**ST207 Project**

London Tube System Database

GitHub Repository:

<https://github.com/lse-st207/project-at2024-agoodteam.git>

Candidate Numbers

39246

44040

47813

48590

**London Tube System Topic Description**

The London Tube system is one of the world's most iconic and complex urban transportation networks, serving millions of passengers daily across an extensive network of stations, lines, and routes. Our database captures critical aspects of this system, including station details, passenger counts, route information, weather conditions, carbon emissions, and other operational metrics.

This database has wide-ranging applications, including enhancing the efficiency of Tube operations, reducing environmental impact, and improving the passenger experience. Transport authorities can leverage these insights to make data-driven decisions, such as targeting high-emission routes for sustainability initiatives, optimizing service schedules for underused stations, and identifying bottlenecks in the system to improve reliability. By providing a clear view of system-wide performance, the database serves as a vital tool for managing and optimizing one of the world's busiest urban transit networks.

**Data Description**

**Real Data:**

Data Source:

* Transport for London (TfL) Open Data, URL: <https://tfl.gov.uk/info-for/open-data-user>
* TfL Live Updates API, URL: [api.tfl.gov.uk](https://api.tfl.gov.uk/)

Our dataset integrates real-world data from official sources to ensure accuracy and consistency. All station information, including station names, StopNaptanId (used as StationID in our database), latitude, longitude, and fare zones, was retrieved directly from the TfL Open Data API. This will be explained in detail in next subsection. The fare zone details were initially identified with the assistance of ChatGPT and later manually validated against official TfL sources to ensure correctness.  
In addition to station data, all route and line information was manually extracted from TfL’s official datasets. These datasets provide a comprehensive view of the underground network, including distinct operational directions and branch routes. To complement this, we incorporated real passenger count data, which was derived from average Sunday passenger flow in 2022. This data, originally recorded in 15-minute intervals, was aggregated into hourly segments to maintain usability and consistency.

Extracting Data Process:

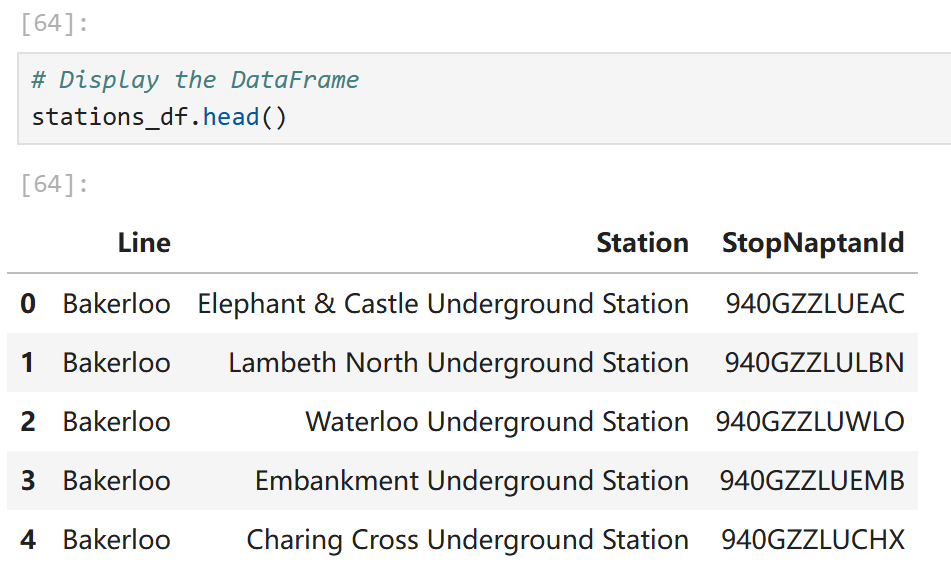
* Web Scraping for Station Information:

Target URL:

base\_url = "<https://tfl.gov.uk/tube/route/>"

(Exception: elizabeth\_url = "https://tfl.gov.uk/elizabeth-line/route/elizabeth”)

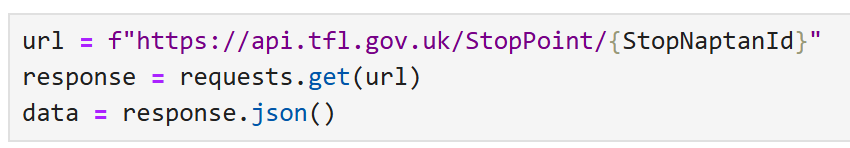
To gather London Underground station details from the TfL website, we developed a Python-based web crawler utilizing libraries such as requests, BeautifulSoup, pandas, and re. The scraper automated the process of extracting relevant station attributes. An HTTP request was sent to the TfL website to retrieve the list of underground stations. Then the HTML content was parsed using BeautifulSoup, allowing us to extract key attributes such as Station Name, StopNaptanId (Unique Identifier), Latitude & Longitude. Finally, the extracted data was cleaned and structured into a tabular format, making it compatible with our database.



