An Introduction to the PyData World

Jake VanderPlas @jakevdp

Index Conf 2018

```
$ whoami
jakevdp
```

```
$ whoami
jakevdp
```

```
$ whoami
jakevdp
```

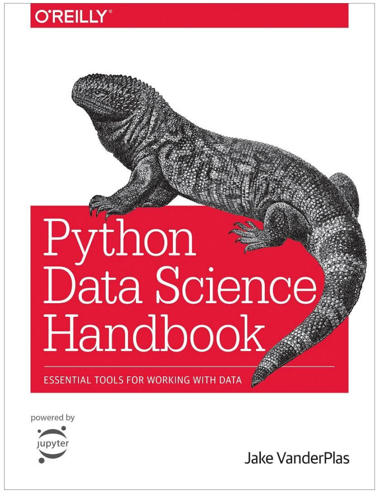
Blog: http://jakevdp.github.io

Code:

scikit learn

astropy

Books:



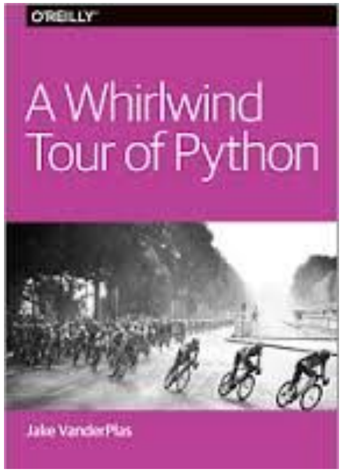
O'REILLY

Python Data Science Handbook

ESSENTIAL TOOLS FOR WORKING WITH DATA

powered by jupyter

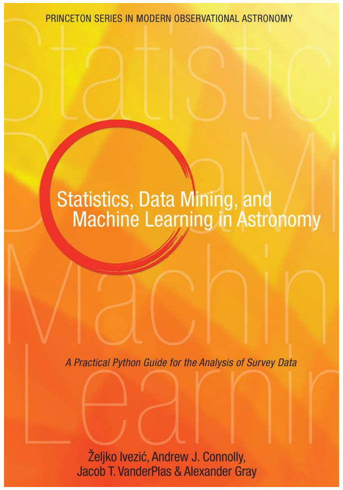
Jake VanderPlas



O'REILLY

A Whirlwind Tour of Python

Jake VanderPlas



PRINCETON SERIES IN MODERN OBSERVATIONAL ASTRONOMY

Statistics, Data Mining, and Machine Learning in Astronomy

A Practical Python Guide for the Analysis of Survey Data

Željko Ivezić, Andrew J. Connolly, Jacob T. VanderPlas & Alexander Gray

History: how Python led to PyData

~

Tools: Getting to know the landscape

Python is not a data science language.

python

Python was created in the 1980s as a teaching language, and to “bridge the gap between the shell and C” [1]

[1] Guido Van Rossum The Making of Python

“I thought we'd write small Python programs, maybe 10 lines, maybe 50, maybe 500 lines — that would be a big one”

Guido Van Rossum The Making of Python

How did Python become a data science powerhouse?

1990s: The Scripting Era

* yes, this is overly simplified . .

1990s: The Scripting Era

Motto: “Python as Alternative to Bash”

* yes, this is overly simplified . .

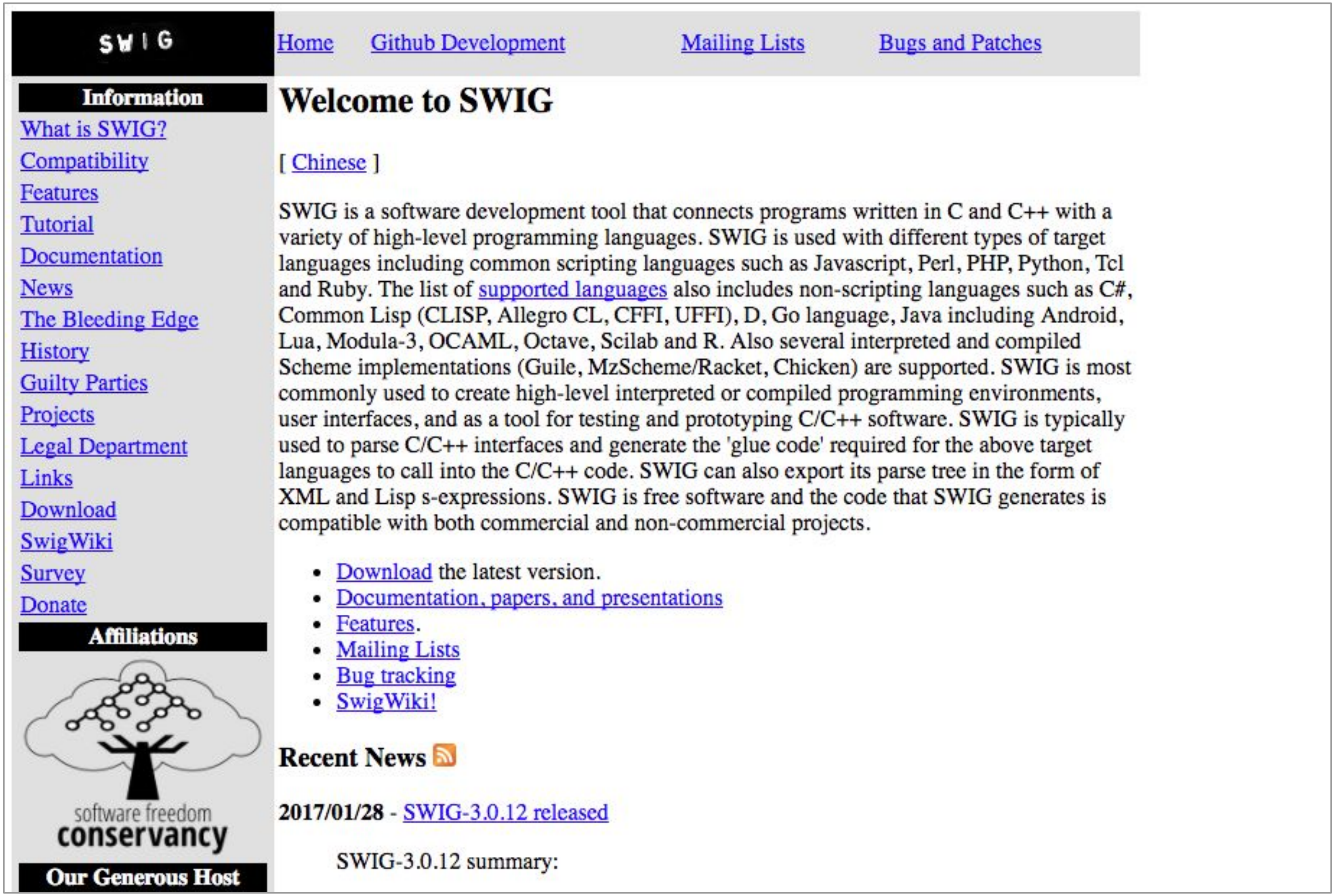
1990s: The Scripting Era

"Scientists... work with a wide variety of systems ranging from simulation codes, data analysis packages, databases, visualization tools, and home-grown software-each of which presents the user with a different set of interfaces and file formats. As a result, a scientist may spend a considerable amount of time simply trying to get all of these components to work together in some manner..."

- **David Beazley**
  Scientific Computing with Python
  (ACM vol. 216, 2000)

1990s: The Scripting Era

“Simplified Wrapper and Interface Generator” (SWIG)



SWIG

Home Github Development Mailing Lists Bugs and Patches

Information

- What is SWIG?
- Compatibility
- Features
- Tutorial
- Documentation
- News
- The Bleeding Edge
- History
- Guilty Parties
- Projects
- Legal Department
- Links
- Download
- SwigWiki
- Survey
- Donate

Affiliations

software freedom conservancy

Our Generous Host

Welcome to SWIG

[Chinese]

SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. SWIG is used with different types of target languages including common scripting languages such as Javascript, Perl, PHP, Python, Tcl and Ruby. The list of supported languages also includes non-scripting languages such as C#, Common Lisp (CLISP, Allegro CL, CFFI, UFFI), D, Go language, Java including Android, Lua, Modula-3, OCAML, Octave, Scilab and R. Also several interpreted and compiled Scheme implementations (Guile, MzScheme/Racket, Chicken) are supported. SWIG is most commonly used to create high-level interpreted or compiled programming environments, user interfaces, and as a tool for testing and prototyping C/C++ software. SWIG is typically used to parse C/C++ interfaces and generate the 'glue code' required for the above target languages to call into the C/C++ code. SWIG can also export its parse tree in the form of XML and Lisp s-expressions. SWIG is free software and the code that SWIG generates is compatible with both commercial and non-commercial projects.

- Download the latest version.
- Documentation, papers, and presentations
- Features.
- Mailing Lists
- Bug tracking
- SwigWiki!

Recent News

2017/01/28 - SWIG-3.0.12 released

SWIG-3.0.12 summary:

http://www.swig.org/

1990s: The Scripting Era

2000s: The SciPy Era

* yes, this is overly simplified . .

1990s: The Scripting Era

2000s: The SciPy Era

Motto: “Python as Alternative to MatLab”

* yes, this is overly simplified . .

2000s: The SciPy Era

“I had a hodge-podge of work processes. I would have Perl scripts that called C++ numerical routines that would dump data files, and I would load them up into MatLab to plot them. After a while I got tired of the MatLab dependency… so I started loading them up in GnuPlot.”

-**John Hunter**
creator of Matplotlib
SciPy 2012 Keynote

2000s: The SciPy Era

“Prior to Python, I used Perl (for a year) and then Matlab and shell scripts & Fortran & C/C++ libraries. When I discovered Python, I really liked the language... But, it was very nascent and lacked a lot of libraries. I felt like I could add value to the world by connecting low-level libraries to high-level usage in Python.”

- **Travis Oliphant**
  creator of NumPy & SciPy
  via email, 2015

2000s: The SciPy Era

"I remember looking at my desk, and seeing all the books on languages I had. I literally had a stack with books on C, C++, Unix utilities (awk/sed/sh/etc), Perl, IDL manuals, the Mathematica book, Make printouts, etc. I realized I was probably spending more time switching between languages than getting anything done.."

- **Fernando Perez**
creator of IPython
via email, 2015

2000s: The SciPy Era

Key Software Development:

matplotlib *Released circa 2002*

SciPy *Released circa 2000*

IP[y]: *Released circa 2001*

Numarray *1995*
Numeric *2002*
(Early array libraries)

2000s: The SciPy Era

Originally, the three projects each had much wider scope:

	Visualization	Computation	Shell
matplotlib	✓	✓	✓
SciPy	✓	✓	✓
IP[y]:	✓	✓	✓

Numarray
Numeric

Array Manipulation

2000s: The SciPy Era

With time, the projects narrowed their focus:

	Visualization	Computation	Shell
matplotlib	✓		
SciPy		✓	
IP[y]:			✓

NumPy *Unified Array Library Underneath*

1990s: The Scripting Era

2000s: The SciPy Era

2010s: The PyData Era

* yes, this is overly simplified . .

1990s: The Scripting Era

2000s: The SciPy Era

2010s: The PyData Era

Motto: “Python as Alternative to R”

* yes, this is overly simplified . .

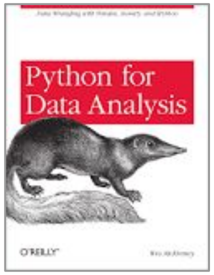
2010s: The PyData Era

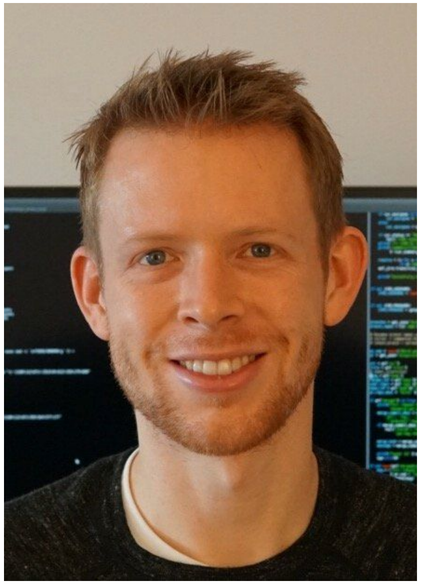
"I had a distinct set of requirements that were not well-addressed by any single tool at my disposal:

- Data structures with labeled axes . . .
- Integrated time series functionality . . .
- Arithmetic operations and reductions . . .
- Flexible handling of missing data
- Merge and other relational operations . . .

I wanted to be able to do all these things in one place, preferably in a language well-suited to general purpose software development"

- **Wes McKinney**
  creator of Pandas
  (in *Python for Data Analysis*)





2010s: The PyData Era

Key Software Development:

pandas $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$ — 2011: *Labeled data*

scikit learn — 2010: *Machine Learning*

CONDA — 2012: *Packaging*

IP[y]: Notebook — 2012: *Compute Environment*

jupyter — 2015: *polyglot notebook*

1990s: The Scripting Era

Motto: “Python as Alternative to Bash”

2000s: The SciPy Era

Motto: “Python as Alternative to MatLab”

2010s: The PyData Era

Motto: “Python as Alternative to R”

* yes, this is *all* overly simplified . . .

People *want* to use Python because of its intuitiveness, beauty, philosophy, and readability.

People *want* to use Python because of its intuitiveness, beauty, philosophy, and readability.

So people build Python packages that incorporate lessons learned in other tools & communities.

A Quick Tour of the PyData World . . .

