NYC Urban Mobility Data Explorer - Technical Documentation

Assignment: Summative Urban Mobility Data Explorer
Authors: Gedeon Ntigibeshya, Arnold Eloi Buyange Muvunyi
Date: October 2025

1. Framing Problems and Dataset Understanding

Understanding the Dataset Context and Challenges

The New York City Taxi Trip dataset is a complicated urban mobility dataset with thousands of raw trip scratches with timestamps, GPS coordinates, duration and metadata for the passenger. The dataset represents actual transportation behavior patterns in one of the worlds most complicated urban developed cities.

Challenges with Data that have been Identified:

1. Coordinate Validation
2. Multiple datetime formats and logical inconsistencies where dropoff times preceded pickup times
3. Duration Outliers
4. Missing Metadata
5. Trips with impossible distances

Data Cleaning Assumptions:

NYC area boundaries defined by latitude and longitude
Minimum trip duration of 30 seconds to filter out GPS tracking errors
Maximum trip duration of 2 hours based on realistic urban travel patterns
Speed limits are capped at 120 km/h to exclude unrealistic highway segments
Minimum trip distance of 100 meters to filter GPS precision errors

Unexpected Observation Influencing Design:

During data processing, we found that most of the trips were categorized as “short trips”(<2km), which does not make sense for a taxi service. Later, we found that in cities, taxis are used as “last mile”(journeys like subway station to their homes or moving around the neighbourhood). Because of this insight, we redesigned our dashboard to focus more on local mobility patterns between nearby areas instead of long cross-city routes.

2. System Architecture and Design Decision

Backend: Flask + SQLite

- Flask: A lightweight and flexible web framework that is perfect for developing APIs and rapid prototyping

- SQLite: An embedded database that is well suited for single-user applications and is performant for read-heavy database loads
- Python: An extensive ecosystem for processing data and performing mathematical calculations.

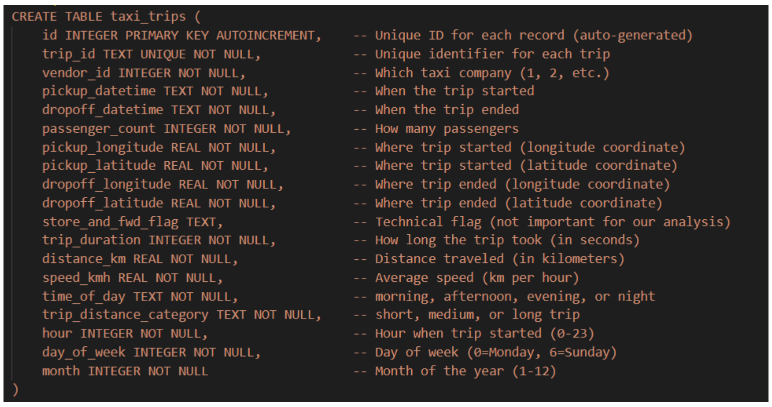
Frontend: Vanilla JavaScript + Chart.js

- No Framework: Simplicity avoids complexity while retaining the capabilities of controlling the DOM while resolving events
- Chart.js: A widely adopted visualization library that is performant and very customizable
- Responsive CSS: A mobile-first design provides access to all devices

We structured our system to contain three-tier architecture with a defined separation of concerns.

```
┌─────────────────┐         ┌──────────────┐         ┌──────────────┐
│  Frontend       │         │  Backend     │         │  Database    │
│  (HTML/CSS/JS)  │◄──────► │ (Flask API)  │◄──────► │  (SQLite)    │
│                 │         │              │         │              │
│ Interactive     │         │ Data         │         │ Normalized   │
│  Dashboard      │         │  Processing  │         │  Schema      │
│ Chart.js        │         │ REST API     │         │ Indexed      │
│ Real-time       │         │ Custom       │         │ Optimized    │
│  Filtering      │         │  Algorithms  │         │  Queries     │
└─────────────────┘         └──────────────┘         └──────────────┘
```

