MLOps Engineering

Machine Learning Operations V2.0.0

1st Group Assignment

MsC in Business Analytics and Data Science

Madrid, May 2025

ie UNIVERSITY

Expectations for both best practices to be showcased, and the modules to be delivered for this first phase

General best practices

1. **Modularization**: Break down code into smaller, reusable functions or classes
2. **Configuration Management**: Use configuration files or environment variables for settings
3. **Logging and Monitoring**: Add logging and possibly some basic monitoring metrics
4. **Error Handling**: Include comprehensive error handling and data validation
5. **Dependency Management**: List all dependencies and versions for reproducibility
6. **Documentation**: Include sufficient comments and documentation for maintainability
7. **Code Quality**: Ensure the code is clean, readable, and follows PEP 8 standards
8. **Testing**: Include unit tests to validate the functionality
9. **Security**: Remove any sensitive or secret info (.env)

ML Pipeline specific (7 ML modules)

1. **data_loader.py**: Load data from CSV, databases, or APIs
2. **data_validation.py**: Validate schema, check types, handle missing data
3. **preprocessing.py**: Clean, scale, encode, and transform input data
4. **features.py**: Create and select engineered features
5. **model.py**: Define, train, save, and load ML models
6. **evaluation.py**: Compute and log performance metrics
7. **inference.py:** Apply trained model to new data (predict pipeline)
8. **Main.py**: Entry point and orchestrator

Other:

1. config.yaml, main.py, envrionemnt.yml, tests suit (mirrors modules src structure), .gitignore, README.md, logs/ main_log.log
2. Executive summary of Business Case

1st Group assignment expectations and guidelines (Tech section)

Category	Details
Code Quality	• The code is following the **PEP8 coding style** (No errors on VSC > PROBLEMS tab) e.g. Meaningful names and syntax; properly commented and formatted; Imports are ordered, etc.
Documentation	• There's a comprehensive **README.rm** providing a general (preliminary) overview of the project, instructions on how to use the code, run tests, etc. [Anyone should be able to run the code from those instructions] • All functions and files have **docstrings**, and **commenting** is used appropriately i.e. **Why NOT What**
Testing	• **Tests cover all functions** and relevant artifacts e.g. Inputs are in the expected file format? Outputs are saved in the right places? Edge cases? (pytest-cov)
Logging	• **Each function** is complete with **logging** (INFO or otherwise), and Logs are stored in a **.log file** (Distinguishing between logs relevant to keep track of, and those to be shown on the terminal)
Config mgmt.	• A **config.yaml** (**python_env.yaml**) file(s) is used for Configuration Management and environment variables for settings (Avoid hardcoding variables!!)
Dependencies	• **Dependencies** are managed through a **environment.yml** and/or **conda.yml** file, listing all dependencies and versions for reproducibility
Error handling	• Error Handling thru **try/ except/ raise** provides a comprehensive error handling and **data type validation**
Artifacting	• Relevant reports, notebooks, or images of plots from the EDA phase are stored on a respective folder • A model(s) is properly stored (joblib or pickle) for future production use in inference phases
ML Pipeline Modules	The code modules cover and end-to-end ML process e.g. **1.** **main**.py (with **argparse** for CLI) **2.** **data_loader**.py: Load data from CSV, databases, or APIs **3.** **data_validation**.py: Validate schema, check types, handle missing data **4.** **preprocessing**.py: Clean, scale, encode, and transform input data **5.** **features**.py: Create and select engineered features **6.** **model**.py: Define, train, save, and load ML models **7.** **evaluation**.py: Compute and log performance metrics **8.** **inference**.py: Apply trained model to new data (predict pipeline) 9. utils.py (optional): Shared utilities (e.g. logging setup, plotting functions)
Version Controlling	GitHub is used for Version Control and development, demonstrating: • Proficiency with basic operations e.g. add, commit, push, etc. • Follow best practices (as discussed in class) for: 1. Project creation 2. Branching (Check out features, debugging, releases, development vs. main) 3. Pulls Request reviewing (Open, review and merge) 4. Tagging and releasing (version control and release) 5. Collaboration & Communication (Discussions and resolutions)

Overall project expectations and guidelines (Tech Section)

Criterion	Fundamentals missing (0–4)	Basic attempt (5–6)	Good implementation (7–8)	Excellent – Industry grade (9–10)
Single Entry Point & Modularization	Workflow poorly extracted from notebooks; minimal modularization	Basic script exists, limited/ not complete modularization, manual intervention required	Main script runs most modules; partial orchestration	Fully orchestrated modular pipeline
Code Quality & Efficiency	Frequent violations of best practices (PEP 8), inefficient code	Occasional issues; some inefficient loops or duplication	Generally clean, minimal duplication; minor efficiency gaps	Zero PEP 8 issues, fully optimized, vectorized, and efficient code
Documentation & Clarity	Missing/incomplete README, no docstrings	README exists but minimal; incomplete docstrings, comments miss the "Why?"	Useful README; most functions documented, further clarity improvements needed	Comprehensive README; all functions have clear docstrings, and relevant comments (Why)
Testing & Coverage	Minimal tests (<20% coverage), no edge cases	Basic tests (~50% coverage), lacks critical edge cases	Good tests (~75% coverage), some edge cases covered	Extensive tests (≥90% coverage), thorough edge-case tests and mock-up data
Logging & Monitoring	Using print statements only, no logging framework	Basic logging setup; no structured file logging	Good logging practices; minor gaps in structured logging (Console vs. file)	Robust structured logging, logs saved properly and optimal levels
Config & Dependency Management	Hard-coded paths, no environment management	Basic use of configuration or dependencies managed	Proper dependency management, partial config usage	Fully configurable, reproducible environments, secure secrets
Error Handling & Validation	Frequent uncaught exceptions, no validation	Basic try/except blocks; insufficient custom error handling	Adequate error handling; some schema validation implemented	Comprehensive error handling with custom exceptions, full validation
Artifacting & Reproducibility	No artifacts or outputs systematically stored	Partial storage; unclear or inconsistent model artifact paths	Clear storage for models; some reports or images documented	Fully reproducible artifacts; clearly structured assets
Pipeline Completeness	Partial implementation (≤4 core modules)	Majority implemented (≥5 core modules); loosely coupled	All core modules implemented; some coupling issues	Complete modular pipeline; seamless integration
Version Control & Workflow (Optional)	Minimal GitHub use; unclear or no evidence of individual contributions	Basic GitHub use; partial evidence of contributions from some members	Good GitHub practices; clear but uneven distribution of contributions	Strong GitHub use; explicit evidence each member owns distinct code/tests modules, showing significant individual contributions

(Preliminary) Basic Business Section

Key takeaway: <Why should the client embark on this MLOps project vs. BAU e.g. Jupyter notebooks?>

Client
– Client name and industry

Business Unit
- Business Unit or department

What's the maturity of the client
– Data, Tools, Processes, People, Strategy

Goal of Project (Objective metric, Improvement over baseline)
– In business or client terms
– **As measured by (quantifiable KPI)?**

Problem Statement
– What's the key pain point for the client?
– State the business pain point and current baseline metric
– **As measured by (quantifiable KPI)?**

Solution Description & Key Functionalities
- Simple, executive level description of the solution (Non-technical terms)

Solution Scalability
– How can this solution be applied to other use cases, industries, business functions?
– Can the solution grow in performance and scope, how?

Client Benefit (Over non-AI approach)
– Tangible (or intangible benefits for the client that wouldn't be attainable thru conventional non-AI approaches
– Short and long term
– Competitiveness
– Core business or new adjacent opportunities ?
– **As measured by (quantifiable KPI)?**

Cost estimation ($000) (ball park)
– Talent:
  – AI specialist
  – Product Mgr.
  – ML/SW Engineer
  – Data Engineer
  – SME
– The Client to cover:
  – Data
  – Infrastructure
  – Licenses
– Time: 12+ wks

Risk and challenges?
– What are undesired outcomes or potential problems that need to be addressed e.g. data quality, skills gap, security reviews
– Planned mitigations

Deliverable check-list review (1st group assignment)

