Tests for CommonShare Data Science internship Fall 2024

Realized by:
Mohamed Ben Gourar

PLAN:

[1]. Introduction

[2]. What is OpenAQ ?

[3]. The ETL Script Overview

a. Data Extraction
b. Data Cleaning
c. Data Transformation
d. Data Loading

[4]. Testing with Bad Data

[5]. SQL Database Integration

[6]. Conclusion

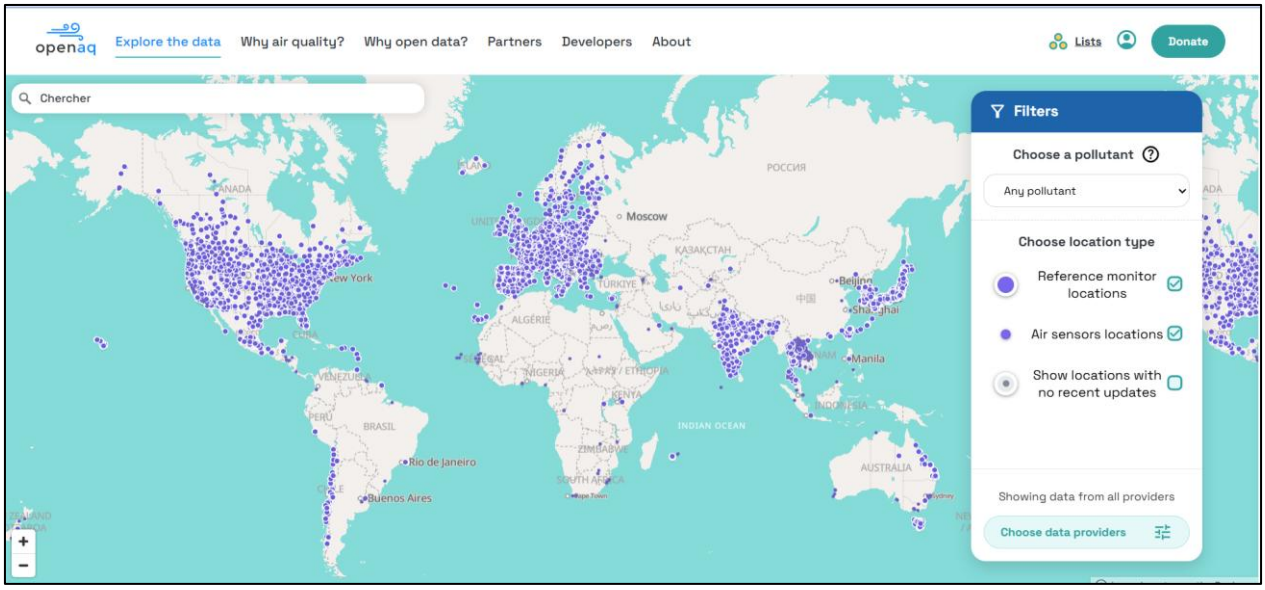
I. Introduction

In this project, I was tasked with building an ETL script (Extract, Transform, Load) pipeline that fetches environmental data from an external API, processes it, and loads it into a relational database. I chose to work with OpenAQ, a free and open-source API that provides air quality data from cities and monitoring stations all around the world. This project is for the purpose of demonstrating the important steps of data engineering concerning ETL processes especially data retrieved using API in order to do : data acquisition, data cleaning and transformation, error handling, and database integration.

II. What is OpenAQ?

OpenAQ (https://openaq.org) is a global open-source platform that aggregates and shares air quality data. It collects data from official government and research sources and provides access to metrics such as concentrations of particulate matter (PM2.5 and PM10), carbon monoxide (CO), nitrogen dioxide (NO2), sulfur dioxide (SO2), ozone (O3), and more.

In this project, I used the OpenAQ v3 API to fetch data about specific locations, including their environmental sensors and metadata.



Sustainable data of OpenAQ

Using Your API Key

Include your API key in the `X-API-Key` header of each request. For example:

```
curl --request GET \
--url "https://api.openaq.org/v3/locations/2178" \
--header "X-API-Key: YOUR-OPENAQ-API-KEY"
```

Replace `YOUR-OPENAQ-API-KEY` with your actual key. This key verifies your requests and ensures they are processed.

Retrieving API key and establishing connection

III. The ETL Script Overview

The Python script I built handles the full ETL (Extract, Transform, Load) process:

- Extract: Connect to the OpenAQ API using an API key to fetch JSON data for given location IDs that the user gives to specify the exact location the data will pass the ETL process.
- Transform: Clean and reformat the data by handling missing values, normalizing fields, and creating structured dictionaries.
- Load: Insert the cleaned and transformed data into an SQLite relational database.

