PyData Keynote

November 9, 2013

1 IPython: The Attributes of Software and How They Affect Our Work

1.1 Front matter

Brian Granger
Department of Physics, California Polytechnic State University
Core Developer, IPython Project
This talk was presented at PyData New York 2013 on November 10, 2013.

```
In [62]: from IPython.display import display, Image, HTML
         from talktools import website, nbviewer
```

```
In [12]: Image('images/ipython_logo.png')
```

Out[12]:

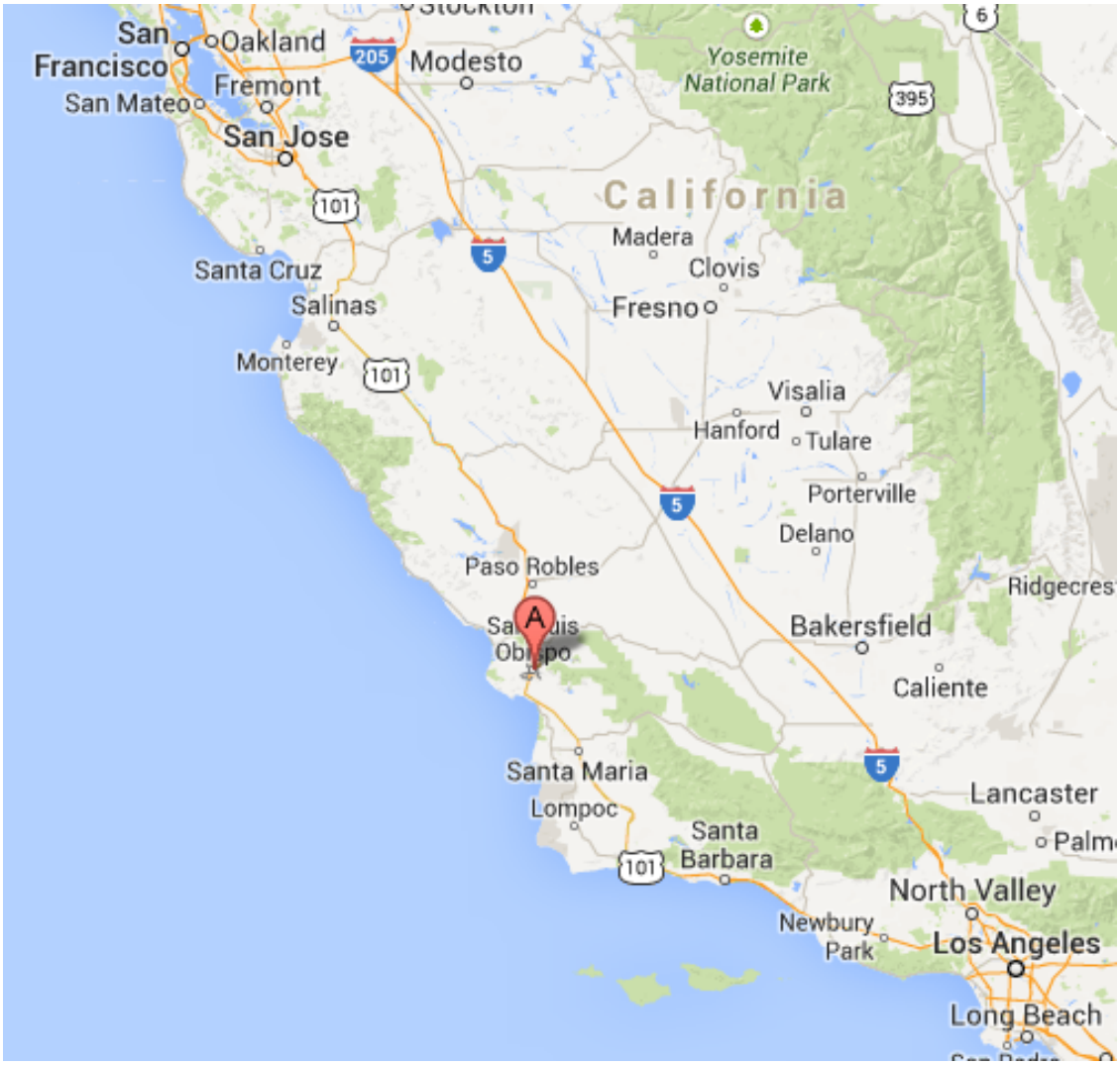

```
In [11]: Image('images/calpoly_logo.png')
```

