Improving Your Jupyter Notebook GitHub Portfolio for Junior Data Scientists

If you're a student or seeking entry-level data jobs and you have a Jupyter notebook-intensive GitHub portfolio, *this is for you.*

If you have or plan to create repos with or without Jupyter notebooks, *this may also be for you.*

If you are like, "unit tests? obviously I already have unit tests!", *this is definitely not for you, but that's awesome!*

Your GitHub portfolio can help you:

1. **Convey in an easy-to-understand way what you did, how you did it, and why you did it**: You want to show that you understand your work, that your methods make sense for your problem, and that you can communicate well.
2. **Make it easy for people to run your code and add to it**: Replication is really important. You want to make it easy for people to run your code (even if they probably won't).
3. **Show that you understand industry norms and best practices**: This is an opportunity to show that you've looked around and seen what people in the field you want to join are doing, and spent a little time figuring out how to do those things, too.

These things overlap, but not entirely. And what it means to be doing each of these and how much to do depends on your background, existing skills, and the kinds of jobs you're targeting. The purpose of this is to give you some guidance to get the most out of whatever time you want to put in.

Basic and Quick

These relatively quick tasks are the low-hanging fruit if your repo is meant to be looked at:

- **Organization/package management: put all of your install statements and imports at the top of your notebook**. This makes your code more readable. It also makes it clear what dependencies your code has, and makes it easy for someone else to install them and run your code.
- **Explain at the top of the notebook what you're doing**. This doesn't have to be long, but explain your approach, goals, and processes a bit. What can they expect to see if they keep reading? Use markdown, use lists, don't make it too long.

- **Remove irrelevant code and figures to focus on your narrative.** When you're doing data analysis, you often try a bunch of things that don't end up in your final version. Keep it focused on the parts that matter for the story you're trying to tell. You're not trying to prove you're familiar with a lot of different methods or that you wrote a lot of code, you're just showing what wound up being important. This also gives you less to document and reorganize.
- **Make sure your code runs.** Don't have error messages/code that broke when you ran it in your most recent commit.

Important but Potentially Less Quick

If you want to spend more time, I think these are where you should focus.

- **Organize your code in functions. Add docstrings explaining your functions**. This is a step towards making your code more modular and easier to understand. Instead of having a long script with everything happening in order, you can break it up into logical chunks that can be reused and tested independently. Docstrings are a way to explain what each function does, what parameters it takes, and what it returns.
- **Write documentation to explain inputs, outputs, and usage**. In addition to your previous documentation, briefly explain each of your inputs (API keys, data) and your outputs (graphs/tables), and explain how to run your code.
- **Automate all of the pieces of your work**: For instance, if your analysis involves fetching data from an API or scraping a website, write a script to do that automatically. This makes it easier for others to reproduce your work, and also makes it easier for you to update your analysis if the data changes. You could also include the specific version numbers of the packages you're using and use relative file paths. Realistically, this is more about conveying you understand best practices than it is about making your code easier to run, since probably no one will actually run your code.

Next Steps – If You Want To

If you want to keep going, you can pull most of your stuff out of your notebook and use the full repo structure:

- Put your documentation into a readme.
- Put your functions into one or more .py files.
- Use a requirements.txt file for your dependencies.
- If you still want to show figures and walk readers through what you did, import your functions into your notebook from your .py and call them from there.

Do This By Committing and Pushing

For all of the changes you make, it's good to get into the habit of doing this not by modifying the code or documentation from the GitHub GUI but instead by making the changes locally and then committing and pushing them to your repo. This is a cleaner, better process.

Use LLMs To Help You

I very much think you should use an LLM to help you with this stuff.

LLMs can help you by:

- Writing docstrings
- Giving you directions with git/troubleshooting your errors
- Helping you write clearer documentation
- Helping you organize your code into functions - specifically, functions that each do one thing. You do need to get specific when you ask it to structure your code about this.

You should anticipate all of these being a back-and-forth where you ask, see its responses, and iterate/modify its outputs if they don't work for you.

Conclusion

You don't have to do any of these things.

First, I can't make you.

Second, this will rarely make the difference between you having no job options vs. some job options.

But nonetheless, I think you should consider at least some of these. Here's the thing: you'll eventually need to learn some or all of this anyway. This is standard data science workflow, but often overlooked in training programs. I'm not going to tell you how much time to invest in this, but if you are investing time, my hope is that this will give you some ideas about where to focus.