SuperCourier ETL Pipeline - LaPoste Data Engineers Project Overview

Made by Angel Gaspard-Fauvelle, LaPoste Data Engineers Teacher Assistant

This project contains a robust **ETL** (Extract, Transform, Load) pipeline designed to process and analyze delivery data for the fictional company "**SuperCourier**." The primary goal is to generate a **clean**, **structured**, and **enriched** dataset that can be used by a data science team **to build a predictive model for delivery delays**.

The pipeline simulates a real-world data engineering scenario by:

1. **Generating** synthetic source data (a logistics database and weather logs).
2. **Extracting** data from a SQLite database.
3. **Transforming** the data by cleaning it, enriching it with time-based features and weather conditions, and applying complex business logic to determine delivery status.
4. **Loading** the final, analysis-ready dataset into multiple formats.

The script is interactive, scalable, and includes robust error handling and logging.

Project Features

This **ETL pipeline** successfully implements all the core requirements outlined in the project presentation and adds several key enhancements for better usability, performance, and maintainability.

Core Requirements Checklist

Requirement	Status	Implementation Details
Create SQLite Database	✅	`data_generators.generate_sqlite_database()` creates a logistics DB.
Generate Weather Data	✅	`data_generators.generate_weather_data()` creates a JSON file with hourly weather.
Join & Transform Data	✅	`etl_pipeline.transform_data()` **merges sources** using Pandas.
Calculate Delivery Metrics	✅	`domain.calculate_delivery_status()` applies a complex formula with **multiple coefficients**.
Handle Missing Values	✅	**Weather data** enrichment gracefully handles missing lookups.
Export to CSV	✅	**The script supports CSV and four other formats.**
Implement Logging	✅	**A robust logging setup (**`config.py`**)** records events to both the console and a file.
Modular Project Structure	✅	**The code is split into logical modules (**`config`, `domain`, `etl_pipeline`, etc.).

Enhancements and Upgrades

Beyond the base requirements, the following features were added to improve the project:

- **Interactive Configuration :** The user is prompted to set the size of the generated dataset (number of deliveries and weather days).

- **Advanced File Management** (Clean Architecture): Before running, the script offers to **archive**, **replace**, or **completely delete** previous output files, preventing data loss and allowing for clean runs.
- **Multiple Output Formats :** The final dataset can be saved in five different formats, catering to various needs :
    - **CSV :** For universal compatibility.
    - **Parquet :** For efficient, compressed storage ideal for big data analytics.
    - **JSON :** For use in web applications and APIs.
    - **SQLite DB :** For easy querying and integration with other database tools.
    - **Excel (.xlsx) :** For business users and manual analysis.
- **Robust File Handling:** Automatically archives or deletes previous runs based on user choice.
- **Performance-Optimized Code** : Transformations use vectorized **Pandas** and **NumPy** operations for maximum efficiency, and the Excel export uses a memory-optimized engine.
- **Comprehensive Testing** (Continuous Integration & Unit Testing + Benchmarks Performances): The project includes a `pytest` suite (`test_pipeline.py`) for unit and integration testing, and a performance benchmarking script (`benchmark_tests.py`).
- **Containerization** : The entire environment is reproducible and portable thanks to **Docker** and **Conda** configuration files.
- **Distributable:** Can be compiled into a single executable file using PyInstaller.

Benchmark Performances

The pipeline has been benchmarked to showcase its performance and scalability. The following table shows the execution times for generating datasets of various sizes.

- **No xlsx :** Exports to CSV, Parquet, JSON, and SQLite.
- **All :** Exports to all formats, including the more time-intensive Excel (.xlsx) file.

Deliveries	Weather Days	Output Format	Execution Time (in seconds)
1,000	**90**	**No xlsx**	**0.15**
1,000	**90**	**All**	**0.33**
5,000	**450**	**No xlsx**	**0.40**

Deliveries	Weather Days	Output Format	Execution Time (in seconds)
5,000	450	All	0.91
10,000	900	No xlsx	0.67
10,000	900	All	1.62
100,000	9,000	No xlsx	5.90
100,000	9,000	All	15.59
1,000,000	90,000	No xlsx	60.52
1,000,000	90,000	All	154.40

Getting Started

There are three ways to run the ETL pipeline, from the simplest method for non-developers to a fully containerized environment.

1. Using the Pre-compiled Program (Easiest)

A pre-compiled executable (`SuperCourierETL.exe`) is available for users who do not have Python or Docker installed.

1. **Download the `SuperCourierETL.exe` file.**
2. **Double click on it to open the program.**

or :

1. **Open a terminal or command prompt and navigate to the folder containing the file.**
2. **Run the program:**

   ```
   ./SuperCourierETL
   ```

3. **Follow the interactive prompts in the terminal to configure and run the pipeline.**

2. Using Conda

This method is ideal for developers who want to run the script in an isolated and reproducible Python environment.

1. **Clone the repository :**

```
git clone <your-repository-url>
cd <repository-folder>
```

2. **Create the Conda environment** from the `environment.yml` file. This will install all necessary dependencies.

```
conda env create -f environment.yml
```

3. **Activate the environment :**

```
conda activate pysupercourier
```

4. **Run the main script :**

```
python main.py
```

5. **Follow the interactive prompts to run the pipeline.**

3. Using Docker

This method provides the highest level of isolation and portability by running the application inside a container. It's the recommended approach for ensuring the script runs identically across any machine.

1. **Ensure Docker Desktop is installed and running.**
2. **Clone the repository :**

```
git clone <your-repository-url>
cd <repository-folder>
```

3. **Build and run the container** using Docker Compose. This command handles everything from building the image to running the script interactively.

   ```
   docker-compose build
   docker-compose run --rm app
   ```

4. **The script will start inside the container. Follow the interactive prompts in your terminal.**
5. **Output files** generated by the script will be available in the `output_files` directory on your local machine, as it is mapped to the container via a volume.

Project Structure

The project is organized into a modular structure to separate concerns and improve maintainability.

```
.
│   .dockerignore
│   .gitignore
│   docker-compose.yml
│   Dockerfile
│   environment.yml
│   main.py
│   pytest.ini
│   README.md
│   SuperCourierETL.spec
│
├───docs
│
├───monolithic_scripts
│   │   de-code-snippet.py
│   │   monolithic_etl.py
│
├───src
│   │   config.py
│   │   data_generators.py
│   │   domain.py
│   │   etl_pipeline.py
│   │   file_manager.py
│   │   __init__.py
│
```

```
    └───tests
        │   benchmark_tests.ps1
        │   benchmark_tests.py
        │   test_pipeline.py
        │   __init__.py
```

What can we do to make the project suitable for a company ?

We can :

- modify config.py to take on a `.env` file, storing credentials
- replace data_generators.py file to fetch real datas from raw databases
- delete first user input asking for a number of deliveries and weather days to generate.

Then we can call it a day !

You can learn even more about the technical part by reading my #docs/Technical_Report.pdf and #docs/Technology_Watch.pdf !