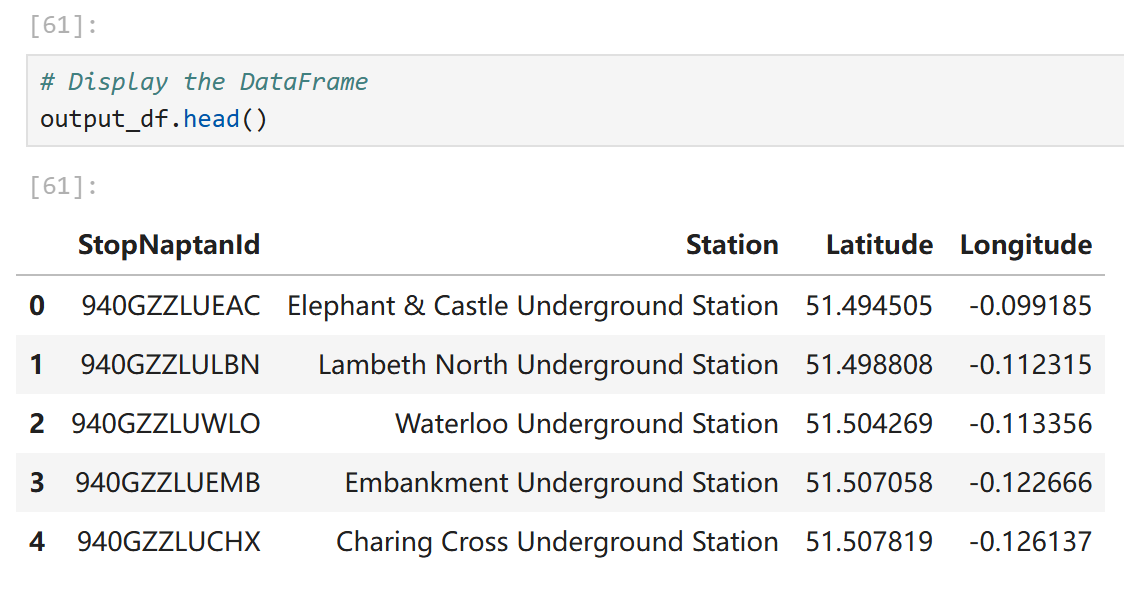
* API Retrieval for TfL Data:

TfL API Description:[Our Unified API - Transport for London](https://tfl.gov.uk/info-for/open-data-users/api-documentation)

To enrich our dataset with additional station metadata, we utilized the TfL Open Data API, specifically the StopPoint Endpoint. This allowed us to retrieve real-time details associated with each StopNaptanId. A structured request was formulated as follows:

1  


1  
The API response provided additional details, including: Geolocation (Latitude & Longitude), Fare Zone Classification, Accessibility and Station Facilities. But we only request for the Geolocation.



There will be some replicated data because many stations may be the interchange stations so recorded repeatedly in this way of extracting data. When imported into an excel file, we use excel tools to delete any replicated data.

**Synthetic data:**

* Carbon Emission is calculated by Total Emissions (kg)=Distance (km)×Passenger Count×Emission Factor (kg CO2 /km per passenger). For the Emission Factor, we used the data from the European Energy Agency: Electric train: 0.041 kg CO2e/km per passenger; Modern electric trains: 0.035 kg CO2e/km per passenger.
* Due to the absence of publicly available train scheduling data, we assumed a fixed allocation of 15 vehicles per route as a placeholder. This assumption allows for basic analysis while providing flexibility for future refinements.
* Weather Condition and Delay information was generated by AI tools, where we only generated data for the week of 2024-12-30 to 2025-1-5. For Weather Conditions, we randomly generate rainfall, windspeed, and temperature and make sure that there is both severe and normal weather. For Delay, we generate four factors: passenger, weather, operational, and technical. The delay caused by the weather factor was ensured to correspond to the severe weather conditions.

