NYC Trip Analysis Web Application

Overview

This is a comprehensive and full documentation for the NYC Trips web-app, which analyzes data from the New York City Taxi Trip dataset, which contains trip records including timestamps, distances, durations, pickup/dropoff locations, and other metadata of one taxi.

This file will help you understand our design thinking and engineering decisions, as it encompasses the System Architecture and Design Decisions with relevant justifications of why we have chosen to use NodeJS for the backend and HTML, CSS, and JavaScript in the frontend. In addition to that, you will be able to understand the algorithms and data types used and their justifications. Moreover, we have added our insights and reflections on the work and have recommended the future implementation of the codebase and additional features for future scalability of the web application.

Problem Framing and Dataset Analysis

Initially, the datasets given have a total of 1,383,552 trips, which is a massive amount of data to clean and organize from a CSV file, with some missing fields that should be there for us to understand and present the data with our clean dashboard. Originally, it contains these fields: Pickup and drop-off timestamps

- Pickup and drop-off coordinates
- Trip duration and distance
- Fare amount and tip
- Passenger and payment metadata

Upon scanning the datasets, we think we are missing some important fields; however, we decided to use available data to generate new fields. We felt like we were missing.

- **Provided Data Challenges and Assumptions of the required data**

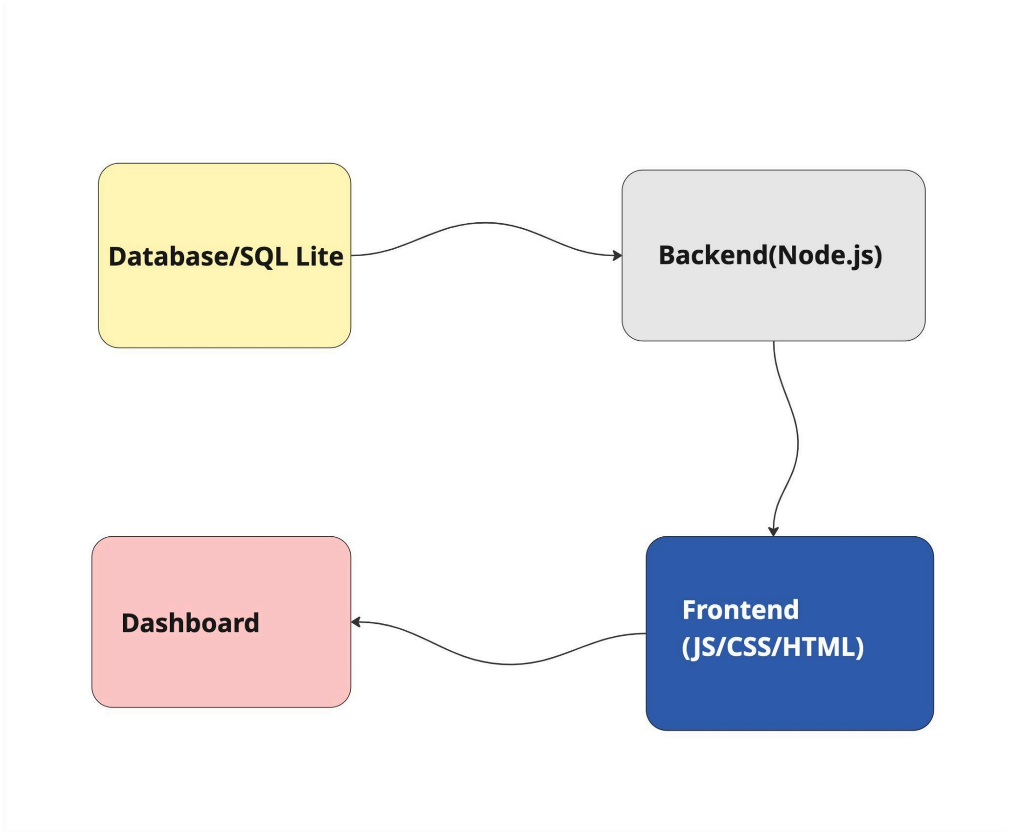
    1. The duration of the trip is missing, but required!
    2. The distance travelled over one trip is missing, but is required
    3. Time of Day is missing, but somehow required
    4. Average speed is missing but required.