Out[11]:

Cal Poly is located in San Luis Obispo, CA, half way between LA and SF.

```
In [20]: Image('images/slo_map.png', width=500)
```

Out[20]:



In [21]: Image('images/beach.png', width=500)

Out[21]:

1.2 IPython Project

IPython is an open source, interactive computing environment.

```
In [31]: website('ipython.org')
```

```
Out[31]: <IPython.core.display.HTML at 0x10173fb10>
```

IPython was started in 2001 by Fernando Perez, who continues to lead the project. Over the last decade the project has grown to a dedicated and talented team of ≈ 12 core developers and a larger community of ≈ 100 contributors.

IPython has been funded by:

- Volunteers (mostly)
- NASA, DOD/DRC, NIH
- Enthought
- Rackspace
- Alfred P. Sloan Foundation
  - 1.15 million dollar grant for 2013-2014
- Microsoft
  - 100,000 in August 2013

1.3 Software attributes

In this talk, I will cover many aspects of IPython, especially the IPython Notebook and its surrounding ecosystem. But that is not the main point of the talk. The main point is an argument about the attributes of the tools we use and how they affect our work. Here is the argument:

1. Computing, and thus software tools, are one of the foundations of modern technical work.
2. Like all tools, software tools have attributes.
3. When humans use those tools, these attributes affect behavior and thinking.
4. The technical work itself inherits those attributes through the behavior and thinking.

What are these attributes?

Fast, slow, cluttered, simple, complex, open, documented, tested, testable, buggy, robust, heirarchical, flat, single-purpose, multi-purpose, opaque, transparent, etc.

How are attributes different from features?

I am not sure, but the attributes I want to talk about today are more conceptual and abstract than most things I label with the word *feature*. But, maybe they are the same thing.

Can you give an example?

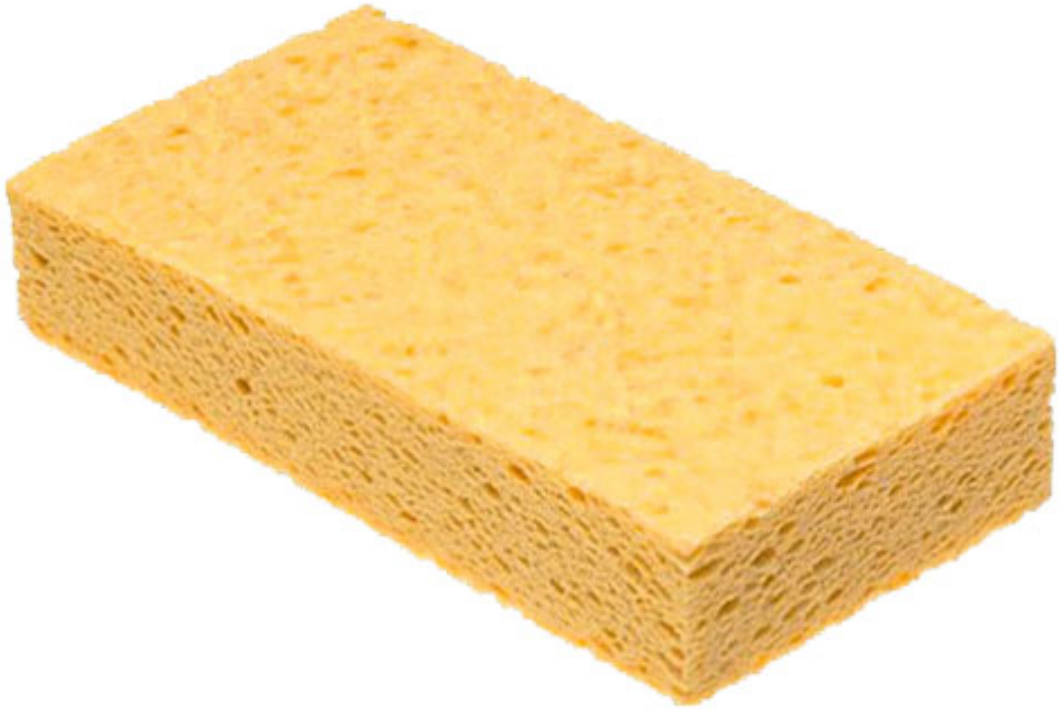
If you are doing science using software that is cluttered, slow and buggy, your science is going to be cluttered, slow and buggy. More importantly, your frustration will affect your thinking and restrict technical progress.

