Problem Framing and Dataset Analysis

Problem Context

To understand the patterns in urban mobility, trip behaviour, and revenue generation in New York City (NYC), our app analyses NYC taxi trip data. The dataset includes trip-level records such as pickup and dropoff timestamps, trip distance, passenger count, fare, tip amount, and pickup and dropoff locations.

Data Challenges

While preprocessing the data to start up our projectr, we noticed several data quality issues. One of them were the missing values in fields such as **`congestion_surcharge.`** Another was inconsistency in datetime formats (i.e. day-first timestamps). Records with **`passenger_count = 0`**, which may indicate errors or edge cases were also spotted. Lastly, some of the monetary fields contained zero values where tips or tolls were expected to be stated.

Data Cleaning Assumptions

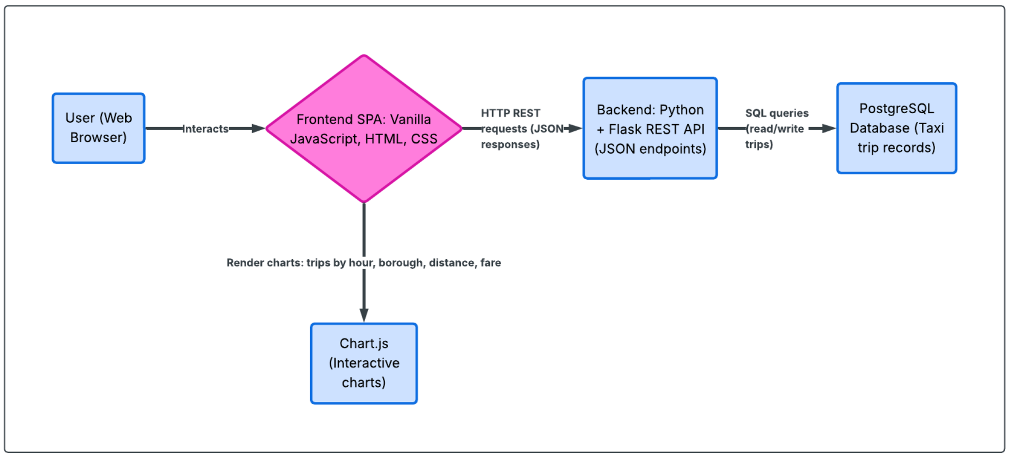
Now, in order to ensure there was consistency, some things had to be put in place. The datetime fields were converted using day-first parsing. Missing surcharge values were treated as zero instead of just being left empty. Trips with invalid timestamps were dropped. Lastly, passenger counts of zero were retained but they were flagged as anomalies.

Unexpected Observation

One unexpected thing we found was that trips with short distances sometimes had excessively high total fares, which were likely due to fixed surcharges or traffic-related fees. As a result of this finding, we decided to include both trip distance and total amount in the dashboard visualisations.

System Architecture and Design Decisions

System Architecture


User (Web Browser)
interacts
Frontend SPA: Vanilla JavaScript, HTML, CSS
HTTP REST requests (JSON responses)
Backend: Python + Flask REST API (JSON endpoints)
SQL queries (read/write trips)
PostgreSQL Database (Taxi trip records)
Render charts: trips by hour, borough, distance, fare
Chart.js (Interactive charts)

Our system follows the standard three step architecture:

- Frontend - HTML, CSS, and JavaScript dashboard for visualisation
- Backend - Flask API calling processed taxi data
- Data layer - Cleaned CSV dataset imported in the database (PostgreSQL)

The frontend communicates with the backend using REST API calls, and the backend process and formats data before returning JSCON responses.

Algorithmic Logics and Data Structures

For this project, we implemented custom algorithms in **`algorithm.py`** to analyse NYC taxi trip data without depending on already built Python functions.

- Top Earning Trips
  - Goal: find the top N trips with the highest total revenue
  - Approach: using buble sort to arrange the trips manually in descending order based on the **`total_amount`** field.
  - Pseudocode:

```
For i = 0 to n-1
    For j = 0 to n-i-1
```

```
        If trip[j].total_amount < trip[j+1].total_amount
            Swap trip[j] and trip[j+1]
Return first top_n trips
```

- The time complexity is $O(n^2)$ and space complexity is $O(n)$.

- Group Trips by Borough
  - Goal: counts total trips per pickup borough
  - Approach: it iterates over each trip and maintains a dictionary with borough counts
  - Pseudocode:

```
For each row in dataset:
        borough = row["Borough"]
        If borough not in dictionary:
            set borough_counts[borough] = 0
        increment borough_counts[borough] by 1
 Return borough_counts
```

  - Time complexity is $O(n)$ and space complexity is $O(b)$ where b is the number of boroughs

- Busiest Pickup Zones
  - Goal: identify the top N busiest pickup zones
  - Approach: it counts the trips per zone and manually sort using selection sort
  - Pseudocode:

```
For each row:
        zone = row["Zone"]
        increment zone_counts[zone]
    Convert dictionary to list of tuples
    Sort list manually in descending order by count
```

