Section 2 - numpy, scipy, and pandas

Jonathan Woodring *Los Alamos National Laboratory*

modules

- *numpy*: provides fast matrix and array operations
- *scipy*: a set of specific analysis tools
- *pandas*: a tabular data model for numpy arrays

data input/output in this section

- *sqlite3*: to and from SQLite data
- *numpy*: to and from binary data

In [ ]:
```
import numpy as np
# the standard name for numpy is np
import numpy

# of course you can do:
# from numpy import *
# but lets not
```

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

modules

- *numpy*: provides fast matrix and array operations
- *scipy*: a set of specific analysis tools
- *pandas*: a tabular data model for numpy arrays

data input/output in this section

- *sqlite3*: to and from SQLite data
- *numpy*: to and from binary data

In [ ]:
```
import numpy as np
# the standard name for numpy is np
import numpy

# of course you can not
# from numpy import *
# but, let's not

# also, we'll need pylab
%pylab inline --no-import-all
```

comparing numpy arrays to Python lists

Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-copy) 9:26

Home | tutorial/ | Section2_numpy_...

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

In [1]:
```
import numpy as np
# the standard name for numpy is np
import numpy

# of course you can do:
# from numpy import *
# but, lets not

# also, we'll need pyplot
%pylab inline --no-import-all
```

```
Populating the interactive namespace from numpy and matplotlib
```

comparing numpy arrays to Python lists

In [ ]:
```
import time

# two million long lists
A = range(0, 1000000)
B = range(0, 1000000)
# let me preallocate Cl, rather than Cl.append
Cl = [0] * 1000000

# lets time C = A + B (adding two vectors)
start = time.time()
```

Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) 9:27

Populating the interactive namespace from numpy and matplotlib

comparing numpy arrays to Python lists

In [ ]:
```
import time

# two million long lists
A = range(0, 1000000)
B = range(0, 1000000)
# let me preallocate Cl, rather than Cl.append
Cl = [0] * 1000000

# lets time C = A + B (adding two vectors)
start = time.time()
for i in range(0, len(A)):
    Cl[i] = A[i] + B[i]
list_time = time.time() - start

print 'done in %f seconds' % list_time
```

In [ ]:
```
# C = A + B, with numpy arrays

# initializing A and B from the lists
A = numpy.array(A, numpy.int64)
```

```
# lets time C = A + B (adding two vectors)
start = time.time()
for i in range(0, len(A)):
    Cl[i] = A[i] + B[i]
list_time = time.time() - start

print 'done in %f seconds' % list_time
```

```
done in 0.257647 seconds
```

In [3]:
```
# C = A + B, with numpy arrays

# initializing A and B from the lists
A = numpy.array(A, numpy.int64)
B = numpy.array(B, numpy.int64)

# lets time C = A + B, again
start = time.time()
Ca = A + B
array_time = time.time() - start

print 'done in %f seconds' % array_time
```

```
done in 0.002438 seconds
```

In [ ]:
```
# how much faster is numpy?
print 'C = A + B with lists:', list_time
```

```
B = numpy.array(B, numpy.int64)

# lets time C = A + B, again
start = time.time()
Ca = A + B
array_time = time.time() - start

print 'done in %f seconds' % array_time
```

```
done in 0.002438 seconds
```

In [ ]:
```
# how much faster is numpy?
print 'C = A + B with lists:', list_time
print 'C = A + B with numpy:', array_time
print 'speedup: ' + str(list_time / array_time) + ' times faster'
# just to show they are equal
print 'does Cl == Ca?', numpy.all(Cl == Ca)
```

- Numpy is fast
  1. Uses machine native code, rather than intepreted
  2. Data elements are typed, rather than PyObjects
  3. Vectorization (SIMD)
- Multi-dimensional indexing is more natural: [i,j] vs. [i][j]
- Operations over collections (vectors/arrays) like R and Matlab (implicit vectorization):
  - There's a reason we use Matlab, Fortran, R and Numpy for numerical

Home | tutorial/ | Section2_numpy

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help Python 2

```
print 'C = A + B with numpy:', array_time
print 'speedup:  ' + str(list_time / array_time) + '  times faster'
# just to know they are equal
print 'does Cl == Ca?', numpy.all(Cl == Ca)
```

```
C = A + B with lists: 0.257647037506
C = A + B with numpy: 0.002438068389
speedup: 105.676706435 times faster
does Cl == Ca? True
```

- Numpy is fast
  1. Uses machine native code, rather than intepreted
  2. Data elements are typed, rather than PyObjects
  3. Vectorization (SIMD)
- Multi-dimensional indexing is more natural: [i,j] vs. [i][j]
- Operations over collections (vectors/arrays) like R and Matlab (implicit vectorization):
  - There's a reason we use Matlab, Fortran, R and Numpy for numerical computing
- Critical mass from the community
  - It's the non-standard "standard" library for all things numerical.

A simple heat transfer simulation

Word Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-copy) 9:28

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Notebook saved | Python 2

A simple heat transfer simulation

In [ ]:

```
A = numpy.zeros((50, 50), numpy.float32)

A[20,20] = 1000000.0

print A[20,20], A[..]

def show_temp(array):
    plt.figure()
    plt.imshow(array)
    plt.set_cmap(...)
    plt.colorbar()
    plt.show()

show_temp(A)
```

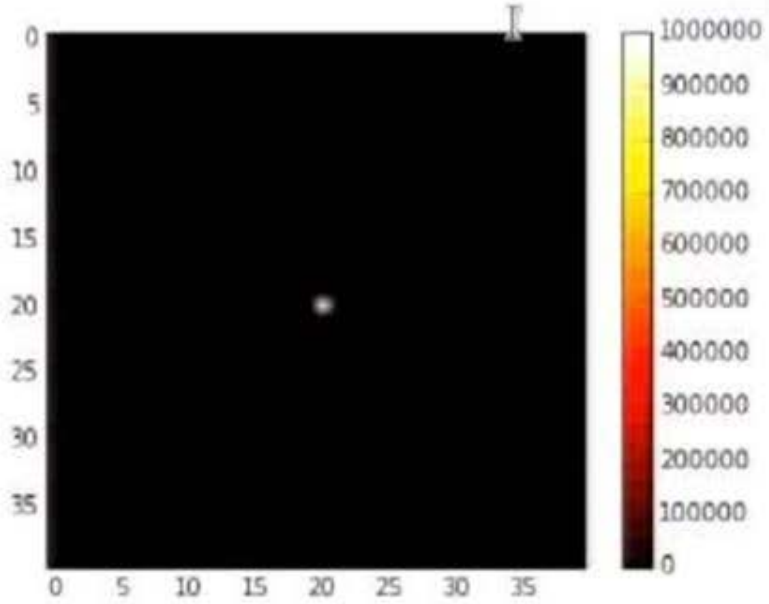
In [ ]:

```
def iterate(A, steps):
```

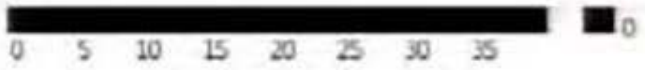
Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-copy)

```
    plt.figure()
    plt.imshow(array)
    plt.set_cmap('hot')
    plt.colorbar()
    plt.show()

# the colors will rescale each time we plot
show_temp(A)
```

```
A[20,20]: 1e+06
```




```
In [ ]: # our little simulation
```



In [6]:
```
# our little simulation
def iterate(A, steps):
    scale = .5 / 4.0
    # iterate for a bit
    for i in range(0, steps):
        # roll shifts an array on an axis
        left = numpy.roll(A, -1, 0)
        right = numpy.roll(A, 1, 0)
        up = numpy.roll(A, 1, 1)
        down = numpy.roll(A, -1, 1)

        # vectorized math
        delta = left + right + up + down - 4 * A

        # update A with dh/dt
        A = delta * scale + A

    return A
```

In [ ]:
```
# run it
A = iterate(A, 200)

# show where we ended
show_temp(A)
```

```
    return A
```

