

Python_Notebook_Review

January 22, 2023

1 FDS II Jupyter Notebook Review

1.1 Introduction

One of the things that can confuse newcomers that wish to dabble in Python is the huge variety of options – not options for Python *per se*, but options for how to interact with Python.

If you use R, then you almost certainly use RStudio as an IDE (Integrated Development Environment); it's pretty much the only game in town. If you use MATLAB, then you use the MATLAB IDE, full stop.

With Python, by contrast, there are a dizzying array of options; you can program in Python using

- Python in the terminal,
- iPython in the terminal,
- IDLE,
- Thonny,
- VS Code,
- PyCharm,
- Spyder,
- etcetera, etcetera, etcetera...

Which one should I choose?!?!

In this class, we'll use Jupyter notebooks (though we will occasionally give you a glimpse of other options).

1.2 Why Jupyter Notebooks?

Jupyter notebooks are a great option for a class like this for a couple of main reasons. First, the nature of Jupyter notebooks encourage short chunks of code (in code cells) interspersed with explanatory text (markdown cells). Moreover, when run, the code cells can produce output (such as data graphics) directly in the notebook. This makes Jupyter notebooks great for both teaching and learning coding, because both the code and the results can be presented and digested in small bites.

Second, because

- the markdown text and the code output are interweaved in a single document, and
- this document can be output in a number of formats including pdf and html,

Jupyter Notebooks have become a defacto standard tool for data scientists. For small to intermediate sized projects, everything can be done a notebook or a small number of notebooks. For larger

projects involving lots of complicated code, the projects itself can be done in something like VS Code or PyCharm, and results and summary can still be presented in Jupyter Notebooks simply by reusing the plotting code from the main project.

More minor reasons include the simple easy-to-master interface and ready availability of help due to its ubiquity.

For these reasons, Jupyter notebooks provide us with an ideal jumping in point for learning Python and some fundamentals of data science!

1.3 Notebook Anatomy

Notebooks are composed of two things:

- code cells - where you write and execute your code
- markdown cells - where you write formatted text

1.3.1 Markdown cells

Markdown cells hold text that can take advantage of the markdown formatting syntax. Everything so far in this notebook has been made of markdown cells that have “run”. Just like running code generates output (or an error – haha), running a markdown cell converts the markdown formatting syntax into the desired formatted text.

Markdown is also the usual way to make formatted documents on GitHub. And, if you took PSY420 here, I’ll the lab materials were done in markdown.

Some common things we use markdown formatting for are

1.3.2 Headers

and

subheaders

of various sizes

We also use markdown for

- lists
- that have
- bullets

and

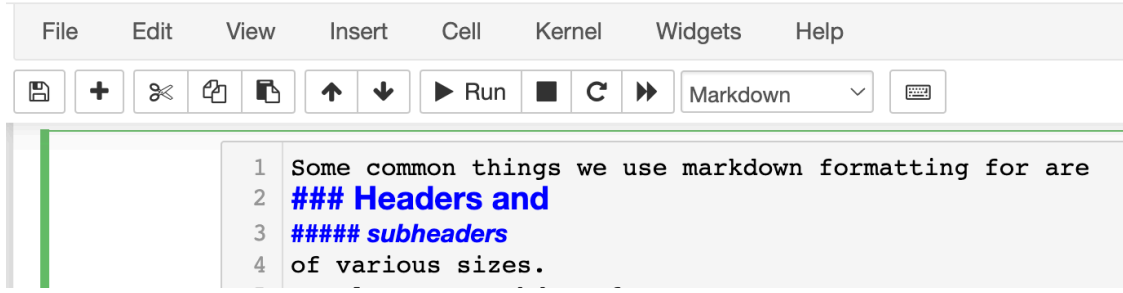
1. numbered
2. lists
3. too

You can also specify *italic*, **bold**, or ***bold italic*** formatting.

To make even fancier notebooks, you can add

links to, for example [the class Canvas page](#)

or images, like a screenshot of this notebook:



So, as you can see, markdown allows us to have a pretty, formatted document that, because it's in a Jupyter notebook, allows our nice document to also contain Python code and its output in *code cells*.

1.3.3 Code cells

Code cells are the heart and soul of Jupyter notebooks. Below is a simple code cell to greet you:

```
[1]: print('hello')
```

hello

Notice that you can identify a code cell because it has a Python prompt to the left just like the prompt you would see running iPython in a terminal:

```
((base) $ ipython                                     22:10 ]
Python 3.8.13 (default, Mar 28 2022, 06:16:26)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: _
```

Code cells can be as long as you want. If they wanted to, a graduate student could put the entirety of the code to analyze the data from their entire dissertation in one code cell, and it would run just fine. It would defeat the purpose of code cells – the whole point is to break up your code into smaller logical chunks interspersed with text that, in effect, becomes a narration of your analysis. This makes it easier on everybody – your dissertation advisor, your boss, whomever – including, importantly, future-you!

Each code cell is **not** a separate Python program! All the cells in a notebook make one single program. So any given cell is “aware” of anything that’s happened in cells that have been already run. For example, let’s assign a variable in one cell, and check the value and type in another:

```
[1]: ans = 42
```

Now we’ll check its value in a separate cell:

```
[2]: ans
```

[2]: 42

And now we can check its type in yet another cell (what will that be?):

```
[3]: type(ans)
```

[3]: int

Hope you guessed “int” :)

1.4 Working With Notebooks

First, start a notebook so you can play along! To start a new notebook, we fire up a terminal window first, navigate to the folder where you want the new notebook, and type

```
<jupyter notebook>
```

Pro tip for mac users: if you have the path bar visible in your finder windows, you can navigate to your folder of choice, right click (or two-finger click) on its icon in the path bar, and select “services -> new terminal at folder” to open a terminal there.