How about a more concrete example?

A sponge is a tool that can be used for washing dishes. It has the attributes of being **slow** and **cheap**.

```
In [34]: Image('images/sponge.jpeg', width=300)
```

```
Out[34]:
```



A dishwasher is also a tool for washing dishes. It has the attributes of being **fast** and **expensive**.

```
In [36]: Image('images/dishwasher.jpeg', width=300)
```

```
Out[36]:
```

How do these attributes of **slow/fast** and **cheap/expensive** affect human behavior and thinking?

- If you live with others, using a sponge encourages you to talk to each other ("I'll wash, you dry") as you do the dishes.
- If you live alone, using a sponge can give you time to reflect and think.
- If you are really busy, using a sponge can lead to frustration.
- If you use a dishwasher, you have more time for other activities.
- If you use a dishwasher, some of that extra time goes into working more to pay for it.

As another example, Edward Tufte has written a beautiful essay on how the attributes of Microsoft's Powerpoint Software affect the quality and nature of presentations.

```
In [53]: website('http://www.edwardtufte.com/tufte/powerpoint')
```

```
Out[53]: <IPython.core.display.HTML at 0x100428a10>
```

Put another way: **If you want to change the character of the technical work you are doing, have a look at your tools.**

2 Attribute #1: Useful in multiple contexts

Scientific computing and data science are complex activities that involve a wide range of contexts:

1. Individual, interactive exploration
2. Debugging, testing
3. Production runs
4. Parallel computing
5. Collaboration
6. Publication
7. Presentation
8. Teaching/Learning

There are a large number of software tools that we use across these different contexts:

Python Ruby Perl C C++ Fortran Numba Cython MPI Hadoop Excel LaTeX Powerpoint Word Keynote Vim Emacs Make JavaScript Matlab Mathematica

This places a massive cognitive burden on users. This burden has nothing to do with the challenging technical problems users are trying to solve. This burden pulls them away from solving their actual problems.

We are working really hard to make sure that IPython is useful in the following contexts.

2.0.1 Interactive exploration

First and foremost, IPython is an interactive environment for writing and running code. We provide features to make this as pleasant as possible.

