Why I don’t use Jupyter Notebooks anymore

Jupyter Notebooks are an awesome way to explain code to people, but for the past year I've tried to utilize them as a convenient way of prototyping data analyses in a layered fashion. My aim was to not having to rerun expensive calculations everytime some later part of my code raised an exception. But in the end, these notebooks hampered my progress and I've since switched back to running plain Python scripts from the terminal. Here's why I don't use notebooks anymore.

Published on **Monday, February 7th, 2022** by Hendrik | **5 min reading time**

In the past year, my go-to tool for conducting my initial data analyses for my PhD project were Jupyter Notebooks. These are modeled after RMarkdown files and allow to interleave descriptions using Markdown syntax with runnable code in Python. The beauty of these files lies specifically in the fact that they offer you a nice way of describing your code not just with comments, but with formatted text which allows you to emphasize words, create lists, and other goodies.

However, I’ve recently hit a bottleneck. While notebooks are very convenient for explaining code, they have a few drawbacks when you use them for fast prototyping. For the uninitiated: prototyping describes a process during which you change code fast and add new features at high speeds. That can become a problem when using notebooks. Right now I am prototyping a topic model (Latent Dirichlet Allocation) and therefore have to conduct frequent hyperparameter sweeps, exchange metrics, and optimize the code to achieve faster run times. Notebooks, however, are very good at slowing me down, due to several reasons.

The Cursed Blessing of IPython

Technically, Jupyter Notebooks build upon the IPython kernel. That is basically a daemonized Python shell that persists any global variable to declare. Whenever you run one of the cells in the notebook, the code gets send to the process, and any results produced are displayed as the cell’s outputs. This means that, unlike regular Python scripts, an IPython kernel is *stateful*, and that means that any change you make to a variable is persisted until you re-run the code that created the variable in the first place or restart the kernel completely.

This means when you’re frequently rewriting code, you always have to remember which cells to run before the one you’re currently working on, lest you will run into errors caused by some variable not having the contents you expect it to have.

On comparison, if you work with Python files directly, you will always start with a fresh state. This does mean that you have to check that the full code runs perfectly fine before letting the program loose on your full dataset (because you don’t want a `ValueError` to occur after some part of your script has pre-processed the data for two hours straight). But this was a much cleaner approach since you do not have to remember where you declared which variable to contain some specific value. Additionally, by not having to care about cells, it becomes easier to split up the code into functions which makes the overall code look more clean. It is much harder to judge what pieces of code belong to one cell, and what should be placed in their own, respective cells.

PyCache and Notebooks

Another thing I found is that notebooks don’t play well with cached Python code, which is stored in a folder called `__pycache__`. While coding, you will frequently find yourself wanting to factor out certain utility functions since you need them in multiple places and they don’t form the core of the code you’re trying to run. One of Python’s optimization strategies is to cache modules so that the code runs faster the second time you call it.

Each time a Python process starts up, it will compare the modification times of the cached files versus the modification times of the actual source code. If the source code is newer, it will re-cache the code again. However, an IPython kernel is persistent, and therefore, even if you run the cell that imports your utility functions again, it will not perform that check, but simply load the cached values again. This means, whenever you edit this external code, you have to restart the kernel. Any potential benefit of not having to rerun all cells during data-processing evaporates.

To Comment or Not to Comment

Finally, let us talk about the pros and cons of having the ability to write Markdown code. Do you really need to write Markdown code all the way? While utilizing notebooks for prototyping, I realized that most text I wrote would’ve also fit into comments and still be readable. Whenever I wrote longer texts, I realized that I could’ve just put these into their own notes in Zettlr rather than having them unconnected in some arbitrary notebook where I would’ve never found them again.

So even the ability to write Markdown descriptions was ultimately not a benefit and more of a gimmick.

When to use Notebooks

But when are notebooks actually useful? I am very certain that Jupyter Notebooks are amazing for two use-cases. The first is to create labs for university courses: Provide descriptions of the task using Markdown cells, add some skeletons for functions in Python cells, and then have the students complete the functions. This way you give the students guidance while also providing a convenient way to solve the tasks in the same spot as where the task description is. This is what our instructor in the Natural Language Processing course in early 2021 did, and it felt amazing. I still look back to this course as one of the most useful courses in my PhD so far and one of the reasons is that I learned so much using this method.

The other way to really let notebooks shine is when you already know what some code does and you want to explain it to other people. Comments in the code are pretty fine, but especially if you’re dealing with an audience that does not use code for the sake of using code, but rather to solve a task, then adding descriptions of what the mathematical correlation of a certain algorithm looks like is better written in Markdown. Especially for mathematical formulas it can be amazing to have the ability to drop in MathTeX equations intermittently.

In conclusion, Jupyter Notebooks are the most beautiful way to explain code to others, but for actual development of functional code in the first place, they are absolutely useless and hamper your progress in many ways. Ever since I switched to running scripts directly from the command line, my work has been much faster.

Suggested Citation

Erz, Hendrik (2022). “Why I don’t use Jupyter Notebooks anymore”. *hendrik-erz.de*, 7 Feb 2022, https://www.hendrik-erz.de/post/why-i-dont-use-jupyter-notebooks-anymore.
