Big Data: Project
NYC Violation Tickets

Blaž Pridgar, Drejc Pesjak
63220482, 63180224

August 16, 2024

1 Introduction

This report presents an in-depth analysis of New York City's parking violations data from 2014 to 2024. This study is part of the Big Data Project 2023/24. Using the CRISP-DM methodology, we followed a structured approach to analyze the data with various tools in the Python data science ecosystem. Our analysis involved several key steps: importing and storing data efficiently, adding extra contextual information, performing exploratory data analysis, conducting streaming data analysis, using machine learning for batch data analysis, and comparing different data processing methods. The goal of this study is to use advanced data science techniques to uncover insights into parking violations in New York City and demonstrate the practical application of big data methods.

2 Data

The dataset used in this project consists of New York City's parking violation tickets from 2014 to 2024. This data was sourced from the NYC OpenData platform and includes detailed records of each violation.

2.1 Data Characteristics

The dataset includes fields relevant to our analysis, with a focus on spatial and temporal attributes:

- **Spatial Data**: Violation location details including street names, street codes and boroughs. The street codes are supposed to be GIS coordinates of the location where the violation was issued and intersecting street boundaries, but we found no efficient way to transform these into actual longitude and latitude coordinates. We were left with the street names and boroughs which are far less accurate but it was enough to break the analysis into more regions of interest like busiest streets and all the boroughs.

- **Temporal Data**: Issuance dates and times, which makes this a temporal dataset. The EDA later reveals some temporal trends that were useful in the machine learning part of the study. The temporal structure was also useful when emulating the streaming task.
- **Violation Details**: These include information about the type of violation, the issuer of the violation ect. All of these would be viable target variables in the prediction stage of this task.
- **Vehicle Information**: Information about the vehicle responsible for the violation including the registration state, make year and plate details. These are again viable targets for any prediction tasks.

The dataset covers a wide range of violations, predominantly parking-related, providing a comprehensive view of parking enforcement in NYC over a decade.

2.2 Data Quality and Issues

During our exploratory data analysis, we identified several data quality issues:

- **Missing Values**: Some fields, such as "No Standing or Stopping Violation", "Hydrant Violation", and "Double Parking Violation", were completely empty.
- **Erroneous Data**: Notable errors included vehicle years set in the future.
- **Miscellaneous Data**: This was especially noticeable with the boroughs where many different notations where used for a single borough (for example Brooklyn was marked as "K", "BK" and "KINGS").

Addressing these issues is crucial for accurate analysis and modeling. Because of the abundant amount of data-points we simply cleaned the data by removing rows with missing data or obvious errors (like years above 2024) and grouped the different notations for single boroughs together.

2.3 Original Data Format

The original dataset comprises 11 CSV files, each representing a single year of data. The total size of these CSV files is 26.9GB, with records up to April 2024. Each file contains numerous fields, including but not limited to:

- **Summons Number**: Unique identifier for each ticket.

- **Plate ID**: Vehicle's license plate number.
- **Registration State**: State where the vehicle is registered.
- **Plate Type**: Type of license plate.
- **Issue Date**: Date the ticket was issued.
- **Violation Code**: Code representing the type of violation.
- **Vehicle Make and Year**: Manufacturer and year of the vehicle.
- **Violation Location and Time**: Location and time of the violation.

2.4 Data Conversion

To facilitate efficient data processing and storage, we converted the CSV files into Parquet and HDF5 formats using distributed Dask on the HPC Arnes cluster.

Parquet Conversion The conversion to Parquet format, which uses snappy compression, resulted in 415 Parquet files totaling 5.8GB. Parquet was chosen for its columnar storage format, which provides significant storage compression and faster query performance. This format is highly efficient for read-heavy workloads, making it suitable for our analysis needs.

HDF5 Conversion We also converted the data to HDF5 format, resulting in 152 HDF5 files totaling 32.2GB. While HDF5 is designed for storing complex hierarchical data and offers excellent performance for certain types of operations, it proved less efficient for our tabular parking violation data due to the larger file sizes and minimal performance gains in our specific use case.

2.5 Scope of Analysis

Our analysis focuses on:

- **Full Dataset (2014-2024)**: To understand overall trends.
- **Borough-Specific Analysis**: To identify geographical variations specific to different boroughs of the New York City.
- **Key Streets**: Analysis of streets with the highest violation rates.

