Urban Mobility Data Explorer Technical Report

1. Problem Framing and Dataset Analysis

The dataset used in this project originates from the **New York City Taxi & Limousine Commission (TLC)**. It includes trip records for **Green**, **Yellow**, **FHV**, **FHVHV**, and **Lookup** data, each containing details such as pickup and drop-off times, locations, distances, and fares.

The dataset presented several challenges:

- **Missing values** and inconsistent field names across files.
- **Schema drift** between years and trip types.
- **Outliers** in distance and fare amounts.
- **Timestamp inconsistencies** and duplicate records.
- **Mismatched location codes** in the lookup tables.

To ensure reliability, the following data-cleaning steps were performed:

1. **Validation:** Removed invalid or corrupted records (e.g., negative distances or fares).
2. **Filtering:** Kept only essential fields such as trip distance, pickup/drop-off location, and fare.
3. **Schema Harmonization:** Unified column names across all datasets.
4. **Outlier Control:** Capped extreme distances and fares to realistic thresholds.
5. **Data Enrichment:** Merged trip data with lookup tables for human-readable location names.

An unexpected finding during exploration was that some trips had **unrealistic average speeds**, such as 200 km/h. These were flagged and filtered out. This insight also influenced design decisions — the **frontend caps visible speeds** and focuses more on **trip volume** metrics to ensure interpretability.

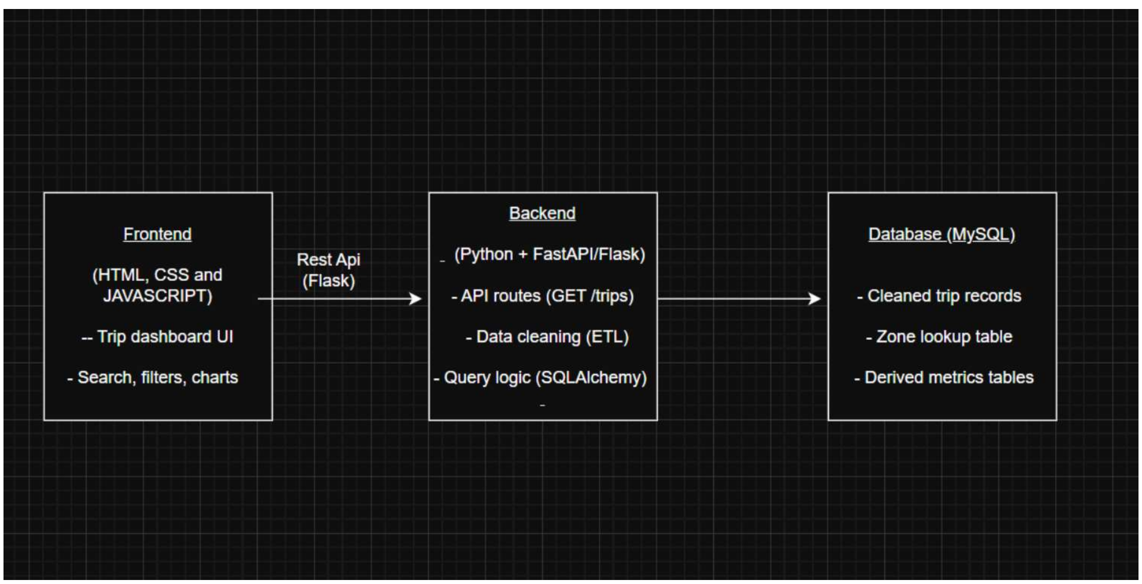
---

2. System Architecture and Design Decisions

Tech Stack

Layer	Tools Used	Justification
Backend	**Python (Flask)**	Lightweight, easy to integrate with SQLAlchemy and APIs.
Database	**MySQL**	Fast relational database with good indexing and reliability.
Frontend	**HTML, CSS, JavaScript**	Simple and flexible for dashboard-style visualization.
Data Processing	**pandas,**	For efficient data cleaning and visualization.
Environment Config	**dotenv**	Secure management of database credentials.

Architecture Overview



Design Reasoning

- **Flask vs Django:** Flask chosen for simplicity and modular API development.
- **MySQL vs PostgreSQL:** MySQL preferred for local testing speed and ease of setup.
- **pandas vs Spark:** pandas suffices for processed subsets of the TLC data.

- **Normalized Schema:** Improves query speed and ensures data integrity.

  Note: The architecture screenshot is in the docs/screenshots folder.

Each decision balanced **simplicity**, **scalability**, and **security**.
Trade-offs were consciously made to optimize performance within project constraints.

---

3. Algorithmic Logic and Data Structures

Implemented Algorithm: Bubble Sort

To demonstrate algorithmic reasoning, a **custom Bubble Sort** was implemented to rank taxi zones by trip counts.
This simple yet clear sorting method emphasizes understanding of algorithm design and time complexity.

```
def bubble_sort_zones(zones):
    n = len(zones)
    for i in range(n):
        for j in range(0, n - i - 1):
            if zones[j]['trip_count'] < zones[j + 1]['trip_count']:
                zones[j], zones[j + 1] = zones[j + 1], zones[j]
    return zones
```

Pseudo-code:

```
For each i in range(n):
    For each j in range(n - i - 1):
        If trip_count[j] < trip_count[j+1]:
            Swap them
```

Complexity:

- Time → $O(n^2)$
- Space → $O(1)$

Although inefficient for large datasets, it effectively demonstrates sorting logic for **aggregated results** after data grouping.

---

4. Insights and Interpretation

Three core insights were derived from the cleaned TLC data:

Insight 1: Trips by Hour

A grouped query by pickup hour revealed peaks between **8–9 AM** and **5–7 PM**, representing rush-hour demand.
This insight validates that urban taxi demand strongly aligns with work-hour transitions.

Insight 2: Average Trip Distance and Driver Pay

Analysis by the vendor showed differences in trip distance and fare patterns.
One vendor consistently had **higher fares for shorter distances**, indicating **premium pricing** in certain zones.

Insight 3: Pickups by Borough

Aggregating trips by borough revealed **Manhattan** as the busiest zone, followed by **Brooklyn** and **Queens**.
This pattern supports the hypothesis that **business districts** dominate taxi demand.

Each insight was visualized using **Plotly and Matplotlib**, generating bar charts and line graphs (included in the `docs/screenshots/` folder).
These findings can guide city planners in **optimizing urban mobility policies**.

---

5. Reflection and Future Work

Challenges Faced

- **Technical:** Large datasets caused performance issues; schema inconsistencies and environment configuration required repeated debugging.
- **Team:** Coordinating consistent file paths and naming conventions across collaborators was initially difficult.

Lessons Learned

- The importance of **data validation and schema consistency** before analysis.
- Modularizing code improves maintainability and debugging efficiency.

Future Improvements

- **Cloud deployment** (e.g., Render, Railway, or AWS) for public API access.
- **Real-time updates** through streaming data ingestion.
- **Geospatial mapping** for route heatmaps and trip density visualization.
- **Pagination and caching** to improve dashboard performance.

Conclusion

The *Urban Mobility Data Explorer* successfully integrates data cleaning, relational modeling, custom algorithm design, and visualization to extract actionable insights from NYC taxi data. Through this project, our team demonstrated technical problem-solving, collaboration, and a strong understanding of data system design directly supporting the ALU mission to **develop leaders who drive innovation in Africa and beyond**.