MLOps Engineering
Machine Learning Operations V2.0.0
Sessions 6 - 7

MsC in Business Analytics and Data Science

Madrid, May 2025

0

main.py

Orchestrate the full ML pipeline from config, logging every action for traceability and robust error handling

Config drives the pipeline
- Loads config.yaml and .env for all parameters
- Pipeline logic is never hardcoded

Support modular stages
- Choose to run full, data, or train pipeline
- Enables rapid debugging and development

Centralize all logging
- Configure and initialize logging once
- Logs are consistent across all modules

Explicit error handling
- Catch, log, and exit on all critical errors
- Never allow silent failures

Pass data between steps
- Each step hands off to the next via clear variables
- Enables flexible pipeline rearrangement

Log every pipeline event
- Logs all key steps, actions, and errors
- All pipeline runs are fully traceable

1

data_loader.py

Load and validate all data using config, preserving raw files and logging every detail for auditability

Config controls all loading

- All paths, types, and file options set in config.yaml
- No hardcoded values used

Validate every config value

- Check for missing or invalid paths before loading
- Fail fast if config has errors

Never overwrite raw data

- Always read, never mutate raw input files
- Protect original source for full reproducibility

Flexible format support

- Load CSV, Excel, or future formats with config switch
- Adapt quickly to data changes

Log every step and file

- Log data paths, file types, and load status
- Always log file shape and success/failure

Fail safe and clearly

- Raise clear errors on missing files or wrong formats
- Log every error with actionable detail

2

data_validation.py

Validate and enforce data schema, types, and completeness before entering the pipeline (GIGO) at every critical data entry point

Schema defined in config

- Column types and expectations set in config.yaml
- Data validation logic is version-controlled and easy to update

Check for missing fields

- Warn or fail on missing required columns
- Never proceed with incomplete schema

Validate data types

- Ensure each column matches expected type
- Raise clear error if mismatch

Handle missing values explicitly

- Identify, log, and optionally impute or reject rows
- Prevent silent data loss

Fail fast on errors

- Action (raise/warn) is configurable from config.yaml
- Stop pipeline or log warning as per project phase

Log all validation results

- Store full validation report as JSON for audit and reproducibility
- Log all checks, issues, and summary for traceability

3

preprocessing.py

Fit preprocessing only on the train split and drive all logic from config for leakage-proof, reproducible ML

Config controls all transforms

- All preprocessing steps set in config.yaml
- Reproducible experiments every run

Always fit on train split only

- Never use test data for fitting
- Prevents all data leakage

Never overwrite raw data

- Keep raw, processed data separate
- Maintain full audit trail

Each function does one job

- Split, rename, scale, encode steps modular
- Enables easy testing

Log every transformation

- Log each step, shape, and file path
- All logs controlled by config

Validate config before use

- Check config keys, warn if missing
- Fail fast on config errors

4

features.py

Create, transform, and select features modularly and track all changes for reproducibility

Config controls feature logic

- All engineering steps defined in config.yaml
- Supports easy iteration and audit

Modular feature functions

- Each transformation in a dedicated function
- Enables isolated testing

Enable feature selection

- Select features by config or method (e.g. importance)
- No hardcoded lists

Track all feature changes

- Log every new or modified feature
- Maintain full audit trail

Preserve original columns

- Never overwrite raw features
- Keep raw, engineered, and selected features separate

Log every feature step

- Log creation, selection, and dropping of features
- Store feature list for reproducibility

5

model.py

Centralize model pipeline control in config.yaml and always split data before preprocessing to guarantee robust results

All pipeline settings in config

- Models, splits, metrics set in config
- No hardcoding anywhere

Split before preprocessing

- Raw data split first, never after
- Strictly prevents leakage

Switch models via config

- Change model type with model.active
- Params per model in config

Keep all data stages separate

- Raw and processed files always distinct
- Easy to trace lineage

Log every pipeline stage

- Log data load, split, train, eval
- Paths and shapes always logged

Modular and testable code

- Every function takes clear inputs
- All parts unit testable

6

evaluation.py

Evaluate model using configurable metrics and log all results for transparent model comparison

Config lists all metrics
- Metrics set in config.yaml, not hardcoded
- Enables flexible evaluation

Calculate each metric modularly
- Each metric in its own function
- Easily add or remove metrics

Log every result
- Log every metric, split, and model version
- Store results for each run

Support multiple splits
- Evaluate and log metrics for each split (train/val/test)
- Enables fair, transparent model comparison

Enable model comparison
- Store metrics for multiple models
- Track changes across experiments

Fail safe on metric errors
- Catch and log errors, never crash
- Continue evaluating all metrics and splits

7

inference.py

Serve predictions using the trained pipeline and log all inference for reproducibility

Load pipeline from artifacts
- Always load model and transformer from saved files
- No training logic in inference

Validate input data
- Check new data matches expected schema
- Warn or fail on mismatch

Apply full preprocessing
- Transform input exactly as in training
- Prevent feature drift

Return predictions and probabilities
- Output both class and probability if available
- Flexible output options

Log all inference events
- Log input shape, prediction output, and time
- Store all requests for audit

Fail safe and trace errors
- Catch and log prediction errors
- Never serve incomplete predictions

Unit tests ensure **code quality** before deployment, while production **monitoring** safeguards **reliability and performance** with real data

