Bindu Ganta https://github.com/bindureddy11/tceq-etl/tree/main

TCEQ Scraping ETL pipeline

Objective:

The aim of this project was to build a local ETL pipeline that collects and structures data about proposed regulations from a U.S. state website. I chose the Texas Commission on Environmental Quality (TCEQ) because it offers open access to regulation details, includes PDFs for each proposal, and has a practical, real-world web layout.

Approach:

I modularized the ETL pipeline into three components:

Extract:

The pipeline starts by visiting the TCEQ Proposed Rules page. It parses the main HTML table that lists proposed regulations. For each row (i.e., proposed rule), it extracts key metadata:

- Title and description (from the same table cell using span and '<br>' tags).
- Identifier.
- Proposal and comment dates (from '<th>' and '<li>' tags).
- Links to supporting PDFs (e.g., chapter revisions).

Transform:

For each rule, the pipeline downloads the associated chapter PDFs. It uses **pdfplumber** to extract full text content from the PDFs. Extracted text is added back into the rule's metadata as full text. All required fields (title, identifier, proposed date, full text) are validated. Warnings are logged for missing or unparseable documents.

Load:

The final list of enriched rules is saved as a structured **JSON file** in the /output directory.

The project is written in **Python**, organized using modular files (main.py, etl/, and a config file for flexibility). **Pytest-based tests** are included for core components like extraction, PDF parsing and JSON output

The pipeline is orchestrated by main.py and includes logging, error handling, and lightweight pytest-based tests, with configurable options such as PDF page limits, timeouts, and file paths in a dedicated config file to ensure flexibility, maintainability, and traceability during debugging.

Challenges Faced:

- Date-related fields were inside ‘<th>’ tags, while other fields like title and identifier were under ‘<td>’, so I handled both cases to extract data accurately.
- The title and its description were combined in a single column. I split them properly using the line break ‘<br>’ tag.
- All document links were grouped together, so I filtered out only the chapter links (starting with "Ch.") for full-text extraction.
- Not every rule had the same number of PDFs. I added error handling and logging to skip broken links without failing the pipeline.
- Reading multiple PDFs sequentially was slow. This can be improved later by adding parallel processing.
- I validated that key fields like title, ID, proposed date, and full text were present before saving each rule.

Future work

- Add new scrapers for other states using the existing ETL framework.
- Use Tesseract or AWS Textract to extract text from image-based PDFs.
- Implement parallel processing (e.g., with ThreadPoolExecutor) to improve performance when handling multiple PDFs concurrently.
- Schedule the ETL pipeline using a Cron/Airflow job to enable automated daily scraping and updates.
- Track last-scraped timestamps to detect changes and avoid redundant downloads.