GENERAL ASSEMBLY

DATA SCIENCE PYTHON INTRO 2

APRIL 16, 2014

AGENDA

PART 1: PYTHON AND THE DATA SCIENCE WORKFLOW

I. THE DATA SCIENCE WORKFLOW

PART 2: USEFUL PYTHON LIBARIES

I. NUMPY
II. PANDAS
III. SCIPY
IV. SCIKITS
V. MATPLOTLIB

PART 3: WRAP-UP & NEXT STEPS

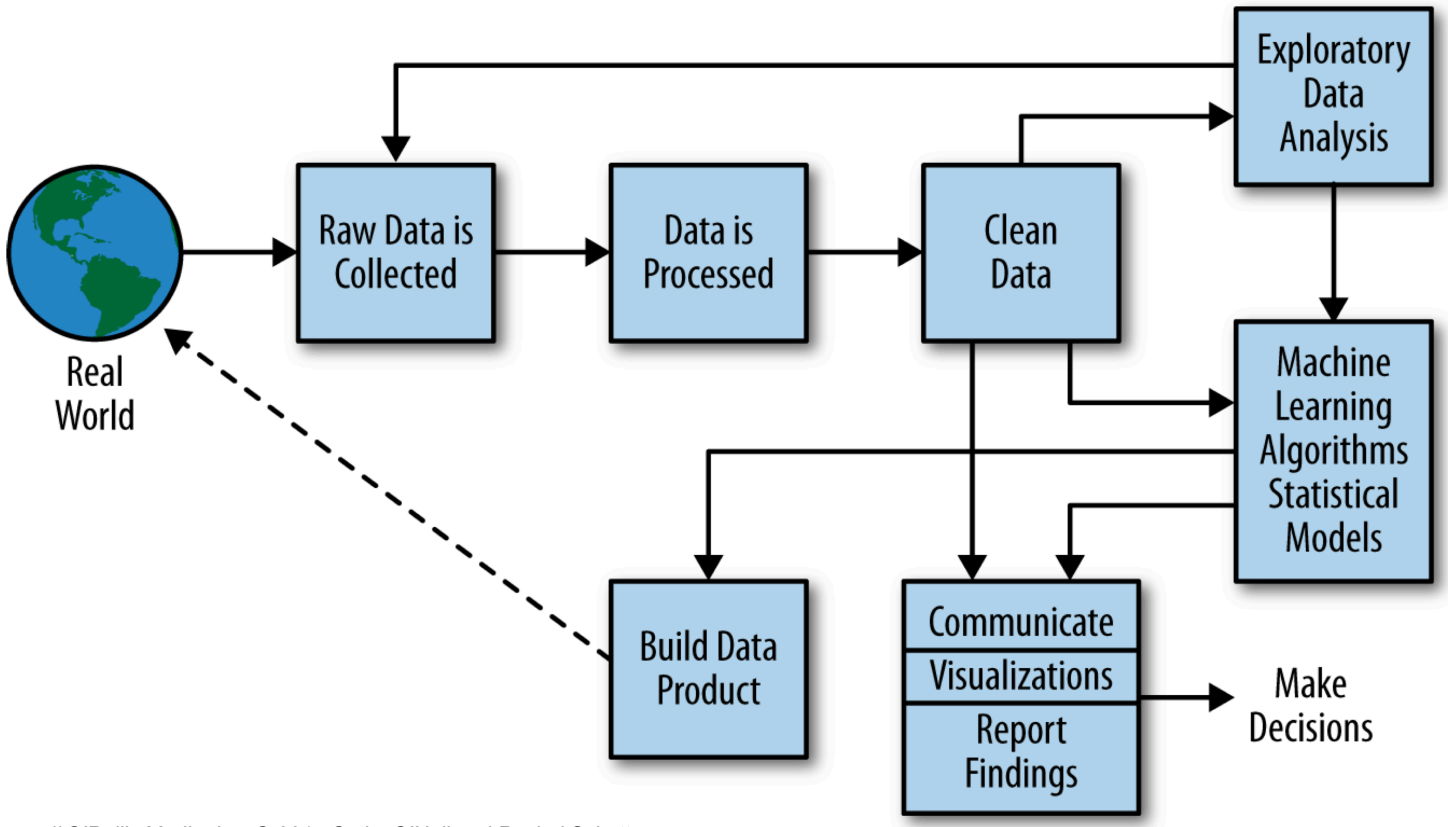
I. RESOURCES FOR FURTHER EXPLORATION

INTRODUCTION

What is data science?

