

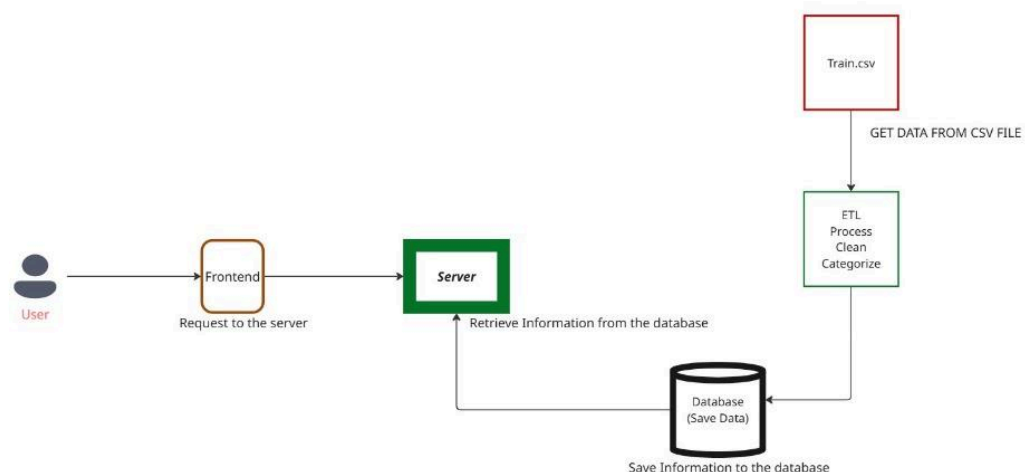
# NYC Taxi Mobility Dashboard

## Project Overview

This project is a full-stack application that can analyze taxi mobility data in the city of New York. It offers an interactive dashboard, trip insights, and a clear reflection of trip information. The users are able to filter their trips according to time, range, and number of passengers.

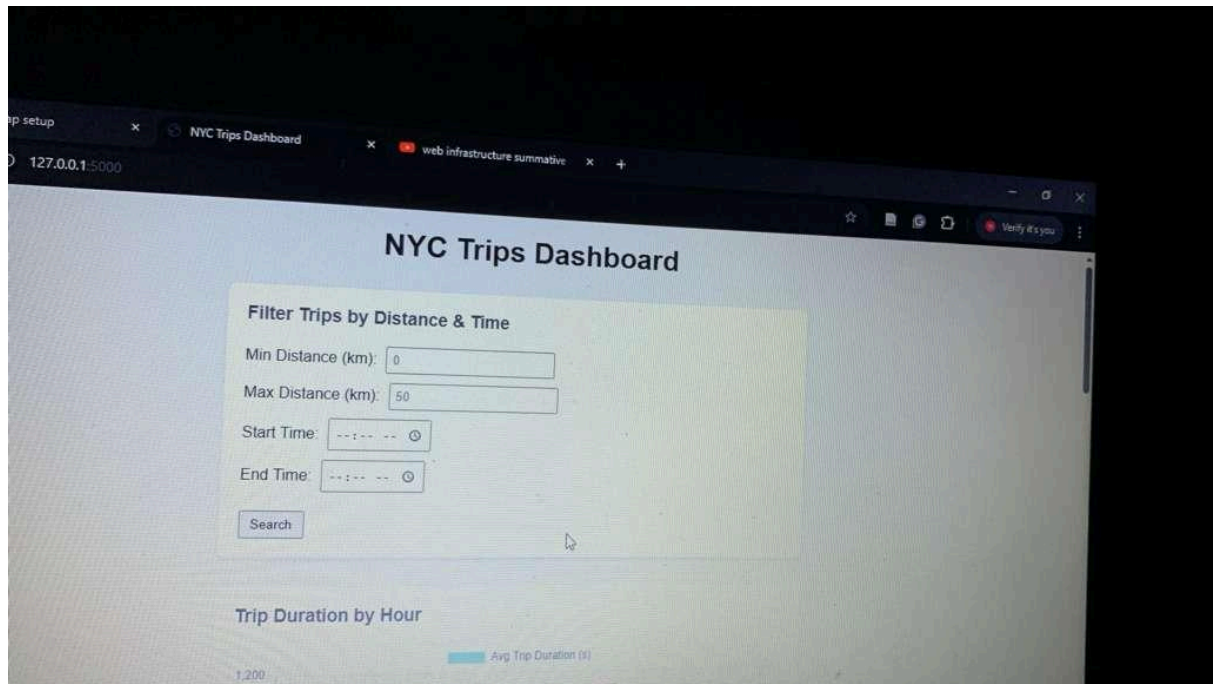
The system is a combination of a front-end program based on HTML, CSS, and JavaScript (Chart.js to visualize data) and a back-end program written in Flask and a SQLite database. The accuracy and reliability of data are guaranteed by data cleaning and preprocessing.

