Jupyter Notebook tips, tricks, and shortcuts

1. Keyboard Shortcuts

- Esc will take you into command mode where you can navigate around your notebook with arrow keys.
- While in command mode:
    - A to insert a new cell above the current cell, B to insert a new cell below.
    - M to change the current cell to Markdown, Y to change it back to code
    - D + D (press the key twice) to delete the current cell
- Enter will take you from command mode back into edit mode for the given cell.
- Shift + Tab will show you the Docstring (documentation) for the the object you have just typed in a code cell - you can keep pressing this short cut to cycle through a few modes of documentation.
- Ctrl + Shift + - will split the current cell into two from where your cursor is.
- Esc + F Find and replace on your code but not the outputs.
- Esc + O Toggle cell output.
- Select Multiple Cells:
- Shift + J or Shift + Down selects the next sell in a downwards direction. You can also select sells in an upwards direction by using Shift + K or Shift + Up.
- Once cells are selected, you can then delete / copy / cut / paste / run them as a batch. This is helpful when you need to move parts of a notebook.
- You can also use Shift + M to merge multiple cells.

2.Pretty Display of Variables

you can alter a modify the *ast_note_interactivity* kernel option to make jupyter do this for any variable or statement on its own line, so you can see the value of multiple statements at once.

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

If you want to set this behaviour for all instances of Jupyter (Notebook and Console), simply create a file ~/.ipython/profile_default/ipython_config.py with the lines below.

```
c = get_config()
# Run all nodes interactively
c.InteractiveShell.ast_node_interactivity = "all"
```

3. Easy links to documentation

by prepending a library, method or variable with **?**, you can access the Docstring for quick reference on syntax.

```
?str.replace()
Docstring:
S.replace(old, new[, count]) -> str

Return a copy of S with all occurrences of substring
old replaced by new.  If the optional argument count is
given, only the first count occurrences are replaced.
Type:      method_descriptor
```

4. Plotting in notebooks

There are many options for generating plots in your notebooks.

- matplotlib (the de-facto standard), activated with %matplotlib inline
- %matplotlib notebook provides interactivity but can be a little slow, since rendering is done server-side.
- Seaborn is built over Matplotlib and makes building more attractive plots easier. Just by importing Seaborn, your matplotlib plots are made 'prettier' without any code modification.
- mpld3 provides alternative renderer (using d3) for matplotlib code. Quite nice, though incomplete.
- bokeh is a better option for building interactive plots.
- plot.ly can generate nice plots - this used to be a paid service only but was recently open sourced.
- Altair is a relatively new declarative visualization library for Python. It's easy to use and makes great looking plots, however the ability to customize those plots is not nearly as powerful as in Matplotlib.

5. IPython Magic Commands

Being based on the IPython kernel, Jupyter has access to all the Magics from the IPython kernel, and they can make your life a lot easier!

```
# This will list all magic commands
%lsmagic
```

6. IPython Magic - %env: Set Environment Variables

You can manage environment variables of your notebook without restarting the jupyter server process. Some libraries (like theano) use environment variables to control behavior, %env is the most convenient way.

```
# Running %env without any arguments
# lists all environment variables

# The line below sets the environment
# variable OMP_NUM_THREADS
%env OMP_NUM_THREADS=4
env: OMP_NUM_THREADS=4
```

7. IPython Magic - %run: Execute python code

%run can execute python code from .py files - this is well-documented behavior. Lesser known is the fact that it can also execute other jupyter notebooks, which can quite useful. Note that using %run is not the same as importing a python module.

```
# this will execute and show the output from
# all code cells of the specified notebook
%run ./two-histograms.ipynb
```

8. IPython Magic - %load: Insert the code from an external script

This will replace the contents of the cell with an external script. You can either use a file on your computer as a source, or alternatively a URL.

```
# Before Running
%load ./hello_world.py
# After Running
# %load ./hello_world.py
if __name__ == "__main__":
        print("Hello World!")
Hello World!
```

9. IPython Magic - %store: Pass variables between notebooks.

The %store command lets you pass variables between two different notebooks.

```
data = 'this is the string I want to pass to different notebook'
%store data
del data # This has deleted the variable
Stored 'data' (str)
```

Now, in a new notebook...

```
%store -r data
print(data)
this is the string I want to pass to different notebook
```

10. IPython Magic - %who: List all variables of global scope.

The %who command without any arguments will list all variables that existing in the global scope. Passing a parameter like str will list only variables of that type.

```
one = "for the money"
two = "for the show"
three = "to get ready now go cat go"
%who str
one         three       two
```

11. IPython Magic - Timing

There are two IPython Magic commands that are useful for timing - %%time and %timeit. These are especially handy when you have some slow code and you're trying to indentify where the issue is.
%%time will give you information about a single run of the code in your cell.

```
%%time
import time
for _ in range(1000):
    time.sleep(0.01)# sleep for 0.01 seconds
CPU times: user 21.5 ms, sys: 14.8 ms, total: 36.3 ms
Wall time: 11.6 s
```

