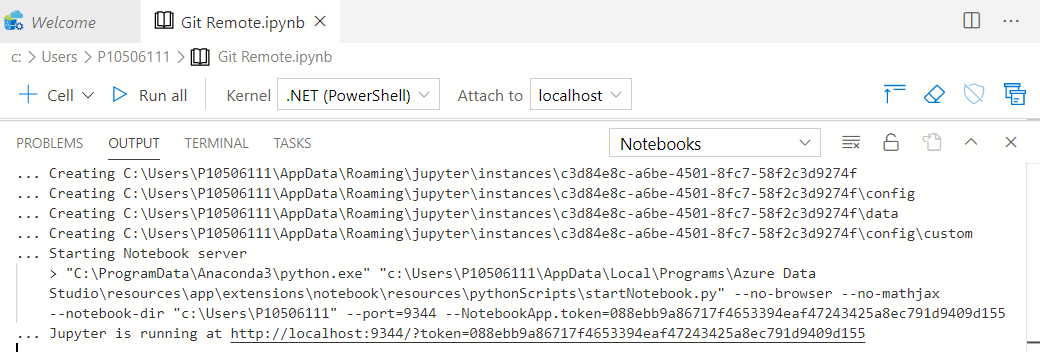


In Part 1 I covered installing a version of Python with the Jupyter server and adding extra Kernels – in particular, .NET Interactive for PowerShell support. After a quick exchange with some friends on twitter it seems I could have explained something better in part 2 about the different ways .NET Interactive works with Azure Data Studio, and visual studio code, which is what I’m going to try to cover here.

When the Jupyter server can work in two different ways. My set up of default anaconda installation + .net Interactive from part 1 gives the ability to start the server and access it through a web browser to work with notebooks stored as .ipynb files.

If Azure data studio (or the Python extension in VS Code) is told to open a .ipynb file it starts python to run exactly the same web server specifying the port and access token and then makes API calls to use the server, without a web front end. You can see this in Azure data studio below, and the link at the bottom will open in a browser allowing two kernels on one server to see the same set of files.



Getting the Process object for the current powerShell process with Get-process -id $pid and then checking it’s parent process, than that process’s parent and so shows that   
Azuredatastudio -runs – cmd – to launch – python –which runs Jupyter starts a kernel -dotnet – which hosts - dotnet-interactive

So Azure data studio is using all Python Jupyter bits except rendering as HTML/Javascript because it does its own rendering.

I said that the .NET interactive extension for VS code uses its own installation – when Python launches the kernel it runs

<home>\.dotnet\tools\dotnet-interactive.exe Jupyter <parameters>

Things are different in VS code

Code runs dotnet tool run <parameters> which runs dotnet <home>\.nuget\…\ Microsoft.DotNet.Interactive.App.dll <parameters>

No python, no Jupyter web server, It goes directly to the Kernel loads files from the .nuget directory instead of the .net one.

I also referred to the difference between Jupyter-generated output and Output from the DotNet Interactive extension for VS code (I’m tempted to start abbreviating that to DNIE pronounced like deny). Jupyter creates JSON inside the ipnyb file like this

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [  
 "Line 1,  
 "Line 2",  
 "Line 3"  
 ]

}

]

It’s also valid to have "text": "A long string with \n for newlines"

*At the moment* the extension for vs code writes like this:

"outputs": [

{ "output\_type": "execute\_result",

"data": { "text/plain": "Line 1" }

},

{ "output\_type": "execute\_result",

"data": { "text/plain": "Line 2" }

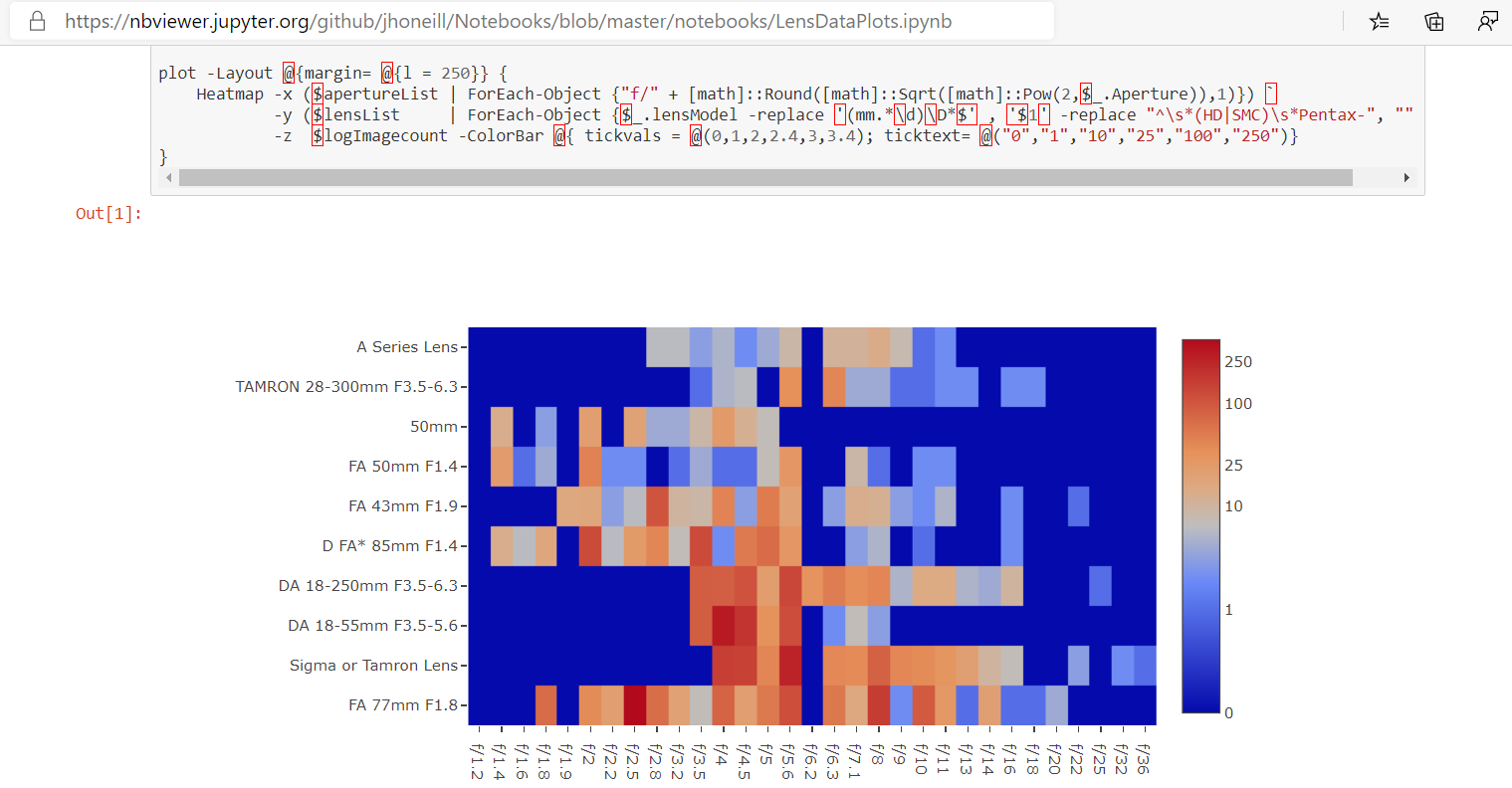
}

]

Both get read everywhere but often the Jupyter format reads more nicely. I would stress *at the moment,* because, to repeat what I said in part 2, this is a preview and things can change.

Being JSON internally, lots of things can read and/or write ipynb format. The book has a metadata block which says what language it uses, and cells block, each cell has a type (code or markdown), source, and – for code cells - outputs. My friend Doug Finke has a PowerShell module to help with that. And there are other useful tools:

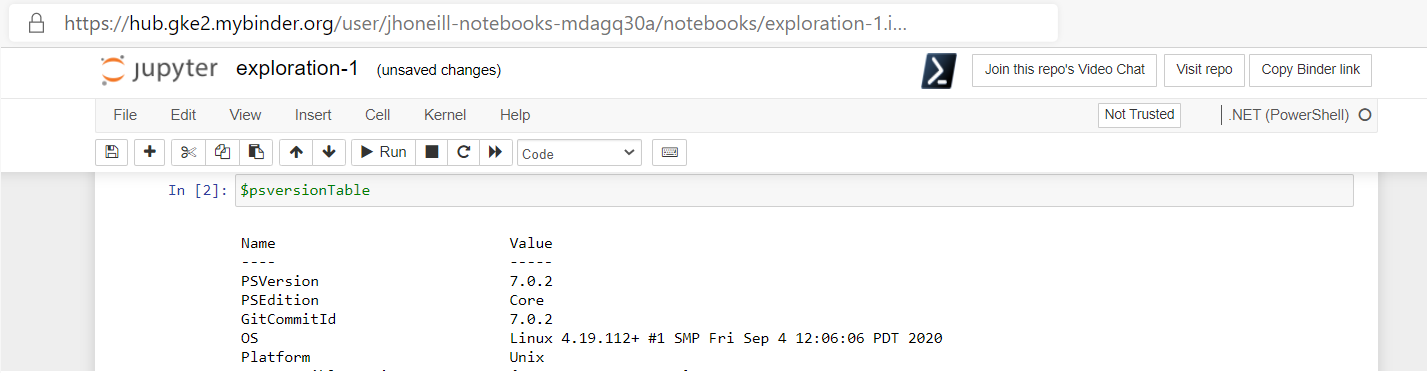
* **GitHub** will display the contents of a .ipynb. file in read-only form -skipping any JavaScript.
* [nbviewer](https://nbviewer.jupyter.org/) seems to have the same templates but doesn’t filter out the Javascript. If I run something on my computer, my results are embedded in the Notebook, and someone I share them with might not have access to my data so being able to show a chart is a consideration



Incidentally, the screenshot above shows a notebook from my git hub repo <https://github.com/jhoneill/Notebooks> - someone asked a question a photography forum about what which led me to do a heat map of what I use from the lightroom data <https://jhoneill.github.io/powershell/photography/2012/08/09/Lightroom-data.html> I’ve been messing with for years

* Binder goes even further. It gives you a docker container running the Jupiter server. It will copy information from a GitHub repo and launch the server. By default it doesn’t have the additional Kernels from .NET interactive, but you can provide a dockerfile to configure the container – I got mine , and the associated nuget file from the dotnet interactive repo. <https://github.com/dotnet/interactive/tree/main/samples/my%20binder>

Binder can take a few moments create and launch the container but it’s live, you can the code and re-run the container doesn’t have any access to your machine (so files you need must be in the repo), but it is possible to install modules, or connect to an Azure cloud shell or run other commands which access resources from the internet.



In case you weren’t aware that .NET interactive is cross platform, the screen shot above show the docker containers are running it on via Python on Linux. Binder disposes of the container then you leave or if you’re inactive for too long.

I can see running shared notebooks with a service like binder, as a way to do “show your working” configuration or analysis of on-line services – and I’d pop over to print preview in Jupyter to generate results to show what had been done.

