CS-GY 6513 FINAL PROJECT

NYC Safety and Livability Analysis

Project Report

Devarsh Patel (DP3324)
Nigel Saurino (NS5329)
Priyank Viradia (PDV8883)

Table of Contents

Introduction

Objective

Identify the preferable place in NYC for students to live in terms of crime and transportation using historic dataset provided by NYC via their open data initiative and daily rental price obtained from Airbnb.

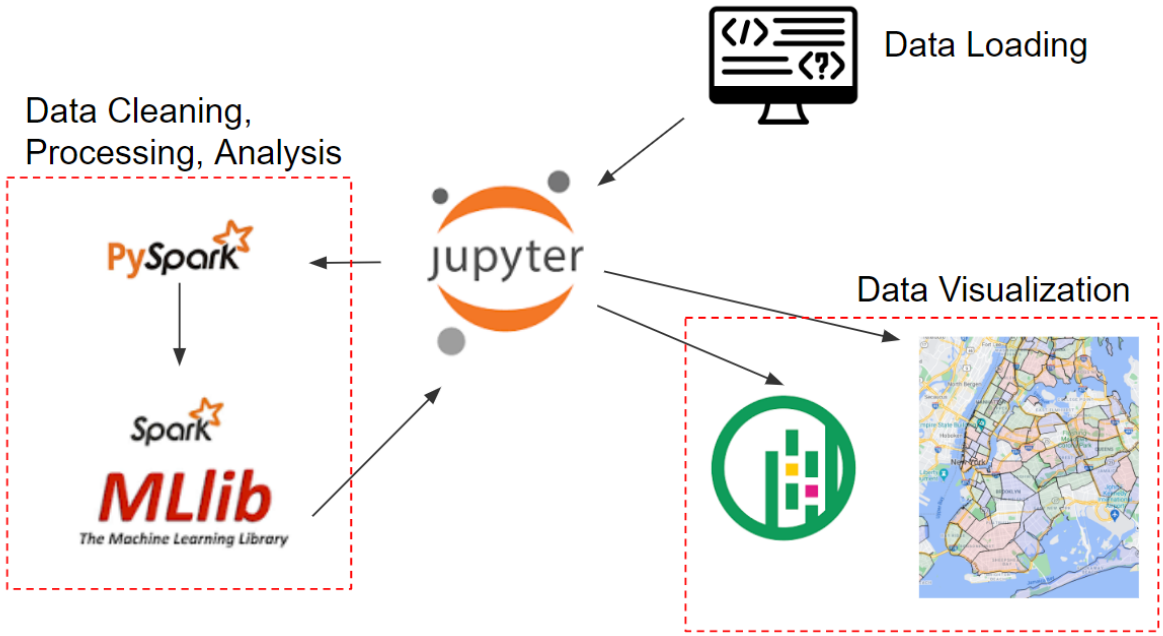
Abstract

New York City brings approximately 8000 out of state students to New York City each year. Choosing a neighborhood to live in is difficult for any city but especially important in a place like New York City where the population is so dense and neighborhoods quickly transition in only a few streets. It's extremely difficult to provide a comprehensive analysis of a city that is as large as New York. The MTA subway system had a daily average ridership of 5.5 million people in 2019 (pre-pandemic) which means the data sets of recorded interactions balloon rapidly over the course of a few years. To get a full picture of the impact of that many people in such a small geographic footprint, we anchored our analysis on the subway stations where individuals are most likely to have the most social interaction and therefore, the largest available datasets. This project analyzes the relationship between subway stations and incidents in New York City (NYC). The project focuses on incidents such as arrests, complaints, and court summons, and aims to determine if these incidents occurred in close proximity to subway stations. The analysis also incorporates predicted rental cost data in order to help new students find relative value in the rental market.

Why Big Data?

