Report on Linting, Formatting, and Testing in the Data Preprocessing Project

Author: Unai Lalana

0. GitHub

The complete repository for this project, including all source code, CLI implementation, and test files, can be accessed at GitHub: Lab0.

1. Introduction

The main objective of this assignment is to understand the fundamentals of Continuous Integration (CI). specifically, some of the basic elements which are the linting, search for syntax errors to guarantee code correctness, automatic code formatting (black) and testing, to ensure the code functionality.

The project was organized following a standard structure:

- **src/** – Contains the main logic (`preprocessing.py`).
- **tests/** – Contains unit and integration tests for (`preprocessing.py`).
- **pyproject.toml** – Configuration file defining dependencies and formatting rules.
- **README.md** – Documentation describing usage and setup.

All dependencies were managed with a virtual environment created using the `uv` package:

```
uv init
uv sync
uv add click black pylint pytest pytest-cov
```

2. Testing Logic

The testing logic was designed to cover both the functional correctness of individual components and the integration between the command-line interface (CLI) and the core logic. The unit test cases were written before the functionalities to follow the test-driven development pipeline.

2.1 Unit Testing

Unit tests were written using the `pytest` framework and covered all preprocessing functions, including:

- Cleaning data (removal and filling of missing values).
- Numeric transformations (normalization, standardization, clipping, integer conversion, logarithmic transformation).
- Text processing (tokenization, stopword removal, alphanumeric selection).

- Structural operations (flattening, deduplication, and shuffling).

The `@pytest.fixture` decorator was used to define reusable test data, such as lists of numbers or text samples. The `@pytest.mark.parametrize` decorator was applied to execute tests with multiple input combinations, verifying both optional and required parameters.

2.2 Integration Testing

Integration tests validated the correct operation of the `click`-based CLI. A shared `CliRunner` fixture (from `click.testing`) was used to simulate command-line invocations. Each test checked that:

- The command executed successfully.
- Output values matched expected results from the logic functions.
- Grouping and help messages were correctly displayed (`cli`, `clean`, `numeric`, `text`, `struct`).

3. Linting and Formatting

- **Linting:** The project was analyzed using `Pylint`. After minor adjustments (mainly related to line length), the score reached:

  ```
  Your code has been rated at 10.00/10
  ```

- **Formatting:** Code formatting was automatically handled with `Black`, ensuring full compliance with PEP8 conventions and consistent code structure.

4. Testing and Coverage Results

All tests were run with `pytest`:

```
uv run python -m pytest -v
```

All tests passed successfully. Coverage was measured with `pytest-cov`:

```
uv run python -m pytest -v --cov=src
```

The resulting report indicated 94% test coverage at preprocessing.py, 80% test coverage at cli.py and 87% coverage across all modules.

Example output:

```
======================== test session starts========================
platform win32 -- Python 3.12.10, pytest-8.4.2, pluggy-1.6.0 python
cachedir: .pytest_cache
rootdir: \Lab0
configfile: pyproject.toml
plugins: cov-7.0.0
collected 20 items
```

```
tests/test_cli.py::test_cli_remove_missing PASSED
tests/test_cli.py::test_cli_fill_missing PASSED
tests/test_cli.py::test_cli_normalize PASSED
tests/test_cli.py::test_cli_struct_shuffle PASSED
tests/test_logic.py::test_remove_missing_values PASSED
tests/test_logic.py::test_fill_missing_values PASSED
tests/test_logic.py::test_remove_duplicates PASSED
tests/test_logic.py::test_min_max_normalize[data0-expected0] PASSED
tests/test_logic.py::test_min_max_normalize[data1-expected1] PASSED
tests/test_logic.py::test_z_score_normalize PASSED
tests/test_logic.py::test_clip_numerical_values[data0-0-20-expected0] PASSED
tests/test_logic.py::test_clip_numerical_values[data1-1-3-expected1] PASSED
tests/test_logic.py::test_to_integer_values PASSED
tests/test_logic.py::test_logarithmic_transform[data0-expected0] PASSED
tests/test_logic.py::test_logarithmic_transform[data1-expected1] PASSED
tests/test_logic.py::test_tokenize_text PASSED
tests/test_logic.py::test_select_alphanumeric_and_spaces PASSED
tests/test_logic.py::test_remove_stopwords PASSED
tests/test_logic.py::test_flatten_list PASSED
tests/test_logic.py::test_random_shuffle_list PASSED

======================== 20 passed in 0.31s========================
```

5. Conclusion

The project demonstrates the essential steps of continuous integration: code linting, formatting, testing, and coverage analysis. By integrating `Pylint`, `Black`, `Pytest`, and `Pytest-cov` within a controlled `uv` environment, we ensured a reproducible and maintainable workflow. All implemented functionalities performed as expected, achieving test coverage and compliance with best practices in MLOps and CI.