Introduction to Data Analysis with IPython and Wakari

Ian Stokes-Rees (http://about.me/ijstokes), @ijstokes (http://twitter.com/ijstokes) (http://www.linkedin.com/in/ijstokes)
PyData Boston (http://pydata.org/bos2013), July 2013

(http://continuum.io)

Outline

- Understand **IPython (http://ipython.org/)**, **IPython Notebook (http://ipython.org/notebook.html)**, and **Wakari (http://wakari.io)**
- Basic data analysis with IPython
- Data visualization
- Publishing results
- Basic parallel computing with **IPython Parallel (http://ipython.org/ipython-doc/dev/parallel/)** *(time permitting)*

A Note About Wakari

- Freemium web-based data analytics service
- Platform for collaborative data intensive analytics, visualization, and publishing
- Combines Continuum's Anaconda Python distribution and IPython Notebook
- This tutorial will focus on "hosted" IPython Notebook service
- Just like IPython makes Python easier to use interactively on your computer, Wakari makes Python-centric analytics easier to do through your browser

First Steps

1. Create a Wakari account at **http://wakari.io (http://wakari.io)** (will need to confirm email within 24 hours)
2. Start a bash terminal in Wakari
3. `git clone https://github.com/ijstokes/wakari_tutorial.git`
4. Refresh file manager (left window)
5. Double click on `wakari_tutorial` directory
6. Click on IPython Notebook file `Introduction to Data Analysis with IPython and Wakari.ipynb`

7. Follow along and execute commands with `SHIFT-ENTER` in each cell.
8. If you are in a classroom setting, introduce yourself to the person on either side. Tell them your name and your favorite Monty Python movie. These two people will be your Python programming buddies for the next hour, so be prepared to help each other out.

(http://continuum.io)

Python Statements and Notebook Cells

Each cell can contain one or more Python statements

In [ ]:
```
print "This is a cell"
```

In [ ]:
```
x = 4
y = 7
result = x + y
print result
```

Practice: Cells and Kernels

In [ ]:
```
import time
time.asctime()
```

In [ ]:
```
print "Seconds since the epoch:", time.time()
print "Years since the epoch (approx): %.2f" % (time.time()/(365 * 24 * 60 * 60))
print "Previously computed value result: ", result
```

1. And cells can be evaluated multiple times -- try re-executing the above `time` cells and watch the time change `
2. A running notebook (like this one, if you've opened it in IPython Notebook") has a Python kernel associated with it, and executing a cell is like running the statements in that cell inside an interactive IPython interpreter.
3. You can re-order cells and then re-execute them, but re-ordering does not re-run all other cells. For example, try moving the above "seconds since the epoch" cell up near the **First Steps** section using the ⬆ button in the toolbar, then re-run it.
4. Notice that the `time` module and the `result` variable are still in scope.

5. When you save your Notebook (with the ⍰ button in the toobar) all your input and output cells are saved as well. Try it now, then select *File*→*Close and Halt* -- you will get a warning message that the kernel has stopped.
6. Close the notebook by clicking the ⊗ in the tab.
7. Re-open the notebook by double clicking on it in the left sidebar menu.
8. Notice the output has been saved -- this hasn't re-executed. Clear the output by selecting *Cell*→*All Output*→*Clear*.
9. Now try re-executing all the cells in the notebook by selecting *Cell*→*Run All*.
10. If you moved the `time` cell up before saving, then you'll get an exception when this runs, since `import time` hasn't been called yet.

```
In [ ]: wont_work = 73 + 'some string'
```

11. Notice that *Run All* will stop if it hits a code cell that throws an exception. Fix `'some string'` by making it a number, then select *Cell →Run All Below*.
12. You can also delete and insert cells. Select the cell above and then cut it out with the ✂ menu bar button. Add a new cell above this one using the ⤒ button. Try adding some code and executing it.
13. You can also copy cells with the copy button (right of the scissors), and paste them with the paste button (clipboard two-right of the scissors). Try copying the "time since epoch" cell.

