NYC Taxi Trip Analytics Dashboard – Technical Documentation

1. Problem Framing & Dataset Analysis

The dataset consists of raw trip-level records from the NYC Taxi & Limousine Commission, including:

- Pickup and dropoff timestamps
- Pickup and dropoff coordinates
- Trip duration and distance
- Fare and tip information
- Passenger and payment metadata

Challenges identified:

- ❖ Missing values for coordinates and timestamps
- ❖ Duplicate records and trips with invalid durations
- ❖ Outliers in trip duration and fare

cleaning steps:

- Trips with negative or zero duration were removed
- Missing coordinates were excluded or imputed where possible

Derived features added:

Trip speed **(distance ÷ duration)**

Fare per km

Trip time of day **(hourly aggregation)**

Unexpected observation:
A small percentage of trips had extremely high speeds (>200 km/h), suggesting GPS errors or data corruption. These were logged and excluded from analytics.

2. System Architecture & Design

The system uses a three-tier architecture:

Frontend (HTML/JS/Chart.js)

↕

Backend (Node.js)

↕

Database (Mysql)

Structure

```
backend/
├── src/
│   ├── controllers/
│   ├── models/
│   ├── routes/
│   ├── services/
│   ├── utils/
│   ├── config/
│   ├── data/
│   └── cleaned_data/
├── package.json
├── excluded_records.json
├── screenshots/
frontend/
│   ├── index.html/
│   ├── script..js/
│   ├── style.css/
└── README.md
```

Stack Choices:

Node.js & Express: Fast, scalable backend for API endpoints

Mysql: Relational database for structured trip data and indexing

Chart.js + Vanilla JS: Interactive dashboards on the frontend

Database Design:

Trips table normalized with indexes on pickup_datetime, vendor_id, and passenger_count for efficient queries

Derived features stored for faster analytics (e.g., trip speed)

Trade-offs:

Chose pre-calculated derived features over dynamic computation for performance

Limited frontend to charts and tables for clarity, avoiding heavy UI frameworks

3. Algorithmic Logic & Data Structures

Custom Algorithm – Top K Trips by Metric:

Pseudo-code

```
function quickselect(arr, k, compareFn):
    define partition(left, right):
        pivot = arr[right]
        i = left
        for j in range(left, right):
            if compareFn(arr[j], pivot) > 0:
                swap(arr[i], arr[j])
                i += 1
        swap(arr[i], arr[right])
        return i

    define select(left, right, k):
        if left >= right: return
        pivotIndex = partition(left, right)
        count = pivotIndex - left + 1
        if k == count: return
        else if k < count: select(left, pivotIndex - 1, k)
        else select(pivotIndex + 1, right, k - count)

    select(0, arr.length - 1, k)
    return arr.slice(0, k)
```

Complexity:

Time: O(n*k), Space: O(k)

Solves the problem of ranking trips by speed, fare per km, or duration without relying on library sorting

Usage:

Used for frontend “Top Trips” chart

Ensures API response is fast and memory-efficient

4. Insights & Interpretation

Insight 1 – Peak Trip Hours:

Most trips occur between 8–10 AM and 5–7 PM (commute peaks)

Helps NYC optimize taxi deployment

Insight 2 – Trip Duration vs Passenger Count:

Longer trips tend to carry fewer passengers

Suggests efficiency patterns for ride-sharing optimization

Insight 3 – Vendor Performance:

Vendor 2 handles fewer trips but higher average fare

Could indicate service differences or geographic coverage

(Insert screenshots of Vendor Chart, Duration Distribution, Hourly Trips here)

5. Reflection & Future Work

Challenges:

- ➔ Handling large raw CSVs with memory constraints
- ➔ Cleaning messy geolocation data and detecting anomalies
- ➔ Integrating backend APIs with interactive frontend charts
- ➔ Future Improvements:
- ➔ Implement real-time data ingestion and dashboard updates
- ➔ Add route clustering and heatmaps for visual insights
- ➔ Deploy on cloud with authentication for multi-user analytics