### NYC Taxi Mobility

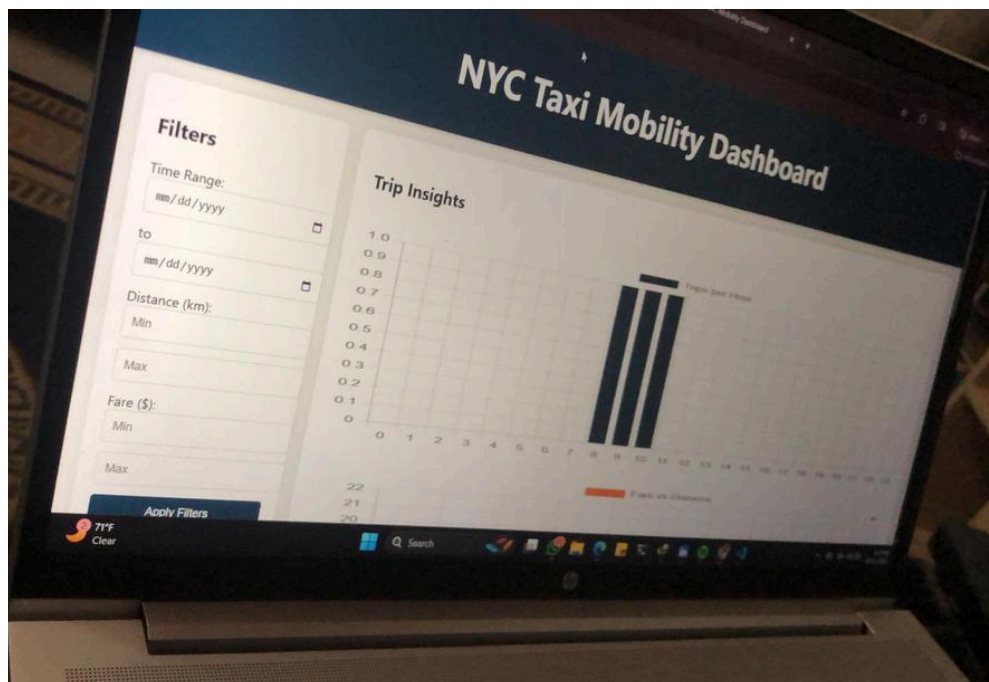


## Frontend

The NYC Taxi Mobility Dashboard was designed and developed to provide a clean, interactive interface for visualizing taxi trip data. The dashboard layout includes a header, sidebar, and main content area, all linked and structured for easy navigation. CSS styling was applied to ensure the dashboard is organized, responsive, and visually balanced.



Using Chart.js, multiple charts were implemented to display key insights such as distance, time. Interactive elements, including buttons like “Filter Data” and “Show Insights,” were added to enhance user engagement. Summary statistics appear dynamically when insights are requested, offering a quick overview of essential data.



During development, issues such as chart rendering errors, file linking problems, and layout misalignment were identified and resolved through iterative testing and code review.

The final version of the dashboard presents NYC Taxi trip data in a simple and intuitive visual format and is fully prepared for integration with the backend and database systems developed by the other team members.

## HTML Structure

The application interface contains:

**Header:** Displays the project title.

**Sidebar:** Filters for time range, distance.

**Main Content:**

- Trip charts (Trips per Hour, Distance) as it shows the busiest hour to be 3pm
- Trip insights (Total trips, Average distance, Busiest hour, Longest/Shortest trip, Average passengers)
- Trip data table

## CSS Styling

The general design of the page was developed in order to give the page a clean and easy-to-use format to navigate trip information. The layout consists of a header, sidebar, and main content section, which is designed in an easy navigation style and a uniform appearance throughout the web. The layout and spacing of the body were defined using CSS, which made the page look modern and balanced.

It is also completely responsive in design, and thus the interface can scale well to a smaller screen, such as a phone or tablet, without compromising the readability or functionality. Moreover, the trip data table was made stylish with distinct borders, space, and slight highlights to make the information readable and decipherable.

With CSS, we focused on this :

- General page styling for body, header, sidebar, and main content
- Responsive design for smaller screens
- Table styling for trip data

## JavaScript Functionality

The frontend was developed to connect with the backend and present trip data in an interactive and meaningful way. It fetches data from the backend endpoints and uses Chart.js to initialize dynamic charts such as Trips per Hour and Distance. The data table is automatically populated with the retrieved information, allowing users to view and interpret results clearly.

The system also applies filters based on user input, making it easy to explore specific time ranges or trip details while generating insights in real time.

