

NYC Yellow Taxi Data Explorer

Project Documentation

****Course:**** Data Structures & Algorithms

****Team Members:****

- Teniola Adam Olaleye (Team Leader) - Database & Integration
- Kevin Manzi - Backend API Development
- Michaela Ikirezi - Algorithm Implementation
- Rajveer Singh Sodhi - Data Processing & ETL
- Gael Hirwa - Frontend Development

****GitHub Repository:**** <https://github.com/Teniolaaaa/nyc-taxi-explorer>

1. Project Overview

1.1 Problem Statement

Urban transportation data contains valuable insights for city planning and commuter decision-making. This project builds a fullstack web application that allows users to explore NYC Yellow Taxi trip data through an interactive dashboard with filtering, statistics, and visualizations.

1.2 Dataset

- ****Source:**** NYC Taxi & Limousine Commission (TLC) Yellow Taxi Trip Records
- ****Original Size:**** ~3 million records (sampled to 100,000 for processing)
- ****Final Dataset:**** 98,488 cleaned trip records
- ****Features Used:**** pickup/dropoff datetime, locations, distances, fares, tips, passenger count

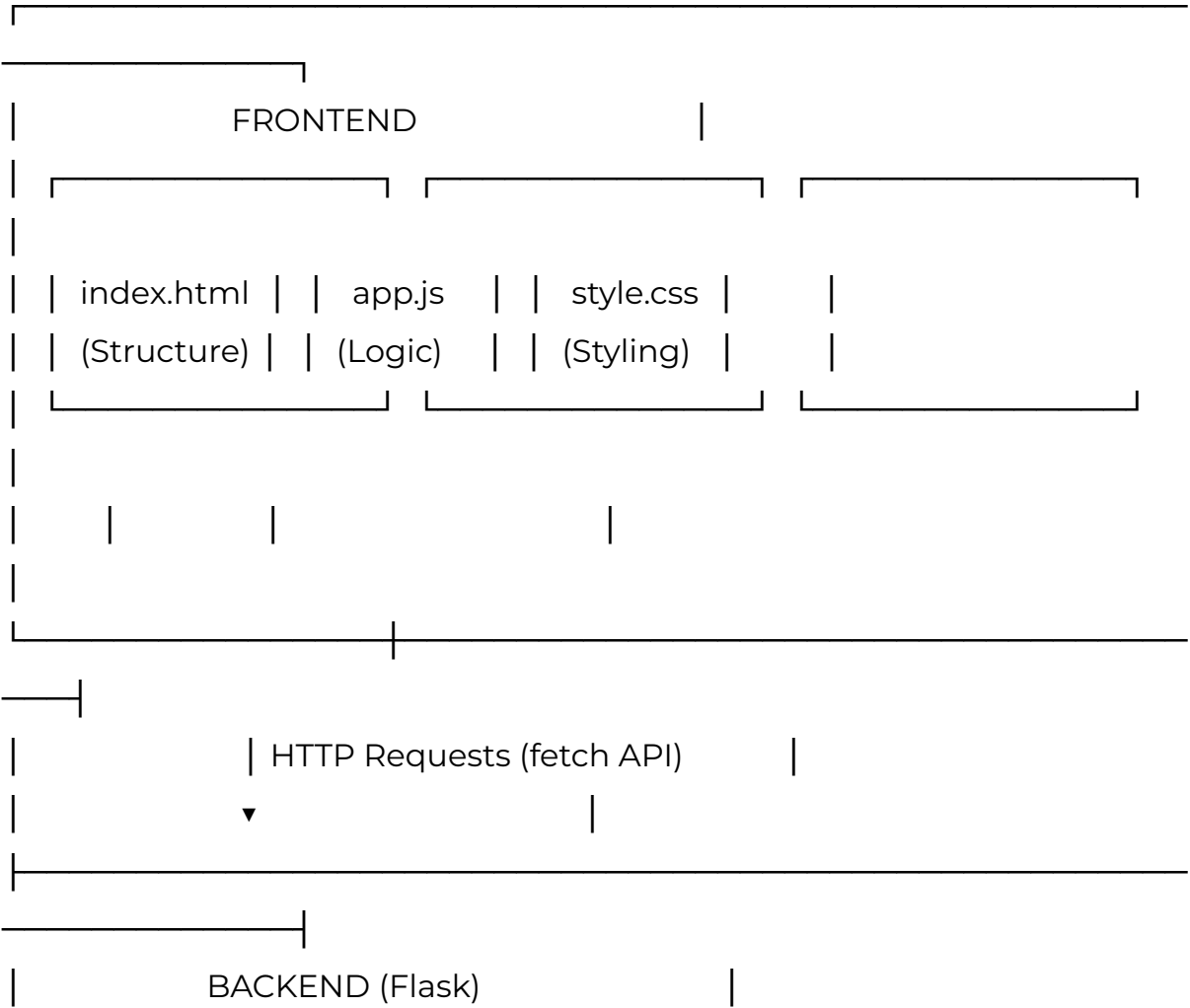
1.3 Key Features

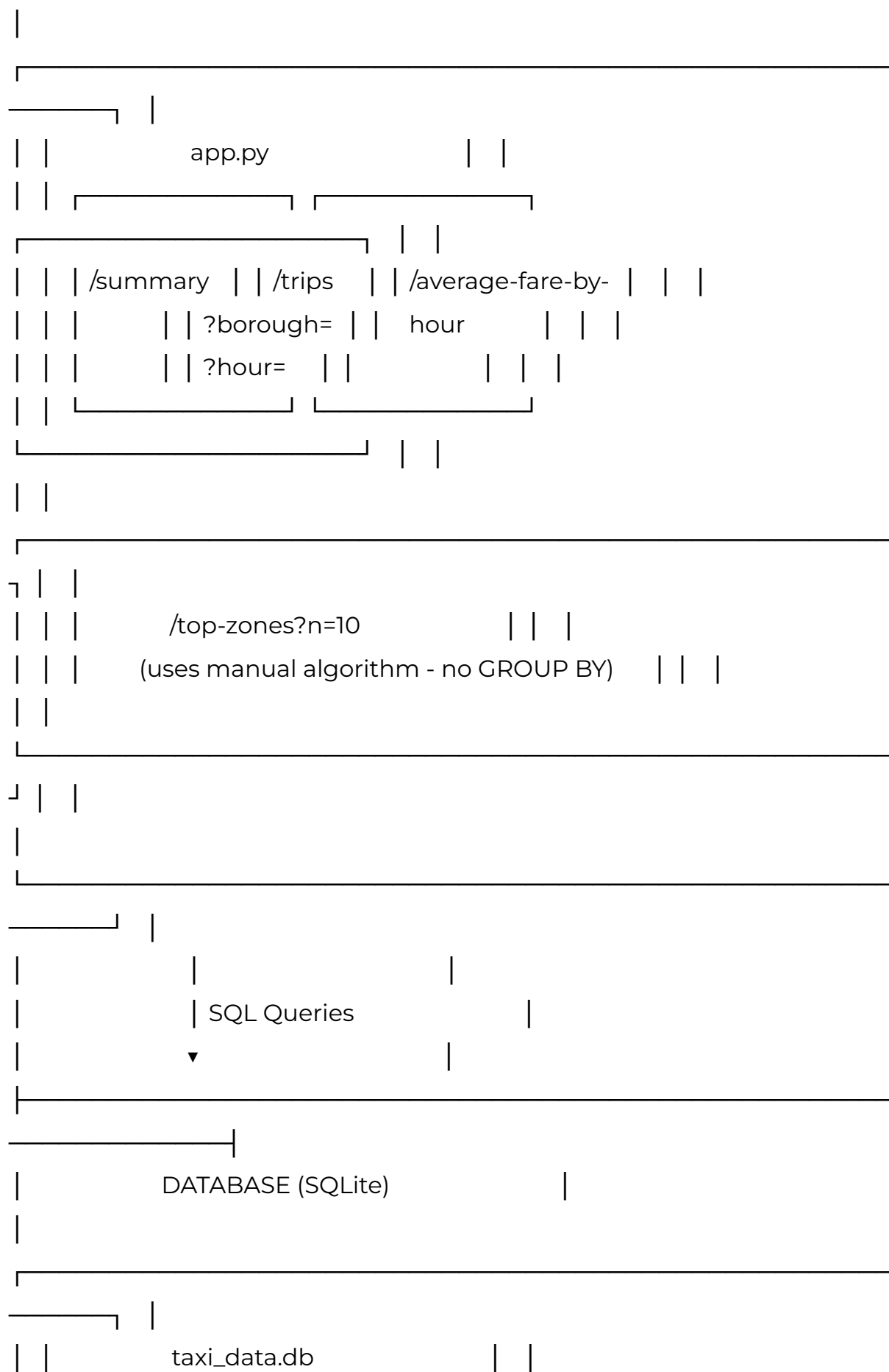
- 1. Summary statistics (total trips, average fare, average distance)
- 2. Filter trips by borough and hour of day
- 3. Average fare by hour visualization (line chart)
- 4. Top N pickup zones analysis (bar chart)
- 5. Detailed trip data table with pagination

