NYC Urban Mobility Data Explorer

1. Project Overview

This project analyzes New York City taxi trip data. The goal is to clean raw taxi data, store it in a database, and make it available through an API and dashboard so users can understand urban mobility patterns.

This document clearly show: - How data is loaded - How data is cleaned - How data is stored in the database - How the API works - What screenshots to include for proof

---

2. Project Structure

urban-mobility-data-explorer/

```
│
├── backend/
│   │
│   ├── data_pipeline/
│   │   ├── load_data.py        # Loads raw NYC taxi data (CSV / Parquet)
│   │   ├── clean_data.py       # Cleans and validates the data
│   │   └── rejection_log.py    # Logs rejected/invalid records
│   │
│   ├── api/
│   │   └── routes.py           # API endpoints (send data to frontend)
│   │
│   └── database/
│       └── db_connection.py    # Connects Python to the database
│
├── frontend/
│   ├── index.html              # Main web page (UI)
│   ├── app.js                  # JavaScript logic (fetch data, charts)
```

```
│   └── style.css            # Page styling (colors, layout)
│
└── README.md                # Project explanation (optional but recommended)
```

---

3. Loading the Data

What happens here?

- The raw NYC taxi CSV file is read using Python
- The file is checked to make sure it exists
- The data is passed to the cleaning step

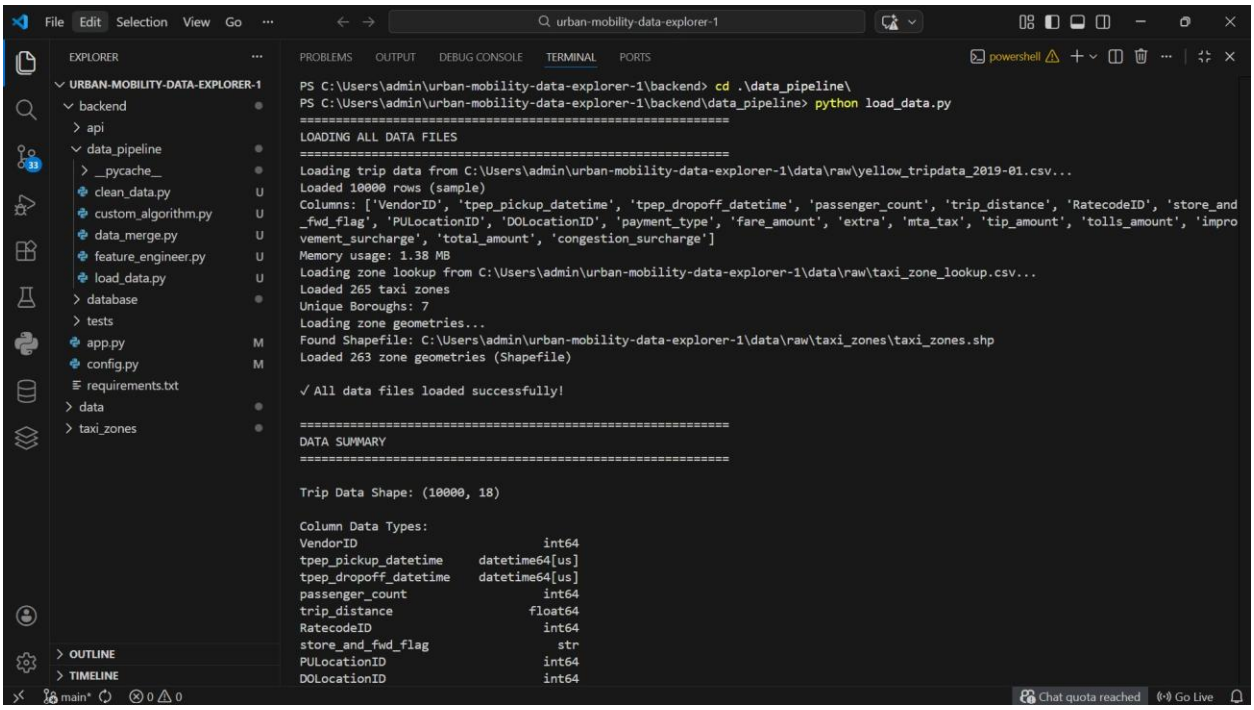
Script Used

`load_data.py`

Screenshot to include □

- VS Code terminal showing:

```
python -m backend.data_pipeline.load_data
```



- Terminal output confirming data loaded successfully

---

4. Cleaning the Data

Why cleaning is needed

Raw taxi data contains: - Missing values - Invalid trip distances - Wrong timestamps