Installation

Conda is a cross-platform package and dependency manager, focused on Python for scientific and data-intensive computing,

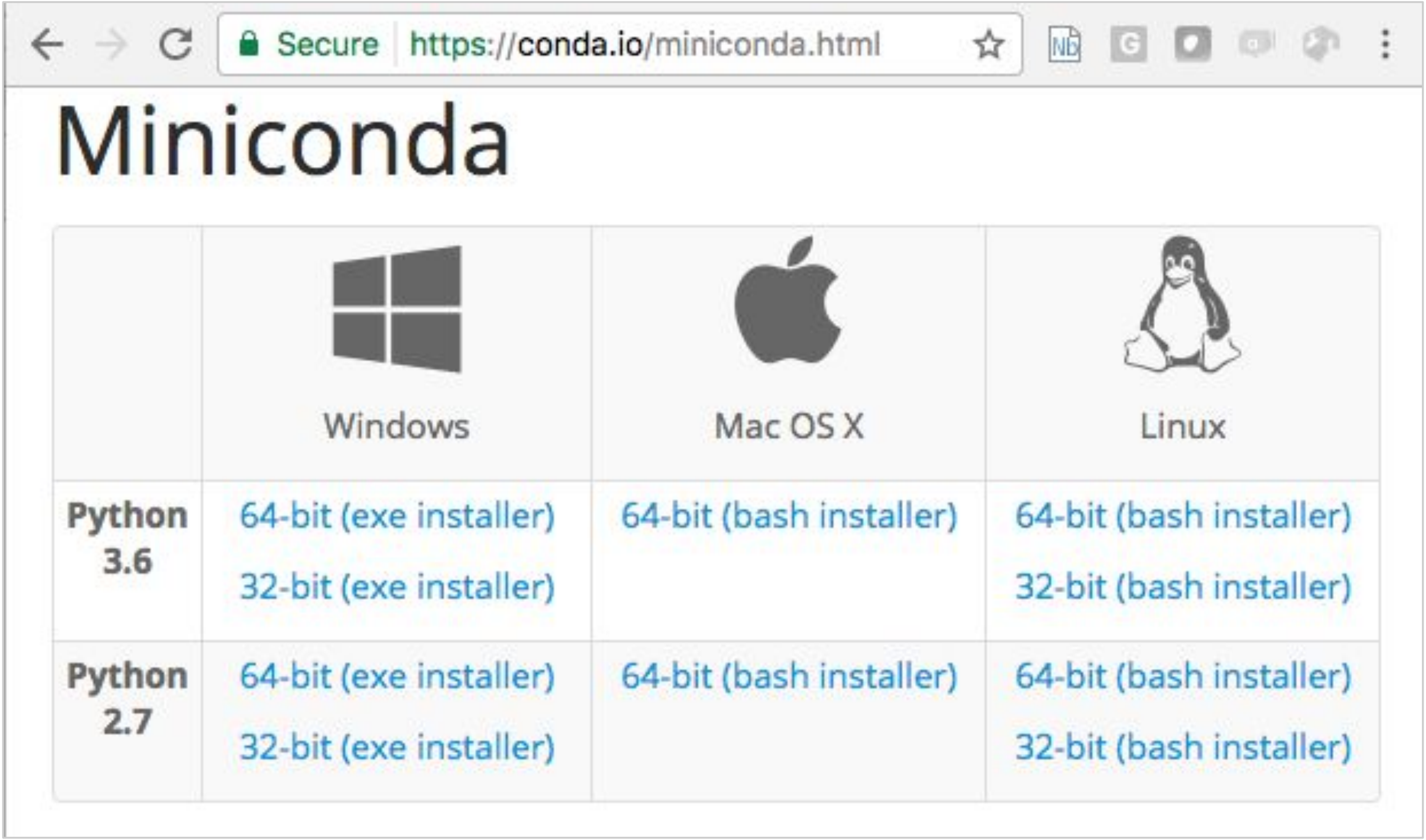
It comes in two flavors:

- *Miniconda* is a minimal install of the conda command-line tool
- *Anaconda* is miniconda plus hundreds of common packages.

I recommend Miniconda.

http://conda.pydata.org/

Installation



Secure https://conda.io/miniconda.html

Miniconda

	Windows	Mac OS X	Linux
Python 3.6	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer)	64-bit (bash installer) 32-bit (bash installer)
Python 2.7	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer)	64-bit (bash installer) 32-bit (bash installer)

Anaconda and Miniconda are both available for a wide range of operating systems.

http://conda.pydata.org/

Installation

```
$ bash ~/Downloads/Miniconda3-latest-MacOSX-x86_64.sh

Welcome to Miniconda3 4.3.21 (by Continuum Analytics, Inc.)

In order to continue the installation process, please review
the license
agreement.
Please, press ENTER to continue
>>>
```

Miniconda is a lightweight installation (~25MB) that gives you access to the **conda** package management tool. It creates a sandboxed Python installation, entirely disconnected from your system Python.

http://conda.pydata.org/

Installation

```
$ which conda
/Users/jakevdp/anaconda/bin/conda

$ which python
/Users/jakevdp/anaconda/bin/python

$ python
Python 3.5.1 |Continuum Analytics, Inc.| (default ...
Type "help", "copyright", "credits" or "license" ...
>>> print("hello world")
hello world
```

Both **conda** and **python** now point to the executables installed by miniconda.

http://conda.pydata.org/

Installation

```
$ conda install numpy scipy pandas matplotlib jupyter
Fetching package metadata .........
Solving package specifications: .

Package plan for installation in environment
/Users/jakevdp/anaconda/:

The following NEW packages will be INSTALLED:

    appnope:              0.1.0-py36_0
    bleach:               1.5.0-py36_0
    cycler:               0.10.0-py36_0
    decorator:            4.0.11-py36_0
```

Installation of new packages can be done seamlessly with `conda install`

http://conda.pydata.org/

Installation

```
$ conda create -n py2.7 python=2.7 numpy=1.13 scipy
Fetching package metadata .........
Solving package specifications: .

Package plan for installation in environment
/Users/jakevdp/anaconda/envs/py2.7:

The following NEW packages will be INSTALLED:

    mkl:          2017.0.3-0
    numpy:        1.13.0-py27_0
    openssl:      1.0.2l-0
    pip:          9.0.1-py27_1
```

New sandboxed environments can be created with specific versions of Python and its packages. Here we create an environment named **`py2.7`** with Python 2.7

http://conda.pydata.org/

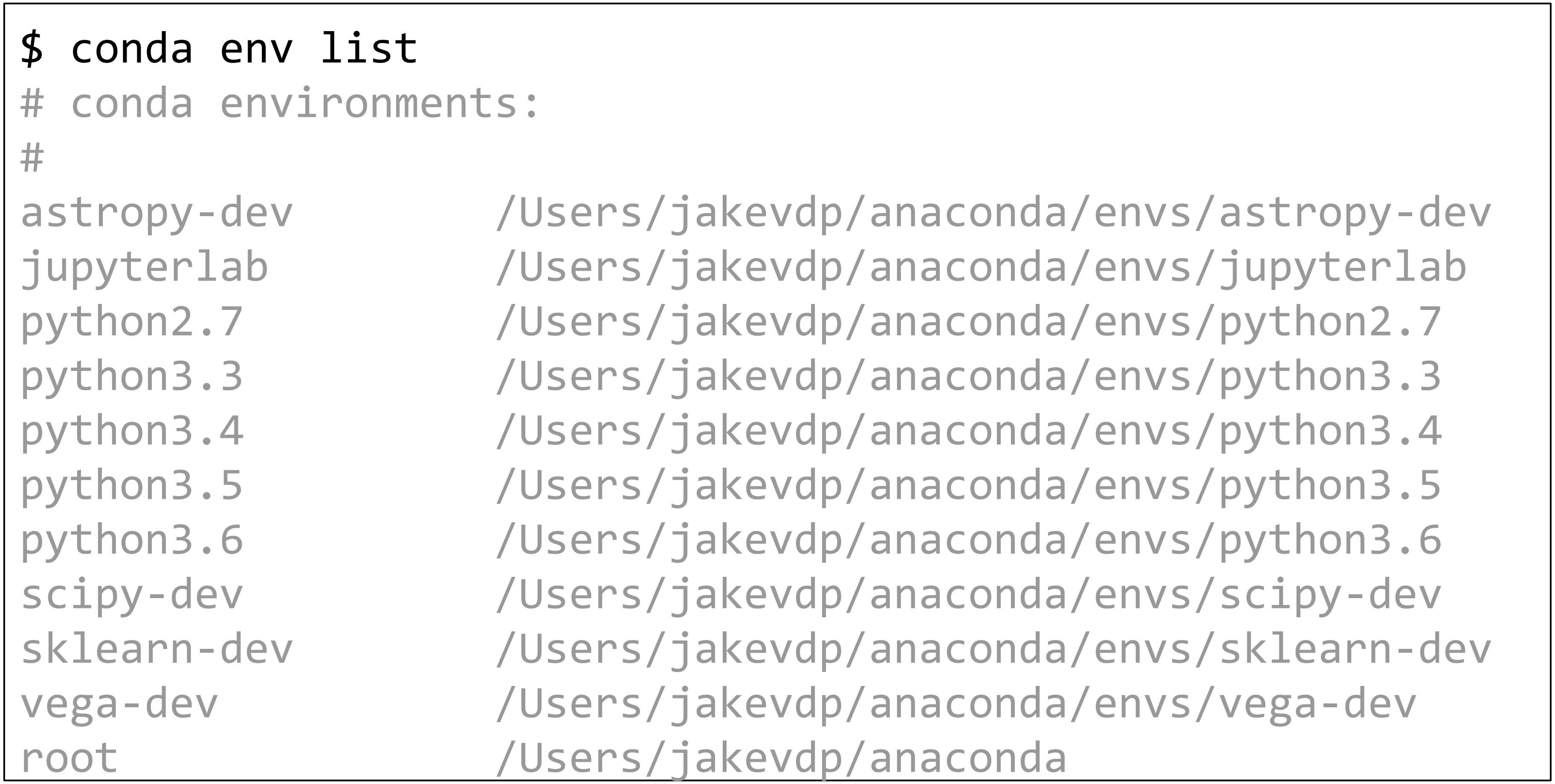
Installation

```
$ conda activate python2.7

(python2.7) $ which python
/Users/jakevdp/anaconda/envs/python2.7/bin/python

(python2.7) $ python --version
Python 2.7.11 :: Continuum Analytics, Inc.
```

By “activating” the environment, we can now use this different Python version with a different set of packages. You can create as many of these environments as you’d like.

http://conda.pydata.org/

Installation



```
$ conda env list
# conda environments:
#
astropy-dev        /Users/jakevdp/anaconda/envs/astropy-dev
jupyterlab         /Users/jakevdp/anaconda/envs/jupyterlab
python2.7          /Users/jakevdp/anaconda/envs/python2.7
python3.3          /Users/jakevdp/anaconda/envs/python3.3
python3.4          /Users/jakevdp/anaconda/envs/python3.4
python3.5          /Users/jakevdp/anaconda/envs/python3.5
python3.6          /Users/jakevdp/anaconda/envs/python3.6
scipy-dev          /Users/jakevdp/anaconda/envs/scipy-dev
sklearn-dev        /Users/jakevdp/anaconda/envs/sklearn-dev
vega-dev           /Users/jakevdp/anaconda/envs/vega-dev
root               /Users/jakevdp/anaconda
```

I tend to use conda envs for just about everything, particularly when testing development versions of projects I contribute to.

http://conda.pydata.org/

Installation

So… what about `pip`?

In brief:

> “`pip` installs *python* packages within *any* environment;
> `conda` installs *any* package within *conda* environments”

For many more details on the distinctions, see my blog post, *Conda: Myths and Misconceptions.*[1]

[1] https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/

Coding Environment:

```
$ conda install jupyter notebook
```

http://jupyter.org/

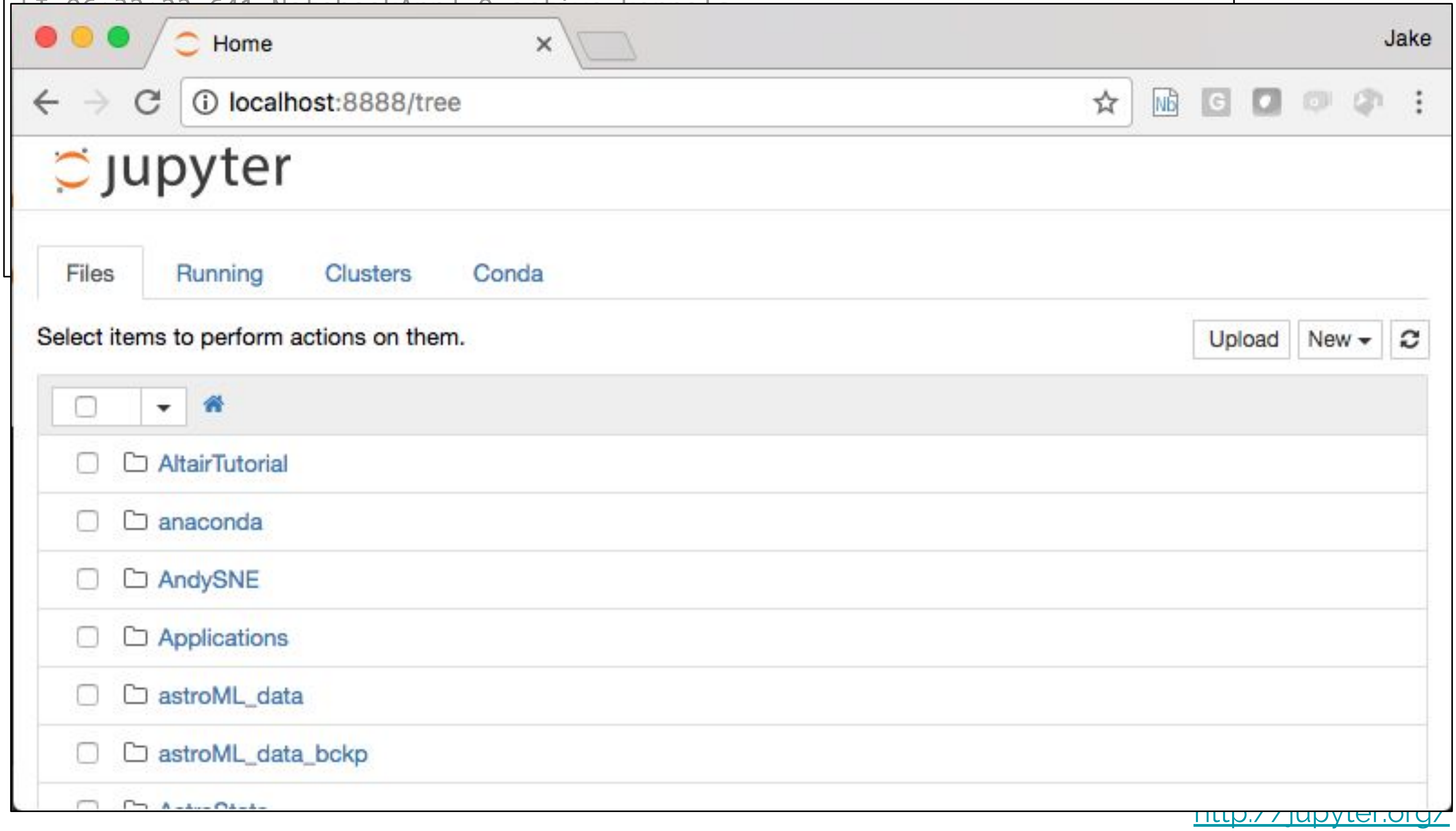
Coding Environment:

```
$ jupyter notebook
[I 06:32:22.641 NotebookApp] Serving notebooks from local directory:
/Users/jakevdp
[I 06:32:22.641 NotebookApp] 0 active kernels
[I 06:32:22.641 NotebookApp] The IPython Notebook is running at:
http://localhost:8888/
[I 06:32:22.642 NotebookApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
```

http://jupyter.org/

Coding Environment:

```
$ jupyter notebook
[I 06:32:22.641 NotebookApp] Serving notebooks from local directory:
/Users/jakevdp
```