By leveraging the comprehensive dataset and addressing its quality issues, we aim to extract valuable insights into parking violations in New York City.

3 Additional Data

To enhance our analysis of NYC parking violation tickets, we integrated several supplementary datasets. These datasets provide context and additional variables that could influence parking violations, such as weather conditions, events, school locations, businesses, landmarks, and holidays.

3.1 Weather Data

We acquired hourly weather data for NYC from 2013 to 2024 using the VisualCrossing Weather API. Due to the limitations of the free tier, we opted for the professional plan at $35 per month, allowing us to download up to 10 million records per month. This dataset includes weather conditions recorded hourly at the Central Park weather station and comprises 12 CSV files totaling 99,299 lines or 18.5MB. The data includes key weather parameters such as temperature, humidity, precipitation, wind speed, and visibility. This comprehensive weather information helps us assess the impact of weather conditions on parking violations.

3.2 Event Data

We included data on special events and construction permits from the NYC OpenData website. This dataset contains information on approved event applications from 2008 to 2024, capturing events that impact traffic, such as street closures and large public gatherings. By analyzing this data, we aim to identify correlations between events and spikes in parking violations, providing insights into how public events affect parking behavior.

3.3 School Locations

School location data for the 2019-2020 academic year was sourced from the NYC Department of Education. This dataset includes detailed information on school locations and their administrative details. Understanding the distribution of schools helps us investigate whether there are higher violation rates near schools, possibly due to drop-off and pick-up activities.

3.4 Businesses Data

We integrated data on legally operating businesses in NYC, provided by the Department of Consumer and Worker Protection (DCWP). This dataset includes licenses issued to businesses, offering a view of commercial activity that may affect parking patterns. By correlating business density with parking violations, we can assess how commercial zones contribute to parking congestion and enforcement.

3.5 Landmarks Data

Data on individual landmark sites was obtained from the NYC Landmarks Preservation Commission (LPC). This dataset includes site boundaries and administrative information for all designated individual landmarks. Landmarks often attract visitors, and understanding their locations allows us to analyze whether these sites have higher parking violation rates due to increased tourist activity.

3.6 Holiday Data

We compiled a comprehensive list of holidays affecting NYC, including national, NYC-specific, general, religious, and school holidays. These holidays were categorized into:

- National Holidays
- Religious Holidays
- Special Days
- School Holidays

This categorization helps us identify potential trends in parking violations during holiday periods when parking regulations might be more frequently ignored or misunderstood.

3.7 Data Integration

The additional datasets were integrated with the primary parking violation data to enrich our analysis. We aligned each supplementary dataset with the parking violation records based on temporal and spatial attributes (borough), enabling us to explore correlations and build more accurate predictive models.

By incorporating these diverse data sources, we aim to provide a more comprehensive understanding of the factors influencing parking violations in New York City.

4 Exploratory Data Analysis

4.1 Vehicle Year Distribution

The vehicle year distribution is illustrated in Figure 1. The distribution resembles a Gaussian bell curve, with a significant spike at zero, indicating missing values. Anomalies are observed for vehicle years beyond 2024, suggesting entry errors.

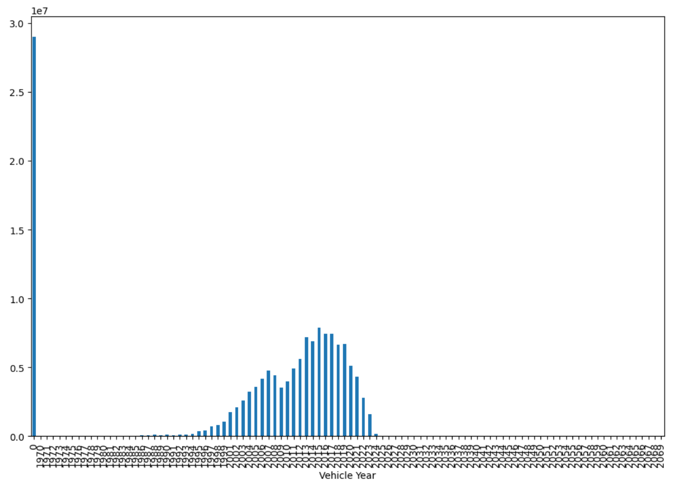


Figure 1: Vehicle Year Distribution. The spike at zero represents missing values, and anomalies are seen for years beyond 2024.

