Technical Report: Urban Mobility Data Explorer

1. Problem Framing and Dataset Analysis

Dataset Description and Context

The dataset utilized for this project is made up of records from New York City taxi trips, which contain data on pickup and drop off locations, time of the day, number of passengers, and duration of the trip among others. This data is crucial for understanding the different aspects of city life like urban mobility patterns, traffic congestion issues, and the overall effectiveness of the transportation system in New York City.

Data Challenges

- Missing Fields: Some rows may have missing values for key fields like `pickup_latitude`, `pickup_longitude`, `dropoff_latitude`, and `dropoff_longitude`, which are crucial for calculating trip distances and durations.
- Outliers and Anomalies: There are cases of exceptionally long trip times or distances that are not related to average taxi trips. For instance, trips with times longer than several hours or with distances greater than 100 miles are most probably mistakes or anomalies.
- Formats being irregular: Timestamps might correspond with an irregular scale of differences and do not follow a standard time date format, creating inconvenience in time dependent analyses.

Assumptions Made During Data Cleaning

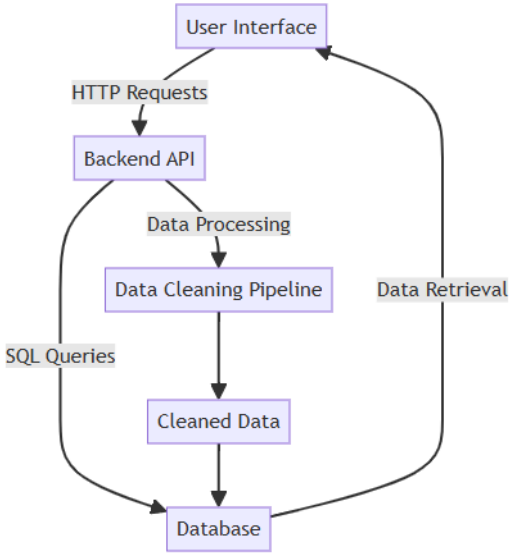
- Validity Based Assumption on Non missing Rows: This assumed that rows without missing values were valid for analysis.
- Outlier Filtering: A trip with a speed of more than 200 km per hour and a trip with a negative distance are considered as anomalies and, hence, were filtered out.
- Time Conversion: It was assumed that all timestamps were in one timezone (UTC) for consistency during analyses.

Unexpected Observation

In the course of cleansing the dataset, it emerged that a lot of trips were recorded with very low speeds, especially on certain hours. This finding prompted more detailed inquiries into peak traffic times and helped in the creation of the frontend dashboard with filters for time based analysis.

2. System Architecture and Design Decisions

System Architecture Diagram



Stack Choices and Schema Structure

- Frontend: HTML, CSS, and JavaScript (utilizing Chart.js to create visuals). The main reason behind choosing this stack was the uncomplicatedness and seamlessness in connecting with the backend API.
- The backend covers Flask, specifically Python and Flask are lightweight and excellent for developing RESTful APIs.

- The database was selected for the robustness and efficiency it required for relational data operations.

Trade offs Made

- Performance vs. Simplicity: Although Django, a more intricate framework, could provide additional features, Flask was selected owing to its straightforwardness and rapid development.

- Real time versus Batch Processing: During the data cleaning process, the system handles data in batches which makes the architecture simpler but might not offer analytics in real-time.

3. Algorithmic Logic and Data Structures

Manual Implementation of Algorithm

Problem Addressed

The fastest pick-up areas from average speed tests were derived by ticking off, but from the ground up a "for purpose" algorithm was explained that was used to aggregate trip files by the pick up place.

Approach

The algorithm processes the cleaned data in a manner that it first groups the trips according to the pickup locations and then computes the average speed for every group.