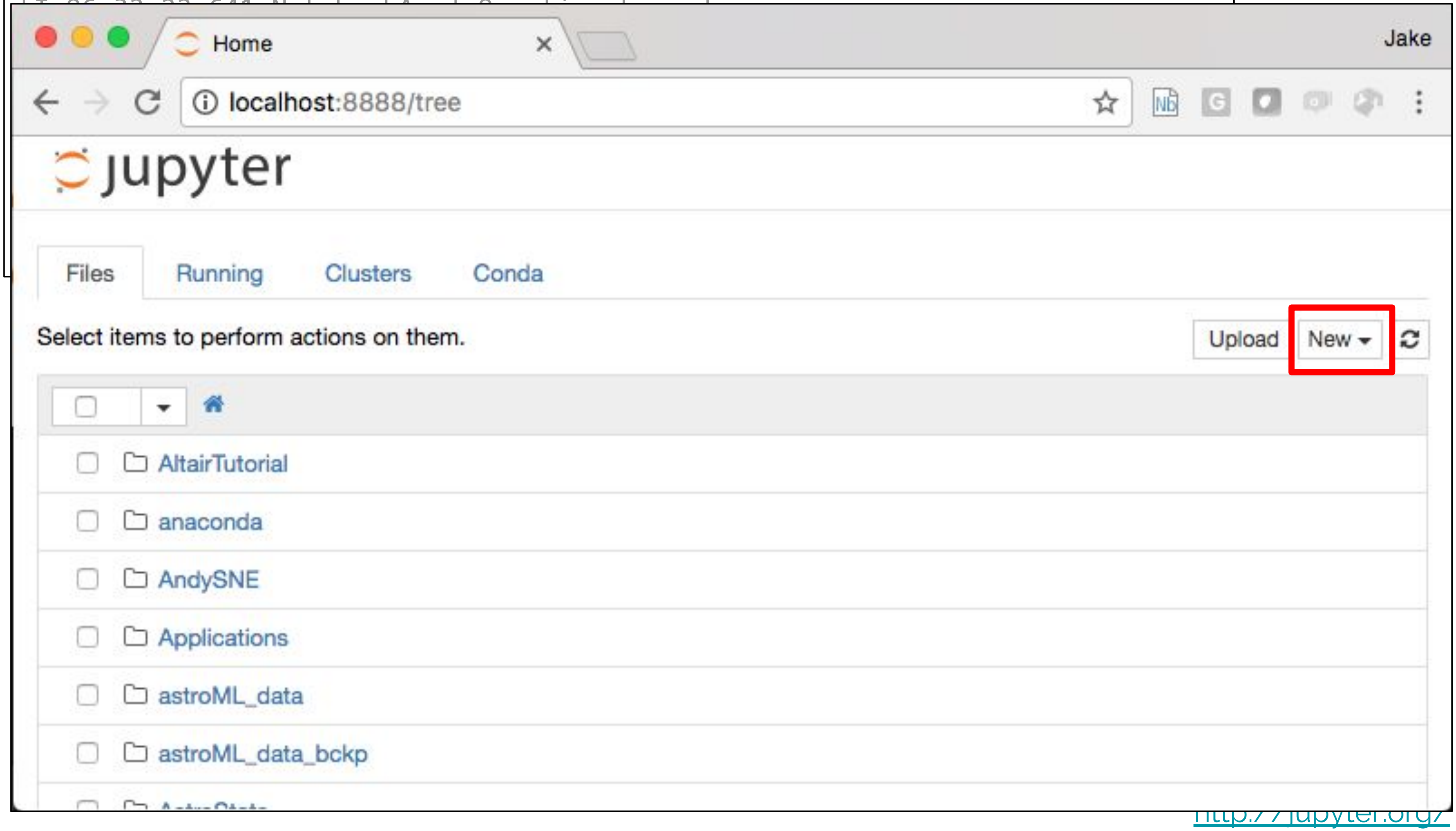
Select items to perform actions on them.

- □ AltairTutorial
- □ anaconda
- □ AndySNE
- □ Applications
- □ astroML_data
- □ astroML_data_bckp

http://jupyter.org/

Coding Environment:

```
$ jupyter notebook
[I 06:32:22.641 NotebookApp] Serving notebooks from local directory:
/Users/jakevdp
```



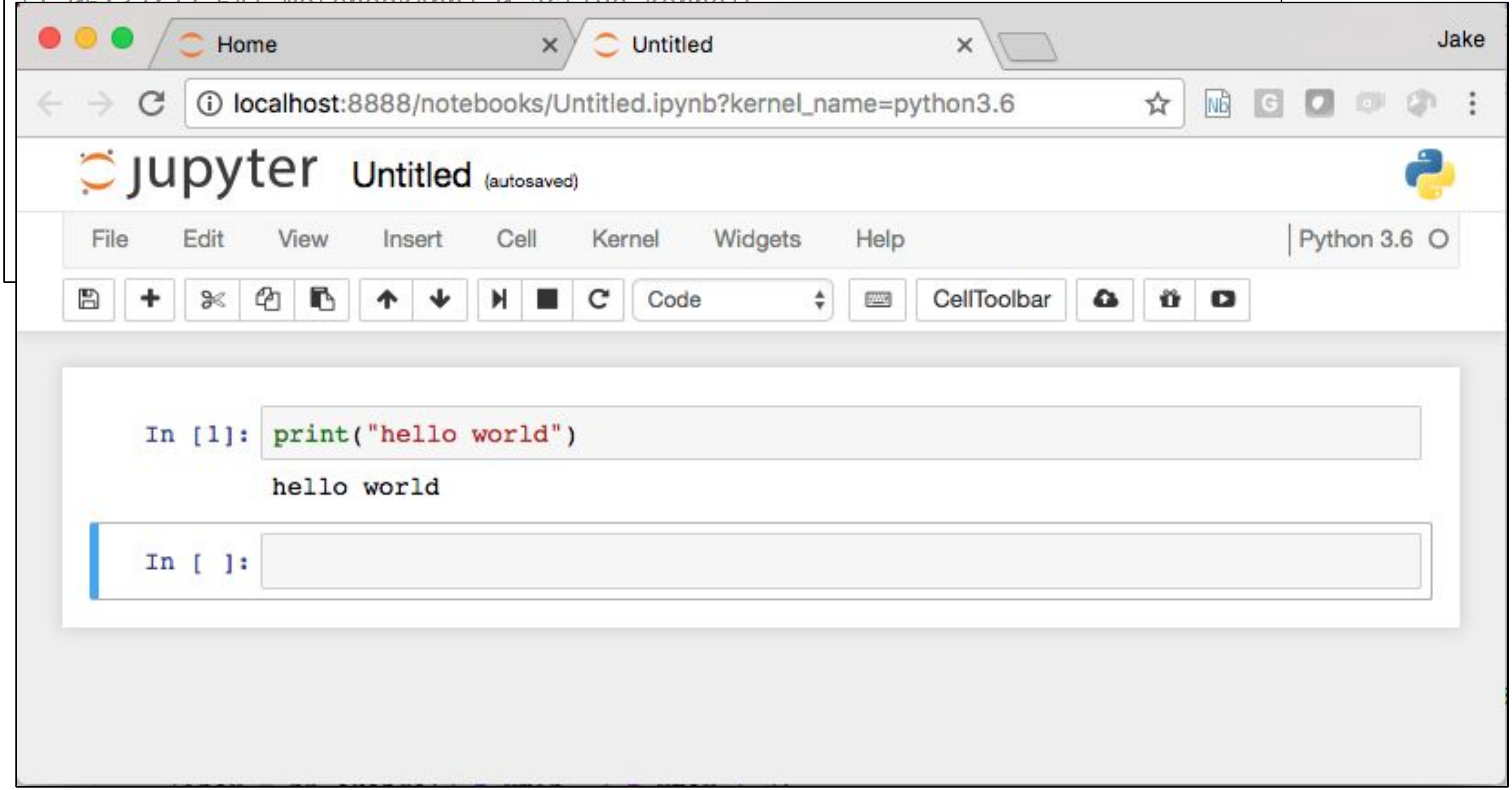
Select items to perform actions on them.

- □ AltairTutorial
- □ anaconda
- □ AndySNE
- □ Applications
- □ astroML_data
- □ astroML_data_bckp

http://jupyter.org/

Coding Environment:

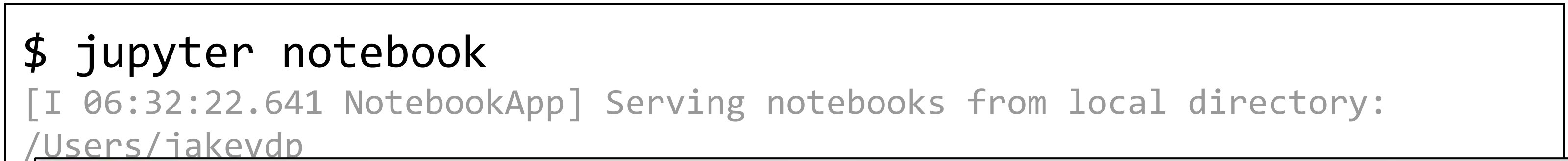
```
$ jupyter notebook
[I 06:32:22.641 NotebookApp] Serving notebooks from local directory:
/Users/jakevdp
[I 06:32:22.641 NotebookApp] 0 active kernels
```



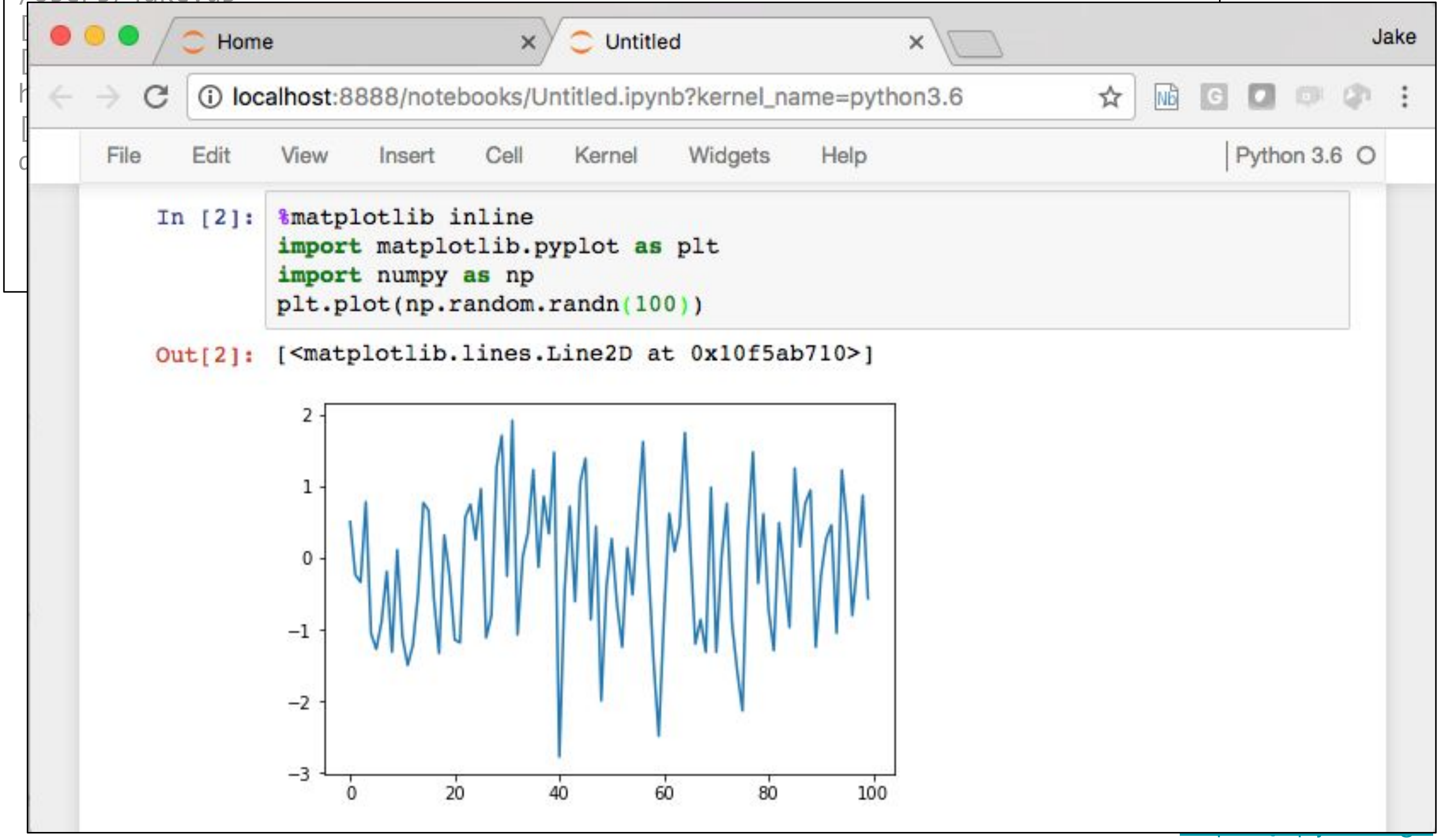
```
In [1]: print("hello world")

        hello world

In [ ]:
```

http://jupyter.org/

Coding Environment:



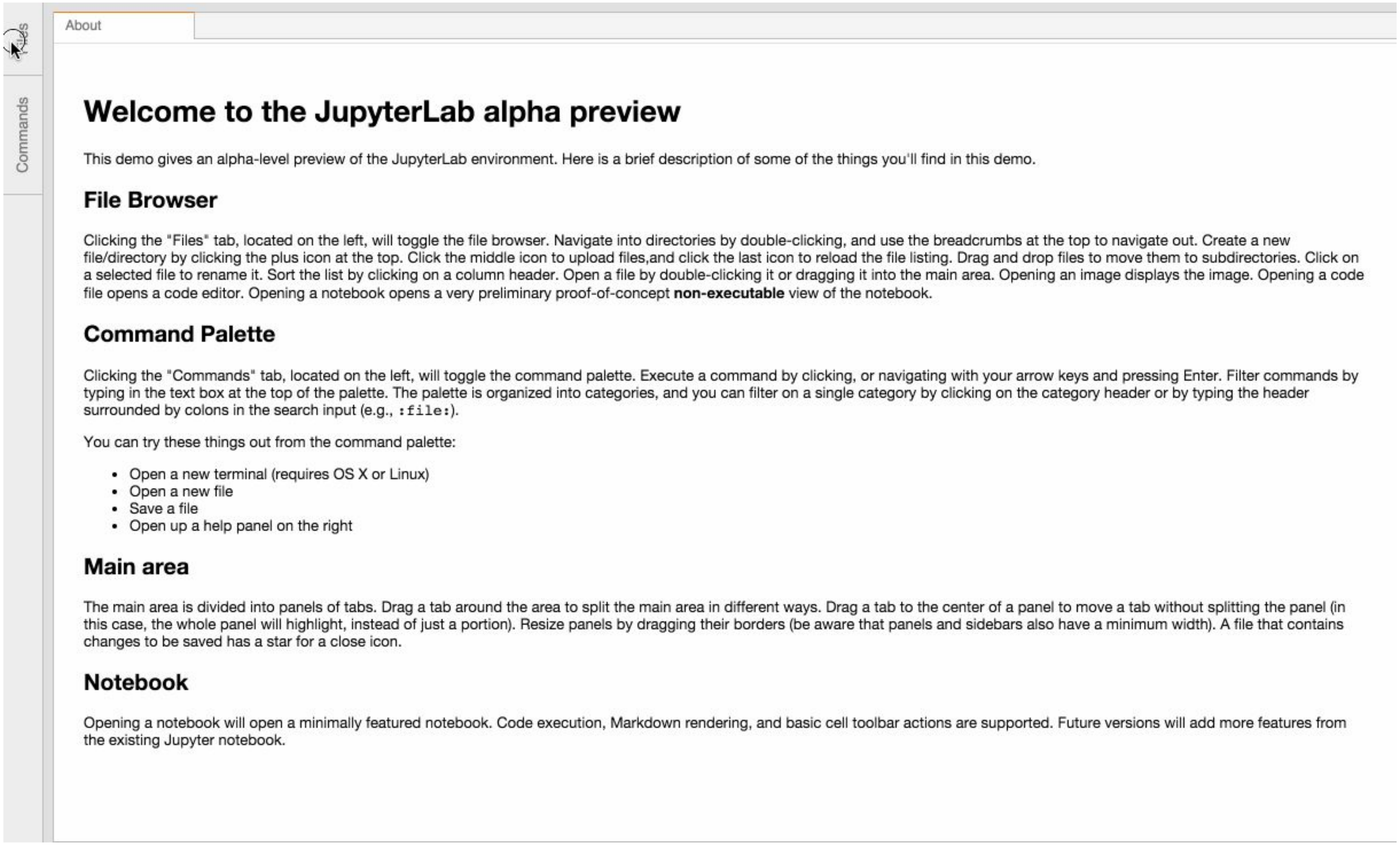
```
$ jupyter notebook
[I 06:32:22.641 NotebookApp] Serving notebooks from local directory:
/Users/jakevdp
[
[
h
[
c
```



```
In [2]: %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        plt.plot(np.random.randn(100))

Out[2]: [<matplotlib.lines.Line2D at 0x10f5ab710>]
```

Coding Environment:

JupyterLab has recently been released: making the notebook one component of a full-featured IDE.