The extraction of useful information and knowledge from large volumes of data, in order to improve decision-making
... or to tell a compelling story.

(PYTHON IN) THE DATA SCIENCE WORKFLOW



Web data-access APIs:

- urllib // (python standard library web crawling)
- Beautiful Soup (html/xml tree parsing)
- scrapy // (web crawling to extract structured data)
- python-twitter
- python-linkedin
- python-instagram

The goal of “Pre-processing” is to convert data into a standard format.

A standard format allows for input to algorithms to be standardized.

Some algorithms require inputs to be particularly formatted.

Relevant Libraries:

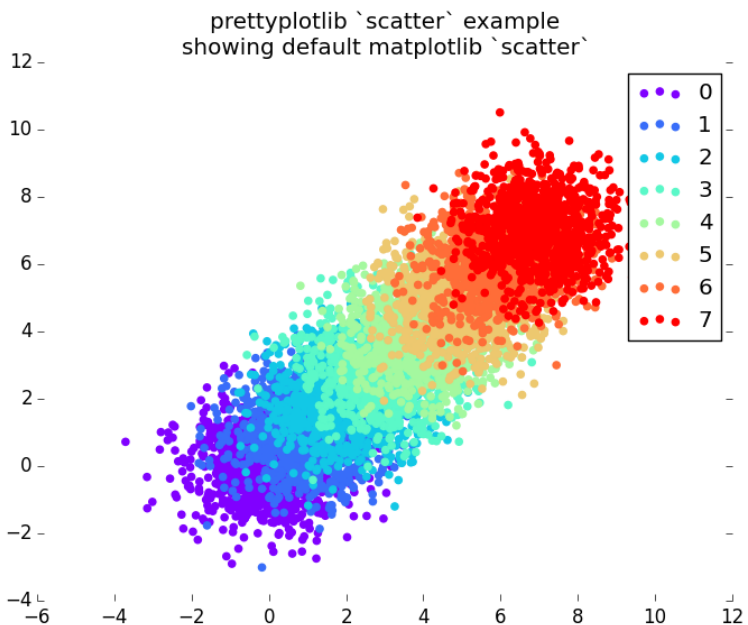
- NumPy
- Pandas

Try different algorithms to determine the optimal choice

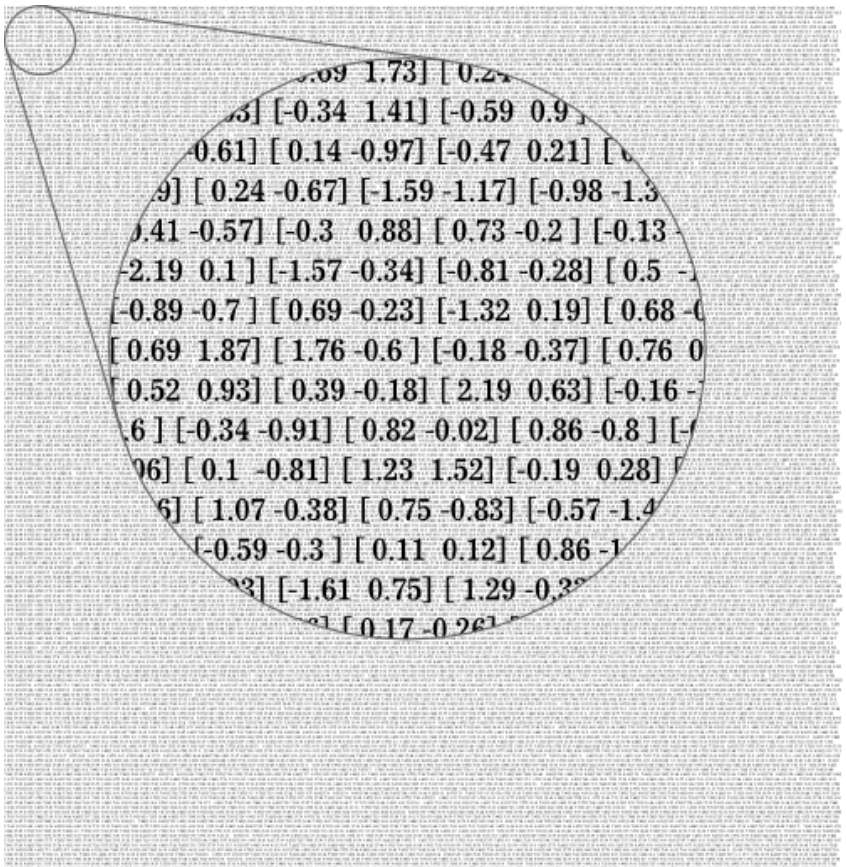
Relevant Libraries:

- SciPy
- scikit-learn

Since seeing is believing...