Non-Code Cells

- You can use the tool bar drop-down menu to change a particular cell from *code* to something that will allow you to enter plain text, HTML, or Markdown.
- This can even include mathematical notation via LaTeX
  $c = \sqrt{a^2 + b^2}$
- Click on any text cell in this notebook to see examples of this.
- **Markdown (http://daringfireball.net/projects/markdown/basics)** is a great way to quickly write readable text that can also render to HTML for the web. Here is a **short tutorial (http://www.markdowntutorial.com/)**

13. The loop below will take a long time to run. In IPython Notebook we can interrupt the kernel similar to a CTRL-C signal by clicking on the stop button "■" in the menu bar. Selecting *Kernel →Interrupt* will do the same thing. Start the loop running, then after a few seconds stop it with this method.
14. It is often necessary to "start fresh", and that is done by resetting the Notebook kernel with *Kernel →Reset*. After you've done this, you'll need to re-execute cells from the *Cells* menu or by selectively using `SHIFT-ENTER` in cells of interest.

```
In [ ]: for i in xrange(1000):
            print i
            time.sleep(0.5)
```

IPython and IPython Notebook

IPython extends the functionality of the standard python interactive interpreter.

?? Should you ever use IPython to run Python programs?

?? What are some features of IPython?

- IPython should only be used interactively, never `ipython myprog.py`.
- Extended commands and syntax are only meant for interactive use, not to be added to Python modules.
- Many great features:
  - tab-completion
  - logging of commands (and output)
  - file system interaction (`cd ls cp` etc.)
  - extensions
  - "magics"
  - QT console (inline graphics)
  - Notebook

In [ ]:
```
pwd
```

In [ ]:
```
ls data
```

In [ ]:
```
# tab completion on attributes and functions with code-insight
time.
# NOTE: module must be loaded already
```

```
In [ ]: # Help text
        time?
```

```
In [ ]: # tab completion on file names
        fh = open('data/cla
```

```
In [ ]: ls data/eur
```

In [ ]:
```
%%bash
for f in data/*.dat; do
    echo -n $f " "
    cat $f | wc -l
done
```

In [ ]:
```
%load_ext autosave
%autosave 300
```

In [ ]:
```
time?
```

In [ ]:
```
%run power
```

Explore IPython and IPython Notebook

- you can run ipython yourself:
    - Install it (if you don't have it already): `$ pip install ipython`
    - Run it from the command line: `$ ipython`
- And if you have ipython then you can start your own ipython notebook: `$ ipython notebook --pylab=inline`
- But for now, stick with **Wakari (http://wakari.io)**

1. In a new cell, `import math` and execute it.
2. In a new cell, type the following:

    - `math.<TAB>`
    - `math.cos(<TAB>`

3. In a new cell, execute:

    - `math.atan?`

4. In a new cell, execute:

    - `%run weather.py`

5. Double click on `weather.py` in the left menubar file browser to open it in a file editor. Take a look at what it is doing.
6. In this notebook, write a small script that opens `data/europe-cities-temp.dat` and prints out each line.

    - If that is easy for you, convert the data into a list of tuples, each tuple containing the data on each row, with fields converted to ints or floats if possible.
    - If you still have time, change this into a dictionary, with city names as keys, and Lat/Lon converted to floats.

Publishing Your Work

- IPython Notebooks are simply JSON files
- Any code and all output (including generated graphics) are embedded in the JSON file
- This means you can share your published notebook by email or post it to the web
- Does not include and custom code, data files, or other dependencies
- Wakari facilitates publishing notebooks
- Wakari can also publish *bundles* that include all files in a directory
- And via *conda environments*, Wakari can automatically describe any package dependencies
    - If you know virtualenv, imagine this as a manifest of all special packages required for the bundle to work

Create Your Own Shared Notebook

1. Click on *New Notebook*
2. This will create a notebook with the name "*Untitled X*". Change the notebook name to "Weather Analysis" and save it.
3. Make the first cell Markdown and enter a title similar to the one used for this notebook, including a title and your name.
4. Add in a code cell with the `autosave` extension loaded and activated.
5. Add the following code block and execute it:
6. If you get an error regarding `basemap` then go to the *Terminals* tab in Wakari and start a *Shell* terminal then execute (don't inculde the dollar sign): `$ conda install basemap`

