Data Science @UL-FRI

FRI Big Data Project Report

Analysis of Parking Violations in New York

Cristian Bassotto, Nikolay Kormushev and Ahmet Çalış

Introduction

In the bustling metropolis of New York City, parking violations are a common occurrence, reflecting the challenges of urban transportation and the city's rigorous traffic regulations. This project aims to delve into the vast troves of data generated by parking violations over multiple fiscal years from 2014 to 2024. Our goal is to provide a comprehensive analysis that can inform policy, improve urban planning, and enhance the overall efficiency of parking enforcement.

Datasets

The datasets used are gotten from the *NYC OpenData* online database. We are looking at the fiscal years between 2014 to 2024 for the Parking Violations. For further analysis of outside factors effects the datasets used are the following: *NYC Permitted Event Information - Historical for events* which contains information on when an event happened and if a street was closed for it; *DOE High School Directory from 2016-2021* which contains high school location; *Legally Operating Businesses* for business locations; *348 New York Tourist Locations Kaggle* dataset to find locations of tourist attractions. On the other hand, weather data is gathered from meteostat [1], a python library that uses pandas to retrieve the data from different public interfaces. For information about the columns of these datasets refer to Tables 1-6.

Table 1. Main features for Parking Violations dataset

Feature Name	Description
Summons number	Unique ID
Plate ID	Plate of the vehicle
Vehicle Make	Manufacturer of vehicle
Issue Date	Date of the violation
Street Code 1	The street of the violation
Street Code 2	Intersection street
Street Code 3	Another intersection street
Violation County	Borough name
Street Name	Name of street violation occurred
More	More unlisted features exist

Table 2. Main features for High School Datasets

Feature Name	Description
dbn	Unique ID
school name	name of school
borocode	borough code
primary_address_line_1	primary address
borough	borough code
More	More unlisted features exist

Table 3. Main features for Business Dataset

Feature Name	Description
DCA License Number	Unique ID
License status	Active or inactive license
Business Name	Name of business
Address Street Name	primary address
Borough	Borough name
Industry	In which industry is the business
More	More unlisted features exist

Table 4. Main features for Attractions Dataset

Feature Name	Description
Tourist Spot	Name of attraction
Address	Full address plus code for borough
Zip code	-

Table 5. Main features for Events dataset

Feature Name	Description
Event ID	Unique ID
Event Name	-
Start Date time	start date and time
End Date time	end date and time
Event Borough	borough
Event Location	Address
Street Closure type	How and was the street closed?
More	More unlisted features exist

Converting file formats

The parking violation dataset was split into multiple CSV files. One for each year. To ease analysis our work we wanted to

Table 6. Main features for Weather dataset

Feature Name	Description
Date and hour	Date and hour of the recording
borough	Borough of the recording
temp	Air temperature in °C
dwpt	Dew point in °C
rhum	Relative humidity in percent (%)
prcp	One hour precipitation total in mm
wdir	Average wind direction in degrees (°)
wspd	Average wind speed in km/h
pres	Average sea-level air pressure in hPa
coco	Weather condition code

combine them into on file and use a different format that is quicker to parse. We chose Parquet and HDF5 formats with the plan of comparing them. Due to the big input file size we used Dask to read them in chunks making the combination possible.

Combining files

Combining we discovered that some of the column names were not consistent and we created a shell script to identify and rename them. To find the mismatched columns we read the first line of the source CSV files and compared the strings. The results were these 2 columns: no_standing_or_stopping different from no_standing_or_stopping_violation and violation_in_front_of_or from violation_in_front_of_or_opposite. For both cases we used the second version.

Also before combining to each file we added a DataYear column to make it easier to identify which fiscal year it was related to and which is the source file of each row.

Data type issues

To convert to parquet we needed to specify all the data types of the original CSV files. The reason for this was that between files the column values were not of consistent types or would often have null values which confused the automatic inference. As a result we decided to make most type either object or string.

HDF5 issues

After this converting to parquet was easy but with HDF5 we had issues due to the length of the strings. For it it was mandatory to specify the max length of each string. To solve this we calculated the max length of each field and gave them to the hdf5 conversion method and all worked.

Parquet partition sizes

Finally I used repartition to make the parquet chunks while in memory of a size of 100MB. I read online this was a good value for quick performance but I could not find out why exactly 100MB. It was mostly a magic number. Nevertheless the small size helped in later tasks making the information loaded in memory smaller.