4.2 Monthly Violations Over Time

Figure 2 shows the monthly violations over the full time span. Notably, there are zero violations from July 2019 to April 2020, corresponding to the COVID-19 pandemic lockdown period. Additionally, there is an unexplained spike in violations during August and September 2023.

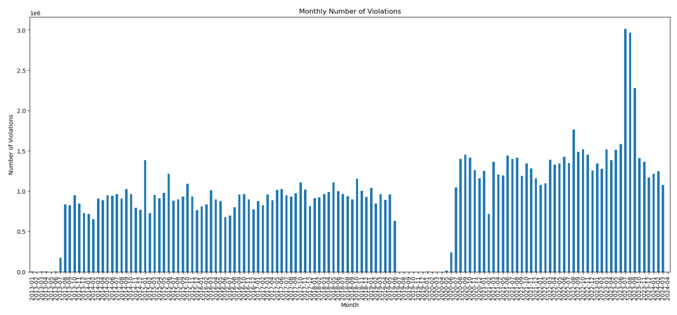


Figure 2: Monthly Violations Over Time. Zero violations are observed during the COVID-19 lockdown period from July 2019 to April 2020, and a spike in August and September 2023.

4.3 Top 10 Streets with Most Violations

The analysis identified the top 10 streets with the highest number of violations, depicted in Figure 3. These streets include Broadway, 3rd Ave, 5th Ave, Madison Ave, Lexington Ave, 2nd Ave, 1st Ave, Queens Blvd, 8th

Ave, and 7th Ave. This insight highlights areas with significant parking enforcement activities.

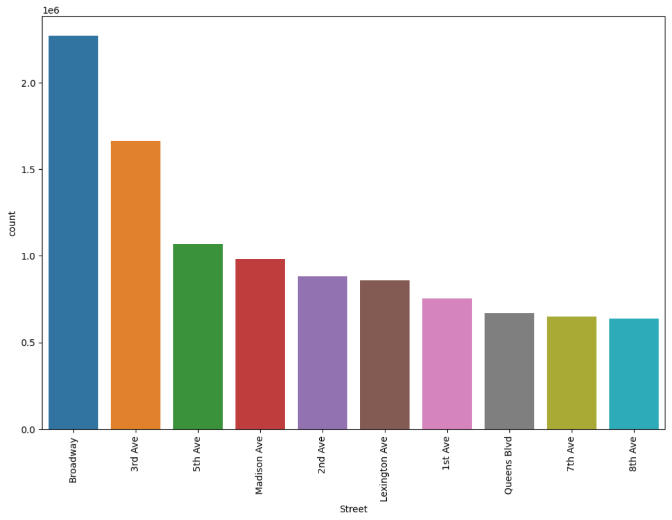


Figure 3: Top 10 Streets with Most Violations. The streets listed have the highest number of parking violations, indicating hotspots for parking enforcement.

4.4 Geographical Distribution of Violations

The geographical distribution of violations across NYC counties is shown in Figure 4. This map highlights the different boroughs with their average monthly density of parking violations. We created an even better representation of the distribution of violations in all the boroughs through time in a form of a GIF that can be found on our GitHub repository.

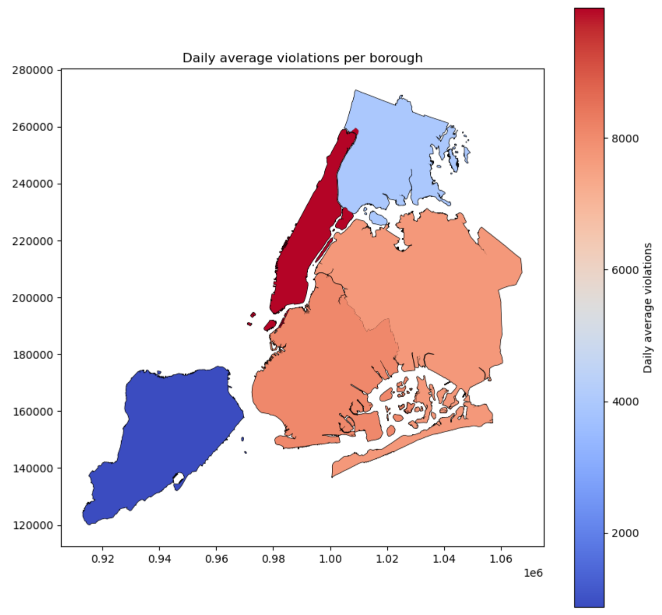


Figure 4: Distribution of Violations with respect to all the boroughs of NYC. The map displays the average daily density of parking violations in each borough.