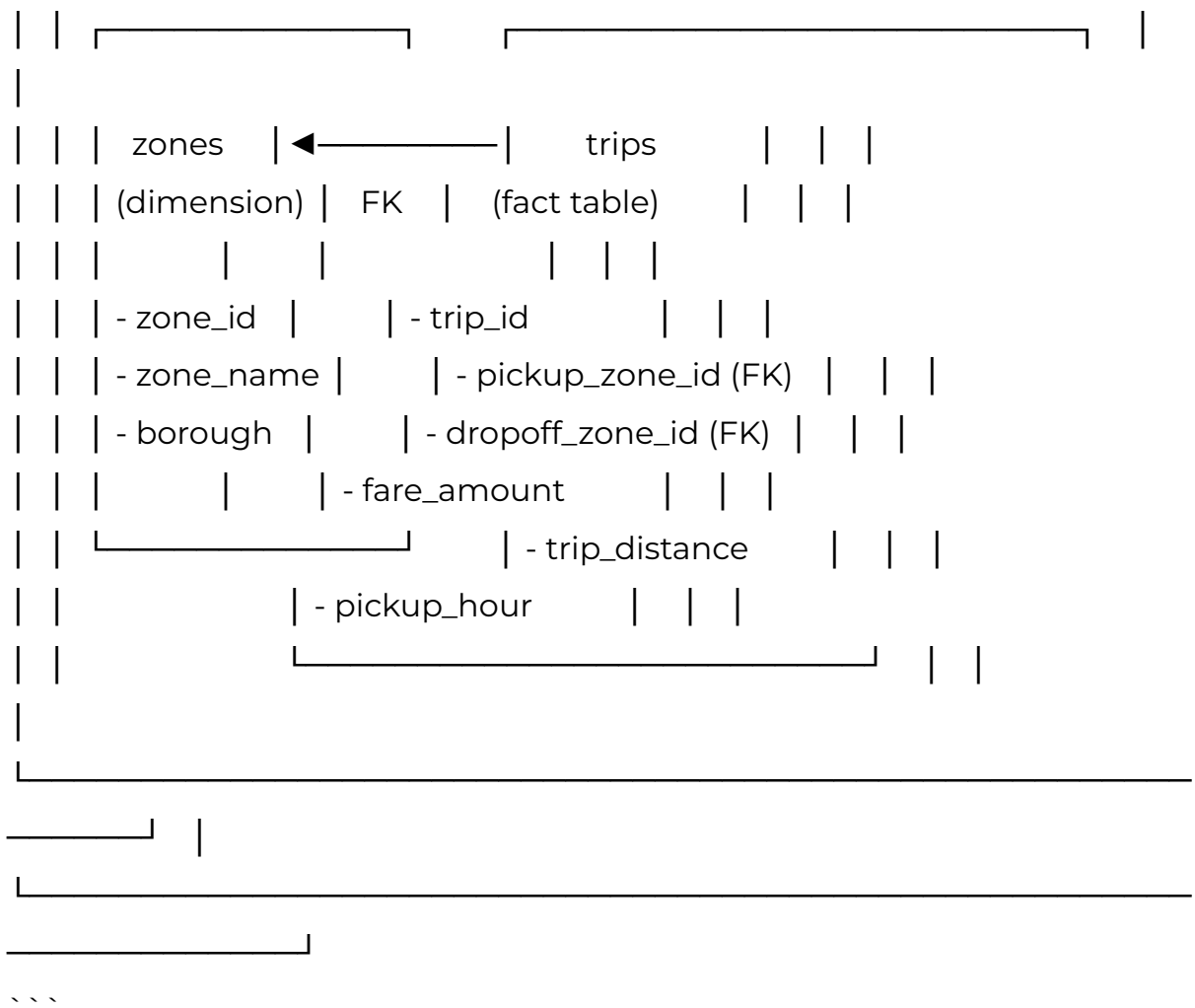
2. System Architecture