In [ ]:
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(figsize=(12,12))
map = Basemap(projection='merc', lat_0 = 50, lon_0 = 5,
    resolution = 'i',
    llcrnrlon=-15, llcrnrlat=35,
    urcrnrlon=20, urcrnrlat=62)

map.drawcoastlines()
map.drawcountries()
map.fillcontinents(color = 'coral')
map.drawmapboundary()

plt.show()
```

7. Share your notebook by clicking on the `[Share]` button beside it in the left menubar file manager. Provide a short description. Do not check "Share Environment" or set a password.
8. Try sharing your link with one of your tutorial buddies. Make sure you can open and view it in your browser.
9. Click on the *code* link near the top to hide the code in the notebook. Depending on the notebook content, this is sometimes a more useful and readable view.
10. This published notebook view is a static, read-only snapshot of your notebook. Others can clone your notebook into their own Wakari environment, or they can download the notebook to load in their own IPython Notebook environment.
11. Return to your "live" notebook and enter the code below after your `import` statements, but before the figure is generated into the existing code cell:

```
In [ ]: 'Reykjavik   61  (64,45,"N")     (21,56,"W") 0.9 6.2 3.7 9.8 7.1 12.4    8.8 13.9'
        regexp = "".join(['([^\t]+)\t([^\t]+)\t\((\d+),(\d+),"(\w)"\)\s+\((\d+),(\d+),',
                 '"(\w)"\)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)\t([^\t]+)',
                 '\t([^\t]+)\t([^\t]+)\n'])

        ndtype=[('city',str),
                ('elev', float),
                ('lat', float),
                ('lats', float),
                ('latns', '|S1'),
                ('lon', float),
                ('lons', float),
                ('lonew', '|S1'),
                ('apr_min', float),
                ('apr_max', float),
                ('may_min', float),
                ('may_max', float),
                ('jun_min', float),
                ('jun_max', float),
                ('jul_min', float),
                ('jul_max', float),
        ]

        with open('data/europe-cities-temp-mod.dat') as fh:
             cities = np.fromregex(fh, regexp, ndtype)

        print cities.size
        print cities['jul_max']
```

This should print out `17` and a list of values which are the July maximum temperatures for 17 European cities. If it does not, and you can't see what the problem is, ask your tutorial buddy for debugging help.

Finally, add the code below into the code cell just *before* the `plt.show()` statement.

```
In [ ]: lat = cities['lat'] + cities['lats']/60
        lon = cities['lon'] + cities['lons']/60
        lon = lon*np.where(cities['lonew'] == 'E', 1, -1)
        x,y = map(lon, lat)
        map.plot(x, y, 'bo', markersize=30, alpha=0.5)
```

The last few steps are to save the figure as a file in PDF and PNG formats, save your work, and then share the notebook again. This time we will also share a bundle of all the work, for comparison.

1. Add the code below as a new cell at the end of your notebook, and execute it:

```
In []: plt.savefig('weather.pdf')
       plt.savefig('weather.png')
```

2. Save your Notebook.
3. Click the `[Share]` button in the Wakari file manager to re-publish your updated notebook. Check that the link works.
4. Click the `Go Back` link in the Wakari file manager to return to your home directory. Now click once on the `wakari_tutorial` directory to select it. You can now click the `[Share]` button in the menu bar above the file manager (middle button: box with an arrow going out of it). As before, do not share your environment.

Further References

- **Wakari (http://wakari.io)**
- **IPython (http://ipython.org/)**
- **IPython Notebook (http://ipython.org/notebook.html)**
- IPython Notebook Gallery **1 (https://www.wakari.io/gallery/)** **2 (http://nbviewer.ipython.org/)**
- **IPython Parallel (http://ipython.org/ipython-doc/dev/parallel/)**
- **IPython Parallel Tutorial (http://continuum.io/blog/ipcluster-wakari-intro)**