5 Kafka streaming

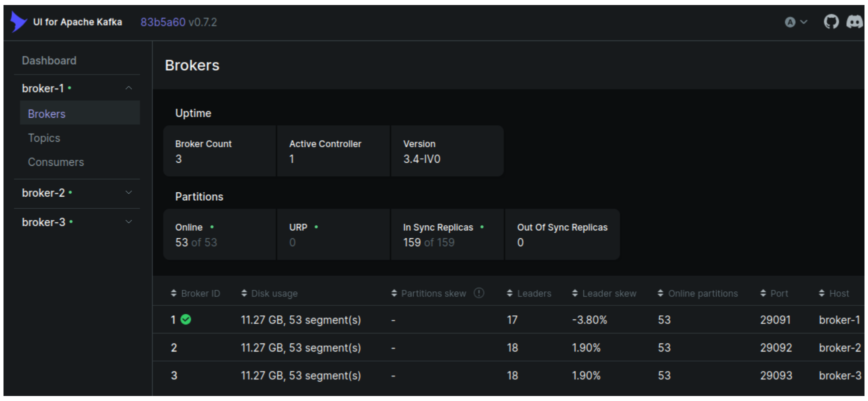
For real-time processing and analysis of NYC parking violation tickets, we set up a Kafka streaming architecture. This setup includes three Confluent Kafka brokers with replication factor of 3 and a retention policy of 5 minutes or 5GB. The brokers are managed by Confluent Zookeeper, ensuring coordination and synchronization across the cluster. Additionally, we utilized Kafka UI from ProvectusLabs for monitoring and management.

5.1 Kafka Setup

We configured three Kafka brokers to ensure high availability and fault tolerance. Each broker is set up with the following parameters:

- **Replication Factor**: 3
- **Retention Rate**: 5 minutes or 5GB

The Kafka UI from ProvectusLabs provides a user-friendly interface for monitoring the Kafka cluster, making it easier to track topic metrics, broker statuses, and consumer lag (see Figures 5 and 6).



Brokers

Uptime

Broker Count	Active Controller	Version
3	1	3.4-IV0

Partitions

Online	URP	In Sync Replicas	Out Of Sync Replicas
53 of 53	0	159 of 159	0

Broker ID	Disk usage	Partitions skew	Leaders	Leader skew	Online partitions	Port	Host
1 ✓	11.27 GB, 53 segment(s)	-	17	-3.80%	53	29091	broker-1
2	11.27 GB, 53 segment(s)	-	18	1.90%	53	29092	broker-2
3	11.27 GB, 53 segment(s)	-	18	1.90%	53	29093	broker-3

Figure 5: UI for Apache Kafka showing 3 brokers. Each broker's disk usage is at 11.27 GB, indicating data replication across the cluster, where 3 replicas contribute to the total disk space usage.

5.2 Producer and Consumer Configuration

The Kafka streaming architecture includes one producer and multiple consumers:

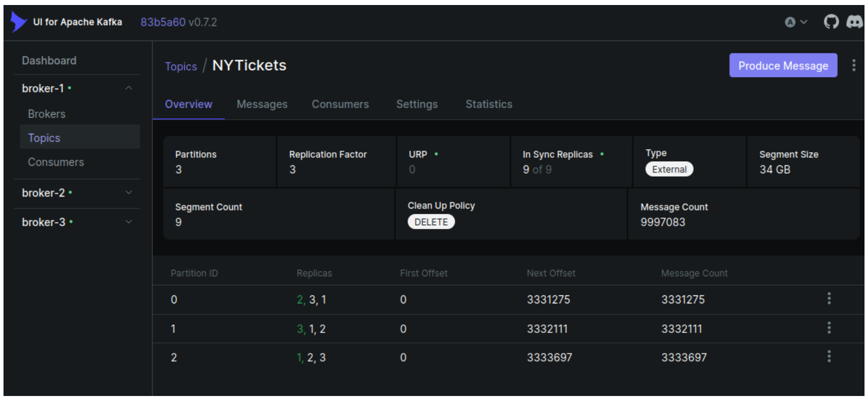
- **Producer**: Streams all NYC parking violation tickets (excluding augmented datasets) to a single topic, `NYTickets`.
- **Consumers**: Three consumers are configured to compute running statistics for the full dataset, specific counties/boroughs, and the top 10 streets with the most violations.

