Python–SQL ETL Project: Data Cleaning, Validation, and KPI Reporting

1. Project Overview and Summary

This project showcases a complete **end-to-end ETL (Extract–Transform–Load) pipeline** built using **Python (pandas)** and **SQLite/SQL**. The goal is to simulate a real-world analytics workflow where raw data is ingested, cleaned, validated, structured into a relational database, and queried to generate business KPIs.

The dataset used is a **Netflix synthetic viewing dataset** designed to represent typical streaming activity. It contains three core entities:

- **Users**: customer profile-level attributes (e.g., signup date, country, plan type)
- **Titles**: content catalog attributes (e.g., title name, genre, content type, maturity rating, release year)
- **Views**: viewing sessions (e.g., view start/end timestamps, watch minutes, completion flag)

The pipeline focuses on two key real-world data engineering goals:

1. **Data quality and standardization** (Python): cleaning raw CSV files, normalizing inconsistent values (like genre/rating formats), enforcing valid data types, and validating session logic (timestamps and watch minutes).
2. **Analytics-ready storage and reporting** (SQL): loading cleaned datasets into a relational SQLite database and running SQL KPI queries (joins, aggregations, rankings) to produce insights like top content, user engagement, and country-level watch patterns.

Key Outputs (Deliverables)

- Cleaned datasets saved as CSVs in data/cleaned/
- Invalid / rejected records stored in data/rejects/ (with reasons)
- SQLite database created (netflix.db) containing relational tables: users, titles, views
- KPI query results generated using sql/02_kpis.sql
- Execution proof screenshots stored in docs/results_screenshots/ (ETL run, DB load, KPI query run)

2.Repository Structure

Folder / File	Purpose
src/etl.py	Reads raw CSVs, cleans/standardises fields, validates viewing log rules, and outputs cleaned + rejects CSVs.
src/load_sqlite.py	Creates/opens netflix.db and loads cleaned tables into SQLite.

src/generate_data.py	*(Optional)* Generates synthetic raw data files for testing/demo, if included.
src/config.py	Central configuration for file paths and constants (RAW_DIR, CLEAN_DIR, REJECT_DIR).
sql/01_create_tables.sql	SQLite schema definitions for users, titles, and views.
sql/02_kpis.sql	KPI queries (top titles, engagement by plan, country-level watch patterns, etc.).
data/raw/	Raw input CSVs (often excluded in real projects; included here as sample inputs).
data/cleaned/	Cleaned output CSVs generated by the ETL pipeline.
data/rejects/	Reject records with clear reject_reason values for data quality auditing.
docs/results_screenshots/	Evidence screenshots showing successful runs (ETL run, DB load, KPI output).

Python (with pandas) is used for transformations and checks that are faster and clearer in code than SQL**,**

- safely parsing timestamps,
- standardising inconsistent text values (genre, ratings, country),
- coercing data types (e.g., numeric watch minutes),
- applying validation rules across multiple columns,
- and generating a **rejects file** with explicit reasons for auditability.

Cleaning and validation rules implemented

Users

- Parse signup_date into a consistent date format.
- Replace missing country and plan_type with "UNKNOWN".

Titles

- Standardise genre and maturity_rating using mapping rules.
- Convert release_year to an integer (invalid values handled safely).

Views

- Parse view_start and view_end as timestamps.
- Validate that watch_minutes is numeric and non-negative.
- Enforce session logic (e.g., end time must be after start time).
- Write valid rows to cleaned outputs and invalid rows to rejects with reject_reason.

Data quality handling (Rejects)

Invalid view rows are written to data/rejects/ with one or more reasons, such as:

- **bad_timestamp** – view_start or view_end cannot be parsed
- **end_before_start** – view_end occurs earlier than view_start
- **bad_watch_minutes** – missing, non-numeric, or invalid watch time

2. Relational Model and SQL (SQLite)

Schema Design

A relational database is created using **SQLite**. The schema is defined in:

- sql/01_create_tables.sql

It contains three core tables:

- **users** (Primary Key: user_id)
- **titles** (Primary Key: title_id)
- **views** (Primary Key: view_id)

The views table links back to users and titles using the user_id and title_id fields (logical foreign-key relationships). This structure enables join-based analysis such as watch minutes by country, plan type, or title.

