PyData 101

A Quick Tour of the PyData World . . .

Installation

推荐使用Anaconda发行版（中国大陆用户请使用[清华大学TUNA镜像，https://mirror.tuna.tsinghua.edu.cn/help/anaconda/），Windows、Linux 和Mac OS X操作系统的安装和使用方式类似。

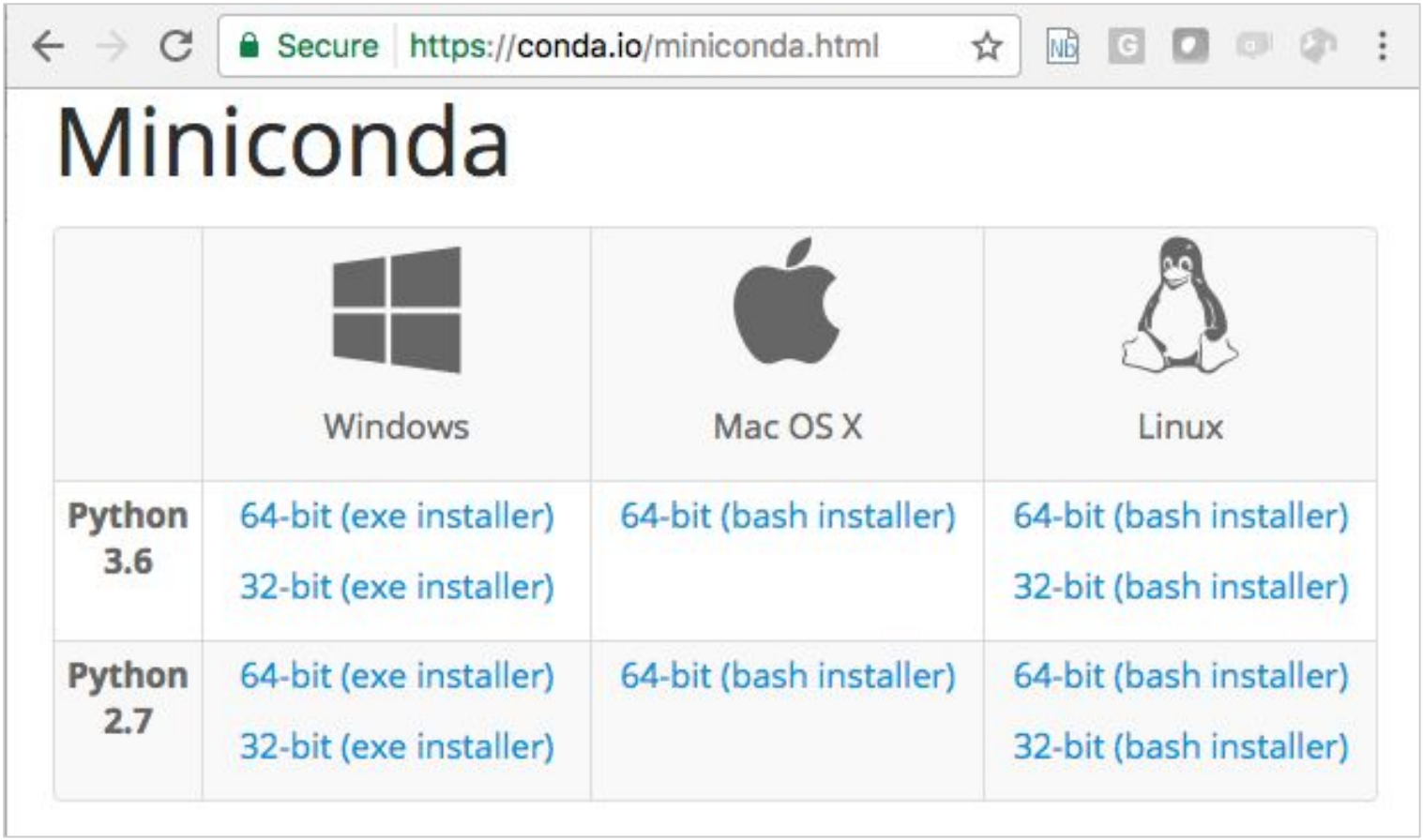
Anaconda发行版有两种。

- Miniconda，只包含Python解释器和一个_conda_命令行。它是一个跨平台的程序包管理器，可以管理各种Python程序包，类似于Linux用户熟悉的apt和yum程序包管理器。
- Anaconda，除了包含Python和conda之外，还同时绑定了许多（四五百个）科学计算程序包。由于预安装了许多包，因此安装它需要占用几个G的存储空间。

推荐 Anaconda 全家桶

http://conda.pydata.org/

Installation



Anaconda 与 Miniconda 支持主流操作系统

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
```

conda install 安装包，有时需要配置源

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
```

conda create 配置虚拟环境

http://conda.pydata.org/

Installation

```
$ source 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.
```

activate 激活虚拟环境

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
```

查看环境列表

http://conda.pydata.org/

Coding Environment:


```
$ conda install jupyter notebook
```

IPython(交互式科学计算)的功能扩展，支持多种编程语言（kernel），多种操作界面。

有一堆“魔法命令”（magic command）：

用符号?获取文档
通过符号??获取源代码

性能分析：

%time 对单个语句的执行时间进行计时。
%timeit 对单个语句的重复执行进行计时以获得更高的准确度。
%prun 利用分析器运行代码。
%lprun 利用逐行分析器执行代码。
%memit 测量单个语句的内存使用。
%mprun 通过逐行的内存分析器运行代码

http://jupyter.org/

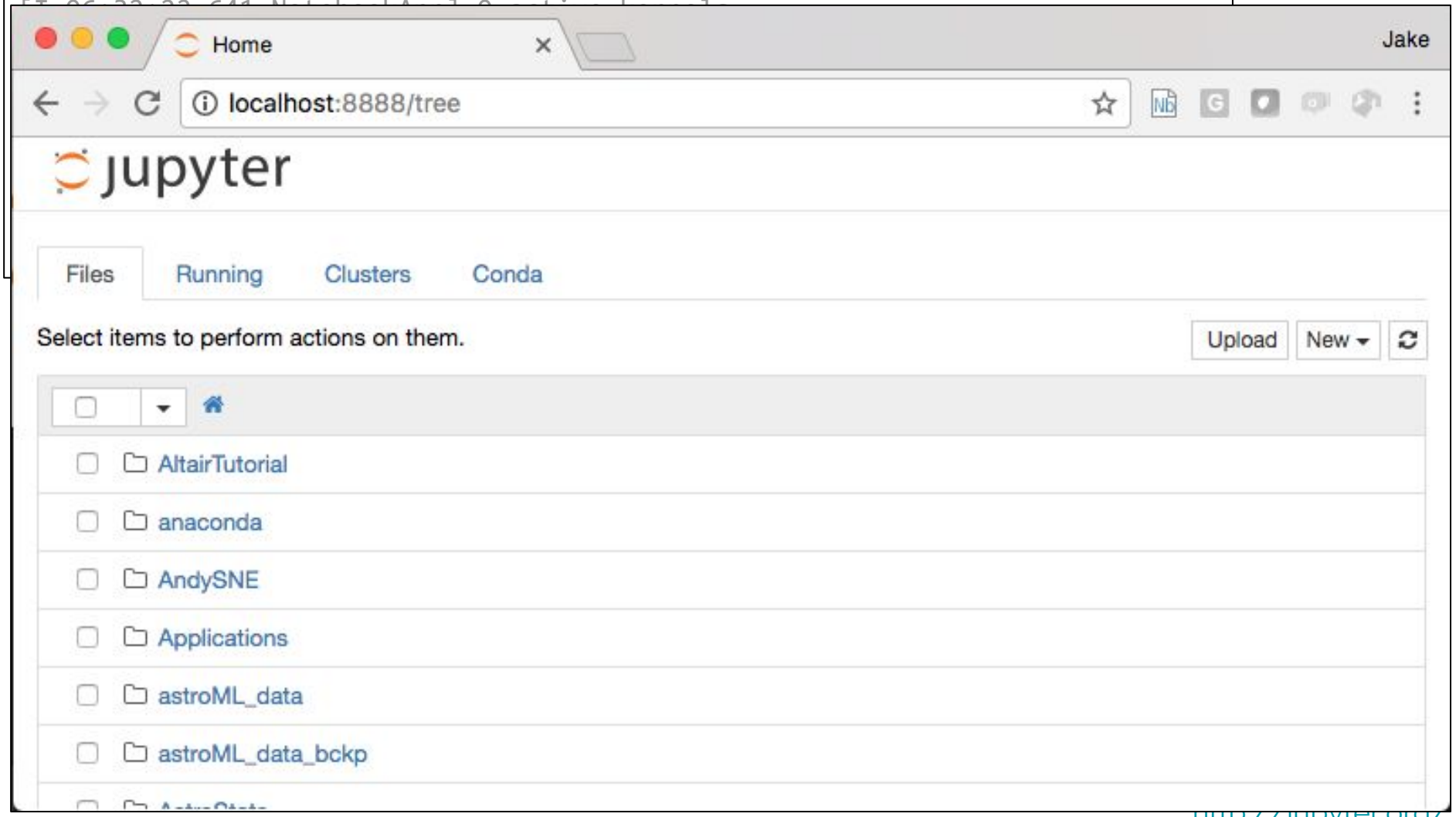
Coding Environment:


```
$ 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:

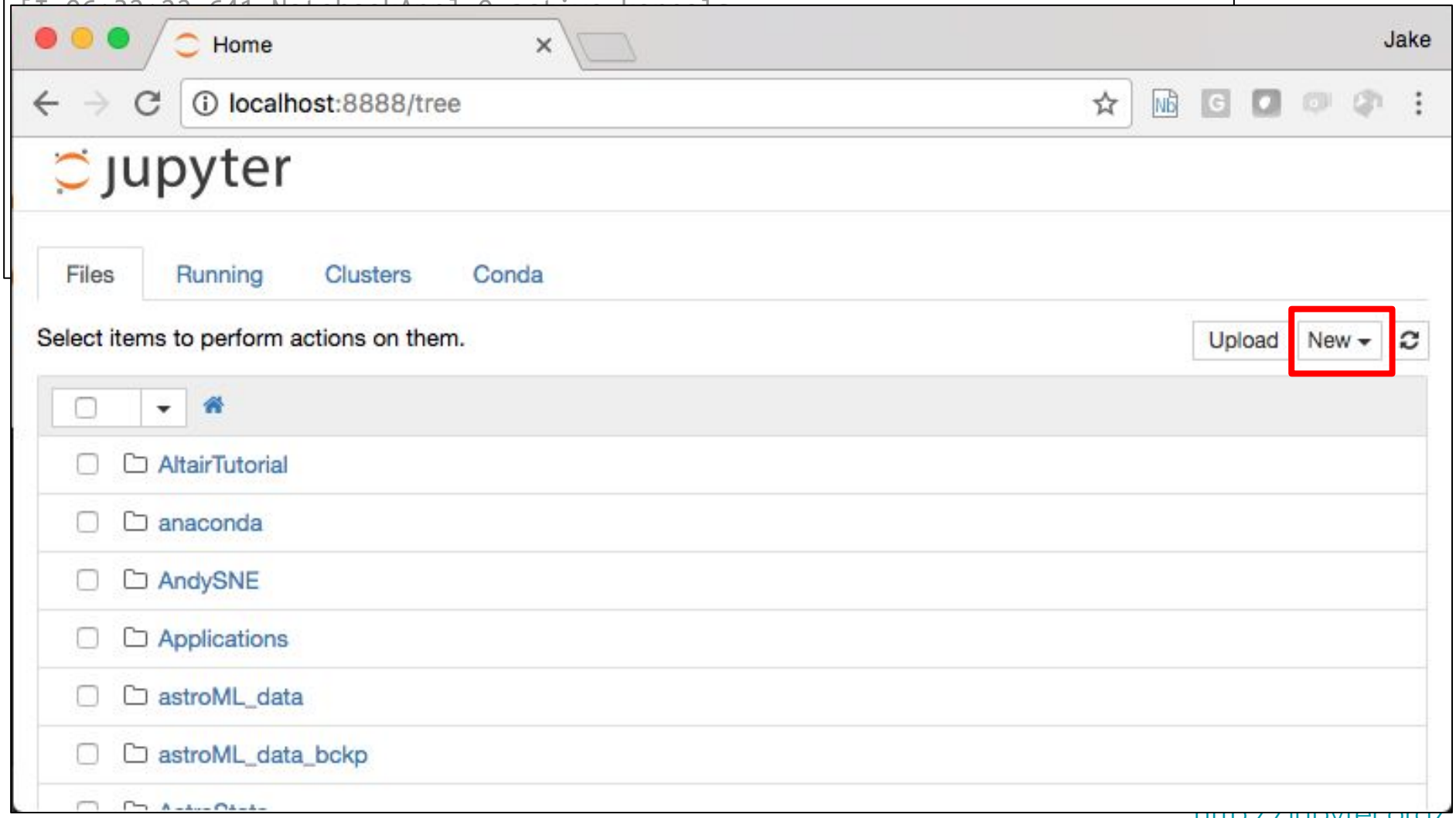

```
$ jupyter notebook
[I 06:32:22.641 NotebookApp] Serving notebooks from local directory:
/Users/jakevdp
```



http://jupyter.org/

Coding Environment:

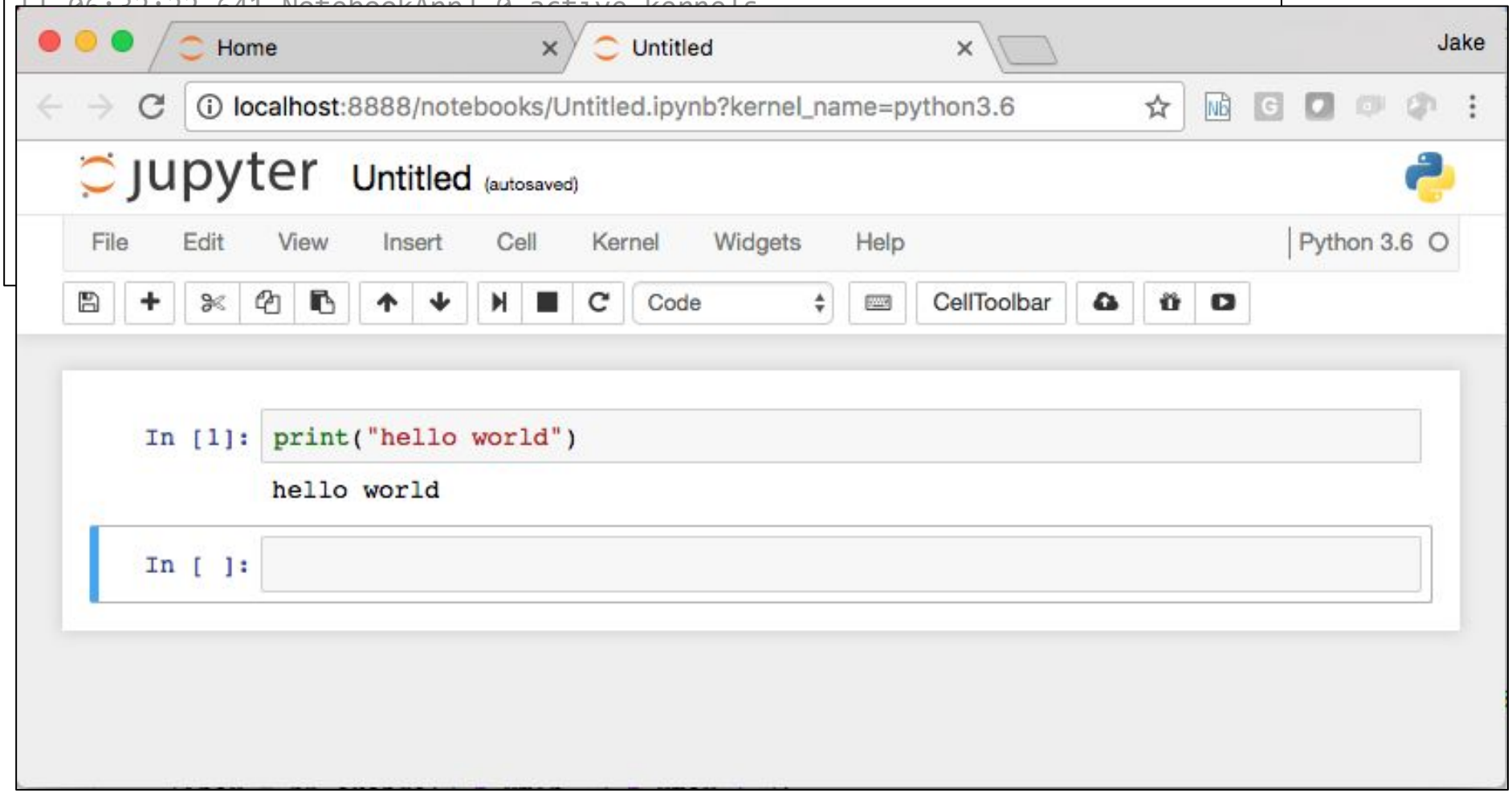
```
$ jupyter notebook
[I 06:32:22.641 NotebookApp] Serving notebooks from local directory:
/Users/jakevdp
```



http://jupyter.org/

Coding Environment:


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



http://jupyter.org/

Coding Environment:


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

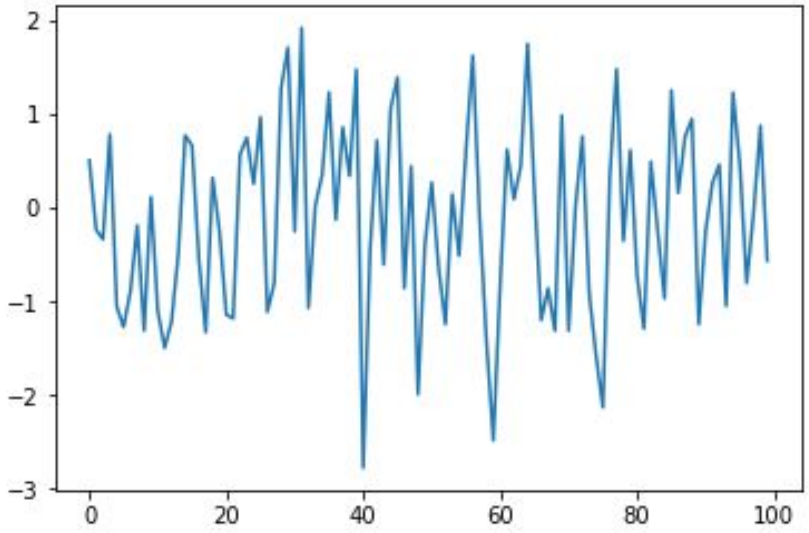
Home | Untitled | Jake

localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3.6

File Edit View Insert Cell Kernel Widgets Help | Python 3.6

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>]



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.

```
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:

```
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
```

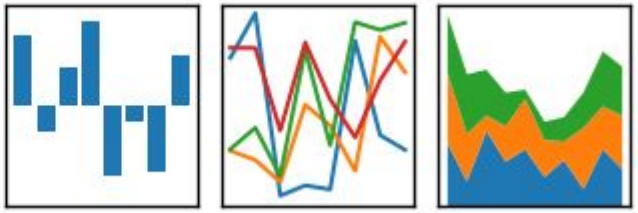
```
10 loops, best of 3: 58.6 ms per loop
```

~ 100x speedup!

Labeled Data:

pandas

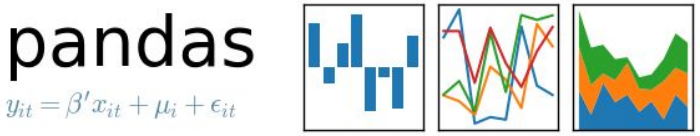
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



```
$ conda install pandas
```

http://pandas.pydata.org/

Labeled Data:

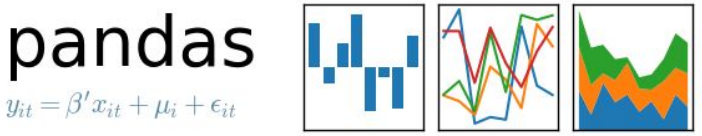


Pandas provides a **`DataFrame`** object which is like a NumPy array, but has labeled rows and columns:

```
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
```

Labeled Data:

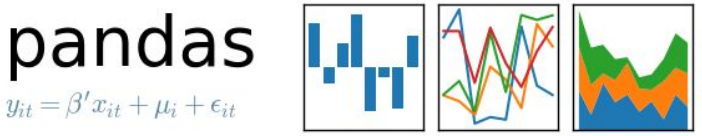


Like NumPy, arithmetic is element-wise, but you can access and augment the data using column name:

```
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
```

Labeled Data:



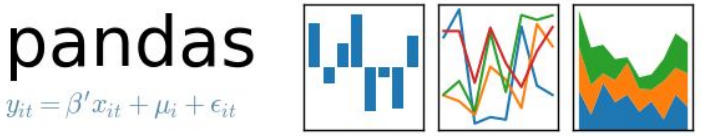
Pandas excels in reading data from disk in a variety of formats. Start here to read virtually any data format!

```
# 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
```

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:


```
$ conda install matplotlib
```

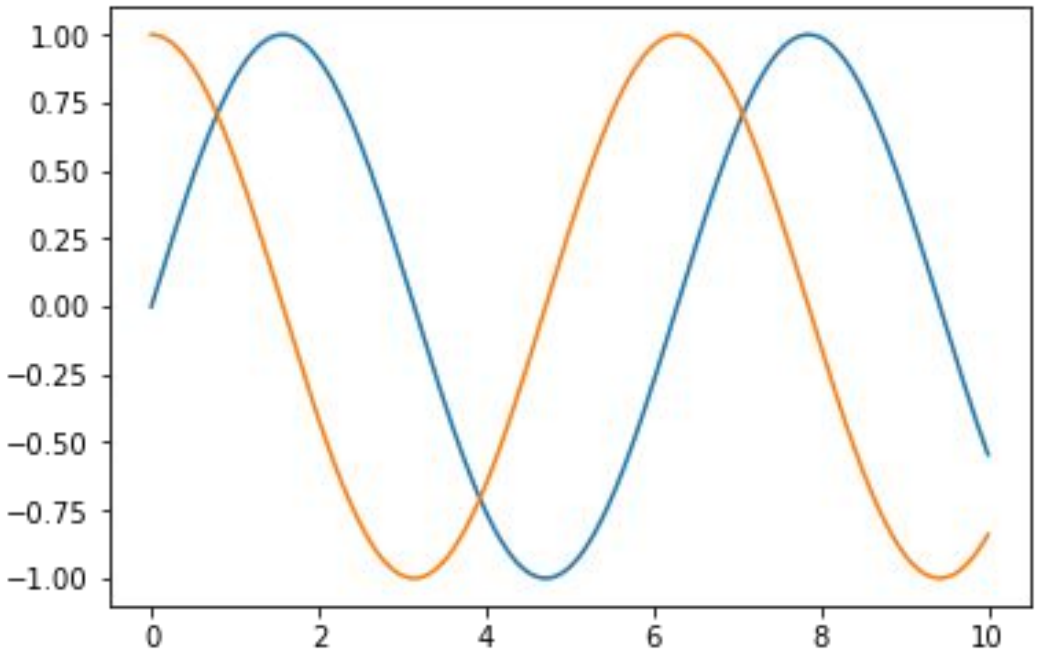
http://www.matplotlib.org/

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))
```