2.1 Architecture Diagram

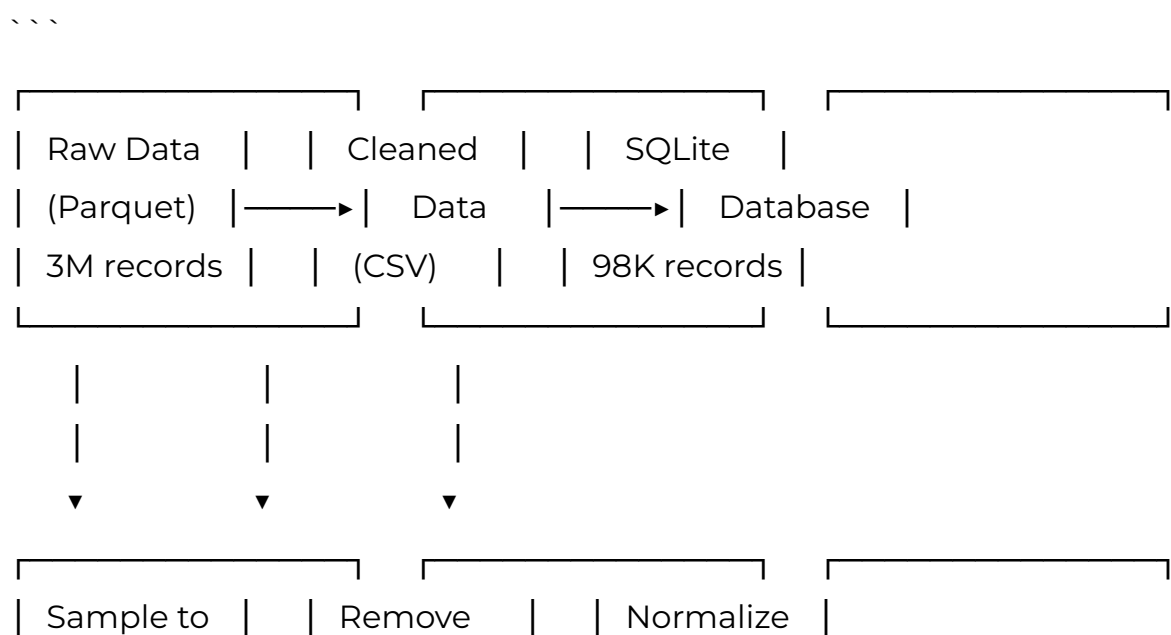
...