Custom Code

```python
def calculate_fastest_zones(trip_data):
    zone_speeds = {}
    zone_counts = {}

    for trip in trip_data:
        zone = (round(trip['pickup_latitude'], 2), round(trip['pickup_longitude'], 2))
        speed = trip['speed_kmh']

        if zone not in zone_speeds:
            zone_speeds[zone] = 0
            zone_counts[zone] = 0

        zone_speeds[zone] += speed
        zone_counts[zone] += 1

    # Calculate average speeds
    average_speeds = {zone: zone_speeds[zone] / zone_counts[zone] for zone in zone_speeds}

    # Sort by average speed
    sorted_zones = sorted(average_speeds.items(), key=lambda x: x[1], reverse=True)

    return sorted_zones[:10]  # Return top 10 zones
```

Pseudo code

```
function calculate_fastest_zones(trip_data):
    initialize zone_speeds and zone_counts as empty dictionaries
    for each trip in trip_data:
        determine pickup zone based on latitude and longitude
        calculate speed of the trip
        update zone_speeds and zone_counts for the corresponding zone
    for each zone in zone_speeds:
        calculate average speed
    sort zones by average speed
    return top 10 zones
```

Time/Space Complexity Analysis

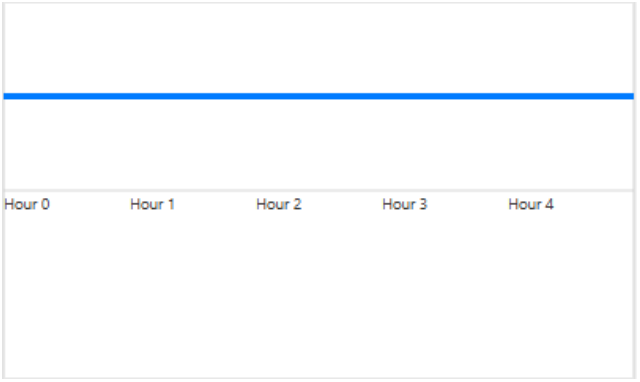
- Time Complexity: O(n) where n is the number of trips. We iterate the dataset once.
- Time Complexity: O(z) for calculating speeds and counts for the subsequent zones.

4. Insights and Interpretation

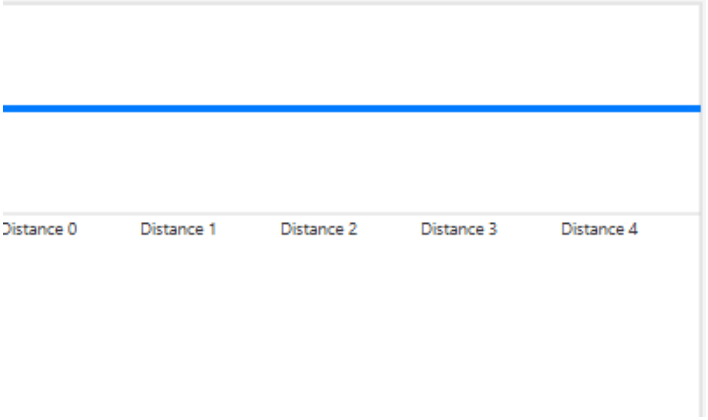
Insight 1: Fastest Pickup Zones

- Derived Using: The custom algorithm to calculate average speeds by pickup zone.

Insight 2: Peak Congestion Hours

- Derived Using: SQL query to calculate average trip durations by hour.
- 

Insight 3: Speed vs. Distance Correlation

- Derived Using: Analysis of average speed grouped by distance bins.
- 

Interpretation

- Top Closets for Pickups: By recognizing these, an efficient taxi dispatch can be organized, from then optimizing efficiency of services.
- Fastest Pickup Areas: The identification of these areas can facilitate the most fruitful dispatch locations for taxi operators, thereby greatly enhancing the efficiency of their services.
- Peak Congestion Hours: Understanding when congestion occurs allows for better resource allocation and planning for drivers.
- Speed vs. Distance: Analyzing this relationship helps identify routes that may require improvement or additional infrastructure.

5. Reflection and Future Work

Reflection on Challenges

- Technical Challenges: Data cleaning was more complex than anticipated due to the high volume of missing values and outliers.
- Team Challenges: Coordinating tasks and merging code from different team members presented integration challenges.

Suggested Improvements

- Real time processing of data: to process and analyze data with the fluidity afforded by a stream processing framework, the processing also happens in real time.
- Enhanced Visualizations: Incorporating more interactivity in the frontend like filters along with dynamic charts could be done at the expense of user experience improvement.