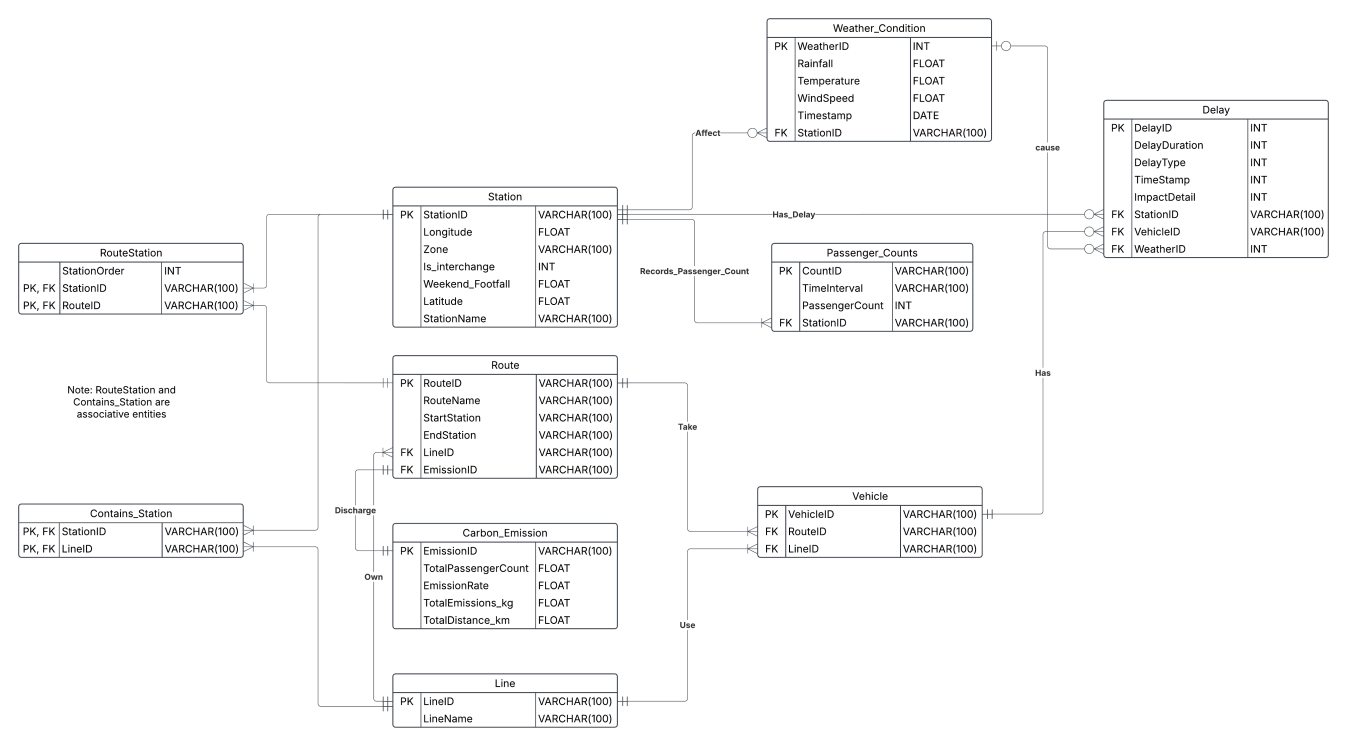
**Challenges Encountered:**

* One of the primary challenges we encountered was the strict request quota imposed by the TfL API. Since each station query required an individual request, excessive calls within a short timeframe led to rate-limiting issues.
* While our carbon emission calculations followed an established formula, real-world emissions are inherently more dynamic. The UK’s electricity grid composition fluctuates throughout the day, meaning that emissions from electric trains vary depending on the time of operation. Furthermore, different underground lines consume power differently. Deep Tube lines, which require extensive ventilation and cooling, likely have higher energy consumption compared to Overground services, which benefit from natural temperature regulation.
* Another challenge was the assumption of a uniform train allocation across all routes. While our dataset assigns 15 vehicles per route, real-world scheduling is highly dynamic. High-demand lines, such as the Central Line, operate at significantly higher frequencies compared to low-demand routes like the Waterloo & City Line. Additionally, scheduling adapts to peak and off-peak travel times, with increased train deployment during rush hours and reduced frequency during late-night operations. Without access to real-world operational data, it remains difficult to model these fluctuations accurately.

**Model Description**

**ER diagram**

We obtain the following ER diagram by using the Lucid Chart:

****

**Entities:**

**Station:** StationID (PK) uniquely identifies each station. StationName specifies its name, Zone indicates its location classification, Latitude and Longitude provide its geographical location, Is\_interchange shows whether it is a major transfer point, and Weekend\_Footfall measures passenger volume on weekends. Data Source: TfL Open Data API & Web Scraping, with StopNaptanId used as StationID.

**Line:** LineID (PK) uniquely identifies each tube line. LineName specifies the name of the line. Data Source: Extracted manually from TfL datasets.

**Route:** RouteID (PK) defines a specific tube path. RouteName gives its name, StartStation and EndStation indicate its terminal points. Data Source: Extracted from TfL datasets, ensuring accurate mapping of network branches.

**Vehicle:** VehicleID (PK) represents a transport unit. Data Source: Synthetic data

**Passenger\_Counts:** CountID (PK) uniquely identifies a count. TimeInterval specifies the time frame of the count, and PassengerCount measures the number of passengers. Data Source: Aggregated from 2022 Sunday passenger flow data provided by TfL.

**Carbon\_Emission:** EmissionID (PK) uniquely identifies the track of environmental impact. TotalPassengerCount reflects the number of passengers, EmissionRate shows emissions per kilometer, TotalEmissions\_kg provides total emissions, and TotalDistance\_km measures route length. Data Source: Synthetic data based on European Energy Agency emission factors.

**Weather\_Condition:** WeatherID (PK) uniquely identifies each weather data. Rainfall, Temperature, and WindSpeed capture meteorological conditions, while Timestamp records the time of observation. Data Source: AI-generated for a fixed period to simulate real-world variations.

**Delay:** DelayID (PK) uniquely identifies each disruption. DelayDuration indicates the length, DelayType classifies the cause, TimeStamp logs the occurrence, and ImpactDetail provides additional context. Data Source: AI-generated, with weather-related delays linked to severe conditions.

**Relationships:**

**1. Station Relationships:**

Station ↔ Passenger\_Counts (1:M)

* Each station can have multiple passenger count records, but each passenger count record belongs to one station.
* Example: "Oxford Circus" may have hourly passenger count records.

Station ↔ Weather\_Condition (1:M)

* Each station has multiple weather condition records recorded at different timestamps.
* Example: "King’s Cross" may have temperature and rainfall data logged hourly.

Station ↔ Delay (1:M)

* Each delay occurs at one station, but a station can have multiple delays over time.
* Example: A signal failure at "Liverpool Street" causes multiple delays over the day.

Station ↔ Line (M:N) via Contains\_Station

* A station can belong to multiple lines, and each line consists of multiple stations.
* Example: "Bank" station is served by the Central Line and Northern Line.

Station ↔ Route (M:N) via RouteStation

* A station can be part of multiple routes, and a route consists of multiple stations in a fixed order.
* Example: Route A passes through "Victoria" and "Green Park" in a set order.