## Backend

## **Database Design**

### **1. Data Cleaning**

The trip data cleaning process was developed to ensure accurate and reliable information for analysis. The system validates pickup and drop off coordinates to make sure they fall within the NYC area. Trip distances are calculated using the Haversine formula, providing precise measurements between locations. The process also identifies missing or invalid timestamps and flags trips with nonpositive durations. Duplicate trips are detected by comparing key attributes such as pickup and dropoff times, coordinates, and passenger counts.

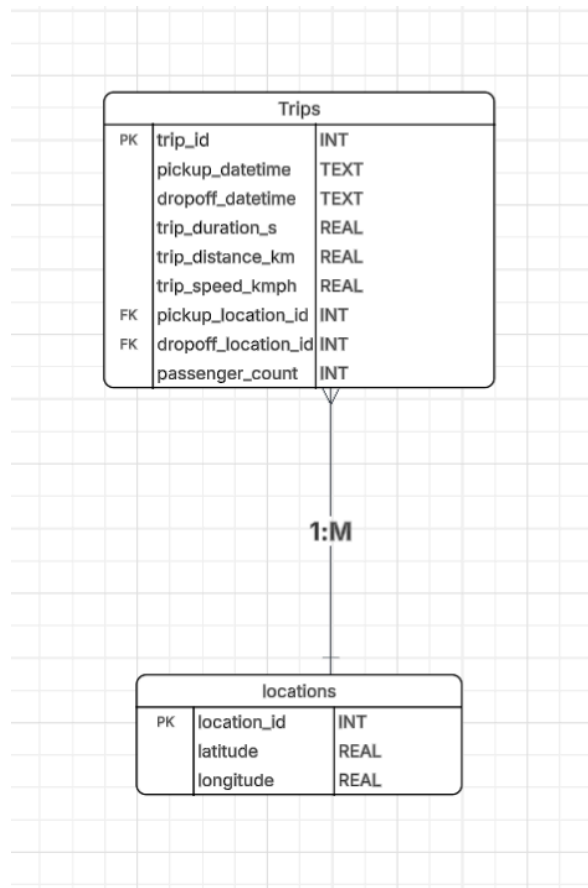
Finally, the workflow generates separate datasets for cleaned trips and excluded trips, making it easy to work with high-quality data.

So what we did in data cleaning was to:

- Validates pickup and dropoff coordinates within NYC bounds
- Computes trip distance using the Haversine formula
- Detects invalid or missing timestamps and nonpositive trip durations
- Filters duplicates based on trip signature (pickup/dropoff datetime, coordinates, passenger count)
- Generates cleaned and excluded trip datasets

### **2. The ERD of Urban Mobility Project**

This structure ensures efficient storage of passenger and location data and enables analysis of trip patterns and travel distances.



This ERD shows the main entities and relationships in the Urban Mobility database:

- **Passengers → Trips (conceptual)**
- Each passenger can take multiple trips.
- Represented by **passenger\_count** in the Trips table.
- **Trips → Locations (pickup and dropoff)**
- Each trip has a pickup location (**pickup\_location\_id**) → Locations table.
- Each trip has a dropoff location (**dropoff\_location\_id**) → Locations table.
- Many trips can share the same location.

- **Relationship**

**Locations → Trips (1:M):** A location can appear as a pickup or dropoff point in many trips.

## 2. Database implementation

I created and populated the database using SQLite commands.  
Here are the steps:

- To build the database from scratch, I ran:

```
sqlite3 database/dump.db
.read schema.sql
.read insert_data.sql
```

- Alternatively, to import the full dump:

```
sqlite3 database/dump.db < dump.sql
```

- After executing these commands, I confirmed the setup using:

```
.tables
```

- Output showed all required tables successfully created.

```
sqlite> .read c:\Users\pc\OneDrive\Desktop\urban-mobility-data-explorer\backend\database\dumb.sql
sqlite> .tables
locations  passengers  payments    trips
```

### 3. Flask App Integration

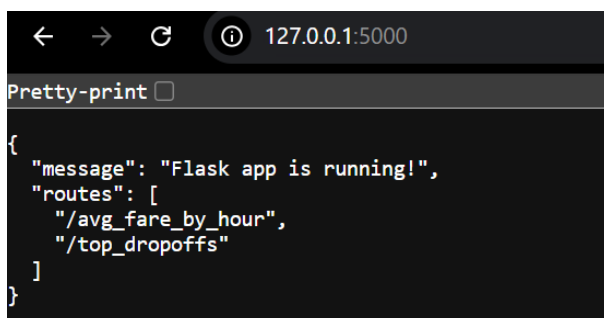
- I connected the Flask backend (app.py) to the database.

```
DB = "database/dump.db"
```

- This is how the Flask app runs in my terminal:

```
(venv) PS C:\Users\pc\OneDrive\Desktop\urban-mobility-data-explorer\backend> python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production
ion WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 252-840-155
127.0.0.1 - - [14/Oct/2025 11:45:46] "GET / HTTP/1.1" 200 -
```

