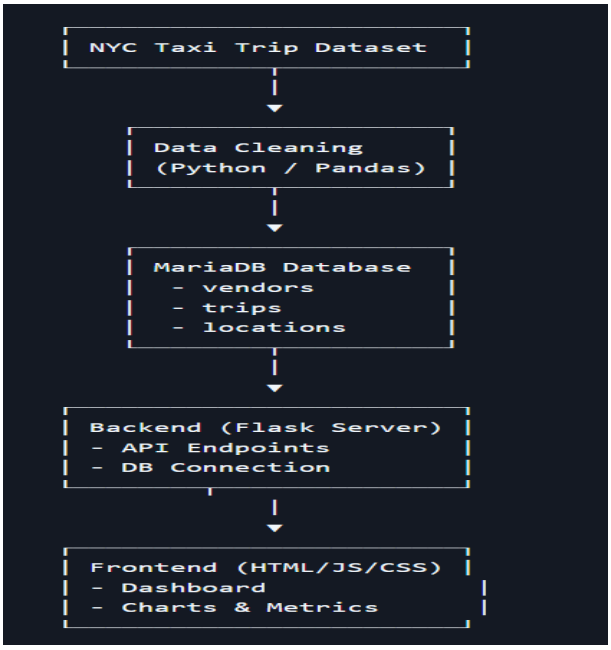
1. The train.csv file contains over one million taxi trip records across eleven columns, including id, vendor_id, timestamps, passenger_count, pickup/dropoff_coordinates, store_and_fwd_flag, and trip_duration. These fields enable the analysis of urban trips and the calculation of deliverables, including travel distances, average speeds, pickup hours, and weekdays. During processing, we handled missing coordinates/timestamps, duplicates, unrealistic durations/speeds, missing fare/tip data, and the absence of driver/taxi IDs. Trips with missing critical fields were excluded, trips exceeding 120 km/h were treated as outliers, and durations were assumed accurate in seconds. Travel distances were calculated using the Haversine formula, and very short trips with extremely high speeds, likely due to GPS or timestamp errors, motivated an outlier filter based on an average speed threshold

2. Our system uses a three-tier architecture with a web-based frontend (HTML, CSS, JavaScript) for visualization, a Flask backend in Python for database queries and custom algorithms, and a MySQL database with a normalized, indexed schema for efficient filtering. Flask was chosen for simplicity and Python integration, and MySQL for reliable relational queries. Trade-offs include using MySQL over NoSQL, implementing the Top-K algorithm in Python for manual control, and keeping the frontend lightweight at the expense of dynamic scalability.



3. The user needs to see the 10 longest taxi trips from the NYC dataset, so we implemented a custom Top-K algorithm. The algorithm iterates through each trip and compares it with the current top list. If a trip is longer than one in the list, it is inserted in the correct position, keeping only the ten longest trips. The following pseudo-code illustrates the approach for clarity.

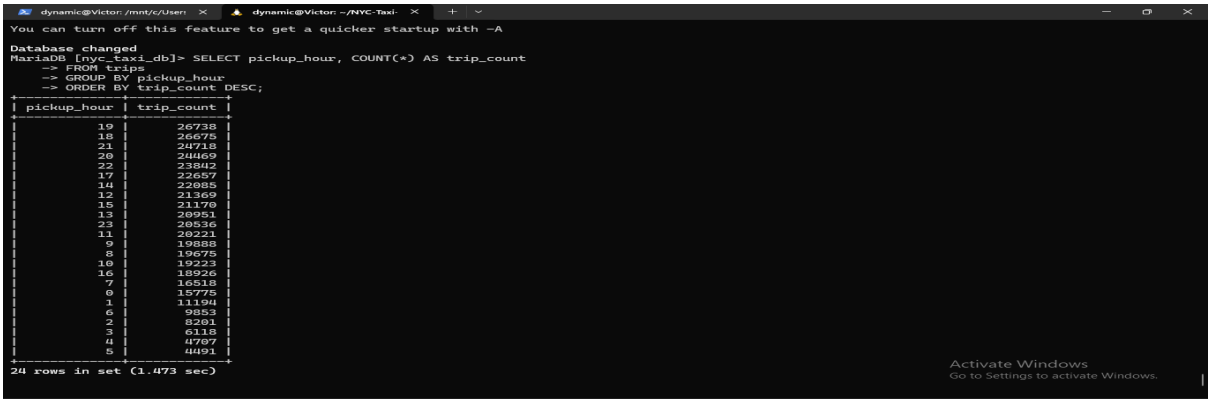
```
FUNCTION top_k_trips(trips, K):
    top_list = []                          # Initialize empty list to store top K trips

    FOR each trip IN trips:                 # Loop through all trip records
        inserted = FALSE                    # Flag to check if trip is added to top_list

        FOR i FROM 0 TO length(top_list)-1:
            IF trip.distance > top_list[i].distance:
                INSERT trip AT position i IN top_list
                inserted = TRUE
                BREAK

        IF inserted == FALSE AND length(top_list) < K:
            APPEND trip TO top_list        # Add trip if list has fewer than K items

        IF length(top_list) > K:
            REMOVE last element            # Keep list size at K

    RETURN top_list                         # Return top K trips
```