5.3 Running Statistics

The consumers compute real-time running statistics on the incoming data, focusing on:

- **Mean**: The average value of vehicle years or other relevant metrics.
- **Standard Deviation (Std)**: Measures the amount of variation or dispersion of the dataset.
- **Minimum (Min)**: The minimum value observed in the dataset.
- **Maximum (Max)**: The maximum value observed in the dataset.
- **Anomalies/Invalid Data**: Detection of data anomalies, such as vehicle years outside the valid range (e.g., years in the future or before the invention of cars).

These statistics are computed over a sliding window to ensure they reflect the most recent data. The consumers process the data as follows:



Partitions	Replication Factor	URP	In Sync Replicas	Type	Segment Size
3	3	0	9 of 9	External	34 GB

Segment Count	Clean Up Policy	Message Count
9	DELETE	9997083

Partition ID	Replicas	First Offset	Next Offset	Message Count
0	2, 3, 1	0	3331275	3331275
1	3, 1, 2	0	3332111	3332111
2	1, 2, 3	0	3333697	3333697

Figure 6: UI for Apache Kafka displaying the "NYTickets" topic with 3 partitions. The figure shows how replicas are distributed across brokers, along with the offset and message count for each partition.

5.3.1 Full Dataset Statistics

This consumer calculates the running statistics for the entire dataset, providing a comprehensive overview of the data trends.

5.3.2 County/Borough-Specific Statistics

This consumer breaks down the statistics by county or borough, allowing for geographic-specific analysis. This helps identify areas with higher violation rates and potential patterns unique to certain regions.

5.3.3 Top 10 Streets Statistics

This consumer focuses on the top 10 streets with the highest number of violations (as determined by our analysis, see Figure 3), providing detailed statistics for these key locations. This analysis highlights specific hotspots for parking violations, useful for targeted enforcement and urban planning.

By setting up this Kafka streaming architecture, we ensure that our analysis is both real-time and scalable, capable of handling large volumes of data efficiently.

5.4 Vehicle Year Prediction/Imputation

For the vehicle year prediction and imputation, we employed an online learning approach utilizing nearest neighbor techniques. This method leverages streaming nearest neighbor theory, including locality sensitive hashing, learning-to-hash techniques, and synopses for subsampling.

5.4.1 Streaming Nearest Neighbor Theory

The streaming nearest neighbor approach is based on several key techniques:

- **Locality Sensitive Hashing (LSH)**: Uses random hash functions to group similar items, enabling efficient nearest neighbor searches [1].
- **Learning-to-Hash**: Applies machine learning methods, such as Principal Component Analysis (PCA), to learn hash functions that preserve the proximity of data points [2].
- **Synopses**: Utilizes subsampling methods to maintain a compact representation of the data, allowing efficient updates and queries.

5.4.2 Implemented Methods

We implemented three different methods for vehicle year prediction, each incorporating comprehensive feature extraction steps including OneHotEncoding for categorical features, conversion of numeric features to float, parsing date features into year, month, and day, and standard scaling where applicable.

Method 1: Centroid-based k-Nearest Neighbors (Centroid-kNN) This method maintains a running average vector (centroid) for each vehicle year category. For each new sample, its features are compared to the centroids of all categories, and the nearest centroid (k=1) determines the predicted vehicle year. This effectively reduces the complexity of searching by using precomputed centroids as representatives of the categories. The centroids are periodically updated to reflect new incoming data.

