Urban_Mobility_Report_EWD_Group_2.pdf.

Team Name: EWD_Group_2

Course: Enterprise Web Development

1. Problem Framing and Dataset Analysis

• The dataset and its context

The New York Taxi City Trip Dataset contains millions of individual trip records gathered from yellow taxis operating throughout NYC. Each entry captures details such as pickup and drop-off times, geographic coordinates (latitude and longitude), trip distances, fare amounts, passenger counts, and payment types. This dataset offers an in-depth understanding of urban mobility patterns, providing valuable insights into city traffic flow, pricing trends, and travel behaviours.

Data Challenges

- During preprocessing, several data quality issues were identified.
- Missing fields in passenger count, trip distance, and fare amounts.
- Outliers such as trips with zero duration, negative fares, and unrealistic distances.
- Duplicate entries and invalid coordinates outside New York's geographic boundaries.

Assumptions Made.

- Trips with zero or negative values for distance or duration were considered invalid and excluded.
- Missing passenger counts were filled with the most frequent value (1).
- Coordinates were assumed valid only if within NYC's latitude and longitude range.
- All timestamps were normalized to UTC and formatted as ISO strings.

Unexpected Observation

An unexpected trend was that Vendor 2 consistently recorded slightly higher average speeds than Vendor 1, despite having more total trips. This insight influenced the dashboard design to include a vendor performance comparison chart for better interpretation of vendor-level differences.

2. System Architecture and Design Decisions

System architecture diagram (Frontend, Backend, Database).

Raw CSV (Data/raw/train.csv) —> Backend/data_cleaning.py (Pandas ETL) —> Cleaned CSV (Data/cleaned/train_cleaned.csv) —-> Backend/init_db.py / Backend/load_data_to_db.py (create table / load) —--> SQLite DB (Backend/urban_mobility.db) —->Backend/app.py (Flask API + serves Frontend) —->Frontend (index.html + script.js) <---> User Browser (requests + AJAX)

Stack Justification

- Python + pandas: fast CSV ETL and feature engineering
- Flask: minimal API + static file server
- SQLite: zero-config, local file DB for demos.

Schema Structure

- Purpose: one denormalized trips table for analytics (simple reads, dashboards).
- Core columns: vendor_id, pickup_datetime, dropoff_datetime, passenger_count, pickup/dropoff coordinates, trip_duration_min, trip_distance_km, speed_kmh.
- Indexes & rationale: index pickup_datetime and vendor_id to make time-series and vendor aggregation fast.
- Design choice: store derived fields (duration, distance km, speed) so dashboards query quickly without reprocessing raw CSVs.

• Trade-offs

- SQLite was used for testing due to its simplicity but would need migration (e,g PostgreSQL/MySQL) for higher concurrency and faster indexing.
- Pre-computation for derived features (e.g, trip_duration_min, speed_kmh) reduced frontend latency but increased preprocessing time and requires re-running the ETL when logic changes.
- Keeping data cleaning separate from API (data_cleaning.py, load_data_to_db.py, schema.py) improved modularity and debugging at the cost of an extra pipeline step.

3. Algorithmic Logic and Data Structures

Custom algorithm or data structure implemented manually

Sliding-window ring buffer maintaining sum and sum-of-squares for online mean/std (no built-ins like sort, heapq, Counter, etc.).

Problem: Detect and drop/flag local outliers (e.g., speed_kmh or trip_duration_min) during chunked ETL so you can filter noisy GPS/glitch records without loading the entire CSV into memory.

Pseudo-code :

```
// check outlier against current window

m = window.sum / window.count

var = (window.sumsq - (window.sum * window.sum) / window.count) / window.count

sd = sqrt(max(var, 0))

if sd > 0 and abs(x - m) > k * sd:     // k is z-score threshold, e.g., 3.0

    drop_or_flag(row)                // treat as outlier

else:

    window.push(x)                   // include in window
```

Complexity Analysis: Time Complexity O(n), Space Complexity O(w).