    We assumed that the distance travelled was critical and required, but it should be calculated from longitudes and latitudes. similar to all those numbered missing fields above, like explicit trip duration in minutes and the speed by taking the calculated distance and the time duration.

- **Unexpected observation during data cleaning**

  Before cleaning data, we thought the dataset could not be full or be beneficial in any way, but we found out that we had all the things we needed, and the data provided was enough.

System Architecture and Design Decisions



SQLite

We used SQLite because it's lightweight, file-based, and doesn't require setting up a full database server. For an urban mobility project where data can be stored locally (like trip logs or coordinates), the data fetching needed to be fast. The trade-off is that even though it is lightweight, it can't handle millions of data as the one we have.

Node.js

Node.js is also fast, handles many simultaneous connections efficiently, and works well with JavaScript on both the client and server. That made development simpler since we could use one language across the stack. The trade-off is that it is single-threaded, which might cause the CPU to lag while loading a large amount of data.

JavaScript / HTML / CSS (Front End)

Together, they provide a responsive, browser-based interface where users can visualize mobility data easily, and it is available for all devices and browsers. However, it is hard to implement a multicomponent web application with this, and in the future, this web application must have features like security, etc, and that stack can't handle this well.

Algorithmic Logic and Data Structures

Algorithm Steps

1. **Load Data**
    - Read CSV line by line.
    - Store each row in an array, rawData.
    - **Complexity:** O(n), n = number of rows.
2. **Remove Duplicates**
    - Use a Set to store unique keys per trip (pickup/dropoff times + coordinates).
    - Filter rawData to keep only unique rows.
    - **Data Structure:** Set for uniqueness.
    - **Complexity:** O(n).
3. **Handle Missing Values**
    - Iterate rawData and remove rows where any required field is missing or null.
    - **Complexity:** O(n).
4. **Validate Coordinates**
    - Check if latitude/longitude are numbers and within NYC bounds.
    - Remove rows with invalid coordinates.
    - **Complexity:** O(n).
5. **Validate Timestamps**
    - Parse pickup/dropoff times.
    - Ensure dropoff > pickup and year is in a valid range.

6. **Remove Negative or Invalid Values**

   Remove rows with negative or zero trip durations or invalid passenger counts.

7. **Remove Outliers**

   - Sort trip durations, compute IQR.
   - Filter rows outside [Q1 - 1.5*IQR, Q3 + 1.5*IQR].
   - **Data Structure:** Array for sorting.
   - **Complexity:** O(n log n) (due to sorting).

8. **Create Derived Features**

   Compute:

   - distance_km via Haversine formula.
   - speed_kmh = distance / time.
   - Temporal features: hour_of_day, day_of_week, is_weekend.
   - Categorical flags: time_category, is_rush_hour.

9. **Save Cleaned Data**

   Write an array to CSV or insert into the database.

Data Structures Used

- **Array:** rawData, cleanedData – stores rows.
- **Set:** For duplicate detection.
- **Object:** For logging counts and suspicious records.
- **Sorted Array:** For IQR/outlier removal.