Cleaning rules applied

- Remove rows with missing pickup or dropoff time
- Remove trips with distance ≤ 0
- Remove trips with duration ≤ 0
- Log rejected rows into a rejection file

Script Used

`clean_data.py`

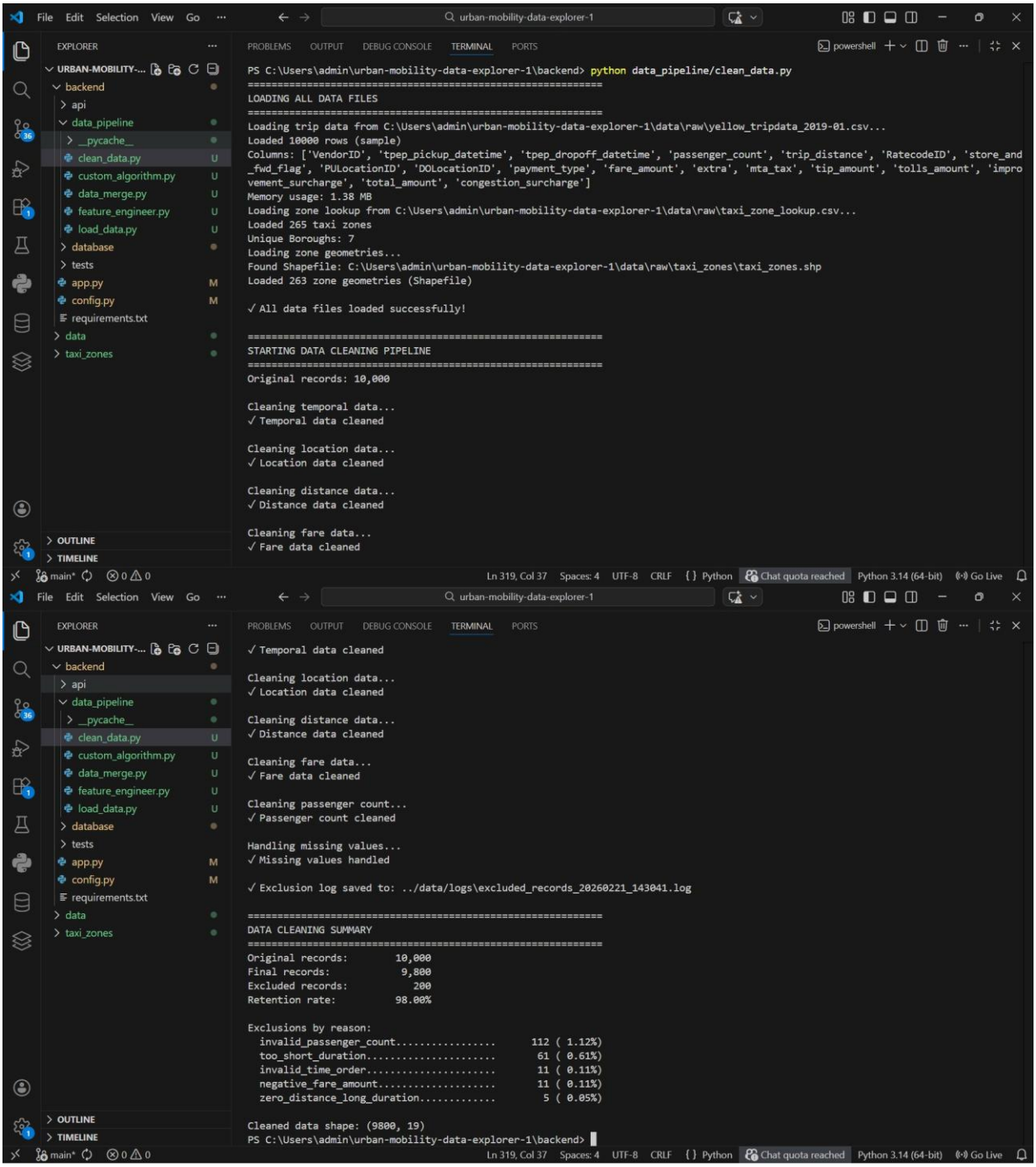
How to know data is cleaned ✅

- Cleaned data file is created
- Rejected rows are saved in a log file
- Terminal shows a success message

Screenshot to include

- Terminal showing:

```
python -m backend.data_pipeline.clean_data
```



- File explorer showing cleaned dataset file
- Rejection log file

---

5. Saving Data to the Database

What happens here?

