

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

```

PS C:\Users\admin\urban-mobility-data-explorer-1\backend> cd .\data_pipeline\
PS C:\Users\admin\urban-mobility-data-explorer-1\backend\data_pipeline> python load_data.py
=====
LOADING ALL DATA FILES
=====
Loading trip data from C:\Users\admin\urban-mobility-data-explorer-1\data\raw\yellow_tripdata_2019-01.csv...
Loaded 10000 rows (sample)
Columns: ['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime', 'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag', 'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge']
Memory usage: 1.38 MB
Loading zone lookup from C:\Users\admin\urban-mobility-data-explorer-1\data\raw\taxi_zone_lookup.csv...
Loaded 265 taxi zones
Unique Boroughs: 7
Loading zone geometries...
Found Shapefile: C:\Users\admin\urban-mobility-data-explorer-1\data\raw\taxi_zones\taxi_zones.shp
Loaded 263 zone geometries (Shapefile)

✓ All data files loaded successfully!

=====
DATA SUMMARY
=====

Trip Data Shape: (10000, 18)

Column Data Types:
VendorID                int64
tpep_pickup_datetime    datetime64[us]
tpep_dropoff_datetime   datetime64[us]
passenger_count         int64
trip_distance            float64
RatecodeID              int64
store_and_fwd_flag      str
PULocationID            int64
DOLocationID            int64

```

- 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
```

```
PS C:\Users\admin\urban-mobility-data-explorer-1\backend> python data_pipeline/clean_data.py
=====
LOADING ALL DATA FILES
=====
Loading trip data from C:\Users\admin\urban-mobility-data-explorer-1\data\raw\yellow_tripdata_2019-01.csv...
Loaded 10000 rows (sample)
Columns: ['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime', 'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag', 'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge']
Memory usage: 1.38 MB
Loading zone lookup from C:\Users\admin\urban-mobility-data-explorer-1\data\raw\taxi_zone_lookup.csv...
Loaded 265 taxi zones
Unique Boroughs: 7
Loading zone geometries...
Found Shapefile: C:\Users\admin\urban-mobility-data-explorer-1\data\raw\taxi_zones\taxi_zones.shp
Loaded 263 zone geometries (Shapefile)
✓ All data files loaded successfully!

=====
STARTING DATA CLEANING PIPELINE
=====
Original records: 10,000

Cleaning temporal data...
✓ Temporal data cleaned

Cleaning location data...
✓ Location data cleaned

Cleaning distance data...
✓ Distance data cleaned

Cleaning fare data...
✓ Fare data cleaned

Cleaning passenger count...
✓ Passenger count cleaned

Handling missing values...
✓ Missing values handled

✓ Exclusion log saved to: ../data/logs/excluded_records_20260221_143041.log

=====
DATA CLEANING SUMMARY
=====
Original records:      10,000
Final records:         9,800
Excluded records:       200
Retention rate:        98.00%

Exclusions by reason:
invalid_passenger_count..... 112 ( 1.12%)
too_short_duration..... 61 ( 0.61%)
invalid_time_order..... 11 ( 0.11%)
negative_fare_amount..... 11 ( 0.11%)
zero_distance_long_duration..... 5 ( 0.05%)

Cleaned data shape: (9800, 19)
PS C:\Users\admin\urban-mobility-data-explorer-1\backend>
```

- File explorer showing cleaned dataset file
- Rejection log file

5. Saving Data to the Database

What happens here?

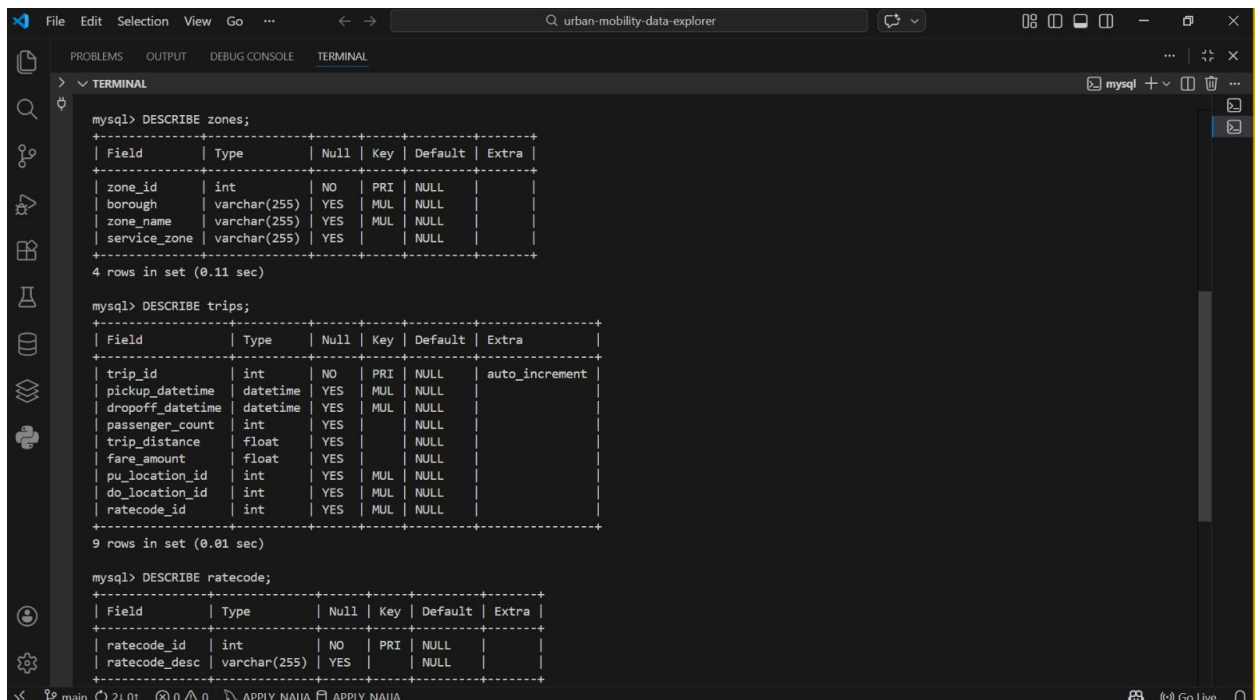
- Cleaned data is inserted into the database
- Tables are created if they do not exist

Database Used

- MySQL

Screenshot to include ☐

- MySQL terminal showing database and tables data