It's extremely difficult to provide a comprehensive analysis of a city that is as large as New York. The datasets we worked with were large-scale, requiring the processing power of big data platforms to handle the massive amounts of data effectively. Big data technology allows us to scale our analysis as new data becomes available, ensuring that our insights remain up to date. Advanced calculations, aggregations, and joins across multiple datasets were required for our analysis, which big data platforms efficiently support. Big data frameworks enable parallel processing and distributed computing, resulting in faster execution and reduced processing time. Big data platforms facilitate the integration of diverse data sources, allowing us to bring together different datasets for a comprehensive analysis.Big data technology offers flexibility in handling various data formats and sources, accommodating the diverse data requirements of our project.

Architecture



Technologies

In our project, we leveraged the power of Big Data and Machine Learning to gain valuable insights and provide a comprehensive analysis.

Pyspark

We utilized Pyspark, a powerful framework for distributed data processing, to load, clean, and combine large-scale datasets. This allowed us to efficiently handle the vast amount of data involved in our analysis.

Pyspark ML

To perform predictive analysis, we employed Pyspark ML, a machine learning library, which enabled us to build models that determine the relative value of safety to cost for incoming students. By utilizing machine learning algorithms, we were able to derive meaningful patterns and make predictions based on the available data.

Geopandas

Visualization played a crucial role in our project, allowing us to present the insights in an intuitive and engaging manner. For this purpose, we utilized Geopandas, a Python library for geospatial data manipulation, to create a geodataframe and map it onto a grid of New York City. This provided a visual representation of the data and allowed us to analyze spatial patterns and trends.

Leafmap

To enhance the interactive experience for users, we integrated Leafmap, a Python library for interactive mapping, into our visualization. Leafmap enabled us to create interactive maps with pop-ups by subway stations, providing additional details such as arrests, complaints, total traffic, and local rental costs. This interactive approach allowed users to explore the data further and gain more insights by interacting with the map.

Challenges

In our project, we encountered several challenges related to the Spark instance and the dataset itself. These challenges required careful optimization and modification of our approach to ensure efficient processing and accurate analysis.

One of the main challenges we faced was optimizing the Spark instance. This involved fine-tuning parameters such as executor memory, maxStringToFields, number of cores, and instances. Finding the right configuration for these parameters was crucial to ensure optimal performance and avoid issues like out-of-memory errors or slow execution.

Another challenge was related to the dataset itself. We had to partition the data to equally distribute it among the executors. This partitioning process involved careful consideration of the data size, distribution, and the available resources in the Spark cluster. By partitioning the data effectively, we were able to achieve parallel processing and leverage the full capacity of the cluster.

Additionally, we had to modify and select the appropriate columns from the dataset to perform the desired operations in the executors. This involved filtering out irrelevant or missing data, casting data types to ensure compatibility, and selecting only the necessary columns for analysis. This step was crucial in optimizing the processing time and reducing the memory footprint

Overall, these challenges required us to have a deep understanding of Spark's architecture, data partitioning techniques, and data manipulation capabilities. By addressing these challenges and optimizing our approach, we were able to overcome the limitations imposed by the dataset size and complexity, and successfully perform the required data processing and analysis.