In [ ]:
```
# run it
A = iterate(A, 200)

# show where we ended
show_temp(A)
```

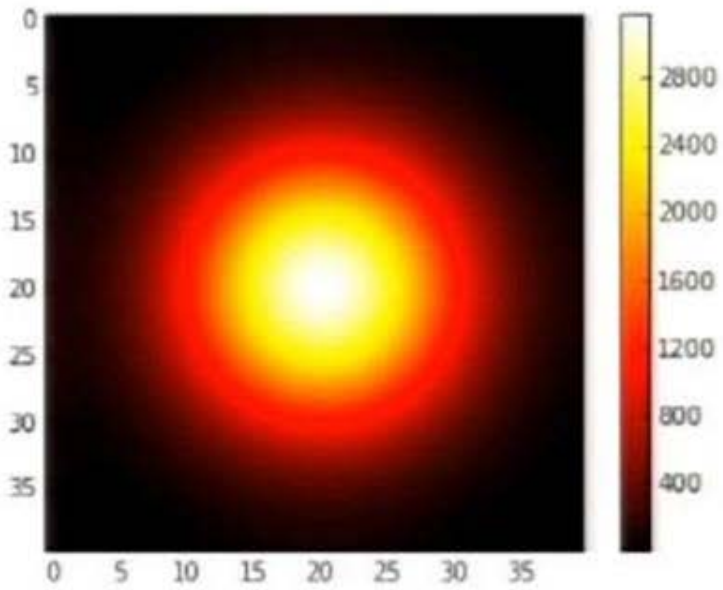
In [ ]:
```
print A[20, 20] # scalars

print A[20, 20] - A[21, 21]

print numpy.mean(A) # reductions

print numpy.sum(A - .95 * A)

print A*A # element-wise multiply

print A.dot(A) # matrix multiply (numpy.dot(A, A), too)
```

Reading and Writing Binary Data

In [ ]:
```
# store the state/checkpoint as a 'raw' binary file
```

```
    return A
```

In [7]:
```
# run it
A = iterate(A, 200)

# show where we ended
show_temp(A)
```



In [ ]:
```
print A[20, 20] # scalars
```

In [ ]:
```
print A[20, 20] # scalars

print A[20, 20] * A[21, 21]

print numpy.mean(A) # reductions

print numpy.sum(A - .95 * A)

print A*A # element-wise multiply

print A.dot(A) # matrix multiply (numpy.dot(A, A), too)
```

Reading and Writing Binary Data

In [ ]:
```
# store the state/checkpoint as a "raw" binary file
A.tofile('A.bin')

# just to show we actual alter B
B = numpy.zeros((40, 40), numpy.float32)
show_temp(B)
```

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

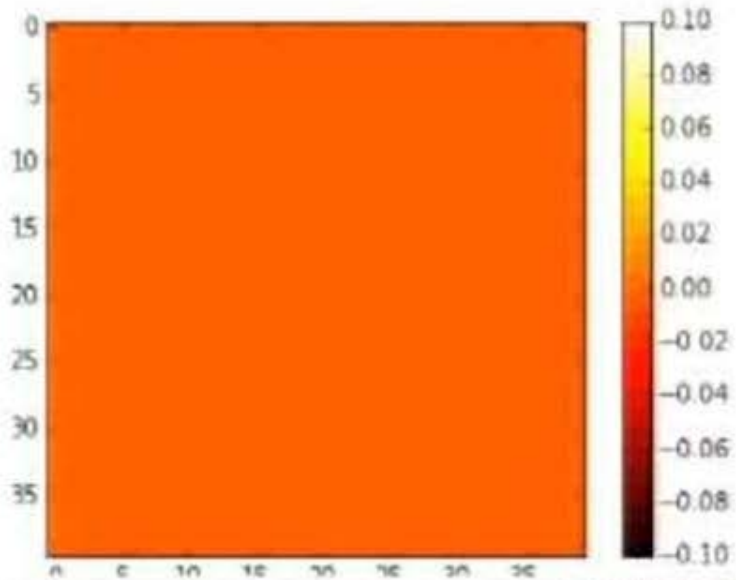
File Edit View Insert Cell Kernel Help | Python 2

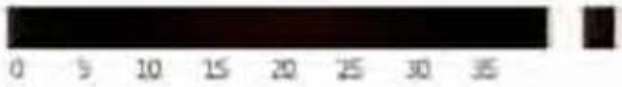
```
6
    31.54674721]
 ...,
 [  46.08278656   52.93860245   76.8356781  ...,  128.51898193   76.8356781
    52.93860626]
 [  27.44467163   31.54674721   45.85525513 ...,   76.8356781    45.8552627
6
    31.54674721]
 [  18.84791756   21.67605591   31.54674721 ...,   52.93860626
    31.54674721   21.67605591]]
[[ 173429.734375  185506.84375   222377.28125  ...,  285888.3125
   222377.28125   185506.84375 ]
 [ 185506.875     198424.984375  237862.953125 ...,  305796.8125
   237862.96875   198424.984375]
 [ 222377.265625  237862.9375    285139.625    ...,  366575.9375
   285139.65625   237862.96875 ]
 ...,
 [ 285888.3125    305796.8125    366575.9375   ...,  471270.875
   366575.9375    305796.8125  ]
 [ 222377.265625  237862.9375    285139.625    ...,  366575.9375
   285139.65625   237862.96875 ]
 [ 185506.875     198424.984375  237862.953125 ...,  305796.8125
   237862.96875   198424.984375]]
```

Reading and Writing Binary Data

Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) 9:31

```
# just to show we actual alter B
B = numpy.zeros((40, 40), numpy.float32)
show_temp(B)

# to read A back, have to specify the data type
# and possibly endianess (byte-order)
B = numpy.fromfile('A.bin', numpy.float32)

# we also have to reshape it to the same shape as A
# otherwise B will be a 1D array
B = numpy.reshape(B, (40, 40))

show_temp(B)
```





In [ ]:
```
# though if we use numpy's binary format,
# we don't have to worry about shape, type, and endianess
# when we reload it
numpy.save('A.npy', A)
C = numpy.load('A.npy')

# but this does mean if you are sharing binary data
# with another program it has to know how to parse
# .npy -- 'raw' binary is easier to share data in the
# short term

print numpy.all((A == B) & (A == C))
show_temp(C)
```

In [ ]:
```
# struct is one way to read binary if you want
# more control over reading the binary data
# say for instance, reading binary headers
import struct

f = open('A.bin', 'r')

# read 16 bytes and unpack them into 4 floats
print struct.unpack('ffff', f.read(16))
```

0 5 10 15 20 25 30 35

In [10]:
```
# though if we use numpy's binary format,
# we don't have to worry about shape, type, and endianess
# when we reload it
numpy.save('A.npy', A)
C = numpy.load('A.npy')

# but this does mean if you are sharing binary data
# with another program it has to know how to parse
# .npy -- "raw" binary is easier to share data in the
# short term

print numpy.all((A == B) & (A == C))
show_temp(C)
```

```
True
```

Home | tutorial/ | Section2_numpy_s

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

In [11]:
```
# struct is one way to read binary if you want
# more control over reading the binary data
# say for instance, reading binary headers
import struct

f = open('A.bin', 'r')

# read 16 bytes and unpack them into 4 floats
print struct.unpack('ffff', f.read(16))

f.close()
```

```
(4.0478854179382324, 4.3414187431335449, 5.2387661933898926, 6.7884297370910645)
```

Indices and Slices

In [ ]:
```
# clear it for now
A = numpy.zeros((40, 40), numpy.float32)

# ":" means all indices in that dimension
# row, column indexing
A[10,:] = 10000.0
```

Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) 9:33

Home | tutorial/ | Section2_numpy_

localhost[illegible]

Indices and Slices

In [ ]:

```
# [illegible]
A = numpy.zeros([illegible], numpy.float32)

# [illegible]
# [illegible]
A[[illegible],:] = [illegible]

show_temp(A)
```

In [ ]:

```
# [illegible] for a slice is
# [illegible]
# [illegible]
A[[illegible]:2] = A[[illegible]:2] * [illegible]

show_temp(A)
```