**2. Line Relationships**

Line ↔ Station (M:N) via Contains\_Station

* A line has multiple stations, and a station can be part of multiple lines.
* Example: The Jubilee Line includes "Westminster" and "Canary Wharf," but "Westminster" is also served by the District Line.

Line ↔ Route (1:M)

* A line has multiple routes, but each route belongs to one line.
* Example: The Victoria Line has multiple routes, including northbound and southbound.

**3. Route Relationships**

Route ↔ Station (M:N) via RouteStation

* Each route includes multiple stations, and each station can appear in multiple routes.
* The order of stations in a route is maintained using StationOrder.

Route ↔ Vehicle (1:M)

* A route can have multiple vehicles assigned, but each vehicle operates on a single route at a time.
* Example: Multiple trains operate on the Piccadilly Line (Heathrow Route).

Route ↔ Carbon\_Emission (1:1)

* Each route has one carbon emission record, representing the total emissions for that route.
* Example: Route A has a total CO₂ emission of 500kg per day.

**4. Vehicle Relationships**

Vehicle ↔ Route (M:1)

* Each vehicle follows a specific route, but a route can have multiple vehicles.

Vehicle ↔ Delay (1:M)

* A delay is linked to a vehicle, meaning multiple delays can happen to the same vehicle over time.
* Example: A train on the Central Line experiences two delays in one day.

**5. Carbon Emission Relationships**

Carbon\_Emission ↔ Route (1:1)

* Each route has one associated carbon emission record based on the distance and passenger load.
* Example: Route X has higher emissions than Route Y due to longer distance.

**6. Delay Relationships**

Delay ↔ Station (M:1)

* Each delay is associated with one station, but a station can have many delays.

Delay ↔ Vehicle (M:1)

* Each delay affects one vehicle, but a vehicle can experience multiple delays.

Delay ↔ Weather\_Condition (M:1, Optional FK)

* Delays may be linked to weather conditions, but some delays (like signal failures) are unrelated to weather.
* Example: Heavy rain might be linked to delays in open-air stations.

**Constraints:**

* Primary Key Constraints: Each entity has a unique identifier, such as StationID for Station, RouteID for Route, and VehicleID for Vehicle, ensuring no duplicate records exist.
* Foreign Key Constraints: Referential integrity is maintained by linking entities. For example, StationID in Passenger\_Counts references Station(StationID), ensuring all passenger records correspond to an existing station.
* Not Null Constraints: Critical attributes such as StationName, RouteName, and DelayDuration cannot be null, preventing incomplete records.
* Unique Constraints: Attributes like LineName and RouteName must be unique to avoid confusion in route identification.

**Assumptions:**

* Each station belongs to at least one line.
* Routes always start and end at a designated station.
* A vehicle is assigned to a single route at any given time.
* Passenger counts are recorded at predefined time intervals.
* Weather conditions are recorded at stations, not along the entire route.
* Carbon emissions are calculated based on route distance and passenger volume.
* Routes are bidirectional, meaning the same set of stations applies for both directions.
* Vehicles cannot switch routes during operation.
* Missing passenger data is assumed to be zero rather than an error.
* Each line operates continuously unless there is a recorded delay.
* Scheduled maintenance or temporary closures are not considered in this dataset

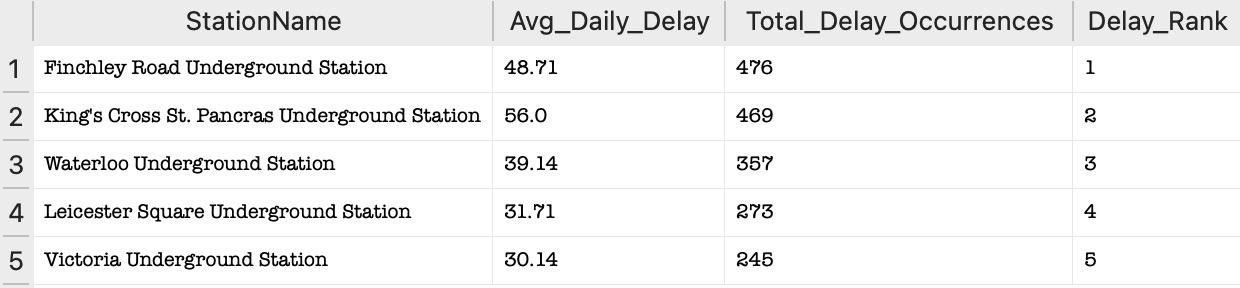
**Data Operations**

**Query:**

**Query 1: Identify the stations with high average daily delay and rank them with respect to the number of delays occurred.**

***Use of query*:** This SQL query analyzes delay patterns at Tube stations by aggregating delay data and ranking stations based on the number of delay occurrences. The goal is to identify stations with frequent delays and significant average daily delay durations.