KPI Queries

All analytics queries are stored in:

- sql/02_kpis.sql

This allows anyone to re-run KPI reporting **without modifying Python code**. Typical KPI outputs include:

- total watch minutes by title,
- engagement by subscription plan type,
- country-level viewing patterns and averages.

Running the Pipeline

Run the following commands from the **project root folder** (same level as src/, sql/, data/):

Step 1 — Create and activate a virtual environment

```
python3 -m venv .venv
source .venv/bin/activate
```

Step 2 — Install dependencies

```
pip install -r requirements.txt
```

Step 3 — Run the ETL process (creates cleaned + rejects files)

```
python3 src/etl.py
```

Outputs created:

- data/cleaned/ → cleaned CSV files
- data/rejects/ → rejected rows with reasons (if any)

Step 4 — Load cleaned data into SQLite

```
python3 src/load_sqlite.py
```

Output created:

- netflix.db (SQLite database)

Step 5 — Run KPI SQL queries

```
sqlite3 netflix.db < sql/02_kpis.sql
```

This prints KPI results to the terminal.

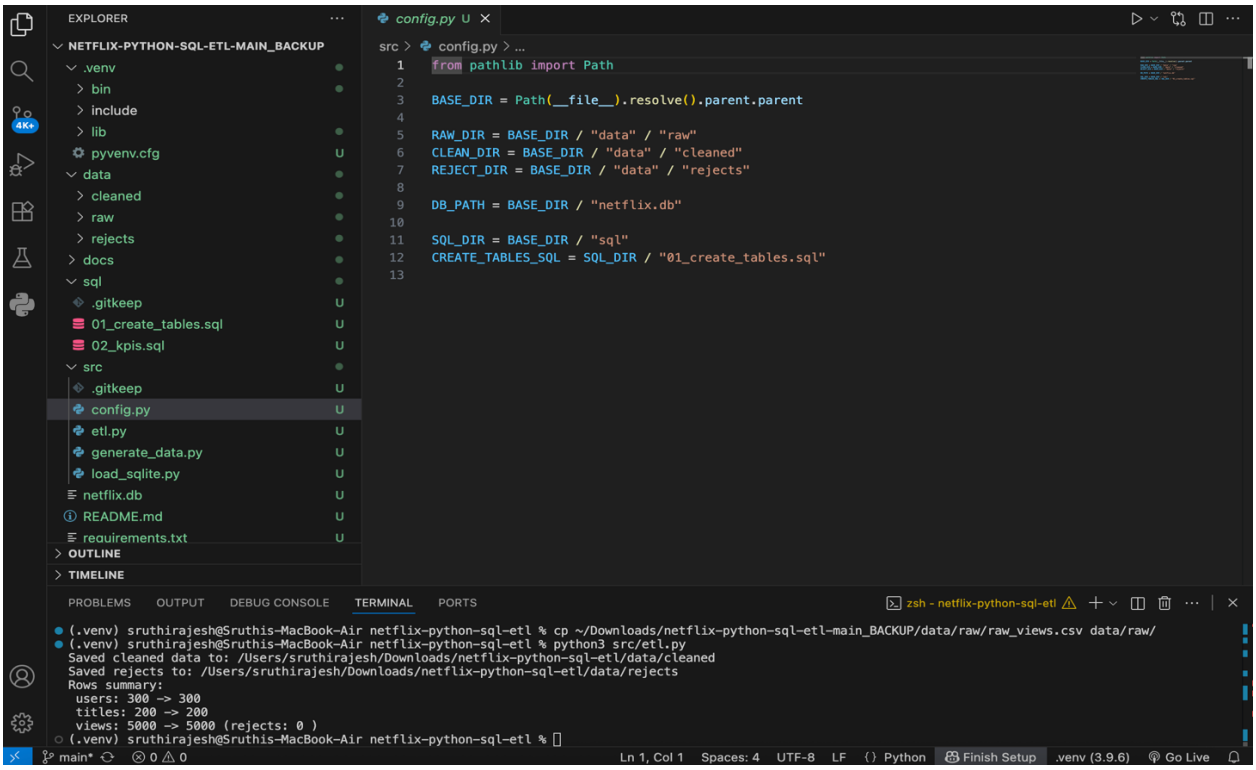
Screenshots:

The following screenshots provide proof that each stage of the pipeline ran successfully. These screenshots are stored in the repository under:

- docs/results_screenshots/

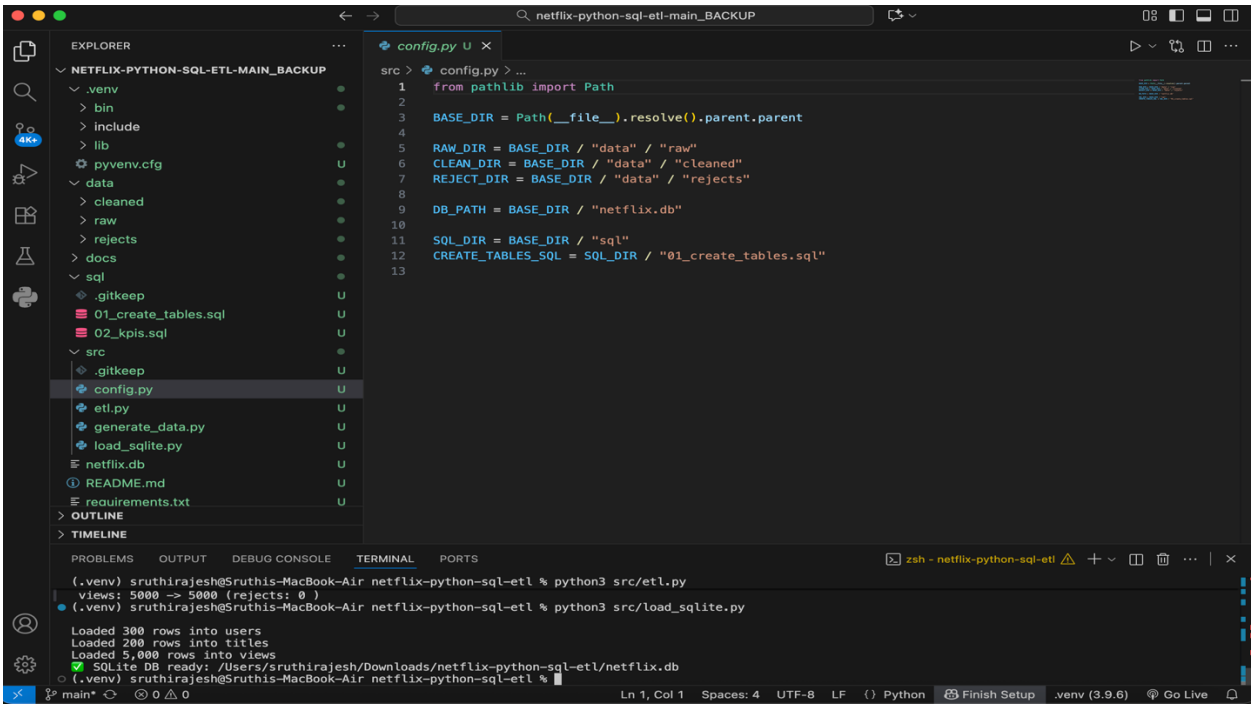
If the assessor requires the evidence inside one PDF, paste them below in the placeholders before exporting.

Screenshot 1 — ETL run output
Confirms the ETL ran successfully and produced outputs in data/cleaned/ and data/rejects/ with row counts summary.



