MLOps Blog: Should You Use Jupyter Notebooks in Production?

By: Prabhat Kumar Sahu

Dated: April 19, 2023

From: Neptune.ai in blog: MLOps (https://neptune.ai/blog)

In the past couple of years, Notebooks have become a popular tool in fields like data science and machine learning, scientific research, genomics, and more.

Jupyter Notebooks have been around for quite some time now. They're used a lot in machine learning, mainly for experimentation and visualization. However, recently notebooks have been making progress into production environments.

In this article, we're going to discuss Jupyter Notebooks and the use of Notebooks in production environments.

What does Production mean?

'Production' means different things to different people. For us, it means the code that's being used or consumed by some end-user. Many organizations use notebooks in production, especially when they come to a point where they want to share the functionality of a notebook with other people or non-technical users.

Usually, we see two types of notebooks in production:

- Static reports notebooks
- Voilà applications

Static reports notebooks are made with the help of Papermill. It's a tool for parameterizing and executing notebooks programmatically. Also, it transforms the notebook into a data workflow tool, executing each cell linearly without having to open the notebook interface. It generates a report for each execution and collects metrics across the notebooks.

It gets easier to use Jupyter Notebooks as templates to generate reports by automating the process of notebook execution.

Voila is an open-source fully-fledged Python dashboarding framework. Users can convert their notebooks into a standalone interactive web dashboard application. With the help of Voila, you can share your work as a web application and deploy it on any cloud service, so that people can see and use the dashboard.

Production notebooks also mean running the notebook in some automated fashion. They are no longer scratch pads for quickly iterating or experimenting. You're using these tools to bring some libraries together with some data and present it all together with text and markdown to get some end results or reports.

The need for production

Notebooks have been great for prototyping, but in recent years we've been seeing different business problems and technical challenges. One prominent challenge is the demand for productionizing analyses and experiments, and another is the rapid adoption of the cloud.

The need for production, creating data products has grown through the years.

The Jupyter notebook was meant for prototyping and exploration, not for production. But over the years, the ecosystem has grown. We now have a different set of tools, JupyterLab, plugins, new kernels, and many others.

These changes came over the years thanks to:

- **Experiments on the cloud** – many people started preferring cloud for large computations and bigger datasets.
- **Developer workflow** – many machine learning teams started adopting software engineering practices like version control, git-flow, containerization, and more.
- **Analysis to production** – if the analysis code is written keeping the best practices then it should be easily reused for production.

Pros and cons of using notebooks in production

Pros

- You can make standalone applications and dashboards using voilà and can serve the end-user.
- Jupyter notebooks can be scheduled as jobs over the cloud.
- You can make *templatized* notebooks and execute them via a papermill.

Cons

- No proper code versioning.
- Reproducibility can be an issue because of state-dependent execution.
- Unit testing is difficult.
- Dependencies management is not proper.
- Caching is an issue.
- No CI/CD.

The cons aren't a huge limitation, as there are many ways to deal with them. Let's talk about all those problems and their solutions.

Problems with notebooks in production

Jupyter is markdown-savvy. It uses base64 for its image serialization, and we get to use its functionality like code execution, all through a web interface.

But it comes with its own problems:

- Version control and file size
- Modularity and code reuse
- Hidden state
- Testing/debugging

Version control and file size

Jupyter notebooks with the extension of .ipynb containing Python code aren't Python files. They're basically large JSON objects. They're not very suitable for a Git-like workflow. If we commit the notebook after changes, the diff becomes really big, making it difficult to review them or to merge into the main branch. This makes it challenging to use them in teams.

If the notebooks include images and a lot of plots, then the file size increases considerably.

Solutions:

- nbdime – it's a tool that helps to generate different views of notebooks.
- nbstripout– it strips the output from the notebooks which can help us for easier parsing and comparison.

Modularity and code reuse

Modularity is one of the most crucial concepts in creating robust applications.

Code modularity is important. But with notebooks, we put most of our codes into cells. The good way to reuse the code in Python is through functions and classes. Also, Notebooks don't allow proper packaging.

Solutions:

We can use the Do not Repeat Yourself (DRY) principle. You should generalize and consolidate your code as much as possible. Functions, for example, should only have one job, abstracting your logic without over-engineering. However, you should keep an eye out for creating too many modules.