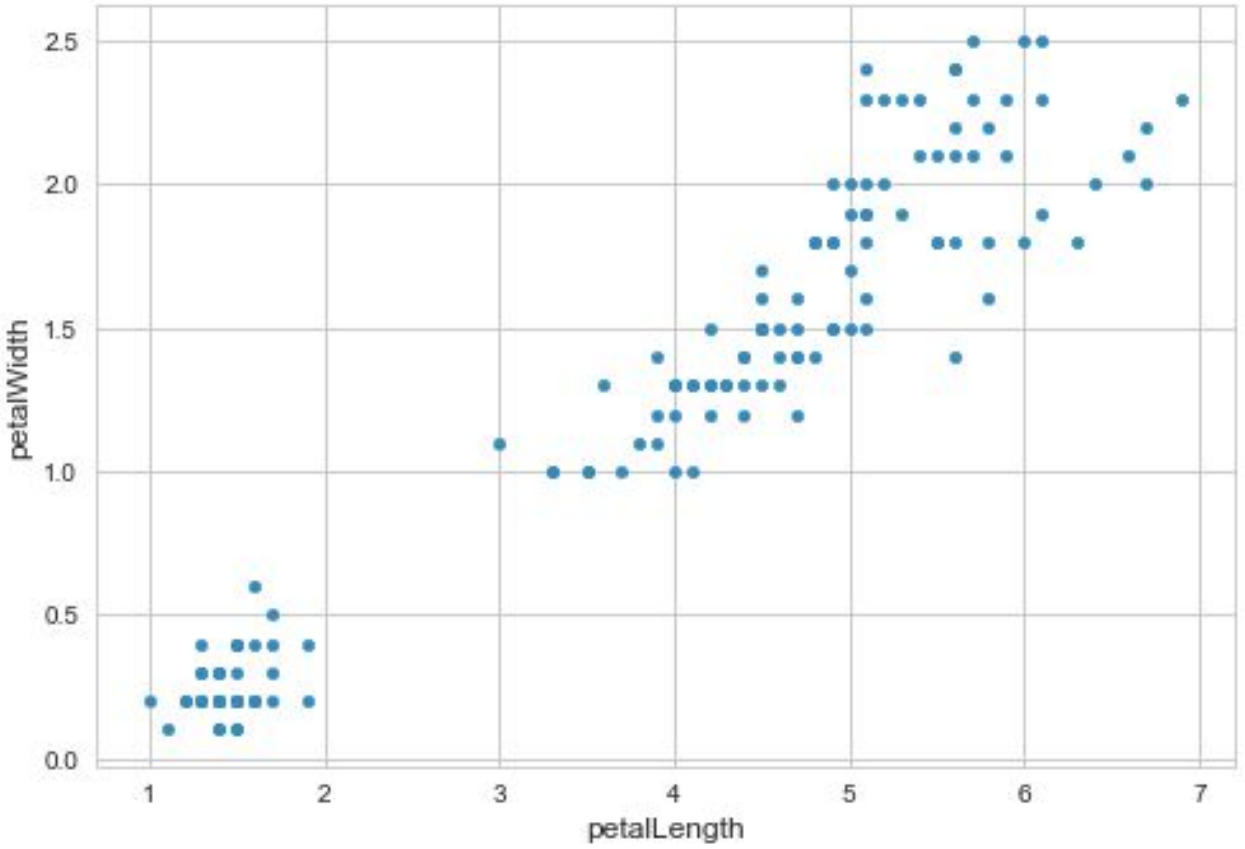


http://www.matplotlib.org/

Visualization Beyond Matplotlib . . .

Pandas offers a simplified Matplotlib Interface:

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

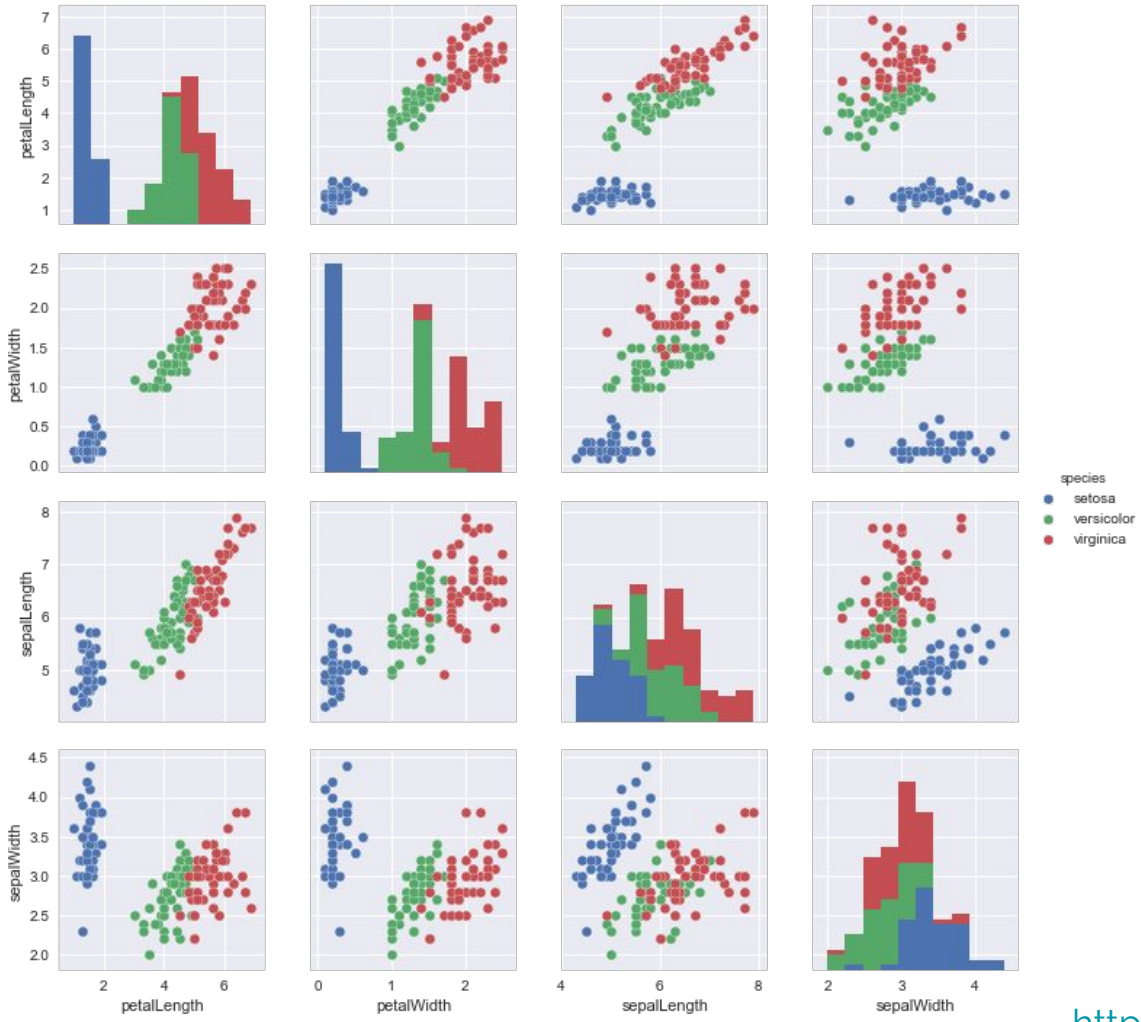


http://pandas.pydata.org/

Visualization Beyond Matplotlib . . .

Seaborn is a package for statistical data visualization

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

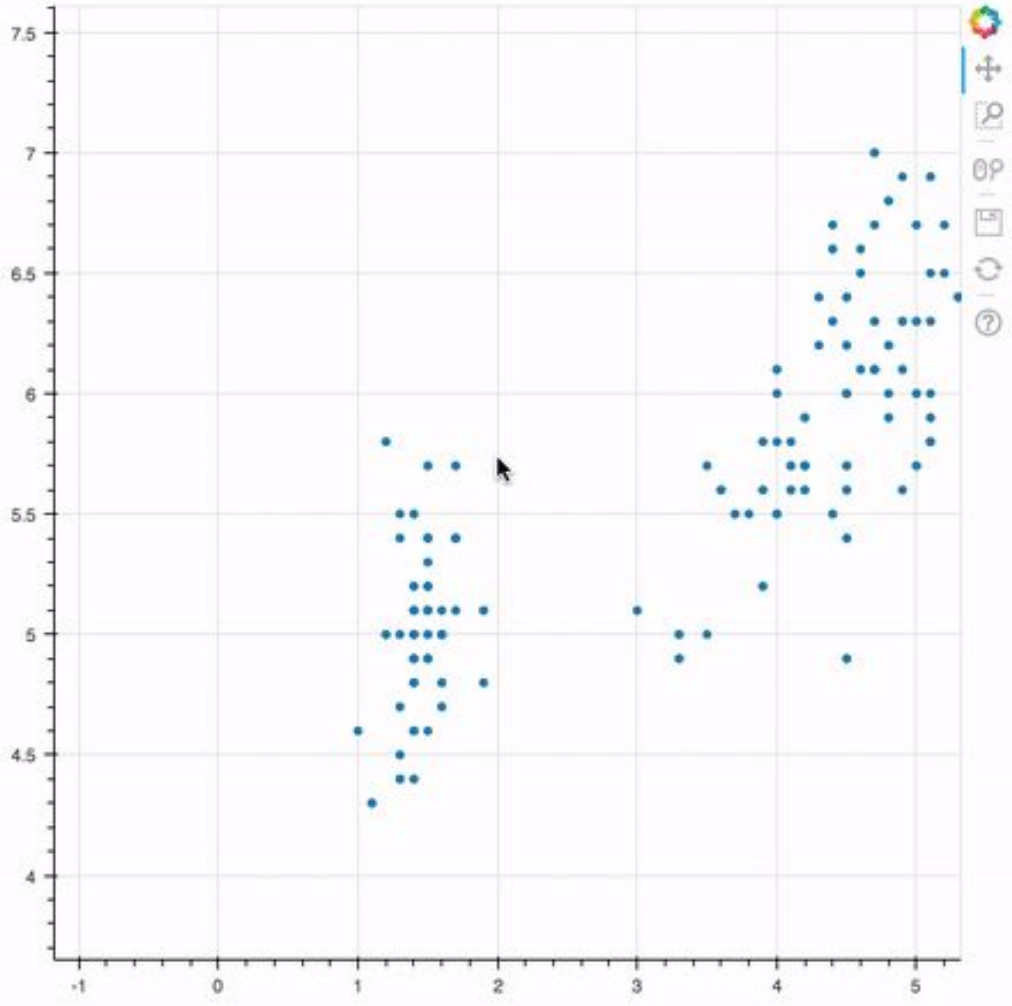


http://seaborn.pydata.org/

Visualization Beyond Matplotlib . . .

