TEAM REPORT

1. Problem Framing and Dataset Analysis

The NYC Train Trip dataset contains 1,458,644 trip records with 11 columns, including vendor_id, pickup and dropoff locations, passenger count, pickup and dropoff datetime, trip_duration_sec, and store_fwd_flag. This dataset represents real-world urban mobility patterns and serves as the foundation for analyzing transportation efficiency, vendor performance, and trip characteristics across New York City.

Challenges encountered while analyzing the dataset are:

- Invalid coordinates like (0,0)
- Invalid datetime format

Key assumptions made are:

- Fare is measured per kilometer
- The highest speeds beyond 120km/h are outliers
- All timestamps were normalized to UTC to ensure consistency across the dataset.
- Coordinates were also precisely rounded off to 6 decimal places.
- Trips with zero coordinates were removed from the dataset, assuming they represent wrong locations.

Apart from processing the records, our pipeline went through the dataset to remove all missing and invalid data, ensuring that the data we used is of high quality. There were no missing values and duplicates in the dataset. Unexpected observations that influenced our design are simple ways of handling errors, reduced intensive data validation, which led us to focus more on feature creation.

2. System Architecture and Design Decisions

View System Architecture Diagram

The sections in the system architecture allow a UI → API → Backend request flow for a user interacting with the UI via mobile apps/web portals. The frontend consumes data/API from the backend. The CDN, DNS, and load balancers distribute frontend and API traffic. External partners/regulators are included so that in the future, the app will integrate with vendors for ticket payments and refunds, and government/regulatory agencies for compliance.

How data/API is consumed:

- *API Gateway:* entry point that routes and secures calls to APIs for trip, ticketing, user, schedule, and notification management. APIs were structured with FastAPI due to its high performance in handling concurrent requests and API documentation auto-generation with Swagger. The trip service manages searching, booking, updating, and canceling train trips. The analytics service collects and analyzes journey data, ridership patterns, delays, and trip performance, and generates operational insights. The user management service handles user registration, authentication, authorization, and in the future; ticket holder management. The notification/event bus notifies users of ticket confirmations, changes, and delays via event-driven updates.

- *Data Layer:* The relational DB stores users, trains, and trips. SQLite3 was used as it is sufficient for analytical workloads, and it supports files, which makes it easy for deployment and backup. The analytics data warehouse (to be implemented in the future) centralizes aggregated data for historical analysis, performance dashboards, and reporting. The caching (to be implemented in the future) provides rapid access to commonly queried data like train schedules and session management.
- *Infrastructure:* Applies security checks on every layer of the stack, covering API calls, data access, trip activity, and user management. Firewall and JWT enhance robust authentication, encryption, and audit for regulatory compliance and system integrity. JWT are secure and scalable, and require no server-side session storage, making frontend integration simple. Docker manages, scales, and deploys all backend microservices efficiently. For the CI/CD pipeline, GitHub Actions automates testing and deployment of application changes. Monitoring and logging (to be implemented in the future) track system health, traffic, errors, and logs for rapid incident response.

View Entity Relationship Diagram

The schema structure captures operational relationships accurately while ensuring clean data, ease of updates, and reliability for analytics and reporting needs. It follows:

- *Separation of Entities*: The **Vendors** and **Trips** tables are distinct entities, to avoid data redundancy and to enable flexible management when a vendor has multiple trips.
- *One-to-Many Relationship:* One vendor can provide many trips, but each trip is associated with only one vendor.
- *Use of Primary and Foreign Keys:* The **primary key** (vendor_id for Vendors, trip_id for Trips) uniquely identifies records, securing data integrity and fast lookups. The **foreign key** (vendor_id in Trips) enforces referential integrity. Every trip must reference a valid vendor, so data remains consistent and valid.
- *Attribute Selection:* Trip attributes (location, duration, etc.) fully capture details needed for analytics, operations, and customer experience tracking in train mobility.
- *Clarity and Scalability:* Adding features such as new trip attributes, vendor metadata, or analytics can be done without disrupting existing data or relationships, making the schema clear, scalable, maintainable, and future-proof.

3. Algorithmic Logic and Data Structures

The NYC Train Mobility App, needed an efficient way to track and store anomalous trip records during data processing. This required a data structure that could: dynamically grow as outliers are detected, maintain the insertion order of outliers, efficiently add new records, and convert to list format for analysis.

A custom singly linked list data structure was implemented using two classes: Node class that stores individual trip records and a reference to the next node, and the LinkedList class that manages the chain of nodes with a head and tail pointer.

View Custom Code or View Pseudo-Code

Complexity Analysis determined that the add operation has a time complexity of O(1), as it performs constant-time insertion using the tail pointer. Converting the structure to a list takes O(n) time because it requires traversing all n nodes. And, the space usage for storing outlier records is O(n), reflecting linear space proportional to the number of records. Regarding space complexity, each node requires O(1) constant space. Overall, the entire data structure uses O(n) space, linearly proportional to the number of nodes.

4. Insights and Interpretation

Average and Distribution of Trip Speeds
Derivation and Visualization
Most trips at moderate speed, and a few very fast/slow signals healthy transit flow except at traffic hotspots. Extended slow-speed tails may indicate congestion or route inefficiencies.

Vendor Idle Time Patterns
Derivation and Visualization
Long idle times suggest inefficient vehicle usage; possibly poor demand prediction or excess fleet size. Clusters of low idle time mean high utilization and efficiently scheduled operations, a key metric for maximizing service and minimizing cost in urban transport.

Fare per Kilometer Efficiency
Derivation and Visualization
Low variance and reasonable means in fare_per_km indicate fair, transparent pricing. High outliers or inconsistent pricing highlight fare structure inefficiencies or potential overcharging. Urban mobility stakeholders can use this to inform pricing reforms.

5. Reflection and Future Work

The data processing scale was a technical challenge. Processing 1.458 million records efficiently without overwhelming the system was unavoidable. However, we implemented streaming processing using pandas to optimize memory usage by selectively loading columns. It was also challenging to find a meeting time that was convenient for all team members. As a solution, we split into two to work on the tasks separately and shared our findings in the evenings, which was still inconvenient for some members.

Future work includes: database migration from SQLite3 to PostgreSQL, implementing Apache Kafka for streaming data, and adding predictive models for demand forecasting and route optimization