Codebase Guidelines

- **Python Naming Conventions**

  - **Snake Case**: Use snake_case for variable names in Python. For example: learning_rate, training_data, model_parameters.
  - **Abbreviations**: Use consistent and well-documented abbreviations in variable names. For example: num_samples, not n_samp.
  - **Constants**: Use uppercase with underscores for constants. For example: MAX_EPOCHS, DATA_DIR.
  - **Class Names**: Use CamelCase for class names. For example: DataLoader, NeuralNetwork.
  - **Private Variables**: Prefix private variables with a single underscore. For example: _hidden_variable
  - **Function Parameters**: Follow the same naming conventions as for variables.
  - **Pluralization**: Use plural names for collections or lists.
  - **Consistency**: Maintain consistency in naming across the codebase. If a variable represents the same concept, use the same name throughout.
  - **Self-Documenting**: Strive for self-documenting variable names that convey their purpose.

- **Java Naming Conventions**

  - **Camel Case**: Use camelCase for variable names in Java. For example: userName, databaseConnection.
  - **Abbreviations**: If using abbreviations, keep them consistent and well-documented.
  - **Constants**: Use uppercase with underscores for constants, just like in Python. For example: MAX_CONNECTIONS, APP_NAME
  - **Class Names**: Continue to use CamelCase for class names. For example: UserController, DatabaseManager.
  - **Method Parameters**: Apply the same naming conventions as for variables.
  - **Packages and Imports**: Use meaningful packages and import names. Avoid wildcard imports (import com.example.*) to maintain code clarity
  - **Interfaces and Implementations**: Use the "I" prefix for interface names and provide meaningful names for their implementations. For example: UserService (interface) and UserServiceImpl (implementation).
  - **Pluralization**: Use plural names for collections or lists. For example: users, orders.
  - **Consistency**: Follow established naming patterns and conventions within your team.

- **Python Coding Standards**
    - **Indentation**: Use 4 spaces for indentation, as recommended in Python's PEP 8 style guide.
    - **Line Length**: Limit lines to 79 characters for code and 72 characters for docstrings and comments, as suggested by PEP 8.
    - **Imports**: Import modules in a consistent order: standard library modules first, then third-party libraries, and finally your own modules. Use separate lines for each import statement.
    - **Whitespace**: Follow PEP 8 guidelines for whitespace, including one space after commas and operators, and no spaces around parentheses in function calls and definitions.
    - **Docstrings**: Include docstrings for all classes, functions, and modules, following the PEP 257 guidelines. Use triple quotes for multi-line docstrings.
    - **Comments**: Add comments to explain non-obvious code sections, but aim for self-documenting code. Avoid unnecessary or redundant comments.
    - **Naming Conventions**: Adhere to the variable naming conventions discussed earlier, such as snake_case for variables and CamelCase for classes.
    - **Exception Handling**: Use specific exception types rather than generic Exception. Handle exceptions gracefully and provide informative error messages.
    - **File Organization**: Organize code into logical modules and packages. Each module should have a clear purpose and be named appropriately.
    - **Testing**: Encourage the use of unit tests for functions and classes. Follow a consistent naming convention for test files and test functions (e.g., test_function_name).

- **Files and Directory Structure**
    - **Project Root**: Create a project root directory that encapsulates the entire project.
    - **Data Directory**: Use a directory named "data" to store datasets, preferably organized into subdirectories based on dataset sources or categories.
    - **Code Directory**: Create a "code" directory for storing Python scripts and modules.
    - **Models Directory**: Maintain a "models" directory for saving trained machine learning models and related files. Subdirectories may be used for different model versions or experiments.
    - **Docs Directory**: Use a "docs" directory for documentation, including a README.md file that provides an overview of the project and instructions on how to run it.
    - **Utils Directory**: If necessary, create a "utils" directory for utility scripts and helper functions that are used throughout the project.
    - **Experiments Directory**: Consider having an "experiments" directory where you store records of different experiments, including configuration files, logs, and results.
    - **Logs Directory**: Keep a "logs" directory for storing log files generated during training or experimentation.