Comparison between HDF5 and Parquet

Main difference is the file size of the HDF5 and compression used as seen in Table 7. For us it was around 26GB with the highest compression which slowed down performance while the one for parquet was 6.7GB with the default snappy compression which allowed for quick reading.

To compare the reading speed I read both the hdf and parquet files and executed the same count operation for number of rows that have a value for street_code1. As we can see Parquet is almost 6 times faster than the hdf file. This is by no means a perfect comparison since execution time likely depends on types of operation but we used a simple enough operation for a general idea of execution times. Another issue we ran into is that DuckDB which we use later does not support HDF.

As a results due to the large file size and the slower execution time we decided to just use parquet for future tasks.

Table 7. File types comparison

File format	File size	Compression	Exec Time
HDF5	26GB	level 9 blosc:lz4	614 sec
Parquet	6.7GB	snappy	136 secs

Merging with 3rd party datasets

In this section we merge the parking dataset with the datasets related to high school, attractions, weather, businesses and events listed before. The idea is to use these new datasets for further analysis on how violations can be effected by these factors.

Merging approaches

When we started merging it was not clear on what to merge. Most datasets have a borough representing the region of New York in which the high school or the business is located. This might be good enough for weather events since they don't differ in each street but likely a high school on one side of a borough does not affect a parking violations on the other side.

We also found that there are listed the names of the streets but they are not following a consistent format between the datasets which made merging more or less impossible except if we do not use fuzzy matching. The issue with fuzzy matching is that it is slow and takes too long to check all the rows. We also found a small library that converts street names to a standard format but that did not work as well. We tried to also do it by hand with rules to see how many we could match but there were too many edge cases.

It was possible to use a geolocation API to standardize the names of the streets. The issue is we have millions of rows and these APIs are rate limited except if we don't pay which we are not willing to do. So we opted for other solutions.

After we saw that except the names of the streets there were the codes of the streets. Initially it was unclear how these codes were chosen and also they were missing from the other datasets we wanted to merge with. After a bit of research we found the Street Name Dictionary on the NYC OpenData website. In it was a mapping between the multiple namesaliases of a street and its 10 digit street code. The code includes the borough and the first 5 digits identify the street. The last 5 digits identify the alias of the street. After a bit of experimentation we found that the first 5 digits are the same as the ones used in the street code fields in the parking violation dataset. In the end we decided to use this dataset to merge with the smaller high school, business, attraction datasets and merge the results with the parking violation dataset based on street code. This finally worked but it was not without issues.

Issue with our approach

One problem is that some streets are long. Maybe a high school or a business is on one side and the violation on the other. We do not have a way to resolve this. Even having coordinates wouldn't help since it is unclear which part of the street the coordinates represent.

Another issue is we did not manage to successfully merge the events based on street code. Partly this is because during a lot of events the streets where the event took place was closed so there are no parking violations there. Because of this we decided to just merge only on the borough field and the day of the event.

The third problem is that a lot of streets do not have a street code in the parking dataset. Often they are missing but due to the hard to use formats for the street names it was not possible to merge based on names or aliases to assign street codes.

Type of Join

We also decided to use a left join and keep all entries on these datasets that have no matching business on their street for example. We wanted to be able to compare streets with a lot of businesses or schools with ones with less. The issue here is that it is unclear if a street actually has no businesses or a match was not possible or some data was lost in the previous sections. This might result in our results not being entirely accurate for this case.

Events date and time

For events we needed also to merge on date and time so that our results are accurate. Because some events spanned multiple days we remove those and kept only the ones that last a single day. This way we could more easily merge only on the date.

Merging with dask vs merging with DuckDB

In Table 8 we have added the results for merging the attractions dataset which has around 150 entries with the parking dataset. In this case we can see that DuckDB performs twice as fast compared to dask if not parallelised. On the other hand, SLURMCluster boosts dask decreasing the time to merge and save the parquet file by half compared to DuckDB.

Table 8. Join time comparison

Technology	Dataset	Time
DuckDB	Attractions	8.6 min
Dask Single process	Attractions	15.1 min
Dask using SLURM cluster	Attractions	5.0 min
Dask using SLURM cluster	Weather	9.0 min
Dask using SLURM cluster	Schools	5.2 min
Dask using SLURM cluster	Events	16.8 min
Dask using SLURM cluster	Business	11.0 min

File size issue

When I merged using DuckDB with other larger datasets like events and businesses I did not succeed and ran into a disk quota error. Looking into it I found out that there is a bug in DuckDB where output parquet file sizes are more than 4 times bigger with the same compression (12 GB with dask and 48 with duckDB before reaching the quota). Due to the difference in size I couldn't do a good comparison of join times because it took more than 3 hours to write the file due to the size. With dask merging and writing with a single process took 1 hour and 30 minutes.