Welcome to the JupyterLab alpha preview

This demo gives an alpha-level preview of the JupyterLab environment. Here is a brief description of some of the things you'll find in this demo.

File Browser

Clicking the "Files" tab, located on the left, will toggle the file browser. Navigate into directories by double-clicking, and use the breadcrumbs at the top to navigate out. Create a new file/directory by clicking the plus icon at the top. Click the middle icon to upload files,and click the last icon to reload the file listing. Drag and drop files to move them to subdirectories. Click on a selected file to rename it. Sort the list by clicking on a column header. Open a file by double-clicking it or dragging it into the main area. Opening an image displays the image. Opening a code file opens a code editor. Opening a notebook opens a very preliminary proof-of-concept **non-executable** view of the notebook.

Command Palette

Clicking the "Commands" tab, located on the left, will toggle the command palette. Execute a command by clicking, or navigating with your arrow keys and pressing Enter. Filter commands by typing in the text box at the top of the palette. The palette is organized into categories, and you can filter on a single category by clicking on the category header or by typing the header surrounded by colons in the search input (e.g., `:file:`).

You can try these things out from the command palette:

- Open a new terminal (requires OS X or Linux)
- Open a new file
- Save a file
- Open up a help panel on the right

Main area

The main area is divided into panels of tabs. Drag a tab around the area to split the main area in different ways. Drag a tab to the center of a panel to move a tab without splitting the panel (in this case, the whole panel will highlight, instead of just a portion). Resize panels by dragging their borders (be aware that panels and sidebars also have a minimum width). A file that contains changes to be saved has a star for a close icon.

Notebook

Opening a notebook will open a minimally featured notebook. Code execution, Markdown rendering, and basic cell toolbar actions are supported. Future versions will add more features from the existing Jupyter notebook.

http://jupyter.org/

Numerical Computation:

```
$ conda install numpy
```

http://www.numpy.org/

Numerical Computation:

NumPy provides the **`ndarray`** object which is useful for storing and manipulating numerical data arrays.

```python
import numpy as np
x = np.arange(10)
print(x)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

Arithmetic and other operations are performed element-wise on these arrays:

```python
print(x * 2 + 1)
```

```
[ 1  3  5  7  9 11 13 15 17 19]
```

http://www.numpy.org/

Numerical Computation:

Also provides essential tools like pseudo-random numbers, linear algebra, Fast Fourier Transforms, etc.

```
M = np.random.rand(5, 10)  # 5x10 random matrix
u, s, v = np.linalg.svd(M)
print(s)
```

```
[ 4.22083   1.091050  0.892570  0.55553   0.392541]
```

```
x = np.random.randn(100)  # 100 std normal values
X = np.fft.fft(x)
print(X[:4])              # first four entries
```

```
[ -7.932434 +0.j        -16.683935 -3.997685j
   3.229016+16.658718j   2.366788-11.863747j]
```

http://www.numpy.org/

Numerical Computation:

Key to using NumPy (and general numerical code in Python) is **vectorization:**

```
x = np.random.rand(10000000)
```

If you write Python like C, you'll have a bad time:

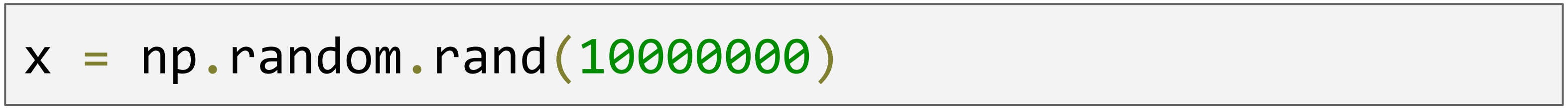
```
%%timeit
y = np.empty(x.shape)
for i in range(len(x)):
    y[i] = 2 * x[i] + 1
```

```
1 loop, best of 3: 6.4 s per loop
```

http://www.numpy.org/

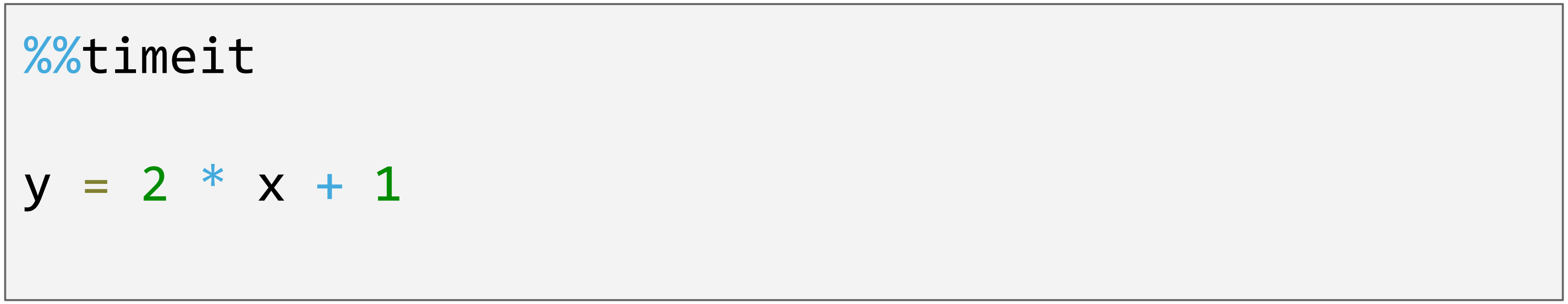
Numerical Computation:

Key to using NumPy (and general numerical code in Python) is **vectorization:**



```
x = np.random.rand(10000000)
```

Use vectorization for *readability* and *speed*



```
%%timeit

y = 2 * x + 1
```

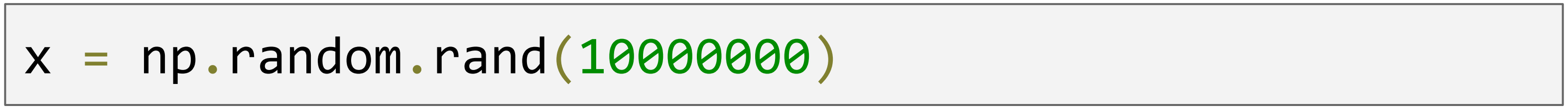
```
10 loops, best of 3: 58.6 ms per loop
```

~ 100x speedup!

http://www.numpy.org/

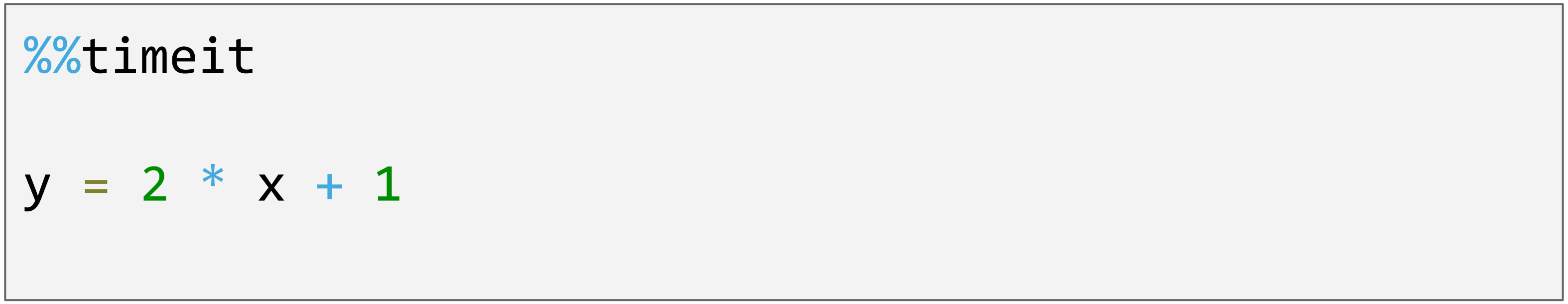
Numerical Computation:

Key to using NumPy (and general numerical code in Python) is **vectorization:**



```
x = np.random.rand(10000000)
```

Use vectorization for *readability* and *speed*



```
%%timeit

y = 2 * x + 1
```

```
10 loops, best of 3: 58.6 ms per loop
```

~ 100x speedup!

For a more complete intro to vectorization in NumPy, see *Losing Your Loops: Fast Numerical Computation in Python* (my talk at PyCon 2015)

https://www.youtube.com/watch?v=EEUXKG97YRw
https://speakerdeck.com/jakevdp/losing-your-loops-fast-numerical-computing-with-numpy-pycon-2015

Labeled Data:

```
$ conda install pandas
```

http://pandas.pydata.org/

Labeled Data:

Pandas provides a **`DataFrame`** object which is like a NumPy array, but has labeled rows and columns:

```python
import pandas as pd
df = pd.DataFrame({'x': [1, 2, 3],
                   'y': [4, 5, 6]})
print(df)
```

```
   x  y
0  1  4
1  2  5
2  3  6
```

http://pandas.pydata.org/

Labeled Data:

Like NumPy, arithmetic is element-wise, but you can access and augment the data using column name:

```
df['x+2y'] = df['x'] + 2 * df['y']
print(df)
```

```
   x  y  x+2y
0  1  4     9
1  2  5    12
2  3  6    15
```

http://pandas.pydata.org/

Labeled Data:

Pandas excels in reading data from disk in a variety of formats. Start here to read virtually any data format!

```
# contents of data.csv
name, id
peter, 321
paul,  605
mary,  444
```

```
df = pd.read_csv('data.csv')
print(df)
```

```
    name   id
0  peter  321
1   paul  605
2   mary  444
```

http://pandas.pydata.org/

Labeled Data:

Pandas also provides fast SQL-like grouping & aggregation:

```
df = pd.DataFrame({'id': ['A', 'B', 'A', 'B'],
                   'val': [1, 2, 3, 4]})
print(df)
```

```
  id  val
0  A    1
1  B    2
2  A    3
3  B    4
```

```
grouped = df.groupby('id').sum()
print(grouped)
```

```
   val
id
A    4
B    6
```

http://pandas.pydata.org/

Visualization:

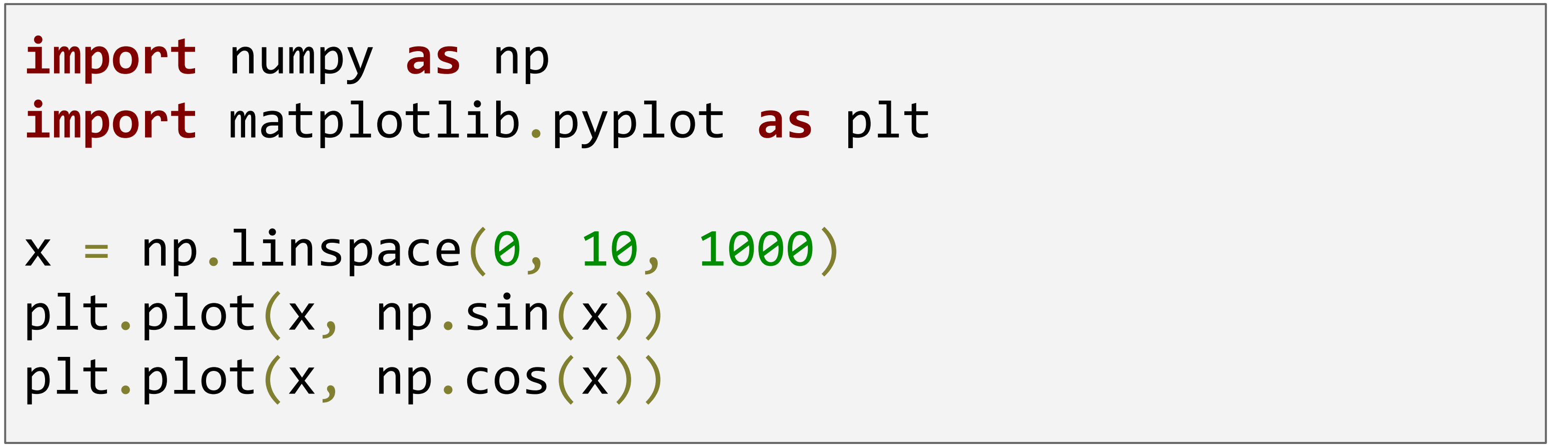
matplotlib