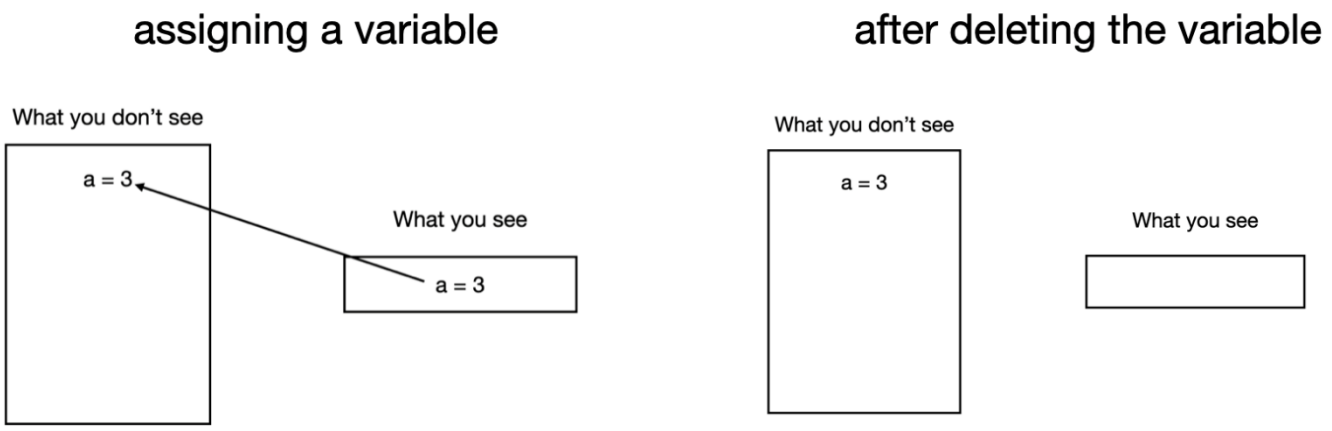
Hidden state

Jupyter notebooks are an interface for writing and experimenting with code. But Jupyter has a weak spot. What you see is not always what you get.

Many people say that notebooks are good for reproducibility. And it's true when you're running the code in a linear order from start to finish. But we can also run cells in a non-linear order.

Jupyter runs code in the order that you execute it. And it remembers assignments whether or not they're still there. The image below illustrates this.

The smaller box on the left represents the hidden state. This is code that you have already executed. In the next frame, we've deleted the variable, but it remains loaded in memory. This can lead to really weird situations.



Jupyter notebooks: What you see is not always what you get

Solutions:

- If your code is behaving strangely, a good first step is to restart the kernel.
- Write code with modularity and with linear order, it's good for production.

Testing/debugging

Notebooks are hard to debug and test even if they are linear. That's because of two reasons. Firstly, when you're working on a project and the notebook grows large enough, there are too many things to keep track of (variables, functions, etc) and it gets difficult to figure out the execution flow.

The second reason is that it's difficult to unit test because we can't directly import functions that are defined in the notebook into a testing module. There are ways to do it, but it's not straightforward.

Solutions:

- testbook – a unit testing framework that will help us test code inside the notebook.
- nbval, pytest-notebook – nbval is a great library for reproducible notebooks. It compares the stored outputs of a notebook with the outputs generated by a notebook.

In the case of production notebooks, we want to encourage best practices and we want to avoid a lot of pitfalls and anti-patterns that have been mentioned in this section.

Embracing notebooks in production

Notebooks are good when you're just playing around and experimenting. But, as soon as you need to share your code or deploy a machine learning system into production, notebooks become quite challenging to work with.

We want a production notebook that's testable in some form, deployable in some way, and extensible.

Also, these notebooks are linearly executed notebooks. When we're running the notebook in an automated way, we're executing the notebook top-to-bottom one time.

Here are the things to think about before making it into the production workflow.

Data: how do you get the data?

- Where are you getting your data from? Are you using live data or are you using an extract? Or are you connecting to a database and getting the data? Or pulling from an S3 bucket or another data store?
- Is the dataset already prepared, or do you need to prepare it?

Code: how is your code organized?

- In terms of how your code is organized, there are a lot of options. Like do you keep things in notebooks or do you export to a standard Python script?
- Do you break out your code into modules or packages? Or you just keep it in one place.
- Are you calling Python a Jupyter kernel, or converting it into a standard Python script?
- Is all your code in one file? Are you using different functions? Classes? Are you making packages?

Where does the code run?

- Is it running on your personal laptop or on a server? Are you deploying the notebook with a service like Lambda?
- How are you handling your code dependencies? Are you creating environments? Using Docker containers?

These things matter. For example, just because you can run something on your laptop doesn't mean you should. You should always consider the long-term and plan for sustainability.

Also, if you're more on the software development side of things, thinking about containerization, or even just handling your different environments and dependencies, that is a big step in going into production.