The reason I was measuring the JOIN time with the writing was due to the big input data size. DuckDB uses lazy evaluation which means the JOIN was not being executed until I used the data. The only option was to read in chunks due to large input size and write to a file to force the JOIN operation to occur. Another option was to read chunks until we pass the whole dataset but I am unsure how this would effect the calculations.

The JOIN is really fast if I want just a subset of the data that fits into memory because it does not process everything but just enough so I get the amount of data I requested.

For reference the Github DuckDB issue is called: 'Too large parquet files via "COPY TO"'.

In conclusion for merging and saving larger files Dask is currently better.

Exploratory Data Analysis

In the following section, we conduct exploratory data analysis using both DuckDB and Dask. Our initial focus is on the number of tickets issued per year.

The data reveals (Figure 1) a consistent increase in the number of tickets over the years. Notably, the year 2024 shows a significantly lower number of tickets, which is expected as the year is still ongoing.

Speed violations (Figure 2) are by far the most common type of infraction, followed by parking violations related to street cleaning and no-standing regulations. The term "No Standing" refers to restrictions on stopping or remaining in a vehicle at a given location, regardless of whether the engine is running.

As expected, the streets with the highest number of violations (Figure 3) are the major thoroughfares in New York City.

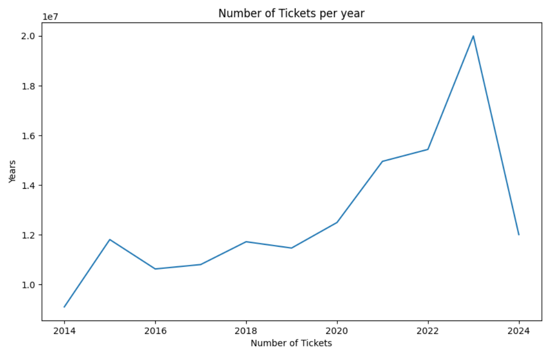

Figure 1. Number of Tickets per Year

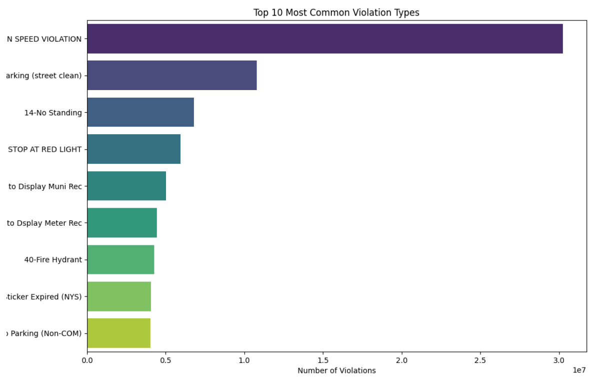

Figure 2. Most Common Type of Violations

Broadway tops the list, followed by 3rd Avenue and then 5th Avenue.

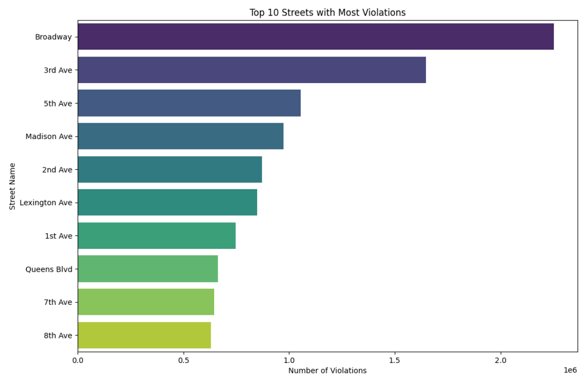

Figure 3. Top Streets for Num. Violations

The car brands with the highest number of violations (Figure 4) are Toyota, Ford, and Honda, respectively. This trend correlates with the market share of these brands in the U.S., as they are among the best-selling vehicles in the American market.