Relevant Library: matplotlib



http://bit.ly/1gCXPtP // http://http://blog.olgabotvinnik.com

USEFUL LIBRARIES FOR DATA SCIENCE (ANALYSIS, MODELING, & VISUALIZATION)

NumPy	"Fundamental package for scientific computing" [1]
Matplotlib	Plotting (& histograms, power spectra, bar charts, errorcharts, scatterplots, etc)
SciPy	"Fundamental library for scientific computing" [2]
Pandas	Python Data Analysis
Scikits	Application domain toolkits

[1] n-dimensional array object, broadcasting functions, linear algebra

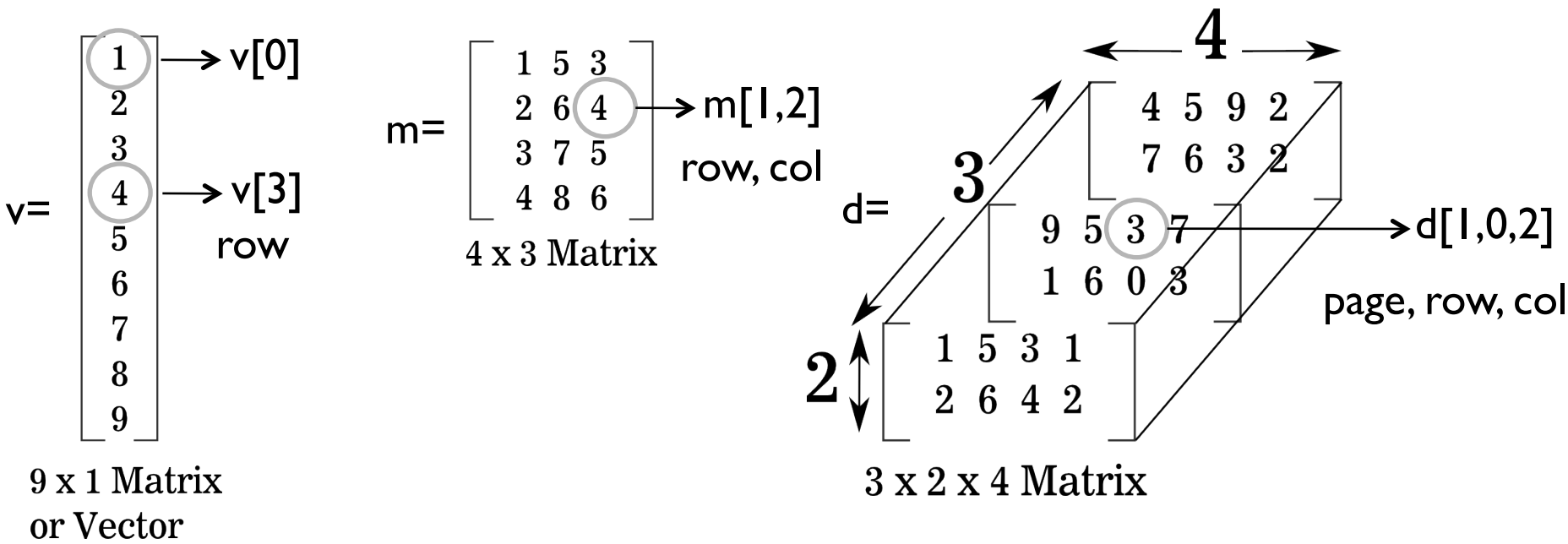
[2] numerical integration, interpolation, optimization, linear algebra^{++}, FFT

