CodeCamp Programming Project

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

Summary of requirements

This list is a summary of what your team shall do to pass the CodeCamp project. Further details on each bullet point and possible project extensions are given in the next subsection. Missing bullet/sub-bullets result in a "Fail" grade.

1. Deliver code via a GitHub repo:
    a. The main.py script shall use the 10% TI wind files and ultimately save the requested plot(s) to a PNG file.
    b. The main.py script shall run at most ~10 minutes on a standard student computer.
    c. The main.py script shall run error-free when the repo is freshly cloned and produce expected results.
    d. The remote repo may not include any time-response files ("*_resp.txt") that are created during an execution of main.py.
2. The GitHub repo a README that includes:
    a. A quick-start guide for running the code.
    b. An explanation of how the code works.
    c. An explanation of your team's git workflow and collaboration techniques.
3. The GitHub repo has at least 1 PR with different authors and reviewers.
4. The repo will be locked on **Monday March 9, 23:59**, and all necessary files must be merged into main before then.
5. In your team, before class on Week 6, prepare peer evaluation forms for 3 other teams.
6. In your team, in class on Week 6, present your peer evaluation to other teams.

Details

Code & plot

Overall

Your code shall use a collection of wind time series at different mean wind speeds to ultimately save a plot of the MEAN and STANDARD DEVIATION of the blade and tower deflections VERSUS MEAN WIND SPEED.

- In other words, for each wind speed i, you should simulate the blade/tower response, calculate $\mu_{xb,wsp\ i}$, $\sigma_{xb,wsp\ i}$, etc., and eventually plot them versus mean wind speed.
- The design/format of the plot(s) is up to you. Multiple figures/subplots are fine.

Other notes

- Although you may not push response time series ("*_resp.txt") created by your code to the remote, you may push files with intermediate variables (e.g., statistics) if needed for speed.
- The design of the main script is up to you.
- You are welcome to add more functions to __init__.py.
- If your code is slow, consider designing main.py such that it has a "demo mode" and a "full mode", where "demo mode" is designed to run faster[1]. Demo mode must still create the requested plot.
- You should avoid pushing additional files to your repo; the files initially provided should be sufficient.

README

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 CodeCamp project. The README must contain a quick-start guide for running your code, an explanation of how the code works, and a description of your team's git workflow and collaboration techniques.

Git workflow

Use feature branches that are merged into main via reviewed PRs. The person accepting the PR should not be the one who authored it. Commit messages should be precise and clear.

"Extra credit"

- Make your main.py such that it can process/visualize results from all 3 TIs.
- Ensure that every function and module has a clear and correct docstring.

[1] Example: perhaps you could save some intermediate variable to file, then add/push just that variable but not the full time series. So in "demo mode", the script re-uses the intermediate variable, but in "full mode" it re-simulates everything, including the intermediate variable.

- Enable a linter extension in VS Code (e.g., pylint or flake8) to improve your code quality.

Evaluation rubric

Red boxes indicate pass/fail requirements.

Category	**Item**	**Fail/missing**	**Very Poor**	**Poor**	**Okay**	**Very good**	**Excellent**
Git/GitHub	Commits/commit messages	All commits are generic and made through the GitHub website (e.g., “Updating <filename>”).			Commit messages generally made from terminal. But not always clear from message what was changed in the commit.		Almost all commit messages are specific, clear, and have been made using the terminal (e.g., no commit messages “Adding <filename>” or “Updating <filename>”.
	Pull requests (PRs)	There are no open or closed PRs on the repo.			There are several closed PRs but with minimal description and discussion.		There are several closed PRs with clear descriptions. Each PR has a different author and merger. Generally good discussion on all PRs.
	Git history/branches	Commits are made directly on main by a single team member.			Feature branches are used, but branch history is a little convoluted. Commits are primarily made by a single author.		History of repo is very clean, with feature branches cleanly merged into main. The commits are evenly distributed amongst the team members.
Code and folder structure	Files on remote	Response files created by code are pushed to the remote. Other in-progress			A few unnecessary files on the remote.		No unnecessary files[2] on remote.

[2] E.g., .DS_Store, desktop.ini, pycache folders, .pyc files, etc. Week 3 and 4 code and test files are fine.

		or irrelevant files[2] may be included on remote.					
	Main script task	Script exits with an error, does not create requested plot OR has serious mistake in the methodology.			Script analyses a single TI and generates quality plot(s) that is correct.		Script analyses all 3 TIs and generates plot(s) that is correct and of very good quality.
	Main script runtime[3]	Script does not complete after ~10 minutes.			Script completes after 5 minutes.		Script runs in less than 1 minute.
	main.py 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.
	main.py "understandability"	Script has almost no comments, is poorly organized, and/or is extremely difficult to understand.			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 clearly grouped together. There are both in-line comments and module/function docstrings where applicable.
	__init__.py "understandability"	Functions have little to no comments and are difficult to understand.			There are some in-line comments/ docstrings, but room for improvement in some functions.		Code in functions is easy to understand. There are both in-line comments and module/function docstrings.
Documentation	Git workflow/ collaboration description (README.md)	No explanation of git workflow/collaboration methodology given in the README.			Collaboration methodology is generally explained but lacking detail.		Collaboration methodology is clear and well-explained.
	Quick-start guide (README.md)	No quick-start guide is given in the README.			Quick-start instructions are		Process to quickly get started with the code is

[3] 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", as discussed in an earlier footnote.

					provided but lacking some key details (e.g., working directory).		extremely clear, correct and easy to follow.
	Explanation of how the code works (README.md)	No explanation of how the code works is given in the README.			Explanation is given but not very specific/clear.		The explanation of how the code works is clear, creative, and includes at least one diagram of very good quality. Assumptions of data-file formats are clearly explained.

Feedback before Week 6

After the deadline but before Week 6 class, your team will be randomly assigned to give feedback to 3 other teams. Details TBD.

Feedback in class Week 6

In class Week 6, you will present your code and feedback to each other. Details TBD.