Figure 5 illustrates a positive correlation between the number of students and the number of violations. This trend may

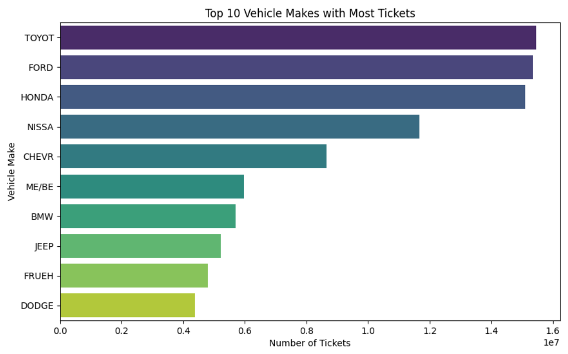

Figure 4. Top 10 Vehicle Brands for Num. Violations

be attributed to parents picking up students from school, which often leads to parking violations in the vicinity.

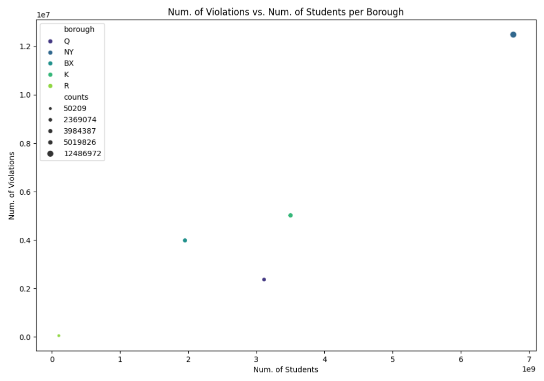

Figure 5. Correlation Between Num of Students vs Num. Violations

The highest number of violations by industry (Figure 6) is observed in tobacco dealers and shops. This may indicate that individuals associated with smoking are more likely to commit violations, or it could be attributed to delivery trucks associated with these businesses, which might violate parking regulations while loading and unloading goods.

When examining the number of violations by event type (Figure 7), the farmers market ranks highest. This is likely due to people parking their cars briefly while they quickly buy something, leading to various parking violations. Following the farmers market are street events and sidewalk sales. The reasons for violations during sidewalk sales are similar to those for farmers markets: people often park inappropriately due to the convenience of quick purchases. For street events, the large influx of visitors results in a high demand for parking spaces, which becomes limited. Consequently, people may park in unauthorized areas, paying less attention to parking regulations. This increased demand for parking likely contributes to street events and sidewalk sales also appearing among the top three for violations.

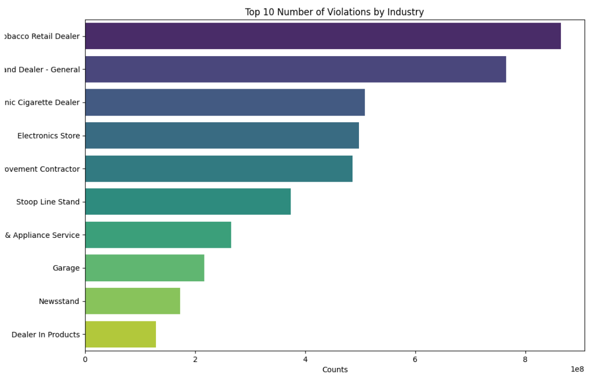

Figure 6. Number of Violations by Industry

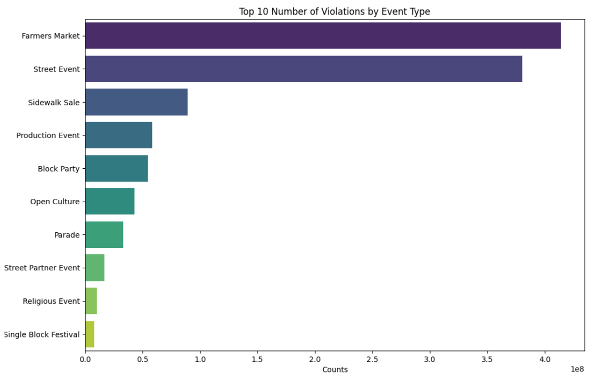

Figure 7. Number of Violations by Event Type

Streaming analysis

In this section, we perform data analysis in a streaming manner, treating the data as a continuous stream. We utilize Apache Kafka as a message broker to facilitate the real-time ingestion, processing, and analysis of the data. Kafka enables the handling of large volumes of data in a fault-tolerant and scalable manner.