- I tested all endpoints in the browser and confirmed that they returned valid JSON responses:
- /: Home route



The screenshot shows a web browser window with the address bar set to 127.0.0.1:5000. The page displays a JSON response in a dark-themed editor. The response is a dictionary with a 'message' key and a 'routes' key. The 'message' value is 'Flask app is running!'. The 'routes' value is a list containing two strings: '/avg\_fare\_by\_hour' and '/top\_dropoffs'.

```
{
  "message": "Flask app is running!",
  "routes": [
    "/avg_fare_by_hour",
    "/top_dropoffs"
  ]
}
```

- /avg\_fare\_by\_hour: Returns average fare per hour

```
127.0.0.1:5000/avg_fare_by_hour

Pretty-print ☐

[
  {
    "avg_fare": 12.0,
    "hour": "15",
    "trips_count": 3
  }
]
```

- **/top\_dropoffs**: Returns top dropoff locations by frequency

```
127.0.0.1:5000/top_dropoffs

Pretty-print ☐

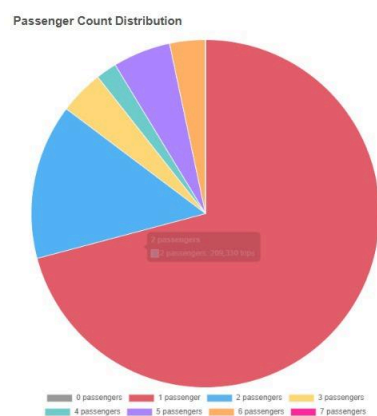
[
  {
    "avg_dist_km": 5.0,
    "avg_fare": 12.0,
    "location_id": 9999,
    "location_label": "Lat: -1.94000, Lon: 30.06000",
    "trips_count": 3
  }
]
```

## 4. Insights and Interpretation

### ➤ Passenger Count Distribution

Most trips are taken by a single person:

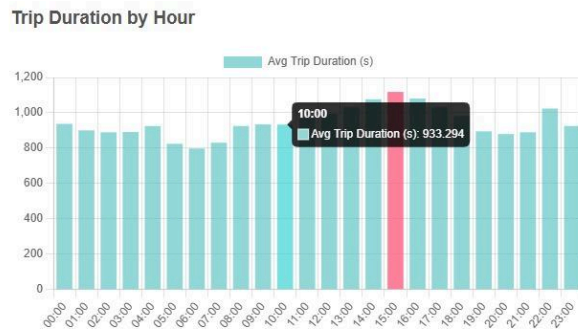
- The vast majority of trips (the red part) have only 1 passenger.
- The next biggest group (blue part) has 2 passengers (209,330 trips).
- All other passenger counts (0, 3, 4, 5, 6, 7) make up a very small percentage of total trips.



### ➤ Trip Duration by Hour

This shows the average time a trip takes throughout the day:

- Average trip duration is usually between **800 and 1,000 seconds**.
- The **longest average trips** happen around **3:00 PM (15:00)**.
- Trips are generally shortest in the early morning (around 6:00 AM and 7:00 AM).
- At **10:00 AM**, the average trip was **933.3 seconds**.



### ➤ Vendor Performance

This compares the total number of trips for two vendors:

- Vendor 2 completed more total trips than Vendor 1.
- Vendor 2 had 775,946 trips.
- Vendor 1's trips were slightly less, around 650,000 to 700,000.



## 5. Challenge and Solution

**Challenge:** PowerShell blocked virtual environment activation

**Solution:** Used `Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass`

## Data Insertion Script

The data insertion script was created to populate the database with sample information, including passengers, locations, and trips. It ensures that a demo trip is always present,



providing a reliable example for testing and development. Each entry is carefully linked so that the trip connects a passenger, and pickup and dropoff locations, making it easy to test queries, filters, and visualizations without relying on real-world data.

This is what it does:

- Inserts sample data into passengers, locations, and trips tables
- Ensures demo trip exists for testing

## **Reflection and Future Work**

### **Reflection**

Building the NYC Taxi Mobility Dashboard came with both technical and teamwork challenges. Handling inconsistent trip data, linking frontend charts with backend APIs, and making filters responsive were tricky. Collaborating across frontend, backend, and database work required good communication and version control.

### **Future Work**

For a real-world version, we could:

- Optimize performance with caching and better database indexing.
- Add more insights like peak traffic trends and predictive fares.
- Improve the user experience with interactive maps and exportable reports.
- Make the backend more scalable for larger datasets.
- Streamline teamwork with automated testing and CI/CD pipelines.

These steps would help turn the prototype into a fully usable, scalable product.

## **Conclusion**

This project provides a comprehensive tool for visualizing and analyzing NYC taxi mobility data. It integrates front-end interactivity, backend data handling, and rigorous data cleaning. Users can explore trends, insights, and trip details effectively.