Tab completion

```
In [39]: import math

In []: math.
```

Interactive help

```
In [40]: math.cos?
```

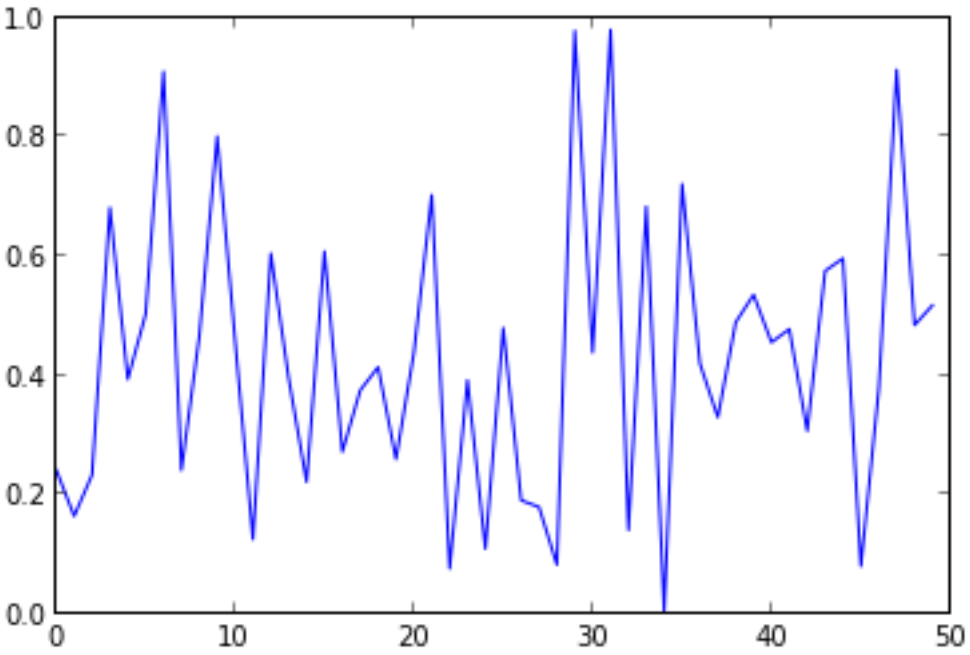
Inline plotting

```
In [37]: %pylab inline

Populating the interactive namespace from numpy and matplotlib

In [38]: plot(rand(50))

Out[38]: [<matplotlib.lines.Line2D at 0x1043f2610>]
```



2.0.2 Publishing

IPython Notebook contain everything related to a computational and its results: code, narrative text, equations, plots, images, videos, HTML, JavaScript. We have developed tools for "publishing" these Notebook documents in different contexts:

- nbconvert, which ships with IPython 1.0, converts Notebooks to different static formats: HTML, LaTeX, slideshows.
- nbviewer provides a static HTML view of any Notebook on the internet.

Let's generate a static PDF of this talk's Notebook:

```
In [50]: !ipython nbconvert --to latex --post pdf "PyData Keynote.ipynb"
```

```
[NbConvertApp] Using existing profile dir: u'/Users/bgranger/.ipython/profile_default'
[NbConvertApp] Converting notebook PyData Keynote.ipynb to latex
[NbConvertApp] Support files will be in PyData Keynote_files/
[NbConvertApp] Loaded template latex_article.tplx
[NbConvertApp] Writing 28448 bytes to PyData Keynote.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running pdflatex 3 times: ['pdflatex', 'PyData Keynote.tex']
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
libpng warning: iCCP: known incorrect sRGB profile
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'PyData Keynote']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no citations
[NbConvertApp] Removing temporary LaTeX files
[NbConvertApp] PDF successfully created
```

Here is the nbviewer website.

```
In [23]: website('nbviewer.ipython.org')
```

```
Out[23]: <IPython.core.display.HTML at 0x10173f890>
```

We also maintain a gallery of interesting Notebooks that contains a curated list of IPython Notebooks on various topics.

Cam Davidson-Pilon has written an entire book on Bayesian Statistics as a set of IPython Notebook that are hosted on GitHub and viewed on http://nbviewer.ipython.org.

```
In [24]: website('http://camdavidsonpilon.github.io/Probabilistic-Programming-and-Bayesian-Methods-for-
```

```
Out[24]: <IPython.core.display.HTML at 0x10173f410>
```

Matthew Russell has written an O'Reilly published book that includes IPython Notebooks for all examples.

```
In [25]: website('http://shop.oreilly.com/product/0636920030195.do')
```

```
Out[25]: <IPython.core.display.HTML at 0x10173f850>
```