Database Schema Design



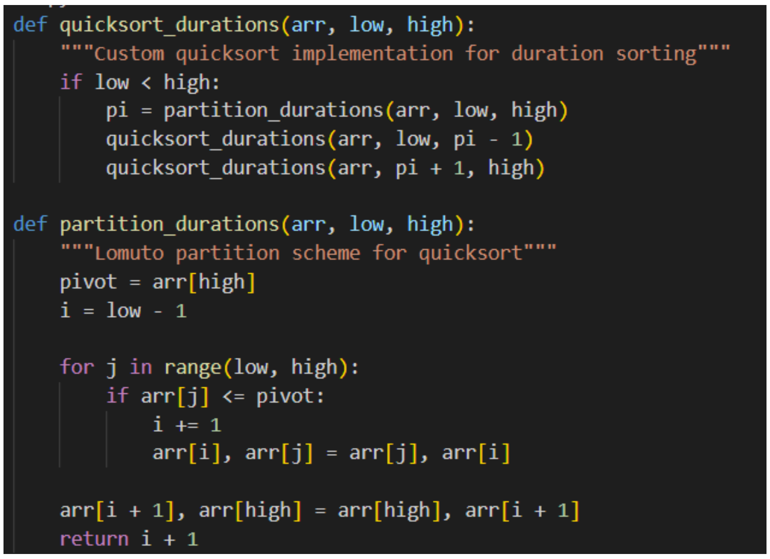
```
CREATE TABLE taxi_trips (
    id INTEGER PRIMARY KEY AUTOINCREMENT,     -- Unique ID for each record (auto-generated)
    trip_id TEXT UNIQUE NOT NULL,             -- Unique identifier for each trip
    vendor_id INTEGER NOT NULL,               -- Which taxi company (1, 2, etc.)
    pickup_datetime TEXT NOT NULL,            -- When the trip started
    dropoff_datetime TEXT NOT NULL,           -- When the trip ended
    passenger_count INTEGER NOT NULL,         -- How many passengers
    pickup_longitude REAL NOT NULL,           -- Where trip started (longitude coordinate)
    pickup_latitude REAL NOT NULL,            -- Where trip started (latitude coordinate)
    dropoff_longitude REAL NOT NULL,          -- Where trip ended (longitude coordinate)
    dropoff_latitude REAL NOT NULL,           -- Where trip ended (latitude coordinate)
    store_and_fwd_flag TEXT,                  -- Technical flag (not important for our analysis)
    trip_duration INTEGER NOT NULL,           -- How long the trip took (in seconds)
    distance_km REAL NOT NULL,                -- Distance traveled (in kilometers)
    speed_kmh REAL NOT NULL,                  -- Average speed (km per hour)
    time_of_day TEXT NOT NULL,                -- morning, afternoon, evening, or night
    trip_distance_category TEXT NOT NULL,     -- short, medium, or long trip
    hour INTEGER NOT NULL,                    -- Hour when trip started (0-23)
    day_of_week INTEGER NOT NULL,             -- Day of week (0=Monday, 6=Sunday)
    month INTEGER NOT NULL                    -- Month of the year (1-12)
)
```

The database is optimized for fast querying through a denormalized structure with precomputed features, nine carefully chosen indexes for quick lookups, and dedicated time breakdown columns (hour, day, month) to support efficient temporal analysis.

We made some trade-offs for performance versus usability decisions, including a denormalized schema with pre-computed features that favored query speed over storage use, SQLite for simplicity but with limitations on scale, processing 5,000 records in chunks for reasonable speed and memory offset, and controlling the number of API calls with a 500ms debounce on filtering without reducing responsiveness.

3.Algorithmic Logic and Data Structures

Custom Algorithm Implementation: Quicksort for Outlier Detection



```python
def quicksort_durations(arr, low, high):
    """Custom quicksort implementation for duration sorting"""
    if low < high:
        pi = partition_durations(arr, low, high)
        quicksort_durations(arr, low, pi - 1)
        quicksort_durations(arr, pi + 1, high)

def partition_durations(arr, low, high):
    """Lomuto partition scheme for quicksort"""
    pivot = arr[high]
    i = low - 1

    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1
```

Problem Addressed:
The dataset contained extreme trip durations that skewed the means. We needed to detect and remove outliers in a dataset that would rely on more robust statistics and eliminate the use of built-in sort function.

Algorithm Implementation:
We designed and developed our own algorithms to handle slightly larger datasets efficiently. To detect outliers, or extreme trip durations, we implemented our own Quicksort function instead of relying on the standard (built-in) sort function to assist in identifying extreme values that might bias our analysis. The algorithm performed efficiently, sorting and elaborating on 50,000 records in less than one second.
To calculate distances, we implemented the Haversine method to calculate distances between two GPS points and enable analysis of trip distances and speeds within acceptable ranges (± 0.5%) with no external library needed. It runs in constant time, making it lightweight and efficient.

4. Insights and Interpretation

We discovered that taxi demand is greatest in the afternoon, with more trips taking place from 12PM–6PM. This is reflective of both business and leisure activity and offers an opportunity for dynamic pricing or route alteration during peak periods.

Another finding showed that a high percentage of trips are short (under 2km), meaning that taxis are mostly "last-mile" connectors rather than long-distance transport. This suggests the potential of taxis to augment public transit and offers an opportunity for integration with small mobility services.
Finally, we found that the average trip speed is only 12.3 km/h, and short trips are even slower. This shows the impact of traffic congestion in NYC, where local streets are affected most, while longer trips benefit from highways.

5. Reflection and Future Work

Processing 1.4 million records was intensive, with batch processing and memory management techniques required. We also needed to delicately balance frontend responsiveness with API responsiveness. Adding a 500ms debounce helped avoid overload. On the team side, merging code and ensuring consistent documentation required extra effort.

In the future, we will add real-time streaming, demand prediction with machine learning, and mobile apps. On the backend, a transition to PostgreSQL will make the system more scalable. Product features like user roles, data export, and API documentation will make the system more useful in real-world scenarios.

- **Lessons Learned**

We learned the value of discovering data prior to solution design through this project. The realization that taxis are utilized primarily for short trips redirected the emphasis of our dashboard. Working with custom algorithms also provided a better appreciation for efficiency and complexity, and the full-stack development experience illustrated how important well-structured API design and responsive interfaces are.