NYC Taxi Trips Dashboard: Technical Documentation

Team Members:

- Samuel NIYONKURU
- David NGARAMBE
- Attorney Valois NIYIGABA
- Prudence Browns

1. Problem Framing and Dataset Analysis

Dataset Description

The New York City Taxi Trip Data set, the data that we have analyzed, contains the detailed information of taxi trip in the form of timestamps, pickup/dropoff points, time spent on the trip, distance covered by the taxi, and cost of the taxi trip. This dataset is critical to the knowledge of the patterns of urban mobility in NYC.

Data Challenges

- Missing Values: A lot of records did not contain such important fields as timestamps or co-ordinates.
- Server errors: There were invalid records (e.g. dropoff times earlier than pickup times or location out of NYC).
- Outliers: There were a couple of trips that took an incredibly long amount of time or distance, which probably caused some mistake in data.
- Duplicates: Duplicate records had to be removed so that the data integrity would be guaranteed.

Assumptions

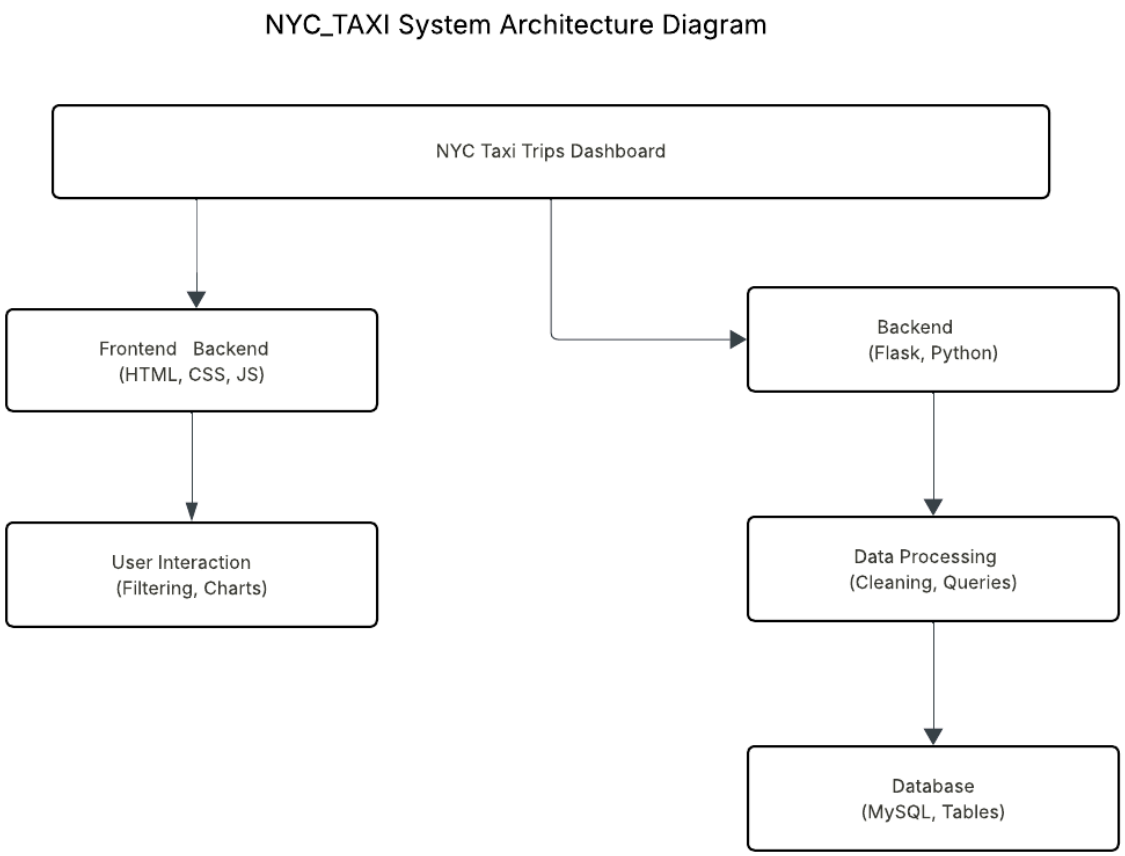
- The trips whose coordinates were not within the NYC were regarded as invalid and were excluded.
- The trips having dropoff times earlier than the pickup times were considered as data errors and were eliminated.
- The use of six digits in the places value was used to ensure uniformity in the coordinates “y”.