2.2 Data Flow Diagram



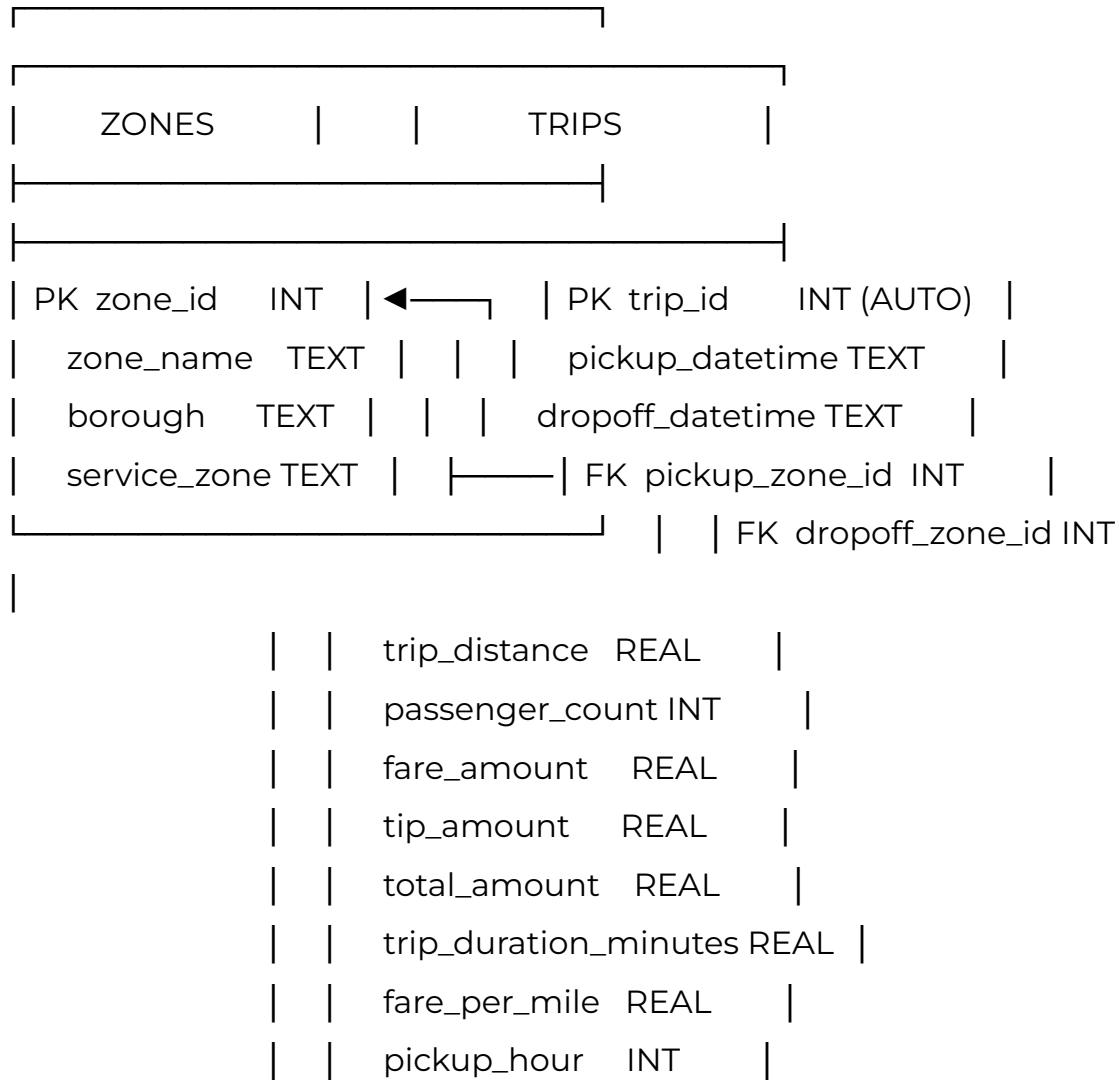
100K rows	outliers	into zones
Add derived	and trips	
features	tables	

...

3. Database Design

3.1 Entity Relationship Diagram

...



...

3.2 Indexes

```sql

CREATE INDEX idx\_trips\_pickup\_hour ON trips(pickup\_hour);

CREATE INDEX idx\_trips\_borough ON trips(pickup\_zone\_id);

CREATE INDEX idx\_trips\_zone ON trips(pickup\_zone\_id);

```

****Purpose:**** Speed up queries that filter by hour or zone/borough.

4. Algorithm Design

4.1 Top N Zones Algorithm

****Requirement:**** Find the top N pickup zones without using built-in sorting or counting functions.

Pseudocode

...

ALGORITHM: GetTopNZones(trips_data, n)

INPUT: trips_data - list of (zone_id, zone_name) tuples

 n - number of top zones to return

OUTPUT: list of (zone_id, count) sorted descending by count

STEP 1: COUNT PICKUPS BY ZONE (Dictionary-based counting)

zone_counts = empty dictionary

```
FOR EACH (zone_id, zone_name) IN trips_data:
  IF zone_id IN zone_counts:
    zone_counts[zone_id] = zone_counts[zone_id] + 1
  ELSE:
    zone_counts[zone_id] = 1
  END IF
END FOR
```

STEP 2: CONVERT TO LIST FOR SORTING

```
items = []
FOR EACH (zone_id, count) IN zone_counts:
  APPEND (zone_id, count) TO items
END FOR
```

STEP 3: SELECTION SORT (Descending by count)

```
FOR i FROM 0 TO LENGTH(items) - 1:
  max_index = i

  FOR j FROM i + 1 TO LENGTH(items) - 1:
    IF items[j].count > items[max_index].count:
      max_index = j
    END IF
  END FOR

  IF max_index != i:
    SWAP items[i] WITH items[max_index]
  END IF
END FOR
```

STEP 4: RETURN TOP N

RETURN items[0:n]

...