Data

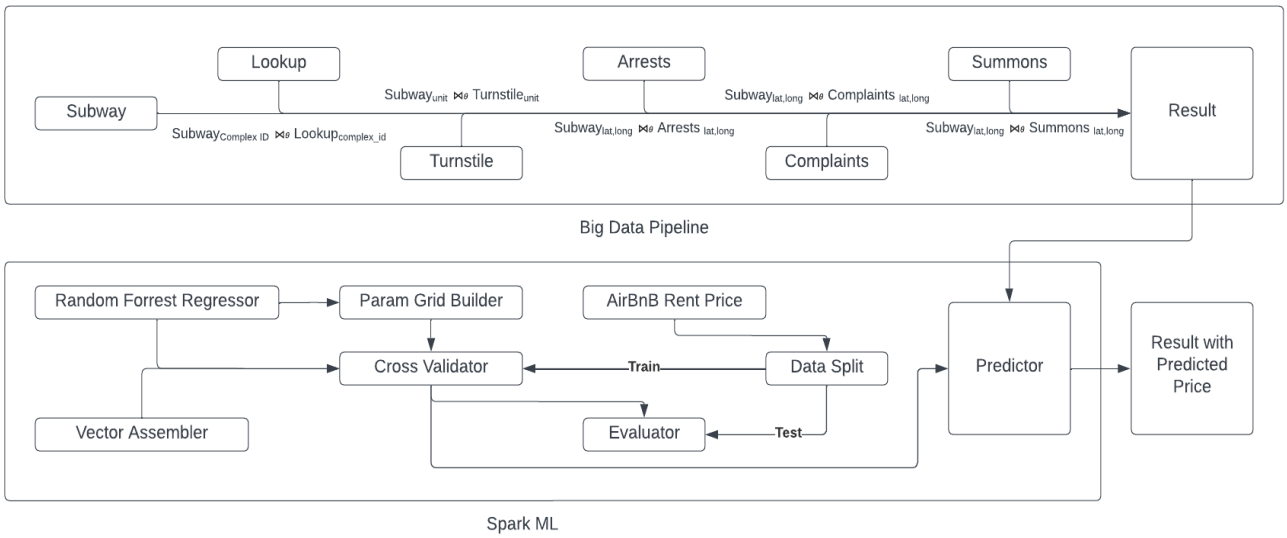
The project relies on several key datasets sourced from the NYC Open Data project's API and a publicly available Airbnb dataset. The datasets include NYC arrests, court summonses,

complaints, subway stations, turnstile activity, and rental cost data. The data files are imported using Spark and are repartitioned based on the executor instances during fetching.

 # data urls

- historic_arrest_loc = { 'url': 'https://data.cityofnewyork.us/resource/8h9b-rp9u.json?$limit=15000000', 'filename': 'arrest.json' }
- historic_complaint_loc = { 'url': 'https://data.cityofnewyork.us/resource/qgea-i56i.json?$limit=15000000', 'filename': 'complaint.json' }
- historic_court_summons_loc = { 'url': 'https://data.cityofnewyork.us/resource/sv2w-rv3k.json?$limit=15000000', 'filename': 'summons.json' }
- traffic_speed_loc = { 'url': 'https://data.cityofnewyork.us/resource/i4gi-tjb9.json?$limit=15000000', 'filename': 'speed.json' }
- turnstile_loc = { 'url': 'https://data.ny.gov/resource/i55r-43gk.json?$limit=15000000', 'filename': 'turnstile.json' }
- subway_loc = { 'url': 'http://web.mta.info/developers/data/nyct/subway/Stations.csv?$limit=10000', 'filename': 'subway.csv'

Analysis



Data Processing

We have performed a series of data transformations and joins on several large datasets. We have also managed to process 65 GB of data after run-time repartitioning and shuffle read/write operations, while using less than 2.5GB of driver memory.

- Big Data Pipeline - The datasets undergo the following preprocessing steps:
    - Column Selection: Relevant columns related to each incident type (arrests, complaints, summonses) and the subway stations are selected for further analysis.
    - Data Cleaning: Columns with invalid data or missing values are dropped to ensure data quality.
    - Filtering: Null values in the latitude and longitude columns are removed to retain only valid geospatial data.
    - Data Type Casting: The latitude, longitude, and rental cost columns are cast to appropriate numeric data types for subsequent analysis.

- Big Data Pipeline steps:
    - Turnstile Dataset Analysis:
        - The turnstile dataset is grouped by subway station units (identified by the 'unit' column).
        - The total entries and exits for each unit are calculated using the 'sum' aggregation function.