Kafka

Producer

A custom Kafka producer was developed to read data from CSV files. The data, which is assumed to be ordered by the ”summons number,” is processed sequentially. Each line from the CSV file represents a record, and the producer sends these records as Kafka messages to the appropriate Kafka topic. To optimise data processing, we read the data in chunks to avoid storing a large quantity of data in memory. When sending data, we choose not to read line by line to take advantage of the parallelisation capabilities of the pandas library.

We concluded that a single topic was sufficient for analysing the streaming data and opted to store the minimum number of variables in classes that are updated in real-time, eliminating the need to save objects on a list.

Consumer

A custom Kafka consumer was implemented to process the streamed data. The consumer subscribes to the relevant Kafka topics and processes the incoming data in real time. The processing involves calculating rolling descriptive statistics and applying a spatial stream clustering algorithm.

Data Analysis

To gain insights into the data as it streams, rolling descriptive statistics were computed for various categories, including overall data, individual boroughs, and the ten most significant streets (based on the highest number of tickets). We store and print the total count, mean, standard deviation, minimum and maximum values. We have chosen to calculate these statistics in order to optimise memory usage. Our aim was to find a solution that would not require storing a list of items, but would instead update the values in real time with O(1) memory usage.

Clustering

Street Map

The street map depicted in Figure 8 illustrates the spatial distribution of street locations that were analyzed in this study. This map serves as a visual reference for understanding the geographical context in which the clustering analysis was conducted.

We were able to map the streets to their correct location thanks to a dataset of NYC Street Segments [2]. This dataset contains a lot of informations about each segment of each street in New York City.

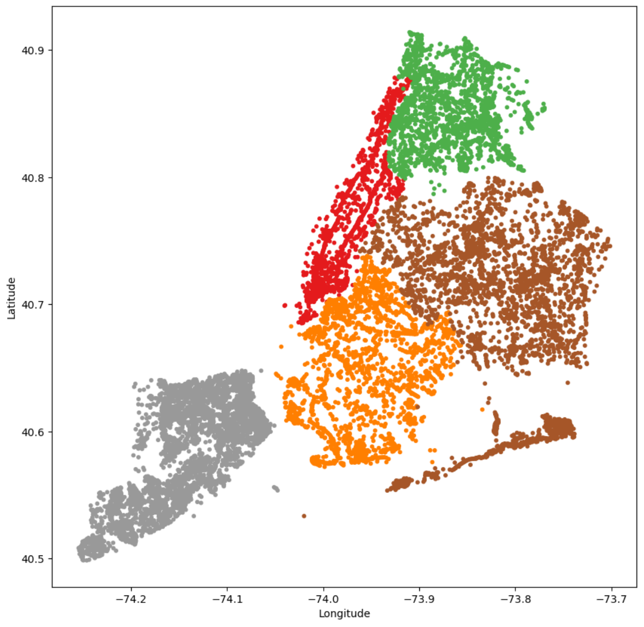

Figure 8. Street Locations

We focus on the spatial geometry of the streets, their street code and street name. In particular, street code contains also informations about the borough and street names are the official names of the segment. Sadly, street code and names are not unique to each other because some street codes has multiple names based on the segment related. The geometry, on the other hand, has multiple spatial details, so we first decide to calculate the centroid of each geometry and then translate the coordinate in latitude and longitude that are easier to plot.

In this way we manage to create a dataset with unique street codes (grouping by them), multiple street names, latitude and longitude data.

DenStream

As a spatial streaming algorithm, we leverage DenStream [1] to effectively manage the continuous influx of geospatial data while maintaining the ability to identify dynamic clusters in real time. DenStream's ability to handle spatial data is particularly valuable in scenarios where the data points are associated with specific geographic locations, such as the clustering of traffic violations across different streets in a city.

By using DenStream, we can continuously monitor the formation and evolution of clusters, such as areas with high traffic violation density, without the need to store all past data points. The algorithm's capacity to distinguish between stable, dense regions and transient, outlier regions enables us to focus on significant patterns that persist over time, while also allowing for the detection of emerging trends. This spatial streaming approach ensures that our clustering analysis remains both timely and relevant, adapting to changes in the spatial distribution of the data as new information is streamed.