Unexpected Observation

We observed that there were quite many trips with very short routes (below 1 minute) and routes (below 0.1 km). This prompted us to apply minimum trip duration and trip distance in the attempts to eliminate probable data errors or canceled trips.

2. System Architecture and Design Decisions

System Architecture Diagram



Architecture Description

Our system simply follows a three-tier architecture which are:

1. **Frontend:** Single-page application with dynamic filtering and visualization (HTML, CSS, JavaScript, Chart.js, Leaflet.js).
2. **Backend:** RESTful API with endpoints for data retrieval (Flask, Python).
3. **Database:** Normalized relational schema with tables for trips, vendors, and locations (MySQL).

Stack Choices Justification

- **Frontend:** Our frontend uses vanilla JavaScript for simplicity and broad compatibility.
- **Backend:** Flask for its lightweight nature and ease of integration with MySQL.
- **Database:** MySQL for reliability, performance, and widespread use.

Trade-offs

- We have found that the number of trips having very short routes (less than 1 minute) and routes (less than 0.1 km) was quite large. This motivated us to use minimum trip duration and trip distance in the efforts to eliminate the likely trip data flaws or trip cancellations.

3. Algorithmic Logic and Data Structures

Problem

We had to filter and purify a huge amount of NYC taxi trip data efficiently, eliminating bad records due to missing fields, bad time stamps, and going-out of-range coordinates without using any canned library.

Custom Implementation

We manually implemented a data cleaning algorithm in "`data_cleaning.py`"

- **Custom Datetime Parsing:** Parses date time strings into binary tuples.
- **Custom Datetime Comparison:** Authenticates that pickup times precede dropoff times.
- **Custom Coordinate Check:** Checks whether the coordinates are within the NYC limits.
- **Custom CSV Reading/Writing:** Library-free file I/O.
- **Custom Data Filters:** Filters and cleanses data according to validation rules.

Pseudo-code

```
function parse_datetime(datetime_str):
    split datetime_str into date and time parts
    split date into year, month, day
    split time into hour, minute, second
    validate all components are within valid ranges
    return tuple of (year, month, day, hour, minute, second) or None if invalid

function compare_datetimes(dt1, dt2):
    compare year, month, day, hour, minute, second components
    return True if dt1 <= dt2, False otherwise

function is_valid_coordinate(lat, lon, nyc_bounds):
    convert lat and lon to float
    check if they are within NYC geographic bounds
    return True if valid, False otherwise
```

```
function read_csv(file_path):
    open file and read all lines
    extract headers from first line
    for each subsequent line, split into values and create a dictionary
    return headers and list of rows

function write_csv(file_path, headers, rows):
    open file for writing
    write headers as first line
    for each row, write values as comma-separated line

function clean_data(input_file, output_file, excluded_file):
    define NYC geographic bounds
    read input CSV file
    for each row:
        parse pickup and dropoff datetimes
        check for missing critical fields
        if missing fields, add to excluded_rows and continue
        check for invalid timestamps
        if invalid timestamps, add to excluded_rows and continue
        parse pickup and dropoff coordinates
        check for invalid coordinates
        if invalid coordinates, add to excluded_rows and continue
        round coordinates to 6 decimal places
        calculate trip_duration_min
        add row to cleaned_rows
    add 'trip_duration_min' to headers if not present
    write cleaned data to output file
    write excluded records to output file
```