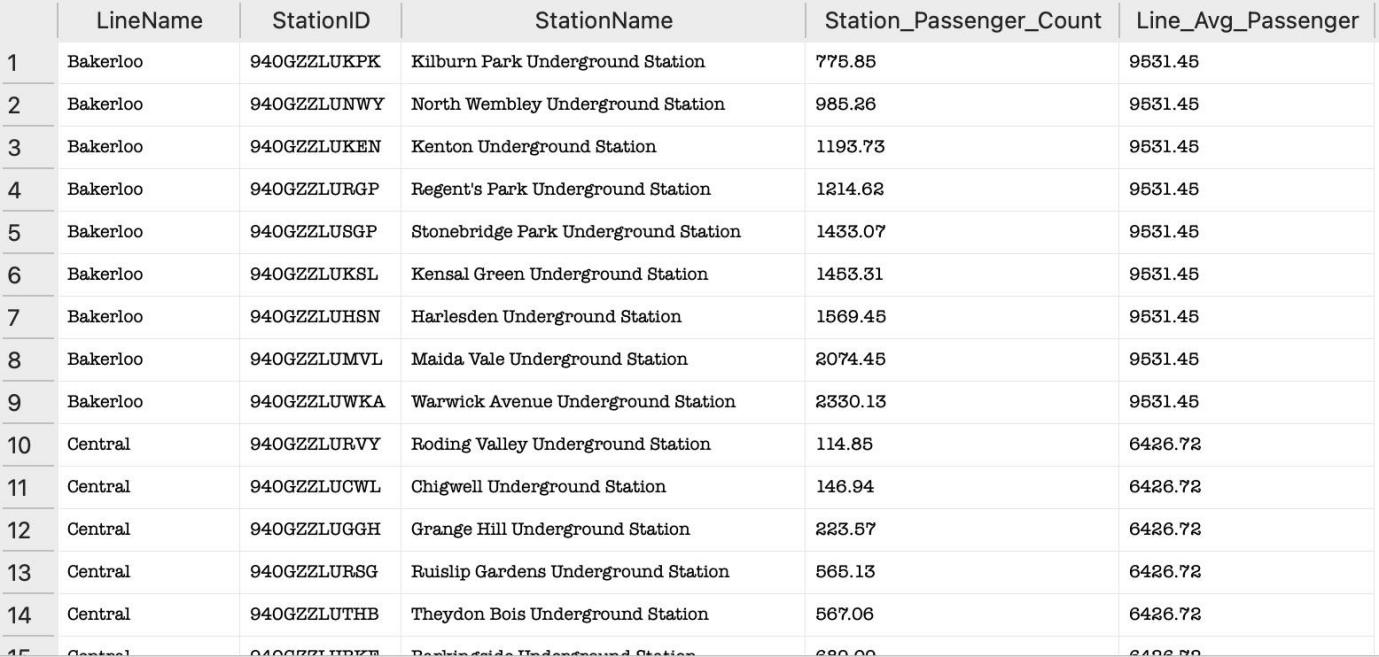
This analysis is particularly useful for identifying high-priority stations requiring operational improvements. By highlighting frequently delayed stations, transport authorities can allocate resources effectively, develop targeted solutions to reduce delays, and enhance overall passenger satisfaction.

**Operation explained:**The query first aggregates daily delay durations for each station by summing the delays grouped by station and date (DailyDelays CTE). It then computes key metrics for each station: the average daily delay (Avg\_Daily\_Delay) and the number of days delays occurred (Delay\_Occurrences) in the StationDelayStats CTE. The results are ranked using the number of delay occurrences, allowing for a comparative analysis of stations. The final output includes the station name, its average daily delay, the number of delay occurrences, and its rank based on frequency.

**Query 2: Identify underutilized stations on each lines**

***Use of query:*** This query analyzes passenger flow across London Tube lines to identify stations with low passenger volumes compared to their line's average. It specifically flags stations where the total passenger volume is less than 25% of the line-wide average passenger volume. This analysis helps identify underutilized stations that may require attention for improvement or optimization.

This query helps identify underutilized stations across the Tube network, allowing transport authorities to develop strategies to increase usage, such as improving accessibility, adjusting service frequency, or marketing campaigns, and optimize resource allocation to focus efforts where they are needed most.