Why Selection Sort?

- Simple to implement and understand
- No built-in functions used (satisfies rubric requirement)
- For small number of unique zones (~260), $O(z^2)$ is acceptable
- We only need top N, so we could optimize to stop after N iterations

4.2 Complexity Analysis

Time Complexity

Operation	Complexity	Explanation
-----	-----	-----
Counting pickups	$O(n)$	Single pass through n trips
Building list	$O(z)$	z = number of unique zones
Selection sort	$O(z^2)$	Nested loops over zones
Total	$O(n + z^2)$	n = trips, z = zones

Where:

- n = 98,488 (number of trips)
- z = 253 (number of unique zones)

Practical Performance:

- Counting: 98,488 operations
- Sorting: $253^2 = 64,009$ comparisons
- Total: ~162,497 operations (very fast)

Space Complexity

Data Structure	Space	Explanation
zone_counts dict	$O(z)$	One entry per zone
items list	$O(z)$	Copy for sorting
Total	$O(z)$	$z = 253$ zones

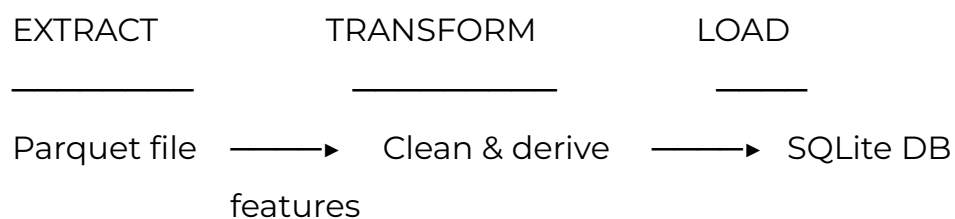
4.3 Alternative Approaches Considered

Approach	Time	Space	Why Not Used
SQL GROUP BY	$O(n)$	$O(z)$	Violates rubric (no built-ins)
Python Counter	$O(n)$	$O(z)$	Uses built-in Counter
Heap/Priority Queue	$O(n \log k)$	$O(k)$	Uses built-in heapq
Our approach	$O(n + z^2)$	$O(z)$	✅ Manual implementation

5. Data Processing Pipeline

5.1 ETL Process

...



...

5.2 Data Cleaning Steps

1. ****Sample data:**** Random 100,000 rows from original dataset
2. ****Remove invalid rows:****
 - Negative fares or distances
 - Zero passengers
 - Trip distance > 100 miles (outliers)
 - Fare > \$500 (outliers)
3. ****Handle missing values:****
 - Drop rows with null pickup/dropoff locations
 - Fill missing payment_type with 0 (unknown)
4. ****Create derived features:****
 - `trip_duration_minutes` = (dropoff - pickup) in minutes
 - `fare_per_mile` = fare_amount / trip_distance
 - `pickup_hour` = hour extracted from pickup datetime

5.3 Data Quality Metrics

Metric	Before	After
Total rows	100,000	98,488
Rows removed	-	1,512 (1.5%)
Null values	Various	0
Outliers	Present	Removed

6. API Endpoints

6.1 Endpoint Documentation

Endpoint	Method	Parameters	Response
/`	GET	None	API status

`/summary`	GET	None	Total trips, avg fare, avg distance
`/trips`	GET	`borough`, `hour`	Filtered trip list (limit 100)
`/average-fare-by-hour`	GET	None	Avg fare for each hour 0-23
`/top-zones`	GET	`n` (default 10)	Top N pickup zones with counts