DSA Notes

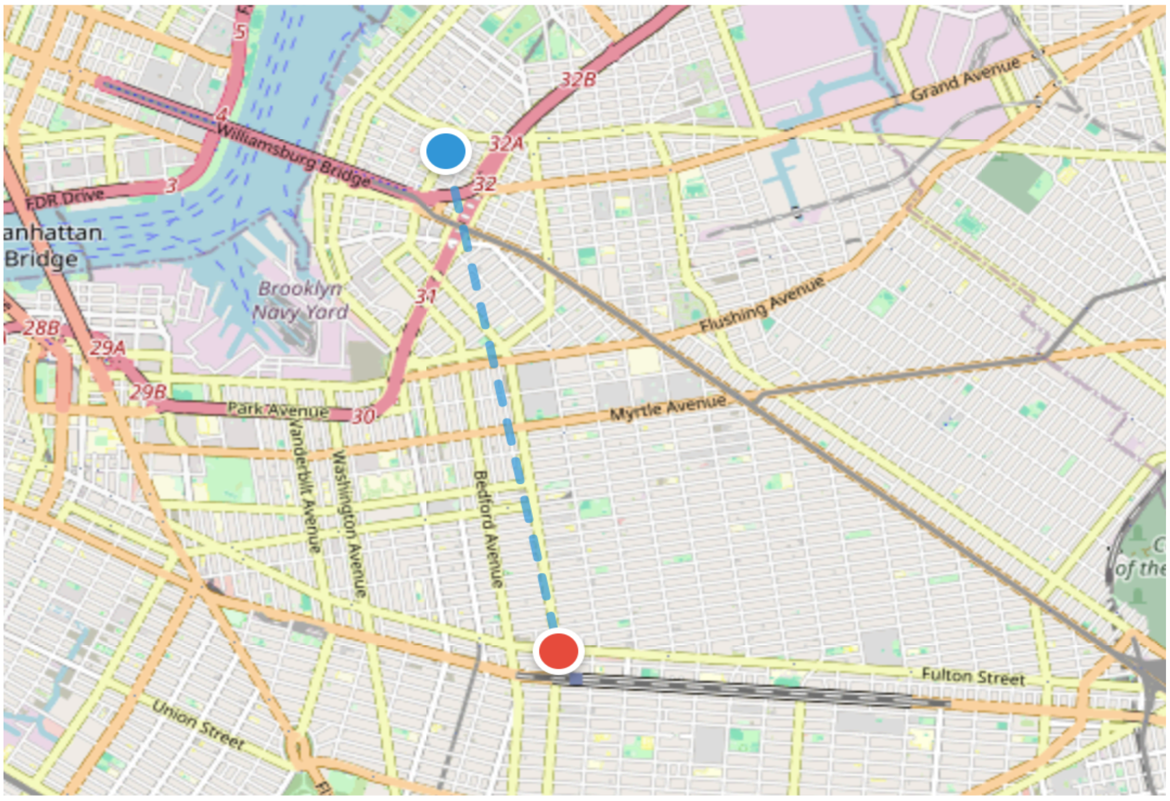
- Duplicate removal → hashing with Set.
- Outlier removal → sorting + IQR filtering.
- Derived features → arithmetic and date operations per row.
- Linear passes dominate most cleaning steps; sorting is the only O(n log n) step.

Insights and Interpretation

There are alot of meaningful insights derived from the data provided and the data we were able to derive from the original dataset. Three of them include:

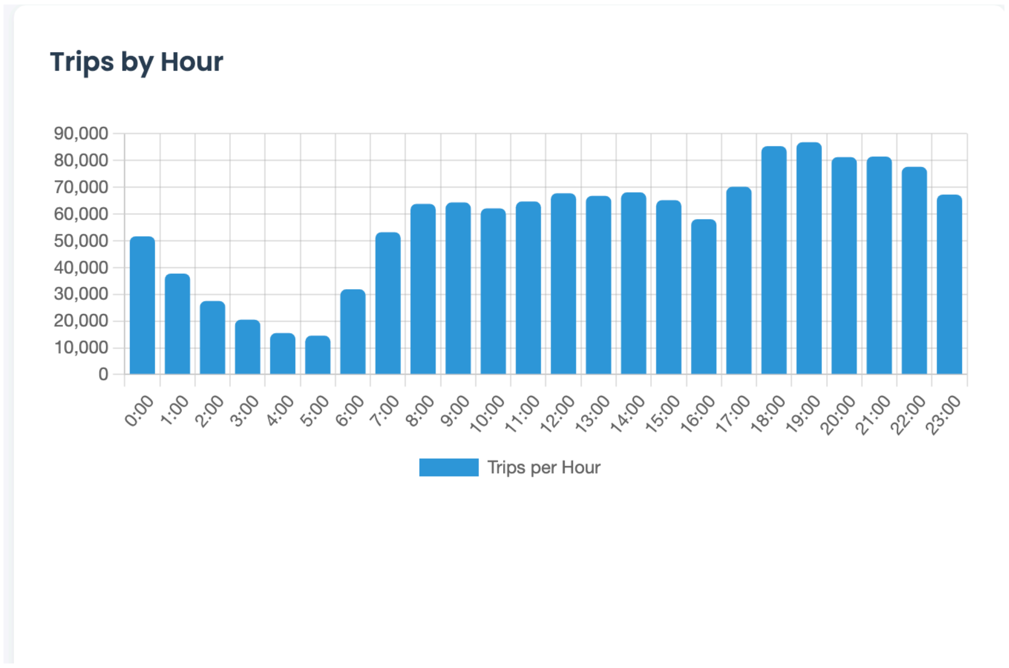
1. **Trip visualization on a map (leaflet map)**