Bokeh: interactive visualization in the browser.

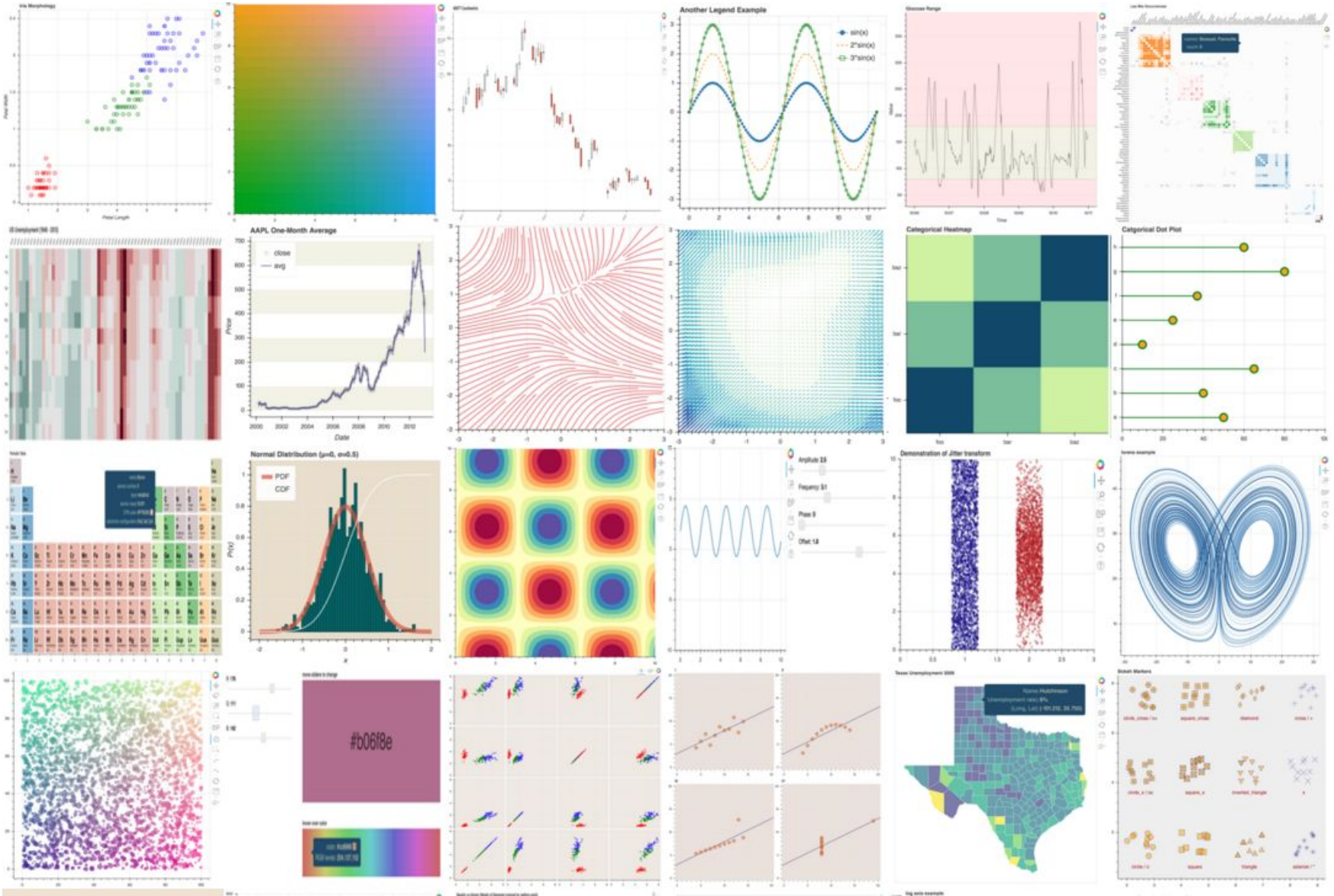
```
In [10]: p = figure()
         p.circle(iris.petalLength, iris.sepalLength)
         show(p)
```



http://bokeh.pydata.org/

Visualization Beyond Matplotlib . . .

Bokeh: interactive visualization in the browser.



rg/

Visualization Beyond Matplotlib . . .