Method 2: Incremental PCA with k-Nearest Neighbors (IPCA-kNN) This approach applies incremental PCA to perform dimensionality reduction on the data, reducing it to 20 principal components. For each new sample, its features are first transformed using the PCA model and then k-Nearest Neighbors (k=5) is applied to predict the vehicle year. Incremental PCA processes data in small batches and updates the PCA transformation incrementally, allowing the model to handle streaming data. The scaler and PCA are updated every 1000 samples to ensure they adapt to new data distributions.

Method 3: Stochastic Gradient Descent Regressor (SGDRegressor) The SGDRegressor model performs online learning by training on one sample at a time after an initial warmup phase with 1000 samples. This method uses a linear model with stochastic gradient descent optimization. It predicts the vehicle year by using a mean shift (predicting the offset from the year 2000) and a standard scaler, which is incrementally updated with each new sample to normalize the feature set. This method continuously learns and adapts to new data, providing robust performance over time.

5.4.3 Performance Metrics

Evaluation of the models is performed every 3000 samples using a set of 100 samples. The performance metrics used are Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). The results are summarized in Table 1.

Method	**RMSE**	**MAE**
Centroid-kNN	8.7	6.7
IPCA-kNN	5.8	4.7
SGDRegressor	5.1	4.0
Baseline (mean=2013)	6.2	4.6

Table 1: Performance Metrics for Vehicle Year Prediction Methods

By implementing these online learning methods, we aim to accurately predict or impute vehicle years in real-time, leveraging the continuous stream of parking violation data.

6 ML prediction

For our batch prediction task we wanted to find a practical problem so we decided on predicting the number of violations in some time frame. We partitioned the task into two stages: an hourly based prediction where we predicted the number of violations for every hour; and a daily based prediction, with the number of violations per day as our main target. For both cases we took the data up to January 2023 as training data and the whole year of 2023 as test. Additionally we performed a separate predictions task for each borough. This required splitting the violations dataset into 5 parts and add the additional data specific to those boroughs. Luckily we gathered a very extensive additional dataset, so there were borough specific information easily accessible.

6.1 Pre-processing

We pre-processed the data, generating an hourly violation count for each day from 2013 to 2024. In the same way we processed the additional data (number of elementary schools opened at that time, number of electronic businesses opened at that time, number of constructions going on, number of concerts and events happening...). We also excluded the COVID-19 period where there were no violations recorded, since it would only confuse the models. After pre-processing most hours were violation-less which makes this a very difficult task for any machine learning model.

From the hourly dataset it was simple aggregation by day to generate a daily dataset: summation for number of violations, maximum value for number of events, open schools, constructions etc., and averaging for continuous values like most of the weather based columns.

6.2 Methods

We used three different methods to predict hourly and daily violation count with the addition of predicting the average as a baseline. We wanted to test methods of different complexities as well as different sources.

Linear Regression (Dask-ML) This is the simplest method from the three. We used the implementation from the dask_ml module which distributes the computation automatically [3].

XGBoost (third-party) The next method is XGBoost from the public python library [**PLACEHOLDER REF**]. This method supports distributed computing by default, which was great for comparison with other methods [4].

SGD Regression (Sklearn with partial-fit) The last method is Stochastic Gradient Descent Regression from the Sklearn module, which performs computations linearly by default but can be toggled to preform parallel with the partial_fit boolean [5].

6.3 Results

Since this is a regression task we used the same metrics as before: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). To evaluate the efficiency we also tracked the time each method needed to perform fitting and inference.

Daily violation count prediction From Table 2 we can see that among the three methods, XGBoost performed best, but even so much worse than the baseline. The baseline was achieved by predicting the mean number of violations per day which was 4832.5 in our case. One reason for poor performance of our models is their simplicity, but it is also a very difficult task. We saw in figure 2 that in 2023 (which is our test set) there was a huge spike in the number of violations, unprecedented in all previous years. With no such spikes in the training data the models had little chance to have accurate predictions.

The fit and inference time analysis shows that XGBoost out-performed the other two methods again. Linear Regression was the slowest but only at fitting.