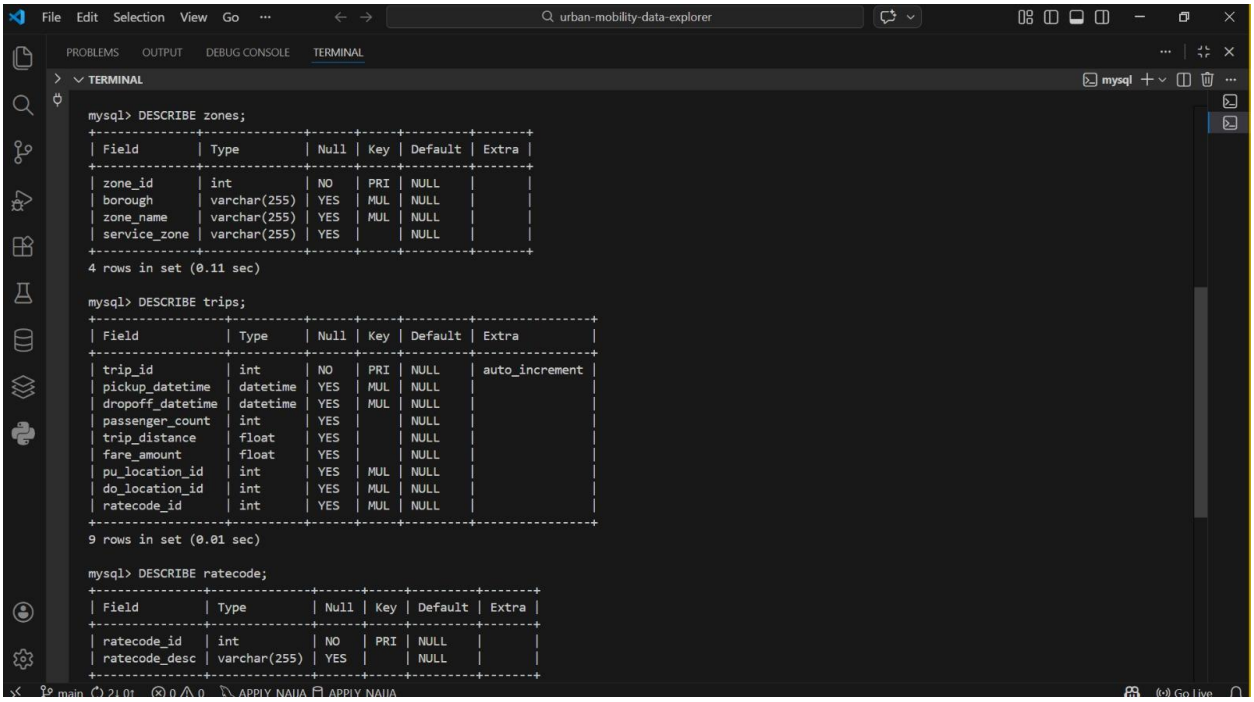
- Cleaned data is inserted into the database
- Tables are created if they do not exist