DEV

- Uses only **mock or synthetic** data
- Validates code logic in **isolation**
- Covers both **success and failure** scenarios
- Fast, repeatable, and **deterministic**
- Never **touches production** systems or data
- **Automated** and **run frequently** (CI/CD)
- Focuses on **small, isolated** functions or modules
- Can use **monkeypatching** or mocks
- Designed for **debugging** and fast **feedback**
- Ensures functions handle **edge cases** robustly

PROD

- Uses **real, live** production data
- Monitors model and **pipeline behaviour continuously**
- Detects **data drift and schema changes**
- Validates **live predictions** for **outliers**
- **Runs in parallel** to actual production workloads
- Utilizes **shadow or canary** deployments
- Relies on **alerts**, **dashboards**, and **logging**
- Focuses on **performance** and **reliability**
- Ensures **rapid rollback** if failures are detected

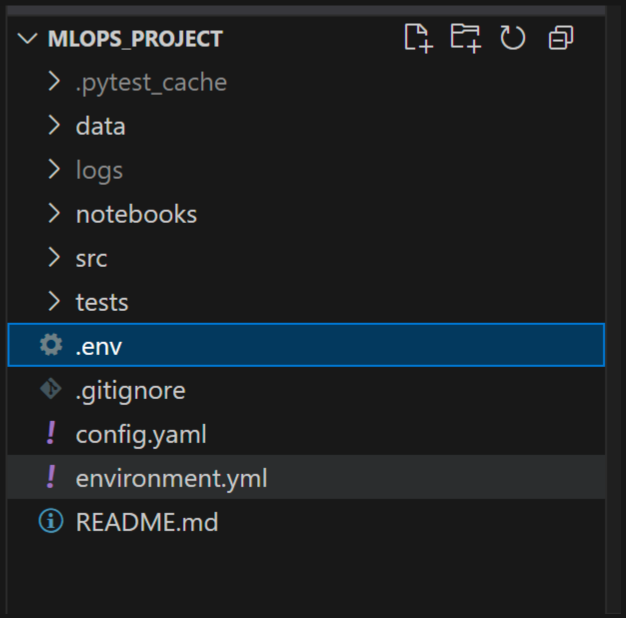
Running robust tests:

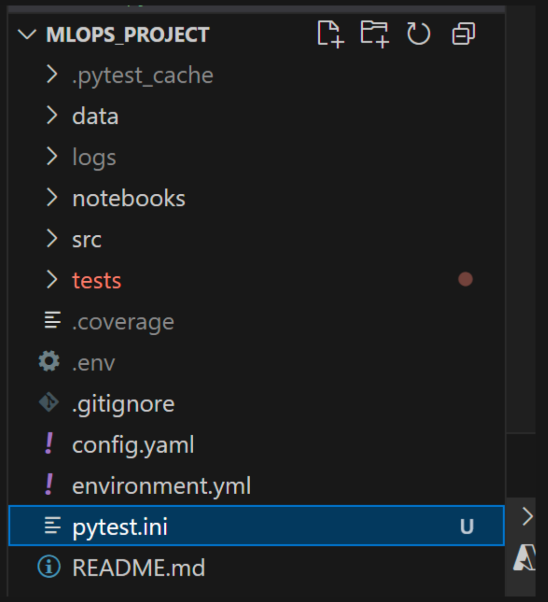
1. Install pytest-dotenv

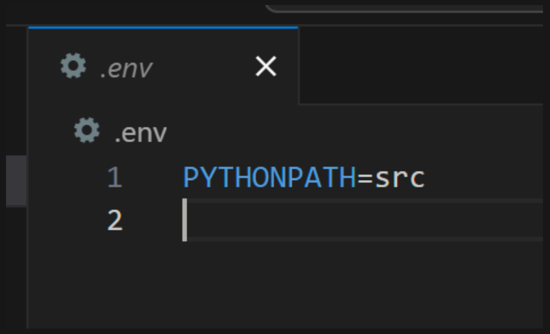
2. *pytest.ini*:
```
[pytest]
env_files = .env
```

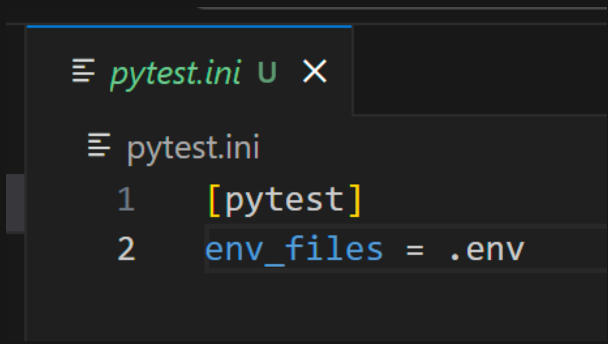
3. *.env*:
```
PYTHONPATH=src
```

Ensure all project code is discoverable by pytest









Most common pytest calls

From your root folder:

```
pytest
pytest -q
pytest –v
pytest tests/test_file_name.py
pytest tests/test_file_name.py::test_function_name

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

```
============================================================ tests coverage =============================================================
___________________________________________ coverage: platform win32, python 3.10.17-final-0 ____________________________________________

Name                              Stmts   Miss  Cover
-----------------------------------------------------
src\__init__.py                       0      0   100%
src\data_load\__init__.py             0      0   100%
src\data_load\data_loader.py         53     16    70%
src\main.py                          57     57     0%
-----------------------------------------------------
TOTAL                               110     73    34%
=========================================================== 4 passed in 1.06s ===========================================================
(mlops_project) PS C:\Users\idiaz\OneDrive - IE University\00. IE Courses\01. 2025_H1\4. MLOps\mlops_project>
```