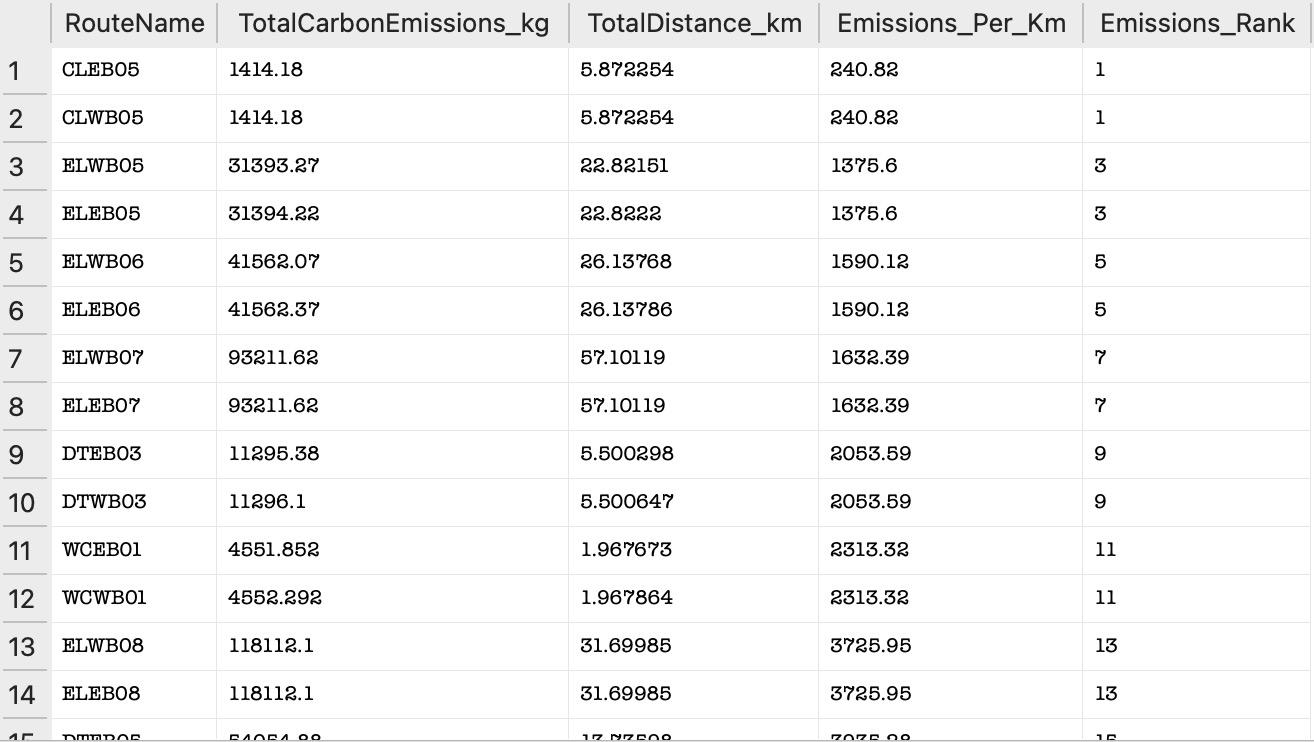


**Operation explained:**The query uses two key input parameters and conditions to analyze passenger flow. First, in the `LinePassengerStats` CTE, the query aggregates the total passenger volume for each station across all time intervals by calculating the sum of passenger counts (`Station\_Passenger\_Count`). This is grouped by the `LineID`, `LineName`, and `StationID`, associating each station's data with the respective Tube line. Additionally, the `StationName` is included for easier interpretation of the results. Second, in the `LineAverage` CTE, the query calculates the average passenger volume for all stations on each line (`Line\_Avg\_Passenger`) by taking the average of the station-level totals computed in `LinePassengerStats`. This ensures the query establishes a benchmark for comparing individual station performance. Finally, the main query filters stations where the total passenger volume is less than 25% of their line’s average, effectively highlighting significantly underutilized stations. These conditions collectively ensure that the analysis captures meaningful comparisons between station and line-level passenger patterns.

**Query 3: Rank routes with highest carbon emission per kilometers**

**Use of query:**This query evaluates the environmental efficiency of different Tube routes by calculating and ranking their carbon emissions per kilometer. Routes are ranked based on their emissions per kilometer, with lower emissions earning a better rank (e.g., Rank 1 is the most efficient).

This query is useful for assessing the environmental efficiency of Tube routes. By identifying routes with high emissions per kilometer, transport authorities can prioritize efforts to optimize or reduce their environmental impact. This could involve introducing cleaner technologies, adjusting route designs, or investing in alternative energy sources. Additionally, the ranking provides a clear, actionable comparison across routes, enabling data-driven decisions to improve the sustainability of the transportation network.