The script follows this order to make the ETL process successful and have the data ready to be used by the client:

1. Data Extraction:

The script sends HTTP GET requests to the OpenAQ API using the provided API key. I specified one or more location IDs (e.g. 2178 for "Del Norte") to get information about sensors, parameters, and air quality metrics at those locations.

The API request uses the following headers and endpoint:

- URL: https://api.openaq.org/v3/locations/{id}
- Header: X-API-Key: <my_api_key>

The difficulties that are encountered and are considered important are :

- 404 Not Found errors (invalid IDs),
- rate limiting (with retries and backoff),
- connection errors,
- and invalid/missing data in the response.

These errors are essential to take into consideration when launching your application. After taking into consideration these elements you cover all the possibilities that may the application face when working to different data and to various possibilities and ways that the requests and rates of the requests are possible to be. In addition, you also handle missing data preventing the script to crash.

2. Data Cleaning:

After retrieving the raw data, the script performs several cleaning tasks:

- If the provider, country, or coordinates are missing, default values like "Unknown" or "XX" are used.
- Nested fields are handled safely using Python's .get() method to avoid crashes.
- Dates and location names are standardized.
- Invalid or missing sensor entries are skipped or logged for review.

3. Data Transformation

The script transforms the data into Python dictionaries with consistent field names and types.

- For example, a location with missing provider info will still be inserted with "Unknown" as the provider.
- Each sensor and parameter is extracted into flat structures to make it easier to load into SQL tables.
- The script logs warnings about any bad data, making the data quality issues visible.

4. Data Loading:

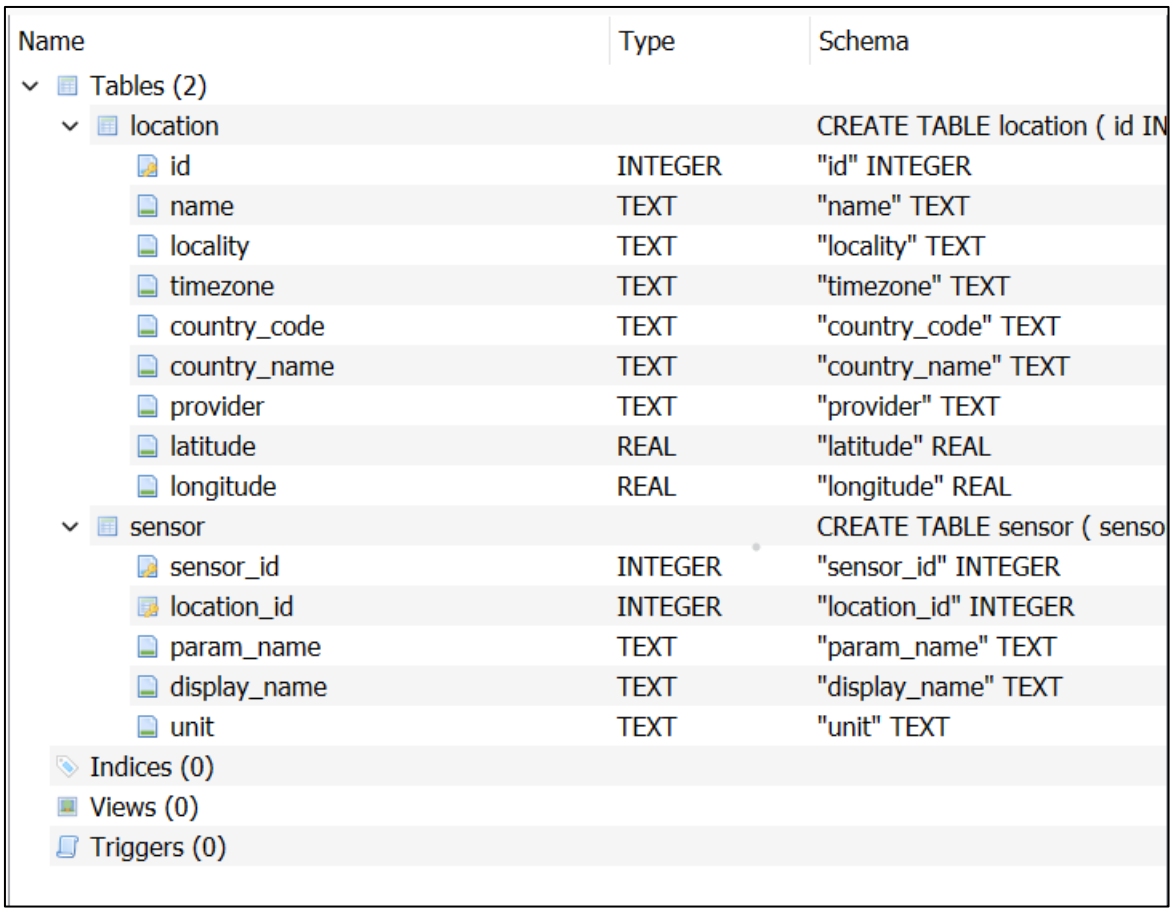
Once cleaned, the data is inserted into an SQLite database (openaq_data.db). I used SQLite for simplicity since it's easy to use locally and requires no setup.