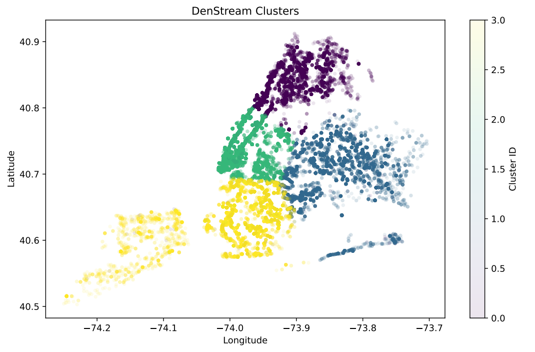

Figure 9. Clusters at 100k violations

Initially, the clustering analysis using the DenStream algorithm resulted in four distinct clusters, as depicted in the Figure 9. The clusters, denoted by different colors, indicate areas where parking violations were concentrated, providing insights into patterns or hotspots of violations.

The color intensity of the cluster markers suggests the density or significance of the violations within each cluster. The image reveals an interesting aspect of the clustering, as it differentiates between clusters even in areas where streets seem connected. For instance, in Manhattan, two streets that appear contiguous are divided between the green and purple clusters. This indicates that while the DenStream algorithm is effective in identifying areas of high parking violation density, it may also segment closely situated streets into different clusters based on subtle differences in the data, such as variation in violation frequency or patterns over time. This level of detail could be valuable for more granular analysis but also suggests the need for careful interpretation when assessing the continuity of clusters in geographically adjacent areas.

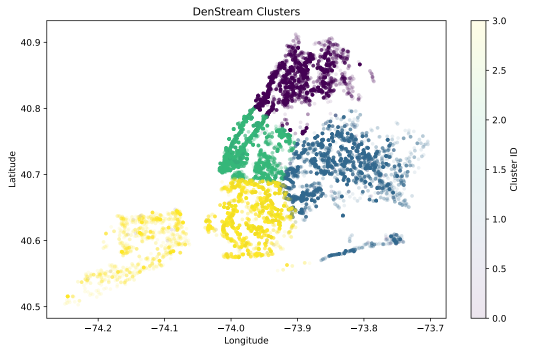

Figure 10. Clusters at 2M violations

As we can see from Figure 10, even as we proceed with streaming data, the clusters remain relatively consistent with each other. This indicates that the clustering behavior is stable over time, despite the continuous input of new data. The DenStream algorithm appears to maintain coherence in identifying clusters, ensuring that the spatial patterns of parking violations are preserved and comparable throughout the analysis period. This consistency is crucial for reliable long-term monitoring and can provide confidence in the robustness of the clustering approach applied to the streaming data.

Machine Learning

In this section, we focus on data preprocessing and the application of various machine learning algorithms.

Days with high number of tickets

The initial machine learning task was to predict days with a high number of tickets, which involves a binary classification problem. Since most of the machine learning algorithms cannot process NaN values and we are working with a large dataset, we began by removing NaN values. We then aggregated the data by day to count the number of tickets for each day. Days with ticket counts above the median were labeled as high-ticket days. We created new features based on the issue date, such as day of the week and month. Finally, we converted the feature matrix X and target vector y into Dask arrays for processing.

Violation Type Prediction

As a second task, we focused on predicting the violation type using the violation code as labels. Similar to our first task, we

[1] https://riverml.xyz/latest/api/cluster/DenStream/

began by removing any NaN values from the dataset. Next, we created new datetime features based on the issue date. Specifically, we categorized the hour into different times of the day—morning, afternoon, evening, and night—to create a new feature called ”time of the day.”

We also categorized features such as Borough, Precinct, Street Name, Time of the Day, and Vehicle Make using Dask. After these categorizations, we prepared the features and target variables. The categorized features were then encoded using Dask's preprocessing utilities. Finally, we converted the data into Dask arrays, as we did in the previous task, to facilitate further processing and model training.

For both tasks, we employed the following machine learning models:

1. **Dask XGBoost**: We used Dask XGBoost with the following configurations:
    - `objective`: `multi:softmax`
    - `learning_rate`: 0.1
    - `max_depth`: 6
    - `n_estimators`: 100
2. **Logistic Regression**: We applied logistic regression from the Dask ML library for classification tasks.
3. **SGD Classifier**: To handle the streaming nature of the data, we used the SGD (Stochastic Gradient Descent) classifier from the scikit-learn library, leveraging the `partial_fit` method for incremental learning.

References

[1] Meteostat. Meteostat python library, 2024. Accessed: 2024-08-17.

[2] NYC Department of City Planning. Lion: Nyc's street base map, 2024. Accessed: 2024-08-17.