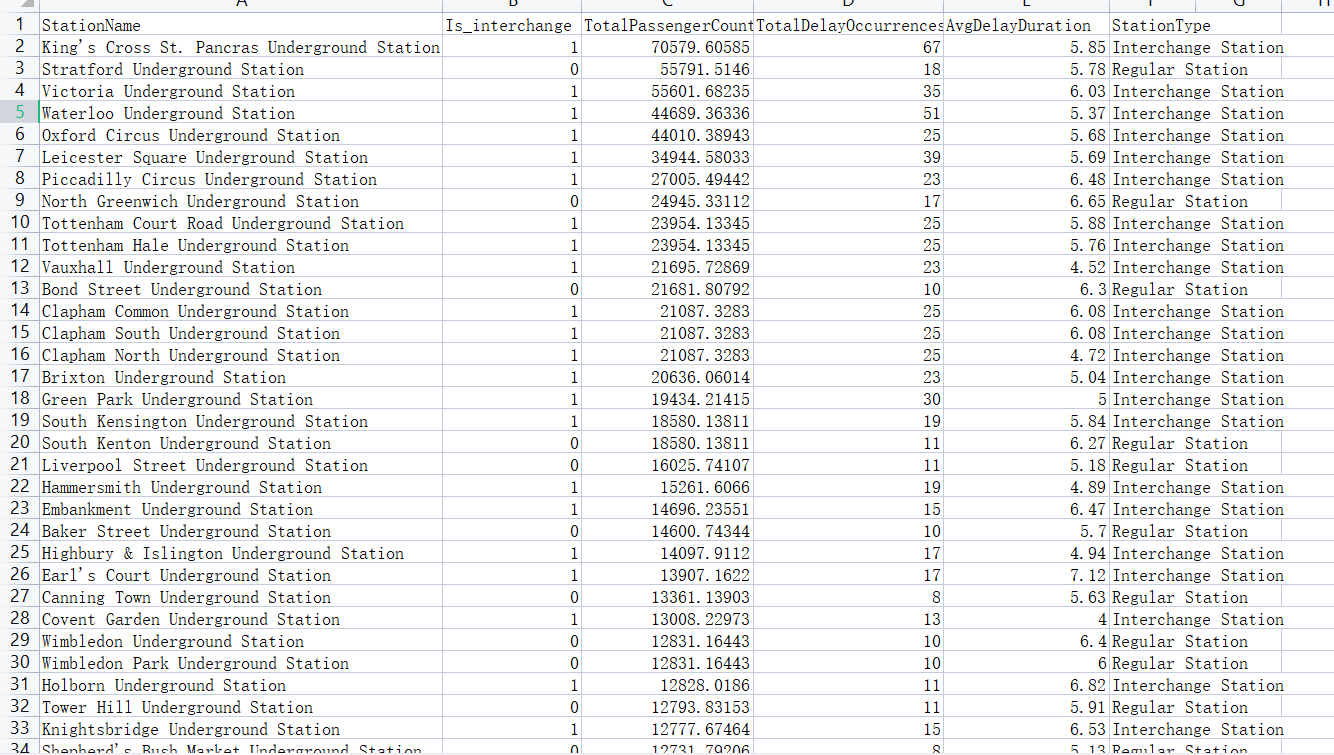


**Operation explained:** It joins the Route table with the Carbon\_Emission table on the EmissionID column to associate each route with its corresponding carbon emission data, including total emissions (TotalCarbonEmissions\_kg) and total distance covered (TotalDistance\_km). To calculate emissions per kilometer, the query divides TotalCarbonEmissions\_kg by TotalDistance\_km, ensuring that division by zero is avoided using the NULLIF function, which returns NULL if the distance is zero. The result is rounded to two decimal places for clarity. A window function, RANK(), is applied to rank routes based on their emissions per kilometer in ascending order, with the most efficient route (lowest emissions per kilometer) receiving Rank 1.

**Query 4: Correlate delays with station traffic volume and interchange status**

**Use of query:**The query is designed to analyze passenger traffic and delay occurrences at various stations. Its purpose is to provide insights into how delays correlate with passenger volume and whether station type (e.g., interchange vs. regular station) influences delay occurrences and durations.

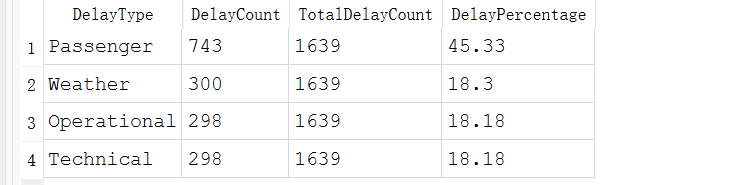
This information is valuable for transportation planners to improve scheduling efficiency, allocate resources, and identify bottlenecks at high-traffic stations. Specifically, the query resolves issues where overlapping delays might cause inflated passenger counts, ensuring the results are reliable for decision-making.



**Operation explained:**First, it aggregates passenger counts from the Passenger\_Counts table for each station and time interval, converting time intervals into minutes for easier matching with delay records. Next, it identifies unique delay records from the Delay table and matches them with the pre-aggregated passenger data. A key feature is the use of a grouping mechanism to ensure each time interval’s passenger count is only matched once with delay records, preventing duplication. Finally, the results are consolidated to calculate total passenger counts, unique delay occurrences, and average delay durations for each station, distinguishing between interchange and regular stations.

**Query 5: Analyzing Delay Types and Their Proportions**

**Use of query:** The main goal of this query is to identify the relative contribution of each delay type to the total number of delays. It provides a clear percentage breakdown of different delay categories.This analysis helps prioritize efforts to address delays by focusing on the most frequent delay types. It can guide operational improvements, resource allocation, and decision-making processes to reduce delays.

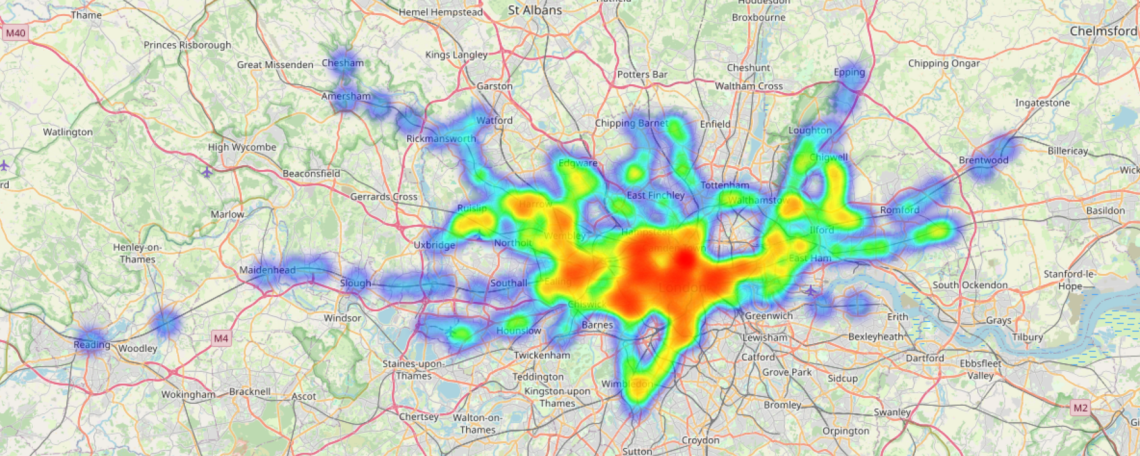


**Operation explained:**This query operates in three steps. First, it groups delays by DelayType and counts occurrences for each type. Then, it calculates the total delay count across all types. Finally, it combines these results using a CROSS JOIN and computes the percentage of each delay type as (DelayCount / TotalDelayCount) \* 100, sorting the results by percentage in descending order. This provides a clear breakdown of how each delay type contributes to the total.