1. Is the code clean and modular?
2. Can I understand the code easily?
3. Does it use meaningful names?
4. Is there duplicated code?
5. Can I provide another layer of abstraction?
6. Is each function and module necessary?
7. Is each function or module too long?
8. Is the code efficient?
9. Are there loops or other steps I can vectorize?
10. Can I use better data structures to optimize any steps?
11. Can I shorten the number of calculations needed for any steps?
12. Can I use generators or multiprocessing to optimize any steps?
13. Is the documentation effective?
14. Are inline comments concise and meaningful? i.e. Why not What
15. Is there complex code that's missing documentation?
16. Do functions use effective docstrings?
17. Is the necessary project documentation provided? i.e. README.md
18. Is the code well tested? i.e. each module has a test suit?
19. Does the code high test coverage?
20. Do tests check for interesting/ edge cases?
21. Are the tests readable?
22. Can the tests be made more efficient?
23. Is the logging effective?
24. Are log messages clear, concise, and professional?
25. Do they include all relevant and useful information?
26. Do they use the appropriate logging level?

Source: Udacity

Recommended Steps for 1st group assignment (technical section)

1. **Create new conda environment from yml file e.g.**
   - conda env create -f environment.yml
   - conda activate <env_name>
2. **Create new project structure (cookiecutter)** <- **preferred** to do it **manually**
   - cookiecutter [https://github.com/drivendataorg/cookiecutter-data-science -c v1](https://github.com/drivendataorg/cookiecutter-data-science)
   - cookiecutter https://github.com/fmind/cookiecutter-mlops-package
3. **Create a GitHub repo e.g.**
   - **Locally**
     - git init
     - git add .
     - git commit -m "Initial commit“
   - **On GitHub**
     - ‘New’
     - ‘Create repository’
     - Copy URL
   - **Locally**
     - git remote add origin https://github.com/username/repository.git
     - git branch -M main
     - git push -u origin main

1

(illustrative)

Example of environment.yml

```yaml
name: mlops_project  # Name of the conda environment

channels:
  - conda-forge  # Community channel with most up-to-date packages

dependencies:
  - python=3.10     # Stable and widely-used Python version for compatibility
  - pandas          # Core library for data loading (CSV, Excel) and
manipulation
  - openpyxl        # Excel support for pandas (read/write .xlsx files)
  - pyyaml          # Read configuration from YAML files for flexibility
  - python-dotenv   # Load environment variables from a .env file (for
secrets/config)
  - pytest          # Testing framework for robust, maintainable code
  - pip             # Allows pip-only packages to be installed below

  - pip:
      - pytest-cov  # Test coverage reporting for code quality and
completeness
      - black       # Code formatting to enforce style and readability
      - flake8      # Linting to check code for errors and best practices
```

2

Simplified project structure (illustrative)

```
project_name/
├── data/            # Raw and processed data
├── notebooks/       # Jupyter notebooks for exploration
├── src/             # Source code modules
│   ├── data/        # Data loading and preprocessing
│   ├── features/    # Feature engineering
│   ├── models/      # Model definitions and training
│   └── utils/       # Utility functions
├── tests/           # Unit and integration tests
├── configs/         # Configuration files (YAML/JSON)
├── environment.yml
└── README.md
```

2

(illustrative) Module example for **data_loader.py** ran as script, and from entry point **main.py**

```
.
├── README.md
├── config.yaml
├── data
│   ├── processed
│   ├── raw
│   │   └── opiod_raw_data.csv
│   └── splits
├── environment.yml
├── logs
│   └── data_loader.log
├── notebooks
│   └── opiod_analysis_v01.ipynb
├── src
│   ├── __pycache__
│   │   └── main.py
│   ├── data
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.py
│   │   │   └── data_loader.py
│   │   └── data_loader.py
│   └── main.py
└── tests
    └── test_data_loader.py
```

```
> python -m src.data.data_loader

2025-05-15 23:45:57 - INFO - root - Loaded data from ./data/raw/opiod_raw_data.csv (csv), shape=(1000, 22)
Data loaded successfully. Shape: (1000, 22)

> python -m src.main

2025-05-16 01:12:19 - INFO - root - Pipeline started
2025-05-16 01:12:19 - INFO - root - Loaded data from ./data/raw/opiod_raw_data.csv (csv), shape=(1000, 22)
2025-05-16 01:12:19 - INFO - root - Data loaded successfully. Shape: (1000, 22)
Data loaded. Shape: (1000, 22)
2025-05-16 01:12:19 - INFO - root - Pipeline completed successfully
```