```
$ conda install matplotlib
```

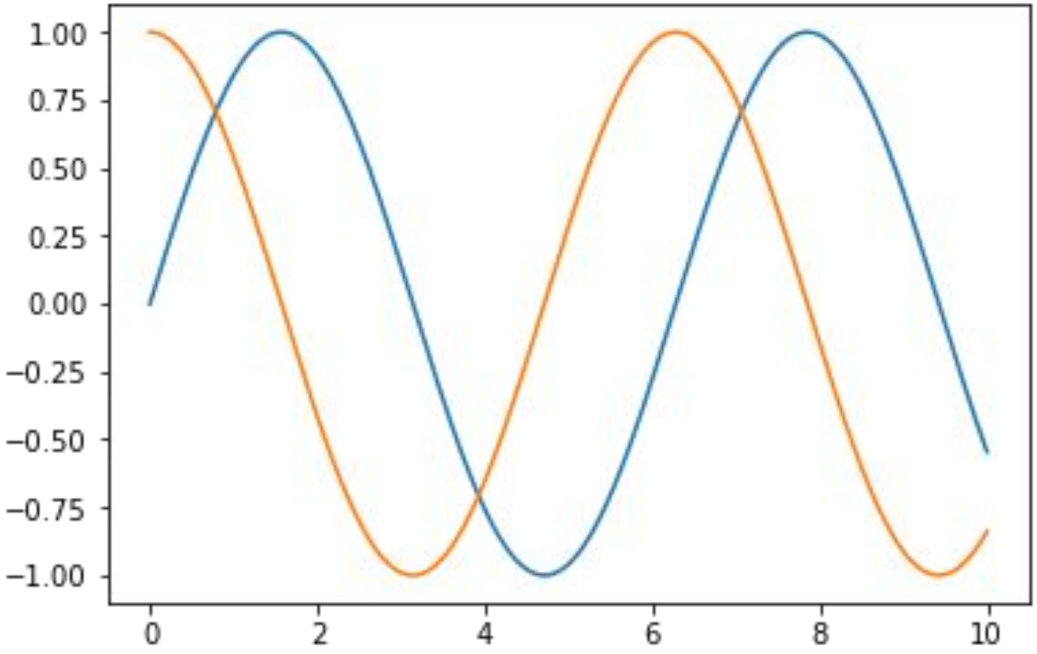
http://www.matplotlib.org/

Visualization:

Matplotlib was developed as a Pythonic replacement for MatLab; thus MatLab users should find it quite familiar:



```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

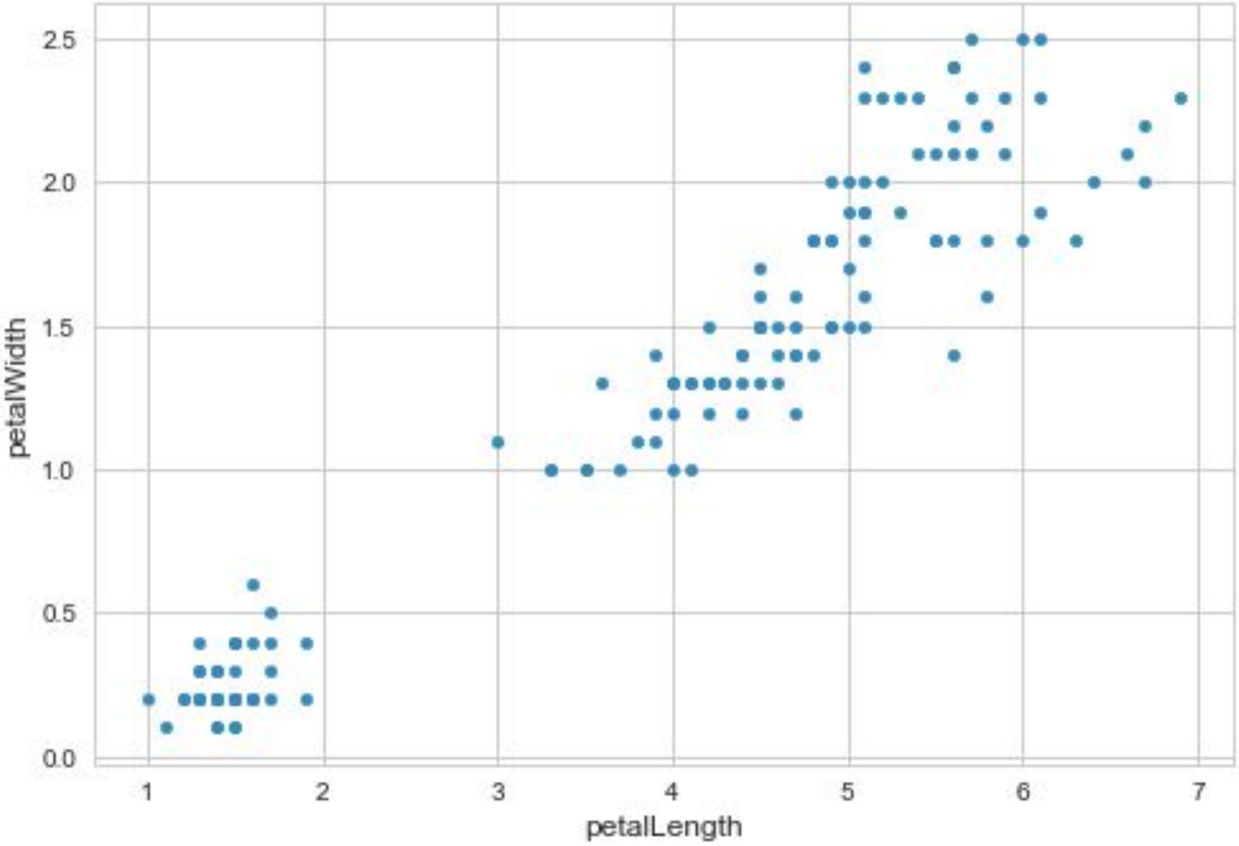


http://www.matplotlib.org/

Visualization Beyond Matplotlib . . .

Pandas offers a simplified Matplotlib Interface:

```
data = pd.read_csv('iris.csv')
data.plot.scatter('petalLength', 'petalWidth')
```

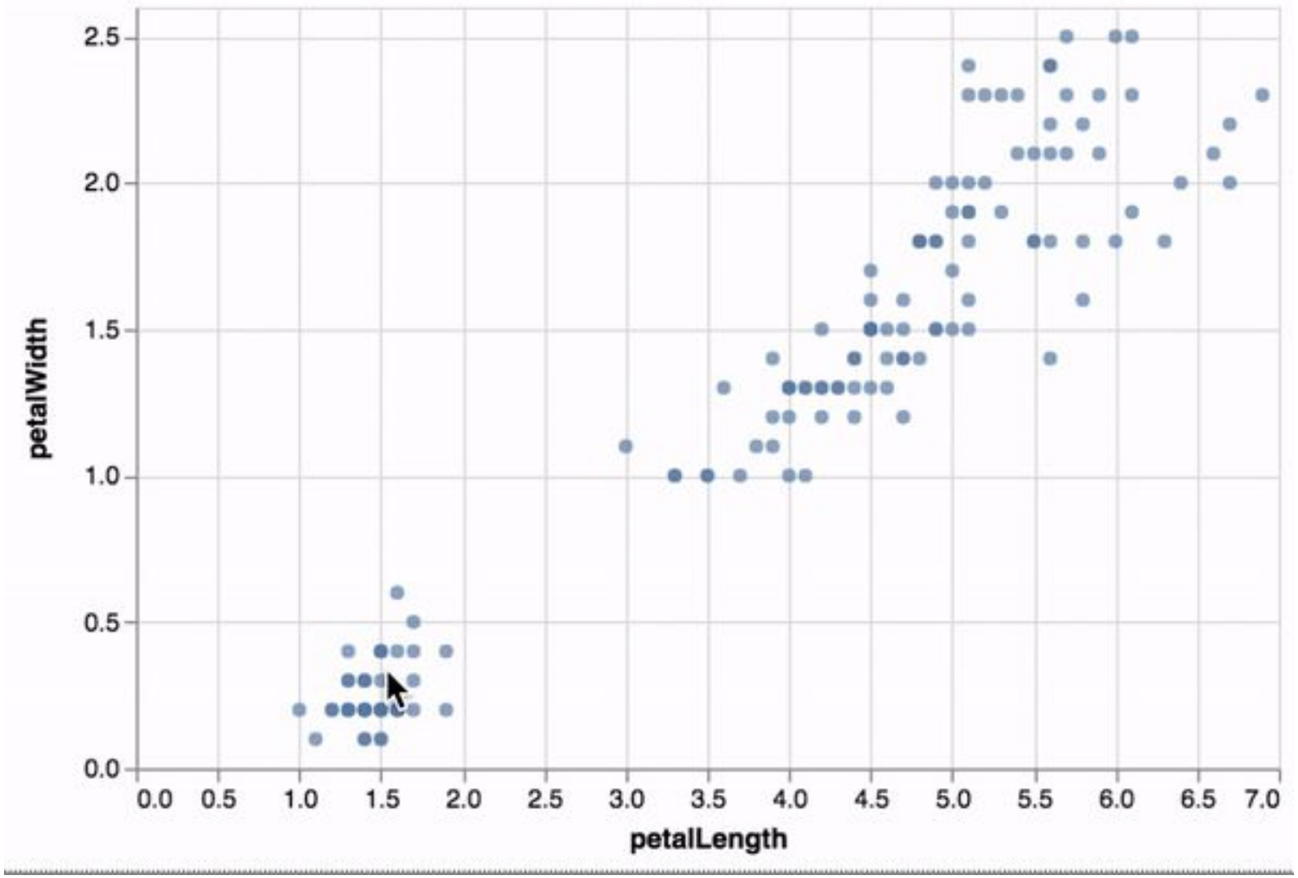


http://pandas.pydata.org/

Visualization Beyond Matplotlib . . .

PdVega gives a similar interface to Vega-Lite:

```
import pdvega  # import makes vgplot attribute available
data.vgplot.scatter('petalLength', 'petalWidth')
```

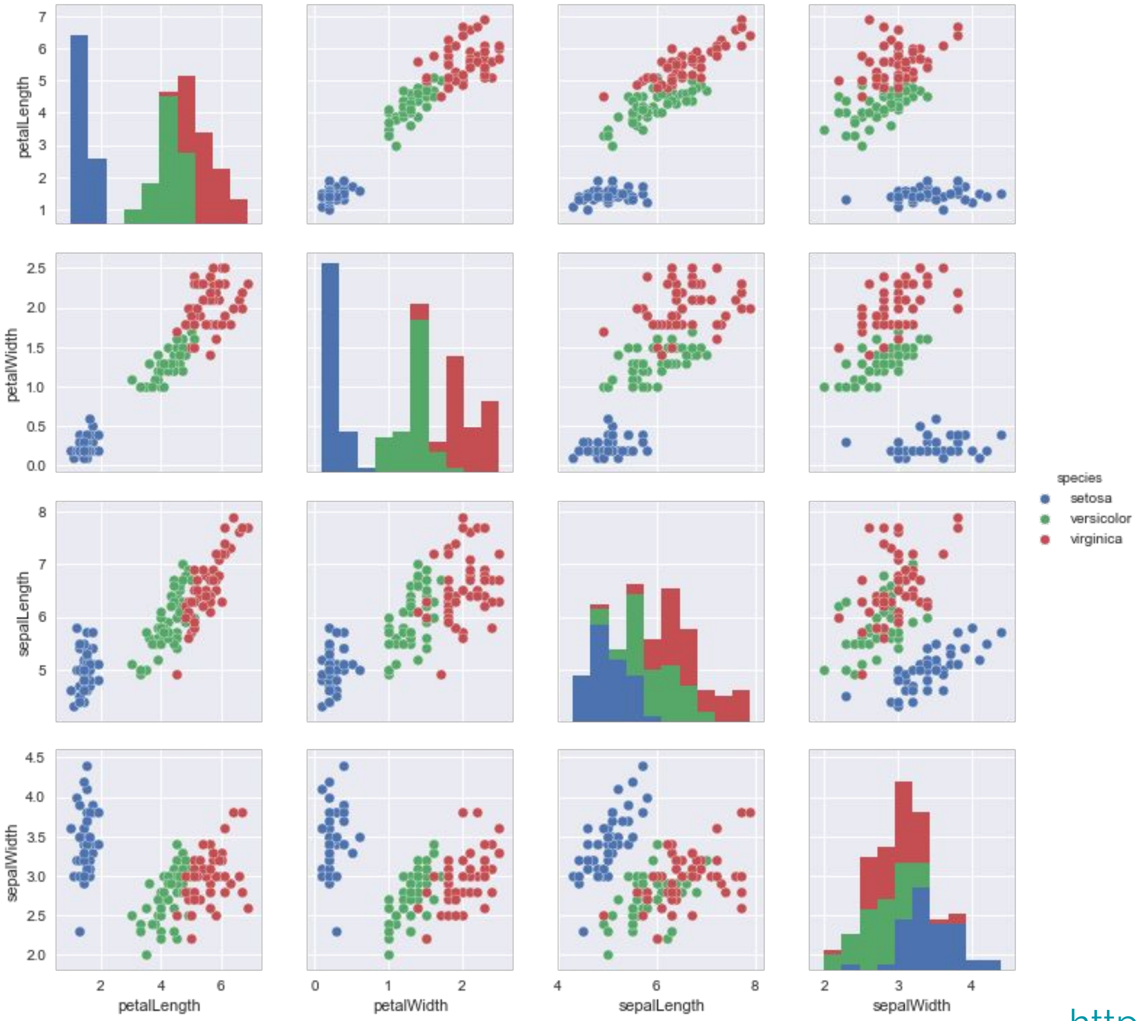


http://jakevdp.github.io/pdvega:/

Visualization Beyond Matplotlib . . .

Seaborn is a package for statistical data visualization

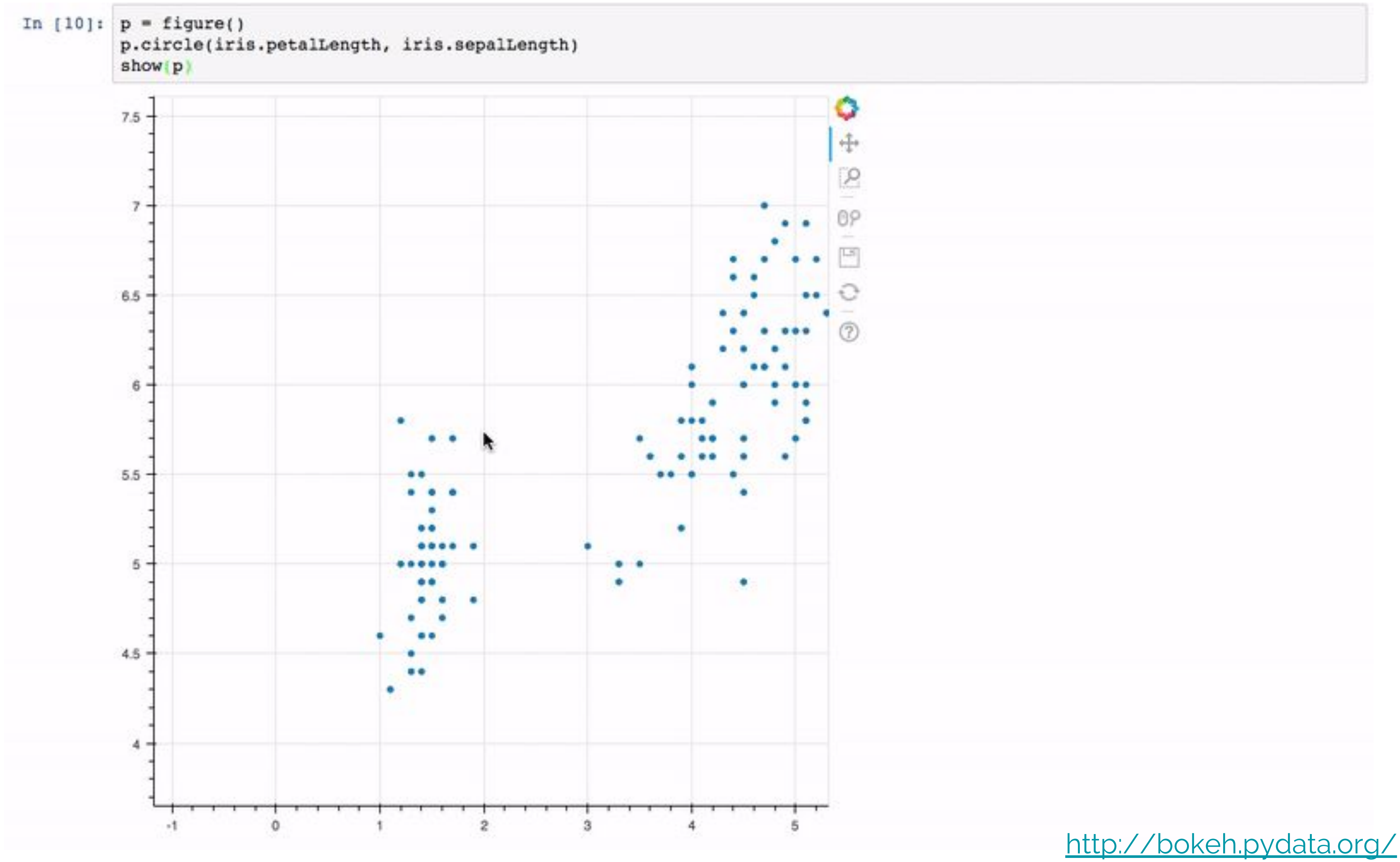
```
seaborn.pairplot(data, hue='species')
```



http://seaborn.pydata.org/

Visualization Beyond Matplotlib . . .

