**ETL Pipeline Report: Epidemiological Data Processing**

1**. Objective**

This project aims to build an ETL (Extract, Transform, Load) pipeline that processes epidemiological data from multiple sources, applies necessary cleaning and transformation, and loads the data into a PostgreSQL database for analysis. The pipeline integrates data from Google Sheets, CSV files, JSON, API endpoints, and MongoDB, and processes them into a unified dataset.

**2. Data Sources**

The following sources of data are integrated into the pipeline:

Google Sheets: Data regarding epidemiological trends, including COVID-19 confirmed cases, deaths, recoveries, and tests.

CSV Files: Local datasets in CSV format containing daily updates on new confirmed cases, deaths, etc.

JSON Files: Data obtained from a JSON format file (e.g., local data or historical data).

API: Data fetched through an external API endpoint (e.g., COVID-19 viral sequences data).

MongoDB: Data stored in MongoDB, extracted and integrated for use.

**3. Pipeline Overview**

The ETL process consists of three main stages:

**Extract**

Google Sheets: Data is fetched using gspread, which provides an easy interface for interacting with Google Sheets using the Google Sheets API.

CSV File: Using pandas, the CSV file is read into a DataFrame.

JSON File: The JSON file is read and converted into a DataFrame for further processing.

API Data: Data is fetched from an external API endpoint. The response is processed into a structured DataFrame.

MongoDB: Data from MongoDB is extracted by connecting to the database and querying for records.

**Transform**

Standardization: The column names are standardized to lowercase and spaces are replaced with underscores.

Data Cleaning: Empty values are replaced with NaN, and forward-filling is used to fill missing data. Duplicate records are removed.

Date Formatting: Dates are formatted into a standard format, and invalid dates are coerced to NaT.

Data Validation: Numeric columns are validated to ensure no negative values are present.

Aggregation: Data is aggregated by date and location\_key, summing up new cases and taking the maximum for cumulative values.

**Load**

PostgreSQL: The transformed data is loaded into a PostgreSQL database using SQLAlchemy, which interacts with the Supabase-managed PostgreSQL database.

**4. Directory Structure**

The project is organized into the following directory structure for modularity and maintainability:

bash

Copy

Edit

ETL\_Pipeline\_Ganesha\_Maheshwari\_<DS-30/2024-2025>/

├── etl\_pipeline.py              # Main ETL script

├── config/

│   └── db\_config.json           # Configuration file for database and API keys

├── data/

│   ├── sample\_data.csv          # Example CSV dataset

│   ├── sample\_weather.json      # Example JSON dataset

│   └── google\_sheet\_sample.csv  # Exported Google Sheets example

├── scheduler.py                 # Scheduler for running the ETL pipeline periodically

├── requirements.txt             # Dependencies for the pipeline

├── README.md                    # Project overview and setup instructions

├── output/

│   └── final\_cleaned\_data.csv   # Output cleaned data

├── load\_to\_db.py                # Script for loading data into the database

├── .github/

│   └── workflows/

│       └── ci\_cd.yml            # GitHub Actions CI/CD configuration

└── report.pdf                   # Project report

**5. Code Breakdown**

Here is a breakdown of key components and their functionalities:

ETL Pipeline Script (etl\_pipeline.py)

Data Extraction: Functions are defined for extracting data from Google Sheets, CSV, JSON, API, and MongoDB.

Data Transformation: The cleaning, formatting, and aggregation functions are modular and can be applied to any dataset.

Final Processing: The cleaned data from all sources is concatenated into a final DataFrame, which is then saved as a CSV file and loaded into a PostgreSQL database.

Database Configuration (config/db\_config.json)

This JSON configuration file stores all necessary credentials and paths for API, Google Sheets, and MongoDB connections. It allows easy management of these details without hardcoding them in the script.

Loading to PostgreSQL (load\_to\_db.py)

After transforming the data, it is loaded into a PostgreSQL database using SQLAlchemy. The database URL is encoded, and the data is uploaded into a specified table, replacing any existing records.

Scheduler (scheduler.py)

This script uses the schedule library to schedule the ETL pipeline to run at regular intervals (e.g., daily at midnight). This allows for automated execution of the pipeline without manual intervention.

CI/CD with GitHub Actions (.github/workflows/ci\_cd.yml)

GitHub Actions is used for Continuous Integration and Continuous Deployment (CI/CD). The configuration file ensures that the pipeline runs whenever changes are pushed to the repository, providing automated testing and deployment.

**6. Requirements**

To run the pipeline, the following Python packages are required:

pymongo for interacting with MongoDB

gspread for accessing Google Sheets

pandas for data manipulation

psycopg2-binary for PostgreSQL database interaction

requests for making HTTP requests to APIs

sqlalchemy for working with the PostgreSQL database

schedule for scheduling periodic executions

The dependencies are listed in the requirements.txt file.

**7. Instructions for Running the ETL Pipeline**

Install dependencies:

nginx

Copy

Edit

pip install -r requirements.txt

Configure credentials: Update the config/db\_config.json file with your database credentials, API URLs, and file paths.

Run the ETL pipeline:

nginx

Copy

Edit

python etl\_pipeline.py

Automate the process: Use the scheduler to run the pipeline periodically:

nginx

Copy

Edit

python scheduler.py

**8. CI/CD Setup**

The project includes a GitHub Actions configuration to automatically run the ETL pipeline on every push to the main branch. The CI/CD pipeline ensures that the latest changes to the project are automatically processed.

**9. Conclusion**

This ETL pipeline processes multiple data sources, applies necessary transformations, and loads the cleaned data into a PostgreSQL database for further analysis. The project is designed to be modular, scalable, and automated, making it suitable for periodic data processing tasks.

**10. Future Enhancements**

Error Handling: More robust error handling mechanisms to deal with failures during data extraction, transformation, or loading.

Scheduling Improvements: A more sophisticated scheduling system with retry logic and logging.

Data Quality Checks: Implementing additional checks to ensure data consistency across sources.