(illustrative) Example: Standard **Cookiecutter DataScience** project structure

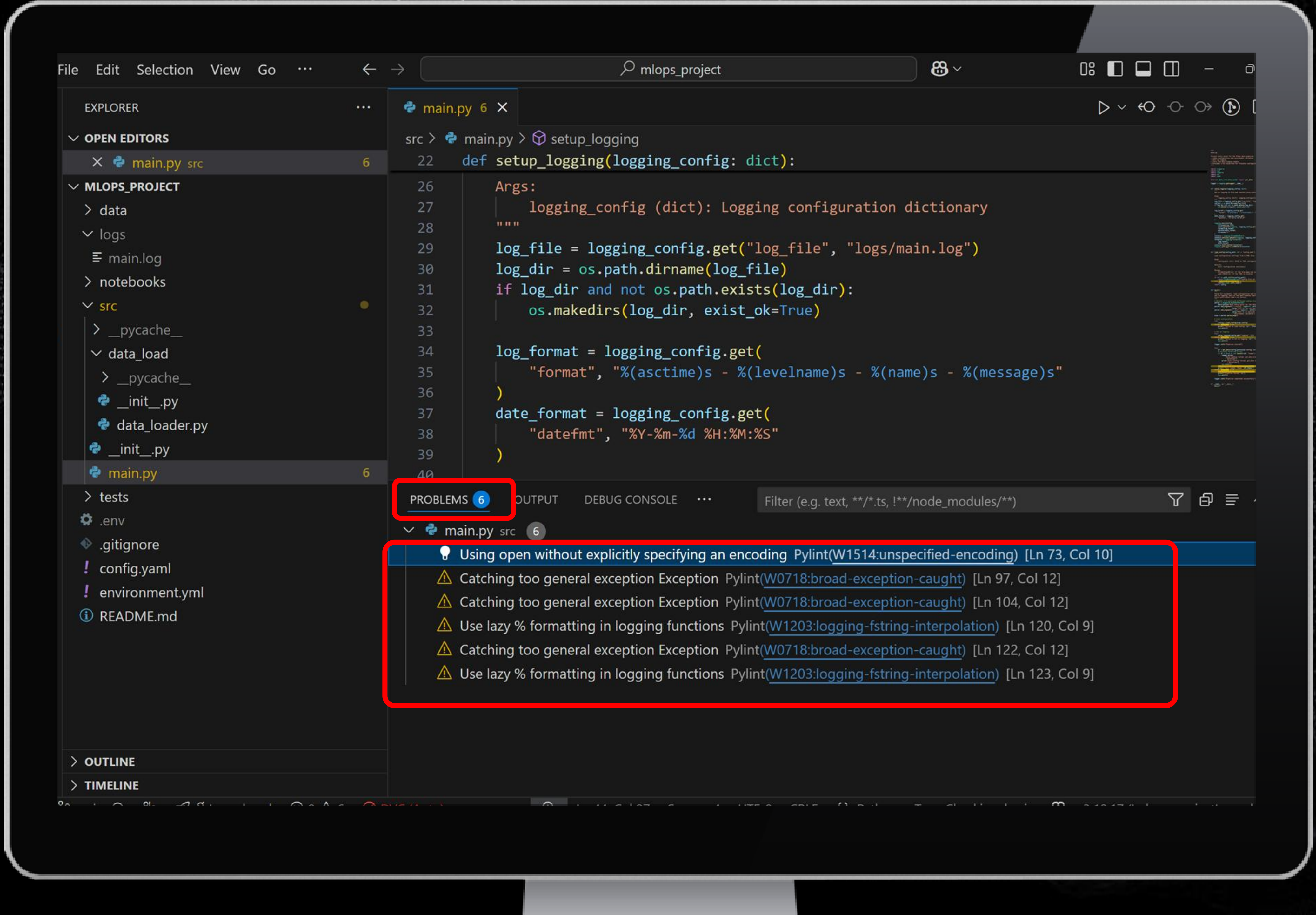
```
├── data
│   ├── external
│   ├── interim
│   ├── processed
│   └── raw
├── docs
├── models
├── notebooks
├── references
├── reports
│   └── figures
└── src
    ├── data
    ├── features
    ├── models
    └── visualization
```

```
environment.yml
name: cookie
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.13
  - pip
  - pip:
      - cookiecutter

> conda env create -f environment.yml
> conda activate cookie

# DataScience
> cookiecutter
https://github.com/drivendataorg/cookiecu
tter-data-science -c v1

# MLOps
> cookiecutter
https://github.com/fmind/cookiecutter-
mlops-package
```

