Final Project Rubric

This is a description of the requirements for the final project and the evaluation rubric that will be used for giving feedback.

The evaluation of the final project focuses more on the structure and quality of the code, not on the performance of your developed model/tool. Thus, for example, for the machine learning models you developed for wind-power forecasting, the accuracy of your model will not be judged.

Contents

Summary of requirements

This list is a summary of what your team shall do to pass the final project. Further details are given in the subsequent subsections.

1. A new user can—after a fresh clone and in a brand-new Python environment—follow the installation instructions in your README.md to successfully install your package and dependencies and then run `main.py` error-free.
    - NOTE: You may replace `main.py` with a Jupyter notebook `main.ipynb` if desired.
2. Your Python package inside `src/` includes at least one class.
3. Your Python package implements the required functions for your project plus 2 extra functions of your choice.
4. The `main.py`/`main.ipynb` clearly demonstrates how the required functions and 2 extra functions are called and executed
5. The `main.py`/`main.ipynb` script generates expected results (e.g., plots) in less than ~10 minutes on a standard laptop.
    - If needed, define a "demo" mode and/or use saved intermediate results.
6. Test coverage on `src/` is 80% or higher.
7. The pylint score on `src/` is 8.0 or higher.
8. The README file shall contain:
    - A brief overview of the package objective.
    - Installation instructions.
    - A description of the package architecture, with at least one diagram of the architecture.
        - **NOTE** that an architecture diagram is not just a diagram of the file structure.
    - A description of the class(es) you have implemented in your package, with clear reference to the file name of the code inside `src`.
    - A description of your chosen git workflow and methodology for team collaboration.
9. Your team prepares and submits feedback forms for 3 other teams by **May 7 at 23:59**.
10. Your team presents your feedback to the other teams in the Round Robin sessions during class on May 8.

Deadlines

- **Repos lock**: Sunday, May 4 at 23:59
- **Feedback forms due**: Wednesday May 7 at 23:59

Details

Pre-defined final projects and required functionality

We provide three pre-defined final projects as follows:

1. **Wind resource assessment based on reanalysis data**
2. **Wind turbine modeling based on Blade Element Momentum (BEM) theory**
3. **Wind power forecasting using machine learning**

Introductions and details on these projects are in the respective links. Your package must fulfill the functional requirements listed in each pre-defined package AND include at least two extra functionalities. If you are doing a custom package, you must complete the functional requirements agreed upon with the instructors plus two extra functionalities.

File structure

Your final project (in the `main` branch) should have the structure and required files as shown in the diagram below.

```
[your_final_project]
├── inputs/
│   ├── data_files_we_provide
│   └── data_files_you_found (optional)
├── outputs/
│   └── data_files_you_generate (no need to push to GitHub)
├── src/
│   └── <package_name>/
│       └── __init__.py
│       └── any other submodules you want (optional)
├── tests/
│   └── python_scripts_you_write_for_tests
├── examples/
│   ├── main.py OR main.ipynb (will run in evaluation)
```

```
│   └── other_example_scripts_you_write (optional)
├── .gitignore
├── LICENSE
├── README.md
├── pyproject.toml
└── any_other_files_you_may_want (optional)
```

Main.py or main.ipynb

Your package should come with a script/notebook that demonstrates the main functionalities of the code. You can use a standard Python script (.py) or a Jupyter notebook (.ipynb) if you would like to intersperse explanatory text. **Note** any reference to "main.py" in this document is also relevant for main.ipynb, if you chose to do a notebook.

To make sure your main.py script fulfills the function requirements, we recommend you copy the listed function requirements for your chosen final project, put them into comments, and organize your code into sections accordingly. For example, each section starts with the comment of the requirement, like `# 1. Load and parse the provided input data`, then followed by the Python code that demonstrates how this required function is executed in your final project.

README.md and architecture diagram

Your target audience is a fellow student who has freshly cloned the repo, has the same terminal/Python skills as you, but is not familiar with the final project of your team. In markdown files, assuming you have a diagram figure (`diagram.jpeg`) put in the `inputs` folder, you can include figure like this:

```
![figure](inputs/diagram.jpeg)
```

Important note: An architecture diagram is NOT a diagram of files, like what is shown in "File structure" section. It could be a UML diagram of your classes/functions, a high-level diagram of how different classes/functions interact (see Week 10 slide 23), or a flowchart of data passing through different functions.

Code must work off fresh clone/environment

A new user should be able to clone your repo, create a new Python environment, install your package, and then run `main.py` successfully. Be sure to include all the necessary data and package dependencies. To test whether your package works in a fresh environment, pull `environment.yml` from Week 13 on the course GitHub repo, then enter the following commands[1]:

```
conda env create -f environment.yml
```

```
conda activate testpack
```

(proceed to pip install as normal)

If you want to delete the environment and try again, use `conda remove --name testpack --all`.

Remember that all necessary data to run `main.py` must be on your repo! Check your `inputs/` folder on github.com.

Test coverage

You shall write tests for your package. We recommend at least 1 test per package functionality, but more is better. The design and complexity of the tests is up to you, but you are welcome to ask instructors for advice. The test coverage on your package of your tests in `tests/` shall be 80% or higher. We evaluate test coverage using `pytest-cov` like this:

```
pytest --cov=src tests/
```

Pylint

Your code should adhere to programming standards. The pylint score for your package shall be 8.0 or higher, as evaluated using pylint:

```
pylint src/
```

You may disable particular codes in a line, block, or module with caution. Our recommendations:

- C0103 (variable not conforming to snake case). This is fine to disable. In scientific programming, it is sometimes better to have short variable names that mean something in a math/physics context.

[1] This creates a new environment with a predefined name and packages as listed in the yml file. See Managing environments — conda 25.3.2.dev29 documentation.

- R0913/R0914/R0917 (refactoring codes). Read up on what the codes mean and why they are telling you to refactor your code. Then, if you decide not to, disable with caution.
- C0301 (line too long). This comes up frequently, but I would recommend adhering to it. 100 characters is plenty.
- Other codes: We prefer that you don't disable these unless you have a well-thought-out reason for doing so.

Git workflow

We recommend you use feature branches that are frequently merged into main via PRs. The person accepting the PR should not be the one who authored it. Commit messages should be precise and clear.

Feedback rubric

Category	Item	Fail/missing	Very Poor	Poor	Okay	Very good	Excellent
Package	"Installability"	Main.py fails with an error after a fresh clone and installation in a new Python environment.			(none – this is yes or no)		Main.py successfully runs after freshly cloning the repo and installing the package in a new environment.
	Classes	No class has been implemented.			One class has been used with clear docstring, and it contains logical attributes and methods.		Multiple classes have been used with clear logical structure and relations (like containment/inheritance), and classes have clear docstrings with them.
	Functionality	The package is missing 1 or more of the required functions OR does not have 2 additional functionalities.			All functions required and two extra functions have been implemented and clearly explained.		All functions required and more than two extra functions have been implemented. Clear and logical structure is used to organize the code with

							clear explanation using comments.
	Structure and understandability	Package has almost no comments, is poorly organized, and/or is extremely difficult to understand. Package folder has extra files and/or executable code in files.			There are some in-line comments, most functions/classes have docstrings. It is possible to understand the code but there is room for improvement.		Package is split into multiple files with clear docstrings indicating the contents of each file. Package is logically organized and easy to understand. There are both in-line comments and module/function docstrings for all classes/functions.
Main script (main.py/ main.ipynb)	Functionality	File is missing 1 or more of the required functions or does not demonstrate 2 additional functionalities.			All functions required and two extra functions are clearly demonstrated.		All functions required and more than two extra functions are demonstrated. Clear and logical structure is used to organize the code with clear explanation using comments.
	Runtime[2]	Code does not complete after ~10 minutes or exits with an error.			Script completes after 6 minutes.		Script runs in less than 2 minutes.
	Robustness	Script has many hard-coded or magic numbers related to the input data files.			Script has some magic numbers.		Script has no magic numbers and makes minimal assumptions about input data files.
	Understandability	Script has almost no comments, is poorly organized, and/or is			Script organization is okay, and there are some in-line comments, but room for improvement.		Script is logically organized and easy to understand by itself. Input parameters to script are

[2] Of course this varies from computer to computer, so consider this a guideline. But you can be clever about how you write the script to make it run fast in a sort of "demo mode".

		extremely difficult to understand.					clearly grouped together. There are both in-line comments and module/function docstrings where applicable.
Tests and standards	Test coverage	Test coverage is below 80%.			Test coverage higher than 85%, test scripts are well organized and easy to understand.		Test coverage higher than 95%, test scripts are well organized and easy to understand.
	Pylint score	Pylint score is below 8.0			Pylint score higher than 9.0		Pylint score higher than 9.5.
README.md	Overview of package	No overview of the package objective.			A brief overview of the package is given.		A clear and well-explained overview of the package is given.
	Installation instructions	No incomplete installation instructions.			Installation instructions are provided but not easy to follow/ missing a few steps.		Process to quickly get started with the code is extremely clear, correct, and easy to follow.
	Description of architecture, including diagram	No description of architecture is given OR there is no diagram of **code architecture** (not file structure!).[3]			Some description and diagram of the architecture are given.		A description of the architecture is given that is clear, understandable, and thorough.
	Description of classes	No description of the implemented class(es) is given.			Some description of the classes is given, but there is room for improvement.		The classes are well explained, including inheritance (if applicable), where they are located in the source code, etc.
	Git workflow/ collaboration description	No explanation of git workflow/collaboration methodology given.			Collaboration methodology is generally explained but lacking detail.		Collaboration methodology is clear and well-explained.

[3] Note as given in the "Details" section that a diagram of the file structure is NOT the same as diagram of code architecture.

Repo structure and git workflow	Folder structure	Folder structure is extremely poorly organized. There are missing files.			Folder structure is okay, but some room for improvement.		Folder structure is well-organized and logical. Files are easy to find.
	Commit history	All commits are generic and made through the GitHub website (e.g., "Updating <filename>").			Some commit messages have been made from GitHub website, but most have been made from the terminal. Commit messages are generally clear.		Almost all commit messages are specific, clear, and have been made using the terminal (e.g., no commit messages "Adding <filename>" or "Updating <filename>".
	Git workflow	Commits are made directly on main by a single team member.			Some commits have been made directly to main.		Commits are made exclusively in feature branches and then merged into main. The commits are evenly distributed amongst the team members.

Feedback before Week 13

The repos will be made public May 4 at 23:59. By **May 7, 23:59**, your team shall prepare and submit feedback forms for 3 other teams.

Link to feedback form

Will be published on Learn.

Peer review task

- Each team reviews 3 other teams.
    - An evaluation Excel will be published on the 46120 repo for your convenience.
- Decide in your team how to split up the evaluation.

    - We suggest that you split by rubric item. I.e., Team Member A evaluates the other teams for Rubric Items #1, #2, #3; Team Member B evaluates Items #4, #5, #6; etc.
- Have a team meeting to fill out and submit the feedback forms together.
    - **REMEMBER**:
        - Check “send me an email receipt of my responses” before submission!”
        - Click “Save my response to edit” after submission in case you made a mistake.

Who you should fill out feedback forms for

Will be published in Week 13 subfolder on 46120 repo.

In class on Week 13

See Week 13 subfolder on 46120 repo. Remember that you are required to be present for the feedback presentations.