The algorithm has O(N × K) time and O(K) space complexity, making it efficient and scalable. It provides meaningful insights to the frontend by manually ranking the top trips without sorting the full dataset.

4. Insights and Interpretation

Insight 1: Peak Travel Hours

Result:



Interpretation:

Most trips occur between 7pm - 10pm with 7pm being the busiest hour with the highest trip count [26738]. This shows most trips occur after work hours and provides important data for optimizing taxi availability in the identified timeframe.

Insight 2: Average Trip Duration by Day of Week

Result:

```
MariaDB [nyc_taxi_db]> SELECT pickup_dayofweek, AVG(trip_duration) AS avg_duration
    -> FROM trips
    -> GROUP BY pickup_dayofweek;
+------------------+--------------+
| pickup_dayofweek | avg_duration |
+------------------+--------------+
|                0 |     904.2779 |
|                1 |     912.0999 |
|                2 |     997.9704 |
|                3 |     967.3560 |
|                4 |    1002.8282 |
|                5 |     995.8047 |
|                6 |     910.4931 |
+------------------+--------------+
7 rows in set (0.329 sec)
```

Interpretation:
Trips on Fridays tend to have longer durations [1002.8282], likely due to heavy traffic at the end of the workweek. This kind of information can guide transportation planners in improving traffic management and encouraging flexible work hours to reduce congestion.

Insight 3: Vendor Market Share
Result:

```
MariaDB [nyc_taxi_db]> SELECT v.vendor_id, COUNT(t.trip_id) AS total_trips
    -> FROM vendors v
    -> JOIN trips t ON v.vendor_id = t.vendor_id
    -> GROUP BY v.vendor_id;
+-----------+-------------+
| vendor_id | total_trips |
+-----------+-------------+
|         1 |      200199 |
|         2 |      229801 |
+-----------+-------------+
2 rows in set (0.197 sec)
```

Interpretation:
Two vendors dominate a large percentage of total trips which shows an imbalance in market share. This shows either a stronger network or brand recognition, but it could also mean limited competition, leading to higher fares.

5. Reflection and Future Work

Technical and Team Challenges

- **Database Design:** Correct referencing in the tables vendors, trips, and locations required close attention and adjustments to match the original data provided.
- **Performance:** Large CSV files and datasets required optimization through batching and indexing to prevent slow insert operations.

```
(nyc_taxi_env)(dynamic㉿Victor)-[~/NYC-Taxi-Mobility-Insights/database]
$ python insert_cleaned_taxi_data.py
Inserted 10000 rows so far...
Inserted 20000 rows so far...
Inserted 30000 rows so far...
Inserted 40000 rows so far...
Inserted 50000 rows so far...
Inserted 60000 rows so far...
Inserted 70000 rows so far...
Inserted 80000 rows so far...
Inserted 90000 rows so far...
Inserted 100000 rows so far...
Inserted 110000 rows so far...
Inserted 120000 rows so far...
Inserted 130000 rows so far...
Inserted 140000 rows so far...
Inserted 150000 rows so far...
Inserted 160000 rows so far...
Inserted 170000 rows so far...
Inserted 180000 rows so far...
Inserted 190000 rows so far...
Inserted 200000 rows so far...
Inserted 210000 rows so far...
Inserted 220000 rows so far...
Inserted 230000 rows so far...
Inserted 240000 rows so far...
Inserted 250000 rows so far...
Inserted 260000 rows so far...
Inserted 270000 rows so far...
Inserted 280000 rows so far...
Inserted 290000 rows so far...
```