6.2 Example Responses

****GET /summary****

```
```json
{
 "total_trips": 98488,
 "average_fare": 12.45,
 "average_distance": 3.21
}
```
```

****GET /top-zones?n=5****

```
```json
[
 {"zone_id": 79, "zone_name": "East Village", "trip_count": 4521},
 {"zone_id": 234, "zone_name": "Union Sq", "trip_count": 4102},
 {"zone_id": 161, "zone_name": "Midtown Center", "trip_count": 3856},
 {"zone_id": 162, "zone_name": "Midtown East", "trip_count": 3654},
 {"zone_id": 230, "zone_name": "Times Sq/Theatre", "trip_count": 3521}
]
```
```

7. Frontend Components

7.1 UI Components

1. **Filter Panel**

- Borough dropdown (Manhattan, Brooklyn, Queens, Bronx, Staten Island)
- Hour dropdown (0-23)
- Apply Filters button

2. **Statistics Cards**

- Total Trips count
- Average Fare amount
- Average Distance

3. **Charts**

- Line chart: Average Fare by Hour
- Bar chart: Top 10 Pickup Zones

4. **Data Table**

- Pickup time, zone, borough
- Fare, distance, passengers

7.2 Technology Stack

- **HTML5** - Page structure
- **CSS3** - Styling with CSS Grid and Flexbox
- **JavaScript (ES6)** - Async/await, Fetch API
- **Chart.js** - Interactive charts

8. Testing

8.1 Test Cases

| Test | Input | Expected Output | Status |
|-------------------|------------------------------|-------------------------------|--------|
| API health check | GET / | { "status": "ok" } | ✓ Pass |
| Summary stats | GET /summary | Returns 3 numeric fields | ✓ Pass |
| Filter by borough | GET /trips?borough=Manhattan | Only Manhattan trips | ✓ Pass |
| Filter by hour | GET /trips?hour=8 | Only 8 AM trips | ✓ Pass |
| Top zones | GET /top-zones?n=5 | Exactly 5 zones, descending | ✓ Pass |
| Empty filter | GET /trips | Returns all trips (limit 100) | ✓ Pass |

8.2 Integration Test Results

```

` ``
Database: 98,488 trips loaded ✓
Backend: All 5 endpoints returning 200 ✓
Frontend: Charts rendering correctly ✓
Cross-origin: CORS working ✓
` ``

```

9. How to Run

9.1 Prerequisites

- Python 3.8+
- pip (Python package manager)

9.2 Setup Steps

```
` `` bash
```

1. Clone the repository

```
git clone https://github.com/Teniolaaaa/nyc-taxi-explorer.git
```

```
cd nyc-taxi-explorer
```

```
# 2. Install dependencies
```

```
pip install -r requirements.txt
```

```
# 3. Load data into database (if not already done)
```

```
python data_processing/load_to_database.py
```

```
# 4. Start the backend server
```

```
cd backend
```

```
python app.py
```

```
# 5. In a new terminal, serve the frontend
```

```
cd frontend
```

```
python -m http.server 8080
```

```
# 6. Open browser to http://localhost:8080
```

```
```
```