Method	**RMSE**	**MAE**	**fit time** [ms]	**inference time** [ms]
Linear Reg.	5353.26	4125.62	120.17	0.00
XGBoost	3915.73	2899.72	0.10	0.00
SGDRegressor	4521.35	3467.84	90.60	14.87
Baseline (4443.74)	2601.55	2219.50	0.0	0.0

Table 2: Daily violation count prediction

Hourly violation count prediction From Table 3 we can see an apparent improvement in the values of the measures (except SGDRegressor), but these results need to be interpreted relative to the task. Since the timeframe for counting violations was cut down by a factor of 24 the errors would have to greatly decrease too. We can see that is not the case since relative to the mean number of violations per hour (184.6) the error for the best performing method - Linear Regression - is still more than 300% off. We did however manage to get closer to the baseline result, but still not beating it.

SDGRegressor completely failed in this task, which is probably a result of much more zero valued entries in both train and the test set. The same reason explains the general bad performance - more than half entries in the train set were zeros, creating an unbalanced dataset, making it harder for any method to figure out any patterns.

The time-complexity analysis stays unchanged from the last experiment.

Method	**RMSE**	**MAE**	**fit time** [ms]	**inference time** [ms]
Linear Reg.	955.18	553.19	136.30	0.01
XGBoost	1190.60	796.16	3.65	0.03
SGDRegressor	161633.10	81394.05	53.39	0.06
Baseline (184.6)	868.11	293.34	0.0	0.0

Table 3: Hourly violation count prediction

Borough specific violation count prediction Lastly we tried to improve the performance by incorporating the geographical information and splitting the hourly predictions to specific boroughs. The analysis reveals that Linear Regression generally provides the most balanced performance across all boroughs, like in the previous experiment. XGBoost, while slightly less accurate, demonstrates much better computational efficiency, with fit times under 4 ms, in all experiments up until this point, making it an excellent choice for scenarios where speed is critical. Interestingly, the Baseline model still outperforms all other models. This suggests that in some cases, the simplicity of the Baseline model can be more effective than more complex models especially in tasks as difficult as these. Conversely, the SGDRegressor model consistently shows the highest error rates, indicating it is not be suitable for these predictive tasks. Overall, while Linear Regression and XGBoost are mostly very close in performance, their comparison is a bit unnecessary since both are beaten by the baseline even with the more specific geographic information.

6.4 Different Data Formats and Scenarios

All the cluster tasks above were performed using parquet data files and reading them with Dask. This is of course only one of possibilities how to perform these tasks, so we also repeated all the experiments with two other methods and analysed the differences. The results of the default scenario are summed up in Table 5 and will be used for comparison.

Parquet and DuckDB DuckDB generates a database (.db) file that can be queried using SQL, enabling us to handle preprocessing tasks, such as column selection and data cleaning, simultaneously with data reading. The processed data is then easily transferred to a pandas DataFrame for reuse across all experiments. The experiment results remained consistent, as anticipated, but our primary focus was on evaluating whether this method was more efficient than the previous approach. Table 6 presents the measured times for various tasks in the process. Although data reading appears slightly slower, it's important to note that some data cleaning occurs during this step. The remaining tasks, including additional data cleaning, showed similar performance to the earlier method.

HDF5 and Dask This scenario was expected to perform worse than other since the beginning, because of the large data files that are slower to read by Dask. After many experiments the smallest size of all the HDF5 files we managed to achieve was 32.2 GB, approximately 6 times larger than the parquet files. This is reflects in the results seen in Table 7.

Manhattan				
Method	**MAE**	**RMSE**	**fit time** [ms]	**predict time** [ms]
Linear Regression	154.96	280.76	131.95	0.01
XGBoost	401.52	519.90	3.81	0.03
SGDRegressor	93916.42	134202.54	51.69	0.06
Baseline (58.12)	83.19	255.34	0.00	0.00

Queens				
Method	**MAE**	**RMSE**	**fit time** [ms]	**predict time** [ms]
Linear Regression	202.52	309.26	130.68	0.00
XGBoost	241.64	344.45	3.73	0.03
SGDRegressor	25069.53	35970.06	53.53	0.06
Baseline (41.03)	82.39	252.42	0.00	0.00

Brooklyn				
Method	**MAE**	**RMSE**	**fit time** [ms]	**predict time** [ms]
Linear Regression	122.30	244.26	132.01	0.01
XGBoost	235.51	336.29	3.90	0.03
SGDRegressor	148233.79	272198.08	53.79	0.06
Baseline (47.21)	78.54	234.22	0.00	0.00