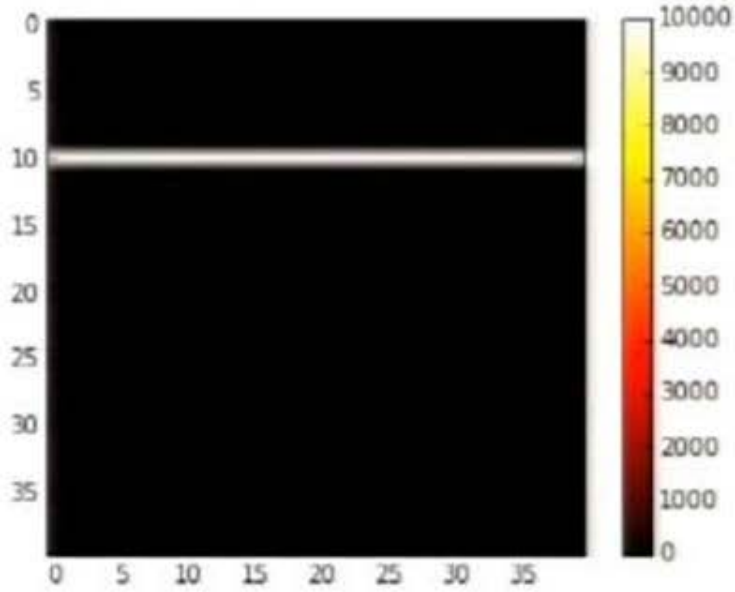
In [ ]:

```
# numpy [illegible]
# [illegible]
A[[illegible],:] = A[[illegible],:] * 1.5

show_temp(A)
```

Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-copy)

Indices and Slices

In [12]:
```
# clear it for now
A = numpy.zeros((40, 40), numpy.float32)

# ":" means all indices in that dimension
# row, column indexing
A[10,:] = 10000.0

show_temp(A)
```



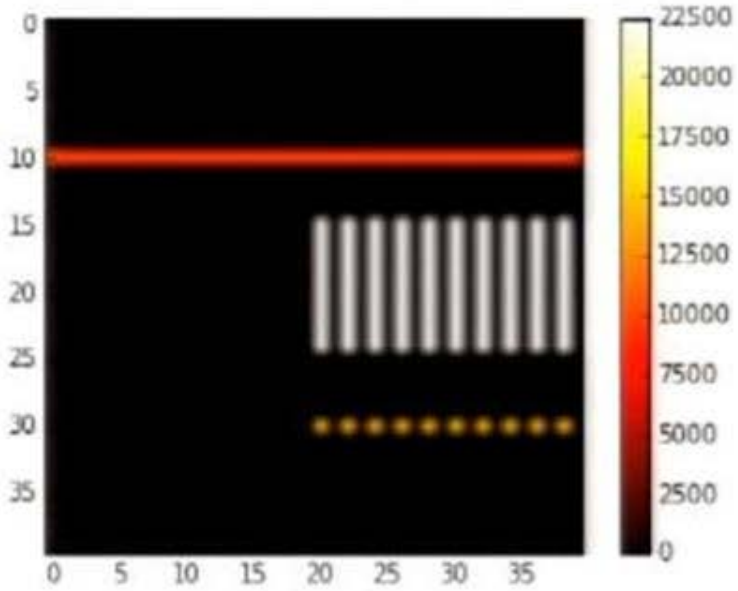
In [ ]:
```
# numpy will "broadcast", i.e., replicate the
# the input, if it is size 1 on a dimension
A[15:25,:] = A[30,:] * 1.5

show_temp(A)
```

In [ ]:
```
# transpose the input to the output
A[:,15:25] = numpy.transpose(A[15:25,:]) * 1.5

show_temp(A)
```

Selections

In [ ]:
```
# let's load our saved data
A = numpy.load('A.npy')

show_temp(A)

# and get the mean
mean_v = numpy.mean(A)
print 'mean:', mean_v
```

In [14]:
```
# numpy will "broadcast", i.e., replicate the
# the input, if it is size 1 on a dimension
A[15:25,:] = A[30,:] * 1.5

show_temp(A)
```



In [ ]:
```
# transpose the input to the output
A[:,15:25] = numpy.transpose(A[15:25,:]) * 1.5
```

Home | tutorial/ | Section2_numpy_s

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

Selections

In [ ]:
```
# let's load our saved data
A = numpy.load('A.npy')

show_temp(A)

# and get the mean
mean_v = numpy.mean(A)
print 'mean:', mean_v
```

In [ ]:
```
# .5 * mean(A) < A < mean(A)
B = (A < mean_v) & (A > .5 * mean_v)

# this is a boolean array where
# each element is True or False based
# on the condition
show_temp(B)
```

In [ ]:
```
# negate A where B is true
# i.e., B = (A < mean_v) & (A > .5 * mean_v)
A[B] = -A[B]

# we could have also put the expression
```

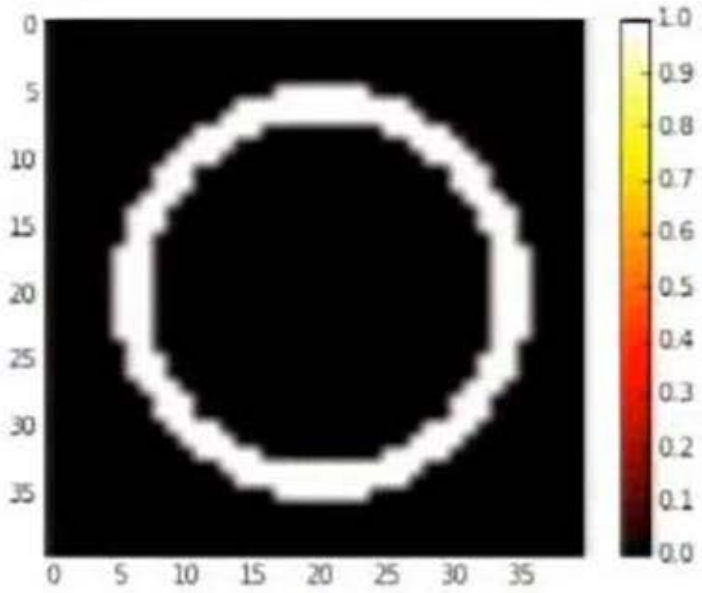
Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) 9:36

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

In [17]:
```
# .5 * mean(A) < A < mean(A)
B = (A < mean_v) & (A > .5 * mean_v)

# this is a boolean array where
# each element is True or False based
# on the condition
show_temp(B)
```



In [ ]:
```
# negate A where B is true
# i.e., B = (A < mean_v) & (A > .5 * mean_v)
```

Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) 9:37

Home | tutorial/ | Section2_numpy_s

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

In [ ]:
```
# negate A where B is true
# i.e., B = (A < mean_v) & (A > .5 * mean_v)
A[B] = -A[B]

# we could have also put the expression
# in the brackets, i.e.,
# A = A[(A < mean_v) & (A > .5 * mean_v)]

show_temp(A)
```

In [ ]:
```
# where is the temp greater than 2500?
I = numpy.where(A > 2500.0)

# I is the indices where it is True
# where I[0] are the is, and I[1] are the js
print I[0]
print I[1]
print I[0][0], I[1][0] # i_0,j_0

# we can modify A with the index array
A[I] = 0.0

# this is equivalent: A[I[0], I[1]] = 0.0

show_temp(A)
```

Work | Machine View Devices Help | pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) | 9:37

0 5 10 15 20 25 30 35

In [19]:
```
# where is the temp greater than 2500?
I = numpy.where(A > 2500.0)

# I is the indices where it is True
# where I[0] are the is, and I[1] are the js
print I[0]
print I[1]
print I[0][0], I[1][0] # i_0,j_0

# we can modify A with the index array
A[I] = 0.0

# this is equivalent: A[I[0], I[1]] = 0.0

show_temp(A)
```

```
[16 16 16 16 16 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 19 19 19 19
 19 19 19 19 19 20 20 20 20 20 20 20 20 20 21 21 21 21 21 21 21 21 21 21 22 22
 22 22 22 22 22 22 22 23 23 23 23 23 23 23 24 24 24 24 24]
[18 19 20 21 22 17 18 19 20 21 22 23 16 17 18 19 20 21 22 23 24 16 17 18 19
 20 21 22 23 24 16 17 18 19 20 21 22 23 24 16 17 18 19 20 21 22 23 24 16 17
 18 19 20 21 22 23 24 17 18 19 20 21 22 23 18 19 20 21 22]
16 18
```

In [ ]:
```
# since I are indices, we can adjust their positions
# and use it to copy that center area around
A[I[0], I[1] + 10] = A[I]
A[I[0], I[1] - 10] = A[I]
A[I[0] + 10, I[1]] = A[I]
A[I[0] - 10, I[1]] = A[I]

show_temp(A)
```