%%timeit uses the Python timeit module which runs a statement 100,000 times (by default) and then provides the mean of the fastest three times.

```
import numpy
%timeit numpy.random.normal(size=100)
The slowest run took 7.29 times longer than the fastest. This could mean that an intermediate result is being cached.
100000 loops, best of 3: 5.5 µs per loop
```

12. IPython Magic - %%writefile and %pycat: Export the contents of a cell/Show the contents of an external script

Using the %%writefile magic saves the contents of that cell to an external file. %pycatdoes the opposite, and shows you (in a popup) the syntax highlighted contents of an external file.

```
%%writefile pythoncode.py

import numpy
def append_if_not_exists(arr, x):
    if x not in arr:
        arr.append(x)

def some_useless_slow_function():
    arr = list()
    for i in range(10000):
        x = numpy.random.randint(0, 10000)
        append_if_not_exists(arr, x)
```

```
Writing pythoncode.py
```

```
%pycat pythoncode.py
```

```python
import numpy
def append_if_not_exists(arr, x):
    if x not in arr:
        arr.append(x)

def some_useless_slow_function():
    arr = list()
    for i in range(10000):
        x = numpy.random.randint(0, 10000)
        append_if_not_exists(arr, x)
```

13. IPython Magic - %prun: Show how much time your program spent in each function.

Using `%prun statement_name` will give you an ordered table showing you the number of times each internal function was called within the statement, the time each call took as well as the cumulative time of all runs of the function.

```python
%prun some_useless_slow_function()
```

```
26324 function calls in 0.556 seconds

Ordered by: internal time

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
 10000    0.527    0.000    0.528    0.000 <ipython-input-46-b52343f1a2d5>:2(append_if_not_exists)
 10000    0.022    0.000    0.022    0.000 {method 'randint' of 'mtrand.RandomState' objects}
     1    0.006    0.006    0.556    0.556 <ipython-input-46-
b52343f1a2d5>:6(some_useless_slow_function)
  6320    0.001    0.000    0.001    0.000 {method 'append' of 'list' objects}
     1    0.000    0.000    0.556    0.556 <string>:1(<module>)
     1    0.000    0.000    0.556    0.556 {built-in method exec}
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

13. IPython Magic - Debugging with %pdb

Jupyter has own interface for The Python Debugger (pdb). This makes it possible to go inside the function and investigate what happens there.
You can view a list of accepted commands for pdb here.

```
%pdb

def pick_and_take():
    picked = numpy.random.randint(0, 1000)
    raise NotImplementedError()

pick_and_take()
```

```
Automatic pdb calling has been turned ON
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-24-0f6b26649b2e> in <module>()
      5     raise NotImplementedError()
      6
----> 7 pick_and_take()

<ipython-input-24-0f6b26649b2e> in pick_and_take()
      3 def pick_and_take():
      4     picked = numpy.random.randint(0, 1000)
----> 5     raise NotImplementedError()
      6
      7 pick_and_take()

NotImplementedError:
```

```
> <ipython-input-24-0f6b26649b2e>(5)pick_and_take()
      3 def pick_and_take():
      4     picked = numpy.random.randint(0, 1000)
----> 5     raise NotImplementedError()
      6
      7 pick_and_take()
```

```
ipdb>
```

14. Executing Shell Commands

It's easy to execute a shell command from inside your notebook. You can use this to check what datasets are in available in your working folder:

```
!ls *.csv
```

```
nba_2016.csv            titanic.csv
pixar_movies.csv        whitehouse_employees.csv
```

Or to check and manage packages.

```
!pip list | grep pandas
Requirement already satisfied (use --upgrade to upgrade): numpy in
/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/site-packages
pandas (0.18.1)
```

15. Using LaTeX for forumlas

When you write LaTeX in a Markdown cell, it will be rendered as a formula using MathJax. This:

```
$$ P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)} $$
```

Becomes this:

$$P(A|B)=P(B|A),P(A)P(B)P(A|B)=P(B|A),P(A)P(B)$$

Markdown is an important part of notebooks, so don't forget to use its expressiveness!

16. Run code from a different kernel in a notebook

If you want to, you can combine code from multiple kernels into one notebook.Just use IPython Magics with the name of your kernel at the start of each cell that you want to use that Kernel for:

- %%bash
- %%HTML
- %%python2
- %%python3
- %%ruby
- %%perl

```
%%bash
for i in {1..5}
do
  echo "i is $i"
done
```

```
i is 1
i is 2
i is 3
i is 4
i is 5
```

17. Install other kernels for Jupyter

One of the nice features about Jupyter is ability to run kernels for different languages. As an example, here is how to get and R kernel running.

Easy Option: Installing the R Kernel Using Anaconda

If you used Anaconda to set up your environment, getting R working is extremely easy. Just run the below in your terminal:

```
conda install -c r r-essentials
```

Less Easy Option: Installing the R Kernel Manually

If you are not using Anaconda, the process is a little more complex. Firstly, you'll need to install R from CRAN if you haven't already.
Once that's done, fire up an R console and run the following:

```
install.packages(c('repr', 'IRdisplay', 'crayon', 'pbdZMQ', 'devtools'))
devtools::install_github('IRkernel/IRkernel')
IRkernel::installspec()  # to register the kernel in the current R installation
```

18. Running R and Python in the same notebook.

The best solution to this is to install rpy2 (requires a working version of R as well), which can be easily done with pip:

```
pip install rpy2
```

You can then use the two languages together, and even pass variables inbetween:

```
%load_ext rpy2.ipython
%R require(ggplot2)
```

```
array([1], dtype=int32)
import pandas as pd
df = pd.DataFrame({
        'Letter': ['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'c'],
        'X': [4, 3, 5, 2, 1, 7, 7, 5, 9],
        'Y': [0, 4, 3, 6, 7, 10, 11, 9, 13],
        'Z': [1, 2, 3, 1, 2, 3, 1, 2, 3]
    })
%%R -i df
ggplot(data = df) + geom_point(aes(x = X, y= Y, color = Letter, size = Z))
```

19. Writing functions in other languages

Sometimes the speed of numpy is not enough and I need to write some fast code.
In principle, you can compile function in the dynamic library and write python wrappers...

But it is much better when this boring part is done for you, right?

You can write functions in cython or fortran and use those directly from python code.

First you'll need to install:

```
!pip install cython fortran-magic
%load_ext Cython
%%cython
def myltiply_by_2(float x):
    return 2.0 * x
myltiply_by_2(23.)
```

Personally I prefer to use fortran, which I found very convenient for writing number-crunching functions. More details of usage can be found here.

```
%load_ext fortranmagic
%%fortran
subroutine compute_fortran(x, y, z)
    real, intent(in) :: x(:), y(:)
    real, intent(out) :: z(size(x, 1))
```

```
    z = sin(x + y)

end subroutine compute_fortran
compute_fortran([1, 2, 3], [4, 5, 6])
```

There are also different jitter systems which can speed up your python code. More examples can be found here.

20. Jupyter-contrib extensions

Jupyter-contrib extensions is a family of extensions which give Jupyter a lot more functionality, including e.g. jupyter spell-checker and code-formatter.
The following commands will install the extensions, as well as a menu based configurator that will help you browse and enable the extensions from the main Jupyter notebook screen.

```
!pip install https://github.com/ipython-contrib/jupyter_contrib_nbextensions/tarball/master
!pip install jupyter_nbextensions_configurator
!jupyter contrib nbextension install --user
!jupyter nbextensions_configurator enable --user
```

21. Create a presentation from a Jupyter notebook.

Damian Avila's RISE allows you to create a powerpoint style presentation from an existing notebook.
You can install RISE using conda:

```
conda install -c damianavila82 rise
```

Or alternatively pip:

```
pip install RISE
```

And then run the following code to install and enable the extension:

```
jupyter-nbextension install rise --py --sys-prefix
jupyter-nbextension enable rise --py --sys-prefix
```

22. The Jupyter output system

Notebooks are displayed as HTML and the cell output can be HTML, so you can return virtually anything: video/audio/images.

In this example I scan the folder with images in my repository and show thumbnails of the first 5:

```
import os
from IPython.display import display, Image
names = [f for f in os.listdir('../images/ml_demonstrations/') if f.endswith('.png')]
for name in names[:5]:
    display(Image('../images/ml_demonstrations/' + name, width=100))
```

We can create the same list with a bash command, because magics and bash calls return python variables:

```
names = !ls ../images/ml_demonstrations/*.png
names[:5]
['../images/ml_demonstrations/colah_embeddings.png',
 '../images/ml_demonstrations/convnetjs.png',
 '../images/ml_demonstrations/decision_tree.png',
 '../images/ml_demonstrations/decision_tree_in_course.png',
 '../images/ml_demonstrations/dream_mnist.png']
```

23. 'Big data' analysis

A number of solutions are available for querying/processing large data samples:

- ipyparallel (formerly ipython cluster) is a good option for simple map-reduce operations in python. We use it in rep to train many machine learning models in parallel
- pyspark
- spark-sql magic %%sql

24. Sharing notebooks

The easiest way to share your notebook is simply using the notebook file (.ipynb), but for those who don't use Jupyter, you have a few options:

- Convert notebooks to html file using the File > Download as > HTML Menu option.
- Share your notebook file with gists or on github, both of which render the notebooks. See this example.
- If you upload your notebook to a github repository, you can use the handy mybinder service to allow someone half an hour of interactive Jupyter access to your repository.
- Setup your own system with jupyterhub, this is very handy when you organize mini-course or workshop and don't have time to care about students machines.
- Store your notebook e.g. in dropbox and put the link to nbviewer. nbviewer will render the notebook from whichever source you host it.
- Use the File > Download as > PDF menu to save your notebook as a PDF.