusions on the processing of said file format being faster, as we saw during the augmentation step.

- For this task, we didn't use DuckDB, as we didn't think it made a lot of sense to have distributed computing, where we have to load the entire dataset into memory prior to performing it. The reason for this is because we couldn't find a true integration between DuckDB and dask-s lazy loading. Thus where we used DuckDB, we read the files via dask, and then computed them into pandas data frames where necessary.

- The performance of sklearn's general MLP, which enables partial fit, is lackluster compared to the other distributed methods tested. This holds for both the regression problems as well as the classification task.

- Generally we tried to limit ourselves to mostly using the augmented features of the dataset, being the ones we constructed from additional data, as we didn't seem to find attributes with good informative values in the original dataset. Perhaps additional feature construction or processing of the primary features, would have improved the performance of used models.

- It is interesting to note, that overall the HDF5 tests had to perform fewer tasks than the ones utilizing parquet files. We measured this via the `dask.distributed.performance_report` function, which gave us some insights into how the code was executed. We aren't able to fit it into the table, but HDF5 processing utilized more system resources including CPU, memory, bandwidth and number of file descriptors. This is consistent between the used machine learning methods, with the only exception being that sklearns general MLP utilized more CPU power when processing parquet files.

- Utilizing a larger number of workers, in our case didn't lead to major performance improvements, both in terms of computational time and performance.

Discussion

Given the limitations of our work, we are still able to find that the HDF5 file type doesn't seem effective for large scale processing. The seemingly long I/O operational times, limits it's scalability as larger files take much longer to both read and write. In addition, the original thought that this might be a trade off for faster computational time, was disproven by both the EDA and ML tests.

While the data itself was difficult to work with, we were able to enrich it with locational data, to such a degree that we were able to clearly visualize some trends regarding the issuing of parking violation tickets on the map of New York City. We know, that these are most common around legally operating businesses (in most situation Parking Meter Authorities), and

least common in areas with scenic landmarks (due to their low number and concentration in only a few areas of New York). We can also see, that the density of points of interests, seems to mirror that of the density of issued parking violations, where the majority of schools and individual landmarks, as well as issued tickets, are concentrated in Manhattan.

Machine learning can be a difficult task when using unregulated data as we had here. With various entries written differently, but meaning the same thing, some data not being in correct formats (e.g. "1700" value given in column "To Hours in Effect", with a format of `%I%M %p`), it is difficult to properly compute predictions. Given a high number of categorical data, that fits the previously listed, we were limited to the attributes that we were able to utilize. Despite this, we were able to achieve some good results, especially in the areas of regression of distance to closest high school and classification of over / under 20000 daily tickets.

If we are looking for computational efficiency for distributed or parallel computing, it is hard to argue that utilizing dask or xgboost with the parquet file format isn't the correct choice. Most of the computational time in these cases was being used for preprocessing the data in order to prepare it for the models. If memory limitations aren't an issue, we assume that utilizing DuckDB to perform these tasks would be the most optimal solution, as it proved to be the fastest when used for exploratory data analysis. Finally, HDF5 seems to utilize the most resources from every aspect in all scenarios. Perhaps if properly prepared, such that we would not use variable byte length arrays to store strings, but instead store them in factorized or One-Hot encoded format, these results would be different.

References

[1] Visual crossing. https://www.visualcrossing.com/. Accessed: 2024-08-15.

[2] Doe middle school directory. https://data.cityofnewyork.us/Education/2021-DOE-Middle-School-Directory/f6s7-vytj/about_data. Accessed: 2024-08-15.

[3] Doe high school directory. https://data.cityofnewyork.us/Education/2021-DOE-High-School-Directory/8b6c-7uty/about_data. Accessed: 2024-08-15.

[4] Nyc open data. https://opendata.cityofnewyork.us/. Accessed: 2024-08-15.

[5] Legally operating businesses. https://data.cityofnewyork.us/Business/Legally-Operating-Businesses/w7w3-xahh/about_data. Accessed: 2024-08-15.

[6] Scenic landmarks. https://data.cityofnewyork.us/Housing-Development/Scenic-Landmarks-Map-/gi7d-8gt5. Accessed: 2024-08-15.

[7] Individual landmark sites. https://data.cityofnewyork.us/Housing-Development/Individual-Landmark-Sites-Map-/ts56-fkf5. Accessed: 2024-08-15.

[8] Nyc permitted event information. https://data.cityofnewyork.us/City-Government/NYC-Permitted-Event-Information-Historical/bkfu-528j/about_data. Accessed: 2024-08-15.

[9] Nyc planning: Geosupport. https://nycplanning.github.io/Geosupport-UPG/#. Accessed: 2024-08-15.

[10] Street name dictionary. https://data.cityofnewyork.us/City-Government/Street-Name-Dictionary/w4v2-rv6b/about_data. Accessed: 2024-08-15.

[11] Rtree: Spatial indexing for python. https://rtree.readthedocs.io/en/latest/. Accessed: 2024-08-15.

[12] Haversine formula. https://en.wikipedia.org/wiki/Haversine_formula. Accessed: 2024-08-15.

[13] Alaettin Zubaroğlu and Volkan Atalay. Data stream clustering: a review. *Artificial Intelligence Review*, 54(2):1201–1236, 2021.