The database contains relational tables like:

- locations
- sensors

Each location is linked to one or more sensors. SQL INSERT queries are used with parameters to avoid injection issues. Tables are created only once using CREATE TABLE IF NOT EXISTS.

In order to visualize the database created and understand the data retrieved, I used **DB Browse for SQLite :**



Name	Type	Schema
Tables (2)		
location		CREATE TABLE location (id IN
id	INTEGER	"id" INTEGER
name	TEXT	"name" TEXT
locality	TEXT	"locality" TEXT
timezone	TEXT	"timezone" TEXT
country_code	TEXT	"country_code" TEXT
country_name	TEXT	"country_name" TEXT
provider	TEXT	"provider" TEXT
latitude	REAL	"latitude" REAL
longitude	REAL	"longitude" REAL
sensor		CREATE TABLE sensor (senso
sensor_id	INTEGER	"sensor_id" INTEGER
location_id	INTEGER	"location_id" INTEGER
param_name	TEXT	"param_name" TEXT
display_name	TEXT	"display_name" TEXT
unit	TEXT	"unit" TEXT
Indices (0)		
Views (0)		
Triggers (0)		

SQLite database for relation tables (locations, sensors)

IV. Testing with Bad Data

In order to make sure the script can handle real-world dirty data, I manually created a file named **test_data.json** that simulates a bad response from the API. This file contains:

- Missing provider info
- Null coordinates
- Empty sensor fields

The script includes a **test_mode** flag. When enabled, the script loads data from **test_data.json** instead of fetching it from the API. This allowed me to observe how the pipeline handles missing or malformed data.

The outcome showed that:

- No exceptions were raised.
- All missing values were replaced with defaults.
- Logging messages were generated for every issue (e.g., “Missing coordinates for location 2178”).

The result can be summarized into two parts:

- If the script doesn’t manage these bad data, it CRASHES !

```
PS C:\Users\User\Desktop\vs Code\technichal_ass> python air_quality.py
Current working directory: C:\Users\User\Desktop\vs Code\technichal_ass
2025-06-01 18:17:39,476 - INFO - Processing location ID: 2178
Traceback (most recent call last):
  File "C:\Users\User\Desktop\vs Code\technichal_ass\air_quality.py", line 156, in <module>
    run_etl(location_ids, test_mode=True)
  File "C:\Users\User\Desktop\vs Code\technichal_ass\air_quality.py", line 145, in run_etl
    location_data = clean_location_data(raw)
                    ^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\User\Desktop\vs Code\technichal_ass\air_quality.py", line 51, in clean_location_data
    "provider": data.get("provider", {}).get("name", "Unknown"),
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'get'
```

Script crashed due to unknown data

After I modified the code to be able to handle bad data it simply can get the raw data without crashing:

```
Current working directory: C:\Users\User\Desktop\vs Code\technichal_ass
2025-06-01 18:25:21,169 - INFO - Processing location ID: 2178
2025-06-01 18:25:21,185 - INFO - Location 2178 processed.
2025-06-01 18:25:21,185 - INFO - ETL completed for all locations.
```

ETL Process on the location id = 2178

V. SQL Database Integration

The final step of the pipeline is writing the cleaned data to an SQLite database.

I used the built-in sqlite3 module in Python to:

- Create and connect to a database file (openaq_data.db).
- Define schemas using SQL CREATE TABLE.
- Insert data using parameterized INSERT statements.

VI. Conclusion

This project helped me apply the full lifecycle of a data engineering task:

- I successfully built an ETL pipeline from scratch using a real-world environmental API.
- I handled error-prone and incomplete data using logging and data validation.
- I designed a relational database schema and integrated it with the ETL process.
- I tested both good and bad data inputs to ensure reliability and robustness.