Bokeh: interactive visualization in the browser.

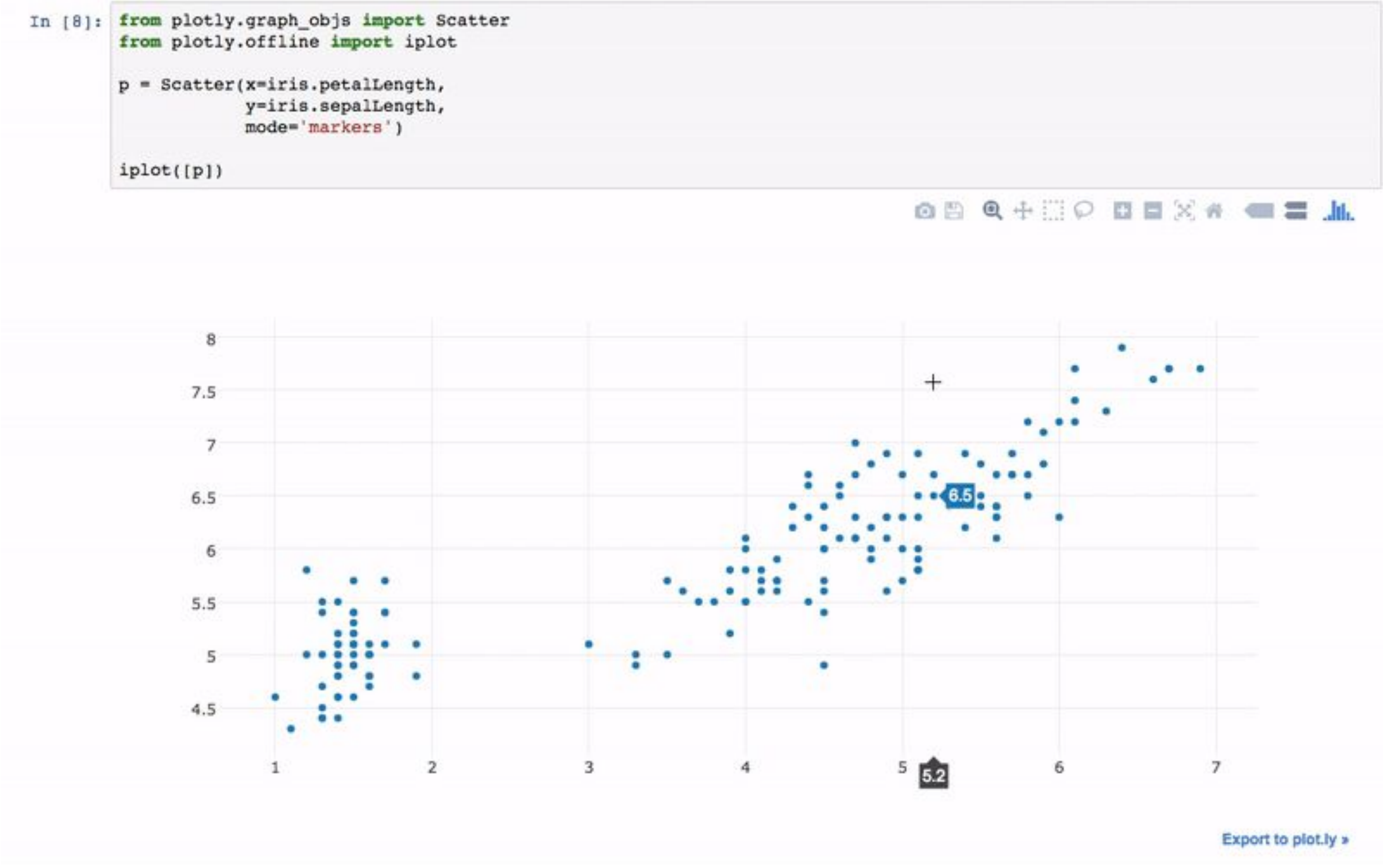


```
In [10]: p = figure()
         p.circle(iris.petalLength, iris.sepalLength)
         show(p)
```

http://bokeh.pydata.org/

Visualization Beyond Matplotlib . . .

Plotly: “modern platform for data science”



In [8]:

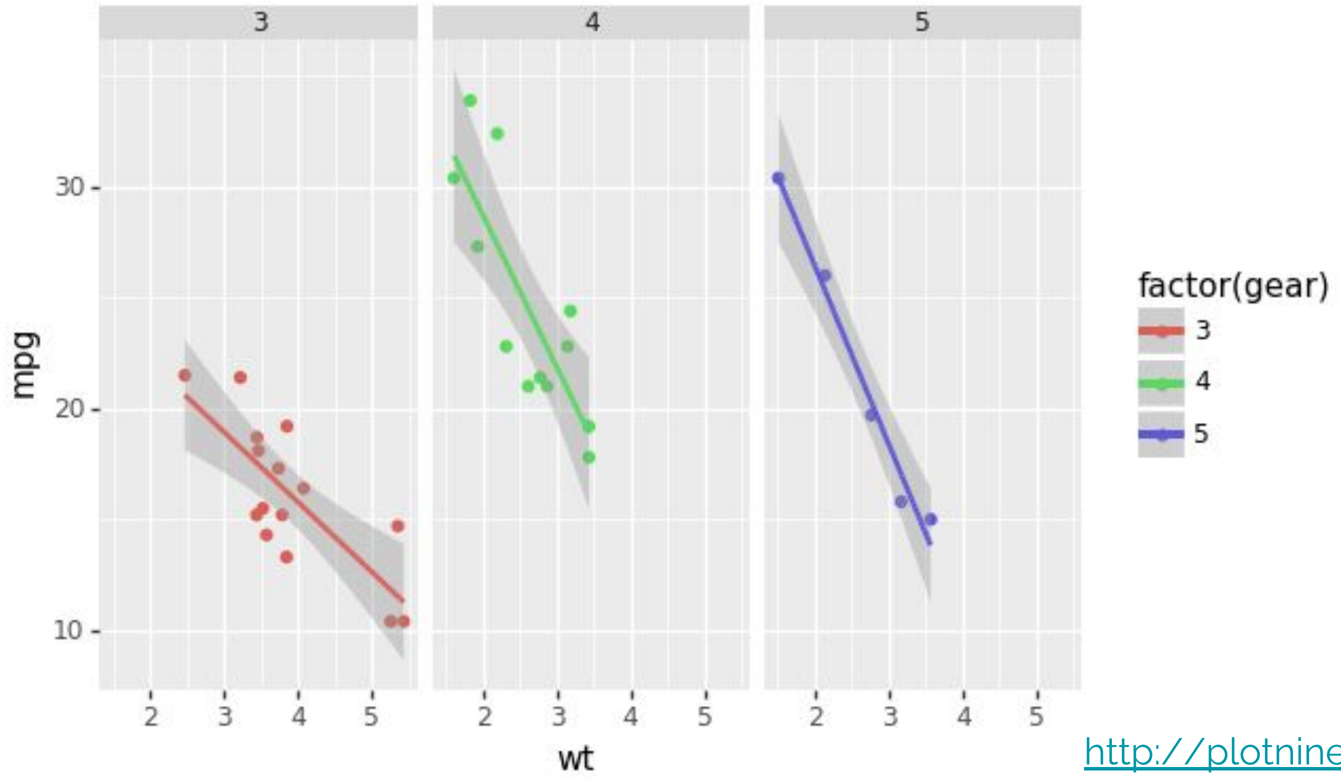
```
from plotly.graph_objs import Scatter
from plotly.offline import iplot

p = Scatter(x=iris.petalLength,
            y=iris.sepalLength,
            mode='markers')

iplot([p])
```

http://plotly.com/

Visualization Beyond Matplotlib . . .

plotnine: grammar of graphics in Python

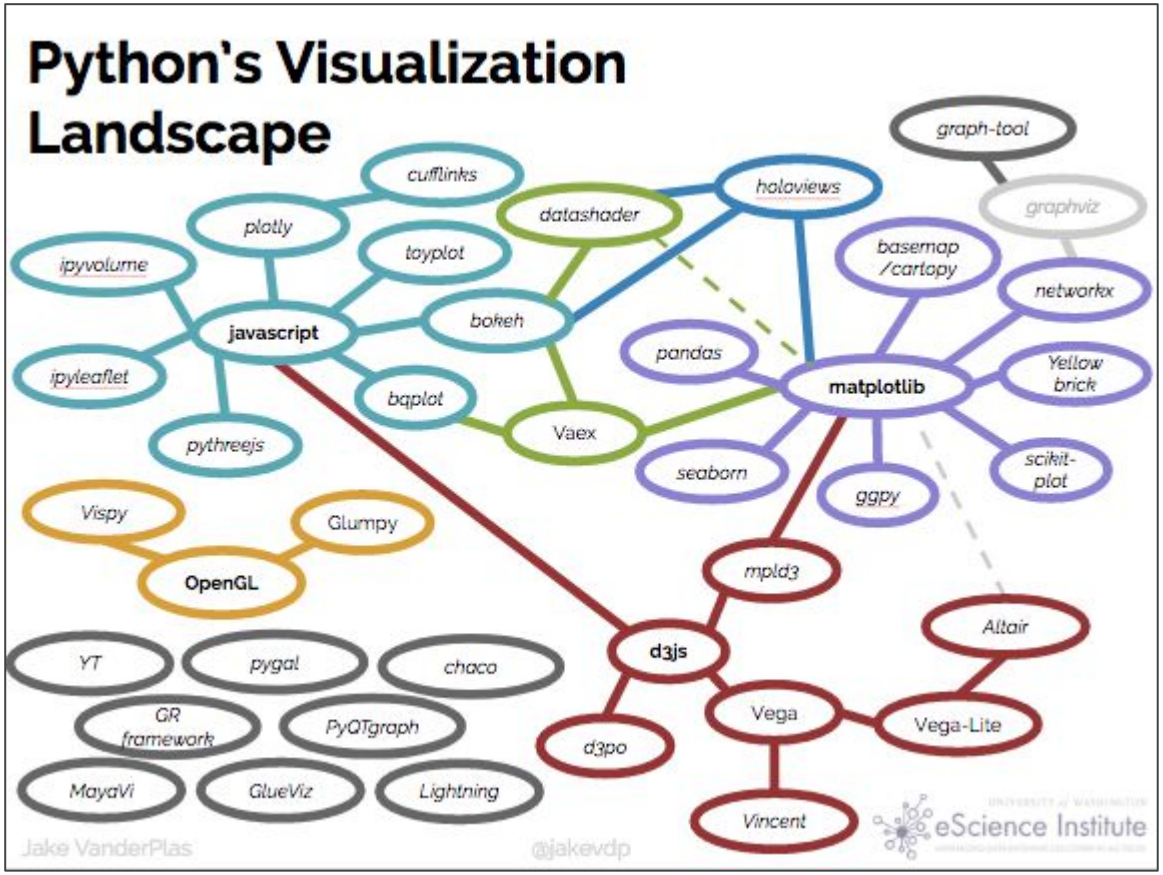
```
(ggplot(mtcars, aes('wt', 'mpg', color='factor(gear)'))
 + geom_point())
 + stat_smooth(method='lm')
 + facet_wrap('~gear'))
```



http://plotnine.readthedocs.io/

Visualization Beyond Matplotlib . . .

Viz in Python is a *huge* and rapidly-developing space:



See my PyCon 2017 talk, *Python's Visualization Landscape*

https://speakerdeck.com/jakevdp/pythons-visualization-landscape-pycon-2017
https://www.youtube.com/watch?v=FytuB8nFHPQ

Numerical Algorithms:

SciPy

```
$ conda install scipy
```

http://www.scipy.org/

Numerical Algorithms:

SciPy contains almost too many to demonstrate: e.g.