```
mysql> DESCRIBE zones;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| zone_id    | int       | NO   | PRI | NULL    |       |
| borough    | varchar(255) | YES  | MUL | NULL    |       |
| zone_name  | varchar(255) | YES  | MUL | NULL    |       |
| service_zone | varchar(255) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.11 sec)

mysql> DESCRIBE trips;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| trip_id    | int       | NO   | PRI | NULL    | auto_increment |
| pickup_datetime | datetime | YES  | MUL | NULL    |       |
| dropoff_datetime | datetime | YES  | MUL | NULL    |       |
| passenger_count | int     | YES  |     | NULL    |       |
| trip_distance | float    | YES  |     | NULL    |       |
| fare_amount | float    | YES  |     | NULL    |       |
| pu_location_id | int     | YES  | MUL | NULL    |       |
| do_location_id | int     | YES  | MUL | NULL    |       |
| ratecode_id | int     | YES  | MUL | NULL    |       |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.01 sec)

mysql> DESCRIBE ratecode;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ratecode_id | int       | NO   | PRI | NULL    |       |
| ratecode_desc | varchar(255) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mobility |
| mysql |
| performance_schema |
| sakila |
| sys |
| urban_mobility |
| world |
+-----+
8 rows in set (0.12 sec)

mysql> USE urban_mobility;
Database changed
mysql> SHOW TABLES;
+-----+
| Tables_in_urban_mobility |
+-----+
| ratecode |
| trips |
| zones |
+-----+
3 rows in set (0.04 sec)
```

```
mysql> SELECT COUNT(*) FROM trips;
+-----+
| COUNT(*) |
+-----+
| 7469100 |
+-----+
1 row in set (4.54 sec)

mysql> SELECT COUNT(*) FROM zones;
+-----+
| COUNT(*) |
+-----+
| 263 |
+-----+
1 row in set (0.01 sec)