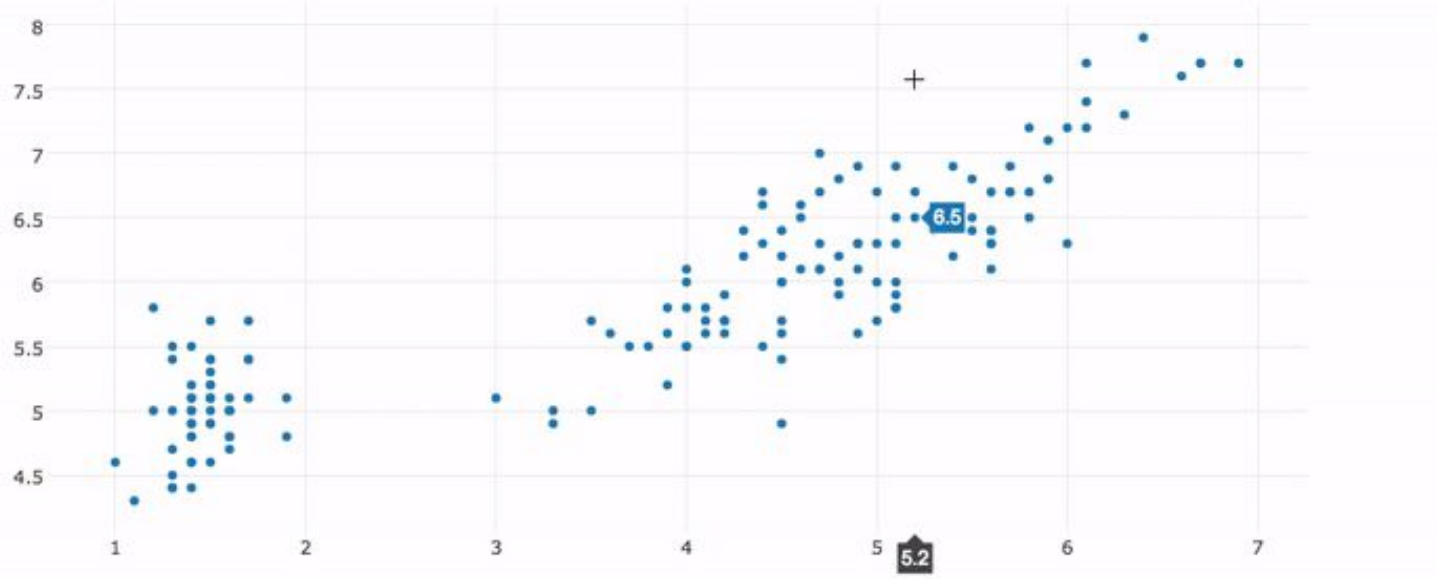
plotly

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])
```

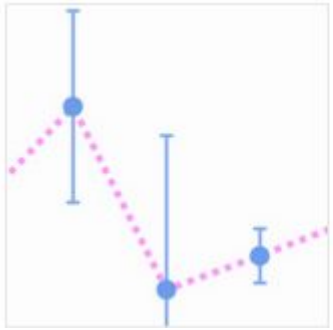
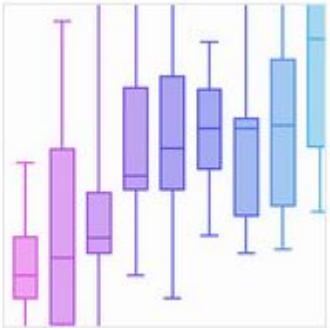
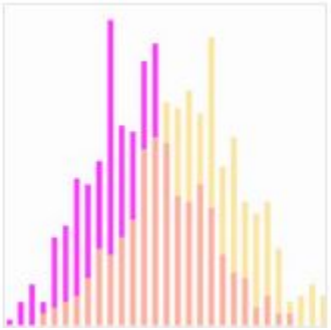
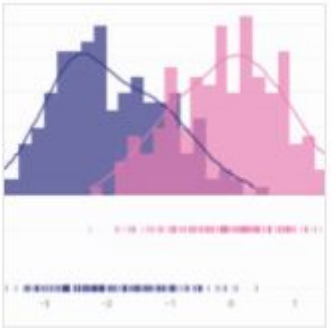
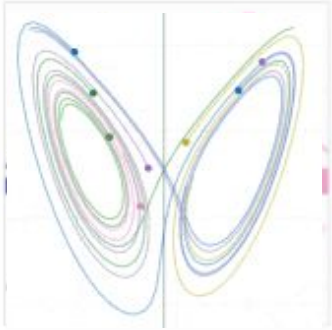
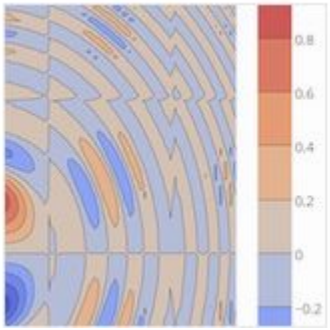


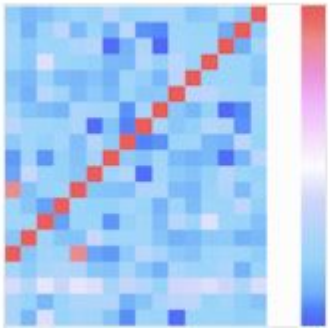
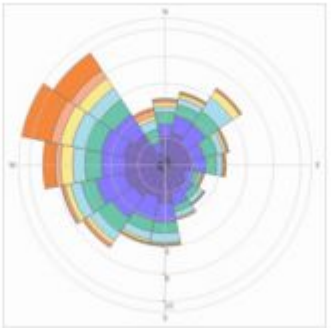
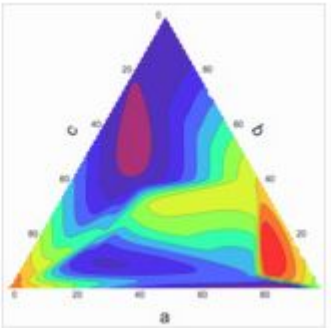
http://plotly.com/

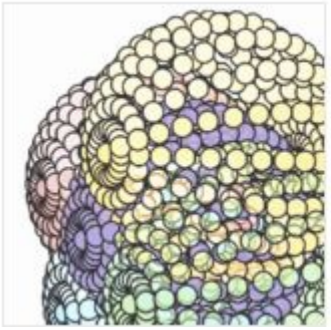
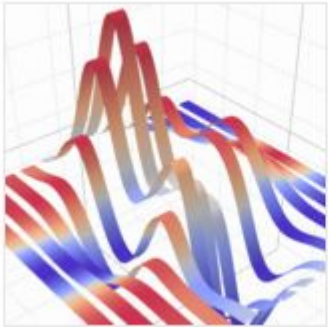
Visualization Beyond Matplotlib . . .

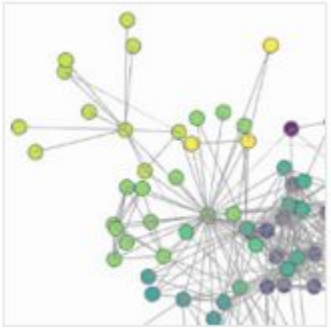
plotly

Plotly: “modern platform for data science”

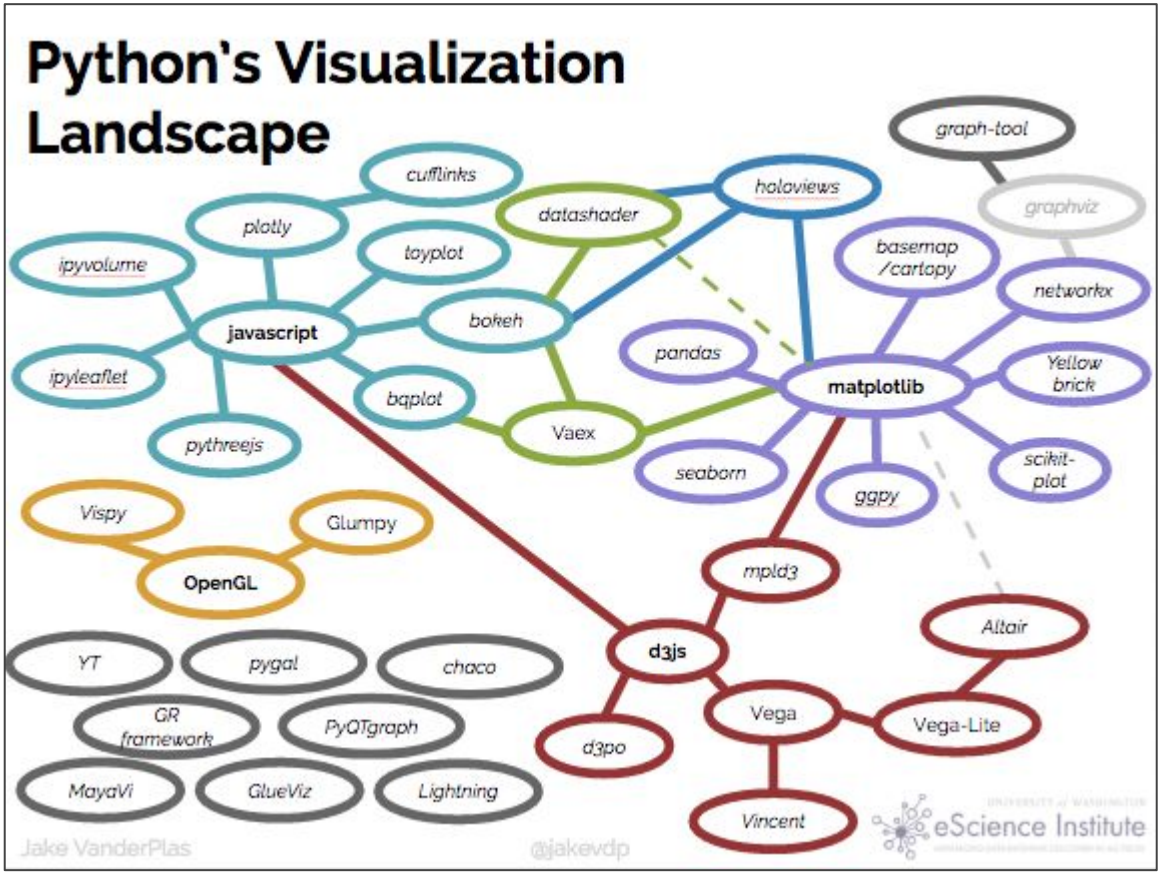







http://plotly.com/

Visualization Beyond Matplotlib . . .



See jakevdp 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.

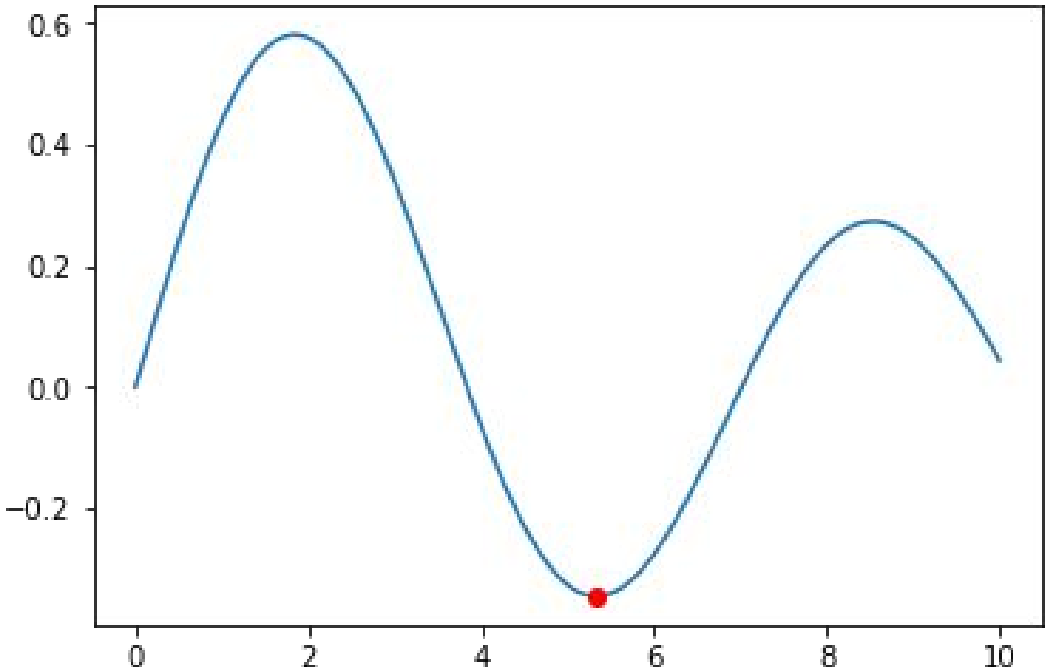
http://www.scipy.org/

Numerical Algorithms:

SciPy