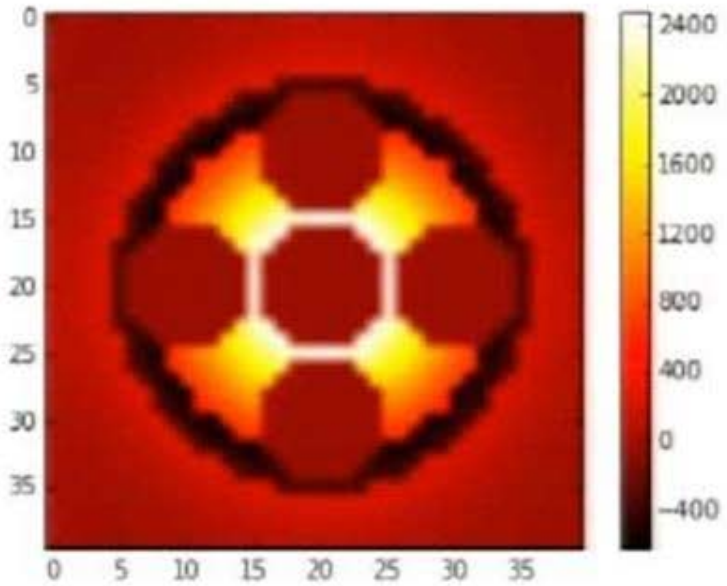
Followup numpy resources

Link to Numpy Reference

- Linear algebra
- Most mathematical functions
- Simple statistics and analysis functions
- Reductions and scans (any, all, sum, prod, etc.)
- Advanced indexing, selection, and set operations
- Masked arrays
- Interfacing with C/Cython
- Creating your own vectorized functions and reductions

0 5 10 15 20 25 30 35

In [20]:
```
# since I are indices, we can adjust their positions
# and use it to copy that center area around
A[I[0], I[1] + 10] = A[I]
A[I[0], I[1] - 10] = A[I]
A[I[0] + 10, I[1]] = A[I]
A[I[0] - 10, I[1]] = A[I]

show_temp(A)
```



Followup numpy resources

Link to Numpy Reference

- Linear algebra
- Most mathematical functions
- Simple statistics and analytic functions
- Reductions and scans (any, all, sum, prod, etc.)
- Advanced indexing, selection, and set operations
- Masked arrays
- Interfacing with C/Cython
- Creating your own vectorized functions and reductions
- Many other features: check out the reference

Reading SQLite Data

In [ ]:
```
# for next section, we'll need some sqlite files lying around
import sqlite3
```

Reading SQLite Data

In [21]:
```
# for next section, we'll need some sqlite this time around
import sqlite3

# get a database connection and a "cursor"
conn = sqlite3.connect('GOOG.sqlite3')
cursor = conn.cursor()

# now we can execute SQL on the database
cursor.execute('SELECT name, sql FROM sqlite_master WHERE type="table"')

# and get data from the cursor
i = cursor.next()
print i[0]
print i[1]
```

```
GOOG
CREATE TABLE GOOG (adj_close real, close real, date datetime, high real, lo
w real, open real, volume integer)
```

In [ ]:
```
# a useful module is sqlalchemy for interacting with many
# databases, but here I am just going to use raw SQL code
cursor.execute('SELECT date, close - open FROM GOOG WHERE close > open * 1.0

# the cursor is iterable
# dates where Google closed 7% higher and by how much
```

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

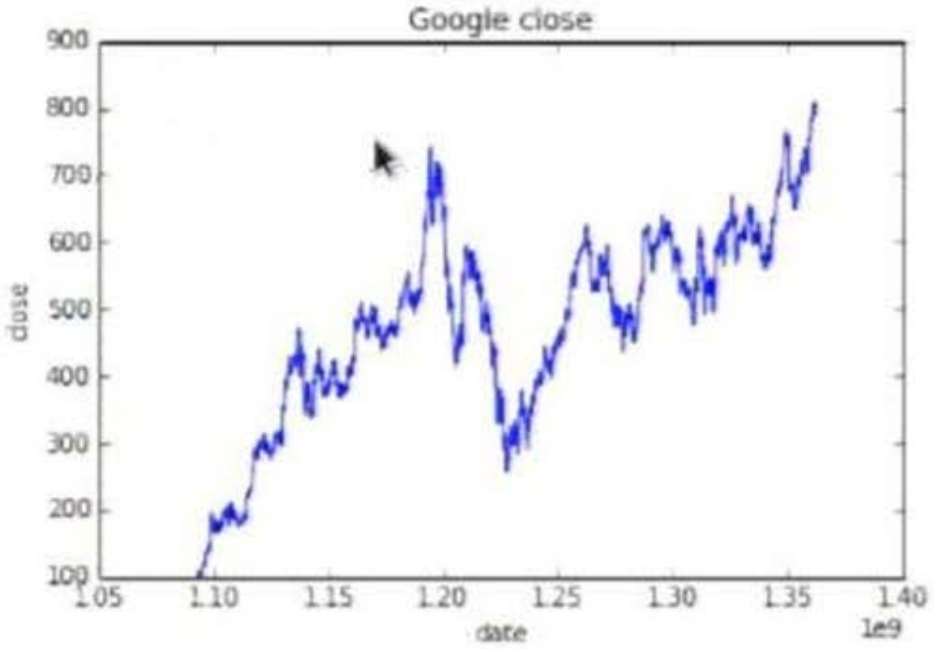
In [22]:
```
# a useful module is sqlalchemy for interacting with many
# databases, but here I am just going to use raw SQL code
cursor.execute('SELECT date, close - open FROM GOOG WHERE close > open * 1.C

# the cursor is iterable
# dates where Google closed 7% higher and by how much
for i in cursor:
    print i
```

```
(u'2004-08-20 00:00:00', 7.299999999999997)
(u'2004-11-11 00:00:00', 13.890000000000015)
(u'2008-10-13 00:00:00', 25.22999999999996)
(u'2008-10-28 00:00:00', 29.69999999999999)
```

In [ ]:
```
# a helper function to convert SQL data to numpy arrays
def query(cursor, command, dtype):
    cursor.execute(command)
    return numpy.array([i[0] for i in cursor], dtype)

# get our data vectors
X = query(cursor, 'SELECT date FROM GOOG', numpy.datetime64)
Y = query(cursor, 'SELECT close FROM GOOG', numpy.float32)

# plot the data
plt.figure()
plt.xlabel('date')
```

Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) 9:40

```
# plot the data
plt.figure()
plt.xlabel('date')
plt.ylabel('close')
plt.plot(X, Y)
plt.title('Google close')
plt.show()
```

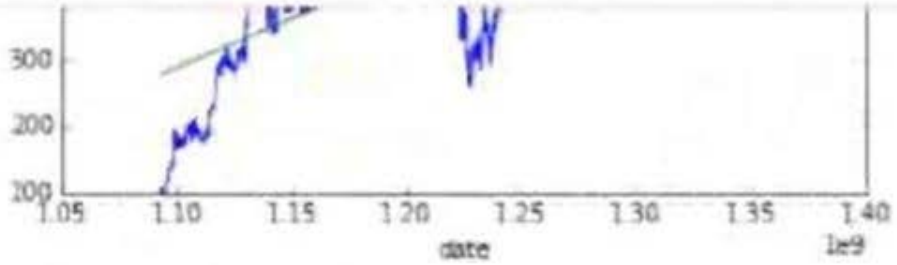


scipy

Polynomial Fit

In [ ]:
```
# Let's fit the Google stock to a curve
import scipy.optimize as optimize

# convert dates to integers for ease
X = numpy.array(X, numpy.int64)

# our linear model
# scipy.optimize uses introspection to figure
# out the number of model parameters
def poly(x, a, b):
    return a + b*x

# do a least squares fit
ab, cov = optimize.curve_fit(poly, X, Y)

# ab are the model parameters, and cov is covariance matrix
print 'model parameters:', ab
print 'covariance matrix:\n', cov
```

In [ ]:
```
# to plot the curve, we need to create
# a fixed function from our parameters
import functools
```