NumPy

> "add[s] support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays"
>
> - Wikipedia

http://en.wikipedia.org/wiki/NumPy

Vectors, arrays, and matrices



ndarray object creation, indexing and slicing (1-dimensional)

```
>>> import numpy as np
>>> a = np.array([0, 1, 5, 7, 6, 5, 2, 3, 8, 9])
>>> a[3]
7
>>> a[3:7]
array([7, 6, 5, 2])
>>> a[7:3:-1]
array([3, 2, 5, 6])
>>> b=np.array([1,5,7])
>>> a[[b]]
array([1, 5, 3])
>>> a[a > 5]
array([7, 6, 8, 9])
```

More methods to quickly create ndarray objects (1-d) and (2-d)

```
>>> b = np.arange(1, 20, 3)
>>> b
array([ 1,  4,  7, 10, 13, 16, 19])
>>> a = np.ones((3, 3))
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> b = np.zeros((2, 2))
>>> b
array([[ 0.,  0.],
       [ 0.,  0.]])
```

MATRIX MULTIPLICATION (REMINDER)

RULE: Two matrices can be multiplied only when the number of columns in the first equals the number of rows in the second[1]

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \bullet \begin{bmatrix} 6 \\ 7 \end{bmatrix} = \begin{bmatrix} 34 \\ 47 \\ 60 \end{bmatrix}$$

$$3 \times 2 \quad = \quad 2 \times 1 \qquad 3 \times 1$$

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \bullet \begin{bmatrix} 6 \\ 7 \end{bmatrix} \rightarrow \begin{bmatrix} 1\text{x}6 + 4\text{x}7 \\ 2\text{x}6 + 5\text{x}7 \\ 3\text{x}6 + 6\text{x}7 \end{bmatrix} = \begin{bmatrix} 34 \\ 47 \\ 60 \end{bmatrix}$$

a b numpy.dat(a,b)

[1] *http://en.wikipedia.org/wiki/Matrix_(mathematics)*

2-d ndarray object (“a matrix”) can be defined; and operations applied

```
>>> a=np.array([[1,2,3],[4,5,6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a.shape
(2, 3)
>>> a.T
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> a.T.shape
(3, 2)
>>> b=np.array([6, 7])
>>> np.dot(a.T,b)
array([34, 47, 60])
```

ndarray operations

```
>>> b = np.arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> b.sum(axis=0)
array([12, 15, 18, 21])
>>> b.sum(axis=1)
array([ 6, 22, 38])
>>> b.sum()
66
>>> b.min(axis=1)
array([0, 4, 8])
>>> b.max(axis=0)
array([ 8,  9, 10, 11])
```

Matric object and 2-d matrix indexing and slicing

```
>>> b=np.mat('1 2 3 4; 5 6 7 8; 9 10 11 12')
>>> b
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]])
>>> b[:,:3]
matrix([[ 1,  2,  3],
        [ 5,  6,  7],
        [ 9, 10, 11]])
>>> b[:,3]
matrix([[ 4],
        [ 8],
        [12]])
```

Matrix object – allows matrix arithmetic using operators

```
>>> b=np.mat('1 2 3 4; 5 6 7 8; 9 10 11 12')
>>> b
matrix([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]])
>>> a=np.mat('1;2;3;4')
>>> b*a
matrix([[ 30],
        [ 70],
        [110]])
```

n-dimension ndarray object (“matrices”) can also be created

```
>>> aa=np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
>>> aa
array([[[1, 2, 3],
        [4, 5, 6]],
       [[1, 2, 3],
        [4, 5, 6]]])
>>> bb=np.array([[[3],[4],[6]],[[6],[5],[7]]])
>>> aa.shape, bb.shape
((2, 2, 3), (2, 3, 1))
>>> np.dot(aa,bb)
array([[[[29],
         [37]],
        [[68],
         [91]]],
       [[[29],
         [37]],
        [[68],
         [91]]]])
```

ndarray element-wise operations – scalar and matrix

```
>>> aa = np.arange(5)
>>> aa
array([0, 1, 2, 3, 4])
>>> aa * 5
array([ 0,  5, 10, 15, 20])
>>> bb = np.array([2, 4, 6, 8, 10])
>>> np.multiply(aa, bb)
array([ 0, 4, 12, 24, 40])
>>> np.divide(aa, bb.astype(float))
array([ 0.        ,  0.25      ,  0.33333333,  0.375     ,  0.4       ])
```

Matrix math

```
>>> a=np.array([[1,2],[3,4]])
>>> a
array([[1, 2],
       [3, 4]]
>>> b=np.array([[1],[2]])
>>> b
array([[1],
       [2]])
>>> a.shape, b.shape
((2, 2), (2, 1))
>>> np.dot(a,b)
array([[ 5],
       [11]])
>>> np.dot(b,a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: objects are not aligned
```

Matplotlib

"... tries to make easy things easy
and hard things possible. You can generate plots,
histograms, power spectra, bar charts,
errorcharts, scatterplots, etc, ..."