Jose Unpingco has written a series of blog posts using the IPython Notebook on Signal Processing. These blog posts were the basis of full length book Python for Signal Processing, Springer (2013).

```
In [26]: website('http://python-for-signal-processing.blogspot.com/')
```

```
Out[26]: <IPython.core.display.HTML at 0x10173f7d0>
```

Jake Vanderplas and others publish technical blogs that are authored as IPython Notebooks.

```
In [27]: website('http://jakevdp.github.io/blog/2013/08/28/understanding-the-fft/')
```

```
Out[27]: <IPython.core.display.HTML at 0x10173f910>
```

2.0.3 Presenting

The IPython Notebook, even in its standard incarnation is being used extensively for presentations on technical topics across a wide range of fields. The Notebook has a cell toolbar for adding slide related metadata to cells. However, we are working on improving this usage case:

1. Prototype live presentation mode.
2. Use nbconvert to export to reveal.js based slideshow.

2.0.4 Teaching

The IPython Notebook is being used for lecture materials and student work in a number of university and high school courses on scientific computing and data science. Most of these courses are being developed publicly on GitHub. Here is a short list:

```
In [47]: %%file courses.csv
         "Course","University","Instructor"
         "Data Science (CS 109)","Harvard University","Pfister and Blitzstein"
         "Practical Data Science","NYU","Josh Attenberg"
         "Scientific Computing (ASTR 599)","University of Washington","Jake Vanderplas"
         "Working with Open Data","UC Berkeley","Raymond Yee"
         "Computational Physics","Cal Poly","Jennifer Klay"
```

```
Overwriting courses.csv
```

```
In [48]: import pandas
```

```
In [49]: df = pandas.read_csv('courses.csv'); df
```

```
Out[49]:                              Course                University              Instructor
         0            Data Science (CS 109)        Harvard University  Pfister and Blitzstein
         1          Practical Data Science                       NYU          Josh Attenberg
         2  Scientific Computing (ASTR 599)  University of Washington         Jake Vanderplas
         3          Working with Open Data               UC Berkeley            Raymond Yee
         4           Computational Physics                  Cal Poly           Jennifer Klay
```

In a teaching context it is also important to be able to includes video content. The IPython display architecture makes this simple for YouTube videos.

```
In [52]: from IPython.display import YouTubeVideo
         YouTubeVideo('Jsiy4TxgIME')
```

```
Out[52]: <IPython.lib.display.YouTubeVideo at 0x10041fcd0>
```

2.1 Attribute #2: Multi-lingual

It is 2013 and in case you haven't noticed, there is now more than one programming language. We *love* Python, but insisting that everyone write everything in Python all the time is naive. The reality is that most large, complex projects involve multiple languages. In fact, it is commmon for projects to use at least three classes of languages:

- Low-level (C, C++, Java, Fortran)
- High-level (Python, Ruby, Perl, Julia)
- Web-based (JavaScript, CoffeeScript, HTML, CSS)

Despite its name and historical development, IPython is now a **multi-lingual** project that supports a wide range of programming languages throughout its architecture.

2.1.1 Kernel architecture

In the IPython architecture, the **kernel** is a separate process that runs the user's code and returns the output back to the frontend (Notebook, Terminal, etc.). Kernels talk to Frontends using a well documented message protocol (JSON over ZeroMQ and WebSockets). The default IPython kernel that ships with IPython, knows how to run Python code. But there are now kernels in other languages:

- Julia (https://github.com/JuliaLang/IJulia.jl)
- Ruby (https://github.com/minrk/iruby)
- Haskell (https://github.com/gibiansky/IHaskell)
- Scala (https://github.com/Bridgewater/scala-notebook)
- node.js (https://gist.github.com/Carreau/4279371)

2.1.2 Magic commands

The standard IPython kernel also allows running code in other languages using the `%%magic` syntax.