mysql> SELECT COUNT(*) FROM ratecode;
+-----+
| COUNT(*) |
+-----+
| 7 |
+-----+
1 row in set (0.04 sec)
```

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

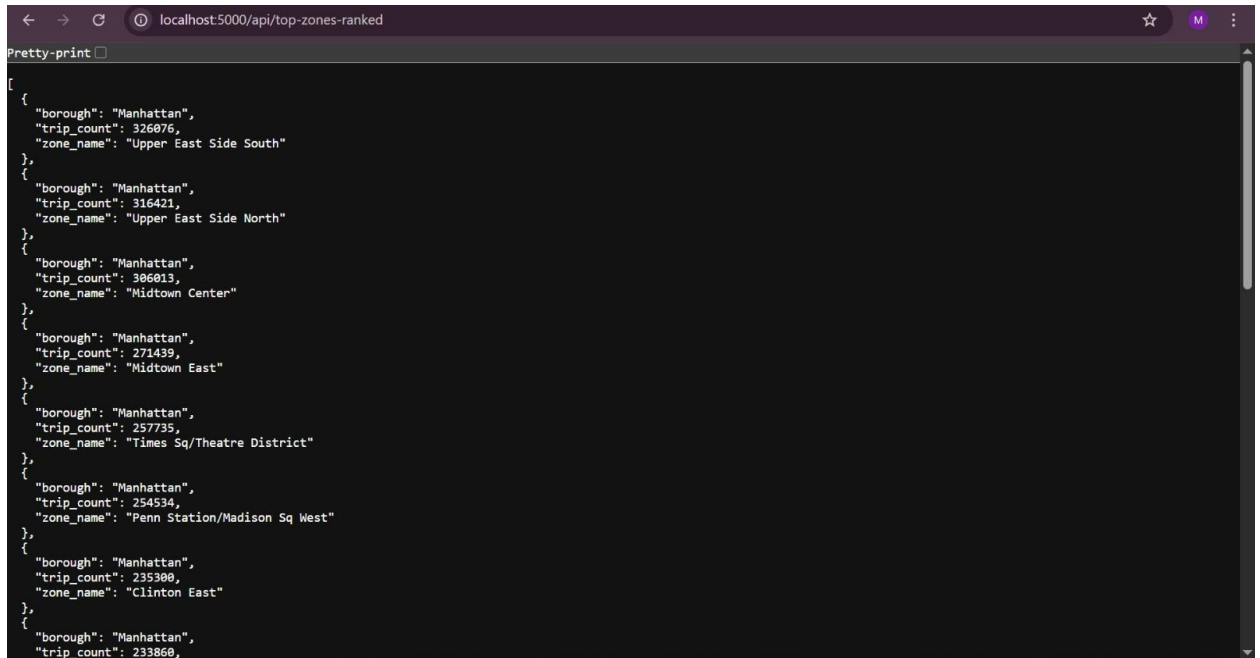
Simple API Algorithm

1. User sends request
2. API receives request

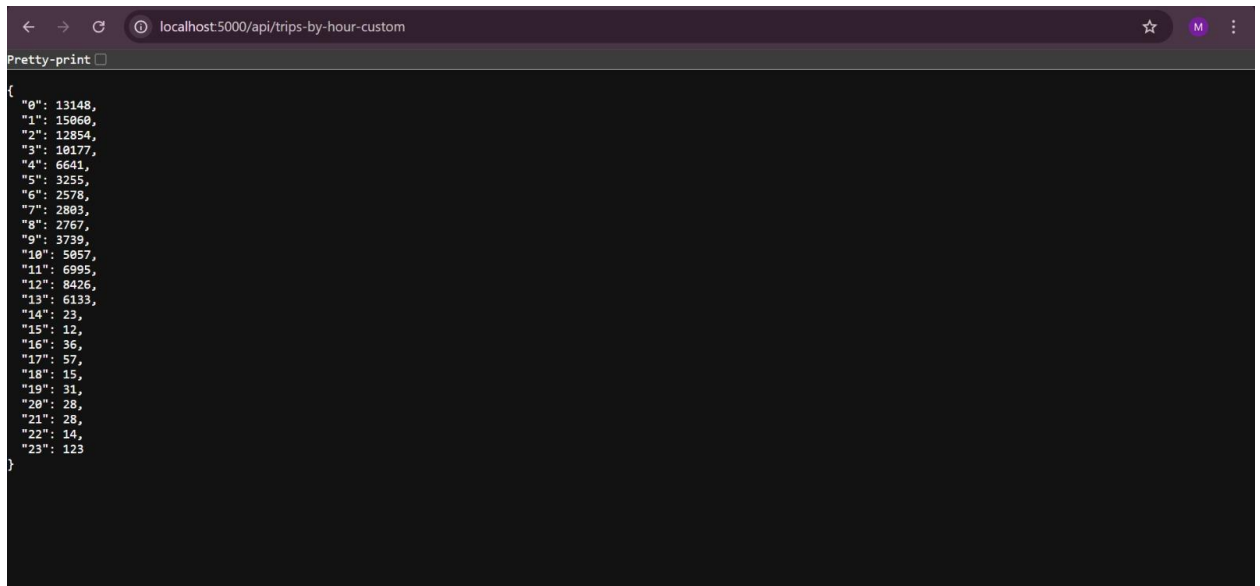
3. API queries database
4. API returns JSON response

Screenshot to include ☐

- Browser or Postman showing API response
- JSON data displayed



```
localhost:5000/api/top-zones-ranked
Pretty-print
[
  {
    "borough": "Manhattan",
    "trip_count": 326076,
    "zone_name": "Upper East Side South"
  },
  {
    "borough": "Manhattan",
    "trip_count": 316421,
    "zone_name": "Upper East Side North"
  },
  {
    "borough": "Manhattan",
    "trip_count": 306013,
    "zone_name": "Midtown Center"
  },
  {
    "borough": "Manhattan",
    "trip_count": 271439,
    "zone_name": "Midtown East"
  },
  {
    "borough": "Manhattan",
    "trip_count": 257735,
    "zone_name": "Times Sq/Theatre District"
  },
  {
    "borough": "Manhattan",
    "trip_count": 254534,
    "zone_name": "Penn Station/Madison Sq West"
  },
  {
    "borough": "Manhattan",
    "trip_count": 235300,
    "zone_name": "Clinton East"
  },
  {
    "borough": "Manhattan",
    "trip_count": 233860,
    "zone_name": "Clinton West"
  }
]
```



```
localhost:5000/api/trips-by-hour-custom
Pretty-print
{
  "0": 13148,
  "1": 15060,
  "2": 12854,
  "3": 10177,
  "4": 6641,
  "5": 3255,
  "6": 2578,
  "7": 2803,
  "8": 2767,
  "9": 3739,
  "10": 5057,
  "11": 6995,
  "12": 8426,
  "13": 6133,
  "14": 23,
  "15": 12,
  "16": 36,
  "17": 57,
  "18": 15,
  "19": 31,
  "20": 28,
  "21": 28,
  "22": 14,
  "23": 123
}
```

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.