- matplotlib website

http://matplotlib.org/

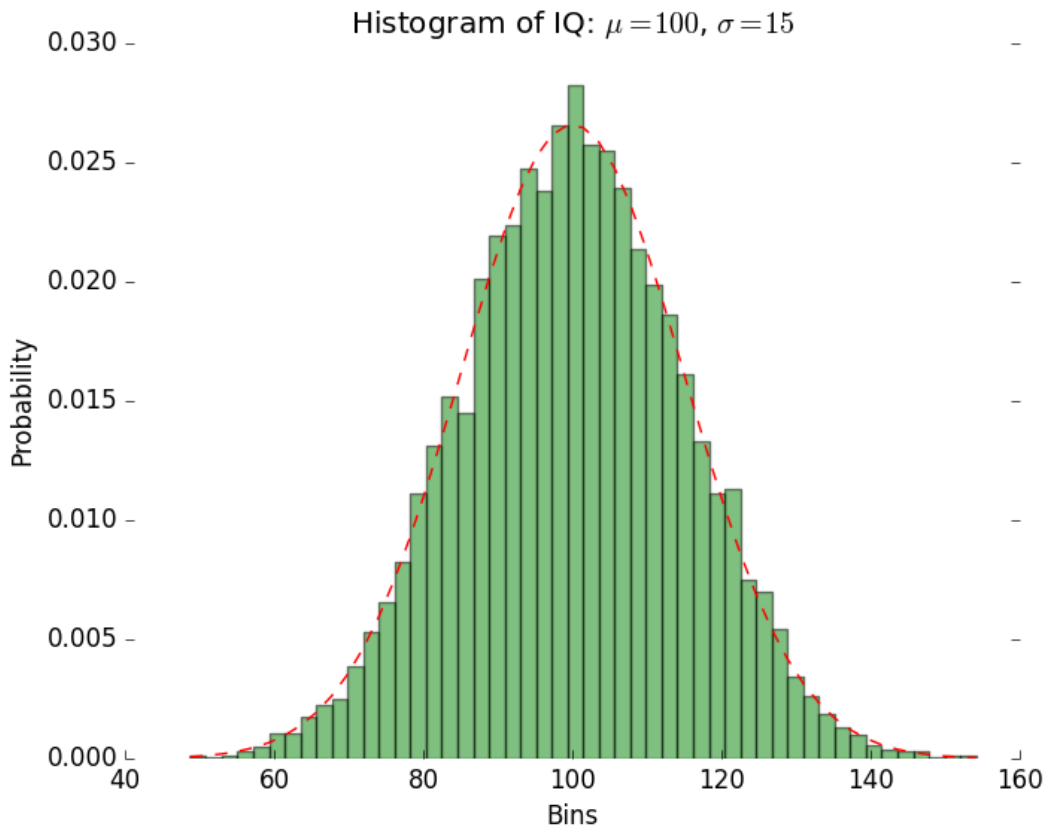
API/toolkit highlights

- matplotlib.pyplot - plotting framework
- matplotlib.mlab - compatibility with MATLAB commands
- matplotlib.backends - the output format of the plot (pdf, screen)
- mpl_toolkits.mplot3d - basic 3D plotting (scatter, surf, line, mesh)

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

mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(10000)

num_bins = 50
# the histogram of the data
n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='green', alpha=0.5)

y = mlab.normpdf(bins, mu, sigma) # obtain a 'best fit' line
plt.plot(bins, y, 'r--') # actually plot the line
plt.xlabel('Bins') # give the plot some context with axis labels
plt.ylabel('Probability')
plt.title(r'Histogram of IQ: $\mu=100$, $\sigma=15$') # and a title

# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()
```

DATA SCIENCE LIBRARY: MATPLOTLIB