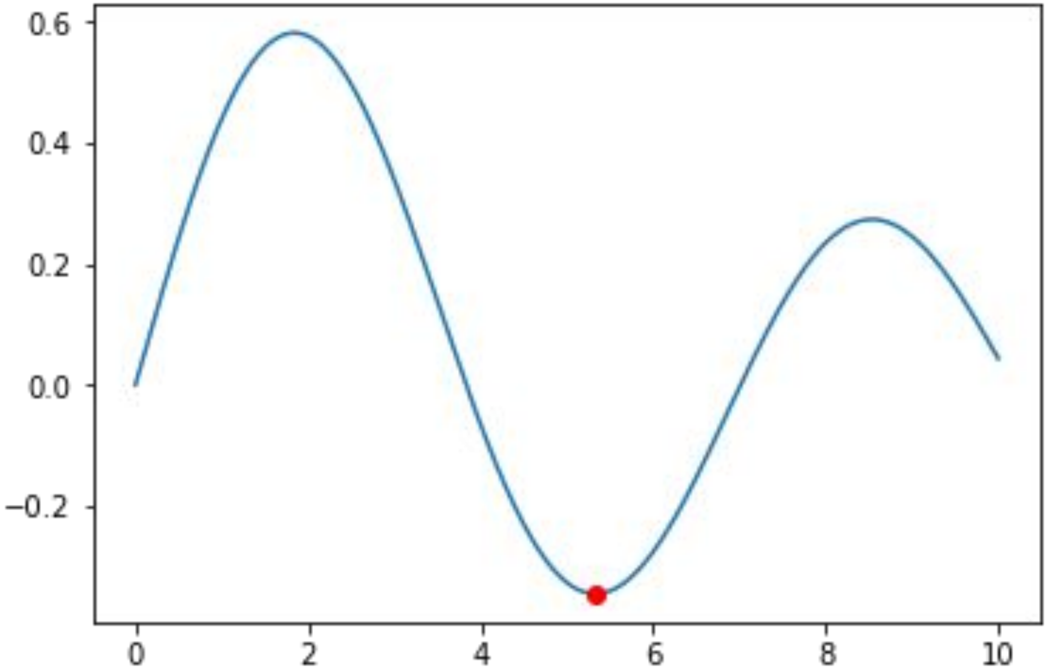
`scipy.sparse`	sparse matrix operations
`scipy.interpolate`	interpolation routines
`scipy.integrate`	numerical integration
`scipy.spatial`	spatial metrics & distances
`scipy.stats`	statistical functions
`scipy.optimize`	minimization & optimization
`scipy.linalg`	linear algebra
`scipy.special`	special mathematical functions
`scipy.fftpack`	Fourier & related transforms

Most functionality comes from wrapping Netlib & related Fortran libraries, meaning it is *blazing* fast.

http://www.scipy.org/

Numerical Algorithms:

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy import special, optimize

x = np.linspace(0, 10, 1000)
opt = optimize.minimize(special.j1, x0=3)
plt.plot(x, special.j1(x))
plt.plot(opt.x, special.j1(opt.x), marker='o', color='red')
```



http://www.scipy.org/

Machine Learning:

```
$ conda install scikit-learn
```

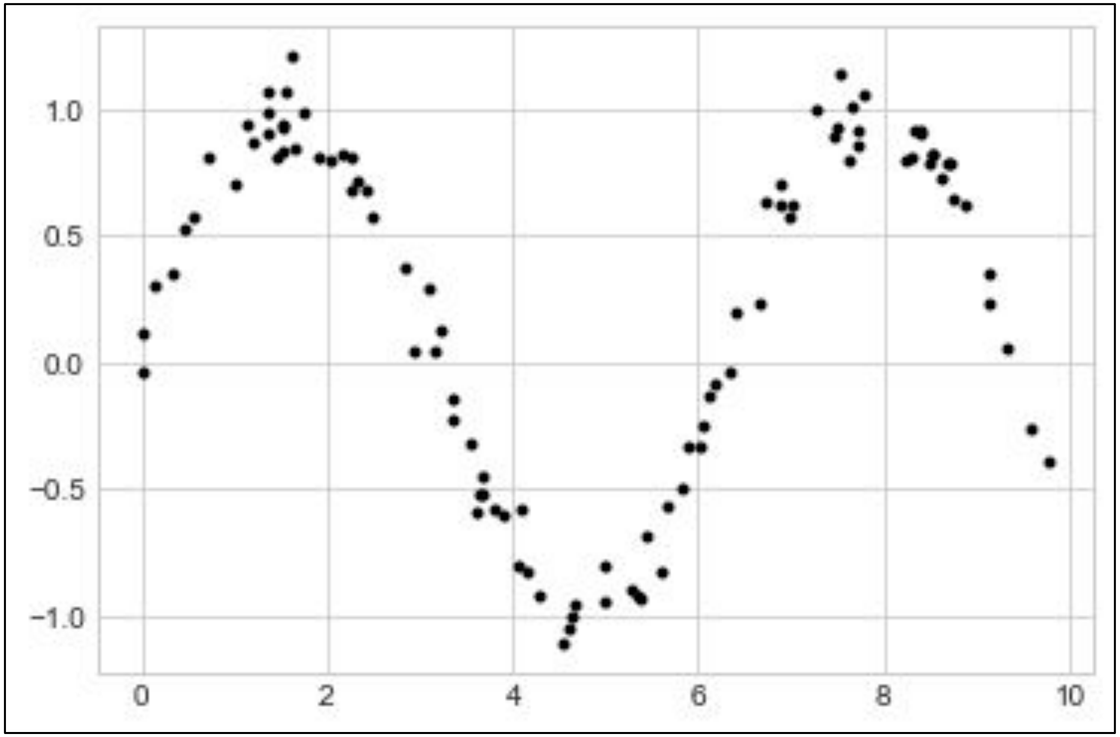
Scikit-learn features a well-defined, extensible API for the most popular machine learning algorithms:

http://scikit-learn.org/

Machine Learning with scikit-learn

Make some noisy 1D data for which we can fit a model:

```
x = 10 * np.random.rand(100)
y = np.sin(x) + 0.1 * np.random.randn(100)
plt.plot(x, y, '.k')
```

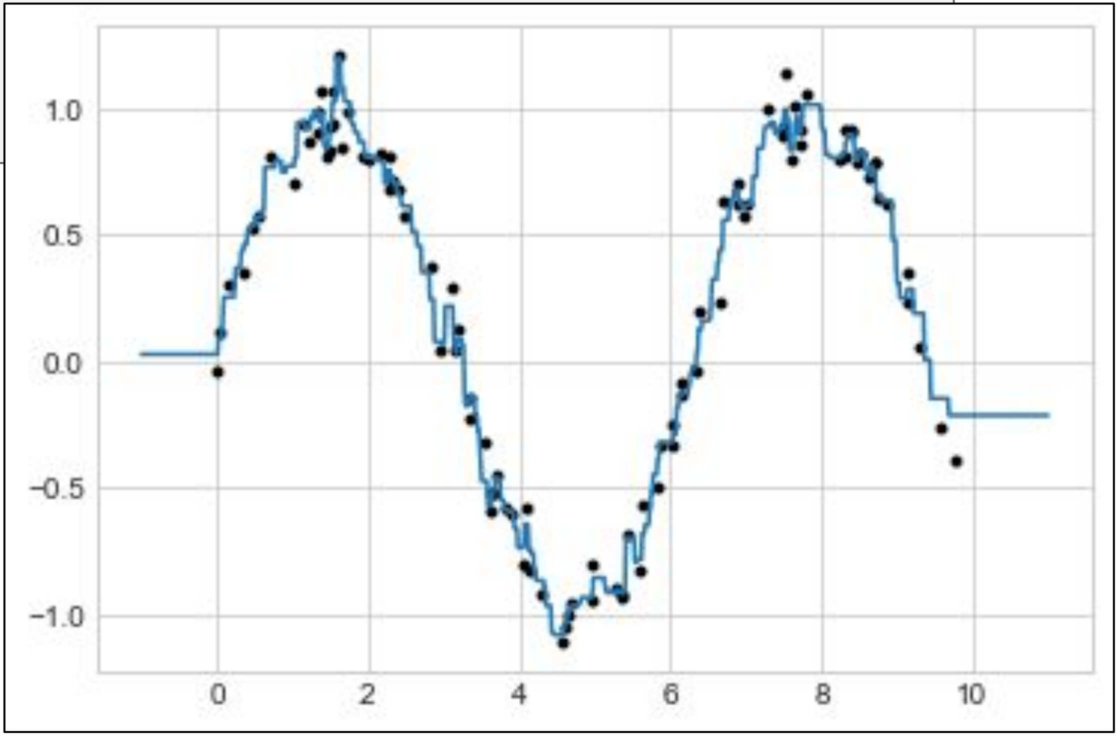


http://scikit-learn.org/

Machine Learning with scikit-learn

Fit a random forest regression:

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()

model.fit(x[:, np.newaxis], y)
xfit = np.linspace(-1, 11, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.plot(x, y, '.k')
plt.plot(xfit, yfit)
```

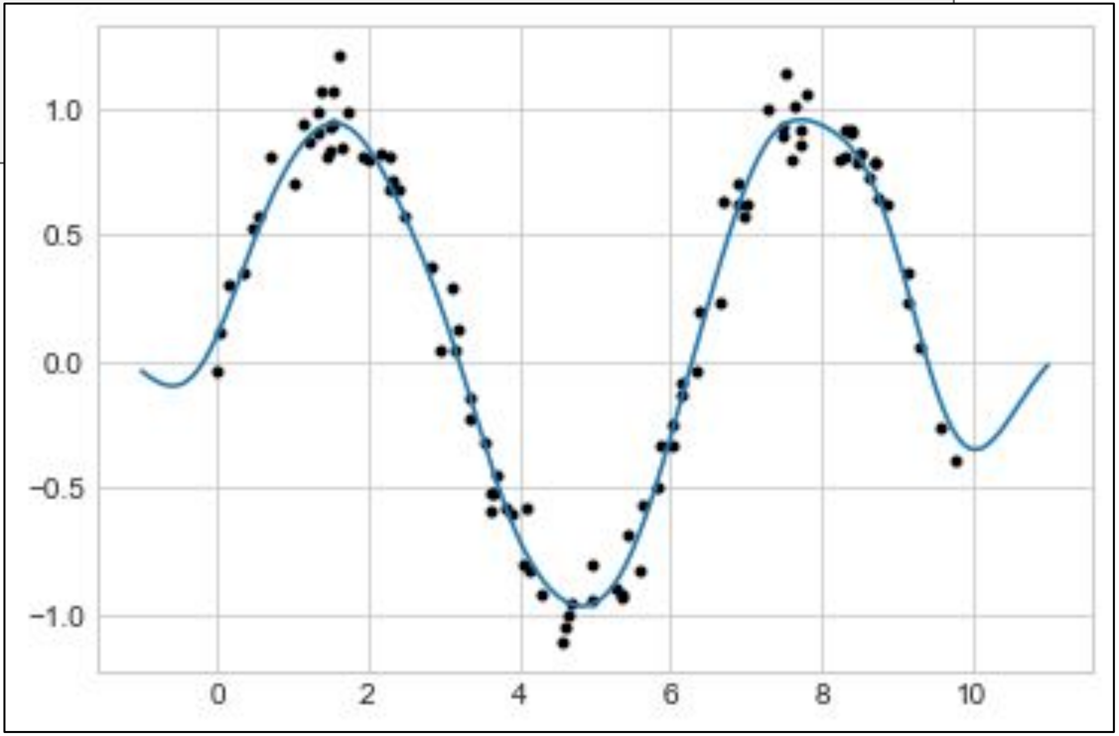


http://scikit-learn.org/

Machine Learning with scikit-learn

Fit a support vector regression:

```
from sklearn.svm import SVR
model = SVR()

model.fit(x[:, np.newaxis], y)
xfit = np.linspace(-1, 11, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.plot(x, y, '.k')
plt.plot(xfit, yfit)
```

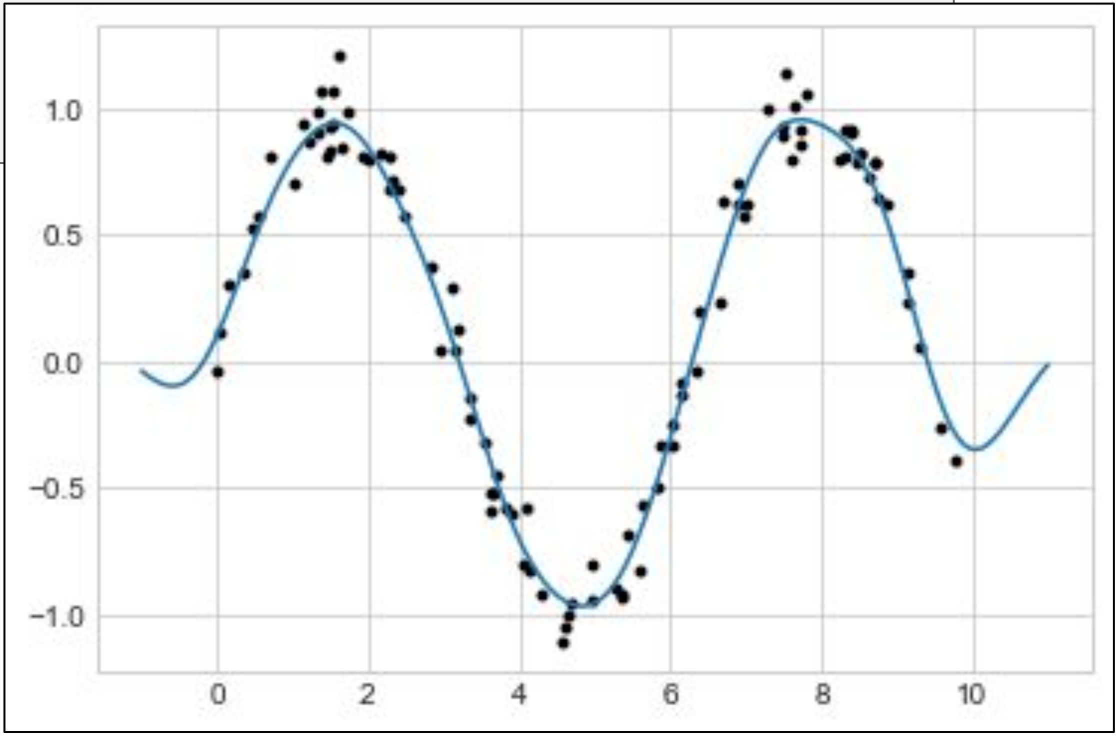


http://scikit-learn.org/

Machine Learning with scikit-learn

Fit a support vector regression:

```python
from sklearn.svm import SVR
model = SVR()

model.fit(x[:, np.newaxis], y)
xfit = np.linspace(-1, 11, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.plot(x, y, '.k')
plt.plot(xfit, yfit)
```



Scikit-learn's strength: provides a uniform API for the most common machine learning methods.

http://scikit-learn.org/

Parallel Computation:

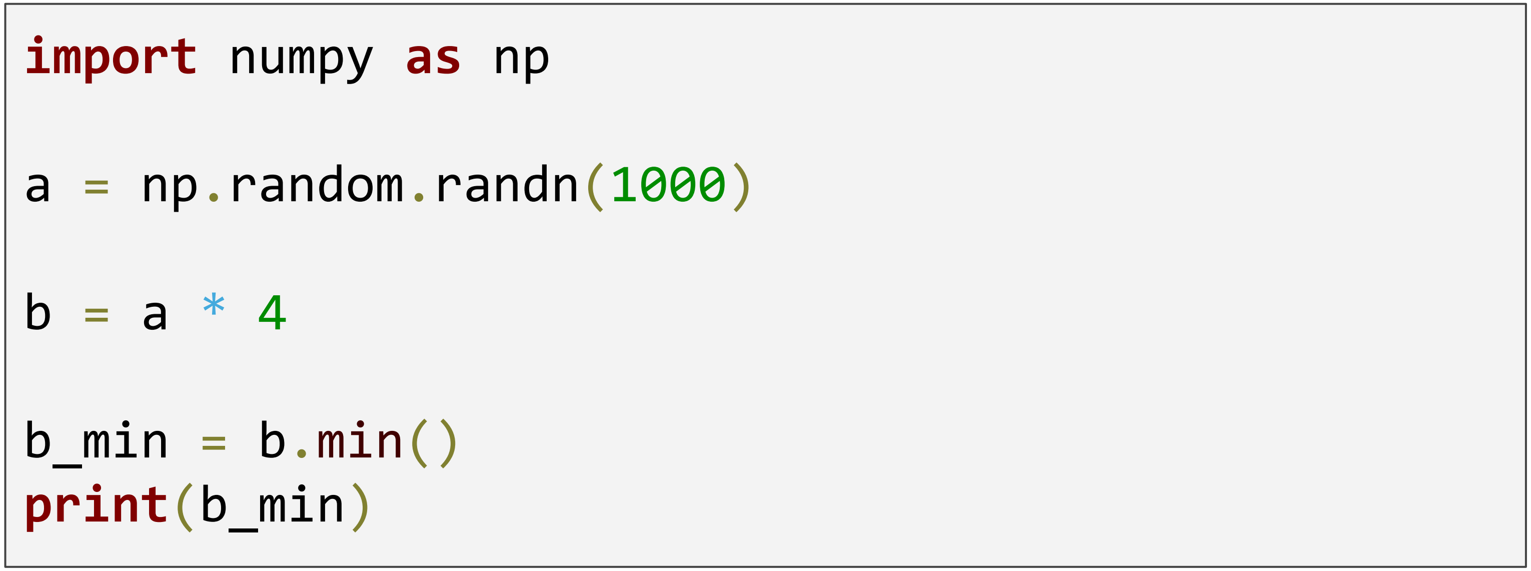
```
$ conda install dask
```

Dask is a lightweight tool for creating task graphs that can be executed on a variety of backends.