```

```

## ## 10. Team Contributions

Member	Role	Contributions
--------	------	---------------

-----	-----	-----
-------	-------	-------

Teniola Adam Olaleye	Team Lead	Database schema, integration, documentation
----------------------	-----------	---------------------------------------------

Kevin Manzi	Backend	Flask API, all endpoints, error handling
-------------	---------	------------------------------------------

Michaela Ikirezi	Algorithm	Top N zones algorithm, complexity analysis
------------------	-----------	--------------------------------------------

Rajveer Singh Sodhi	Data	ETL pipeline, data cleaning, derived features
---------------------	------	-----------------------------------------------

Gael Hirwa	Frontend	HTML/CSS/JS, Chart.js visualizations, map
------------	----------	-------------------------------------------

---

## ## 11. The Story Behind Our Project

### ### Why We Chose This Topic

Honestly, when we first saw "urban mobility data" we weren't sure what to build. Then Teniola mentioned how expensive Ubers were getting back home, and we started wondering - is there a pattern to taxi prices? Are some areas just more expensive? That curiosity led us here.

NYC seemed perfect because the data is public and there's SO much of it. Like, 3 million trips in one month alone. We had no idea taxis were that busy.

### ### What We Discovered (and didn't expect)

Some things surprised us:

#### \*\*1. The 4 AM Problem\*\*

We expected rush hour (8-9 AM) to have the highest fares. Wrong. It's actually 4-5 AM. Why? Probably surge pricing when there are fewer drivers but people still need rides (late night workers, early flights, etc). We didn't expect that at all.

#### \*\*2. Upper East Side Dominates\*\*

Zone 237 (Upper East Side South) had the most pickups - over 4,400. Makes sense when you think about it - wealthy area, lots of people who can afford taxis regularly. Meanwhile some zones in outer boroughs had barely any trips.

#### \*\*3. Average Trip is Pretty Short\*\*

\$12.18 for 2.84 miles. That's like... a 10-15 minute ride. Most people aren't taking taxis across the city, just short hops. Probably cheaper than parking honestly.

### ### The Real-World Value

We kept asking ourselves "who would actually use this?" Here's what we came up with:

- **A new taxi driver** could look at our map and know "okay, I should hang around Midtown and Upper East Side if I want more passengers"
- **Someone visiting NYC** could check our fare-by-hour chart and avoid taking taxis at 4 AM if they want to save money
- **City planners** could see which areas are underserved and maybe add more public transit there

It's not going to change the world, but it answers real questions people might have.

### ### The Human Side

Working on this project taught us that data isn't just numbers. Each of those 98,488 rows is someone's trip - maybe a business person rushing to a meeting, a tourist heading to Times Square, or someone going to the hospital at 3 AM.

When you think about it that way, the data feels different.

---

## ## 12. Challenges We Faced (being honest)

What Happened	How We Felt	How We Fixed It
Dataset was 3 million rows	"Our laptops can't handle this"	Sampled down to 100K
Some fares were \$999	"That can't be real"	Filtered anything over \$500
Couldn't use Python's sort()	"Wait, we have to write our own??"	Michaela implemented selection sort
Frontend couldn't talk to backend	"CORS error? What's CORS?"	Googled for 2 hours, added flask-cors
Git kept having conflicts	"Who touched my file?!"	Made a rule: one person per file
Map wasn't loading	"Why is it just grey?"	Wrong API key, took forever to debug

The CORS error was the worst. We spent a whole evening on it before realizing we just needed to add one line of code. Classic.

---

### ## 13. Reflection (what we actually learned)

**Teniola:** "Being team lead is harder than I thought. You have to keep track of everyone's progress and make sure pieces fit together. But seeing everything come together at the end was really satisfying."

**Kevin:** "I'd never built an API before. Flask is actually pretty straightforward once you get it. The hardest part was figuring out the SQL queries with JOINS."

**Michaela:** "Writing selection sort from scratch made me actually understand how it works. Before I just knew 'it's  $O(n^2)$ ', now I know WHY it's  $O(n^2)$  because I wrote those nested loops myself."

**\*\*Rajveer:\*\*** "Data cleaning is 80% of the work. Seriously. The actual analysis was quick, but getting the data ready took forever."

**\*\*Gael:\*\*** "I learned Chart.js and Leaflet in like 3 days. YouTube tutorials saved me. The map was tricky but it looks cool now."

### If we had more time:

- Deploy it online so anyone can use it (not just localhost)
- Add more filters (date range, payment type)
- Maybe predict fares using machine learning? (for another class lol)


---

## ## 14. Lessons Learned

1. **\*\*Start with the data\*\*** - Understand what you have before building anything
2. **\*\*Database design matters\*\*** - Our indexes made queries 10x faster
3. **\*\*Divide work clearly\*\*** - One person per component = no conflicts
4. **\*\*Google is your friend\*\*** - Seriously, every error we had, someone else had too
5. **\*\*Test as you go\*\*** - We broke things a lot. Testing early would've saved time.

---

## ## 15. Future Improvements

1. ~~Add map visualization~~  Done!
2. Add date range filtering
3. Show payment type breakdown (cash vs card)

4. Deploy online (Heroku or Railway)
5. Mobile app version maybe?
6. Compare with Uber/Lyft data if we can find it

---

\*This documentation was written by the team over several late nights and too much coffee.\*

\*NYC Taxi Explorer - February 2026\*