```
>>> bins
Array([  48.764523     50.87209103   52.97965905   55.08722708   57.19479511
   59.30236313   61.40993116   63.51749918   65.62506721   67.73263523
   69.84020326   71.94777129   74.05533931   76.16290734   78.27047536
   80.37804339   82.48561141   84.59317944   86.70074747   88.80831549
   90.91588352   93.02345154   95.13101957   97.23858759   99.34615562
  101.45372364  103.56129167  105.6688597   107.77642772  109.88399575
  111.99156377  114.0991318   116.20669982  118.31426785  120.42183588
  122.5294039   124.63697193  126.74453995  128.85210798  130.959676
  133.06724403  135.17481206  137.28238008  139.38994811  141.49751613
  143.60508416  145.71265218  147.82022021  149.92778824  152.03535626
  154.14292429])
>>> y # "Probability"
Array([  7.78686799e-05   1.24595710e-04   1.95465329e-04   3.00651063e-04
   4.53400627e-04   6.70390505e-04   9.71851609e-04   1.38133312e-03
   1.92496687e-03   2.63011330e-03   3.52331944e-03   4.62760144e-03
   5.95917639e-03   7.52389798e-03   9.31377883e-03   1.13040822e-02
   1.34515107e-02   1.56939830e-02   1.79523646e-02   2.01342987e-02
   2.21400052e-02   2.38696085e-02   2.52312773e-02   2.61492684e-02
   2.65708969e-02   2.64715427e-02   2.58570308e-02   2.47630662e-02
   2.32517985e-02   2.14059766e-02   1.93214585e-02   1.70990166e-02
   1.48364065e-02   1.26215496e-02   1.05274454e-02   8.60913831e-03
   6.90275939e-03   5.42640433e-03   4.18242277e-03   3.16060393e-03
   2.34173952e-03   1.70111416e-03   1.21158727e-03   8.46062225e-04
   5.79263647e-04   3.88845081e-04   2.55919459e-04   1.65141556e-04
   1.04480635e-04   6.48099365e-05   3.94161094e-05])
```



SciPy

"is a collection of mathematical algorithms
and convenience functions
built on the Numpy extension "

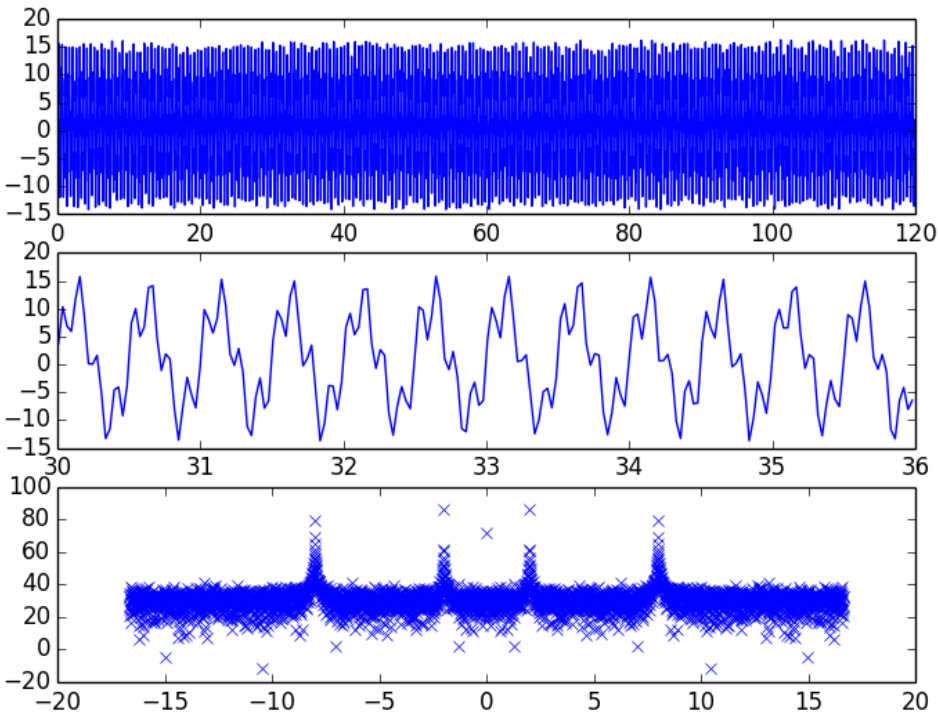
– SciPy documentation

http://docs.scipy.org/doc/scipy-dev/scipy-ref.pdf

Scipy: Fast Fourier Transform