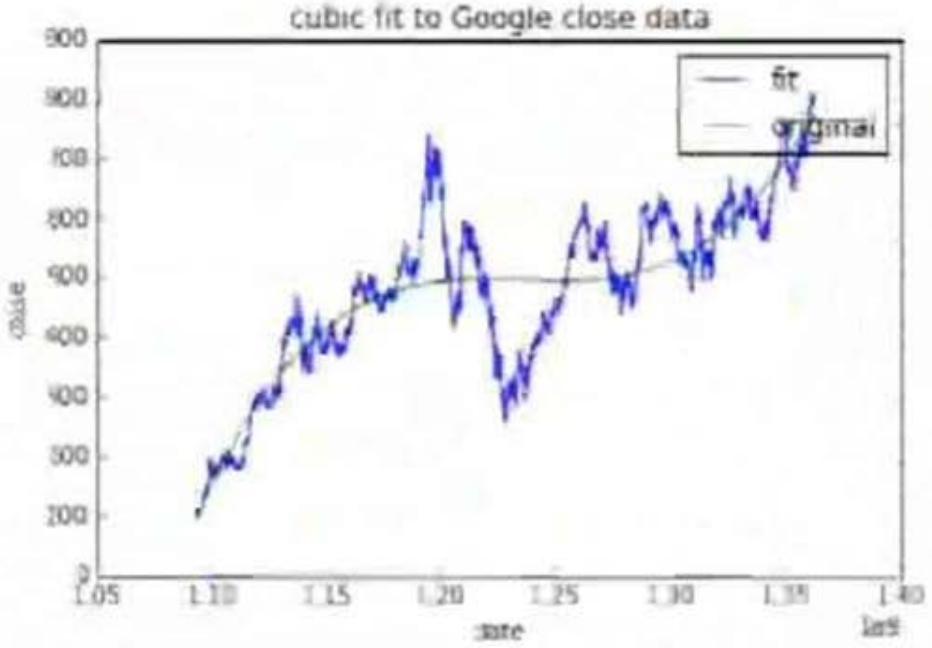
In [ ]:
```
# we can even do cubic fitting
# just add more parameters to your model curve
def poly(x, a, b, c, d):
    return a + b * x + c * x * x + d * x * x * x

# do the optimization and "freezing" the function
ab, cov = optimize.curve_fit(poly, X, Y)
fixed = functools.partial(poly, a=ab[0], b=ab[1], c=ab[2], d=ab[3])

# plot everything
plt.figure()
plt.plot(X, Y)
plt.plot(X, fixed(X))
plt.xlabel('date')
plt.ylabel('close')
plt.legend(['fit', 'original'])
plt.title('cubic fit to Google close data')
plt.show()
```

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

```
plt.plot(X, Y)
plt.plot(X, fixed(X))
plt.xlabel('date')
plt.xlabel('close')
plt.legend(['fit', 'original'])
plt.title('cubic fit to Google close data')
plt.show()
```



Power Spectrum

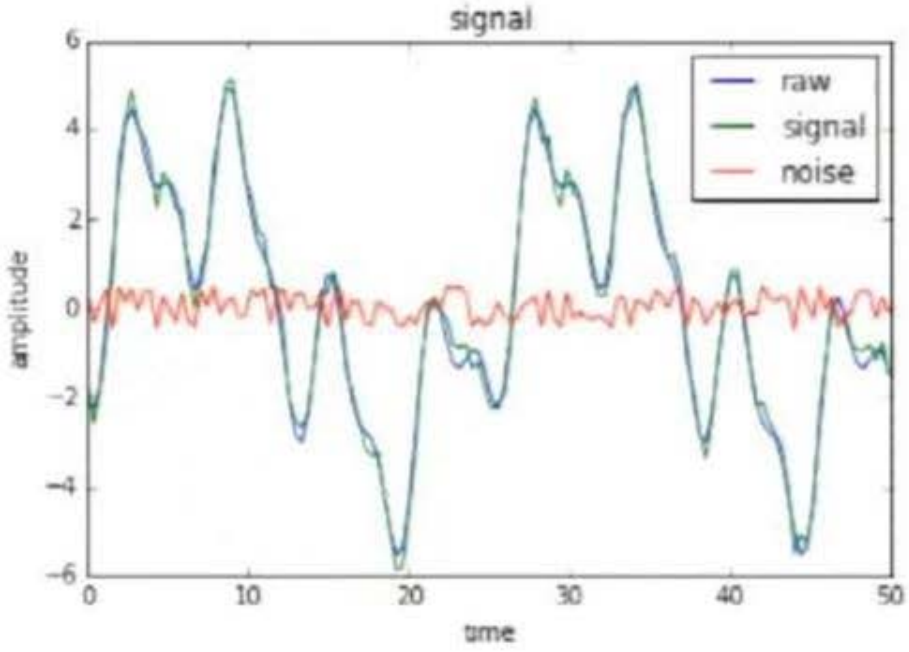
Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-cooy) 9:45

date 1e9

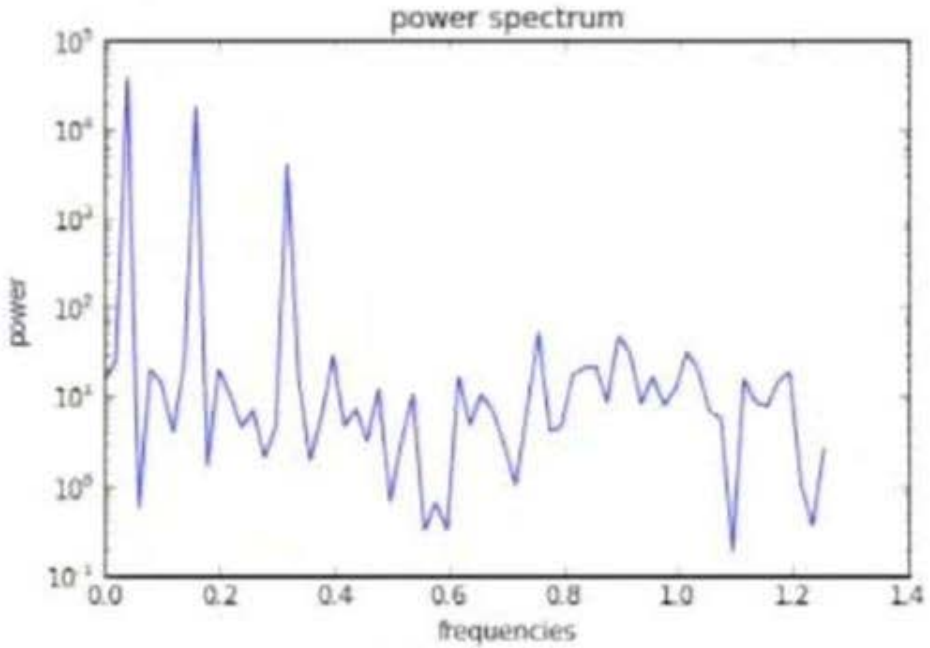
Power Spectrum

In [ ]:
```
# time samples
delta = numpy.pi / 8
time = numpy.arange(0, 16 * numpy.pi, delta)

# a made up signal
raw = numpy.sin(time * 2 + numpy.pi) +\
    numpy.sin(time - numpy.pi * .5) * 2 +\
    numpy.sin(time * .25) * 3

# some noise
noise = (numpy.random.rand(time.size) - .5)

# our noisy signal
signal = raw + noise

# let's plot it
plt.figure()
plt.plot(time, raw)
plt.plot(time, signal)
plt.plot(time, noise)
plt.title('signal')
plt.xlabel('time')
```

```
plt.legend(['raw', 'signal', 'noise'])
plt.show()
```



In [ ]:
```
# let's do some Fourier analysis
from scipy import fftpack

# standard definition of power spectrum
power = numpy.abs(fftpack.fft(signal)) ** 2

# frequencies from sampling spacing
```



Convolution

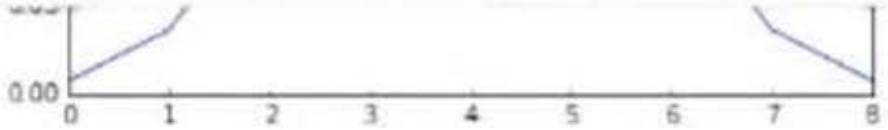
In [ ]:
```
# let's smooth the signal
from scipy.signal import gaussian

# gaussian kernel of 9 width, stddev of 1.5
kernel = gaussian(9, 1.5)
```

frequencies

Convolution

In [ ]:
```
# let's smooth the signal
from scipy.signal import gaussian

# gaussian kernel of 9 width, stddev of 1.5
kernel = gaussian(9, 1.5)
area = numpy.trapz(kernel)
kernel = kernel / area

# plot the kernel
plt.figure()
plt.plot(kernel)
plt.title('Gaussian kernel')
plt.show()
```

