Jupyter outside of data science

Tomislav Maričević

Jupyter and notebook software – Ipython & Jupyter

• Ipython Notebook == Jupyter Notebook

Ċ jupyter	Untitled29 Last Checkpoint: 3 minutes ago (unsaved changes)		Logout
File Edit	View Insert Cell Kernel Widgets Help	Trusted	Django Shell-Plus O
8 + % (
In [4]:	<pre>import this The Zen of Python, by Tim Peters Beautiful is better than ugly. Explicit is better than complex. Complex is better than complex. Complex is better than nested. Sparse is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats gurity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one and preferably only oneobvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than *right* now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea let's do more of those!</pre>		
TH []:			
In []:			

Jupyter and notebook software – JupyterLab



Jupyter and notebook software – JupyterHub

- JupyterLab is single user
 - Multiple people can use it at the same time, but they all share the filesystem and configuration

• JupyterHub spawns new JupyterLab instances for each user

Local client, remote kernel

- In-browser IDE is not optimal UX
- Pycharm & VS Code support remote kernels

Nuanced discussion

<u>"I don't like notebooks" - Joel Grus, JupyterCon 2018</u>

VS

"I Like Notebooks" - Jeremy Howard, almost on JupyterCon 2020

The "never use notebooks" crowd

- Hidden state
 - Need to run cells exactly once, in order, with exact 3rd party dependencies
- Less rigor
- Hard to test
- Promotes bad habits
- Worse UI than your own editor
 - Autocomplete, linters, autoformatting, etc.

The "use notebooks for everything" crowd

- Tests, docs and code are co-located
- Exploratory coding in a live coding environment
- Diffs can show changes in output next to changes in code
- <u>Nbdev</u> solves a lot of the complaints

Our setup

- Python
- Django
- PostgreSQL
- AWS
- JupyterLab

JupyterLab setup

- Dedicated EC2 instance running JupyterLab
- CodeDeploy for delivering new code to server
- Multi-user environment via JupyterLab
- django-extensions kernel for Django integration
 - Use production settings for access to production environment

Issues

- No elegant way to monitor progress if network connection is lost between kernel and frontend
 - Workaround: Save progress to file
- No way to automatically shut down old kernels
 - Workaround: JupyterHub
- AWS doesn't have a nice general-purpose managed solution
 - SageMaker is very data science oriented
- Not very git-friendly
 - Workaround: use Github, or use tools to convert .ipynb to .py files
- It's a shared environment with shared resources
 - Loading 16 GB .csv into memory ruins the party for everyone

Things I learned the hard way:

- Add cell execution timestamps
 - Notebooks save complete state, it's nice to know from how far back the state is
- Add docstrings to your functions
 - You're not going to remember which problem you were solving in 6 months
- Shut down your unused kernels
 - They accumulate and use up resources even when idle

Our use case #1: Operations

- Running one-off ops on production infrastructure
- Notebooks are an alternative to running scripts in shell on production servers
- Easier to develop than writing script locally and then uploading or C/P to production shell

Our use case #2: Data Exploration

- Related to data science, but less formal
- E.g. investigate why some email campaign didn't do well
- Practically impossible to do in the shell need the visualization capabilities

Thank you!



github.com/tmarice