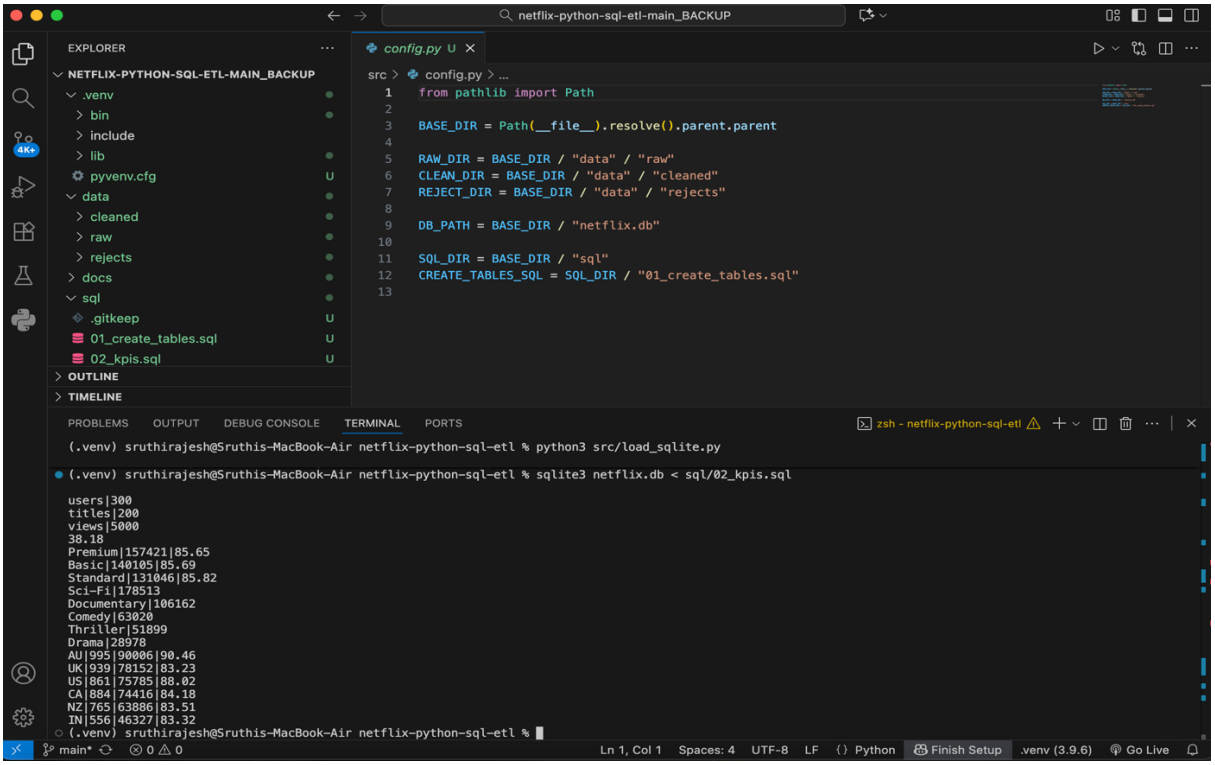
Screenshot 2 — SQLite load output

Confirms cleaned datasets were loaded into SQLite tables (users, titles, views) and netflix.db was created/updated.



Screenshot 3 — KPI query run output

Confirms KPI queries executed successfully and produced results in the terminal.



Conclusion

This project successfully demonstrates a complete **end-to-end ETL workflow** using **Python (pandas)** for data cleaning/validation and **SQLite + SQL** for structured storage and KPI reporting. Raw Netflix-style viewing logs were transformed into consistent, analytics-ready datasets, invalid records were isolated with clear reject reasons, and the cleaned data was loaded into a relational database with a defined schema. Finally, reusable SQL KPI queries were executed to generate insights such as engagement by plan type, country-level viewing patterns, and title-level watch performance. Overall, the project shows practical ability to build a clean data pipeline, enforce data quality, and produce business-ready metrics in a repeatable way.