```
import scipy, scipy.fftpack, matplotlib.pyplot as plt, scipy.constants as c
sig = scipy.linspace(0,120,4000)
acc = lambda t: 10 * scipy.sin(2 * c.pi * 2.0 * t) + \
      5 * scipy.sin(2 * c.pi * 8.0 * t) + 2 * scipy.random.random(len(t))

signal = acc(sig)

FFT = abs(scipy.fft(signal))
freqs = scipy.fftpack.fftfreq(signal.size, sig[1]-sig[0])

f, (ax1, ax2, ax3) = plt.subplots(3, sharex=False, sharey=False)

ax1.plot(sig, signal)
ax2.plot(sig[1000:1200], signal[1000:1200])
ax3.plot(freqs,20*scipy.log10(FFT),'x')

plt.show()
```

Scipy: Fast Fourier Transform Visualized




```
ax1.plot(t, signal)
```

```
ax2.plot(t[1000:1200], signal[1000:1200])
```

```
ax3.plot(freqs,20*scipy.log10(FFT),'x')
```

Additional sub-packages:

cluster	Clustering algorithms
constants	Physical and mathematical constants
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
linalg	Linear algebra
ndimage	N-dimensional image processing
optimize	Optimization and root-finding routines
sparse	Sparse matrices and associated routines

Pandas

"... an open source... library providing
high-performance... data structures
and data analysis tools"

- pandas website

http://pandas.pydata.org/

Series - one-dimensional labeled array

```
>>> import pandas as pd, numpy as np
>>> s = pd.Series(np.random.randn(3), index=['a', 'b', 'c’])
>>> s
a   -0.245247
b   -1.162124
c   -0.698275
dtype: float64
>>> s['a']
-0.24524714260468519
>>> s[0]
-0.24524714260468519
>>> d = {'a' : 0, 'b' : 1, 'c' : 2}
>>> pd.Series(d)
a    0
b    1
c    2
dtype: int64
```

Series – vector operation support and index alignment

```
>>> s = pd.Series(np.arange(5), index=['a', 'b', 'c', 'd'])
>>> s + s
a    0
b    2
c    4
d    6
dtype: int64
>>> t = pd.Series([30,40], index=['b', 'c' ])
>>> t
b    30
c    40
dtype: int64
>>> s + t
a   NaN
b    31
c    42
d   NaN
```

DataFrame – 2-d labeled data structure

```
>>> d = {'one' : [10., 20., 30., 40.], 'two' : [4., 3., 2., 1.]}
>>> pd.DataFrame(d, index=['a','b','c','d'])
   one  two
a   10    4
b   20    3
c   30    2
d   40    1
>>> dd={'0':pd.Series([1,2],index=['a','b']),
...     '1':pd.Series([15,25,35],index=['a','b','c'])}
>>> pd.DataFrame(dd)
    0   1
a   1  15
b   2  25
c NaN  35
```

DataFrame – data alignment and arithmetic operations

```
>>> df = pd.DataFrame(np.floor(np.random.randn(3, 4)*10), columns=['A', 'B', 'C', 'D'])
>>> df
   A   B   C   D
0  4  -1  12  18
1  8  12  -7  11
2  2  13   6  21
>>> df2 = pd.DataFrame(np.floor(np.random.randn(3,2)*10), columns=['B', 'C'])
>>> df2
    B   C
0  -2 -10
1  11   9
2   4   2
>>> df + df2
    A   B  C   D
0 NaN  -3  2 NaN
1 NaN  23  2 NaN
2 NaN  17  8 NaN
```

Panel – 3-d labeled data structure

