**Sample Coding Standards**

# 1. General Formatting

1. **Indentation**
   * Use 4 spaces per indentation level (no tabs).
   * Keep lines at most 79 characters in length. If a line exceeds 79 characters, wrap arguments or break the expression onto subsequent lines, aligning with the opening delimiter.
2. **Line Breaks and Whitespace**
   * Separate top-level function and class definitions by two blank lines.
   * Separate method definitions inside a class by one blank line.
   * Surround logical code blocks (e.g., loops, conditionals) with a single blank line to improve readability.
   * Do not leave trailing whitespace at the end of lines.
   * Add a single space before and after binary operators (e.g., a + b, x == y), but not inside parentheses, brackets or braces.
3. **Imports**
   * Group imports in the following order, with a blank line between each group:
     1. Standard library imports.
     2. Third-party library imports (e.g., numpy, requests).
     3. Local application/library imports (your own modules).
   * Example:
   * # Standard library
   * import os
   * import sys
   * # Third-party libraries
   * import requests
   * from flask import Flask
   * # Local application imports
   * from myapp.utils import helper\_function
   * Avoid wildcard imports (from module import \*).
   * Keep all imports at the top of the file (after any module docstring, before module globals).

# 2. Naming Conventions

1. **Modules and Packages**
   * Use short, all-lowercase names (e.g., utils.py, data\_processing.py).
   * You may separate words with underscores (\_) if it improves readability.
2. **Variables and Functions**
   * Use lowercase with words separated by underscores (snake\_case).
     + Good: def calculate\_average(numbers):
     + Bad: def calculateAverage(numbers):
3. **Classes**
   * Use CapWords (conventionally called “PascalCase” or “CamelCase”).
     + Good: class DataProcessor:
     + Bad: class data\_processor:
4. **Constants**
   * Use all uppercase letters with underscores separating words.
     + Example: MAX\_RETRIES = 5, DEFAULT\_TIMEOUT = 30
5. **Private/Internal Names**
   * Prefix with a single leading underscore if intended for internal use only.
     + Example: \_helper\_function(), \_config.py

# 3. Docstrings and Comments

1. **Module Docstrings**
   * Every Python file should begin with a brief module-level docstring summarizing its purpose.
   * Example:
   * """
   * data\_processing.py
   * Contains utility functions for loading and preprocessing datasets
   * used throughout the ML pipeline.
   * """
2. **Function/Method Docstrings**
   * Use triple‐quoted strings immediately under the function signature.
   * The first line (summary) should be a short description (imperative mood).
   * Leave a blank line after the summary.
   * Optionally include:
     + **Args:** List and describe each parameter (type, purpose).
     + **Returns:** Describe return value (type, semantics).
     + **Raises:** List any exceptions the function may raise.
   * Example:
   * def add\_numbers(a: int, b: int) -> int:
   * """
   * Add two integers and return the sum.
   * Args:
   * a (int): The first integer.
   * b (int): The second integer.
   * Returns:
   * int: The sum of `a` and `b`.
   * Raises:
   * ValueError: If either `a` or `b` is not an integer.
   * """
   * if not isinstance(a, int) or not isinstance(b, int):
   * raise ValueError("Both inputs must be integers.")
   * return a + b
3. **Inline Comments**
   * Use sparingly to explain complex or non-obvious code.
   * Begin with a capital letter and use complete sentences.
   * Place inline comments on the line above the code to which they refer (not at the end of the line).
   * Example:
   * # Convert UNIX timestamp (seconds) to datetime object
   * timestamp = 1625231612
   * dt\_object = datetime.fromtimestamp(timestamp)

# 4. Error Handling

1. **Exceptions**
   * Use built-in exception types where appropriate (e.g., ValueError, TypeError, KeyError).
   * When raising a custom exception, derive from Exception or a more specific built-in exception type.
     + Example:
     + class AuthenticationError(Exception):
     + """Raised when user authentication fails."""
     + pass
2. **Try/Except Blocks**
   * Catch only the exceptions you expect. Avoid broad except Exception: unless re‐raising or logging.
   * Always log or handle exceptions in a way that does not silently swallow errors.
   * Example:
   * try:
   * result = divide(x, y)
   * except ZeroDivisionError as e:
   * print(f"Error: cannot divide by zero ({e})")
   * result = None

# 5. Testing and Quality

1. **Unit Tests**
   * Place tests in a separate tests/ directory, mirroring your source tree.
   * Use pytest or unittest frameworks.
   * Name test files test\_<module>.py and test functions test\_<function>.
   * Example:
   * tests/
   * ├── test\_utils.py
   * └── test\_data\_processing.py
   * Example test function:
   * def test\_add\_numbers\_valid\_input():
   * assert add\_numbers(2, 3) == 5
2. **Code Coverage**
   * Aim for at least 80% test coverage. Use tools like pytest-cov to measure.
3. **Linting**
   * Use flake8 or pylint to enforce style guidelines.
   * Example flake8 command:
   * flake8 .
   * Configure tox or pre-commit hooks to run linters automatically on each commit.

# 6. Version Control and CI/CD

1. **Git Commit Messages**
   * Use imperative mood (e.g., “Add feature X,” “Fix bug Y”).
   * Keep subject lines under 50 characters.
   * Separate subject from body with a blank line.
   * Provide a detailed body if the change is non-trivial.
2. **Branching Strategy**
   * Follow a “git-flow” or “GitHub Flow” model:
     + **main** (or **master**): Production-ready code only.
     + **develop**: Integration branch for feature development.
     + **feature/\***: Individual feature branches.
     + **hotfix/\***: Quick fixes applied to production.
3. **Continuous Integration**
   * Use GitHub Actions, GitLab CI, or another CI tool to run tests, linters, and safety checks on every push.
   * Example GitHub Actions snippet:
   * name: CI
   * on: [push, pull\_request]
   * jobs:
   * build:
   * runs-on: ubuntu-latest
   * steps:
   * - uses: actions/checkout@v2
   * - name: Set up Python
   * uses: actions/setup-python@v2
   * with:
   * python-version: '3.10'
   * - name: Install dependencies
   * run: pip install -r requirements.txt
   * - name: Run linters
   * run: flake8 .
   * - name: Run tests
   * run: pytest --maxfail=1 --disable-warnings -q