```
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
```

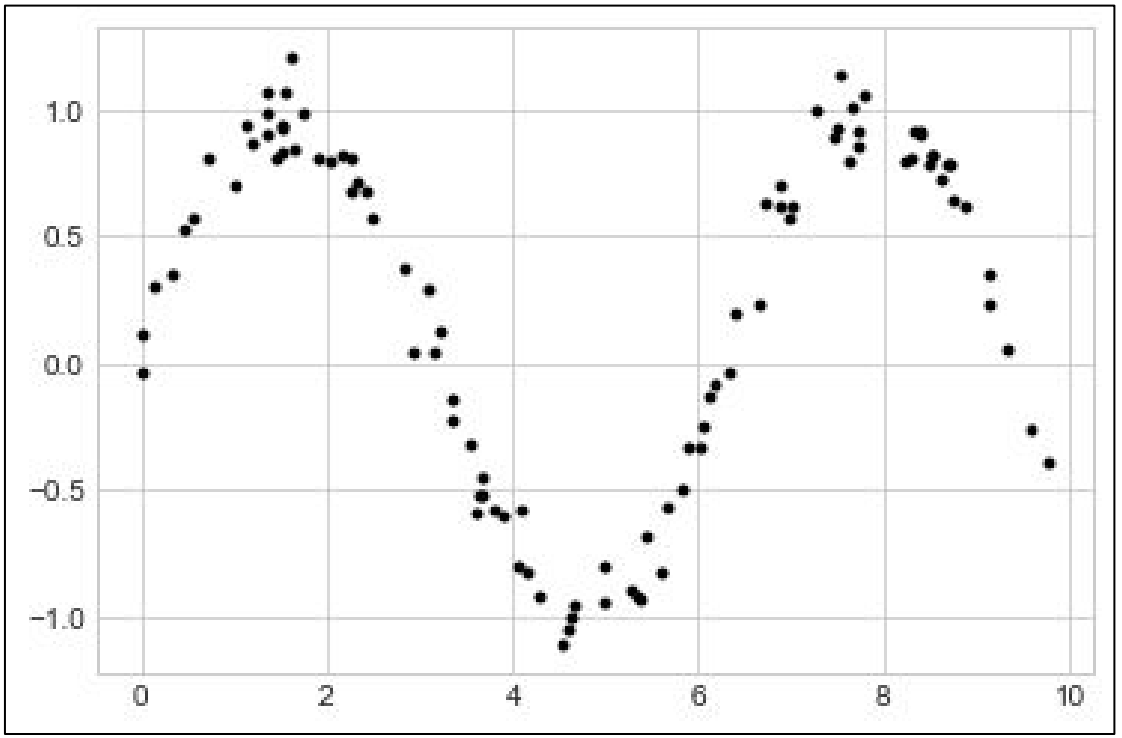
Scikit-learn features a well-defined, extensible API for the most popular machine learning algorithms:

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/

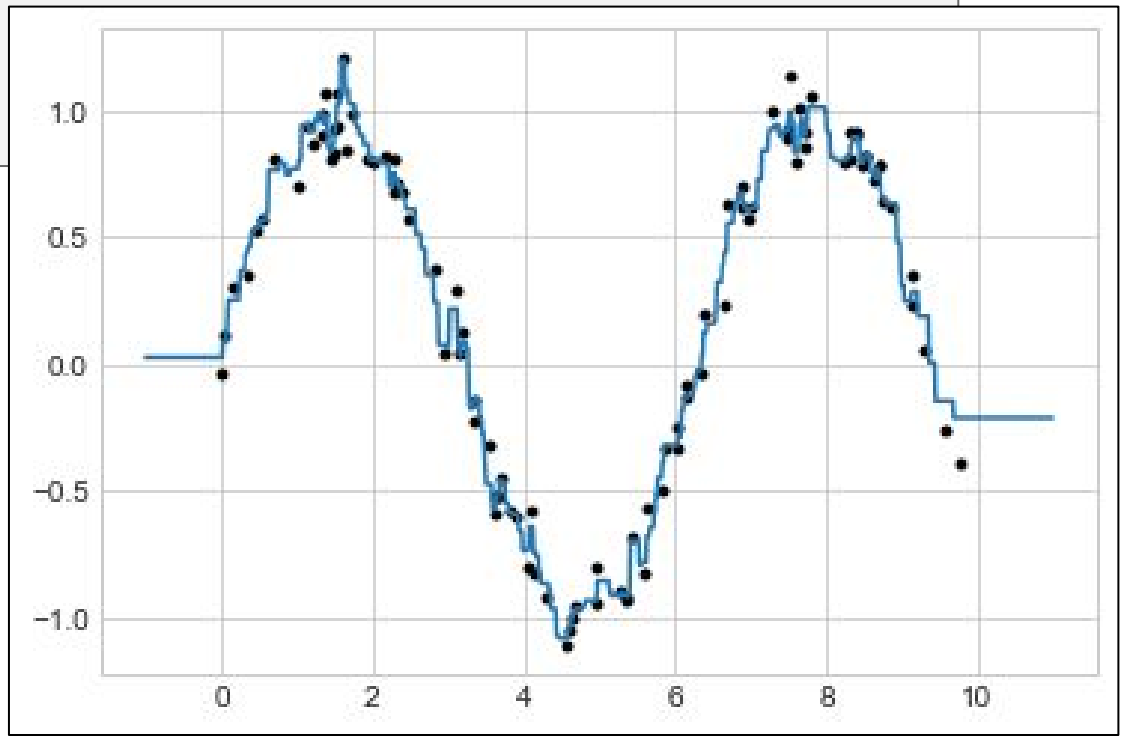
Machine Learning with scikit-learn

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/

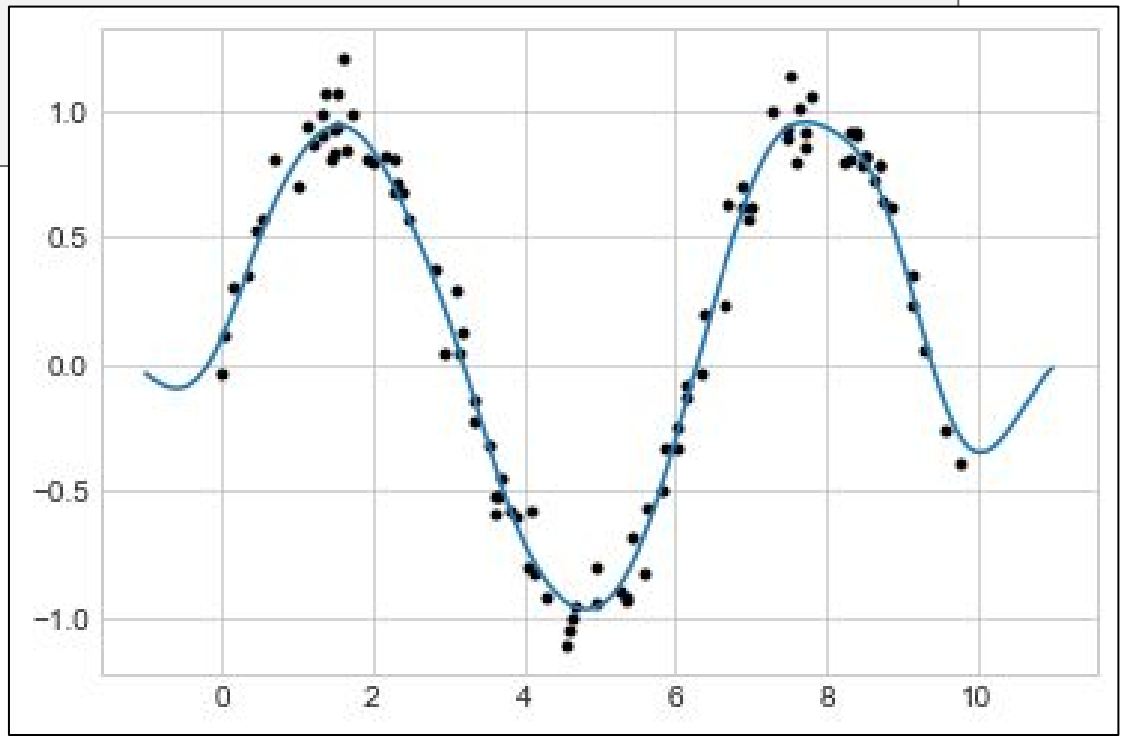
Machine Learning with scikit-learn

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/

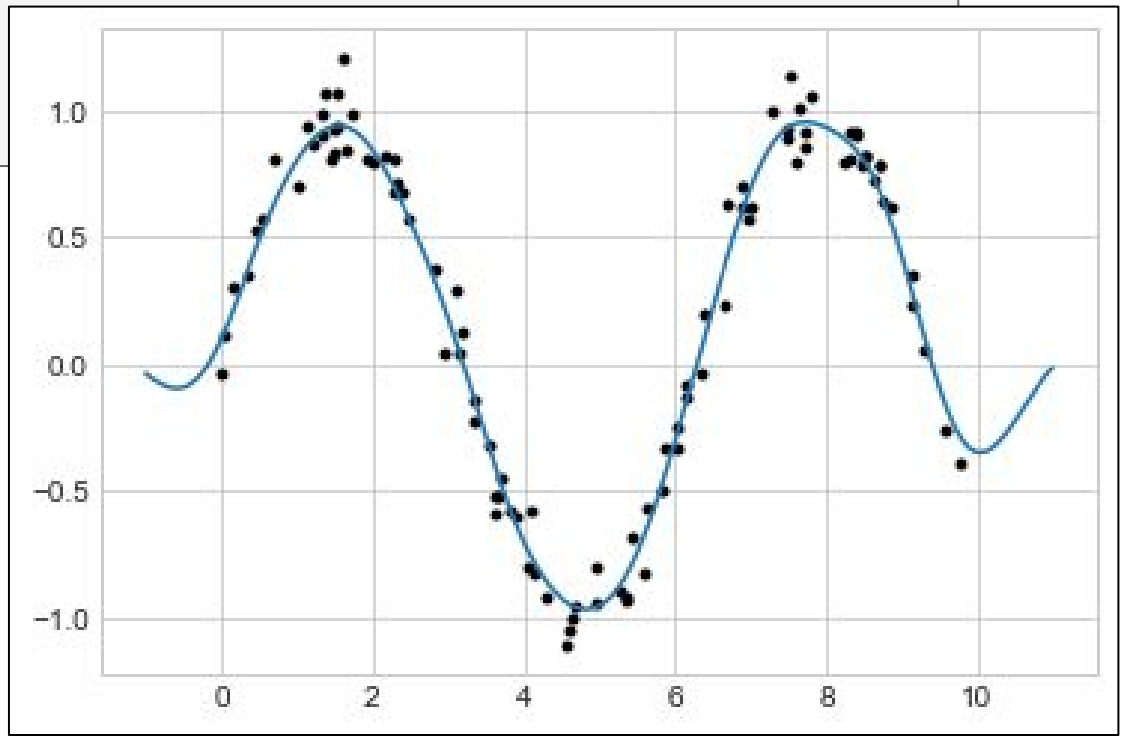
Machine Learning with scikit-learn

Fit a support vector regression:

```
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 common 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:

```
import numpy as np

a = np.random.randn(1000)

b = a * 4

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

```
-13.2982888603
```

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

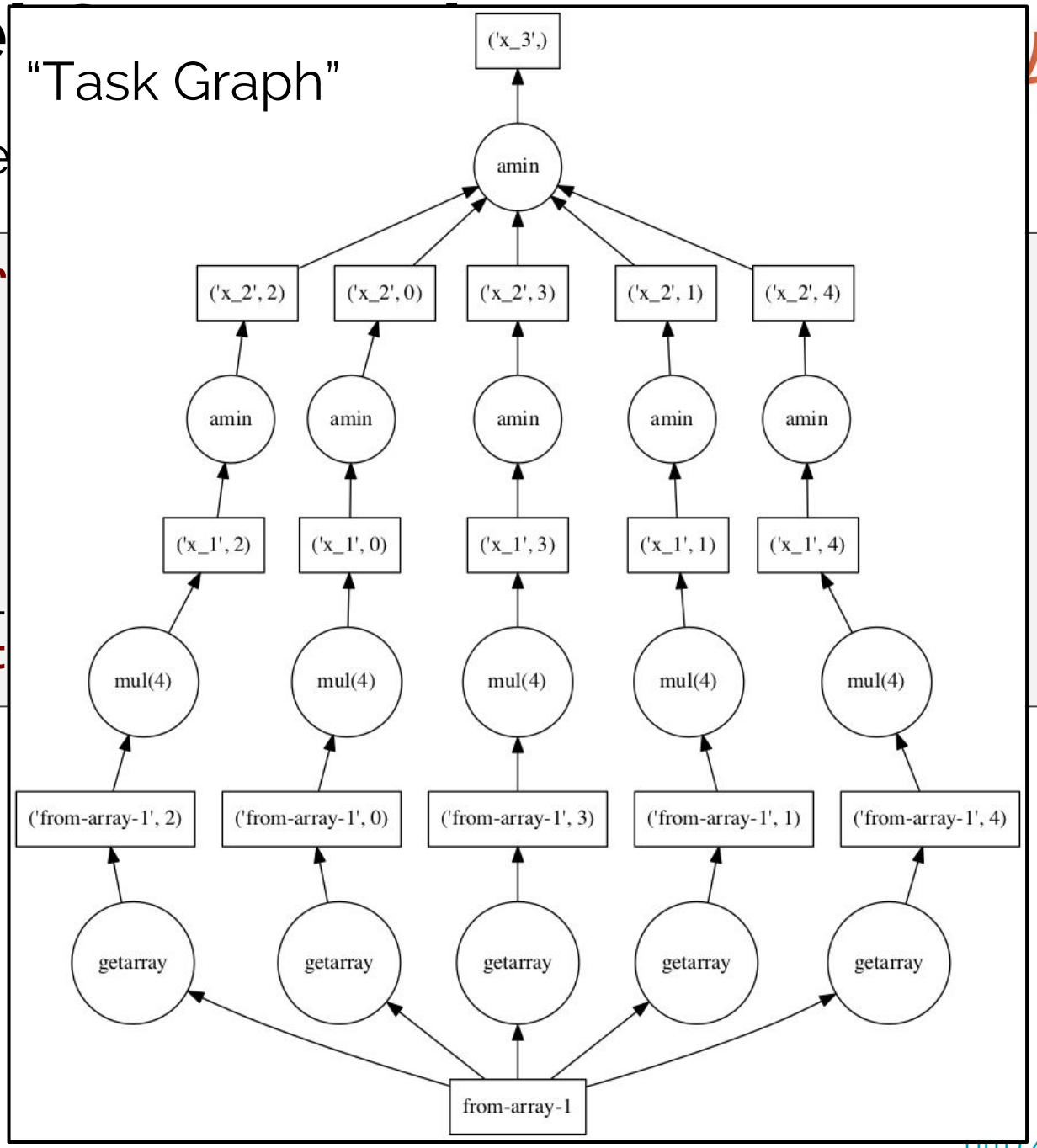
```
impor

a2 =

b2 =

b2_mi
print

dask.
```



DASK

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


```
$ 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

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

With a simple 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

With a simple 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!

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/

Remember:

Python is not a data science language.

But this may be its greatest strength.