📘 Grad Café Analytics (Module 4)

Overview

This project implements a fully tested, fully documented analytics pipeline for the Grad Café admissions results dataset. It extends the Module 3 ETL + Flask system with:

- A complete Pytest suite (web, buttons, analysis, DB, integration)
- ~97% test coverage across all modules
- Sphinx documentation published to Read the Docs
- GitHub Actions CI with PostgreSQL
- A clean, testable Flask application using a factory pattern

The system provides:

- A web dashboard (/analysis)
- "Pull Data" and "Update Analysis" actions
- Scraping → cleaning → loading into PostgreSQL
- Summary analysis queries rendered in the UI

Project Structure

```
module_4/
│
├── src/
│   ├── app/            # Flask app, routes, templates
│   ├── module_2_1/       # Scrape + clean modules from M3
│   ├── load_data.py      # DB loader
│   ├── query_data.py      # Analysis queries
│   └── run.py           # App entry point
│
├── tests/             # Full Pytest suite
│
```

```
├── docs/              # Sphinx documentation
│
├── pytest.ini
├── requirements.txt
├── coverage_summary.txt
├── actions_success.png
└── .github/workflows/tests.yml
```

Running the Application

1. Install dependencies

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

2. Set environment variables

The application uses:

```
DATABASE_URL=postgresql://<user>:<password>@<host>:<port>/<dbname>
```

Example for local development:

```
export DATABASE_URL=postgresql://postgres:postgres@localhost:5432/gradcafe
```

3. Run the Flask app

```
flask --app src.app run
```

Running Tests

Full suite with coverage

```
pytest -q --cov=src --cov-report=term-missing
```

Marker-based execution (required by assignment)

```
pytest -m "web or buttons or analysis or db or integration"
```

Markers Used

Marker	**Purpose**
web	Flask page rendering

Marker	Purpose
buttons	Pull/update behavior + busy-state logic
analysis	Formatting, labels, rounding
db	PostgreSQL schema, inserts, idempotency
integration	End-to-end pipeline tests

All tests in this project are marked as required.

📊 Coverage Achievement

This project achieves **~97% test coverage** — the maximum possible without using # pragma: no cover comments.

Per-Module Breakdown

Module	Coverage	Notes
app/__init__.py	100%	Factory pattern, filters
app/queries.py	100%	Scraper diagnostics
app/routes.py	99%	All routes and error paths
query_data.py	100%	All 6 analysis queries
module_2_1/clean.py	99%	Cleaning + LLM batch
module_2_1/scrape.py	96%	Parallel scraper
load_data.py	94%	DB loader + CLI
run.py	80%	Flask entrypoint

What's Covered ✅

- 100% of all business logic
- 100% of all testable code paths
- All route handlers, including error and busy-state branches
- All ETL stages: scrape → clean → load → query

- Edge cases: empty inputs, invalid data, subprocess failures

What's Not Covered (and Why) ⚠️

The remaining ~3% consists entirely of if __name__ == "__main__": guard blocks — standard Python CLI entrypoints that cannot be executed during import-based test collection. These are present in run.py, load_data.py, clean.py, and scrape.py. All logic within these blocks has been extracted into callable main() functions that are fully tested.

All production business logic has complete test coverage. 🎯

GitHub Actions CI

A full CI pipeline is included under:

module_4/.github/workflows/tests.yml

The workflow:

- Starts PostgreSQL 15
- Installs dependencies
- Sets DATABASE_URL
- Runs the full Pytest suite
- Enforces coverage threshold

A screenshot of a successful run is included as:

module_4/actions_success.png

Sphinx Documentation

Sphinx docs are located in:

module_4/docs/

They include:

- Overview & setup
- Architecture (Web, ETL, DB)
- API reference (autodoc)
- Testing guide (markers, fixtures)
- Build instructions

Build locally

cd module_4/docs

make html

Published Documentation

📘 **Live Documentation**: https://sphinx-demo-erying1.readthedocs.io/en/latest/

Key Features

- **Testable Flask App** — Factory pattern and stable HTML selectors (data-testid="...") for reliable UI tests
- **Full ETL Pipeline** — Scraping → cleaning → LLM-enhanced normalization → PostgreSQL loading
- **Analysis Engine** — Computes summary statistics used by the dashboard
- **~97% Test Coverage** — All modules covered, including error paths and edge cases
- **CI + Documentation** — Automated testing and published developer documentation

Developer Notes

- All external processes (scraper, cleaner, loader) are mocked in tests
- No test depends on live network calls
- Busy-state logic is deterministic and observable
- Database tests use DATABASE_URL to allow CI overrides
- Dead code (unreachable exception handlers, duplicate functions) has been removed during refactoring

License

This project is part of the JHU "Modern Software Development in Python" course (Spring 2026). For educational use only.