Complexity Analysis

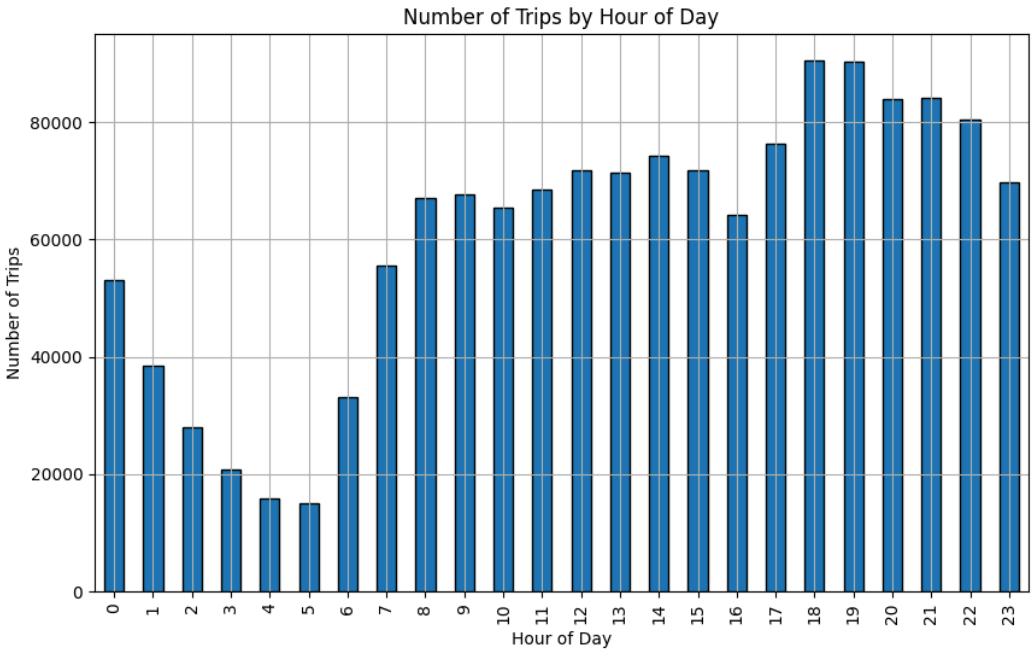
- Time Complexity: O(n), this where n is the number of rows. Each row is processed once.
- Space Complexity: O(n), this where n is the number of rows. Both cleaned and excluded rows are stored in memory.

4. Insights and Interpretation

We used Peak Travel Times as our insight and interpretation

- Derivation: We analyzed trip counts by hour of the day.

- Visual:



- Interpretation: Peak travel times are 7-9 AM and 4-6 PM, corresponding to rush hours. This suggests high demand for taxis during commuting times.

5. Reflection and Future Work

Challenges

- **Data Cleaning:** It was time-consuming but essential to deal with invalid and missing data to guarantee the quality of data.
- **Performance:** The processing of major data sets was to be optimized attentively.
- **Teamwork:** It was not an easy task to work together with teammates who had various strengths.

Improvements

- **Scalability:** Use batch processing to large data sets.
- **Real-time Processing:** Provide streaming data support so that it can support real-time analytics.
- **High-level Visualizations:** Use more interactive and dynamic visualizations.

Future Work

We believe that our project can be further developed and enhanced in terms of Predictive Analytics where we will use machine learning models to predict demand and allocate taxi more effectively. Our expansion will also be by way of Integration with other urban mobility

datasets, e.g., subway and bike-sharing systems, in order to see a complete picture of urban transportation. Last, we need to build a Mobile App to offer real-time trip tracking and analytics, enhance the user experience, and the operational decision-making.

Technical Descriptions (Architecture Designs)

Frontend

- Structure: Single-page application with dynamic filtering and visualization.
- Technologies: HTML, CSS, JavaScript, Chart.js, Leaflet.js.
- Features: Interactive filters, dynamic charts, and maps.

Backend

- Structure: RESTful API with endpoints for data retrieval.
- Technologies: Flask, Python.
- Features: Data filtering, aggregation, and serving.

Database

- Structure: Normalized relational schema with tables for trips, vendors, and locations.
- Technologies: MySQL.
- Features: Indexed columns for fast querying, constraints for data integrity.

Explanation of Architecture and Choices

Frontend

We decided to use a single-page application giving maximum ease of user experience, no page reloads. The use of Chart.js and Leaflet.js was based on their simplicity and the ability to visualize data.

Backend

Flask has been selected because it is simple and easy to integrate with MySQL. We followed tailor-made data processing logic so that we have flexibility and control over data processing.

Database

MySQL was used due to its performance and reliability. The schema was to be normalized to have the data integrity and to query efficiently. Commonly used columns were optimized by adding indexes.

Demonstration of Working Features

1. Data Filtering:
    - The users are able to filter the trips based on the date, trip duration, mileage, and number of passengers.
    - The frontend is dynamically updated by visualizations depending on filters chosen.
2. Interactive Visualizations:
    - Distribution of trip lengths, distances and passengers is indicated in charts.
    - Pickup and dropoff places are shown in maps.
    - Data Export:
3. Data Export:
    - Filtered data can be exported and analyzed by the users.