Urban Mobility Data System Documentation

1. Problem Framing and Dataset Analysis

Dataset Description

NYC Taxi Trip raw train split (trip-level). Fields include yellow_tripdata, taxi_zone_lookup, and taxi_zone. These datasets together form a raw rational environment similar to a fact table and a dimension table

Data Challenge

While working with the dataset, we encountered several issues, such as inconsistent timestamp format, extremely long trips with unrealistic durations, and fare amounts that did not match the trip distance

Data Cleaning

To address this challenge, we made some change like:

- trips with missing values were removed
- All timestamps were standardized to a single datetime format
- Extremely large values were treated as outliers and logged for transparency

Unexpected Observations

One unexpected observation was that some short-distance trips had relatively high fares, especially in busy areas and hours

2. System Architecture And Design Decisions

System Architecture Overview

Our system follows a 3 tier architecture:

- Frontend, which is a web dashboard for data exploration and visualization
- Backend, which is an API service for data processing and querying
- Database which is a relational database storing cleaned and enriched data

Technology Stack Choices

- **Backend:** Flask (Python)
  Chosen for simplicity, readability, and strong support for data processing.

- **Database:** SQL
  Used to implement a normalized relational schema with indexing for performance.

- **Frontend:** HTML, CSS, JavaScript
  Provides a lightweight and responsive dashboard.

- **Data Processing:** Python scripts for cleaning, normalization, and feature engineering.

Database Design

The database schema is normalized and includes

- A trip table containing trip records
- Indexes on timestamps and location ID to improve query performance
- A taxi zone table containing zone data

Design Trade-Offs

- We chose a relational databaseto support structured queries
- The frontend focuses on clarity and usability rather than complex animation
- Some extreme outliers were removed rather than corrected to avoid introducing bias

3. Algorithmic Logic and Data Structure

Algorithm implementation

As required we manually implemented a custom algorithm without relying on built-in functions. We implemented a custom ranking algorithm to identify the top taxi zones based on average fare per trip

Algorithm Description

The algorithm:

1. Iterates through all trips
2. Groups trips by pickup zone
3. Calculates total fare and trip count per zone
4. Computes the average fare per zone
5. Sorts zones manually using a basic comparison-based sorting approach

Pseudo-Code

```
for each trip in trips:
    if zone not in zone_data:
        initialize total_fare and count
    add fare to total_fare
    increment count

for each zone in zone_data:
    average_fare = total_fare / count

perform manual sort on zones based on average_fare
```

Complexity Analysis

- **Time Complexity:** O(n2) due to manual sorting
- **Space Complexity:** O(n) for storing zone aggregates

4. Insights and Interppretation

Insight1: Trips are most frequent duing morning and evening rush hours. This shows strong commuter behavior anf higher transportation demand during work hours

Insight2: Some zones show higher fare per mile than others, this suggest congestion, traffic delays on busy urban areas

5. Reflection and Future Work

Challenge Faced

- Handling large dataset efficiently
- Coordinating backend and frontend integration
- Managing team collaboration and version control
- Cleaning messy real word data

Future Improvement

If this were a rwal world product we would**:**

- Add real time data updates
- Improve map based visualizations
- Implement user authentication
- Deploy the system to a cloud platform
- Optimize algorithms for better performance