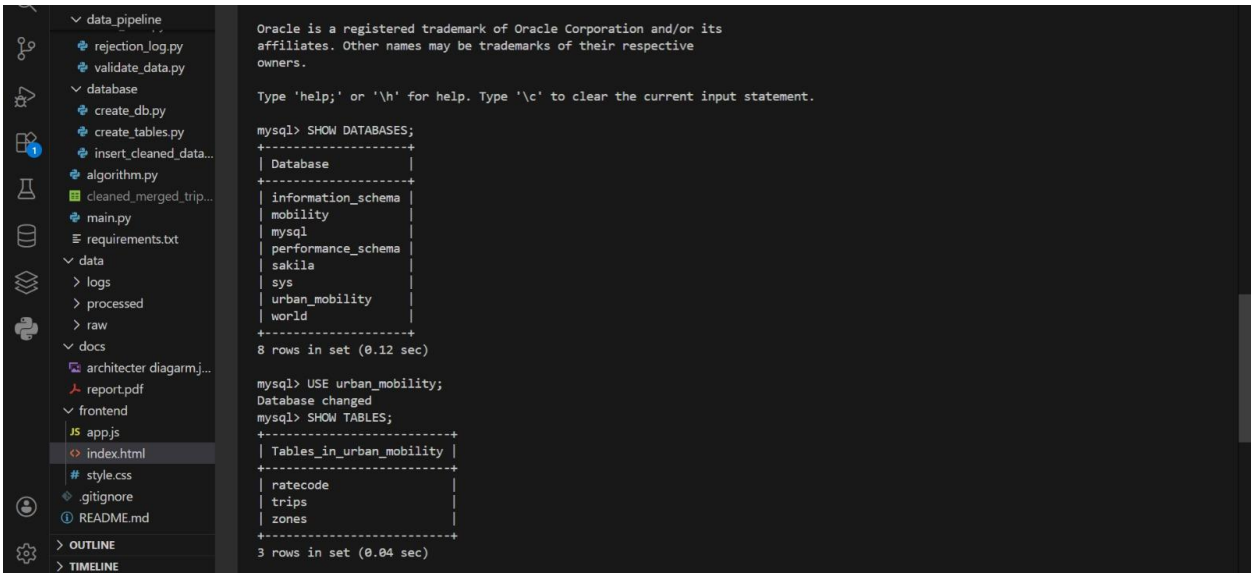
Database Used

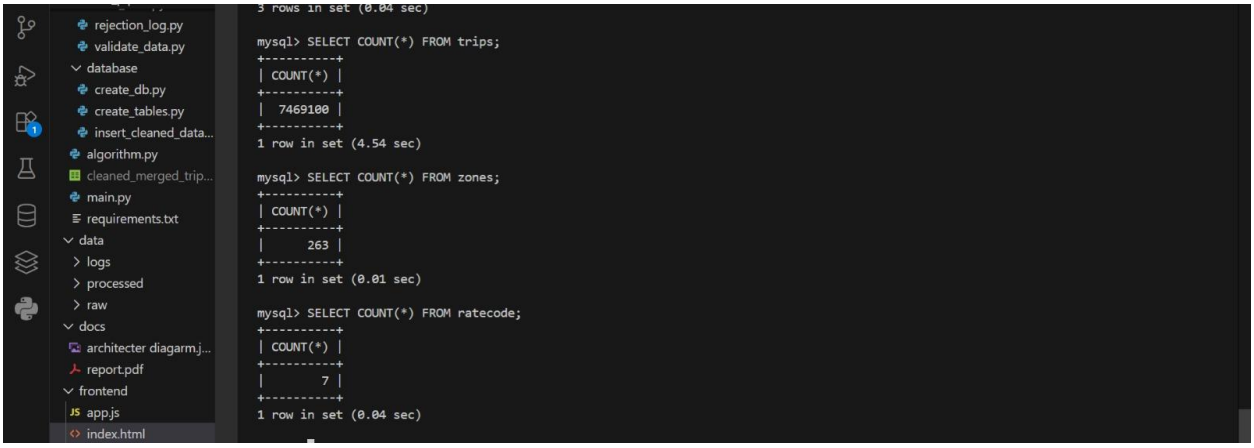
- MySQL

Screenshot to include □

- MySQL terminal showing database and tables data







---

6. API Design (Simple Explanation)

What is the API for?

The API allows other systems (like the dashboard) to request cleaned taxi data.

Main API Endpoints

Endpoint	Purpose
`/trips`	Get all taxi trips
`/trips?date=YYYY-MM-DD`	Get trips for a specific day
`/stats`	Get summary statistics