**HeatMap:**

**Heat Map: Spatial Analysis of Carbon Emissions Across London Tube Stations**

**Use of Heat Map:** This usage calculates and visualizes carbon emissions at London Tube stations, producing a spatial heat map that highlights high and low emission areas. By combining passenger data with route-level emissions and geospatial coordinates, it provides insights into the environmental impact of the transportation network. The map aids transport authorities in identifying emission hotspots, optimizing resources, and developing strategies for sustainability and operational efficiency.

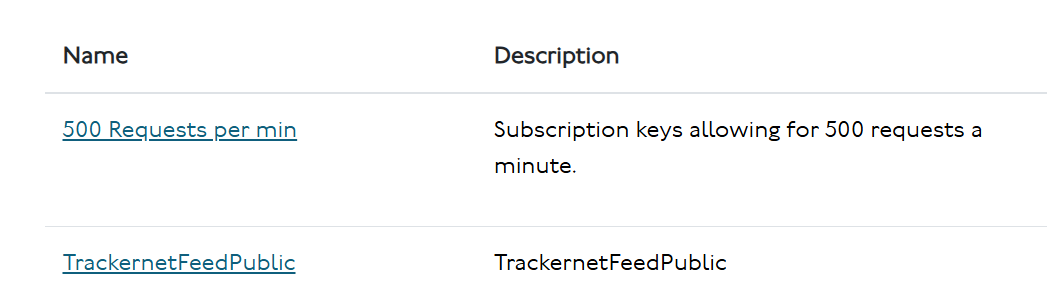


**Operation explained:**The query first aggregates station-level carbon emissions by calculating the proportional contribution of station-level passenger counts to the total route-level emissions to create a new view. The emissions are grouped by both station and route to ensure accurate allocation. The query joins the aggregated station emissions with geospatial coordinates (latitude and longitude) from the Station table, providing the necessary data for mapping. The results are exported to a CSV file for use in visualization. Using the exported data, a Python script processes the station emissions and creates a heat map using the folium library. Each station is plotted on the map with intensity proportional to its total carbon emissions. The resulting interactive map allows stakeholders to identify emission hotspots and analyze spatial patterns, supporting effective decision-making and sustainability planning. The code in this section is all written with jupyter notebook and saved as HTML file.

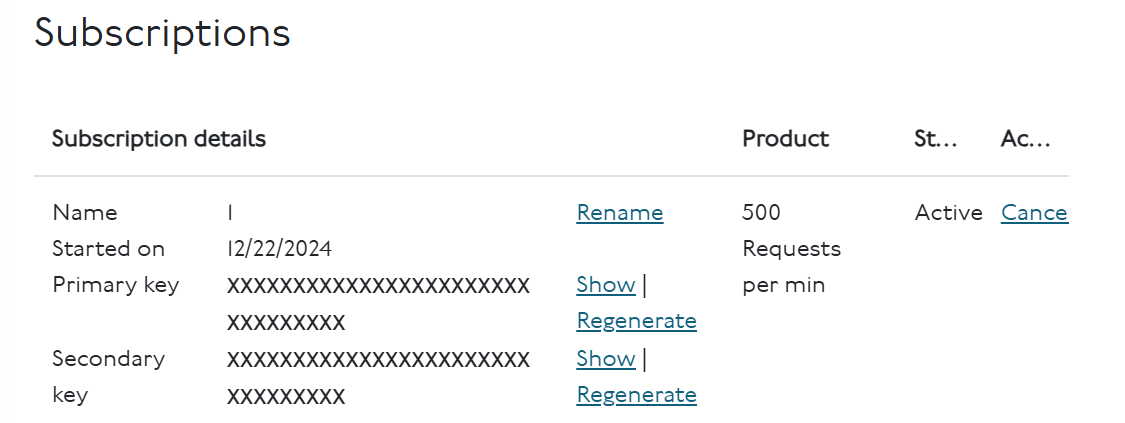
**Justification of the Database Tool**

**Tool 1: TfL API**

1. Create a Transport For London account on [Sign in - Transport for London - API](https://api-portal.tfl.gov.uk/signin)
2. Click Products on the website. Make a Subscription on 500 Requests per min

****

1. Click Profile and find the Name and Key in the Subscriptions section.



1. Browse through [Our Unified API - Transport for London](https://tfl.gov.uk/info-for/open-data-users/api-documentation) and utilise the API in the programming procedure

**Tool 2: folium library**

1. Folium is a Python-based map visualization library that utilizes Leaflet.js to provide interactive maps that can be used in Jupyter Notebooks, Python scripts, and Web applications. folium is suitable for data analysis, geographic information systems (GIS), spatial data visualization and other scenarios. We use folium to generate heat map in the project.
2. Install folium in the jupyter notebook and then import the library when using it. Browse through [Folium — Folium 0.19.4 documentation](https://python-visualization.github.io/folium/latest/) for the detailed usages.

**Tool 3: DB Browser**

1. Download DB Browser at [Downloads - DB Browser for SQLite](https://sqlitebrowser.org/dl/)
2. Click “Open Database” and select LondonTransportation.db file
3. Run SQL queries in the “Execute SQL” tab