        - The resulting DataFrame includes columns for unit, entries, and exits.
    - Subway Station Dataset Analysis:
        - The subway station dataset is joined with the turnstile analysis DataFrame using the 'unit' column as the common key.
        - Relevant columns, such as subway station ID, line, stop name, borough, latitude, longitude, and directional labels, are selected.
        - The entries and exits columns from the turnstile analysis DataFrame are also included in the resulting DataFrame.
    - Arrest Dataset Analysis:
        - The subway station DataFrame is joined with the arrest dataset using the `withinMile` UDF to filter incidents that occurred within 0.1 miles of subway stations.
        - The latitude and longitude columns from the arrest dataset are dropped to avoid redundancy.
        - Rows with missing values in the stop name column are removed.
        - The resulting DataFrame is grouped by subway station ID, line, stop name, borough, latitude, longitude, directional labels, entries, and exits.
        - The count of arrests for each subway station is calculated and stored in a new column named 'arrests'.
    - Complaint Dataset Analysis:
        - The subway-arrest DataFrame (sa_df) is joined with the complaint dataset using the `withinMile` UDF.
        - Similar data cleaning and grouping operations are performed, resulting in a new DataFrame (sc_df).
        - The count of complaints for each subway station is calculated and stored in a new column named 'complaints'.
    - Summons Dataset Analysis:
        - The subway-complaint DataFrame (sc_df) is joined with the summons dataset using the `withinMile` UDF.
        - Data cleaning and grouping operations are repeated to create a new DataFrame (ss_df).
        - The count of court summonses for each subway station is calculated and stored in a new column named 'summons'.
    - Output:
        - The resulting DataFrames from the analysis steps are written to separate CSV files for further analysis and reporting. The key output files include 'st_df.csv', 'sa_df.csv', 'sc_df.csv', and 'ss_df.csv', which contain information about subway stations, arrests, complaints, and court summonses, respectively.

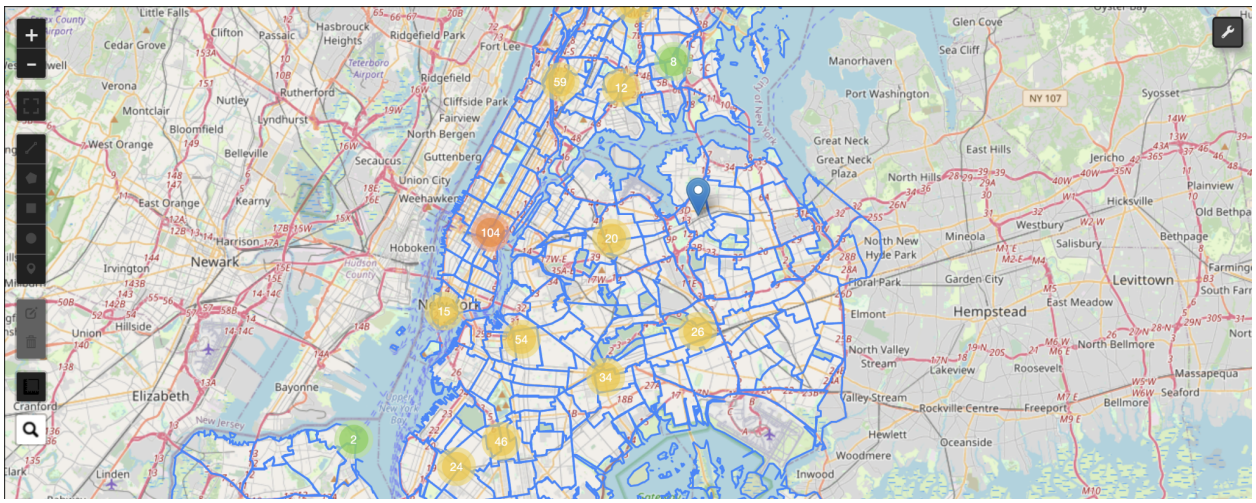
- Spark ML Pipeline:
    - Vector Assembler:
        - Assembler stages utilize the AirBnB price data and creates a new features column based on the latitude and longitude.

    - Random Forest Regressor:
        - The Regression model is built to create a regression equation based on the feature column and the price column. It is trained to evaluate price value based on the geo-coordinates.
    - Cross Validator:
        - Cross validator uses the pipeline containing assembler and regressor along with param grid build using regression to combine and create prediction pipeline. It is trained using the train dataset created by data splitter using 80/20 ratio.
    - Regression Evaluator:
        - Evaluator evaluates the performance of cross validator pipeline model using the test dataset provided by data splitter. It outputs Root Mean Squared Error (RMSE) and accuracy.