http://dask.pydata.org/

Parallel Computation:

Typical data manipulation with NumPy:



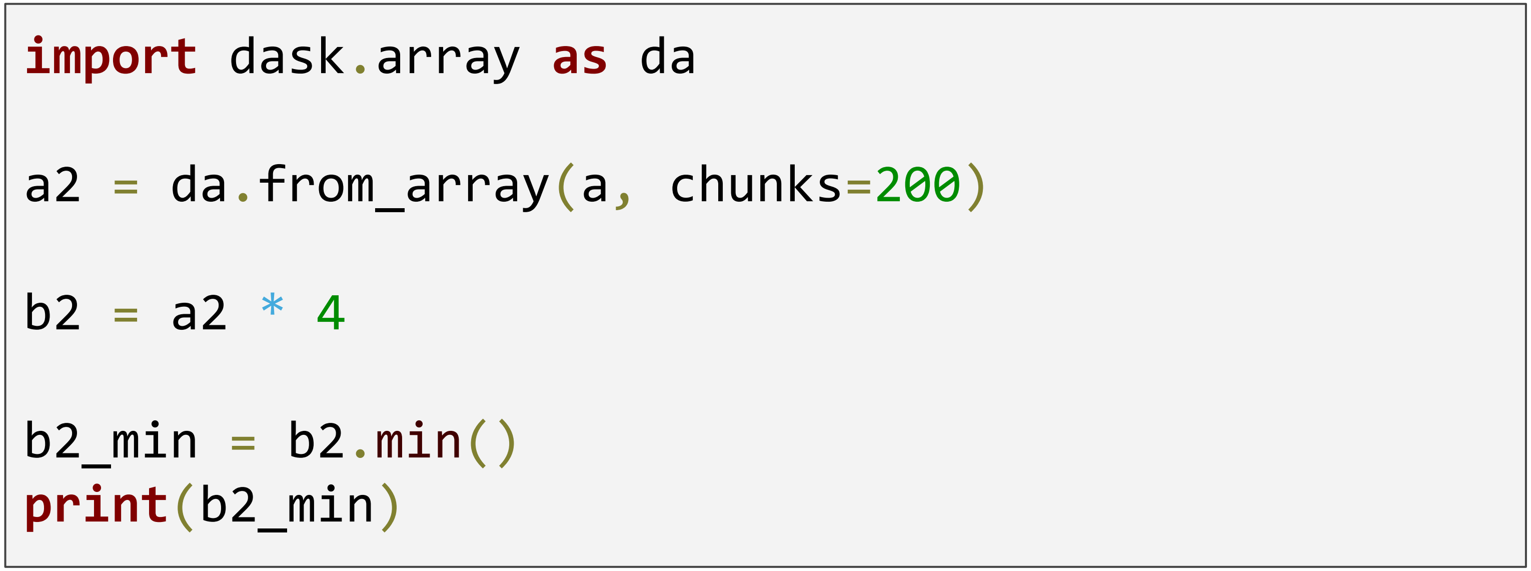
```python
import numpy as np

a = np.random.randn(1000)

b = a * 4

b_min = b.min()
print(b_min)
```

```
-13.2982888603
```

http://dask.pydata.org/

Parallel Computation:

Same operation with dask



```
import dask.array as da

a2 = da.from_array(a, chunks=200)

b2 = a2 * 4

b2_min = b2.min()
print(b2_min)
```

```
dask.array<amin-aggregate, shape=(),
           dtype=float64, chunksize=()>
```

http://dask.pydata.org/

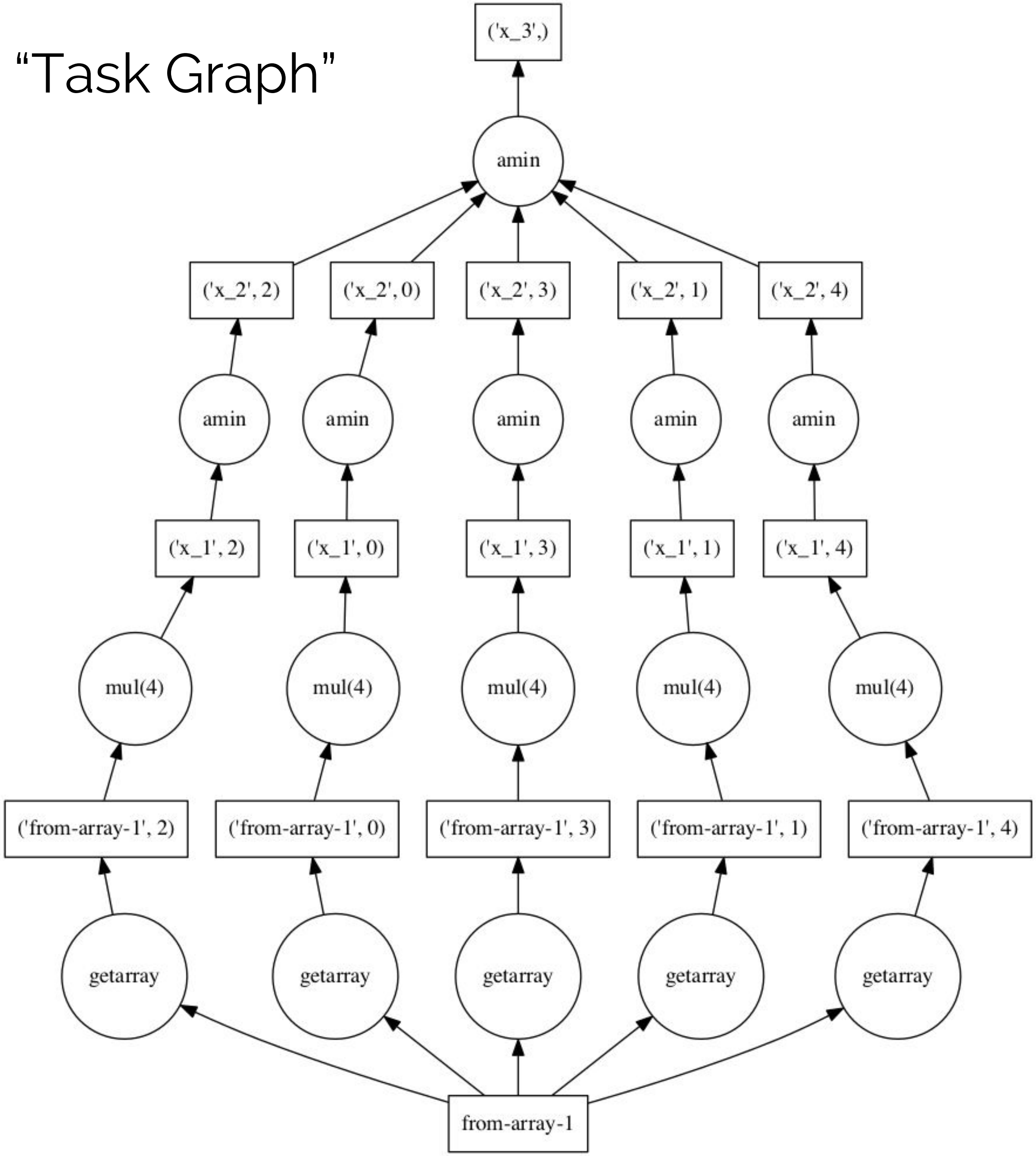
Paralle

Same ope

```
impor

a2 =

b2 =

b2_mi
print
```

```
dask.
```

“Task Graph”



http://dask.pydata.org/

Parallel Computation:

Same operation with dask

```
import dask.array as da

a2 = da.from_array(a, chunks=200)

b2 = a2 * 4

b2_min = b2.min()
print(b2_min)
```

```
dask.array<amin-aggregate, shape=(),
           dtype=float64, chunksize=()>
```

```
b2_min.compute()
```

```
-13.298288860312757
```

http://dask.pydata.org/

Code Optimization

Numba

```
$ conda install numba
```

Numba is a bytecode compiler that can convert Python code to fast LLVM code targeting a CPU or GPU.

http://numba.pydata.org/

Code Optimization

Numba

Simple iterative functions tend to be slow in Python:

```
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

%timeit fib(10000)  # ipython “timeit magic”
```

```
100 loops, best of 3: 2.73 ms per loop
```

http://numba.pydata.org/

Code Optimization

Numba

With a quick decorator, code can be ~1000x as fast!

```
import numba

@numba.jit
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

%timeit fib(10000)  # ipython “timeit magic”
```

```
100000 loops, best of 3: 6.06 µs per loop
```

~ 500x speedup!

http://numba.pydata.org/

Code Optimization

Numba

With a quick decorator, code can be ~1000x as fast!

```python
import numba

@numba.jit
def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a

%timeit fib(10000)  # ipython “timeit magic”
```

```
100000 loops, best of 3: 6.06 µs per loop
```

~ 500x speedup!

Numba achieves this by just-in-time (JIT) compilation of the Python function to LLVM byte-code.

http://numba.pydata.org/

Code Optimization

```
$ conda install cython
```

Cython is a superset of the Python language that can be compiled to fast C code.

http://www.cython.org/

Code Optimization

Again, returning to our fib function:

```
# python code

def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
```

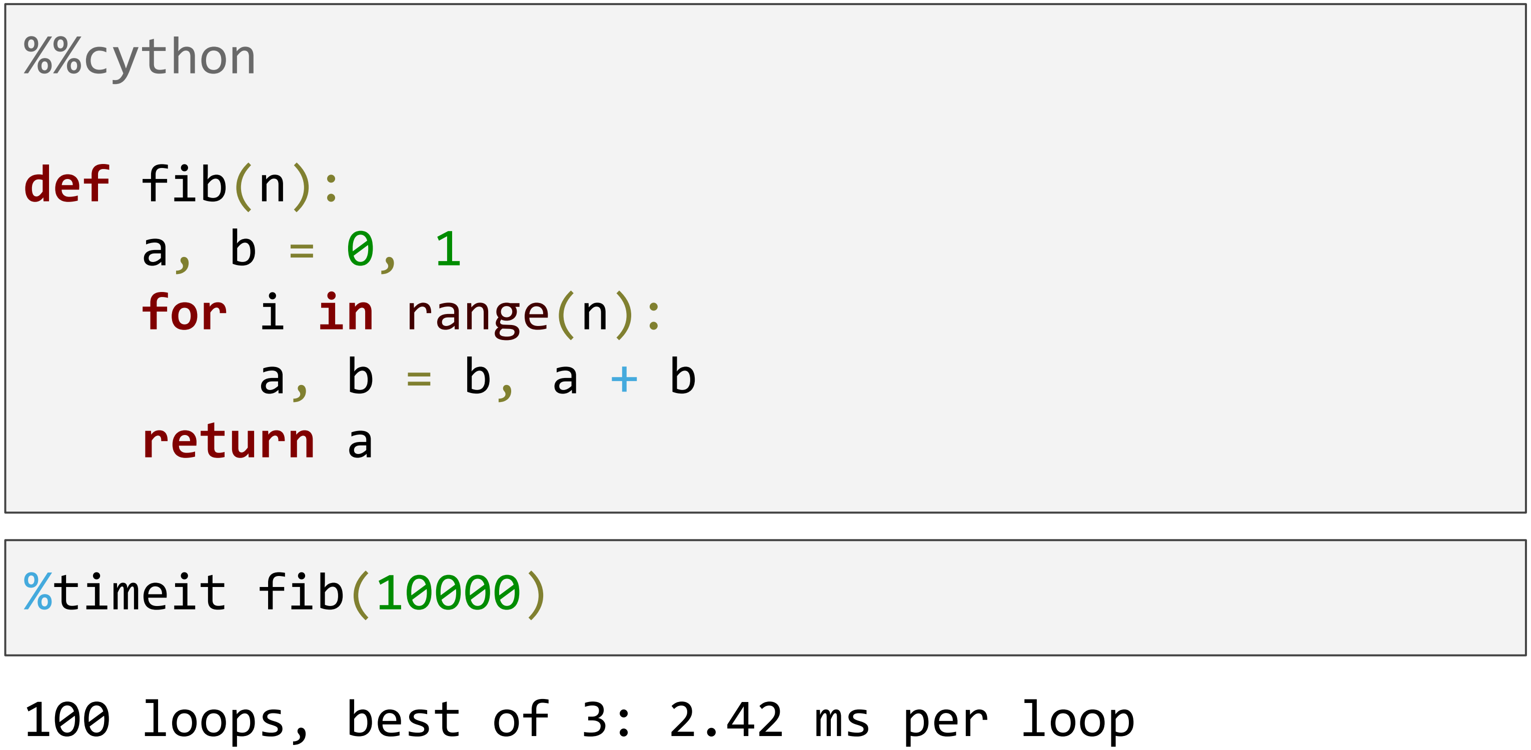
```
%timeit fib(10000)
```

```
100 loops, best of 3: 2.73 ms per loop
```

http://www.cython.org/

Code Optimization

Cython compiles the code to C, giving marginal speedups without even changing the code:



```
%%cython

def fib(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a + b
    return a
```

```
%timeit fib(10000)
```

```
100 loops, best of 3: 2.42 ms per loop
```

~ 10% speedup!

http://www.cython.org/

Code Optimization

Using cython's syntactic sugar to specify types for the compiler leads to much better performance:

```
%%cython

def fib(int n):
    cdef int a = 0, b = 1
    for i in range(n):
        a, b = b, a + b
    return a
```

```
%timeit fib(10000)
```

```
100000 loops, best of 3: 5.93 µs per loop
```

~ 500x speedup!

http://www.cython.org/

Powered by Cython:

The PyData stack is largely powered by Cython:

. . . and many more.

http://www.cython.org/

Python is not a data science language.

~

And this may be its greatest strength.

Thank You!

Email:	jakevdp@uw.edu
Twitter:	@jakevdp
Github:	jakevdp
Web:	http://vanderplas.com/
Blog:	http://jakevdp.github.io/