Here is a bit of Ruby:

```
In [54]: %%ruby
         puts "Hello from Ruby #{RUBY_VERSION}"
```

```
Hello from Ruby 2.0.0
```

And some Bash:

```
In [55]: %%bash
         echo "hello from $BASH"
```

```
hello from /bin/bash
```

The R magic has to be loaded from an extension:

```
In [57]: %load_ext rmagic
```

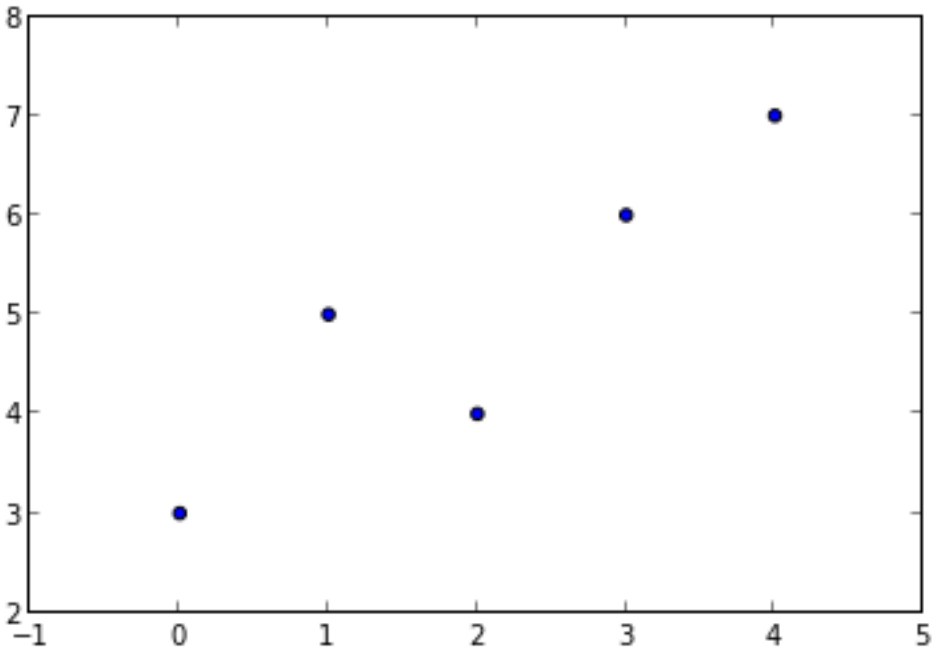
Create two arrays in Python:

```
In [67]: import numpy as np
         X = np.array([0,1,2,3,4])
         Y = np.array([3,5,4,6,7])
```

Use those arrays to make a scatter plot using Matplotlib:

```
In [59]: scatter(X,Y)
```

```
Out[59]: <matplotlib.collections.PathCollection at 0x103368890>
```



Pass the arrays to R and fit a linear model:

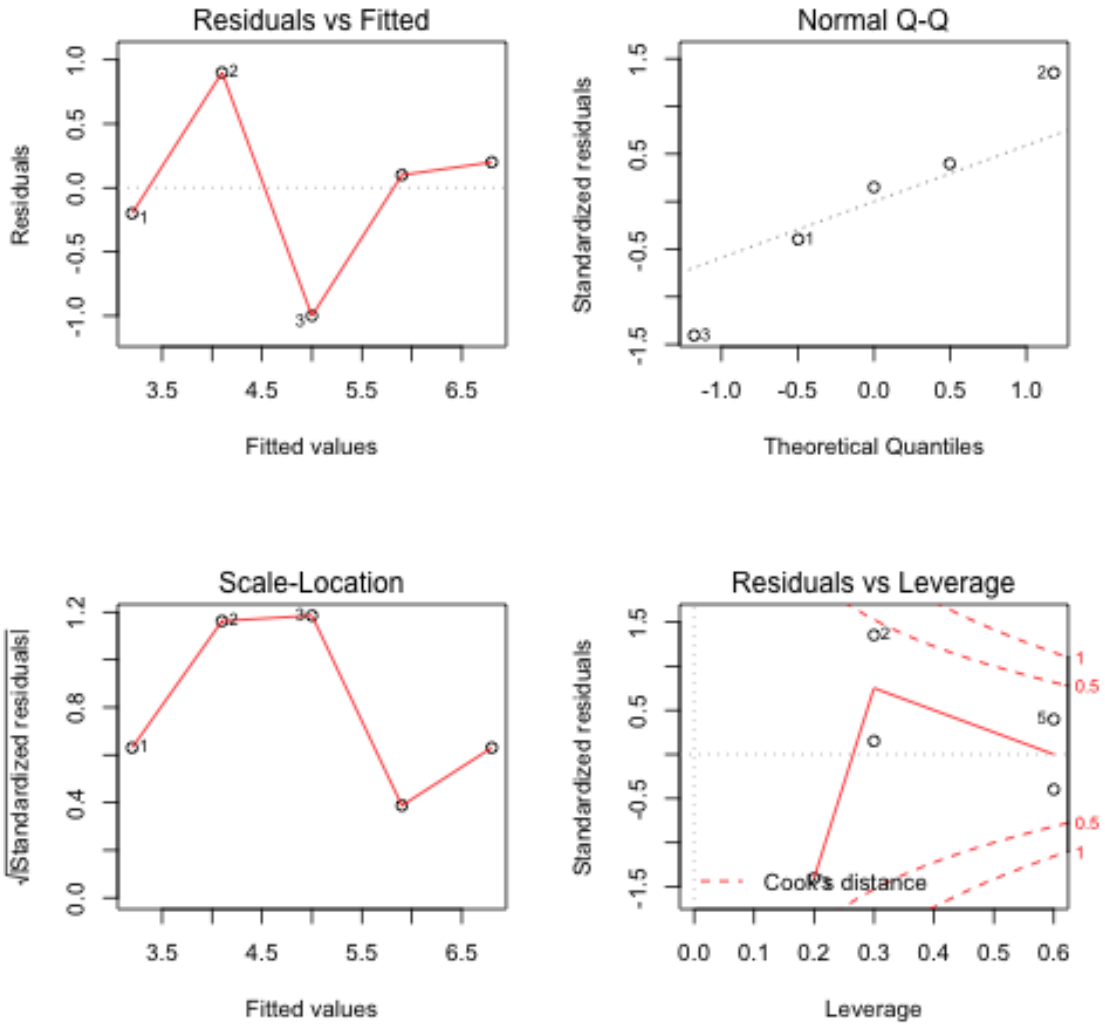
```
In [60]: %%R -i X,Y -o XYcoef
         XYlm = lm(Y~X)
         XYcoef = coef(XYlm)
         print(summary(XYlm))
         par(mfrow=c(2,2))
         plot(XYlm)
```

```
Call:
lm(formula = Y ~ X)

Residuals:
   1    2    3    4    5
-0.2  0.9 -1.0  0.1  0.2

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   3.2000     0.6164   5.191   0.0139 *
X             0.9000     0.2517   3.576   0.0374 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7958 on 3 degrees of freedom
Multiple R-squared:   0.81,Adjusted R-squared:  0.7467
F-statistic: 12.79 on 1 and 3 DF,  p-value: 0.03739
```



Here is a longer set of examples fron Fernando Perez that demonstrates the `%%julia` magic.

```
In [65]: nbviewer('raw.github.com/JuliaLang/IJulia.jl/master/python/doc/JuliaMagic.ipynb')
```

```
Out[65]: <IPython.core.display.HTML at 0x10338bcd0>
```

2.1.3 Notebook documents

Notebook documents are JSON files that store code, output, text, images, HTML, etc. The Notebook as a whole and individual cells have metadata associated with them. This metadata can be used to declare which programming language the code cells are in. **Thus, Notebook documents are multi-lingual.**

2.1.4 nbviewer/nbconvert

Both nbviewer and nbconvert work (mostly) with Notebook written in other languages.

```
In [66]: nbviewer('jdj.mit.edu/~stevenj/IJulia%2520Preview.ipynb')
```

```
Out[66]: <IPython.core.display.HTML at 0x10338be10>
```

2.2 Attribute #3: Open

Openness is built into the project at a deep level. This attribute builds a dedicated and invested community of users and developers that have good feelings about the project. It also blurs the line betweeen users and developers in a good way by providing a single community that spans all project activities.