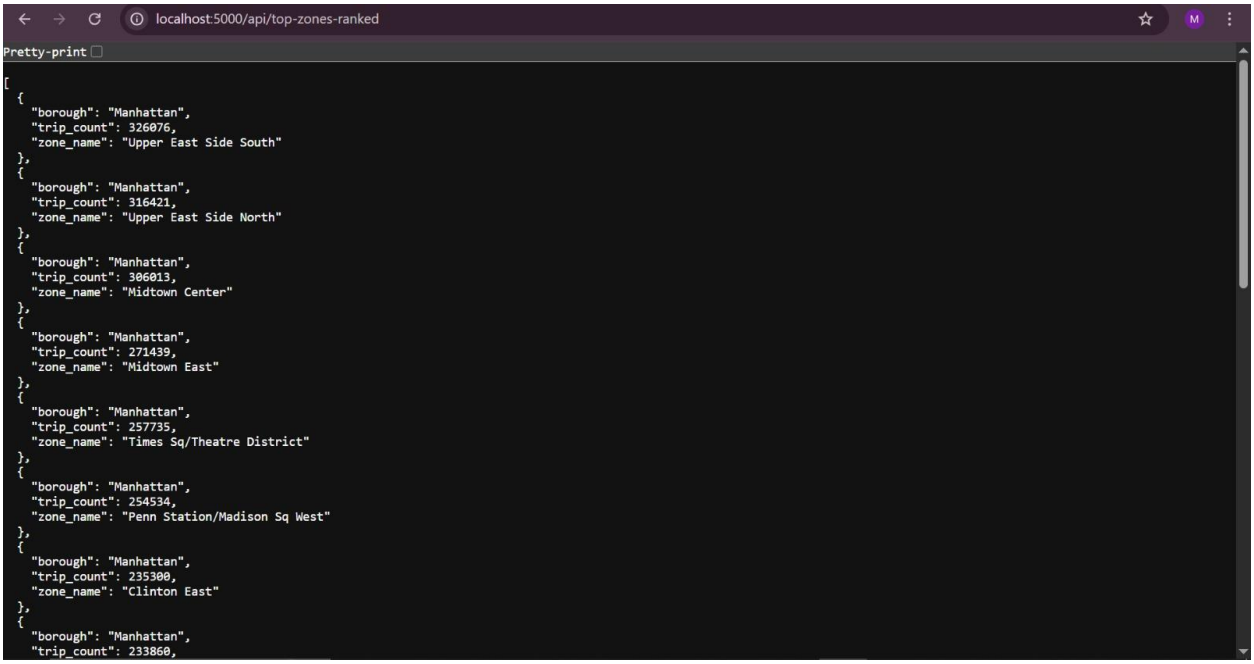
Simple API Algorithm

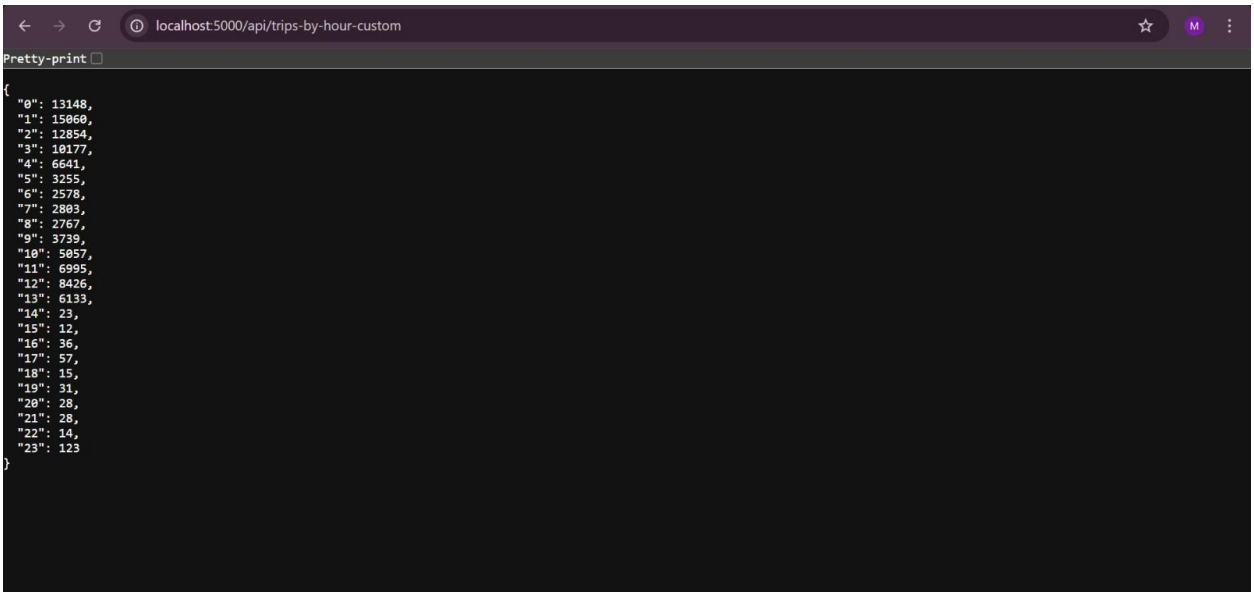
1. User sends request
2. API receives request

3. API queries database
4. API returns JSON response

Screenshot to include □

- Browser or Postman showing API response
- JSON data displayed





7. Dashboard (Optional Screenshot)

If a dashboard is used: - Shows number of trips - Shows average distance - Shows busiest hours

Screenshot to include □

- Dashboard with charts

---

8. Proof Checklist (Very Important)

Include screenshots of: - ✓ Terminal running load script - ✓ Terminal running clean script - ✓ Cleaned data file - ✓ Rejection log file - ✓ Database tables - ✓ API response in browser/Postman

---

9. Conclusion

This project demonstrates a complete data pipeline: - Data loading - Data cleaning - Data storage - Data access through an API

All steps are automated using Python and follow real-world data engineering practices.