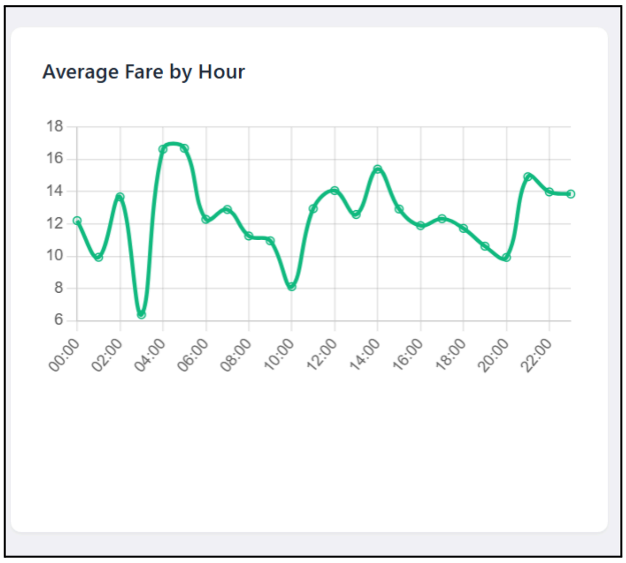
```
Return top N
```

- Time complexity is $O(n^2)$ and space complexity is $O(z)$ where z is the number of zones

Insights and Interpretations

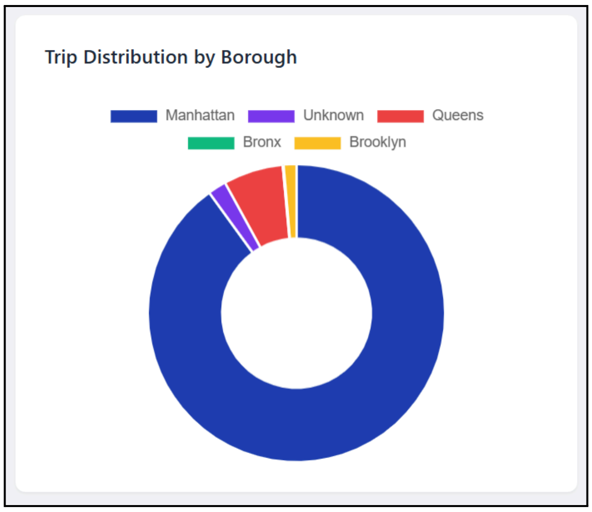
Our dashboard provides insights gotten from the taxi trip data through both visualisation and computation.

- Average Fare by Hour
  - Derived from calculated sum of fares and trip counts per hour, then hourly average is computed
  - Visualisation



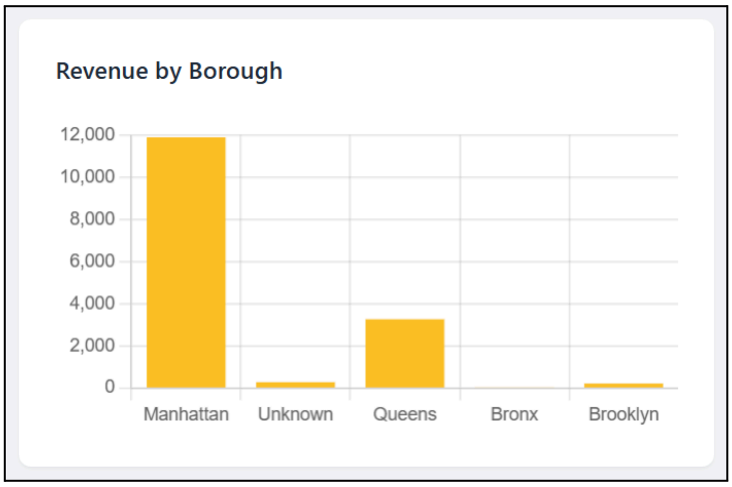
  - Interpretation: it peaks at some early hours and late evening hours, which indicates high demand or surge pricing.

- Trip Distribution by Borough
  - Derived from grouping trips by borough, using a dictionary count
  - Visualisation:



- Interpretation: Manhattan and Queens have the highest trip volumes, which indicates that they have a lot of people that commute.

- Revenue by Borough
  - Derived from summing the total amount for trips in each borough
  - Visualisation:



  - Interpretation: revenue is higher in the central areas compared to the outer boroughs, showing that this is where taxi services are most profitable.

Reflection and Future Work

We faced a few technical challenges during the development of the app. For example, integrating the database, backend, and frontend required us to be very careful and make sure all data fields aligned correctly. We also had to manually preprocess inconsistent CSV data. As a team we also faced a few challenges. Since we're all not in the same country currently we had to divide the work and make sure we were coordinating with each other, which proved to be a bit more difficult due to time constraints. Additionally, when it was time to integrate our individual parts together we had a lot of mismatches in data formats. In the future, if we decide to espand the system we can look into implementing geospatial mapping for pickup/dropoff visualisation.