2.2.1 Open source

The IPython source code is released under the terms of the BSD license. This license allows anyone to use the software, without restrictions for any purpose, commercial or otherwise. Companies are building commercial products based on IPython:

- Wakari by Continuum Analytics
- Anaconda by Continuum Analytics
- Canopy by Enthought
- PiCloud

A wide range of open-source projects and scientific also build on top of IPython.

Other open-source licenses, such as the GPL are more restrictive and would not allow for this extended ecosystem to grow and flourish.

2.2.2 Open process

The entire development process of IPython is carried out in the open:

- All contributions are handled through GitHub Pull Requests and reviewed thoroughly. All review comments are public. Even contributions from the most senior developers are subject to code review.
- All Issues are reported and discussed publicly through GitHub.
- All real time discussion are done using public HipChat rooms (nice web-based IRC chat).
- We maintain a public Roadmap that describes where the project is headed.
- We hold weekly "lab meetings" on Google Hangouts to discuss the project with core developers and guests. The minutes and agenda for these meetings are maintained on a public Hackpad. These meetings are broadcast publicly and recorded.

We have found that an open process doesn't happen automatically:

- It is really tempting for developers, especially those in physical proximity, to rely on private discussions.
- We try to minimize these private discussions as much as possible, but they do still happen.
- We try to always post the results of private discussions on our mailing lists or GitHub.

This open process is a key part of our community building.

2.2.3 Open standards

It is really easy, and sometimes tempting, to invent your own:

- Serialization format
- Domain specific language (for configuration, etc.)
- Network protocols
- Document formats

As much as possible, we try to use established, open standards:

- Notebooks documents are just JSON data and can be read in any language with a JSON parser.
- The actual data stored in the JSON document is also in open standards (HTML, Markdown, PNG, JPEG).

- The Notebook is styled using CSS
- Our message architecture is built on top of JSON, ZeroMQ and WebSockets.

In a few cases we have had to invent our own standards on top of these base standards:

- Notebook format (how we store data in the JSON).
- Kernel message architecture.

We are attempting to make these standards documented, tested, versioned, etc. The board adoption of the IPython Notebook by other language communities would not be possible without this openness.

2.3 Attribute #5: Close to data

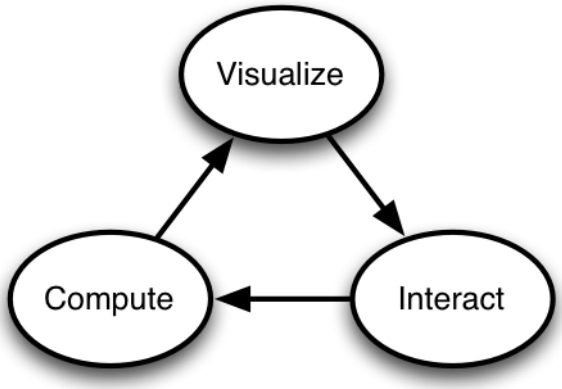
All computations in scientific computing and data science involve data. How many

Software tools that put the data close to users: Microsoft Excel, d3 Data far from users: C, C++, Fortran

2.3.1 See, Interact, Compute

```
In [68]: Image('images/VizInteractCompute.png')
```

Out[68]:




```
In [69]: %load_ext load_style
```

```
In [73]: %load_style talk.css
```

```
<IPython.core.display.HTML at 0x104da5550>
```

```
In []:
```