- Spark ML Pipeline steps:
    - We focused on predicting rental prices based on the latitude and longitude coordinates of properties. We utilized the Random Forest Regression algorithm implemented in PySpark to train a predictive model.
    - First, we selected the required columns from the rental dataset and filtered out any null values. The selected columns were latitude, longitude, and price. We then converted the longitude and latitude columns to double data type and dropped any remaining null values.
    - Next, we defined the features and target variables for our regression model. The features consisted of latitude and longitude, and the target variable was the rental price.
    - To train the regression model, we created a pipeline that included a VectorAssembler to assemble the features into a vector and the RandomForestRegressor as the regression algorithm.
    - We saved the pipeline for future use and defined a hyperparameter space for tuning the model. We used CrossValidator to perform cross-validation on the training data, evaluating the models using the RegressionEvaluator with the root mean square error (RMSE), mean absolute error (MAE), and R-squared (R2) metrics.
    - We divided the data into training and test sets, and then trained the model using cross-validation on the training data. The best fitted model was obtained from the cross-validation results.
    - We evaluated the performance of the model on the test data by calculating the RMSE, MAE, and R2 scores. These metrics provided insights into the accuracy and goodness-of-fit of the model.
    - Finally, we used the trained model to predict rental prices for the given dataset. The predictions were added as a new column in the DataFrame, and the results were saved in a CSV file for further analysis.

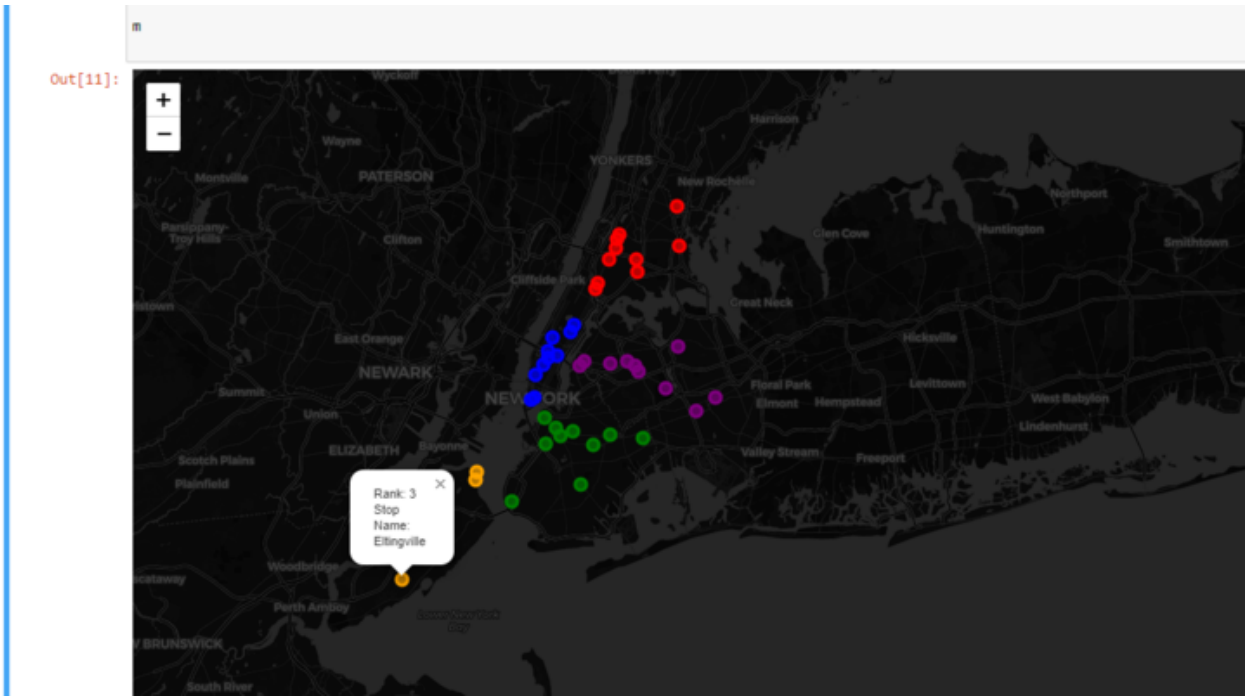
Interactive Mapping

- Once we bound the size of our dataframe based on the number of subway stations in NYC, we made extensive use of pandas, geopandas and leafmap to build an interactive map of NYC.
- We downloaded an NTA shapefile to get a backdrop map of NYC and then converted it to a coordinate reference system (crs) using geopandas so that we could map our subway dataframe directly on the crs.
- Finally, we downloaded demographic data from 2020 and layered that into our map before rendering the final result using leafmap.