The notebook “browser” or whatever we want to call it should appear in your default web browser. Select “New” over from the right to start a new notebook:



This will open a new tab in your browser (or perhaps a new window) with your new notebook in it with one code cell ready to go for you!

One potential source of confusion is Jupyter’s use of the browser as the graphical interface. To new users, it can imply that we’re doing something on the web. *We are not!* If you look at the address bar of your notebook (or of the Jupyter browser thingy), you can see that we are not at an `http://www` or `http://www` address, we are at a `localhost:`, which actually just refers to **your computer!** We are using Python on your computer, and Jupyter is simply using your browser as a “front end” so that the Jupyter people didn’t have to re-invent the wheel.

1.4.1 The shift-enter shortcut

Back to our notebook! Put your cursor in the code cell and type some python! Once you've entered something, hit **shift-enter**.

```
[ ]: name = input("What's your name?")  
print("Hi " + name + "!")
```

```
[ ]:
```

shift-enter is a shortcut for the hitting the “Run” button in the toolbar, or picking “Cell -> Run Cell and Select Below” from the menu. We use **shift-enter** so often, that it's probably better to think of the other two as “long cuts” for it.

Notice that **shift-enter** runs you code *and* selects the cell below it. If there is no cell below, it creates a new code cell below it.

The **shift-enter** key combination also runs markdown cells, that is, it renders the markdown into pretty (hopefully) formatted text. Let's try that.

In your notebook, put your cursor in your code cell. Then, go to the pull-down menu in the toolbar that says “Code” and change it to “Markdown” (We'll learn a much less cumbersome shortcut for this in a minute).

Enter some markdown formatted code in the cell, like this for example:

```
# Here's a header  
Here's some text with a word in bold  
## Here's a sub-header  
*Here's some italic text.*
```

And here

```
* is  
* a  
* bullet  
* list
```

Beware that, like Python, markdown does care about spaces!

There is a guide to markdown formatting (on GitHub) [here](#). There are also tons of cheat sheets available online, one of which is in the Canvas module.

Once you've entered some markdown code, hit **shift-enter**. This will render your markdown cell into nicely formatted text, create a new code cell below it, and make the new cell your active cell. If the results aren't what you expect, go back and tinker with it. (Check the spaces - some marks require spaces after them, and some require no space be present!)

1.4.2 Telling code from markdown cells at-a-glance

When a markdown cell has been rendered (“run”), it looks like regular document text, so the difference between it a code cell is obvious. But when a markdown cell hasn’t been run, it’s just a gray box, much like a code cell. Here’s an un-run markdown cell:

And here’s a code cell:

[]:

As we briefly noted above, code cells can. be spotted by the telltale In []:

1.4.3 Edit mode and Command mode

When you click in the gray area of a code cell or an un-run markdown cell, you’ll have an active cursor blinking inside it. This means you are in **Edit** mode, and are ready to enter text into the cell.

In your notebook, notice that when you click into a cell, the a green border will appear around, including a thick side to the far left. The green border tells you immediately that you are in **Edit** mode in your notebook.

If you now click on the left side of the cell, outside the gray box over by the thick border, the border turns blue. This indicates you are in **Command** mode.

You can switch back and forth between the edit and command modes by using the **esc** key to switch to command mode, and the **return** (or **enter**) key to switch back to command mode.

edit mode Edit mode is, obviously, the mode you need to be in to edit the text in your cells. To jump into edit mode for a markdown cell that’s already been run, you can either

- double click anywhere on the text, or
- click once on the cell to select it in command mode, and then hit **return**

Try this in your notebook! Enter edit mode for the markdown cell you ran above, make some changes, and re-run it.

command mode Mastering command mode is what really speeds you up in working with Jupyter notebooks! When in command mode, single keystrokes do useful things.

Like with code, it’s often good to break long chunks of text into paragraphs or groups of paragraphs in separate cells in case you want to do some rearranging later on.

It is thus really nice to be able to quickly

- run a current markdown cell,
- turn the new code cell that’s automatically created into a markdown cell, and
- jump into edit mode to start writing in the new markdown cell.

Do do this via the menu is a bit cumbersome, but with command mode it’s very fast with a little practice. After hitting **shift-enter** (which, remember, will create a new code cell and put you in edit mode), you can just hit:

- **esc** to jump to command mode

- **m** to convert the cell to a markdown cell, and
- **enter** to jump back to edit mode.

Same goes if you have just run a code cell and want the next cell to be a markdown cell. This is also common, as we often want to write about the results of code in a markdown cell just below the code output.

Try this combo a few times in your notebook!

Some useful command mode keyboard shortcuts (in no particular order) are:

- **a** - insert new cell above
- **b** - insert new cell about
- **c** - copy cell
- **x** - cut cell
- **v** - paste cell below current cell
- **V** (**shift V**) - paste cell above current cell
- **d d** (double click the **d** key) - delete cell

Try a few in your notebook!

The shortcuts for menu items that have one are shown next to the items. You can view the full set and run any of them by hitting the little keyboard in the toolbar. Try a few of these.

(You've no doubt already figured this out, but all the common keyboard shortcuts for your system work as usual when you are in edit mode.)

1.5 Saving your notebook as a pdf file

[]: