Urban Mobility and Weather Impact ETL Pipeline Report

1. Overview

This project demonstrates an end-to-end ETL (Extract, Transform, Load) pipeline designed to integrate multiple data sources in a unified scenario. The goal is to analyze how weather conditions and traffic impact urban mobility (e.g., ride-share and taxi operations) in a major city like New York City (NYC). The project combines historical taxi trip data with real-time and static data (weather, traffic congestion, and geographic metadata) to produce enriched datasets for further analytics and reporting.

2. Project Objectives

- **Integrate Multiple Data Sources:**
  Combine CSV data, weather API responses, Google Sheets metadata, MongoDB static metadata, and a real-time congestion REST API.

- **Data Processing and Transformation:**
  Clean the raw data, standardize units (e.g., convert temperature to Celsius), normalize timestamps to UTC in ISO 8601 format, and engineer features (e.g., compute a “weather impact score”).

- **Automated Loading:**
  Store the final enriched data into a MongoDB collection for easy querying and analytics.

- **Automation and Scheduling:**
  Automate the ETL process to run daily using Python’s scheduling capabilities.

- **CI/CD Integration:**
  Implement a GitHub Actions pipeline for continuous integration, testing, linting, and deployment.

3. Data Sources

3.1 CSV File

- **Source:** Historical NYC taxi trip data (e.g., `green_tripdata_2023-01.csv`)

- **Usage:** Provides baseline trip data such as pickup times, location IDs, fare amounts, and trip durations.

3.2 JSON/API (Weather Data)

- **Source:** OpenWeatherMap API
- **Usage:** Retrieves real-time or historical weather data (temperature, humidity, wind speed) for the area corresponding to trip pickup locations.

3.3 Google Sheets

- **Source:** Zone metadata stored in a Google Sheet named “Zone Metadata”
- **Usage:** Contains zone IDs, names, and borough mappings that are used to enrich the taxi trip data.

3.4 MongoDB

- **Source:** Two collections:
    - **locations:** Static metadata (geospatial coordinates, land use, traffic patterns) for each zone.
    - **cleaned_trip_data:** Destination collection where the enriched, cleaned, and transformed data is loaded.

3.5 REST API

- **Source:** A simulated REST API providing real-time congestion levels based on zone IDs.
- **Usage:** Offers current traffic congestion data to further enhance the analysis.

4. ETL Pipeline Implementation

4.1 Extraction

- **CSV Extraction:**
  Uses Pandas to load and read the CSV file containing taxi trip data.
- **Weather Data Extraction:**
  Calls the OpenWeatherMap API for each unique zone in the dataset, applying a slight delay (`time.sleep`) to avoid rate limits.

- **Google Sheets Extraction:**
  Utilizes `gspread` with a service account (via a `gspread-creds.json` file) to pull metadata from the “Zone Metadata” sheet.

- **MongoDB Static Data Extraction:**
  Uses `pymongo` to connect to MongoDB and retrieve location metadata from the `locations` collection.

- **REST API Extraction:**
  Retrieves real-time congestion data by calling a REST endpoint with a zone identifier.

4.2 Transformation

- **Data Cleaning:**
  Removes rows with missing or erroneous values (e.g., negative fare amounts or zero/negative trip durations).

- **Unit Conversion:**
  Standardizes weather data (e.g., converting temperature values to Celsius when needed).

- **Timestamp Normalization:**
  Converts pickup timestamps to UTC and formats them in ISO 8601.

- **Data Enrichment:**
  Merges the taxi data with Google Sheets zone metadata and MongoDB location metadata, while also attaching weather data and congestion levels.

4.3 Loading

- **MongoDB Loading:**
  Converts the final Pandas DataFrame into a dictionary format and loads it into the MongoDB collection (`cleaned_trip_data`). Existing data is replaced to ensure only the latest batch is available.

5. Automation and Scheduling

5.1 Python Scheduler

Instead of relying on system-level cron jobs, the ETL pipeline is automated using Python’s `schedule` library. A separate script (`scheduler.py`) schedules the `run_etl()` function to run daily at a specified time (e.g., 1:00 AM).

6. CI/CD Pipeline with GitHub Actions

A GitHub Actions workflow (`.github/workflows/ci_cd.yml`) is set up to run automated tests, lint the code, and execute the ETL script upon every commit or pull request to the `main` branch. The pipeline also optionally runs on a schedule.

Key Components:

- **Checkout & Setup:**
  The workflow checks out the repository and sets up the required Python version.

- **Dependencies:**
  Installs project dependencies from `requirements.txt`.

- **Linting and Testing:**
  Runs `flake8` for code quality and `pytest` for unit tests.

- **ETL Execution:**
  Executes the main ETL script while securely injecting necessary environment variables (e.g., API keys).