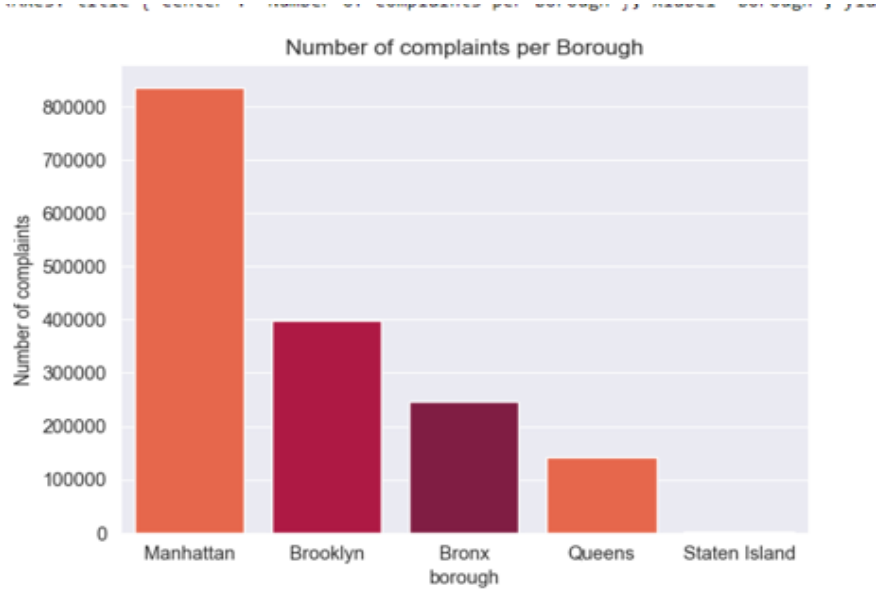


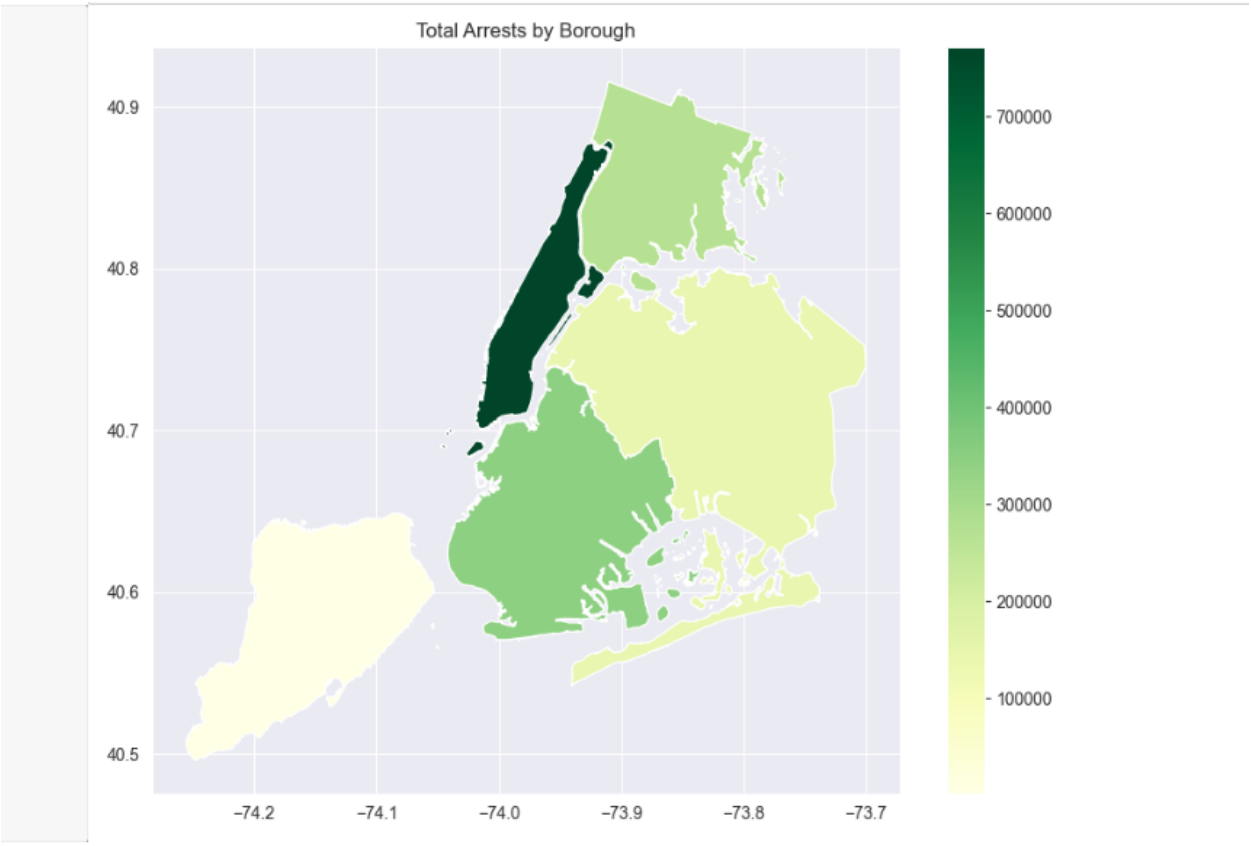
Plots/Charts

The intermediate DataFrames, including the subway-arrest (sa_df), subway-complaint (sc_df), and subway-summons (ss_df), are written to CSV files for further analysis.



Top 10 busiest subway station per borough (in terms of entries)





Number of arrests per borough

Expansion

There are multiple ways to process the results obtained from this project. The price used here are from open sourced airbnb data which use daily rental prices which can be substituted with monthly rent data. The ML regression used here only uses the geolocation to predict price but more parameters like arrests, subway variables, complaints and summons. Furthermore, we can integrate this model into a plugin application which can be incorporated into other applications via Software as a Service (SaaS) platform. As this project follows a big data approach, it is quite easy to expand to multiple geo locations and develop a real-time prediction model based on streaming data.

Conclusion

We set out to build a tool to help new students answer the question - “Where should I live based on rental price, safety, and access to transportation”? In the process, we synthesized 50GB of publicly accessible data and used data from Airbnb to train a regression model and predict median rent near all the subway stations in NYC’s five boroughs. Future NYU Students can use the interactive map to explore locations and compare a suggested rental price with the value of their prospective listing. This gives the student a sense of confidence and stability in order to make an informed decision for a location that fits their budget and safety requirements.

In summary, the project successfully performs data loading, preprocessing, and analysis tasks using PySpark. By leveraging UDFs, inner joins, and data manipulations, it explores the relationship between subway stations and incidents in NYC. The efficient handling of large datasets and the utilization of Spark's capabilities contribute to the project's successful execution. New York City is a vast and dynamic environment, generating a continuous stream of data. By using big data technology, we can easily scale our analysis as new data becomes available. This ensures that our insights remain up to date and applicable over time.

Appendix

Notebook

Setup

Our project made use of the NYC OPEN Data project's API to download five key public datasets: NYC arrests, court summons, complaints, subway stations, and turnstile activity. In addition, a publicly available Airbnb dataset was downloaded as a proxy for rental costs across NYC.
The DataFrames were repartitioned to distribute the data across multiple nodes in the Spark cluster,To improve query performance.

UDF

To assess the proximity of incidents to subway stations, a user-defined function (UDF) called `withinMile` is implemented using PySpark's UDF feature. The `withinMile` UDF utilizes the Haversine formula to calculate the distance between two sets of latitude-longitude coordinates. It returns a boolean value indicating whether the distance is less than 0.1 miles, indicating close proximity between the incident and the subway station.