Outputs: where does your output go?

- Once you're done running your code, where does the output go? Do you want it in a notebook? Can it be in a notebook? Or do you want to export it in a different file? Maybe you want to export it to a different system?

- How you make that determination really affects what the workflow should be for your production code.

Suggested production workflow

Here are some suggestions of patterns, they may vary depending on your requirements.

Data preparation

The first thing to think about is how you handle your data preparation work. You can do a lot of data prep in Python, or any other language of your choice, but when you're working with a lot of data in a database or any other, it doesn't make sense to pull it all in its raw form. Instead, you can run an ETL step at the beginning of the production cycle, ultimately having the grunt work done on the database side. Or you can even have a totally separate pipeline of retrieving the data.

Version Control and CI

You may also want to think about version control and continuous integration. Because we're looking at a batch workflow on production, where you have some data, you do something to the data and then you have your result. Then maybe you repeat the process later on. It's a pretty straightforward process, a lot of things you build in production aren't nearly that simple. Particularly if you're thinking about things like building out APIs or providing streaming services. In these cases, you have to split the deliverables. Suppose you're training a model and you want to do prediction on demand. It gets a little trickier. You might need to jump into development and production very frequently.

Often splitting the work between development and production isn't nearly easy and clean. You may need to go back and forth between the two of them. Tools like Jenkins or Travis help in these processes. Then it becomes really important to start building tests in your code. These are some of the key parts of the production workflow.

Containers and environments

Containers and environments also become really important when you go to production. Say you're running the script from the command line but maybe you want to package that up as a docker container first or define an anaconda environment. Just make sure the production workflow is going to have all the same dependencies and it's going to run much more reliably into the future, particularly if this is something you're going to be using for a long time.

If you plan to automate, you might want to package up your dependencies with your scripts.

Services and deployment

There are different ways you might deploy your notebooks. If you're going to be building things for APIs or running them on serverless architectures like AWS lambda, they have their own set of requirements. This means you need to think about a bit more complexity as you go into the production process.

So, how do you make these choices?

What to consider when choosing your production workflow

1. **Reliability**: Jupyter notebooks are more stable than they were years ago. Still, it can break your code if you're not setting up the notebook in production properly. For example, if you're not spinning up your notebook server uniquely for every project, it can break the workflow of the code.
2. **Accessibility**: There are many apps and tools for easy notebook sharing. But there's still more flexibility in exporting outputs or results to text files, databases, spreadsheets, or saved images.
3. **Reusability**: Various packages and modules are there to make our life easier. Copy-pasted code is hard to reproduce and difficult to maintain.
4. **Interpretability**: Notebooks make it easier to put documentation and results alongside the code. It can easily help us if we're looking at it in the future, we'll know what the code does and that's valuable for production.
5. **Flexibility**: You can almost do most of the data science work in them but they're not the best tool for every job.
6. **Agility**: We all love notebooks because they make data science easier and faster. And getting something new into production fast is usually a big deal.

Future of notebooks in production

- **Notebooks as Voilà applications:** Notebooks are increasingly becoming applications. Notebook is the point, it's not just the way you get to whatever product you're building. Instead, the notebook may be the product itself. As notebooks grow from dev environments to shareable applications, they become an end product themselves.
- **Data Science Platforms:** There are many data science platforms, like Anaconda. They make notebooks a priority of their toolkit, helping and simplifying deployment.
- **The rise of containers:** Containers continue to expand their place in the data science ecosystem, and so notebooks are becoming a more practical tool for production deployments, even for serverless architectures like Lambda.
- **New Jupyter capabilities:** JupyterLab is really blurring the lines between production apps and many development tools even further – for example, by substituting extensions for traditional modules and packages.

Final thoughts

The use of notebooks in production is always a debatable topic. Many people believe and take it as undeniable truth saying Jupyter notebooks are just for experimenting and prototyping, but I don't completely agree with them.

Notebooks are great tools for working with data, especially when leveraging open-source tools like papermill, airflow, or nbdev. Jupyter allows us to reliably execute notebooks in the production system.

References

- https://netflixtechblog.com/notebook-innovation-591ee3221233
- https://www.martinfowler.com/articles/productize-data-sci-notebooks.html
- https://github.com/nteract/papermill

About the Author

Prabhat Kumar Sahu is a Machine Learning Engineer You can follow him on:

- https://mobile.twitter.com/thecaffeinedev
- https://www.linkedin.com/in/prabhat-kumar-sahu-b9a53674/?originalSubdomain=in
- https://www.iprabhat.dev