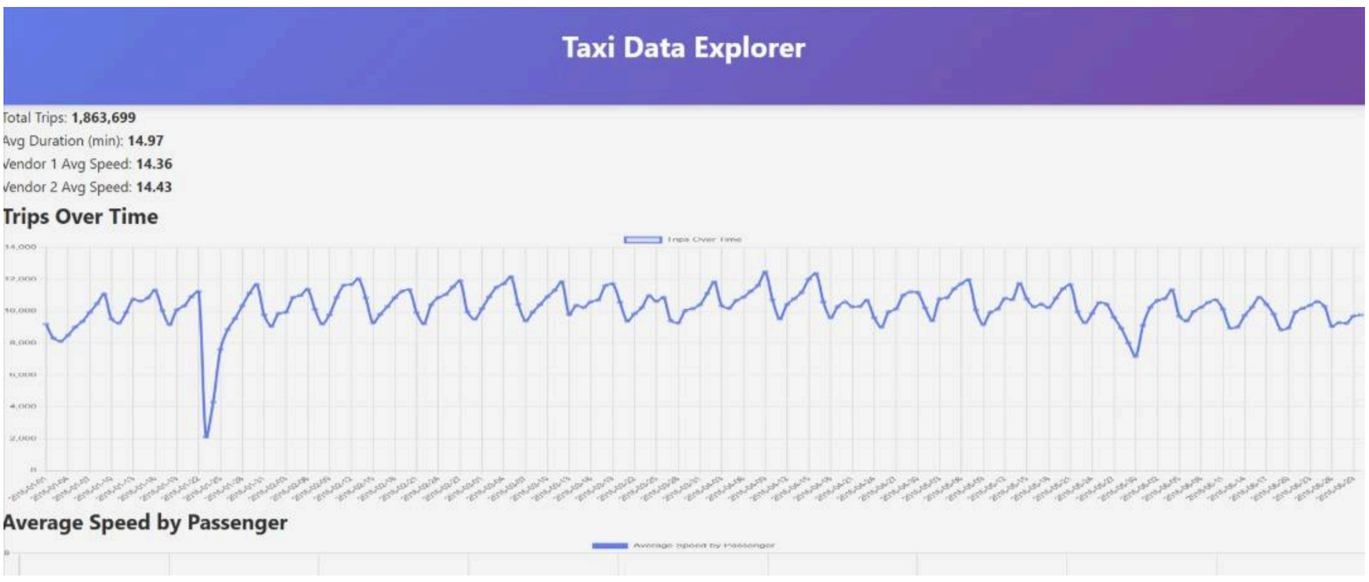
- Works online with bounded memory, no global sort/aggregation.
- Detects local abnormalities only (window and k choice matter).
- Use double precision and occasional rebase if numeric drift appears.

4. Insights and Interpretation

Insight	How Derived (Query / Visualization	Interpretation /
1. Peak demand hours	SQL or pandas group-by on pickup_datetime truncated to hour, count trips per hour, sort descending.	identifies commute peaks for dispatching resources or pricing; caveat: weekday vs weekend differences.
2. Average speed by time of day	compute speed_kmh per trip, group by hour, take mean and median; optionally exclude outliers.	lower speeds during X–Y indicate congestion; informs routing, ETA models, or policy evaluation
3. Short-trip concentration	filter trips with trip_distance_km < 1, aggregate by neighborhood or grid cell.	indicates areas with high short-trip demand (first/last mile, public-transit gaps) or areas to target for more vehicles.

Visualization Screenshot

Average speed by passenger


Taxi Data Explorer

Total Trips: **1,863,699**
Avg Duration (min): **14.97**
Vendor 1 Avg Speed: **14.36**
Vendor 2 Avg Speed: **14.43**

Trips Over Time

Average Speed by Passenger

5. Reflection and Future Work

Technical challenges

- Data load & performance
- Data quality & anomalies
- API / runtime fragility

Team challenges

- Large Dataset unable to push to github with a simple git push command
- Each team had to ignore the train.csv files and other large files
- CSV columns change or people add derived fields in different places.
- Duplicated logic & responsibilities
- Different local environments, no container, and missing tests make it hard for new contributors to run the project

Lessons Learned

- Validate early and often
- Keep logic modular and single-responsibility
- Add metrics (processing time, row counts, outlier rates) and logs to quickly find performance or correctness regressions.
- That github has size limit for free users

Future Improvements

If deployed in a real-world environment, the system could be improved by:

- Integrating real-time trip data from the NYC Taxi API.
- Enhancing data validation using machine learning anomaly detection.
- Expanding to ride-hailing services for comparative insights.
- Automating daily ETL pipelines with Airflow or similar tools for scalability.