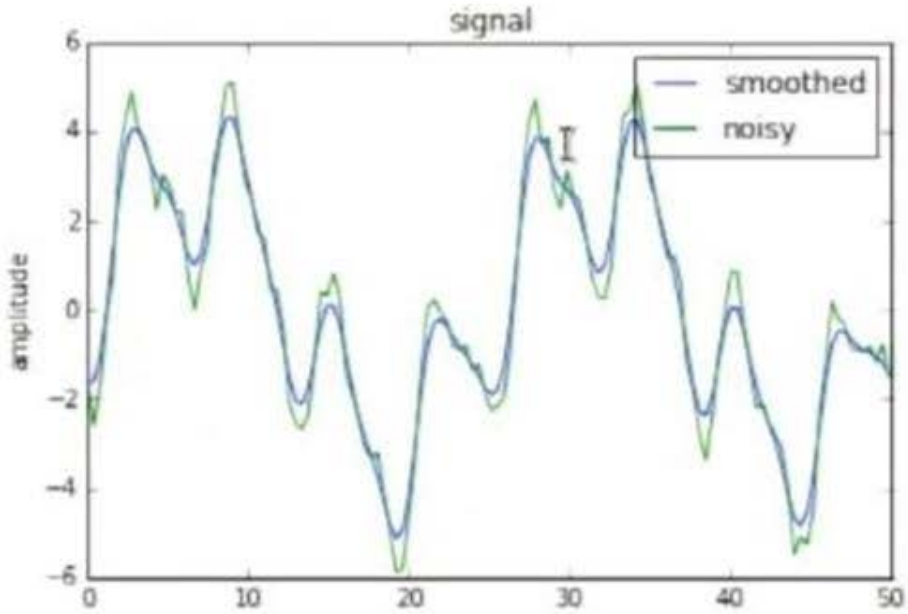
In [ ]:
```
# convolution
from scipy.signal import convolve

# pad the signal on both ends
smooth = numpy.concatenate([signal[-4:], signal, signal[:4]])

# convolve it
```



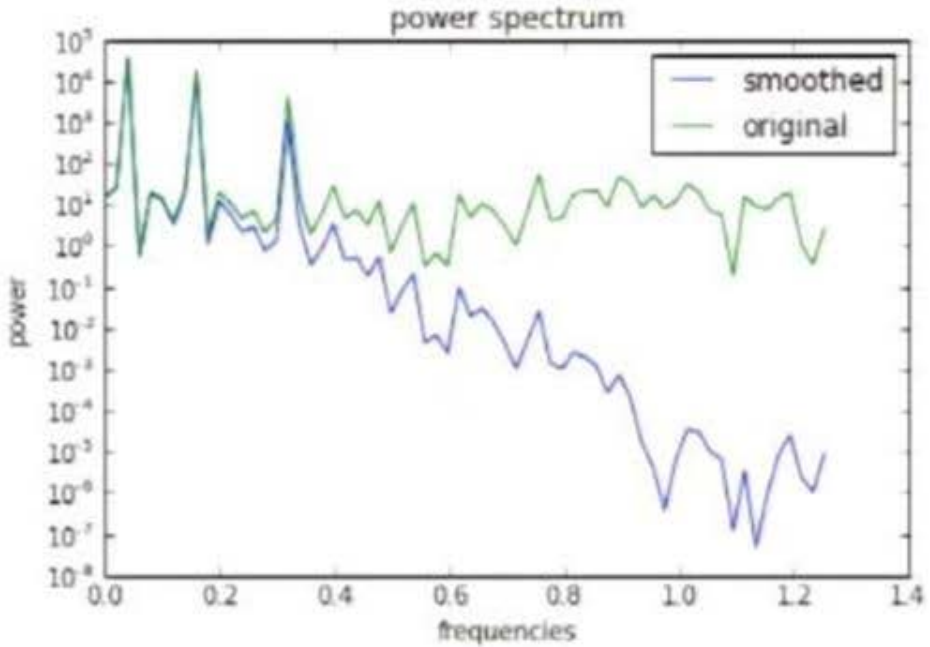
In [ ]:
```
# convolution
from scipy.signal import convolve

# pad the signal on both ends
smooth = numpy.concatenate([signal[-4:], signal, signal[:4]])

# convolve it
smooth = convolve(smooth, kernel, 'valid')

# plot the data
plt.figure()
plt.plot(time, smooth)
plt.plot(time, signal)
plt.title('signal')
plt.xlabel('time')
plt.ylabel('amplitude')
plt.legend(['smoothed', 'noisy'])
plt.show()
```

In [ ]:
```
# power spectrum again
smooth_power = numpy.abs(fftpack.fft(smooth)) ** 2
smooth_power = smooth_power[i]
```

```
# convolve it
smooth = convolve(smooth, kernel, 'valid')

# plot the data
plt.figure()
plt.plot(time, smooth)
plt.plot(time, signal)
plt.title('signal')
plt.xlabel('time')
plt.ylabel('amplitude')
plt.legend(['smoothed', 'noisy'])
plt.show()
```



```
plt.xlabel('frequencies')
plt.ylabel('power')
plt.yscale('log')
plt.legend(['smoothed', 'original'])
plt.show()
```



Writing data to SQLite

In [ ]:
```
# we'll create a new database
```

frequencies

Writing data to SQLite

In [ ]:
```
# we'll create a new database
conn = sqlite3.connect('signals.sqlite3')
cursor = conn.cursor()

# and a table to store our data in it
cursor.execute('CREATE TABLE signal (time real, raw real, smooth real)')

# now we just have to insert all of the data into it
for i, j, k in zip(time, signal, smooth):
    cursor.execute('INSERT INTO signal VALUES (?, ?, ?)', (i, j, k))

conn.commit() # commit the transaction
```

In [ ]:
```
# now we can query the data
cursor.execute('SELECT time FROM signal WHERE abs(raw - smooth) > 1')

# the times where the smooth signal
# deviates from the original by 1
for i in cursor:
    print i[0]
```

```
for i in cursor:
    print i[0]
```

```
6.67568436888
38.4845100065
```

pandas

Link to pandas reference

- Provides "data frames," which are similar to **R** data frames
  - Has functionality similar to plyr/dplyr
  - I.e., some DSL for SQL-like queries without SQL
- Provides tabular representations of your numpy, dict, and list data
- A useful syntax addition on top of numpy for doing complex data slicing and selection
- Easy input and output to tabular/columnar data formats
  - Direct to and from CSV, HDF, SQLite, etc.

In [ ]:
```
# we'll sqlachemy this time around
# which can also connect to other databases
import sqlalchemy
engine = sqlalchemy.create_engine("sqlite:///iris.sqlite3")

# pandas knows how to read a table using sqlachemy
```

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

```
import pandas
import pandas as pd # this is typical

iris = pandas.read_csv('iris.csv', engine)
print iris.columns

# iris is a table/pandas DataFrame
iris.head() # there is also tail()
```

```
Index([u'index', u'Unnamed: 0', u'SepalLength', u'SepalWidth', u'PetalLengt
h', u'PetalWidth', u'Species'], dtype='object')
```

Out[35]:

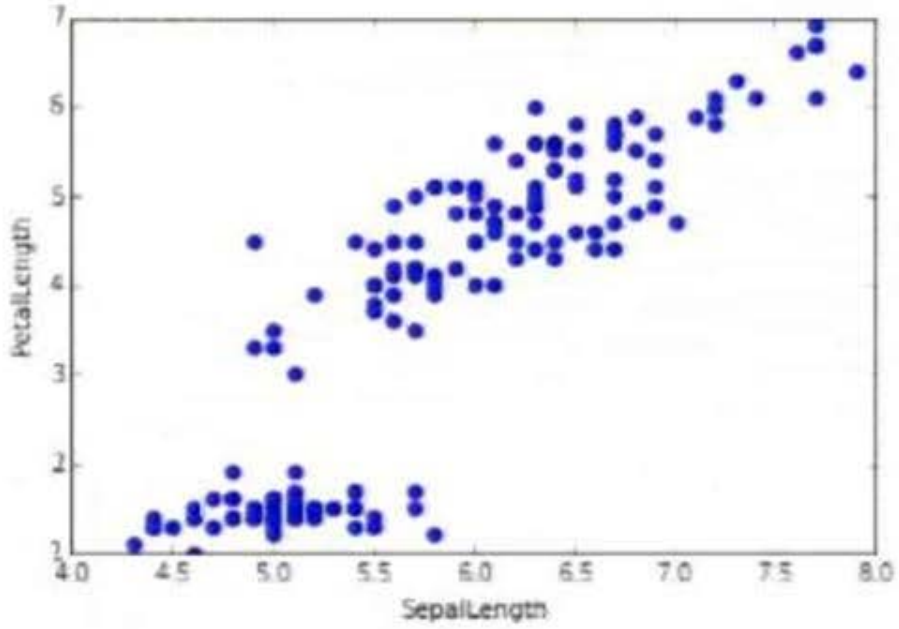
	index	Unnamed: 0	SepalLength	SepalWidth	PetalLength	PetalWidth	Spec
0	0	1	5.1	3.5	1.4	0.2	setos
1	1	2	4.9	3.0	1.4	0.2	setos
2	2	3	4.7	3.2	1.3	0.2	setos
3	3	4	4.6	3.1	1.5	0.2	setos
4	4	5	5.0	3.6	1.4	0.2	setos