   With the help of the provided latitudes and longitudes data, we were able to visualize the entire displacement of each and every trip. But also, we located multiple trips at once. In the context of urban mobility, this means it helps us understand how people travel through the city, which areas are busiest, and what routes are most common.



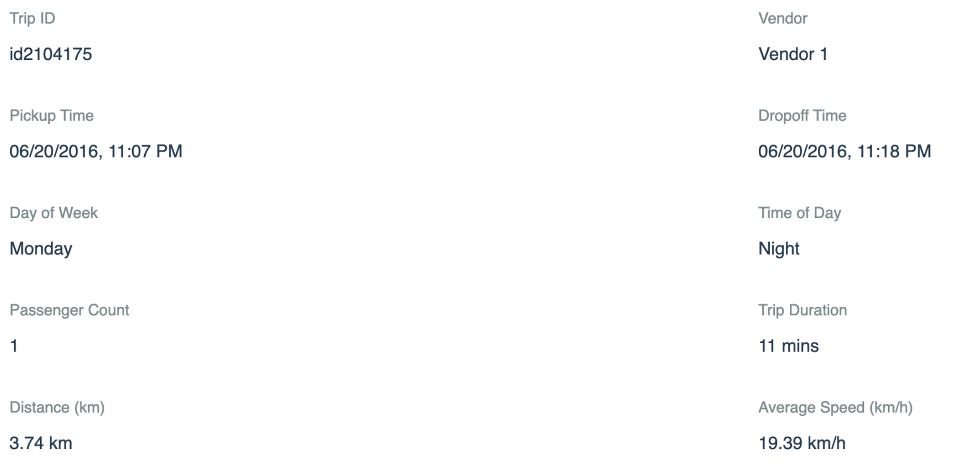
2. **Busiest hours of the Taxi (on the graph)**

   From the data provided, the pickup time should tell us which time the taxi is busy the most, and that can be printed on a graph. In the context of urban mobility, it means we can tell when it is better to travel around the city and when people access transport the most.



3. **Trip day and daytime specifications**

   From the date and the time of the trip, we are able to retrieve some information on whether the trip was in the morning, afternoon, evening, or night. And this also tells us when most of the trips are done in terms of urban mobility.



Trip ID	Vendor
id2104175	Vendor 1
Pickup Time	Dropoff Time
06/20/2016, 11:07 PM	06/20/2016, 11:18 PM
Day of Week	Time of Day
Monday	Night
Passenger Count	Trip Duration
1	11 mins
Distance (km)	Average Speed (km/h)
3.74 km	19.39 km/h

Reflection and Future Work

Technical and Team Challenges

HTML, CSS, and JavaScript implementations don't allow multi-component web applications to be implemented so easily, and it is hard to debug. The implementation of frameworks such as React would simplify collaboration in the team for even small features and complicated api for authentications and many others.

Technical and Team Challenges

In the future, we should include multiple APIs and use complex frameworks that handle everything from cache loading, routers and many others. But also, we should make it more specific and try to add the individual's data (passengers) so that it becomes more useful to multiple vehicles and passengers

The end