**Coding Standards**

This document outlines the coding standards to ensure consistency, readability, and maintainability.

**1. General Guidelines**

* **Readability**: Code should be easy to understand and maintain. Use descriptive variable names, appropriate comments, and follow a consistent structure.
* **Consistency**: Follow the same structure, indentation, and naming conventions throughout the codebase to maintain uniformity.
* **Version Control**: Always use version control (e.g., Git) for tracking changes, and ensure meaningful commit messages.

**2. Naming Conventions**

|  |  |  |  |
| --- | --- | --- | --- |
| **Element** | **Python Style** | **C++/Java Suggested Style** | **Notes** |
| Variables | snake\_case | camelCase | Choose what fits your language community best |
| Functions/Methods | snake\_case | camelCase | Match the naming convention used in that language ecosystem |
| Classes | PascalCase | PascalCase | Consistent across most languages |
| Constants | UPPER\_CASE | UPPER\_CASE | Helps distinguish fixed values clearly |
| Modules/Files | snake\_case.py | camelCase.java / snake\_case.cpp | Follow what’s idiomatic for the language |
| Directories | snake\_case/ | camelCase/ or snake\_case/ | Maintain consistency within a given project |
| Database Fields | snake\_case | snake\_case | Recommended for SQL databases to ensure compatibility and readability |

**3. Documentation and Comments**

* **Docstrings**: Use docstrings for all functions, methods, and classes. Follow the PEP 257 style for docstrings.

* **Inline Comments**: Use comments to explain complex logic or reasoning. Avoid stating the obvious.

* **Class and Function Descriptions**: Each class and function should include a brief description of its purpose and key functionality.

**4. Error Handling**

* **Try-Except Blocks**: Handle exceptions properly, and always log or raise meaningful error messages. Avoid using generic exceptions like except.

* **Assertions**: Use assertions for situations that should never happen.
* Example: assert len(data) > 0, "Data should not be empty"

*HTTP Error Handling (Future Consideration):*   
 As the project evolves, we should define a standardized set of **HTTP error codes** and **associated error messages** for API responses. This will improve consistency across services and make it easier for clients to handle errors effectively.

**5. Code Structure and Organization**

* **Functions**: Keep functions short and focused on a single responsibility. Avoid functions longer than 40-50 lines.
* **Modular Design**: Split the code into logical modules and files based on functionality. Example: Separate data cleaning from model training and inference.
* **Imports**: Group imports at the top of the file and avoid wildcard imports.

**8. Code Formatting**

* **Indentation**: Use 4 spaces for indentation (no tabs).
* **Line Length**: Keep lines no longer than 79 characters.
* **Whitespace**: Use blank lines to separate functions and classes, and around comments for better readability.