In [ ]:

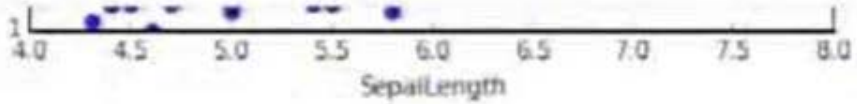
```
# we can plot the data
def plot_iris(iris, x, y):
    plt.figure()
    # select two columns as series in pandas dataframe
```

Work Machine View Devices Help pyvis64 (Linked Base for pyvis64 and pyvis64-copy) 9:50

```
    plt.figure()
    # select two columns (a Series in pandas parlance)
    plt.plot(iris[x], iris[y], 'o')
    plt.xlabel(x)
    plt.ylabel(y)

show_iris(iris, 'SepalLength', 'PetalLength')
```



In [ ]:
```
# selecting multiple columns by a list of column names
iris = iris[['SepalLength', 'Sepalwidth', 'PetalLength', 'Petalwidth', 'Spec
iris.head()
```

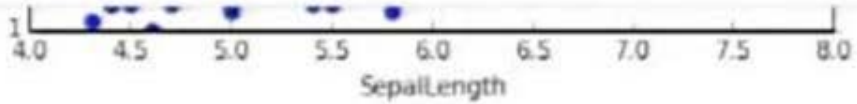


In [ ]:
```
# selecting multiple columns by a list of column names
iris = iris[['SepalLength', 'SepalWidth', 'PetalLength', 'Petalwidth', 'Spec
iris.head()
```

In [ ]:
```
# [] is overloaded to also do index selection
# i.e., row selection
sel = iris[10:51:10]

show_iris(sel, 'SepalLength', 'PetalLength')
sel
```

In [ ]:
```
# with [] you can also boolean select
sel = iris[iris.Species == 'versicolor']

show_iris(sel, 'SepalLength', 'PetalLength')
sel.head()
```

In [ ]:
```
# and use multiple columns in your query
sel = iris[(iris.Species == 'versicolor') & (iris.PetalLength < 3.9)]

show_iris(sel, 'SepalLength', 'PetalLength')
sel
```



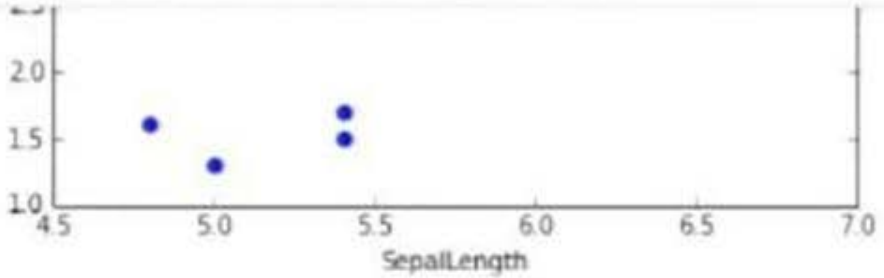
In [37]:
```
# selecting multiple columns by a list of column names
iris = iris[['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Spec
iris.head()
```

Out[37]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [ ]:
```
# [] is overloaded to also do index selection
# i.e., row selection
sel = iris[10:51:10]

show_iris(sel, 'SepalLength', 'PetalLength')
sel
```

In [ ]:
```
# with [] you can also boolean select
```



In [39]:
```
# with [] you can also boolean select
sel = iris[iris.Species == 'versicolor']

show_iris(sel, 'SepalLength', 'PetalLength')
sel.head()
```

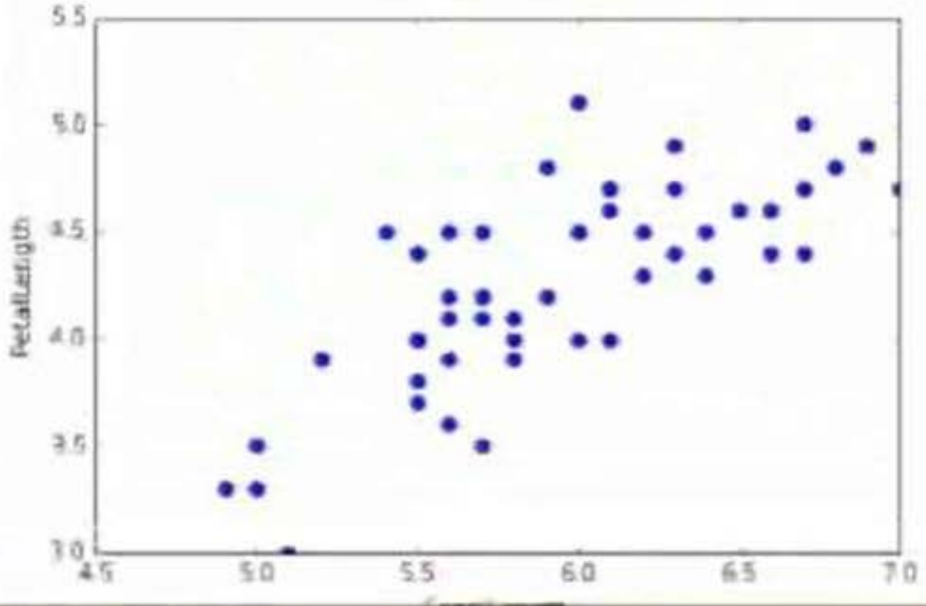
Out[39]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor

```
sel.head()
```

Out[39]:

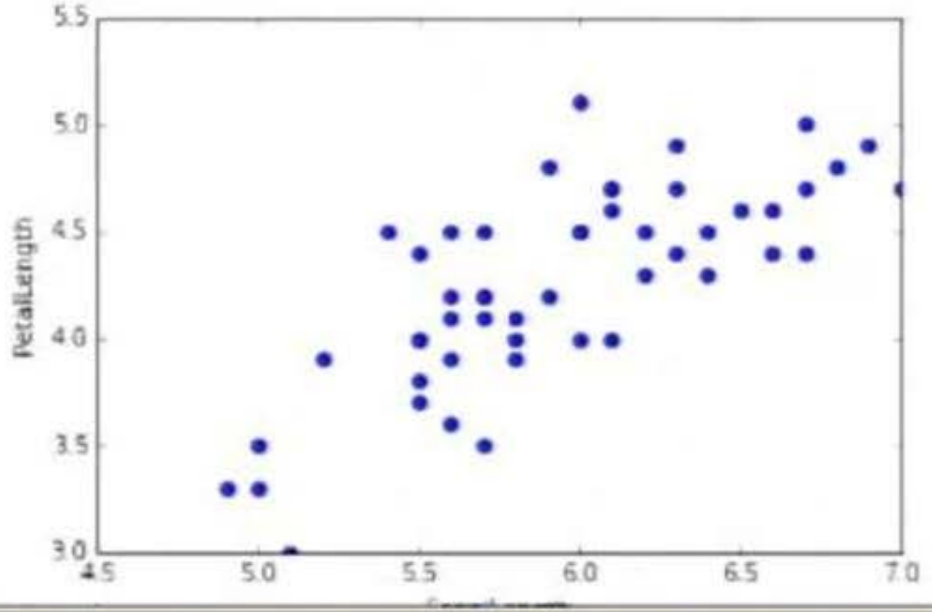
	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor



```
show_iris(sel, 'SepalLength', 'PetalLength')
sel.head()
```