Steps – Refactoring the Jupyter Notebook

4. **Move the Jupyter Notebook to /notebooks**
5. **Run and Check Jupyter Notebook** (Ensure your notebook actually works **end-to-end**!)
6. **Translating the . Ipynb to .py** <- Recommend doing it **manually**
   - jupytext --to py notebook.ipynb
   - Pairing Notebooks (optional)
     - jupytext --set-formats ipynb,py notebook.ipynb
7. **Checking formatting standards**
   - Pylint and autopep8 installed on VSC
   - Use GitHub Copilot (AI agent) for refactoring and code optimization (be **careful** on what it does to your code!)
8. **Create individual modules for each section (Look at GitHub example [mlops-project])**
   1. Checking formatting standards
   2. Config mgmt.
   3. Add logging
   4. Error Handling / Try – Exception – Raise – sys.error(1)
   5. Maintain dependencies updated (environment.yml)
   6. Include sufficient comments and documentation
   7. Code Quality: Ensure the code is clean, readable, and follows PEP 8 standards
9. **Generate test suit for each function** (**Do not move forward** until your module passes your tests)

ie
UNIVERSITY



7

8

(illustrative) Module example for **data_loader.py** ran as script, and from entry point **main.py**

```
.
├── README.md
├── config.yaml
├── data
│   ├── processed
│   ├── raw
│   │   └── opiod_raw_data.csv
│   └── splits
├── environment.yml
├── logs
│   └── data_loader.log
├── notebooks
│   └── opiod_analysis_v01.ipynb
├── src
│   ├── __pycache__
│   │   └── main.py
│   ├── data
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   │   ├── __init__.py
│   │   │   └── data_loader.py
│   │   └── data_loader.py
│   └── main.py
└── tests
    └── test_data_loader.py
```

```
> python -m src.data.data_loader

2025-05-15 23:45:57 - INFO - root -
Loaded data from
./data/raw/opiod_raw_data.csv (csv),
shape=(1000, 22)
Data loaded successfully. Shape:
(1000, 22)

> python -m src.main

2025-05-16 01:12:19 - INFO - root -
Pipeline started
2025-05-16 01:12:19 - INFO - root -
Loaded data from
./data/raw/opiod_raw_data.csv (csv),
shape=(1000, 22)
2025-05-16 01:12:19 - INFO - root -
Data loaded successfully. Shape:
(1000, 22)
Data loaded. Shape: (1000, 22)
2025-05-16 01:12:19 - INFO - root -
Pipeline completed successfully
```

8

(illustrative) Test suit example for **tests/** directory.

```
.
├── data
│   ├── mock_data.csv
│   ├── mock_data.json
│   ├── mock_data.xlsx
│   └── unsupported_file.txt
├── test_data_cleaning.py
├── test_data_exploration_profiling.py
├── test_data_loader_with_mock_data.py
├── test_feature_engineering.py
├── test_model_building.py
├── test_model_fairness.py
├── test_model_interpretability.py
└── test_model_tuning.py
```

```
> pytest tests/

============================== test session starts ==============================
platform linux -- Python 3.x.x, pytest-x.x.x, ...
rootdir: /path/to/your/project
collected XX items

test_data_cleaning.py ..........
test_data_exploration_profiling.py ........
test_data_loader_with_mock_data.py .........
test_feature_engineering.py ........
test_model_building.py .........
test_model_fairness.py .........
test_model_interpretability.py ........
test_model_tuning.py .........

========================== XX passed in 1.23s ================================
```