Bronx				
Method	**MAE**	**RMSE**	**fit time** [ms]	**predict time** [ms]
Linear Regression	91.98	139.30	133.36	0.00
XGBoost	107.33	150.39	3.84	0.03
SGDRegressor	85860.45	141258.47	53.16	0.06
Baseline (25.28)	38.50	110.28	0.00	0.00

Staten Island				
Method	**MAE**	**RMSE**	**fit time** [ms]	**predict time** [ms]
Linear Regression	30.97	47.10	138.45	0.01
XGBoost	30.97	47.41	3.66	0.03
SGDRegressor	3478.99	4141.71	53.75	0.12
Baseline (5.97)	10.82	36.14	0.00	0.00

Table 4: Performance of models for different boroughs

Task	**Time (seconds)**
Data reading	8.73
Data Cleaning	0.14
Fill in missing hours	79.38
Add additional data	0.07
Train-test split	55.54

Table 5: Processing times for different tasks using Dask+Parquet

Task	**Time (seconds)**
Data reading & cleaning	18.29
Data Cleaning	0.12
Fill in missing hours	78.15
Add additional data	0.08
Train-test split	56.81

Table 6: Processing times for different tasks using DuckDB+Parquet

Task	**Time (seconds)**
Data reading & cleaning	37.84
Data Cleaning	0.15
Fill in missing hours	81.93
Add additional data	0.07
Train-test split	64.48

Table 7: Processing times for different tasks using Dask+HDF5

7 Conclusion

In this project, we analyzed New York City parking violations from 2014 to 2024, using big data techniques to uncover important patterns and trends in parking enforcement across the city. We applied various methods, to process and analyze large datasets, making them suitable for efficient querying, storage and analysis. Our analysis looked at different aspects, like time and location, to identify areas and times with the most parking violations.

A significant part of the project was the successful integration of additional datasets, such as weather data, event schedules, and business locations. These extra datasets helped us gain a deeper understanding of the factors that might influence parking violations. Through exploratory data analysis, we uncovered important patterns, like the effect of the COVID-19 pandemic on parking violations and the identification of hotspots with high violation rates.

We also set up a Kafka streaming architecture, which allowed us to process real-time data efficiently. This was a great simulation of what a real-life scenario would look like with violations streaming in live. With this setup, we could continuously monitor and analyze parking violations, ensuring the data was always up-to-date to perfomr analysis and use it for predictive tasks. The predictive models we developed, especially those using methods like Stochastic Gradient Descent Regressor and Incremental PCA with k-Nearest Neighbors, provided reliable forecasts for vehicle years.

For predicting the number of violations per day and per hour using batch machine learning methods, we tested several models, but the results were disappointing. Despite trying different algorithms, none of the models managed to outperform the baseline, which simply predicts the mean number of violations. This outcome suggests that predicting parking violations is more complex than anticipated, and it highlights the challenges of applying machine learning to this kind of data. In future work we would like to test if more complex methods, more features and a more accurate location information would help in this task.

References

[1] Xiaohui Jiang et al. “A survey of real-time approximate nearest neighbor query over streaming data for fog computing”. In: *Journal of Parallel and Distributed Computing* 116 (2018). Towards the Internet of Data: Applications, Opportunities and Future Challenges, pp. 50–62. ISSN: 0743-7315. DOI: `https://doi.org/10.1016/j.jpdc.2018.01.005`. URL: `https://www.sciencedirect.com/science/article/pii/S0743731518300182`.

[2] Jingdong Wang et al. *A Survey on Learning to Hash.* 2017. arXiv: `1606.00185 [cs.CV]`. URL: `https://arxiv.org/abs/1606.00185`.

[3] *dask_ml.linear_model.LinearRegression — dask-ml 2024.4.5 documentation.* URL: `https://ml.dask.org/modules/generated/dask_ml.linear_model.LinearRegression.html` (visited on 08/14/2024).

[4] *dmlc/xgboost at stable.* URL: `https://github.com/dmlc/xgboost/tree/stable` (visited on 08/13/2024).

[5] *SGDRegressor.* scikit-learn. URL: `https://scikit-learn/stable/modules/generated/sklearn.linear_model.SGDRegressor.html` (visited on 08/13/2024).