Out[39]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor



Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help Python 2

In [ ]:
```
# we can change any column to be an index
iris.index = iris.Species
iris.head()

# there is also multi-dimensional indexing,
# for N-D indexing, but we won't go into that
```

In [ ]:
```
# then that means you can do indexing by labels
# label indexing in pandas is right-inclusive
# i.e., different from list/array slicing
sel = iris['setosa':'virginica']

show_iris(sel, 'SepalLength', 'PetalLength')
sel[::25]
```

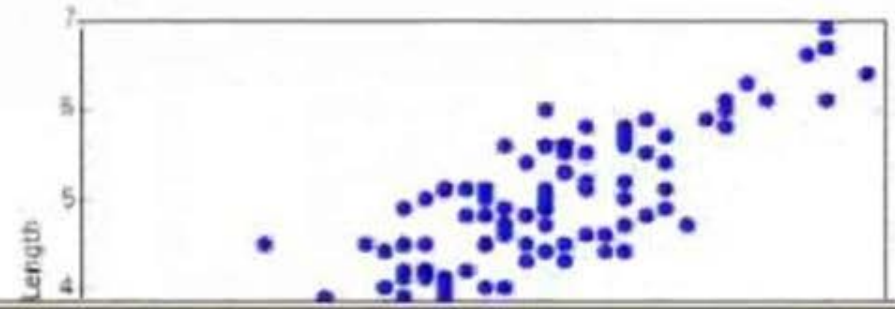
In [ ]:
```
# to do more explicit selection, use
# .loc, .iloc, and .ix (labels, integers, mixed)

# this selects a column (a Series in pandas parlance)
# i.e., dataframe.loc[row(s), column(s)]
sel = iris.loc[iris.PetalWidth < 1.8, 'PetalLength']

plt.figure()
plt.plot(sel, 'o')
plt.ylabel('PetalLength')
plt.show()
```

Workspace 1 9:54

```
# label indexing in pandas is right inclusive
# i.e., different from list/array slicing
sel = iris['setosa':'virginica']

show_iris(sel, 'SepalLength', 'PetalLength')
sel[::25]
```

Out[42]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
Species					
setosa	5.1	3.5	1.4	0.2	setosa
setosa	5.0	3.5	1.6	0.2	setosa
versicolor	7.0	3.2	4.7	1.4	versicolor
versicolor	6.6	3.0	4.4	1.4	versicolor
virginica	6.3	3.3	6.0	2.5	virginica
virginica	7.2	3.2	6.0	1.8	virginica



Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help Python 2

In [ ]:
```
# to do more explicit selection, use
# .loc, .iloc, and .ix (labels, integers, mixed)

# this selects a column (a Series in pandas parlance)
# i.e., datapanel.loc[row(s), column(s)]
sel = iris.loc[iris.PetalWidth < 1.8, 'PetalLength']

plt.figure()
plt.plot(sel, 'o')
plt.ylabel('PetalLength')
plt.show()
```

In [ ]:
```
# you can also do multiple rows and columns
sel = iris.loc[:,['PetalLength', 'SepalLength']]

show_iris(sel, 'PetalLength', 'SepalLength')
sel.head()
```

In [ ]:
```
# as you can see, pandas is very SQL-like/R-like
# and you can even do aggregation
sel = iris.groupby('Species')
sel.aggregate(numpy.mean)
```

In [ ]:
```
# and do something like
B = (iris.SepalLength > 6.0) & (iris.SepalWidth < 2.7)
sel = iris.loc[B,['Species', 'PetalWidth', 'PetalLength']]
```

Workspace 1 9:55

Home | tutorial/ | Section2_numpy_s

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help | Python 2

In [45]:
```
# as you can see, pandas is very SQL-like/R-like
# and you can even do aggregation
sel = iris.groupby('Species')
sel.aggregate(numpy.mean)
```

Out[45]:

	SepalLength	SepalWidth	PetalLength	PetalWidth
Species				
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

In [ ]:
```
# and do something like
B = (iris.SepalLength > 6.0) & (iris.SepalWidth < 2.7)
sel = iris.loc[B,['Species', 'PetalWidth', 'PetalLength']]

show_iris(iris[B], 'PetalWidth', 'PetalLength')
sel.groupby('Species').aggregate(numpy.max)
```

In [ ]:
```
# you can even do joins
df = pandas.DataFrame({'Species': ['versicolor', 'virginica'], 'researcher':
```

Workspace 1 | 9:56

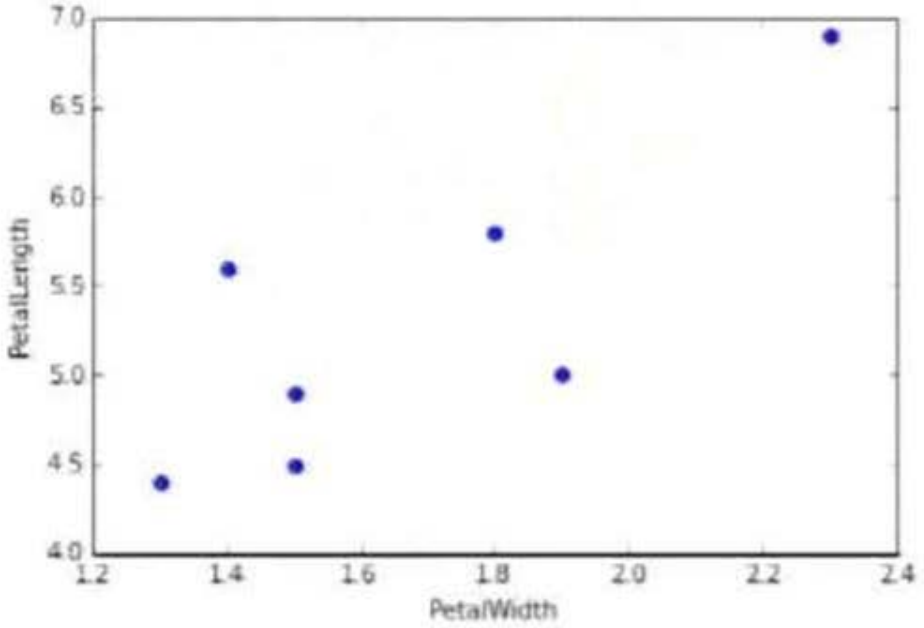
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

In [ ]:
```
# and do something like
B = (iris.SepalLength > 6.0) & (iris.SepalWidth < 2.7)
sel = iris.loc[B,['Species', 'PetalWidth', 'PetalLength']]

show_iris(iris[B], 'PetalWidth', 'PetalLength')
sel.groupby('Species').aggregate(numpy.max)
```

In [ ]:
```
# you can even do joins
df = pandas.DataFrame({'Species': ['versicolor', 'virginica'], 'researcher':

sel = pandas.merge(iris, df, on='Species')
sel[::15]
```

In [ ]:
```
# and when you are all done, output the data back to
# SQLite (or any other database with sqlalchemy)

engine = sqlalchemy.create_engine('sqlite:///select.sqlite3')
sel.to_sql('select', engine)

copy = pandas.read_sql_table('select', engine)
copy[::15]
```

versicolor	1.5	4.9
virginica	2.3	6.9



In [ ]:
```
# you can even do joins
df = pandas.DataFrame({'Species': ['versicolor', 'virginica'], 'researcher':

sel = pandas.merge(iris, df, on='Species')
sel[::15]
```

Home | tutorial/ | Section2_numpy_

localhost:8888/notebooks/tutorial/Section2_numpy_scipy_pandas.ipynb#

File Edit View Insert Cell Kernel Help Python 2

```
sel[::15]
```

Out[47]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species	researcher
0	7.0	3.2	4.7	1.4	versicolor	Roger
15	6.7	3.1	4.4	1.4	versicolor	Roger
30	5.5	2.4	3.8	1.1	versicolor	Roger
45	5.7	3.0	4.2	1.2	versicolor	Roger
60	6.5	3.2	5.1	2.0	virginica	Jane
75	7.2	3.2	6.0	1.8	virginica	Jane
90	6.7	3.1	5.6	2.4	virginica	Jane

In [ ]:

```
# and when you are all done, output the data back to
# SQLite (or any other database with sqlalchemy)

engine = sqlalchemy.create_engine('sqlite:///select.sqlite3')
sel.to_sql('select', engine)

copy = pandas.read_sql_table('select', engine)
copy[::15]
```

In [ ]:

Workspace 1

9:58