```
>>> panel = pd.Panel(np.random.randn(5,3,2).round(decimals=1),
...                  items=['one', 'two', 'three','four','five'],
...                  major_axis=pd.date_range('1/1/2000', periods=3),
...                  minor_axis=['a', 'b'])
>>> panel.to_frame()
                one  two  three  four  five
major      minor
2000-01-01 a     -1.4 -0.1   -0.1  -0.9   2.9
           b     -0.1 -0.1    1.5   0.1   0.5
2000-01-02 a      0.9  0.3   -0.2  -1.1  -0.8
           b     -0.0 -0.6    0.0  -0.0   1.4
2000-01-03 a      0.1  2.0    0.2   2.4  -1.2
           b     -0.9 -1.8    1.0  -1.4  -0.9
```

Some additional notable features:

- Data loading (flat files, Excel, MySQL)
- Data selection using indexes
- Group by (columns or indexes)
- Joins
- NumPy and custom functions vectorized via the .apply() method

Scikits

> "(short for SciPy Toolkits), are add-on packages for SciPy,
> hosted and developed separately
> from the main SciPy distribution."
>
> - SciKits website

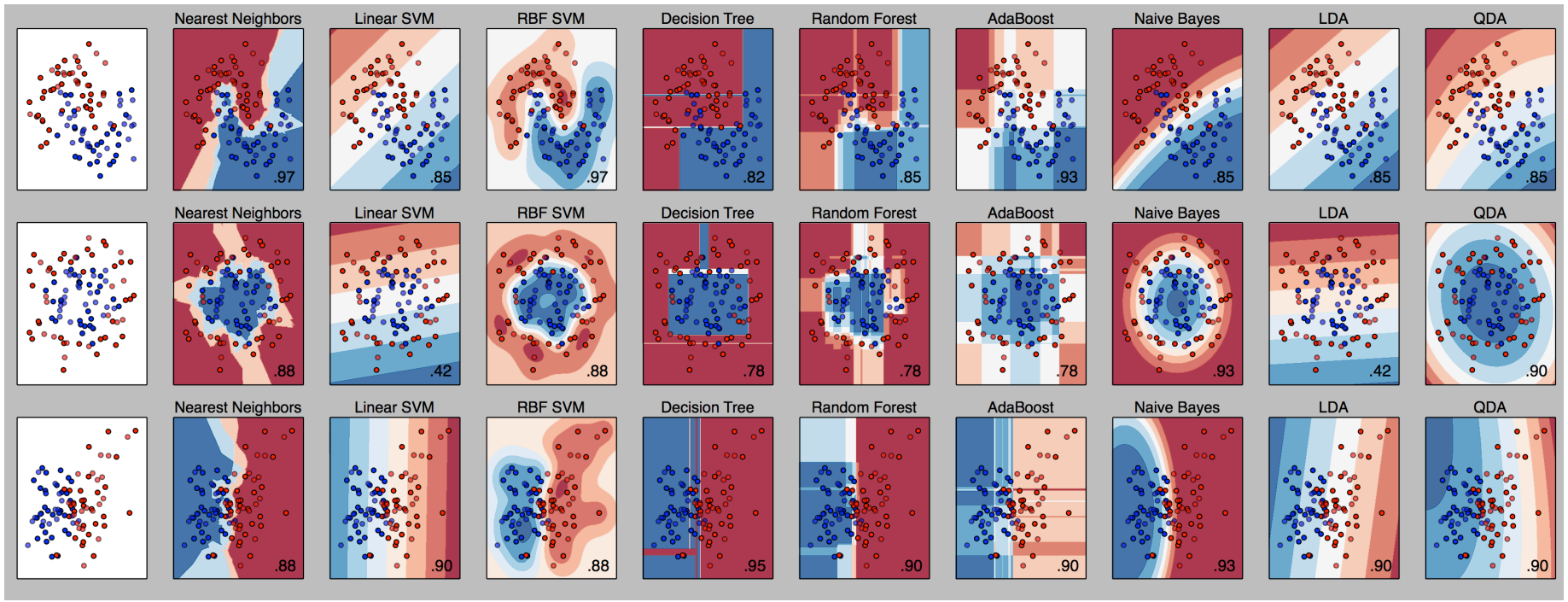
http://scikits.appspot.com/about

Some useful SciKits:

scikit-learn	Machine learning and data mining
scikit-monaco	Monte Carlo integration
statsmodels	Statistical computations and models
optimization	Numerical optimization

http://scikits.appspot.com/scikits

Machine learning library: Models, Optimization, Preprocessing, ...



http://scikit-learn.org/stable/auto_examples/plot_classifier_comparison.html

RESOURCES & NEXT STEPS

Learn the command line interface (free HTML version)
http://cli.learncodethehardway.org

Take a Python tutorial (free HTML version)
http://learnpythonthehardway.org

Refresh your understanding of linear regression
https://www.khanacademy.org/math/probability/regression

Python Distribution (for Analytics) – Annaconda by Continuum Analytics
http://contimuum.io

On-line Python Integrated Development Environment (IDE)
http://wakari.io

LAB

...nothing to see here.

...go back.